

Application of the DeepXDE Library for Solving Mathematical Models with PINNs

Gabriel Barboza(✉) and Isaac P. dos Santos

Federal University of Espírito Santo, Vitória, ES, Brazil
gabriel.i.barboza@edu.ufes.br, isaac.santos@ufes.br

Abstract. In this work, we investigate the capabilities of the DeepXDE library to solve mathematical models based on ordinary and partial differential equations using Physics-Informed Neural Networks (PINNs). PINNs are an important machine learning tool for solving complex problems modeled by differential equations, by incorporating physical laws into the neural network structure. We assess the versatility and effectiveness of DeepXDE by applying the PINNs methodology to several initial and boundary value problems, including the Verhulst logistic model, the 1D Gray-Scott reaction-diffusion system, the SIR epidemiological model, the incompressible Navier-Stokes equations, and an inverse problem derived from epidemiological data. The DeepXDE library facilitates the implementation of PINNs by providing an efficient and easy-to-use tool both for educational purposes (classroom use) and for solving direct and inverse problems in engineering and computational science.

Keywords: DeepXDE · Physics-Informed Neural Networks (PINNs) · Neural Networks

1 Introduction

Many complex problems in fields such as engineering, biology, epidemiology, economy, physics, etc. are described by mathematical formulations based on partial differential equations (PDEs) or ordinary differential equations (ODEs). Solving such equations often require specialized numerical methods for their solution, especially when the problems involve complex domains, high dimensions, or intricate physical processes. Among the various numerical methods, we highlight Finite Differences, Finite Element Method (FEM), and Finite Volume Method.

Recently, due to the rapid progress of artificial intelligence and data science, Physics-Informed Neural Networks (PINNs) have emerged as an important tool for solving such complex problems modeled by differential equations. The concept of PINNs can be traced back to the 1990s with early developments by Psichogios and Ungar (1992), Meade and Fernandez (1994), and Lagaris et al. (1998) [13,15,18,22]. Psichogios and Ungar (1992) introduced a hybrid learning approach that integrated physical laws, such as mass and energy balance,

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
O. Gervasi et al. (Eds.): ICCSA 2025 Workshops, LNCS 15888, pp. 122–139, 2026.
https://doi.org/10.1007/978-3-031-97596-7_8

into the network structure. This integration led to improved generalization and extrapolation capabilities compared to traditional machine learning methods. Lagaris et al. (1998) used a feedforward neural network to solve both ODEs and PDEs in rectangular domains, with a cost function that minimized the residuals of the equations at sampled points over the domain. This approach utilized a Quasi-Newton method for training the network.

In [23], Raissi et al. (2019) expanded the PINNs framework to solve PDEs directly, incorporating residuals from both the PDEs and the initial/boundary conditions into the loss function. The key feature of PINNs lies in their ability to integrate prior knowledge, such as physical laws and differential equations, directly into the neural network structure, ensuring that the network's outputs are consistent with the laws of physics. A considerable number of publications have emerged in recent times, focusing on the utilization of PINNs or deep learning models to solve differential equations [3,5,8,10,26].

One of the important properties of PINNs is their ability to solve high-dimensional problems in complex geometries without the need for meshing, which is a requirement in traditional numerical methods such as finite element or finite difference methods. Furthermore, PINNs provide a flexible framework for solving coupled systems of equations, making them an important tool in interdisciplinary applications that arise especially in engineering.

To facilitate the implementation of PINNs, several libraries and frameworks have been developed, each with distinct features and capabilities. TensorFlow [1] and PyTorch [21] are well known deep learning libraries that can be adapted for implementing PINN-based numerical methodology. Other more specialized libraries, such as DeepXDE [17] and SciANN [9], were created specifically to provide high-level abstractions and tools for solving differential equations using PINNs. These libraries are designed to simplify the process of defining the neural network architecture, integrating physical constraints, and training the network. SciANN, for example, uses TensorFlow and Keras [2] to implement PINNs. As emphasized in [17], DeepXDE is designed to be used both as a teaching tool in educational settings and as a research tool for solving problems in computational science and engineering. Specifically, DeepXDE enables the solution of forward problems, based on initial and boundary conditions, and inverse problems, based on additional measurements. DeepXDE supports domains with complex geometry and allows the creation of compact codes, similar to mathematical formulations.

In this paper, we explore the DeepXDE library for solving boundary and initial value problems with Physics-Informed Neural Networks. We begin by introducing the fundamental framework for solving ODEs and PDEs using PINNs. We then apply this methodology to a series of problems, evaluating the performance and effectiveness of the approach within the context of the DeepXDE library. The remainder of this paper is organized as follows. Section 2 presents the definition and formulation of PINNs. The DeepXDE library is briefly discussed in Sect. 3, while numerical experiments are presented in Sect. 4. Finally, Sect. 5 presents conclusions and concluding remarks.

2 The Fundamentals of PINNs Networks

The core idea of PINNs lies in embedding the governing equations directly into the loss function of the neural network (see Fig. 1). During training, the network aims to minimize a composite loss that accounts for the residuals of the differential equations, along with the initial and boundary conditions. These terms are evaluated at specific points in the spatiotemporal domain, known as collocation points. These points are typically sampled randomly and serve as locations where the model enforces the physical constraints of the problem [23].

In problems where observational data are available, PINNs offer a flexible framework that combines physical knowledge with data-driven learning. The loss function can incorporate both the equation residuals at collocation points and discrepancies between model predictions and known measurements. This unified approach enables the solution of a wide range of tasks, including inverse problems, where unknown parameters must be identified from data, and forward problems, where the objective is to predict the system's evolution from given initial and boundary conditions. When no data are provided, the network relies solely on the physical laws, effectively functioning as a mesh-free solver guided entirely by the underlying differential equations [10].

PINNs are capable of solving differential equations that can be represented in their most general form as:

$$\mathcal{L}(u(\boldsymbol{x},t);\boldsymbol{\lambda}) = f(u,\boldsymbol{x},t), \quad \boldsymbol{x}\in\Omega,\, t\in I, \tag{1}$$

$$\mathcal{B}(u(\boldsymbol{x},t)) = g(\boldsymbol{x},t), \quad \boldsymbol{x}\in\Gamma,\, t\in I, \tag{2}$$

$$\mathcal{I}(u(\boldsymbol{x},t_0)) = q(\boldsymbol{x}), \quad \boldsymbol{x}\in\Omega, \tag{3}$$

where $\Omega\subset\mathbb{R}^d$ is the spatial domain with boundary Γ; d is the dimension of the space; $I=[t_0,t_F]$ is the time interval, with $t_0<t_F$; $\boldsymbol{x}=(x_1,x_2,\ldots,x_d)$ is the spatial coordinate vector; t denotes time; $u=u(\boldsymbol{x},t)$ denotes the unknown solution of the problem; $\boldsymbol{\lambda}$ is the vector of physical parameters; $\mathcal{L}$ is the differential operator associated with the equation; f is a given source term; $\mathcal{B}$ and $\mathcal{I}$ are operators representing boundary and initial conditions, respectively; and g and q are known functions defining those conditions.

In the PINNs approach, the function $u(\boldsymbol{x},t)$ is approximated computationally by a neural network parameterized by a vector $\boldsymbol{\theta}=(\boldsymbol{w},\boldsymbol{b})\in\mathbb{R}^{n_\theta}$, where $\boldsymbol{w}$ and $\boldsymbol{b}$ represent the weights and biases of the network, respectively, and n_θ denotes the total number of parameters. This yields an approximate solution $\widehat{u}(\boldsymbol{x},t;\boldsymbol{\theta})$ for $u(\boldsymbol{x},t)$.

The loss function in PINNs typically includes the residual of the equation, initial/boundary conditions and measured data (if available). Thus, we define the loss function associated with problem (1)–(3) as

$$J(\boldsymbol{\theta}) = \omega_{\mathcal{L}}J_{\mathcal{L}}(\boldsymbol{x},t,\boldsymbol{\theta}) + \omega_{\mathcal{B}}J_{\mathcal{B}}(\boldsymbol{x},t,\boldsymbol{\theta}) + \omega_{\mathcal{I}}J_{\mathcal{I}}(\boldsymbol{x},t_0,\boldsymbol{\theta}) + \omega_{data}J_{data}(\boldsymbol{x},t,\boldsymbol{\theta}) \tag{4}$$

where

$$J_{\mathcal{L}}(\boldsymbol{x}, t, \boldsymbol{\theta}) = \|\mathcal{L}(\widehat{u}(\boldsymbol{x}, t, \boldsymbol{\theta}); \boldsymbol{\lambda}) - f(\widehat{u}(\boldsymbol{x}, t, \boldsymbol{\theta}), \boldsymbol{x}, t)\|^2_{\Omega \times I},$$
$$J_{\mathcal{B}}(\boldsymbol{x}, t, \boldsymbol{\theta}) = \|\mathcal{B}(\widehat{u}(\boldsymbol{x}, t, \boldsymbol{\theta})) - g(\boldsymbol{x}, t)\|^2_{\Gamma \times I},$$
$$J_{\mathcal{I}}(\boldsymbol{x}, t_0, \boldsymbol{\theta}) = \|\mathcal{I}(\widehat{u}(\boldsymbol{x}, t_0, \boldsymbol{\theta})) - q(\boldsymbol{x})\|^2_{\Omega}$$

represent the squared residuals of the equation (PDE/ODE), the boundary conditions, and the initial condition, respectively, evaluated at the collocation points and measured using the Euclidean norms $\|\cdot\|_{\Omega \times I}$, $\|\cdot\|_{\Gamma \times I}$, and $\|\cdot\|_{\Omega}$. When measured data are available, the operator J_{data} is defined as

$$J_{\text{data}}(\boldsymbol{x}, t, \boldsymbol{\theta}) = \|\widehat{u}(\boldsymbol{x}, t, \boldsymbol{\theta}) - u(\boldsymbol{x}, t)\|^2_{\Omega \times I}.$$

The parameters $\omega_{\mathcal{L}}, \omega_{\mathcal{B}}, \omega_{\mathcal{I}}$, and ω_{data} are weighting coefficients that can be used to mitigate imbalance among the different loss terms.

Accordingly, solving problem (1)–(3) is equivalent to determining the parameter vector $\boldsymbol{\theta}$ such that the neural network approximation $\widehat{u}(\boldsymbol{x}, t, \boldsymbol{\theta})$ closely matches the true solution $u(\boldsymbol{x}, t)$ within a prescribed accuracy over the spatio-temporal domain. To this end, a set of N_c collocation points, $\{(\boldsymbol{x}_i, t_i)\}_{i=1}^{N_c}$, distributed across the spatio-temporal domain and its boundary, is used to evaluate the loss function defined in (4).

The optimization problem that must be solved consists of finding

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}} J(\boldsymbol{\theta}), \tag{5}$$

so that $\widehat{u}(\boldsymbol{x}, t, \boldsymbol{\theta}^*)$ is the PINNs solution of (1)–(3).

Most methods for solving (5) rely on gradient-based optimization combined with the backpropagation algorithm. These methods minimize the discrepancy between predicted and expected outputs by adjusting the weights and biases in the direction of the steepest gradient descent. However, some studies have shown that this approach may exhibit undesirable behavior when training PINNs to solve complex problems, particularly when the solutions of the underlying PDEs involve high-frequency or multiscale features [28,29]. In this work, we employ the Adam method (short for Adaptive Moment Estimation) [12] and BFGS-L, a quasi-Newton method that refines the solution by incorporating second-order information [16]. Both optimizers are available in the DeepXDE library.

3 The DeepXDE Library

Presented in [17] as an educational and research tool, DeepXDE is a deep learning library focused on solving differential equations (ODEs/PDEs) using neural networks. This open-source Python tool provides an intuitive and advanced interface for training neural networks to approximate mathematical models based on differential equations.

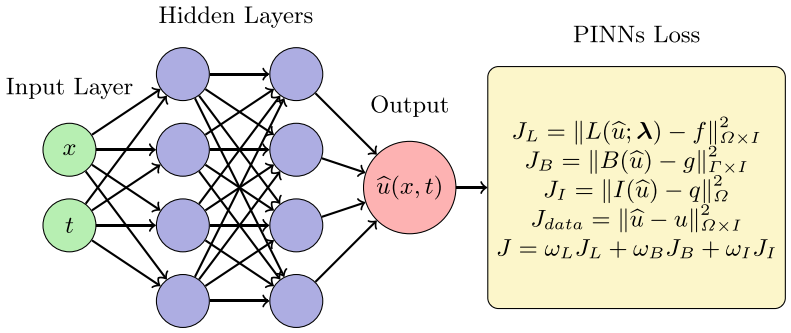

Fig. 1. PINNs graphical representation. Source: elaborated by the authors.

As mentioned in the Introduction, DeepXDE keeps the code concise and elegant, similar to the mathematical formulation. Solving differential equations with DeepXDE involves simply defining the problem using built-in modules, which include the computational domain (geometry and time), the equations (ODE/PDE), boundary/initial conditions, constraints, collocation points, neural network architecture, and training hyperparameters. It is also compatible with the most common back-end engines to perform the heavy computations.

3.1 Workflow for Solving Problems

The resolution of problems with DeepXDE can be easily accomplished by following a sequence of well-defined steps. To illustrate the use of DeepXDE, we consider the Burgers equation as formulated in [20]:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - c\frac{\partial^2 u}{\partial x^2} = 0, \quad x \in (-1, 1), \quad t \in (0, 5], \tag{6}$$

with the initial condition $u(x, 0) = -\sin(\pi x)$ for $x \in [-1, 1]$, and homogeneous Dirichlet boundary conditions $u(\pm 1, t) = 0$ for $t \in [0, 5]$. In Eq. (6), $c > 0$ denotes the viscosity coefficient.

First, import the back-end library and set the virtual environment variable. Then, import DeepXDE. You can choose between TensorFlow, PyTorch, JAX, and Paddle. This step is shown in Listing 1.1.

```
import os
import torch
os.environ['DDE_BACKEND'] = 'pytorch'
import deepxde as dde
```

Listing 1.1. Importing libraries

Any problem involving the solution of PDEs or ODEs requires the definition of a domain of interest. DeepXDE provides functions for representing spatial and temporal intervals, allowing the problem geometry to be easily specified as a combination of these components. The code shown in Listing 1.2 defines the spatial domain $[x_0, x_f] = [-1, 1]$ and the temporal domain $[t_0, t_f] = [0, 5]$.

```
x0, xf = -1, 1
t0, tf = 0, 5
geomx = dde.geometry.Interval(x0, xf)
tinterval = dde.geometry.TimeDomain(t0, tf)
geom = dde.geometry.GeometryXTime(geomx, tinterval)
```

Listing 1.2. Defining the domain

In the next step, the equation or system of equations is defined as a residual to be minimized, as shown in Listing 1.3. The first parameter represents the input features, while the second corresponds to the network output. Use both, along with the tools provided by DeepXDE, to compute the necessary derivatives and formulate the residuals.

```
def pde(training_data, network_output):

    # Compute the Jacobian of u w.r.t. x
    # First-order partial derivative of u w.r.t. x
    du_dx = dde.grad.jacobian(y, x, i=0, j=0)

    # Compute the time Jacobian
    # First-order partial derivative of u w.r.t. t
    du_dt = dde.grad.jacobian(y, x, i=0, j=1)

    # Compute the Hessian (second-order derivatives)
    # Second-order partial derivative of u w.r.t. x
    d2u_d2x = dde.grad.hessian(y, x, component=0, i
        =0, j=0)

    return [
        # Residual of Burgers' equation
        du_dt + u * du_dx - c * d2u_d2x
    ]
```

Listing 1.3. Defining the model

The specification of boundary and/or initial conditions is illustrated in Listing 1.4.

```
# Define the boundary conditions
bc = dde.icbc.DirichletBC(geom, lambda x: 0, lambda _
    , on_boundary: on_boundary)

# Define the initial conditions
ic = dde.icbc.IC(geom, lambda x: -np.sin(np.pi * x[:,
     0:1]), lambda _, on_initial: on_initial)
```

Listing 1.4. Defining the initial and boundary conditions

Neural networks require a dataset for training, and PINNs are no exception. In this case, the collocation points consist of points distributed throughout the domain, including the interior, the boundaries, and the initial time. The configuration of this dataset depends on the specific equations, domain, and conditions of the problem. DeepXDE allows you to define the number of such points, as detailed in Listing 1.5.

```
# Create the collocation points
data = dde.data.TimePDE(geom, pde, [bc, ic],
    num_domain=1024, num_boundary=40, num_initial=50)
```

Listing 1.5. Creating the data set

Then you need to define the neural network architecture. The number of input and output nodes depends on the specific model being solved, while the number of hidden layers and the number of neurons per layer are typically chosen empirically. The choice of activation function also varies with the problem, but the hyperbolic tangent (tanh) is generally recommended for this type of application. You must also configure the optimizer, the algorithm used to solve the optimization problem (5), and specify the learning rate, which controls the size of each update step during training. This configuration is illustrated in Listing 1.6, where the Adam optimizer [12] and a learning rate of 0.001 are used.

```
# 2 inputs node X 2 hidden layers of 5 nodes each X 1
    output node
layer_size = [2] + [5] * 2 + [1]

# Define the architecture and activation function
net = dde.nn.FNN(layer_size, "tanh")
model = dde.Model(data, net)
# Define the optimizer, learning rate
model.compile("adam", lr=1e-3)
```

Listing 1.6. Configuring the neural network

Finally, you can train your model. It is a common practice to train first with a faster, but not quite efficient, optimizer like Adam, then fine-tune your results

with a more robust optimizer. The number of iterations and which optimizer to choose also depend on the problem that is being solved. Listing 1.7 illustrates a two-phase training strategy: first using Adam for fast initial convergence, then switching to BFGS-L, a quasi-Newton method that refines the solution by incorporating second-order information [16].

```
# Train the model for n iterations
losshistory, train_state = model.train(iterations
    =500)
model.compile("BFGS-L", lr=1e-3)
losshistory, train_state = model.train(iterations
    =500)
```

Listing 1.7. Training model

Save your results. DeepXDE offers built-in functionality to visualize the learned solution and the convergence history. Additionally, it allows you to export the generated data and model output for further analysis, as demonstrated in Listing 1.8.

```
model.train(iterations=500, model_save_path="path/to/
    model")
dde.saveplot(losshistory, train_state, issave=True,
    isplot=True)
```

Listing 1.8. Displaying the model

3.2 Inverse Problems

The previous example illustrates how DeepXDE can be applied to solve a forward problem. However, in many cases, we are interested in solving inverse problems, which consist of determining unknown parameters in a model given a known solution, whether analytical or numerical. These parameters often represent physical coefficients. As described by [24], PINNs are capable of solving such problems by treating the unknown parameters as learnable variables during training.

This can be accomplished in DeepXDE by defining the variable of interest with an initial guess, incorporating it into the PDE/ODE residual, and specifying it as a trainable parameter in the model. The learned values can be easily accessed after training, as shown in Listing 1.9.

```
v = dde.Variable(0.5)
model.compile("adam", lr=1e-3,
    external_trainable_variables=[v])
print(f"diffusion coefficient = {v.detach().cpu().
    numpy()}")
```

Listing 1.9. Inverse problems with DeepXDE

4 Applications

In this section, we address a series of initial and/or boundary value problems using DeepXDE. To quantitatively evaluate the quality of the PINNs solutions, we compute the errors using the infinity norm. In the first two examples, the solutions obtained with PINNs are compared to exact solutions. In the third case, the PINNs solution is compared with the explicit fourth-order Runge-Kutta method. The subsequent problem involves the simulation of incompressible fluid flow, where the PINNs results are compared with a reference numerical solution. Finally, we solve an inverse problem aimed at identifying the parameters of the SIR epidemiological model. The implementation of all these cases is available in the GitHub repository, where readers can access and explore the full source code and results.

4.1 The Verhulst Logistic Model

The Verhulst logistic model [19,27], given by the initial value problem

$$\frac{dp(t)}{dt} = (a - bp(t))\, p(t), \quad t > t_0, \tag{7}$$

$$p(t_0) = p_0, \tag{8}$$

describes the growth of a population, denoted by $p(t)$, over time t, considering both reproduction and limited resources. In the ODE (7) a is the growth rate, b is a constant that controls the strength of the limiting effect, whereas Eq. (8) describes the initial solution. This model is widely used in biology and ecology to study population dynamics, providing insights into how populations evolve when resources are finite. The solution of the equation exhibits logistic growth, where the population increases rapidly at first but slows down as it approaches the carrying capacity $K = \frac{a}{b}$ [19]. The carrying capacity represents the maximum population size that the environment can sustainably support, beyond which the population growth stabilizes. The Verhulst model helps understand how a population interacts with its environment and the limitations imposed by finite resources.

The exact solution of the problem (7)–(8) is given by

$$p(t) = \frac{ap_0}{bp_0 + (a - bp_0)e^{-(t-t_0)}}.$$

We solve this problem in the time interval $[0, 5]$, with $a = 1.0$, $b = 0.1$, $p_0 \in \{12, 8\}$, and $t_0 = 0$. Thus, the carrying capacity is $K = 10$. We used a network with three hidden layers, each containing 50 nodes, with the hyperbolic tangent activation function and the Adam optimizer, initialized with a learning rate of 10^{-3}. The network was trained for 20,000 iterations. Figure 2 shows both the PINNs and exact solutions, with virtually no visual difference, demonstrating good agreement between them. To quantify the quality of the PINN solution,

we compute the error between the exact and predicted solutions in the infinity norm. We obtain $\|u - \widehat{u}\|_\infty = 2.336 \times 10^{-4}$ for $p_0 = 12$, and 2.743×10^{-4} for $p_0 = 8$, corroborating the satisfactory accuracy observed in the figures.

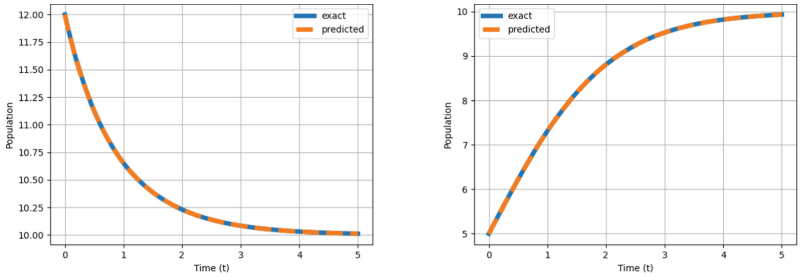

Fig. 2. Verhulst model with initial conditions: $p_0 = 12$ (left) and $p_0 = 8$ (right).

4.2 Grey-Scott Model for Pattern Formation

The Grey-Scott model [7] is a reaction-diffusion system that describes how two interacting chemical species, U and V, evolve over time, leading to complex pattern formation. It is widely studied in mathematical biology, physics, and nonlinear dynamics due to its ability to generate intricate spatial structures. The governing equations are:

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u - uv^2 + F(1-u), \tag{9}$$

$$\frac{\partial v}{\partial t} = D_v \nabla^2 v + uv^2 - (F+k)v. \tag{10}$$

Here, D_u and D_v are diffusion coefficients, F is the feed rate controlling u's introduction, and k is the removal rate of v. The reaction term uv^2 represents an autocatalytic process where v catalyzes the transformation of u into more v.

By tuning D_u, D_v, F, and k, the model produces diverse patterns, such as Turing structures, waves, and self-replicating spots, making it relevant in studying self-organization in nature. It has applications in chemical reactions, biological morphogenesis (e.g., animal coat patterns), and material science, offering insights into emergent complexity in dynamical systems.

We solve the system (9)–(10) in the one-dimensional case, with $D_u = D_v = 1$, and compare the PINNs solutions with the exact ones. Using the transformation $z = x - \theta t$, the analytical solution, as presented in [25], is given by

$$u(z) = \frac{3-\sqrt{\eta}}{4} - \frac{\sqrt{2\xi}}{4}\tanh\left(\frac{\sqrt{\xi}}{4}z\right), \qquad v(z) = \frac{1+\sqrt{\eta}}{4} + \frac{\sqrt{2\xi}}{4}\tanh\left(\frac{\sqrt{\xi}}{4}z\right),$$

where $\eta = 1-4F$, $\xi = 1+\sqrt{\eta}-2F$, and $\theta = \frac{\sqrt{2}}{4}(1-3\sqrt{\eta})$. We used a network with four hidden layers, 20 nodes each, sigmoid activation function, Adam optimizer with learning rate of 10^{-3}, and trained for 25,000 iterations. Figure 3 shows the PINNs and exact solutions. The predicted curves are very close to the reference solutions, with only slight visible deviations, indicating that the PINN model captures the spatiotemporal dynamics of the system with good accuracy. We have obtained the following errors in the infinity norm: $\|u-\widehat{u}\|_\infty = 2.625\times 10^{-2}$ and $\|v-\widehat{v}\|_\infty = 2.467\times 10^{-2}$, which quantitatively reflect the quality of the approximation.

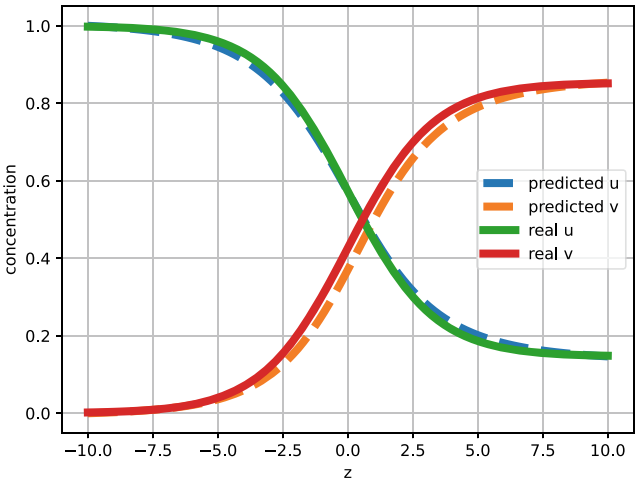

Fig. 3. 1D Grey-Scott model

4.3 The SIR Epidemiological Model

The SIR epidemiological model [11] describes the spread of infectious diseases in a population by dividing individuals into three compartments: Susceptible (S), Infected (I), and Recovered (R). It is a fundamental model in mathematical epidemiology and provides insights into disease transmission dynamics. The governing equations are:

$$\frac{dS}{dt} = -\beta SI, \tag{11}$$

$$\frac{dI}{dt} = \beta SI - \gamma I, \tag{12}$$

$$\frac{dR}{dt} = \gamma I. \tag{13}$$

Here, $S(t)$, $I(t)$, and $R(t)$ represent the number of susceptible, infected, and recovered individuals at time t. The parameter β is the infection rate, controlling how quickly the disease spreads, and γ is the recovery rate, determining how fast infected individuals recover and move into the recovered compartment.

We solve model (11)–(13) using synthetic data with parameters $\beta = 1.4$, $\mu = 0.9$, and initial conditions $S(0) = 0.99$, $I(0) = 0.01$, and $R(0) = 0$ (values given as percentages of the total population). The time domain is defined as $[0, 50]$. Figure 4 presents the PINNs solution (left) and the explicit fourth-order Runge–Kutta solution (right). The PINNs result is very similar to the one obtained with the Runge–Kutta method. The number of infected individuals increases rapidly at first, reaching a peak around days 7–8, and then decreases as more people recover and become immune. This behavior illustrates how the interaction between susceptible, infected, and recovered individuals affects the spread of the disease. For this problem, the network consisted of three layers with 50 nodes each. It was trained for 20,000 iterations using the hyperbolic tangent activation function and the Adam optimizer with a learning rate of 10^{-3}. To quantify the quality of the PINN solution, we compute the error between the Runge-Kutta and predicted solutions in the infinity norm, obtaining the following erros: 7.760×10^{-3}, 6.828×10^{-3} and 1.034×10^{-2} for the functions S, I and R, respectively, corroborating the satisfactory accuracy observed in the plots.

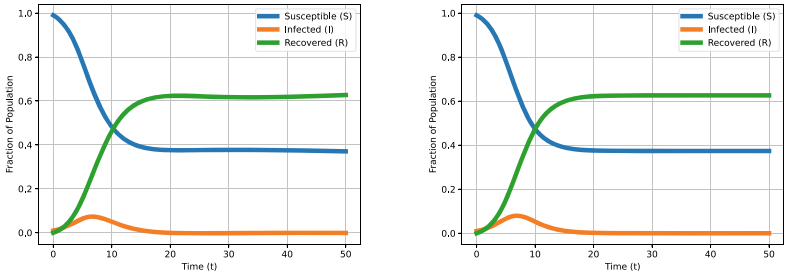

Fig. 4. PINNs (left) and fourth-order Runge-Kutta (right) solutions for the SIR model.

4.4 The Incompressible Navier-Stokes Equations

The incompressible Navier-Stokes equations [14], given by

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f}, \tag{14}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{15}$$

describe the motion of a fluid under the influence of internal and external forces, where $\mathbf{u} = (u, v, w)$ is the velocity field, p is the pressure, ρ is the fluid density, ν is the kinematic viscosity, and $\mathbf{f}$ represents external forces such as gravity. The first equation expresses the conservation of momentum, balancing inertial, pressure, viscous, and external forces. The second equation enforces the incompressibility condition, ensuring mass conservation. To complete the mathematical model, initial and appropriate boundary conditions must be specified for equations (14)–(15).

The incompressible Navier-Stokes equations are essential for describing fluid flows in areas like aerodynamics, ocean currents, and turbulence. Because they are nonlinear, finding exact solutions is very difficult. As a result, numerical methods like the finite element method (FEM) and computational fluid dynamics (CFD) are widely used to solve them.

A well-known example of the model (14)–(15) is the two-dimensional lid-driven cavity flow. It describes the movement of an incompressible fluid inside a square cavity, where the fluid is set in motion by the top boundary (the lid) sliding at a constant speed. Thus, we consider the stationary and dimensionless form of equations (14)–(15) on the unit square, with $Re = 100$ and subject to the following Dirichlet boundary conditions (see Fig. 5):

$$u(0,y) = u(x,0) = u(1,y) = v(0,y) = v(x,0) = v(1,y) = v(x,1) = 0,$$

$$u(x,1) = 1, \quad p(0,0) = 0.$$

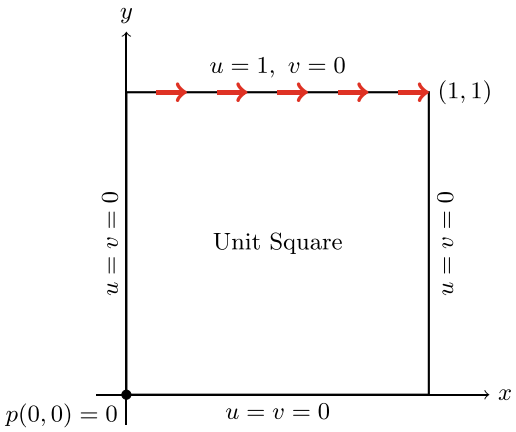

Fig. 5. Lid-driven cavity flow with boundary conditions. Source: elaborated by the authors.

For this problem, we employed a neural network with four hidden layers, each comprising 125 nodes, using the hyperbolic tangent activation function.

The network was initially trained for 30,000 iterations using the Adam optimizer, followed by an additional 15,000 iterations using the L-BFGS optimizer.

Figure 6 shows the heatmaps of velocity and pressure, where the typical laminar flow pattern is clearly visible. Figure 7 compares the model's results with the reference values from [4]. Compared to these reference values, we obtained an infinity norm of 8.327×10^{-1} for the velocity vector.

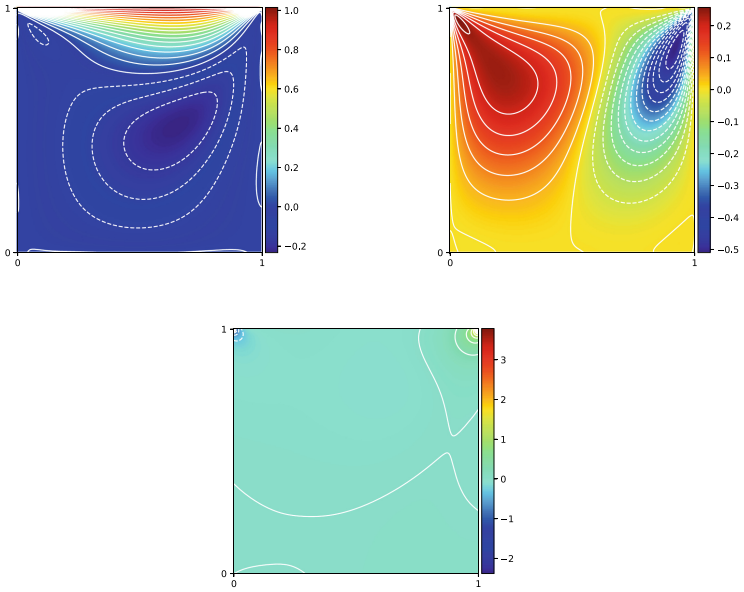

Fig. 6. Horizontal velocity (left), vertical velocity (right), pressure (bottom).

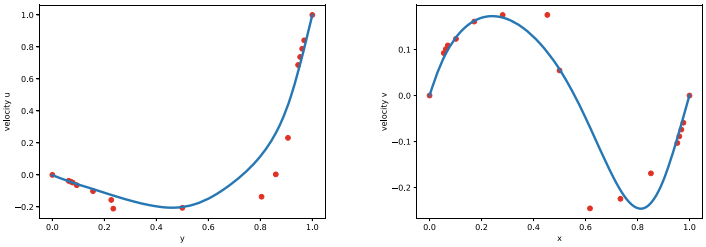

Fig. 7. Horizontal velocity (left) and vertical velocity (right) model solutions compared to reference values.

4.5 Inverse Problems

The four previous examples discussed focus on solving the mathematical models in a forward manner, that is, by determining the behavior of the system based on known initial and boundary conditions, as well as parameters. An inverse problem seeks to determine unknown parameters or inputs of a system based on observed data. These problems arise in various scientific fields, including medical imaging (such as medical tomography, where internal body structures are reconstructed from X-ray measurements), geophysics (in seismic imaging, where Earth's subsurface properties are inferred from wave propagation data), machine learning, and more. Unlike direct problems, where the system's response is computed from known parameters, inverse problems aim to reconstruct causes from effects.

We address an inverse problem associated with the SIR epidemiological model defined by equations (11)–(12). In this case, we are given the following data:

$$
\begin{aligned}
S_{observed} &= (S(t_1), S(t_2), \cdots, S(t_n)),\\
I_{observed} &= (I(t_1), I(t_2), \cdots, I(t_n)),\\
R_{observed} &= (R(t_1), R(t_2), \cdots, R(t_n)),
\end{aligned}
$$

over the time dataset $t_1, t_2, \cdots, t_n$. The goal is to determine the optimal parameters, β and γ, in the system (11)–(12) that best fit the sampled data.

The observed data refer to the Covid-19 pandemic in the state of Espírito Santo, Brazil, from January 2020 to January 2021. The data was collected from the official monitoring platform [6]. The network architecture employed here has the same configuration as that used for the direct problem in Sect. 4.3. The estimated values for the model parameters are $\beta = 0.0350$ and $\gamma = 0.0333$. Figure 8 displays the normalized real data and the predicted values obtained from the model. It is worth noting that, visually, the values are in good agreement.

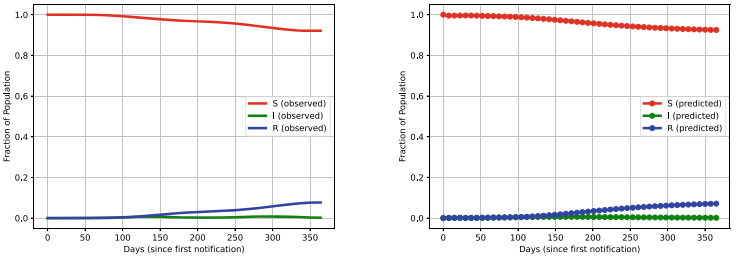

Fig. 8. Observed data and PINNs prediction for the SIR model.

5 Conclusions

Physics-Informed Neural Networks have emerged as a promising alternative to traditional methods for solving differential equations. Our intent with this paper

was to provide a comprehensive and accessible introduction to PINNs, targeting readers who may not be familiar with neural networks or machine learning techniques. Through a series of illustrative examples, ranging from simple ordinary differential equations to complex partial differential equation systems, we demonstrated the versatility and effectiveness of PINNs in solving problems from fields such as epidemiology and fluid dynamics.

One of the greatest advantages of PINNs is their ease of implementation, requiring only the definition of a residual function to be incorporated into the network loss function. Python libraries like DeepXDE facilitate that process even more, by providing an easy-to-use interface to define the loss function, boundary and initial conditions, and the domain of interest. Nevertheless, as discussed in [28], a significant limitation is that the accuracy of PINNs solutions tends to be inferior compared to traditional methods, such as the finite element method, particularly in the case of highly stiff or multiscale problems.

Despite these challenges, the PINNs methodology remains in its infancy, with many open research gaps to be explored. Improving neural network architectures, discovering more effective activation functions, developing adaptive optimization strategies, and integrating techniques from broader machine learning domains, such as convolutional networks, recurrent structures, and attention mechanisms, constitute promising directions for future research. Advancements in these areas are expected to significantly enhance the performance, robustness, and applicability of PINNs across a broader range of scientific and engineering problems.

As future work, we plan to explore examples using alternative network topologies and architectures and to solve problems involving irregular boundaries, as well as three-dimensional unsteady transport phenomena, including both compressible and incompressible flows.

Disclosure of Interests. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Abadi, M., et al.: TensorFlow: a system for large-scale machine learning. In: 12th USENIX Symposium On Operating Systems Design and Implementation (OSDI 2016), pp. 265–283 (2016)
2. Chollet, F.: Keras. In: Proceedings of the 2015 Conference on Machine Learning and Systems (MLSys), pp. 1–6 (2015)
3. Duarte, D., de Lima, P., de Araújo, J.M.: Outlier-resistant physics-informed neural network. Phys. Rev. E **111**, L023302 (2025). https://doi.org/10.1103/PhysRevE.111.L023302. https://link.aps.org/doi/10.1103/PhysRevE.111.L023302
4. Ghia, U., Ghia, K., Shin, C.: High-resolutions for incompressible flow using the Navier-Stokes equations and a multigrid method. J. Comput. Phys. **48**(3), 387–411 (1982). https://doi.org/10.1016/0021-9991(82)90058-4. https://www.sciencedirect.com/science/article/pii/0021999182900584
5. Goswami, S., Anitescu, C., Rabczuk, T.: Deep learning for computational mechanics. Comput. Methods Appl. Mech. Eng. **404**, 115783 (2023)

6. Governo do Estado do Espírito Santo, Brasil: Painel covid-19 - espírito santo (2024). https://coronavirus.es.gov.br/painel-covid-19-es. Accessed 03 Apr 2025
7. Gray, P., Scott, S.K.: Autocatalytic reactions in the isothermal CSTR: oscillations and chaos. Chem. Eng. Sci. **49**(1), 1–7 (1994). https://doi.org/10.1016/0009-2509(94)80044-4
8. Guasti Junior, W., Santos, I.P.: Solving differential equations using feedforward neural networks. In: Gervasi, O., et al. (eds.) Computational Science and Its Applications - ICCSA 2021, pp. 385–399. Springer, Cham (2021)
9. Haghighat, E., Juanes, R.: SciANN: a Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. Comput. Methods Appl. Mech. Eng. **373**, 113552 (2021). https://doi.org/10.1016/j.cma.2020.113552
10. Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., Yang, L.: Physics-informed neural networks: a survey. arXiv preprint arXiv:2103.15024 (2021)
11. Kermack, W.O., McKendrick, A.G.: A contribution to the mathematical theory of epidemics. Proc. Royal Soc. Lond. Ser. A Contain. Papers Math. Phys. Character **115**(772), 700–721 (1927). https://doi.org/10.1098/rspa.1927.0118
12. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
13. Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Networks **9**(5), 987–1000 (1998). https://doi.org/10.1109/72.712178
14. Landau, L.D., Lifshitz, E.M.: Fluid Mechanics, 2nd edn. Pergamon Press (1987)
15. Lee, H., Kand, I.S.: Neural algorithm for solving differential equations. J. Comput. Phys. **91**(1), 110–131 (1990). https://doi.org/10.1016/0021-9991(90)90007-N
16. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization (1989). https://doi.org/10.1007/BF01589116
17. Lu, L., Meng, X., Mao, Z., Karniadakis, G.E.: DeepXDE: a deep learning library for solving differential equations. SIAM Rev. **63**(1), 208–228 (2021). https://doi.org/10.1137/19m1274067. http://dx.doi.org/10.1137/19M1274067
18. Meade, A.J., Fernandez, A.A.: The numerical solution of linear ordinary differential equations by feedforward neural networks. Math. Comput. Model. **19**(12), 1–25 (1994). https://doi.org/10.1016/0895-7177(94)90095-7
19. Murray, J.D.: Mathematical Biology: I. An Introduction, 3 edn. Springer (2002)
20. Nikolopoulos, S., Kalogeris, I., Papadopoulos, V.: Non-intrusive surrogate modeling for parametrized time-dependent PDES using convolutional autoencoders (2021). https://arxiv.org/abs/2101.05555
21. Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, pp. 8024–8035 (2019)
22. Psichogios, D.C., Ungar, L.H.: A hybrid neural network-first principles approach to process modeling. AIChE J. **38**(10), 1499–1511 (1992). https://doi.org/10.1002/aic.690381003
23. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045
24. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics informed deep learning (part II): Data-driven discovery of nonlinear partial differential equations (2017). https://arxiv.org/abs/1711.10566

25. Rodrigo, M.R., Mimura, M.: Exact solutions of reaction-diffusion systems and nonlinear wave equations. Jpn. J. Ind. Appl. Math. **18**(3), 593–619 (2001). https://doi.org/10.1007/BF03167410
26. Toscano, J., Oommen, V., Varghese, A., Karniadakis, G.: From PINNs to PIKANs: recent advances in physics-informed machine learning. Mach. Learn. Comput. Sci. Eng. **1**(15) (2025). https://doi.org/10.1007/s44379-025-00015-1
27. Verhulst, P.F.: Notice sur la loi que la population poursuit dans son accroissement. Correspondance Mathématique et Physique **10**, 113–121 (1838)
28. Wang, S., Teng, Y., Perdikaris, P.: Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM J. Sci. Comput. **43**(5), A3055–A3081 (2021). https://doi.org/10.1137/20M1318043
29. Wang, S., Yu, X., Perdikaris, P.: When and why PINNs fail to train: a neural tangent kernel perspective. J. Comput. Phys. **449**, 110768 (2022). https://doi.org/10.1016/j.jcp.2021.110768