Data-driven Parameter Discovery of One-dimensional Burgers' Equation Using Physics-Informed Neural Network

1st Eduardo F. Miranda.
Applied Computing Post-graduation Program (CAP/INPE), National Institute for Space Research (INPE).
São José dos Campos SP, Brazil.
efurlanm@gmail.com.

2nd Roberto Pinto Souto.
National Laboratory for Scientific Computing (LNCC).
Petrópolis RJ, Brazil.
rpsouto@lncc.br.

3rd Stephan Stephany.
Coordination of Applied Research and Technological Development (COPDT/INPE), National Institute for Space Research (INPE).
São José dos Campos, Brazil.
stephan.stephany@inpe.br.

***Abstract*—This work shows the use of a data-driven parameter discovery for a Partial Differential Equation (PDE) using a Physics-Informed Neural Network (PINN). This approach is then compared to some standard numerical models for parameter discovery. The chosen toy problem is the one-dimensional Burgers' equation, a PDE with spatial and temporal derivatives, which is generally solved by a numerical model. However, recent studies propose the use of Deep Neural Network (DNN) for solving a PDE or for data-driven parameter discovery of the PDE. In any case, the training phase of DNN requires a high number of sample points in order to obtain accurate solutions. In this way, PINNs, which are DNN embedding underlying physical equations as prior knowledge, were proposed in order to make feasible the use of a lower number of sample points in the training phase. Usually, the physical equations are used in the PINN loss function. Like any DNN, PINNs can also be considered universal function approximators. The accuracy and processing times for estimating the parameters of the Burgers' equation are then compared for the data-driven parameter discovery using an MLP-based PINN, and also for some estimators based on standard numerical models. In addition, the influence of the number of sample points, MLP architecture and hyperparameters was also analyzed for the PINN approach.**

***Index Terms*—Machine Learning, System Identification, PINN, HPC**

I. Introduction

Many simulations are mathematically modeled by Partial Differential Equations (PDEs), which have derivatives in space and time. Typically, the coefficients of these derivatives are unknowns that express physical properties of the problem being modeled by the PDE, which is usually solved by standard numerical models (NM) like the finite difference method. Recent works proposed for solving a PDE or for data-driven parameter discovery of the PDE using Deep Neural Network (DNN). The universal approximation theorem states that a DNN can approximate any continuous function, as long as the network has a sufficient number of hidden layers and employs nonlinear activation functions [1]. In any case, for parameter resolution or discovery, the DNN training input is given by sample points, which compose a subset of the full set of known points of the function in the space and time domain. These sample points can be conveniently selected in order to increase its number of data related to initial (IC) and boundary conditions (BC). These sample points are then called Collocation Points (CP) [2], a name that came from standard NMs.

However, either for solving a PDE or for parameter discovery of the PDE, the training phase of DNN requires a high number of sample points in order to obtain accurate solutions. Sample points can be obtained either by observation or using synthetic data from a known model. Physics-Informed Neural Network (PINN), which are DNN embedding underlying physical equations as prior knowledge, were proposed in order to make feasible the use of a lower number of sample points in the training phase.

PINNs can be used for solving the direct problem, also called inference or solution, where the PDE and parameters are known, but not the result of the simulation. In the inverse problem, also called system identification or discovery, the CPs are used to discover the PDE parameters that best fit the exact dataset, thus finding the governing equations that rule the considered dynamic system modeled by the PDE. Dynamic systems are a diverse and well-studied class of mathematical objects used to model systems that change over time. Once identified, these equations can be utilized to predict future states, inform control inputs, or facilitate theoretical research using analytical approaches. Also considering the inverse problem, in the case of noisy datasets, PINNs can also perform data-driven parameter discovery, thus obtaining a more accurate model that allows to reproduce the set of sample points with less or no noise.

PINNs are employed for unsupervised learning when trained only on physical equations, and for supervised learning when dealing with noisy data or solving an inverse problem, and generally the related physical equations are used in the loss function [3].

Furthermore, PINN can be used in cases where the model

(or the PDE that describes it) is known, to reduce the size of the dataset required to train the DNN, thereby increasing efficiency, or when there is noise in the sample, and we want the underlying physical law to help deal with it.

This work evaluates the data-oriented discovery of parameters of the uni-dimensional Burgers' equation (1D Burgers). This equation shapes the velocity of a viscous fluid and is a fundamental PDE with spatial and temporal derivatives that can be obtained from the Navier-Stokes equations for the velocity field, removing the term from the pressure gradient. For small values of viscosity parameters, the Burgers' equation can result in shock formation that is notoriously difficult to solve using standard NMs. The PINN data-oriented parameter discovery seeks to answer the question: "What is the ideal combination of hyperparameters and dataset for this specific problem in order to find the best model for the expected result?".

The work presents two groups of problems, the first compares the discovery of parameters using a PINN method and 4 NM, and the second evaluates the PINN discovery results varying hyperparameters and dataset sizes in terms of accuracy and training processing time. The dataset composed of the full set of exact values was generated by the Gaussian Quadrature Method (GQM). A subset of CP was then extracted from this dataset. The dataset contains the fields x representing space, the field t representing time, and the field $u(t,x)$ representing the exact velocity.

Considering future works, the data-driven parameter discovery of PDEs by PINNs is relatively recent, but this approach may be applied to replace some specific processing-demanding modules that are part of the physics of NMs used for weather and climate prediction.

A. A brief state-of-the-art review on PINN

Raissi et al. (2019) [4] published an article on PINNs with 8956 citations (as of May 2024). The paper describes PINNs as DNNs trained to tackle supervised learning tasks when dealing with noisy data or inverse issues, and unsupervised learning when trained simply on physical equations, but conforming with physical rules, which are commonly described by nonlinear PDEs [3].

It also describes the use of DNNs to solve PDEs and obtain physics-informed surrogates of the physical model that are fully differentiable in all coordinates and free parameters. PINNs form a new class of data-efficient universal function approximators, which can be effectively trained using small datasets, and which may encode any underlying physical law.

DNN training data can be randomly sampled from observational data, or through simulations using synthetic data generated by a NM. Except for the latter, as long as a sufficient number of CP is available, a standard DNN can solve the PDE, but otherwise a PINN would be required. A PINN uses a specific loss function incorporating the related PDE and its parameters, in such a way that during the training phase using the set of CP, the applicable physical law is incorporated [3].

The most common PINN architectures are Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Newer architectures are Auto-Encoder (AE), Deep Belief Network (DBN), Generative Adversarial Network (GAN) and Bayesian Deep Learning (BDL) [5].

Cuomo et al. (2022) [3] published an article with 737 citations (as of April 2024) that provides a complete analysis of PINNs, and discusses the benefits and drawbacks of various PINN variations, including variational hp-VPINN, conservative PINN (CPINN), and physically constrained DNNs (PCNN). Additionally, PINNs can function as Reinforcement Learning (RL) agents. According to the same study, most of the research focused on PINN customization through the use of different activation functions, DNN architectures, gradient optimization techniques, and loss functions. Many PINN applications have been proposed and, according to the case, a PINN implementation can be used with advantage as an substitute to a standard numerical method, as in the case of the Finite Element Method (FEM).

Considering that PINNs were recently proposed, there is still room for improvements and also for new theoretical considerations. As a consequence, many PINN implementations depend on numerical experimentation, i.e. use of trial and error attempts.

Kim et al. (2021) [6] proposed a general taxonomy of the conceptual levels of what is called deep learning informed approaches, based on a literature review about dynamic systems: (i) what type of DNN is used, (ii) how physical knowledge is represented, and (iii) how physical information is integrated. The PINN is not the only DNN approach used to solve PDEs, although it seems to be the leading one in terms of number of articles. The PINN mainstream is still the PDE direct problem, but the number of works proposing PINNs to solve PDE inverse problems has been increasing. For instance, the use of CP is emphasized in some articles [4], [7], [8]. Other works present the Conservative PINN (CPINN) [9], and the Physically Restricted Neural Networks (PCNNs) [10]–[12].

PINNs may model the PDE with unknown initial and boundary conditions (called soft BC), as detailed in Raissi et al. [4], a work that proposed the name Physics-Informed Neural Network (PINN). There are also PCNNs, a class of *data-free* PINNs, which impose known initial and boundary conditions (hard BC) via a customized DNN architecture that also include the PDE in the loss function.

Recently, numerous frameworks have been presented, including the Deep Ritz Method (DRM) [13], in which the loss function value is defined as the energy required to solve the problem. There are alternative implementations based on the Galerkin method, also known as the Petrov-Galerkin method, in which the loss function value is calculated by multiplying the residue by a test function. This approach results in the Deep Galerkin Method (DGM) when the residue is volumetric [14]. Given that a Galerkin technique is utilized with CP, it can be considered a version of PINN, namely a hp-VPINN (Kharazmi et al.) [15].

Challenges and future prospects for PINN include its use and implementation for a variety of application involving

real-world equations. Applications include everything from convergence and stability to boundary condition management, DNN design, general PINN architecture design, and more. PINN have the potential to be a useful tool for solving high-dimensional PDEs that are important in physics, engineering, finance, and other areas. However, PINNs generally cannot accurately approximate PDE solutions in comparison to other specific NMs that are optimized for a single PDE [3]. It is worth to note that some PINN implementations that appear in the literature lack reproducibility and documentation, restricting new potential users to employ them.

The extension of PINN applications for 2D or 3D problems poses additional challenges since training complexity grows, requiring more complex architectures and larger batch sizes. As a consequence, training may be constrained by GPU memory, and longer training times are required for convergence to the solution [16]. There is a trend to include PINN into standard libraries written in Fortran and C/C++, as well as integrating PINN solvers into older high performance computing (HPC) applications [17]. For instance, PINN may be implemented on modern HPC clusters using the Horovod distributed training framework [18].

In the area of weather and climate models, PINNs may accelerate processing times. A work by Chevallier et al. [19] presents a speedup of 7 using a PINN over a standard NM to obtain parameters for the Longwave Radiative Transfer model used by the European Center for Medium-Range Weather Forecasts (ECMWF). This shows the potential of using DNNs to obtain the parametric representation in the numerical modeling of various atmospheric processes. Krasnopolsky et al. [20] also cites speedups between 10 and 10^5 using a PINN over a standard NM in the parametrization of physical models in oceanic and atmospheric NMs.

II. Methodology

The proposed group of problems requires data-driven discovery of parameters for a given 1D Burgers' equation, which estimates the velocity field u of a fluid along the dimension x and over time t (Equation 1). In this equation, two coefficients of the differential operators are defined as the parameters λ_1 and λ_2. The latter is the kinematic viscosity of the fluid (ν), and the velocity (u) subscripts denote partial differentiation in time and space, respectively, as u_t (du/dt), u_x (du/dx), and u_{xx} (d^2u/dx^2).

$$u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0 \tag{1}$$

In the considered problem, the Burgers' equation spatial and time domain, and the initial (IC) and boundary conditions (BC), are:

$$x \in [-1, 1],\ t \in [0, 1], \quad \text{(domain)}$$
$$u(0, x) = -\sin(\pi x), \quad \text{(IC)}$$
$$u(t, -1) = u(t, 1) = 0. \quad \text{(BC)}$$

Two sets of parameters were used, both using $\lambda_1 = 1$, but with $\lambda_2 = 0.01/\pi$ (expressing low viscosity) for the first, and $\lambda_2 = 0.1/\pi$ for the second (high or usual viscosity). It is intended to demonstrate that small viscosity values in the Burgers' equation can cause discontinuities, which are interpreted as shock waves, and which become more pronounced as the viscosity value decreases, thus making modeling using standard NMs more difficult as it requires more precise spatial and temporal resolutions. In any case, some datasets were generated for the PINN and NM implementations with problem size (dimension x by time t) 128x64, 256x100, 256x128, and 512x256 (some were used in specific cases). The datasets were generated using the numerical GQM, which is an iterative numerical algorithm that approximates the definite integral of a function as a weighted sum of the functions values at specified points of the integration domain [21]. The method takes into account the domain, initial and boundary conditions shown above for generating the dataset. The dataset contains the spatial values x, the time values t, and the exact solution $u(t, x)$.

A. PINN Implementation

A part of the code implemented in this work was adapted from the work of Raissi et al. (2019) [4], as described ahead in this work, using Python 3.7 and the framework TensorFlow 1.15 for execution on GPUs. The PINN code is based on an MLP, with an input layer of 2 neurons, a number of hidden layers ranging from 1 to 8, each one with a number of neurons ranging from 10 to 30, and an output layer with a single neuron. The loss function is based on Mean Squared Error (MSE), and is detailed in subsection II-B. Minimization of the loss function is performed by an optimization method, the generalized Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm, a quasi-Newton method. All hidden layers employ the hyperbolic tangent as the activation function.

Listing 1. Snippet of Python code that implements $u(t, x)$, seen in the listing as `net_u(t, x)`.

```
def neural_net(X, weights, biases):
  num_layers = len(weights) + 1
  H = 2.0 * (X - lb) / (ub - lb) - 1.0
  for l in range(0, num_layers - 2):
    W = weights[l]
    b = biases[l]
    H = tf.tanh(tf.add(tf.matmul(H, W), b))
  W = weights[-1]
  b = biases[-1]
  Y = tf.add(tf.matmul(H, W), b)
  return Y

def net_u(t, x):
  u = neural_net(tf.concat([x, t], 1), weights, biases)
  return u
```

The implemented DNN MLP uses the L-BFGS optimizer provided by the SciPy[1] framework, which was configured to a maximum of 50,000 iterations (a value commonly used by SciPy for such optimizer) and also to stop iterations when the hardware floating point precision lower limit is reached, in this case IEEE-75432 single precision binary floating point. The batch size used is the same size as the CP set, which results in

[1] https://docs.scipy.org/doc/scipy/reference/Optimize.minimize-bfgs.html

one iteration per training epoch. Some code snippets extracted from the implementation are shown in the Listings 1, 2, and 3.

Listing 2. Snippet of Python code that implements $f(t, x)$, seen in the listing as `net_f(x, t)`. The `net_u` function is shown in Listing 1. The `tf.gradients` is part of the TensorFlow framework and is used in the gradient-based training and optimization algorithm.

```
def net_f(x, t):
  lambda_1 = lambda_1
  lambda_2 = tf.exp(lambda_2)
  u = net_u(t, x)
  u_t = tf.gradients(u, t)[0]
  u_x = tf.gradients(u, x)[0]
  u_xx = tf.gradients(u_x, x)[0]
  f = u_t + lambda_1 * u * u_x - lambda_2 * u_xx
  return f
```

Listing 3. Snippet of Python code that implements the loss function (`loss`). The `u` is the exact solution. The `net_u` function is shown in Listing 1. The `net_f` function is shown in Listing 2.

```
u_tf = u  # exact solution
u_pred = net_u(t, x)  # predicted solution using the DNN
f_pred = net_f(t, x)  # predicted solution using the PDE
loss = (tf.reduce_mean(tf.square(u_tf - u_pred)) +
            tf.reduce_mean(tf.square(f_pred)))
```

As already mentioned, in the proposed approach, in a first step, the PINN is trained to obtain the parameters of the differential operators, and then the resulting PINN model, which incorporates these parameters, is used to generate a 1D velocity field in the space domain and of time, which will be compared with the exact 1D field generated by GQM.

B. PINN parameter discovery

The PINN training dataset is provided by a set of CPs that were randomly selected from the exact dataset generated by the GQM, for the considered domain and time interval. For a given training iteration k, the value of the loss function to be minimized is given by the sum of two Mean Square Errors (MSE), MSE_u and MSE_f (Equation 3), which can be seen in Figure 1, and the implementation is related to the code snippet in the Listing 3.

The MSE_u represents how well the PINN output matches the exact data points of the solution for the set of CPs. It is calculated as the mean squared error of the difference between the exact solution and the PINN predicted output for each CP of the set. In the equation, N is the number of CPs, the term $[u(t_u^i, x_u^i)]$ refers to the exact solution for each CP, and the term $[u^i]$ refers to if the solution predicted by DNN, thus the difference $[u(t_u^i, x_u^i) - u^i]$ measures the prediction error. The MSE_u can be seen in the Figure 1 and in the Listing 1.

The MSE_f represents how well the automatically calculated derivatives match the actual PDE terms at the CPs within the domain. The mean squared error of the physics-based regularization term ($[f(t_u^i, x_u^i)]$) enforces the compliance to the governing physical equations or constraints. It can be seen in the Figure 1 and in the code snippet shown in the Listing 2.

The MSE is used in the gradient descent optimization algorithm. The algorithm includes derivatives and automatic differentiation [22] to calculate the gradient[2] of the error function with respect to the parameters of the PINN, to update the network weights and biases at each iteration. The gradient provides guidance to adjust the parameters in value and direction in order to minimize the error.

$$f := u_t + \lambda_1 u u_x - \lambda_2 u_{xx} \tag{2}$$

$$MSE^k = MSE_u^k + MSE_f^{k-1} \tag{3}$$

where

$$MSE_u^k = \frac{1}{N}\sum_{i=1}^{N} |u(t_u^i, x_u^i) - u^i|^2$$

and

$$MSE_f^{k-1} = \frac{1}{N}\sum_{i=1}^{N} |f(t_u^i, x_u^i)|^2$$

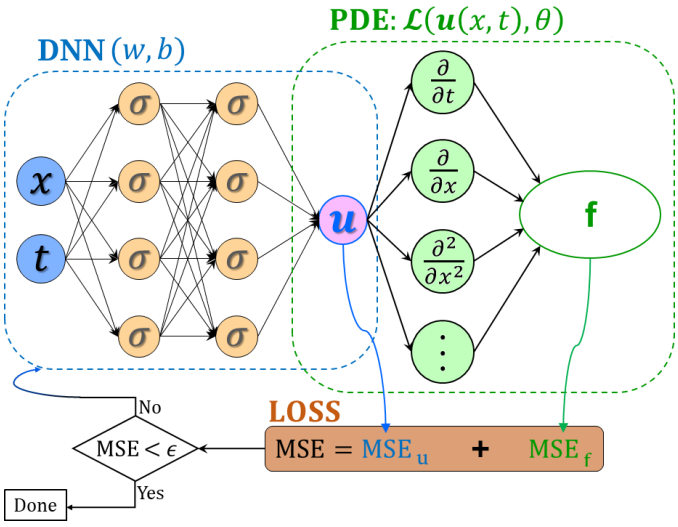

Fig. 1. Schematic showing an example structure and loss function, for the PINN approach. The symbol θ represents the optimization algorithm. The ϵ represents the smallest threshold error in the optimization algorithm. Source: Adapted from Meng et al. (2020) [7].

According to the work of Raissi et al. (2019) [4], the standard division of the CP dataset into training, validation and testing is not used in this specific PINN model. Instead, a random sample of CPs from the dataset is taken and used to train the PINN. During training, a specific validation set is not used. After training, the network is used to generate two results: (*i*) the discovery of the PDE parameters, and (*ii*) a prediction of the PDE solution.

Once the parameters have been discovered, in a second step, the resulting PINN model is used to predict the velocity field for all points in the defined space and time domain. This predicted velocity field can then be compared with the exact velocity field (generated by GQM), using a relative error to evaluate the accuracy of the prediction, as suggested in Xu et al. (2022) [23]. Such a relative error is defined by Equation 4.

[2] https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/gradients

$$R_{L2} = \frac{\|\widehat{U} - U\|}{\|U\|} \tag{4}$$

The $\|U\|$ represents the Euclidean L2 norm, which is given by the Euclidean distance from the vector coordinate to the origin of the vector space, i.e. it is the square root of the sum of the squared components of the vector. In the one-dimensional case, the L2 norm is simply given by the value of the velocity. The $\|\widehat{U} - U\|$ represents the application of the L2 norm to the deviation of the estimated velocity field $\widehat{U}$, with respect to the exact velocity field U, considering all dataset points in the space and time domains.

The PINN hyperparameters, like the number of hidden layers $N_l (l = 1, 2, ...)$, and the number of neurons in each layer $N_{le} (e = 1, 2, ...)$, can also be optimized by numerical experimentation (trial and error). Optimization of N_l and N_{le}, and eventually of other hyperparameters not employed here, is still an unsolved problem, being made empirically [23].

As any neural network, PINNs are subjected to overfitting or underfitting. Overfitting means that the DNN performs very well using the training set, but lacks performance using different, new input data of the test set, i.e. it does not generalize. On the other hand, underfitting means that the model performs poorly on both the training and test sets. Obviously, both overfitting and underfitting imply in poor performance of the neural network [24].

C. NM-based parameter discovery

The inverse problem solved by the implemented PINN to find the coefficients of the differential operators is then compared to the chosen NM-based method, the Sparse Identification of Nonlinear Dynamical Systems (SINDy) method [25]. This is a method developed for solving the identification problem of dynamical systems, which are systems that change over time. SINDy uses sparse regression to create a linear combination of basis functions to captures the dynamic behavior of the considered physical system. It employs observational or synthetic data to obtain the governing equations that fit such data of the dynamical systems. The discovered equations can then be used to predict future states, inform control inputs, or enable theoretical research using analytical techniques [26].

Assuming a physical system that evolves in time t and space x defined by a collection of measurements $x(t) \in \mathbb{R}$. Time evolution of $x(t)$ is modeled by SINDy using a nonlinear function $f(x(t))$ (Equation 5). The vector $x(t) = [x_1(t), x_2(t), \ldots x_n(t)]^\top$ represents the state of the physical system at time t. The problem solved by SINDy is shown in Equation 6, where the function $f(x(t))$ is expressed as a matrix Θ of basis functions applied to the input data X, i.e. $\Theta(X)$, which is then multiplied by a matrix Ξ of coefficients that weighs these basis functions. Along the iterations, these coefficients are optimized until achieving convergence by means of an objective function (Equation 7) that assess the correctness of the solution given by $[\Theta(X)\,\Xi]$ at a given iteration. It is assumed that $f(x(t))$ is usually sparse in the space of a suitable set of basis functions, i.e. that much of the coefficients of matrix Ξ are zero.

$$\frac{d}{dt}x(t) = f(x(t)) \tag{5}$$

$$\dot{X} = \Theta(X)\,\Xi \tag{6}$$

SINDy applies a sparsity-promoting regression (such as LASSO [27]) to a library of nonlinear candidate functions produced from the measurements, but restricting the number of basis functions to obtain a compact representation of the function and avoid overfitting. SINDy has been effectively used to solve a variety of problems, including linear and nonlinear oscillators, chaotic systems, and fluid dynamics [26]. SINDy is implemented by the sparse regression package PySINDy[3] which includes several implementations for the sparse identification method of nonlinear dynamical systems from various authors, being comprehensive literature reviews available at [28] and [29]. Part of the code implemented in this work was reused and adapted from PySINDy library examples. PySINDy runs on CPU.

This work employs four of these implementations, known as Sparse Regression Optimizers (SRO), which are part of the parameter discovery package and can be selected separately, producing a result for each choice: Sequentially Threshold Least Squares (STLSQ), Orthogonal Least Squares of Forward Regression (FROLS), Sparse Relaxed Regularized Regression (SR3), and Sparse Stepwise Regression (SSR). To keep things simple, we'll treat each of these as a separate NM.

A code snippet of this framework is shown in Listing 4, where `optimizer` is the chosen optimization method (exemplified in the code as STLSQ), `SINDy` defines the model, `fit` discovers the parameters, and `print` shows the result.

Listing 4. Python code snippet that implements the NM for data-driven PDE parameter discovery. The STLSQ is one of the optimizers in the PySINDy framework.

```
optimizer = ps.STLSQ(threshold=2, alpha=1e-5,
     normalize_columns=True)
model = ps.SINDy(feature_library=pde_lib,
                 optimizer=optimizer,
                 feature_names=["u"])
model.fit(u, t=dt)
model.print()
```

The dataset used by SINDy for each of the 4 optimizers is the same used in the PINN method, but employing the full dataset instead of the set of CPs. Parameter discovery using PINN required training the network with the set of CPs, use of the Burgers' PDE in the loss function, etc. After training, the predicted velocity field in space and time is then compared to the exact field. In the case of SINDy, the complete dataset is used to obtain the parameters, with no further comparison [26]. However, parameters discovered by PINN and SINDy are compared in order to evaluate the accuracy of both approaches using the coefficients of the known PDE as reference. Although possible, some optimizations were not made to SINDy in order

[3] http://pysindy.readthedocs.io

to obtain better accuracy, being the default settings of the framework used. These possible optimizations can be performed as future work.

Except in the case of SINDy with the SR3 optimizer, the other optimizers employ the objective function that must be minimized, which can be seen in the Equation 7.

$$\| y - X w \|^2 + \alpha \| w \|^2 \tag{7}$$

Above, w is the weight matrix containing the set of basis functions that maps the dataset contained in X to the solution y, i.e. w corresponds to the 2nd member ($[\Theta(X)\,\Xi]$) of Equation 6, and α is a regularization coefficient applied to w, helping prevent overfitting.

In order to minimize the objective function, the STLSQ [26] algorithm uses ridge regression with sequential threshold, and iteratively runs the least squares algorithm, but masking elements below a specific threshold. The FROLS [30] algorithm is a greedy algorithm that iteratively selects the most correlated function in the library. The SSR [25] is also a greedy algorithm that iteratively eliminates the smallest coefficients. A greedy algorithm is any algorithm that uses the heuristic of choosing the locally optimal solution at every iteration [31]. Regardless of not assuredly finding a global optimal solution, greedy heuristics can produce locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time. Along the iterations, the SSR and FROLS algorithms respectively truncate (or add) one nonzero coefficient at each algorithm iteration.

The SR3 algorithm [32] is a relaxed and sparse regularized regression with linear (dis)equality constraints that attempts to minimize the objective function (Equation 8).

$$\begin{aligned} &0.5\|y - Xw\|^2 + \lambda R(u) + (0.5/\nu)\|w - u\|^2 \\ &\text{subject to } Cw = d \end{aligned} \tag{8}$$

over u and w, where $R(u)$ is a regularization function, C is a constraint matrix, and d is a vector of values.

D. Computing environment

The PINN and the four PySINDy codes[4] were executed in two different computing environments. It is important to note that the PINN model runs on GPU, while PySINDy runs on CPU.

The first environment is a local machine, a PC with a 6-core Intel i7 9750h CPU with 6 cores, 8 GB of main memory, and an NVIDIA GTX 1050 GPU (768 CUDA cores and 3 GB of memory). The second environment is the Santos Dumont (SDumont) supercomputer of the National Scientific Computing Laboratory (LNCC), more specifically a single Bull Sequana X1120 processing node with two 24-core Intel Xeon Gold 6252 Skylake 2.1 GHz processors (total of 48 cores), 384 GB of main memory and four Nvidia Volta V100 GPUs, although only one GPU was employed. Both computing environments included the Python[5] 3.7 interpreter and the TensorFlow[6] v1.15 machine learning platform.

III. Results

The results are divided into two parts of experiments: the first one compares the results of PINN with the 4 SINDy versions, and the second evaluates the PINN implementation for various hyperparameters and CP sizes. The first part uses datasets related to different problem sizes (x space versus t time discretization): 128x64, 256x128, and 512x256, while the second part uses only a 256x100 problem size. The GQM is used to generate the dataset corresponding to the exact solution (reference), which are then employed by the SINDy versions, and also to extract the set of CPs used by the PINN implementation. The MLP architecture used for the PINN implementation in the first part has one input layer, 3 hidden layers with 20 neurons each, and one output layer. The PINN undergoes training and PDE parameters are estimated, solving the inverse problem, and the resulting model estimates the solution (space versus time field). The processing-time data obtained in the all experiments is given by the average of 3 executions.

A. Part I - Comparison of PINN and SINDy for parameter discovery

Tables I and II show the average of elapsed times of 3 runs on the local machine for the PINN (using GPU) and the 4 SINDy implementations (only CPU), for the 3 problem sizes and for the two cases of viscosity. In the case of PINN, training time to estimate the parameters and prediction time of the resulting 1D field using the trained model are shown. Processing times of SINDy versions are consistently lower than those of the PINN model, even considering that the latter executes using GPU. Therefore, future studies must consider the tradeoff between accuracy and processing times when comparing PINNs and numerical methods like SINDy.

Tables III and IV show the results obtained for the PDE parameter discovery, comparing the PINN and the 4 SINDy implementations (STLSQ, FROLS, SR3 and SSR, discussed in subsection II-C), and also considering 2 different viscosity values in the 1D Burgers' equation, $0.01/\pi$ and $0.1/\pi$. As already commented, 3 sizes of problems are considered: 128x64, 256x128 and 512x256.

Subscripts denote partial differentiation in time and space, e.g. u_x denotes du/dx, u_xx denotes d^2u/dx^2, and so on. At the top of each table is shown the *Exact PDE* 1D Burgers' equation (e.g., $u_t + 1.0uu_x - 0.003183u_{xx} = 0$) used to generate the datasets.

In the case of PINN, training is done using 2,000 CPs randomly obtained from the dataset, regardless of the size of the problem. In the case of SINDy implementations, the complete dataset is used in the model to obtain the parameters. In III the viscosity value ($0.01/\pi$) is lower than in the second ($0.1/\pi$)

[4] The codes are available at http://github.com/efurlanm/pd1b24

[5] http://www.python.org

[6] http://www.tensorflow.org

Model	Elapsed [s]
128x64	
PINN Train	47.567
PINN Predict	0.371
STLSQ	0.031
FROLS	0.140
SR3	0.054
SSR	0.068
256x128	
PINN Train	53.033
PINN Predict	0.732
STLSQ	0.049
FROLS	0.071
SR3	0.098
SSR	0.086
512x256	
PINN Train	52.633
PINN Predict	3.067
STLSQ	0.105
FROLS	0.625
SR3	0.118
SSR	0.181

TABLE I

COMPARISON OF ELAPSED TIMES (AVERAGE OF 3 RUNS) FOR THE PINN AND THE 4 SINDY MODELS FOR KINEMATIC VISCOSITY OF THE FLUID OF $0.01/\pi$, AND FOR EXECUTION ON THE LOCAL MACHINE (PC).

Model	Elapsed [s]
128x64	
PINN Train	22.633
PINN Predict	0.361
STLSQ	0.013
FROLS	0.060
SR3	0.015
SSR	0.043
256x128	
PINN Train	36.100
PINN Predict	0.995
STLSQ	0.033
FROLS	0.074
SR3	0.015
SSR	0.060
512x256	
PINN Train	23.133
PINN Predict	3.020
STLSQ	0.086
FROLS	0.588
SR3	0.095
SSR	0.262

TABLE II

COMPARISON OF ELAPSED TIMES (AVERAGE OF 3 RUNS) FOR THE PINN AND THE 4 SINDY MODELS FOR KINEMATIC VISCOSITY OF THE FLUID OF $0.1/\pi$, AND FOR EXECUTION ON THE LOCAL MACHINE (PC).

in order to check if SINDy accuracy is compromised for small viscosity values, as confirmed by the results. PINN accuracy was good, even using the set of CPs extracted from the smaller dataset, and such accuracy improves with the problem size. III is for experiments with the higher viscosity value, showing that the accuracy of the SINDy implementations improved in comparison to the results of the preceding table, but are still lower than those obtained by PINN.

Correct PDE: 0.003183 u_xx - 1.0 uu_x (viscosity = 0.01/pi)

Model	Discovered equation and parameters
128x64 problem size	
PINN	0.0033735 u_xx - 0.99912 uu_x
STLSQ	0.06420 u + 0.00505 u_xx - 1.06304 uu_x + + 0.00469 uuu_xx - 0.00001 uu_xxx
FROLS	-0.418 u
SR3	0.064 u + 0.005 u_xx - 1.063 uu_x + 0.005 uuu_xx
SSR	0.064 u + 0.005 u_xx - 1.063 uu_x + 0.005 uuu_xx
256x128 problem size	
PINN	0.0031779 u_xx - 0.99942 uu_x
STLSQ	0.00395 u_xx - 1.00869 uu_x + 0.00126 uuu_xx
FROLS	0.015 u + 0.004 u_xx - 1.003 uu_x
SR3	0.004 u_xx - 1.009 uu_x + 0.001 uuu_xx
SSR	0.011 u + 0.004 u_xx - 1.011 uu_x + 0.001 uuu_xx
512x256 problem size	
PINN	0.0031403 u_xx - 0.99850 uu_x
STLSQ	0.00339 u_xx - 1.00534 uu_x + 0.00041 uuu_xx
FROLS	0.006 u + 0.004 u_xx - 1.006 uu_x
SR3	0.003 u_xx - 1.005 uu_x
SSR	0.006 u + 0.003 u_xx - 1.007 uu_x

TABLE III

COMPARISON OF THE RESULTS OF THE PARAMETER DISCOVERY FOR THE 1D BURGERS' EQUATION USING PINN (IN BLUE) AND THE 4 SINDY VERSIONS (KINEMATIC VISCOSITY OF THE FLUID OF $0.01/\pi$).

Correct PDE: 0.03183 u_xx - 1.0 uu_x (viscosity = 0.1/pi)

Model	Discovered equation and parameters
128x64 problem size	
PINN	0.0318582 u_xx - 0.99928 uu_x
STLSQ	0.03222 u_xx - 1.00012 uu_x
FROLS	0.032 u_xx - 1.000 uu_x
SR3	0.032 u_xx - 1.000 uu_x
SSR	0.003 u + 0.032 u_xx - 1.002 uu_x
256x128 problem size	
PINN	0.0318372 u_xx - 0.99924 uu_x
STLSQ	0.03193 u_xx - 1.00002 uu_x
FROLS	0.032 u_xx - 1.000 uu_x
SR3	0.032 u_xx - 1.000 uu_x
SSR	0.001 u + 0.032 u_xx - 1.000 uu_x
512x256 problem size	
PINN	0.0318292 u_xx - 0.99936 uu_x
STLSQ	0.03186 u_xx - 1.00001 uu_x
FROLS	0.032 u_xx - 1.000 uu_x
SR3	0.032 u_xx - 1.000 uu_x
SSR	0.032 u_xx - 1.000 uu_x

TABLE IV

COMPARISON OF THE RESULTS OF THE PARAMETER DISCOVERY FOR THE 1D BURGERS' EQUATION USING PINN (IN BLUE) AND THE 4 SINDY VERSIONS (KINEMATIC VISCOSITY OF THE FLUID OF $0.1/\pi$).

The next subsections present the solution field $u(t,x)$ generated by the PINN implementation for the 3 problem sizes, and for the 2 viscosity values. For each case, there are figures showing the predicted solution $u(t,x)$ with marks representing the CPs, and a particular snapshot at time $t = 0.5$ comparing the PINN prediction and the exact solution. These figures

allow a visual assessment of the accuracy of the solutions.

1) Problem size 128x64: Figures 2, 3, 4, and 5 show, for the problem size 128x64, the solutions for the viscosities $0.01/\pi$ and $0.1/\pi$. Figures 2 and 4 show the predicted spatio-temporal solution $u(t,x)$, and the CPs used to training are represented as dark marks on the graph. Since the dataset is relatively small and the number of CP is 2,000, the small marks are aligned and may not appear random at first. At the top and bottom of the figure, ($x = 1.0$ and $x = -1.0$), are the CPs related to the boundary conditions (BC) of the PDE, on the left side ($t = 0, 0$) are the CPs relative to the initial conditions (IC), and in the center of the figure ($x = 0.0$) the complex non-linear behavior (sometimes called formation of a shock wave) of the Burgers' equation with lower viscosity is represented. The remaining randomly selected CPs appear on the graph. Although it was not done this way, it is possible to select more points in the BC and IC regions to reinforce network training, if necessary.

Figures 3 and 5 show a snapshot in time comparing the superimposed solution for the PINN method and the exact numerical solution. It is possible to see that even using a few CPs, the PINN was able to accurately capture the complex non-linear behavior of the Burgers' equation, which presents the formation of an accentuated internal layer around t = 0.4, which is generally difficult to solve with traditional NM and requires a laborious spatio-temporal discretization of the equation. In this scenario, we can directly address the nonlinear problem without needing to commit to any prior assumptions, linearization, or local time step.

Due to the number of CPs being constant even when the size of the dataset varies, great variation in the discovery of parameters in the PINN model is not expected. In the case of the NMs seen in Tables III and IV, the variation is expected because the NM uses the complete dataset (does not use CP). In addition, there is also variation related to each NM algorithm. For the Table III with lower viscosity, it is possible to observe that for 128x64 (total of 8,192 data points) the accuracy of the NM was not good, unlike the PINN model which obtained good accuracy even with a smaller amount (2000 CPs) of training data. As for Table IV with higher viscosity, it is possible to observe that for 128x64 the NM showed better accuracy, especially FROLS and SR3, approaching the PINN model.

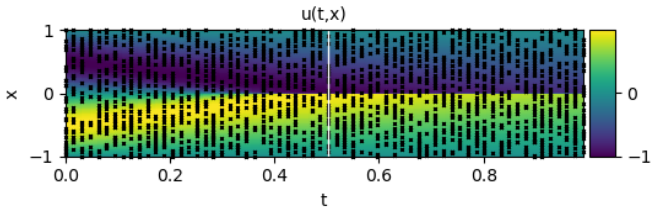

Fig. 2. PINN predicted solution $u(t,x)$ for viscosity $0.01/\pi$ and problem size 128x64.

2) Problem size 256x128: The Figures 6, 7, 8 and 9 show, for the problem size 256x128, the solutions for the viscosities $0.01/\pi$ and $0.1/\pi$. Much of what was presented in the previous section also applies to this section. In Figures 6 and 8 one can initially observe a better distribution of CPs in a more random way, due to the fact that the set of data to be larger and the size of the CP set remains at 2,000 CPs, allowing the CP to be distributed a little better in the graph. Figures 7 and 9 show the same behavior as the previous subsection, even because the amount of CP is the same. For the Table III with lower viscosity, and 256x128 problem size, the NMs were not accurate. For low viscosity, in most cases, regardless of the dataset, the PINN model showed better accuracy. For the Table IV with higher viscosity, and 256x64 problem size, most of the NMs behaved well, differently from what occurs in the case with lower viscosity.

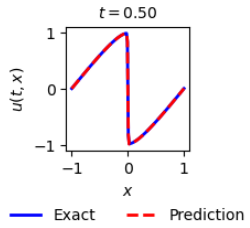

Fig. 3. Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 128x64).

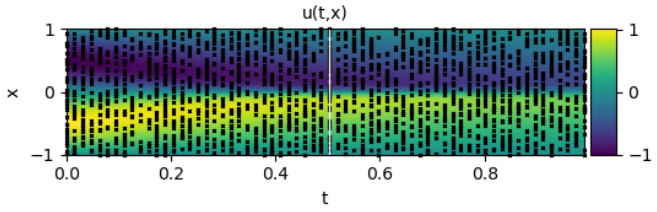

Fig. 4. PINN predicted solution $u(t,x)$ for viscosity $0.1/\pi$ and problem size 128x64.

3) Problem size 512x256: The Figures 10, 11, 12 and 13 show, for the problem size 512x256, the same behavior as the previous subsection. For Tables III and IV, 512x256 problem size, the NMs showed similar behavior to the previous section, with the SSR method standing out.

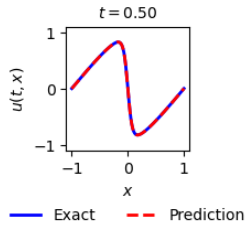

Fig. 5. Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.1/\pi$ and problem size of 128x64).

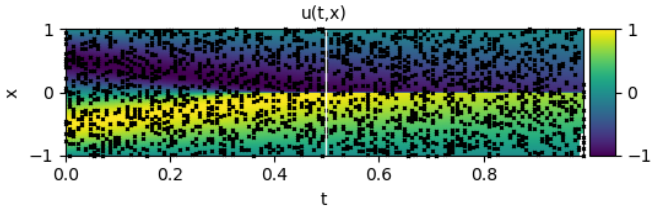

Fig. 6. PINN predicted solution $u(t,x)$ for viscosity $0.01/\pi$ and problem size 256x128.

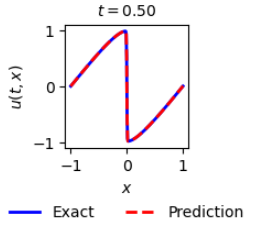

Fig. 7. Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 256x128).

B. Part II - Evaluation of PINN according to hyperparameter set and size of CP set

PINN processing time and accuracy is evaluated using the model generated with different sets of hyperparameters and CP set sizes in experiments executed on SDumont. Resulting predicted 1D fields are presented in figures, which are similar to those of the previous section. Following, PINN accuracy and processing times are evaluated as a function of the number of neurons per hidden layer, the number of hidden layers, and also the size of the CP set.

1) Visual assessment of PINN: In this section, the network architecture is the same employed for the preceding comparisons between PINN and SINDy implementations (single input and output layers and 4 hidden layers with 20 neurons each). A visual assessment of predictive accuracy for problem size 256x100 and fluid viscosity of 0.01 is shown in Figure 14, with time t on the horizontal axis and spatial coordinate x on the vertical axis. The color scale refers to the velocity $u(t,x)$. The black dots on the graph represent the 2,000 CPs randomly generated from the dataset and used for training.

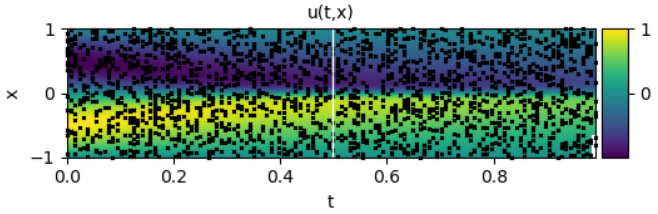

Fig. 8. PINN predicted solution $u(t,x)$ for viscosity $0.1/\pi$ and problem size 256x128.

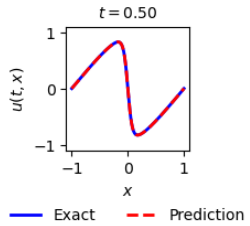

Fig. 9. Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.1/\pi$ and problem size of 256x128).

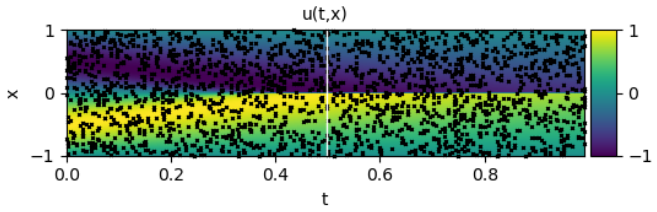

Fig. 10. PINN predicted solution $u(t,x)$ for viscosity $0.01/\pi$ and problem size 512x256.

Figure 15 shows a specific snapshot at $t = 0.5$, where it is possible to observe the overlapping solutions for PINN and GQM (exact). For this specific result, the equation obtained by PINN is $u_t + 0.99967uu_x - 0.0030988u_{xx} = 0$, while the exact PDE is $u_t + uu_x - 0,0031831u_{xx} = 0$. Thus, the PINN could identify the underlying PDE with good accuracy.

2) Influence of PINN Number of Layers and of Neurons: For the results presented below, the hyperparameters N_l (number of hidden layers) and N_{le} (number of neurons per hidden layer)

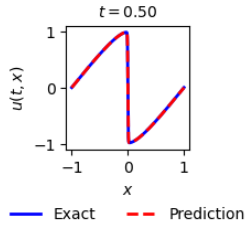

Fig. 11. Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 512x256).

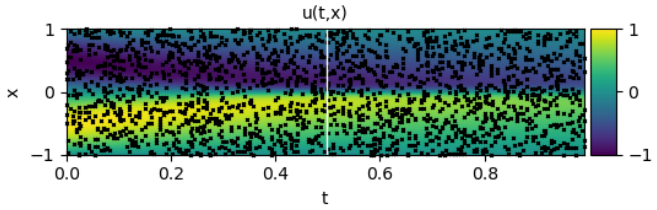

Fig. 12. PINN predicted solution $u(t,x)$ for viscosity $0.1/\pi$ and problem size 512x256.

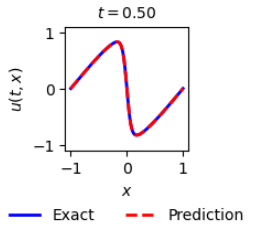

Fig. 13. Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.1/\pi$ and problem size of 512x256).

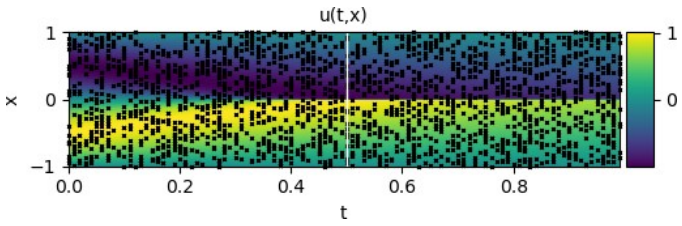

Fig. 14. PINN predicted solution $u(t, x)$ for viscosity $0.01/\pi$ and problem size 256x100 (black dots denote the 2,000 randomly assigned CPs).

were varied, as well as the number of CPs for training. The Table V shows the relative L2 errors and training times of the PINN, for different hyperparameters used: 10, 15, 20, 25, and 30 neurons per hidden layer, and 1, 2, 4, 6, and 8 hidden layers. The number of CPs was set at 2,000. All values shown here are the average of 3 runs. In this table it is possible to observe that there is a tendency for the best values to be concentrated in the center, probably because there is a problem of underfitting or overfitting in the values at the edges of the table. For future work, it would be interesting to better evaluate this behavior. One of the highlights is that the smallest error is obtained with 6 hidden layers, not 8. In this specific case, increasing the number of layers not only does not increase accuracy, it also worsens performance.

The Figure 16 shows that the error for 1 hidden layer is high compared to the other number of layers. For 2 layers, there is a significant improvement in accuracy. For 4, 6, and 8 the gain in precision is not that great, but the curves are similar and are in the region of greater accuracy, showing that they would be the best choices.

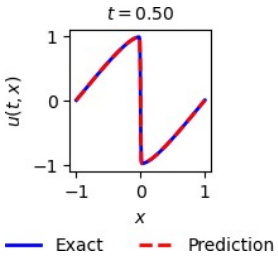

Fig. 15. Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 256x100).

The Figure 17 shows, for 4 hidden layers, a tendency to describe a curve that resembles a parabolic, with a minimum processing time of 20 neurons per hidden layer. This is probably due to the problem of underfitting and overfitting occurring at the beginning and at the end of the curve. Once again, for future work, it would be interesting to better evaluate this behavior.

Hidden layers	Number of neurons per hidden layer				
	10	15	20	25	30
Relative L2 Error (%)					
1	18.54	17.77	17.80	17.53	17.47
2	3.70	2.52	1.52	1.77	1.59
4	0.30	0.41	0.44	0.39	0.16
6	0.22	0.19	0.10	0.18	0.17
8	0.26	0.13	0.19	0.16	0.23
Training - processing time (seconds)					
1	4.2	5.4	5.0	21.7	9.4
2	35.9	51.8	39.3	55.5	70.7
4	51.2	43.1	33.4	40.9	47.4
6	59.7	40.3	42.5	35.2	38.6
8	58.7	60.0	58.6	54.4	84.5

TABLE V
RELATIVE L2 ERRORS AND DNN TRAINING TIMES FOR DIFFERENT NUMBER OF NEURONS AND HIDDEN LAYERS. ON THE COLOR SCALE, THE BEST VALUES ARE HIGHLIGHTED IN GREEN. THE SIMULATION RAN ON THE SDUMONT.

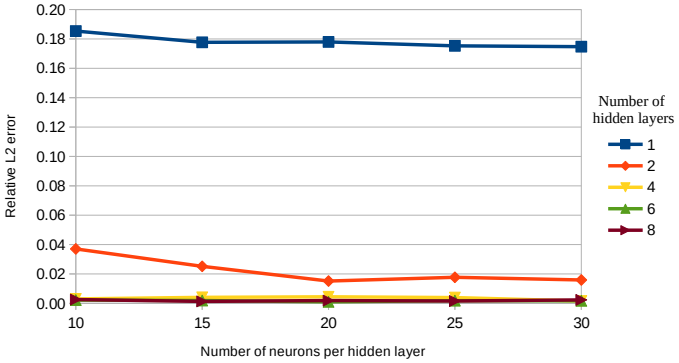

Fig. 16. Relative L2 error (%) in function of number of neurons and hidden layers. The simulation ran on the SDumont.

3) Influence of PINN Number of Neurons and Size of CP Set: The Table VI shows the relative L2 errors and training times of the PINN, for different hyperparameters and number of CP: 10, 15, 20, 25, and 30 neurons per hidden layer, and 400, 800, 1200, 1600, and 2000 CP. The number of layers was set at 8. All values shown here are the average of 3 runs. In this table, as in the previous one, it is possible to observe that there is a tendency for the best values to be concentrated in the center, probably because the problem of underfitting or overfitting is occurring in the values at the edges of the table. Once again, for future work, it would be interesting to better evaluate this behavior. One of the highlights is that considering the smallest

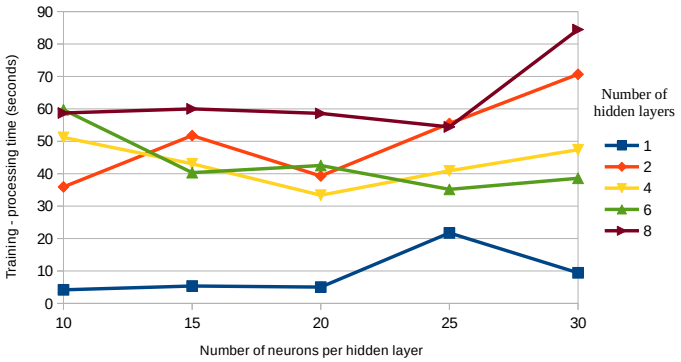

Fig. 17. Processing times (seconds) in function of number of neurons and hidden layers. The simulation ran on the SDumont.

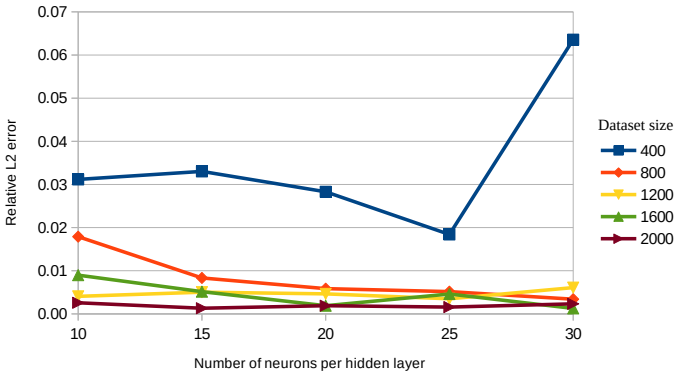

Fig. 18. Relative L2 error (%) in function of number of neurons and dataset size. The number of hidden layers is set to 8. The simulation ran on the SDumont.

error and the shortest processing time, the best dataset size is 1600, and the best number of neurons per hidden layer is 20.

The Figure 19 shows for most curves a tendency to describe a curve that resembles a parabolic, probably due to the problem of underfitting and overfitting occurring at the beginning and end of the curve. Once again, for future work, it would be interesting to better evaluate this behavior. The shortest processing time occurs for 15 neurons per hidden layer, and 1600 CPs.

The Figure 18 shows that the error for 400 CPs is high compared to the others. 800 CPs presents a significant improvement in accuracy, and the other curves are relatively close, not presenting such a relative large accuracy gain.

Dataset size	Number of neurons per hidden layer				
	10	15	20	25	30
Relative L2 Error (%)					
400	3.12	3.30	2.83	1.84	6.36
800	1.79	0.83	0.59	0.52	0.34
1200	0.41	0.50	0.46	0.35	0.61
1600	0.90	0.51	0.19	0.46	0.13
2000	0.26	0.13	0.19	0.16	0.23
Training - processing time (seconds)					
400	57.9	82.3	83.3	59.8	58.6
800	79.7	53.5	63.2	45.0	63.0
1200	63.7	52.2	43.8	42.1	56.8
1600	59.9	27.5	45.3	46.5	56.4
2000	58.7	60.0	58.6	54.4	84.5

TABLE VI
RELATIVE L2 ERRORS AND DNN TRAINING TIMES FOR DIFFERENT NUMBER OF NEURONS AND DATASET SIZE. THE NUMBER OF HIDDEN LAYERS IS SET TO 8. ON THE COLOR SCALE, THE BEST VALUES ARE HIGHLIGHTED IN GREEN. THE SIMULATION RAN ON THE SDUMONT.

4) PINN Prediction Times: Table VII shows the prediction times of the trained PINN model in function of different number of hidden layers (1, 4, 8) and different numbers of neurons per hidden layer (10, 20, 30). As expected, such times are very close, differently from the corresponding training times, which are higher and differ too much. For instance, in the case of 4 hidden layers and 20 neurons per layer, training time was 33.4 s, while prediction time was only 0.724 s, or 2.2%. This is a common issue with neural networks, showing the convenience of avoiding re-training the network whenever possible.

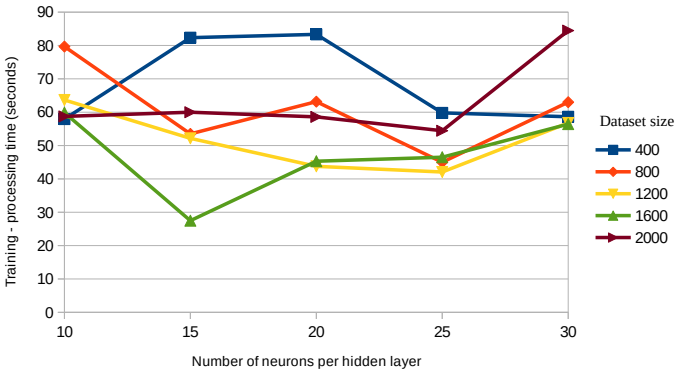

Fig. 19. Processing times (seconds) in function of number of neurons and dataset size. The number of hidden layers is set to 8. The simulation ran on the SDumont.

Number of hidden layers	Number of neurons per hidden layer		
	10	20	30
1	0.647	0.636	0.675
4	0.704	0.724	0.705
8	1.092	0.867	0.789

TABLE VII
PREDICTION TIMES FOR DIFFERENT NUMBER OF NEURONS AND HIDDEN LAYERS. ON THE COLOR SCALE, THE BEST VALUES ARE HIGHLIGHTED IN GREEN. THE SIMULATION RAN ON THE SDUMONT.

IV. CONCLUSIONS

Physics-Informed Neural Networks (PINNs) are a novel approach that combines data-driven and physics-based strategies. This work evaluated the inverse problem of data-driven discovery of PDE parameters using PINN and also a NM-based method for a given toy problem, the 1D Burgers equation. This equation models the velocity of a fluid along space and time. In addition, an evaluation was made of the PINN performance in terms of accuracy and processing times considering different MLP hyperparameters and CP set sizes. Adjustment of these

characteristics was performed by trial-and-error and indicated an optimal performance range of number of neurons per hidden layer, number of hidden layers, and size of the CP set.

Experiments were performed using different sizes of CP datasets for two different values of viscosity in the Burgers equation. It was shown that the PINN model, even trained with the relatively small subset of CPs achieved good accuracy, compared to the 4 SINDy versions, which employ the entire dataset. Therefore, for the considered test case, the PINN accurately reproduced the complex non-linear behavior of the Burgers' equation.

It was observed that the lowest viscosity value implied in the sharp discontinuities, associated with the formation of a shock wave, which was better modeled by the PINN than by SINDy. Considering the high-viscosity case, SINDy accuracy increased, but PINN accuracy was similar. However, in terms of execution times, SINDy versions were always faster, since the PINN implementation requires training.

As future work, it is proposed to explore other recent PINN-based approaches and/or implementations, as well as use other test cases involving direct and inverse problems associated to PDEs. Real world problems with limited-size or noisy datasets may also be focused. Another work would be to optimize the weather forecast model of the European Centre for Medium-Range Weather Forecasts (ECMWF) by means of developing a PINN-based version of its radiative module ecRad. This is a processing-demanding module that cannot be executed at every timestep of the ECMWF model. In addition, new frameworks and/or libraries for HPC can also be explored for CPU and GPU execution on the LNCC Santos Dumont supercomputer. Among these tools, there is the NVIDIA Modulus[7] which is a framework for building, training and fine-tuning PINN models. There is also the Horovod[8] framework, developed by Uber to improve training using GPUs in MPI parallelization, which enables distributed training using TensorFlow, Keras PyTorch, and Apache MXNet.

Acknowledgment

Authors thank LNCC (National Laboratory for Scientific Computing) for grant 205341 AMPEMI (call 2020-I), which allows access to the Santos Dumont supercomputer (node of the SINAPAD, the Brazilian HPC system). This study was financed in part by the Coordination for the Improvement of Higher Education Personnel (CAPES), Brazil, finance Code 001, and also by the CNPq Project 446053/2023-6. The authors also thank the Brazilian Ministry of Science, Technology and Innovation, and the Brazilian Space Agency.

References

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[2] C. Basdevant, M. Deville, P. Haldenwang, J. Lacroix, J. Ouazzani, R. Peyret, P. Orlandi, and A. Patera, "Spectral and finite difference solutions of Burgers equation," *Computers & Fluids*, vol. 14, pp. 23–41, Dec. 1986. [Online]. Available: http://www.researchgate.net/publication/222935980_Spectral_and_finite_difference_solutions_of_Burgers_equation

[3] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next," *J Sci Comput*, vol. 92, no. 3, p. 88, Sep. 2022. [Online]. Available: https://link.springer.com/10.1007/s10915-022-01939-z

[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*, vol. 378, pp. 686–707, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999118307125

[5] M. Vladimirova, J. Arbel, and P. Mesejo, "Bayesian neural networks become heavier-tailed with depth," in *NeurIPS 2018 - Thirty-second Conference on Neural Information Processing Systems*, Montréal, Canada, 2018, pp. 1–7. [Online]. Available: https://hal.science/hal-01950658

[6] S. Kim, I. Kim, J. Lee, and S. Lee, "Knowledge Integration into deep learning in dynamical systems: an overview and taxonomy," *Journal of Mechanical Science and Technology*, vol. 35, 2021. [Online]. Available: https://link.springer.com/article/10.1007/s12206-021-0342-5

[7] X. Meng, Z. Li, D. Zhang, and G. E. Karniadakis, "PPINN: Parareal Physics-Informed Neural Network for time-dependent PDEs," *Computer Methods in Applied Mechanics and Engineering*, vol. 370, p. 113250, 2020, arXiv:1909.10145 [physics, stat]. [Online]. Available: http://arxiv.org/abs/1909.10145

[8] Y. Yang and P. Perdikaris, "Adversarial Uncertainty Quantification in Physics-Informed Neural Networks," *Journal of Computational Physics*, vol. 394, pp. 136–152, 2019, arXiv:1811.04026 [physics, stat]. [Online]. Available: http://arxiv.org/abs/1811.04026

[9] A. Jagtap, E. Kharazmi, and G. 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*, vol. 365, p. 113028, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0045782520302127

[10] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, "Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data," *Journal of Computational Physics*, vol. 394, pp. 56–81, Oct. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999119303559

[11] L. Sun, H. Gao, S. Pan, and J.-X. Wang, "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, vol. 361, p. 112732, Apr. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S004578251930622X

[12] D. Liu and Y. Wang, "A Dual-Dimer Method for Training Physics-Constrained Neural Networks with Minimax Architecture," *Neural Networks*, vol. 136, pp. 112–125, Apr. 2021, arXiv:2005.00615 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2005.00615

[13] W. E and B. Yu, "The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems," Sep. 2017, arXiv:1710.00211 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1710.00211

[14] J. Sirignano and K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1339–1364, Dec. 2018. [Online]. Available: http://arxiv.org/abs/1708.07469

[15] E. Kharazmi, Z. Zhang, and G. E. Karniadakis, "hp-VPINNs: Variational Physics-Informed Neural Networks With Domain Decomposition," *Computer Methods in Applied Mechanics and Engineering*, vol. 374, p. 113547, Feb. 2021, arXiv:2003.05385 [cs, math]. [Online]. Available: http://arxiv.org/abs/2003.05385

[16] T. Nandi, O. Hennigh, M. Nabian, Y. Liu, M. Woo, T. Jordan, M. Shahnam, M. Syamlal, C. Guenther, and D. VanEssendelft, "Progress Towards Solving High Reynolds Number Reacting Flows in SimNet," National Energy Technology Laboratory (NETL), Pittsburgh, PA, Morgantown, WV ..., Tech. Rep., 2021. [Online]. Available: https://www.osti.gov/servlets/purl/1846970

[17] S. Markidis, "The Old and the New: Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers?" *Frontiers in Big Data*,

[7] https://developer.nvidia.com/modulus

[8] https://horovod.readthedocs.io

vol. 4, 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fdata.2021.669097

[18] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," 2018. [Online]. Available: http://arxiv.org/abs/1802.05799

[19] F. Chevallier, J. Morcrette, F. Chéruy, and N. A. Scott, "Use of a neural-network-based long-wave radiative-transfer scheme in the ECMWF atmospheric model," *Quart J Royal Meteoro Soc*, vol. 126, no. 563, pp. 761–776, Jan. 2000, {06}. [Online]. Available: https://rmets.onlinelibrary.wiley.com/doi/10.1002/qj.49712656318

[20] V. M. Krasnopolsky and M. S. Fox-Rabinovitz, "A new synergetic paradigm in environmental numerical modeling: Hybrid models combining deterministic and machine learning components," *Ecological Modelling*, vol. 191, no. 1, pp. 5–18, Jan. 2006, {04}. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304380005003455

[21] J. Burkardt, "Investigating Uncertain Parameters in the Burgers Equation," Mathematics Department, Ajou University, Suwon, Korea, 2013. [Online]. Available: https://people.sc.fsu.edu/~jburkardt/presentations/burgers_2013_ajou.pdf

[22] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *Journal of machine learning research*, vol. 18, no. 153, pp. 1–43, 2018. [Online]. Available: https://www.jmlr.org/papers/v18/17-468.html

[23] S. Xu, Z. Sun, R. Huang, G. Dilong, G. Yang, and S. Ju, "A practical approach to flow field reconstruction with sparse or incomplete data through physics informed neural network," *Acta Mechanica Sinica*, vol. 39, Nov. 2022. [Online]. Available: https://link.springer.com/article/10.1007/s10409-022-22302-x

[24] W. Koehrsen, "Overfitting vs. underfitting: A complete example," *Towards Data Science*, vol. 405, 2018. [Online]. Available: http://www.pstu.ac.bd/files/materials/1566949131.pdf

[25] L. Boninsegna, F. Nüske, and C. Clementi, "Sparse learning of stochastic dynamical equations," *The Journal of Chemical Physics*, vol. 148, no. 24, p. 241723, 2018. [Online]. Available: https://doi.org/10.1063/1.5018409

[26] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016. [Online]. Available: https://www.pnas.org/doi/10.1073/pnas.1517384113

[27] R. Tibshirani, "Regression Shrinkage and Selection via The Lasso: A Retrospective," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 73, no. 3, pp. 273–282, 2011. [Online]. Available: https://doi.org/10.1111/j.1467-9868.2011.00771.x

[28] B. M. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton, "PySINDy: A Python package for the Sparse Identification of Nonlinear Dynamics from Data," 2020. [Online]. Available: http://arxiv.org/abs/2004.08424

[29] A. A. Kaptanoglu, B. M. de Silva, U. Fasel, K. Kaheman, A. J. Goldschmidt, J. L. Callaham, C. B. Delahunt, Z. G. Nicolaou, K. Champion, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton, "PySINDy: A comprehensive Python package for robust sparse system identification," *Journal of Open Source Software*, vol. 7, no. 69, p. 3994, 2022. [Online]. Available: http://arxiv.org/abs/2111.08481

[30] S. Billings, "Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains," *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*, 2013.

[31] P. E. Black, "Greedy algorithm," 2005. [Online]. Available: https://xlinux.nist.gov/dads//HTML/greedyalgo.html

[32] K. Champion, P. Zheng, A. Y. Aravkin, S. L. Brunton, and J. N. Kutz, "A unified sparse optimization framework to learn parsimonious physics-informed models from data," 2019. [Online]. Available: https://arxiv.org/abs/1906.10612v2