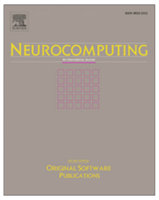
Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



Distributed learning machines for solving forward and inverse problems in partial differential equations

Vikas Dwivedi [a,*], Nishant Parashar [b], Balaji Srinivasan [a]

[a] *Heat Transfer and Thermal Power Laboratory, Department of Mechanical Engineering, Indian Institute of Technology Madras, Chennai 600036, India*
[b] *Computational Fluid Dynamics Laboratory, Department of Applied Mechanics, Indian Institute of Technology Delhi, Delhi 110016, India*

A R T I C L E I N F O

Article history:
Received 7 April 2020
Revised 8 July 2020
Accepted 1 September 2020
Available online 12 September 2020
Communicated by Q Wei

Keywords:
Distributed physics informed neural networks
Transfer learning
Inverse problems
Burgers' equation
Navier–Stokes equation

A B S T R A C T

We conceptualize Distributed Learning Machines (DLMs) – a novel machine learning approach that integrates existing machine learning algorithms with traditional mesh-based numerical methods for solving forward and inverse problems in nonlinear partial differential equations (PDEs). In conventional numerical methods such as finite element method (FEM), the discretization of the computational domain is a standard technique to reduce the representation load of basis functions. Along the same lines, we propose a distributed neural network architecture that facilitates the simultaneous deployment of several localized neural networks to solve PDEs in a unified manner. The most critical requirement of the DLMs is the synchronization of the distributed neural networks. For this, we introduce a new physics-based interface regularization term to the cost function of the existing learning machines like the Physics Informed Neural Network (PINN) and the Physics Informed Extreme Learning Machine (PIELM). To evaluate the efficacy of this approach, we develop three distinct variants of DLM namely, time-marching Distributed PIELM (DPIELM), Distributed PINN (DPINN) and time-marching DPINN. We show that ideas of linearization and time-marching allow DPIELM to be able to solve nonlinear PDEs to some extent. Next, we show that DPINNs have potential advantages over existing PINNs to solve the inverse problems in heterogeneous media. Finally, we propose a rapid, time-marching version of DPINN which leverages the ideas of transfer learning to accelerate the training. Collectively, this framework leads towards the promise of hybrid Neural Network-FVM or Neural Network-FEM schemes in the future.

1. Introduction

The application of deep learning techniques to solve problems in various fields of science and engineering has increased considerably [1–4]. It is because the predictive ability of the learning machines can be enhanced by incorporating domain-specific knowledge into the machine learning process. The domain-specific knowledge may be available in the form of physical principles, physical constraints, conservation laws, etc. Typically, the inclusion of this knowledge in the learning process accelerates the training, reduces the training data requirement, and improves the interpretability and defensibility of machine learning algorithms [5].

In this paper, we are interested in studying systems that are modeled by partial differential equations (PDEs). The derivation of a PDE is a product of the scientists- and researchers- domain expertise. Such knowledge should, therefore, be used to boost the predictive ability of deep neural networks. Lagaris et al. [6,7] did some preliminary developments in this direction. They used neural networks to solve initial and boundary value problems. But their approach did not gain popularity because of the restricted computing resources of the time.

The rapid expansion of computing resources and neural network techniques have recently rejuvenated the research area of using deep neural networks to solve PDEs. A key step in solving a PDE through deep learning is to constrain the neural network to minimize the PDE residual. It can be done in two major ways. First is to solve the PDE in its weak (or variational) form [8,9], and second is to solve the PDE in its strong form directly [10–13].

In the recent past, the pioneering work of Raissi et al. [13] has gained immense popularity. Raissi et al. [13] developed the novel Physics Informed Neural Networks (PINNs) to solve forward and inverse problems for PDEs. Their work has been followed up by various researchers which has led to various rapid developments in this field [14–18]. A python library named DEEP-XDE [19] has also been recently developed to solve a variety of PDEs [17,20] using PINNs.

* Corresponding author.
E-mail addresses: me15d080@smail.iitm.ac.in (V. Dwivedi), sbalaji@iitm.ac.in (B. Srinivasan).

https://doi.org/10.1016/j.neucom.2020.09.006
0925-2312/© 2020 Elsevier B.V. All rights reserved.

Although PINNs have been very successful in solving PDEs, they also have many limitations which are as follows:

1. *Speed.* PINNs use gradient descent optimization, which is prohibitively slow for practical problems [21].
2. *Optimization errors.* PINNs are prone to vanishing gradient problems arising due to the use of very deep networks [11].
3. *Hand-tuning.* The learning process of PINNs is hand-tuned [13]. For example, there is no way to determine how much data or which architecture is good enough for a given choice of sampling points.
4. *Inverse problem in heterogeneous media.* In the case of the inverse problem (See Section 2), the unknown parameters of the PDE are treated as weights of the deep neural network [13]. Since a unique PINN represents the entire computational domain, the PINN will also return unique parameter values for the whole domain. Consequently, PINN is not appropriate for solving PDEs- inverse problems in heterogeneous media (in which parameter value changes spatially).

In our ongoing endeavor to overcome the limitations of PINN, we have recently developed a Physics Informed Extreme Learning Machine (PIELM) and its distributed variant called the Distributed PIELM (DPIELM) [22,28]. However, PIELM and DPIELM are limited in applicability to linear PDEs only. As a result, both PIELM and DPIELM can not be used for solving nonlinear PDEs due to their inherent weakness in handling non-linearity. Since the inverse problem in linear PDE is also nonlinear, PIELM cannot solve inverse problems for both linear and nonlinear PDEs.

The mathematical formulation of PINN, PIELM and DPIELM is given in Appendices A, B, and C respectively. For full details of PINN, PIELM and DPIELM, readers may refer to Raissi et al. [13] and Dwivedi et al. [22] respectively.

1.1. Contributions of this paper

In this work, we modified the original DPIELM (refer Appendix C) for solving viscous Burgers- equation, which is a benchmark case of nonlinear PDE. Due to non-linearity, Burgers- equation admits solutions with a sharp discontinuity (shock), which is hard to resolve by classical numerical methods. Our modified DPIELM is successful in predicting the nonlinear solution except in the vicinity of the shock.

In the pursuit of modifying DPIELM for Burgers- equation, we developed some new distributed neural network architecture-based learning machines to solve both the forward (data-driven PDE solution) and the inverse (data-driven PDE discovery) problems in nonlinear PDEs. We have collectively labeled these learning machines as distributed learning machines (DLMs). DLM-s concept of 'divide and conquer' is quite general and can be used to enhance any of the current physics-informed learning machines, say PINN, PIELM, etc. For example, suppose a specific learning machine X is used to approximate a given PDE solution over a computational domain. Then, the methodology of DLM is to split the computational domain into an analogous multi-domains structure and mount a much simpler learning machine (of same type X) in each of these sub-domains.

The following are the main benefits of implementing this approach:

1. The representation burden of individual learning machines is reduced as the representation within each sub-domain is simplified. By providing adequate constraints to synchronize these machines, we can allow this army of simplified learning machines to solve nonlinear PDEs unitedly.
2. The distributed architecture of a DLM allows us to specify local physics-based constraints at the interface, which makes it ideal for heat transfer problems in heterogeneous media. An example of heterogeneous medium is a composite slab consisting of two different materials (with different *thermal* properties) stacked together. In Section 5.1.1, we will demonstrate through an example that a DLM can solve PDEs in heterogeneous media while a single learning machine, e.g., PINN, can not.
3. As we explain in Section 3 and 4.2, the time marching version of DLMs can easily incorporate ideas from *transfer learning* that leads to a faster and a physically more consistent algorithm for time-dependent nonlinear PDEs.
4. DLMs- distributed architecture has significant parallels to the conventional finite-element approach (FEM) to solve PDEs. Such similarity enables the natural importation of existing domain expertise from the traditional scientific computing community into the framework for machine learning.
5. Since the architecture of the individual learning machine deployed in a sub-domain is usually single-layered or two-layer deep at maximum, DLMs do not suffer from the issues associated with very deep neural networks (like the problem of vanishing gradients [11])

Despite the advantages mentioned above, there are additional constraints associated with DLMs arising from the synchronization of individual learning machines, which complicates the cost function. The hyperparameters associated with DLMs are large in number and need to be tuned. For example, how many domain subdivisions are good enough? Even though the individual machines- architecture is simple, the total number of parameters (including all distributed machines) is still comparable with the original deep learning machines. Therefore, is it correct to say that DLMs are more efficient than conventional learning machines? We deem this study related to hyperparameter tuning (and testing) as outside the scope of this work and would address it in a separate study altogether. This work aims to describe the mathematical principle of DLMs and establish that they can be used to solve the forward and inverse problems in nonlinear PDEs.

1.2. Organization of the paper

The Organization of this paper is as follows. Section 2 briefly describes the forward and inverse problems for PDEs. In Section 3, we present our experiments with a linearized time-marching DPIELM to solve Burgers- equation. Further, we propose two new distributed versions of PINN, which are an improvement over linearized DPIELM in Section 4. The first version is a direct import of the DPIELM idea in the PINN framework. We have named it DPINN or distributed PINN. The second version is a time-marching version of DPINN that takes advantage of transfer learning. In Section 5, we evaluate the performance of DPINN for both forward and inverse problems by testing it on two standard test cases, namely Burgers- equation, and Navier Stokes' equation. We also solve a problem involving heat transfer between heterogeneous media where DPINN outperforms existing PINN for both the forward and inverse problems. Finally, the conclusion and the scope of future work are discussed in Section 6.

2. Problem description

In this paper, we solve two sets of problems. The first type of problem that we solve is called the *forward problem*. The statement of the forward problem is as follows: given a PDE with pre-defined fixed model parameters, predict its solution. This problem requires no prior experiments and simulation data. PDE information and fixed model constants are the only input to the learning machine, and the output is the predicted solution of PDE. For example, consider linear advection equation

$$\frac{\partial u}{\partial t}+\lambda\frac{\partial u}{\partial x}=0, \tag{1}$$

the statement of the forward problem for this equation is: given value of λ, find the solution $u(x,t)$ of the system.

The second problem that we solve is called *inverse problem*. This problem is data-driven and therefore requires prior information either from experiments or from simulations. For example, the statement of the inverse problem for Eq. (1) is as follows: given solution of PDE (with unknown model parameter λ) at a few sampling points, find the value of λ. In other words, find the value of model parameters that best describe the available training data.

3. Motivation

The optimization free training routine of PIELM requires a given PDE to be recast into the form of a system of linear equations. Since only linear PDEs satisfy this requirement, PIELM is applicable for linear PDEs only [22]. The presence of nonlinear terms in a PDE leads to a system of nonlinear equations that cannot be directly solved using PIELMs. So, the next obvious question is: how to handle nonlinear terms? For example, consider the viscous Burgers' equation

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}=\nu\frac{\partial^2 u}{\partial x^2},\ \nu=0.01/\pi,\ x\in(-1,1) \tag{2}$$

$$u(x,0)=sin(\pi x) \tag{3}$$

$$u(-1,t)=u(1,t)=0 \tag{4}$$

The convective term $u\frac{\partial u}{\partial x}$ is the nonlinear term, and we want to approximate it in a way suitable to DPIELM. Why DPIELM, and not the usual PIELM? It is because we know that viscous Burgers' equation admits discontinuous (shock type) solutions that PIELM cannot handle.

Given the intrinsic linearity constraints of PIELM, the idea of local linearization naturally comes up in mind. It is a standard technique used in nonlinear dynamics [23] and fluid mechanics [24] to simplify nonlinear systems. The central idea is to approximate a nonlinear term by linearizing it about a reference state. To incorporate this domain-specific knowledge in our machine learning algorithm, we need to consider a few points. First, we have to choose a well-defined reference state. Next, given a reference state, the linear approximation will work only in the vicinity of that state. For example, if we consider the initial condition to be the reference state, can we use the original DPIELM to solve the linearized Burgers' equation? The answer is: No. It is because DPIELM splits the entire domain and measures all individual PIELM parameters in one go. Since the linearized version of Burgers' will be valid only in a small region, and not in the full domain, the original DPIELM can not be used in this case.

Keeping these ideas in mind, we formulated a time-marching version of DPIELM. A schematic diagram of the time-marching DPIELM is shown in Fig. 1. Unlike the original DPIELM, time-marking DPIELM does not calculate all the weights simultaneously, but the weights are gradually learned in small steps of time. To Fig. 1, it means that all the components of DPIELM don't learn simultaneously. In the first step, we take the given initial condition as the reference state and linearize the nonlinear term $u(x,t)\frac{\partial u(x,t)}{\partial x}$ in Burgers' equation in block-1 as $u(x,0)\frac{\partial u(x,t)}{\partial x}$. Next, we learn the weights of the distributed PIELMs. Once the weights are learnt, the value of u at the top edge of the block serves as the reference state for the next block, i.e., block-2. In the block-2, $u(x,t)\frac{\partial u(x,t)}{\partial x}$ is approximated by $u(x,\triangle t)\frac{\partial u(x,t)}{\partial x}$. This process is repeated until we cover the entire computational domain.

For the nonlinear viscous Burgers' equation, the result of the time-marching approach described so far is shown in Fig. 2. We have divided the spatial domain into 30 equal sub-domains with 15 sampling points in space and 20 points in time. Within each sub-domain, we installed a PIELM with 400 neurons. The size of the time step is 0.01. As we can see, it predicts the correct solution in the initial stages, demonstrating that our idea works in principle. However, the quality of the prediction deteriorates near the shock. The shock structure is not resolved cleanly, and there are two non-physical extrema near the shock.

We can keep on shrinking the size of time steps to see if time-marching DPIELM captures the correct shock structure. However, we found that this exercise is not fruitful because it becomes computationally cumbersome. Why is it so? There are two reasons. The first reason is apparent. With the increase in the number of blocks, the computational effort will also increase. The second reason is that the optimization free approach of PIELM and DPIELM restricts us from taking any benefit of *transfer learning*.

Transfer learning means storing the information gained while learning one problem and using it in a similar but separate problem [18]. Since we are solving the same PDE in all the time steps, it seems logical to make use of the weights learned in the previous iteration. However, in the case of PIELM, the weights of the input layer are randomly chosen, and weights of the outer layer are solved analytically. Thus, using the information of the weights of block-1 as an initial guess for block-2 is not valid for time-marching DPIELM.

What if our learning machine were using a gradient-descent type optimization routine? In that case, transfer learning can be easily incorporated. Since PINNs learn their weights using gradient-descent method, a time-marching distributed version of PINN would be an ideal choice to experiment with transfer learning ideas. Therefore, although we could not solve the Burgers' equation to a satisfactory level with time-marching DPIELM, the framework developed in this process can be used for other type of learning machines, e.g. PINN (see Section 5). In summary, some of the highlights of time-marching approach are as follows:

1. **Low computational load**. The number of parameters to found in one go for time-marching DLM are very less as compared to original DLM.
2. **Interpretability**. Intuitive definition of reference states improves the physical interpretability of the algorithm.
3. **Transfer learning**. Although not applicable for DPIELM, this framework is ideal for exploiting transfer learning for gradient descent-based learning machines.

4. Proposed DLMs

In this section, we firstly describe the distributed version of PINN, i.e., DPINN, for the forward problems. Next, we describe time-marching DPINN for the forward problem. In the end, we describe the algorithm for inverse problems. The algorithm for inverse problems is derived by a slight modification of the cost function of the forward problem.

4.1. Distributed physics informed neural network (DPINN)

Consider the following 1D unsteady problem

$$\frac{\partial}{\partial t}u(x,t)+\mathcal{N}u(x,t)=R(x,t),(x,t)\in\Omega \tag{5}$$

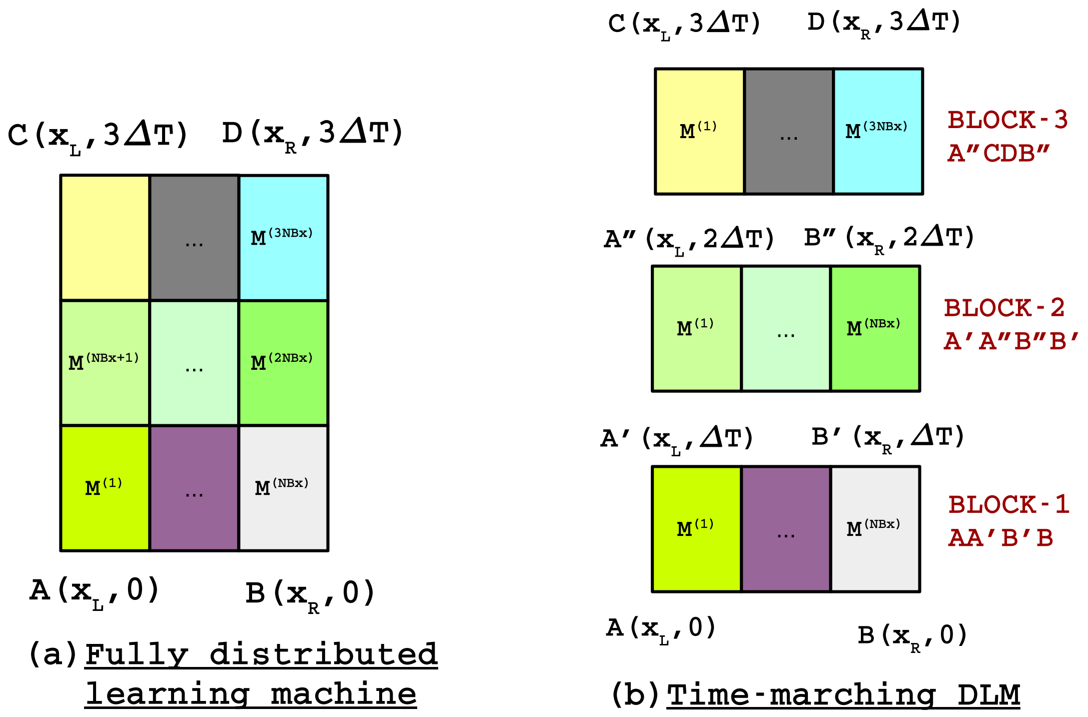


Fig. 1. Schematic diagram of a time-marching distributed learning machine.

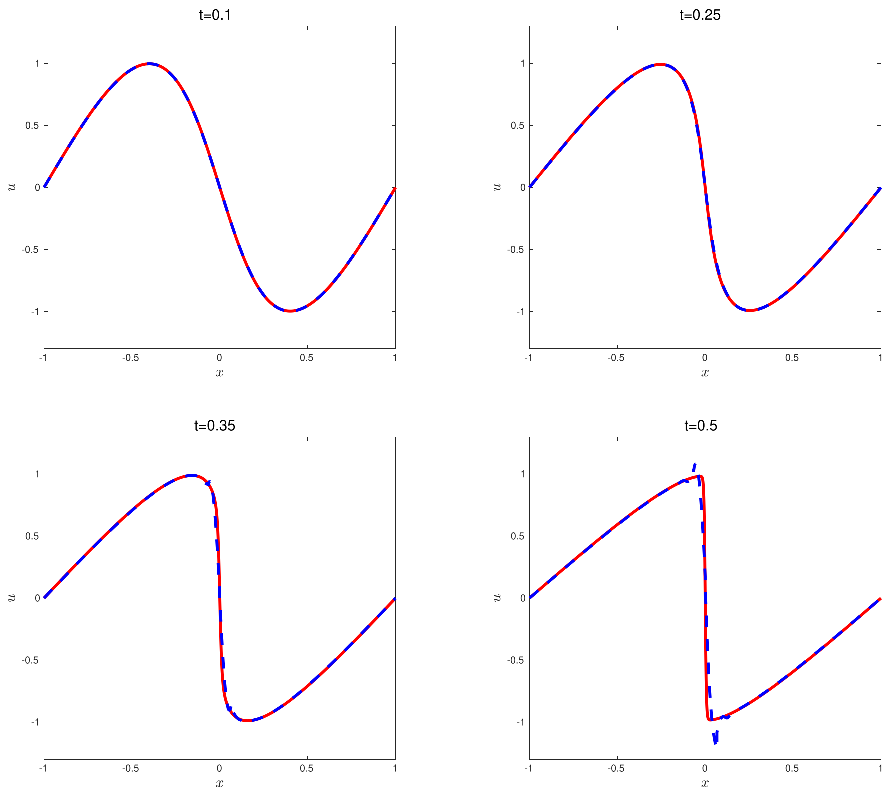


Fig. 2. Comparison of DPIELM prediction with exact solution for Burgers' equation. Solid line represents reference solution and dashed line represents predicted solution.

$$u(x,t) = B(x,t), (x,t) \in \partial\Omega, \quad (6)$$

$$u(x,0) = F(x), x \in (x_L, x_R), \quad (7)$$

where $\mathcal{N}$ is a potentially nonlinear differential operator and $\partial\Omega (= \partial\Omega_L \bigcup \partial\Omega_R)$ is the boundary of computational domain Ω as shown in Fig. 3.

In our problem, the rectangular domain Ω is given by $\Omega = (x_L, x_R) \times (0, T)$. On uniformly dividing Ω into N_c non-overlapping rectangular cells, Ω may be rewritten as

$$\Omega = \bigcup_{i=1}^{N_c} \Omega_i. \tag{8}$$

The boundary of the cell Ω_i is denoted by $\delta\Omega_i$. For rectangular cells,

$$\delta\Omega_i = \bigcup_{m=1}^{4} I_m^{(i)} \tag{9}$$

where $I_m^{(i)}$ represents the mth interface of Ω_i.

Fig. 4 shows the DPINN in a rectangular computational domain with $NB_x \times NB_t$ cells (i.e. $N_c = NB_x \times NB_t$). We denote the individual PINN on the ith cell by $M^{(i)}$. Fig. 4 also shows an individual PINN element with collocation points $\left(\vec{x}_f, \vec{y}_f\right)^{(i)}$ at the interior (red in color) and the boundary points $\left(\vec{x}_{bc}, \vec{y}_{bc}\right)^{(i)}$ at the four faces (blue in color). Each individual PINN has its own unique representation i.e., weights and biases. The output corresponding to a given $M^{(i)}$ is denoted by $f^{(i)}$.

The distributed network architecture simplifies the representation load of an individual PINN, and therefore, we put just two layers of deep neural networks at each sub-domain. However, it also gives us an additional task to synchronize all these individual PINNs in a physically consistent fashion. To connect all these PINNs, we introduce additional interface regularization terms to the original cost function. Depending on the differential operator $\mathcal{N}$, the regularizer may be entirely convective (continuous solution) or diffusive (continuously differentiable solution) or a summation of both. For example, convective regularizer term is sufficient for a linear advection equation. However, for the advection–diffusion equation, both convective and diffusive regularizer terms are required. The inclusion of these terms further constrains the algorithm to ensure consistency of fluxes across PINN interfaces over the whole domain.

For the computational domain shown in Fig. 4, the total loss function is a summation of PINN-based losses (J_{PINN}) and interface regularizer (J_{int}) i.e.,

$$J = \underbrace{J_{PDE}}_{\text{PINN Loss}} + \underbrace{J_{int}}_{\text{Regularizer}} . \tag{10}$$

Expressions for the PINN losses. The expression for the PINN loss for $M^{(i)}$ is given by

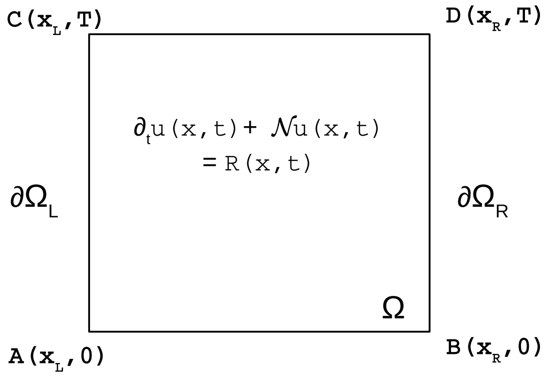


Fig. 3. A diagram of the problem: PDE in a rectangular domain.

$$J_{PDE} = \sum_{i=1}^{NB_x NB_t} \left(\frac{\vec{\xi}_{PDE}^{(i)^T} \vec{\xi}_{PDE}^{(i)}}{2N_f^{(i)}} \right) \tag{11}$$

where

$$\vec{\xi}_{PDE}^{(i)} = \frac{\partial \vec{f}^{(i)}}{\partial t} + \mathcal{N}\vec{f}^{(i)} - \vec{R}^{(i)}, \; on \; \left(\vec{x}_f, \vec{y}_f\right)^{(i)} \tag{12}$$

and $N_f^{(i)}$ represents number of collocation points in the ith cell.

Expressions for the additional interface losses. In general, we consider J_{int} as sum of continuity and differentiability losses. As a particular example, if we select advection–diffusion equation $\left(\mathcal{N} = \frac{\partial}{\partial x} - \nu \frac{\partial^2}{\partial x^2}\right)$, then flux regularization term is given by

$$J_{int} = J_{C_x^0} + J_{C_t^0} + J_{C_x^1} \tag{13}$$

where $J_{C_x^0}, J_{C_t^0}$ refer to convective flux and continuity losses along $x = constant$ and $t = constant$ interfaces and $J_{C_x^1}$ refers to diffusive flux loss along $x = constant$ interfaces. Referring to Fig. 4, the expressions for these losses are as follows:

$$J_{C_x^0} = \sum_{i=1}^{NB_x - 1} \left(\frac{\vec{\xi}_{C_x^0}^{(i)^T} \vec{\xi}_{C_x^0}^{(i)}}{2N_x^{(i)}} \right), \; on \; \left(\vec{x}_{bc}, \vec{y}_{bc}\right)^{(i)}, \tag{14}$$

$$J_{C_t^0} = \sum_{j=1}^{NB_t - 1} \left(\frac{\vec{\xi}_{C_t^0}^{(j)^T} \vec{\xi}_{C_t^0}^{(j)}}{2N_t^{(j)}} \right), \; on \; \left(\vec{x}_{bc}, \vec{y}_{bc}\right)^{(j)}, \tag{15}$$

$$J_{C_x^1} = \sum_{i=1}^{NB_x - 1} \left(\frac{\vec{\xi}_{C_x^1}^{(i)^T} \vec{\xi}_{C_x^1}^{(i)}}{2N_x^{(i)}} \right), \; on \; \left(\vec{x}_{bc}, \vec{y}_{bc}\right)^{(i)}, \tag{16}$$

where

$$\vec{\xi}_{C_x^0}^{(i)} = \left\{ \begin{array}{c} \vec{f}^{\left(\kappa_x^0(1)+i-1\right)} \\ \vec{f}^{\left(\kappa_x^0(2)+i-1\right)} \\ \cdots \\ \vec{f}^{\left(\kappa_x^0(NB_t)+i-1\right)} \end{array} \right\}_{I_2} - \left\{ \begin{array}{c} \vec{f}^{\left(\kappa_x^0(1)+i\right)} \\ \vec{f}^{\left(\kappa_x^0(2)+i\right)} \\ \cdots \\ \vec{f}^{\left(\kappa_x^0(NB_t)+i\right)} \end{array} \right\}_{I_4},$$

$$\vec{\xi}_{C_t^0}^{(j)} = \left\{ \begin{array}{c} \vec{f}^{\left(\kappa_t^0(1)+(j-1)NB_x\right)} \\ \vec{f}^{\left(\kappa_t^0(2)+(j-1)NB_x\right)} \\ \cdots \\ \vec{f}^{\left(\kappa_t^0(NB_x)+(j-1)NB_x\right)} \end{array} \right\}_{I_3} - \left\{ \begin{array}{c} \vec{f}^{\left(\kappa_t^0(1)+jNB_x\right)} \\ \vec{f}^{\left(\kappa_t^0(2)+jNB_x\right)} \\ \cdots \\ \vec{f}^{\left(\kappa_t^0(NB_x)+jNB_x\right)} \end{array} \right\}_{I_1},$$

$$\vec{\xi}_{C_x^1}^{(i)} = \frac{\partial}{\partial x} \left\{ \begin{array}{c} \vec{f}^{\left(\kappa_x^0(1)+i-1\right)} \\ \vec{f}^{\left(\kappa_x^0(2)+i-1\right)} \\ \cdots \\ \vec{f}^{\left(\kappa_x^0(NB_t)+i-1\right)} \end{array} \right\}_{I_2} - \frac{\partial}{\partial x} \left\{ \begin{array}{c} \vec{f}^{\left(\kappa_x^0(1)+i\right)} \\ \vec{f}^{\left(\kappa_x^0(2)+i\right)} \\ \cdots \\ \vec{f}^{\left(\kappa_x^0(NB_t)+i\right)} \end{array} \right\}_{I_4},$$

$$\kappa_x^0 = [1, (1 + NB_x), \ldots, 1 + (NB_t - 1)NB_x]^T$$

and

$$\kappa_t^0 = [1, 2, \ldots, NB_x]^T.$$

These expressions are used to calculate the total loss J. Finally, we may use any gradient-based optimization routine to minimize this loss.

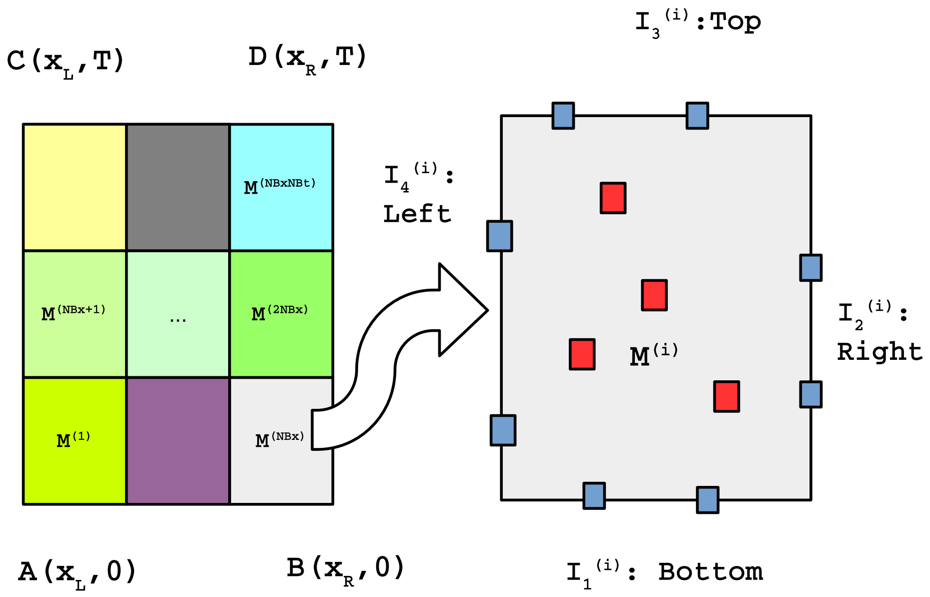


Fig. 4. Left hand side of the figure depicts the uniformly distributed components of DPINN in Ω. Right hand side of the figure shows the exploded view of an individual PINN element ($i = NB_x$). The blue and the red squares denote the boundary and the collocation points respectively.

This completes the mathematical formulation of DPINN for a 1D unsteady problem. We have chosen to describe only 1D unsteady problems because it is easy to understand the implementation of distributed architecture in simple settings. No additional tricks are required in extending this approach to higher dimensions.

A simplified example For the sake of clarity, we explain the learning procedure with the help of a simple 2×2 architecture by setting $NB_x = NB_t = 2$ in Fig. 4. Fig. 5a shows the PINN-based losses. It can be seen that loss due to initial condition residual (J_{IC}) is not applicable for PINN No. 3 and 4.

This configuration has four interfaces which we have indexed as $12, 13, 24$ and 34 in Fig. 5b. Considering $\mathcal{N} = \frac{\partial}{\partial x} - \nu \frac{\partial^2}{\partial x^2}$, we define the $J_{int}^{(p)}$ for a particular interface as follows

$$J_{int}^{(p)} = \alpha J_{C_t^0}^{(p)} + \beta J_{C_x^0}^{(p)} + \gamma J_{C_x^1}^{(p)}. \tag{17}$$

Depending on the position of interface, the values of α, β and γ are tabulated in the Table 1.

The total DPINN loss is summation of the original PINN and the additional interface losses for the whole domain. Fig. 6 depicts the total DPINN loss.

4.2. Time-marching DPINN

The main ingredients of time-marching DPINN are already discussed in Sections 3 and 4.1. Referring to Fig. 1, we first solve the given PDE on a small computational domain $(x_L, x_R) \times (0, \triangle t)$ without doing any linearization. In Fig. 1, the area $(x_L, x_R) \times (0, \triangle t)$ is denoted by block-1. Similarly block-2 refers to $(x_L, x_R) \times (\triangle t, 2\triangle t)$. Unlike DPIELM, we use a gradient-based optimization method to minimize the DPINN (with NB_t=1) loss that we described in the previous subsection. Once the loss function converges to a satisfactory level, we save the prediction of $u(x, \triangle t)$. This prediction is used as an initial condition for the next step calculation in block-2. Unlike time-marching DPIELM, the time-marching DPINN also exploits the advantage of transfer learning. Since we solve the same PDE in all the blocks, the weights learned in a given block are used as initialization in the subsequent block. This transfer of learned weights makes the learning of the subsequent block easier. This time-marching continues until we cover the entire computational domain.

4.3. DPINN for inverse problems

A slight modification in the cost function of the forward problem is needed to make it work for inverse problems as well. For example, reconsider the inverse problem of Eq. (1) that we described in Section 2. Similar to the forward problem, we approximate the solution of the PDE with a DPINN. However, since the value of the model parameter is unknown, we consider λ itself as a weight to be learned. To find the values of network weights, we need to minimize the combined loss function that takes into account the information from available training data as well as information of PDE. Mathematically, the loss function to be minimized is

$$J_{inv} = J_{data} + J_{PDE} + J_{int} \tag{18}$$

where

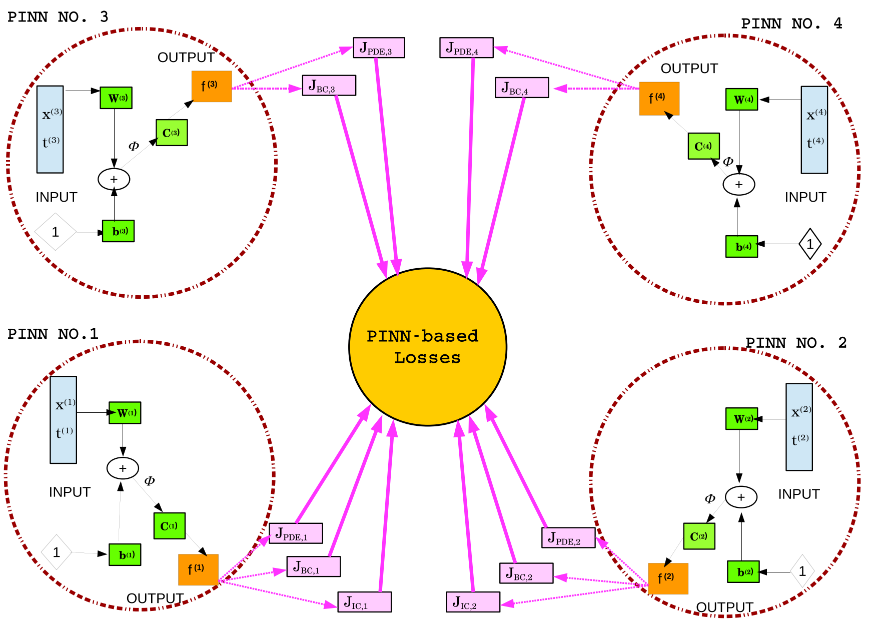
$$J_{data} = mean\left(\sum_{i=1}^{NB_x \times NB_t} \sum_{j=1}^{M(j)} \left(u_{data}\left(x_j^{(i)}, t_j^{(i)}\right) - u_{net}\left(x_j^{(i)}, t_j^{(i)}\right)\right)^2\right), \tag{19}$$

$$J_{PDE} = mean\left(\sum_{i=1}^{NB_x \times NB_t} \sum_{j=1}^{M(j)} \left(\frac{\partial u_{net}}{\partial t} + \lambda \frac{\partial u_{net}}{\partial x}\right)^2_{x_j^{(i)}, t_j^{(i)}}\right), \tag{20}$$

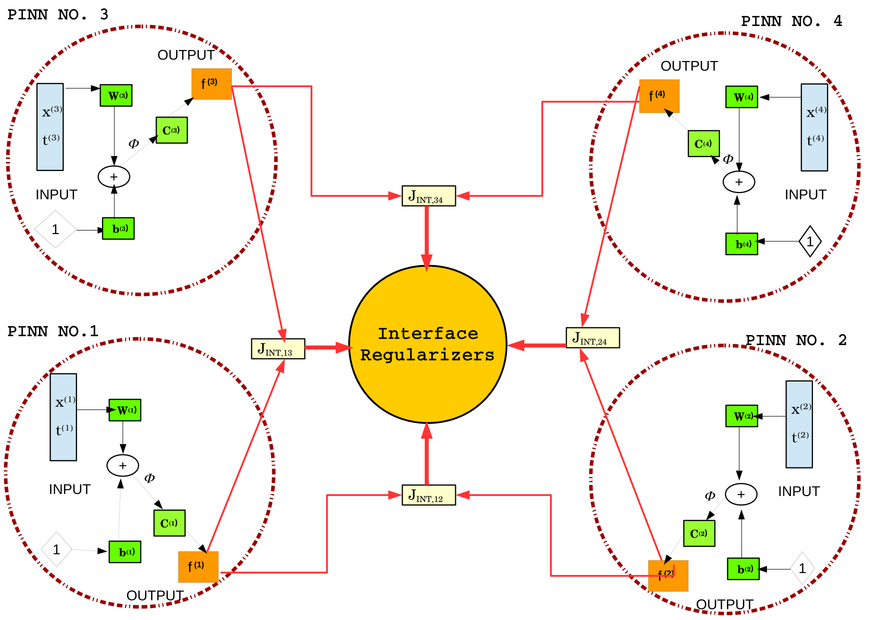
$M(j)$ being the number of data points in the i^{th} box, and u_{net} is neural network prediction and J_{int} is the interface loss (as explained in Section 4.1) due to all the data points that lie on the interface.

4.4. Comparison among the three DLMs

So far, we have discussed the formulation, advantages and limitations of various DLMs. We briefly compare and contrast among them with the help of Table 2.



(a) PINN-based losses.



(b) Interface regularizer.

Fig. 5. Description of PINN-based loss function and the interface regularizer for a 2 × 2 DPINN architecture.

Table 1
Flux coefficients for 2 × 2 DPINN.

	$p = 1$ (or 12)	$p = 2$ (or 13)	$p = 3$ (or 24)	$p = 4$ (or 34)
α	0	1	1	0
β	1	0	0	1
γ	1	0	0	1

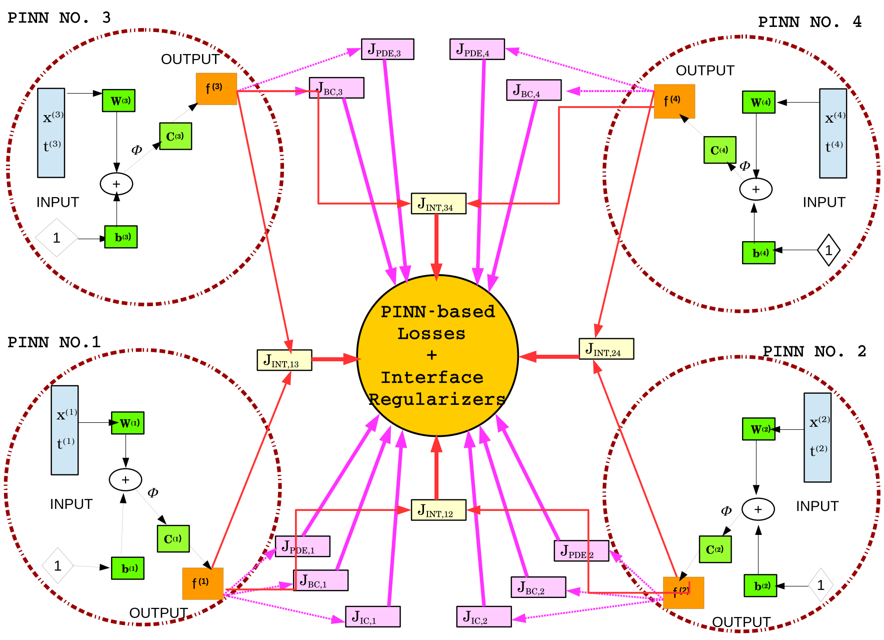


Fig. 6. Description of combined loss function for a 2 × 2 DPINN architecture.

5. Numerical experiments

In this section, we show a detailed analysis of DPINN for solving both the forward and the inverse problems in PDE. We first solve the details of the experiments performed to solve the forward problem in Section 5.1, followed by the inverse problems in Section 5.2.

5.1. Forward problems

We first show the details of the experiments performed for solving the forward problem.

5.1.1. Solution of steady heat conduction equation in heterogeneous media

The majority of the materials used in various thermal engineering applications are heterogeneous. Consider the flow of hot fluid inside a tube covered with a layer of thermal insulation. The "tube + insulation" system constitutes a heterogeneous media due to the difference in thermal conductivity of tube and insulation. Similarly, we find applications of composite slabs (with different thermal properties) in furnaces with masonry structures, etc.

For example, we consider a composite slab consisting of two different materials stacked together, as shown in Fig. 7. Assuming 1D heat conduction, the temperature distribution within a composite slab during steady-state operation is governed by

$$\frac{\partial}{\partial x}\left(K(x)\frac{\partial T(x)}{\partial x}\right) = 0, \tag{21}$$

where $T(x)$ is the temperature distribution and $K(x)$ is the spatial distribution of thermal conductivity. The values of thermal conductivity of the slabs at the left and right side are K_L and $K_R(< K_L)$ respectively. Thus, $K(x)$ is given by

$$K(x) = \frac{K_L + K_R}{2} - \frac{K_L - K_R}{2} sgn(x - 0.5), \tag{22}$$

where sgn(.) is signum function.

For this example, the statement of the forward problem is as follows. Given the PDE, i.e., Eq. (21) with known model parameters $(K_L, K_R) = (1, 0.5)$ and boundary conditions $T(x = 0) = 1, T(x = 1) = 0$, find the temperature distribution $T(x)$.

We solved this problem by using PINN as well as DPINN. We found that the original PINN is not sensitive towards the discontinuity in the value of K at $x = 0.5$. Therefore, it finds it difficult to represent two distinct solutions in two halves of the computational domain.

On the other hand, DPINN consisting of two uniformly distributed PINNs can easily solve this problem. By default, each PINN has a unique representation. The individual PINNs of the DPINN are synchronized by providing additional constraints depending on the nature of the problem. These additional penalties sensitize the training algorithm to take care of sudden changes in K at the interface.

In this case, if T_L and T_R represent the individual PINN solutions in the left and the right side respectively, the additional constraints are as follows.

$$(T_L)_{x=0.5} = (T_R)_{x=0.5} \tag{23}$$

$$-K_L\left(\frac{\partial T_L}{\partial x}\right)_{x=0.5} = -K_R\left(\frac{\partial T_R}{\partial x}\right)_{x=0.5} \tag{24}$$

The exact solution of this problem is given by

$$T_{exact} = \begin{cases} 1 - \frac{2}{3}x & x \in (0.0.5) \\ \frac{4}{3}(1 - x) & x \in (0.5, 1) \end{cases} \tag{25}$$

The prediction of $T(x)$ by PINN and DPINN is given in Fig. 8. The details of the numerical experiment are as follows. A total of 41 equally spaced collocation points with $NB_x = 2$ were used to train the DPINN. The relative L_2 norm error of the predicted solution is 6.2e−5.

5.1.2. Solution of Laplace' equation in two dimensions

The Laplace' equation in two-dimension can be expressed as:

$$\frac{\partial^2 T(x,y)}{\partial^2 x} + \frac{\partial^2 T(x,y)}{\partial^2 y} = 0. \tag{26}$$

Table 2
Comparison among various DLMs discussed so far.

	Linearized time-marching DPIELM	DPINN	Time-marching DPINN
Basis functions	Single hidden layer neural network	Deep neural network	Deep neural network
Unknowns	Weights and Biases of the outer layer only.	Weights and biases of all the layers.	Weights and biases of all the layers.
Physics awareness	Information of physics is embedded in the • cost function due to *linearized* PDE. • interface regularization term along space only.	Information of physics is embedded in the • cost function due to the original PDE. • interface regularization term along space as well as time.	Information of physics is embedded in the • cost function due to the original PDE. • interface regularization term along space only.
Solution/ Training methods	The solution marches in time. At each time step, an optimization-free PIELM algorithm [1] is used for a domain with multiple divisions along spatial axis but only one division along time axis (with a very small length due to*linearization*).	No time marching is involved. The solution is predicted for the whole of discretized computational domain using any gradient-based optimization algorithm in one shot.	The solution marches in time. At each time step, any gradient-descent-based algorithm is used for a domain with multiple divisions along spatial axis but only one division along time axis. The weights are initialized with the converged weights of previous iteration (*Transfer learning*).
Performance	Inaccurate and time-taking for nonlinear PDEs.	Accurate for linear as well as nonlinear PDEs. Faster than linearized time-marching DPIELM.	Accurate and fast (as compared to DPINN) for linear as well as nonlinear PDEs.
Forward problems	Applicable	Applicable	Applicable
Inverse problems	Not applicable	Applicable	Applicable

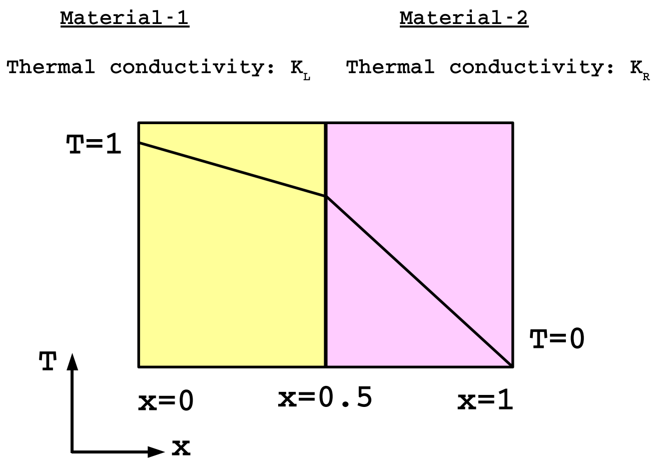


Fig. 7. Schematic diagram of steady heat conduction in a composite slab.

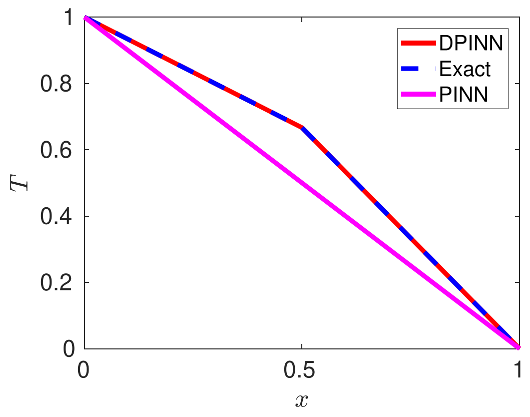


Fig. 8. Solution to steady conduction equation with spatially varying conductivity.

We chose a certain set of boundary conditions for which the analytical solution to the Laplace equation is known:

$$\begin{aligned} T(x,y=1) &= 10sin(\pi x); \\ T(x,y=0) &= 0, \\ T(x=0,y) &= 0, \\ T(x=1,y=0) &= 0. \end{aligned} \tag{27}$$

The analytical solution of Eq. (26) for the prescribed boundary conditions (Eq. 27) is:

$$T(x,y) = 10sin(\pi y)\frac{sinh(\pi x)}{sinh\pi}.$$

We employ DPINN with $NB_x = NB_y = 10, nb_x = nb_y = 11$ and use a two layered neural network with five neurons per layer for each cell. After training DPINN, we obtain the solution on a fine (501×501) grid and evaluate its performance with the analytical solution (Eq. 28). In Fig. 9, we show the contours of DPINN solution along with the contour curves of analytical solution for comparison. The results are in excellent agreement with the exact solution. The relative L-2 norm error of the predicted by DPINN is 3.499e−5.

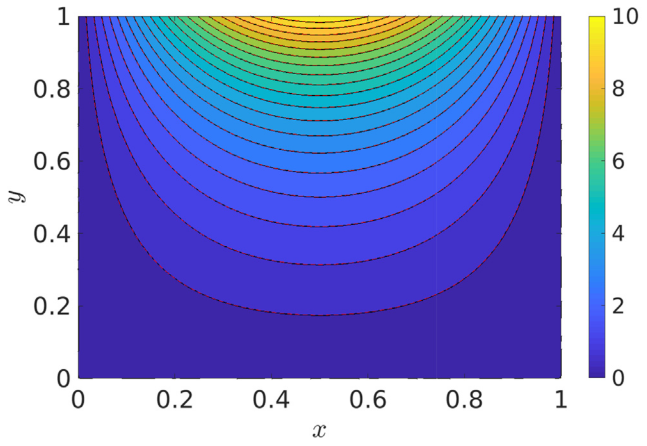


Fig. 9. Contours of solution of two-dimensional Laplace equation. Solid lines represent the contours of analytical solution. The relative L2-norm error between DPINN solution and the analytical solution is 3.499e−5.

5.1.3. Solution of Burgers' equation with DPINN

Now, we consider the viscous Burgers' equation (refer Eq. 2). Viscous Burgers' equation is a fundamental nonlinear PDE that appears in fluid mechanics, gas dynamics, traffic flow modeling, etc. We solve this equation using DPINN and compare our results against the numerical reference solution.

The optimization routine of DPINN is intuitively understood with the help of Fig. 10. It describes the output of DPINN for Burgers' equation after one iteration. The distinct rectangular regions depict the output of individual PINNs employed on different cells of the computational domain. These components of DPINNs communicate among themselves via flux-based interface regularization terms that we described earlier. After convergence, the predicted solution accurately captures the continuous solution in the initial stages as well as the discontinuous solution after the shock formation, as shown in Fig. 11.

Readers will appreciate that DPINN also improves the physical interpretability of the training algorithm. For example, minimization of J_{int} naturally tries to embed flux conservation in the whole domain. Therefore, the inclusion of J_{int} with J_{PDE} serves as a more physically consistent error metric than the typical J_{PDE} loss alone. Another essential point to be noted is that DPINN extracts more information content from the training data. For example, if we don't use a distributed architecture, each training point (collocation or boundary point) contributes to the learning of weights only once (minimizing J_{PDE} or J_{BC}). In a DPINN, however, many of these training points will fall on the interface of the sub-domains. Therefore, in addition to minimizing J_{PDE}, these points will be reused to minimize J_{int} as well.

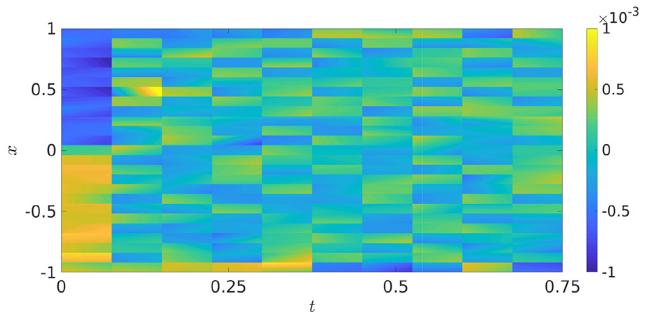
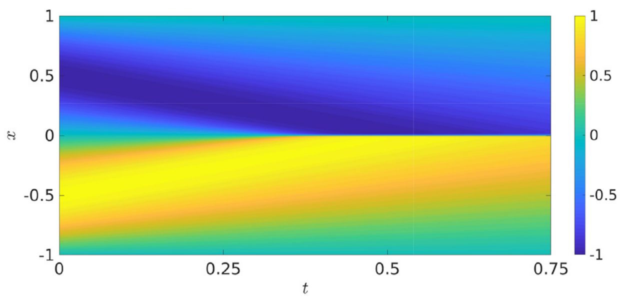
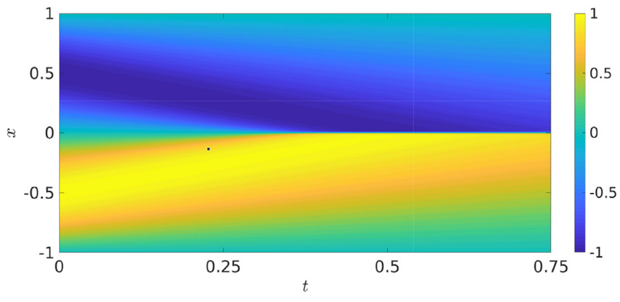


Fig. 10. Output of DPINN for Burgers equation after one iteration. The distinct rectangular regions depict the output of various minial sized neural networks employed on different cells of the computational domain.



(a) DPINN prediction.



(b) Reference solution.

Fig. 11. Comparison of the solution obtained using DPINN with numerically accurate solution for Burgers equation.

The reference solution for estimating error has been obtained by solving Burgers' equation using Roe scheme [25]. A total of 251×211 equally spaced collocation points with $NB_x = 25$, and $NB_t = 10$ was used to train DPINN for solving Burgers' equation. The comparison between DPINN and the exact (ground truth) solution for Burgers' equation at four different time instants is shown in Fig. 12(a–d). DPINN accurately predicts the formation of shock, and its dissipation. The L2 norm error of the predicted solution on a testing dataset of an equally spaced space–time grid of (256, 76) is 3.3e−4. The training history of the DPINN is shown in Fig. 13. The training loss goes down to ~1e−7 in 54,652 iterations. Although the accuracy of the solution obtained by DPINN is quite reasonable, the computational time required for training is high. As we explained earlier (Section 1), there is a scope of speeding up the training algorithm using transfer learning.

5.1.4. Solution of Burgers' equation with time marching DPINN

Reconsider the viscous Burgers' equation (refer Eq. 2). To solve this equation, we created the time-marching version of DPINN by dividing the computational domain into 30 time steps, i.e.,

$$\begin{aligned}\underbrace{(-1,1)\times(0,0.75)}_{comp.domain} = & \underbrace{(-1,1)\times(0,0.025)}_{block-1} \\ & \cup \underbrace{(-1,1)\times(0.025,0.05)}_{block-2} \\ & \cup \ldots \underbrace{(-1,1)\times(0.725,0.75)}_{block-30} \end{aligned} \tag{29}$$

The spatial domain is divided into 25 parts and total number of training points allotted to each PINN is 11×8, i.e., 11 along x-axis, and 8 along t-axis.

The number of cells in x-direction is $NB_x = 25$. The network is trained on a total of 251×211 uniformly spaced collocation points. We show the prediction obtained using DPINN with time-marching against reference solution at four different time instants in Fig. 14(a–d). It can be observed that the solution obtained from DPINN with time-marching is in good agreement with the reference solution. The L2 norm error of the predicted solution on a testing dataset of an equally spaced space–time grid of (256, 76) is 3.9e−4. Fig. 15a shows the number of iterations required in each time step for the loss to converge to ~1e−7. Fig. 15b shows the plot of training loss history for four particular time steps. Based on these results, we observe the following:

1. The number of iterations to converge in the first time step in the DPINN with time-marching and the usual DPINN are of the same order of magnitude. However, the total computation time in each iteration of time-marching DPINN is much lower since the number of parameters in the time-marching DPINN is much smaller than the standard DPINN.
2. Even though we are solving the same PDE in every time step, the required number of iterations for convergence doesn't always go down in all the successive steps. It is because the solution is smooth and continuous only in the initial stages, but its gradient keeps on increasing until the shock formation. As a result, training becomes faster for the first few time steps (initial stages) and then relatively slows for the time steps near shock formation (from t = 0.3 to t = 0.5). Once the shock is formed at t = 0.5, the shape of the shock doesn't change, and it only dissipates without any translation. Therefore after t = 0.5, the time-marching DPINN learns the weights of successive time steps in just a few hundred iterations.

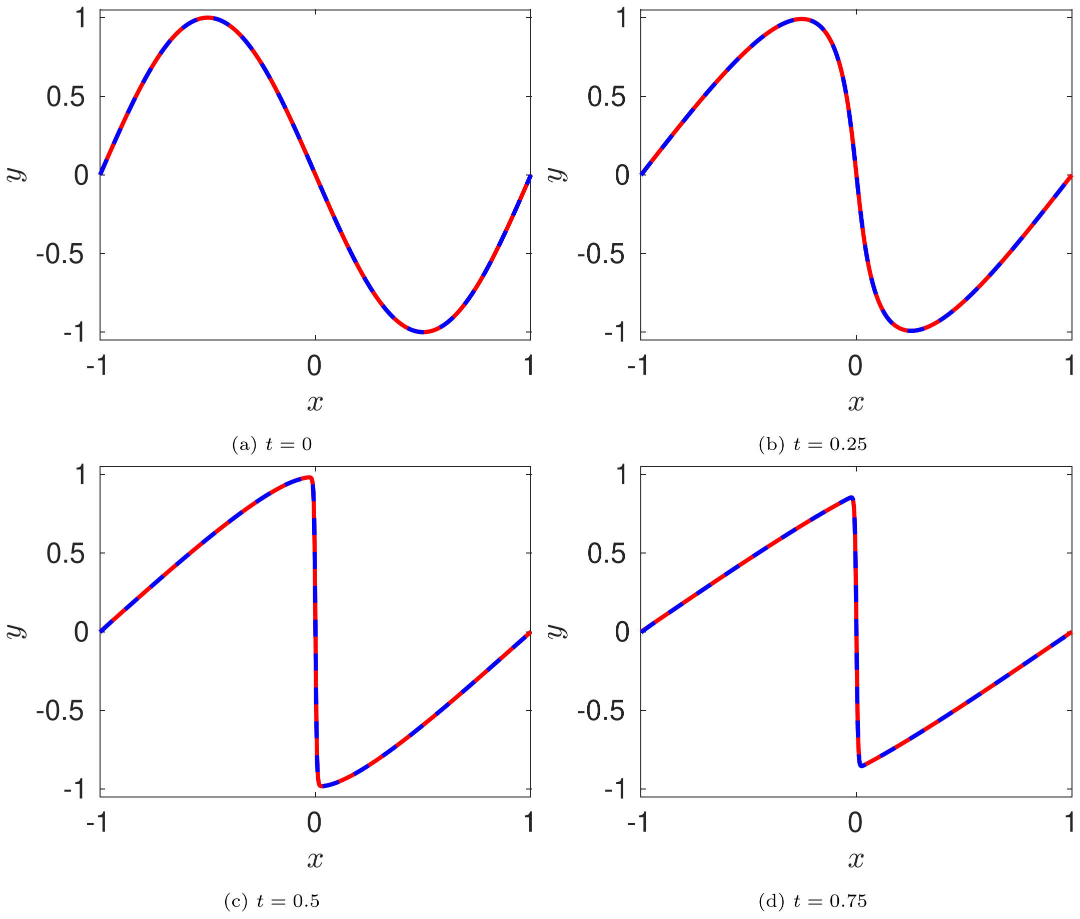


Fig. 12. Solution to the Burgers' equation obtained using DPINN (Dashed line). Solid line is the reference solution.

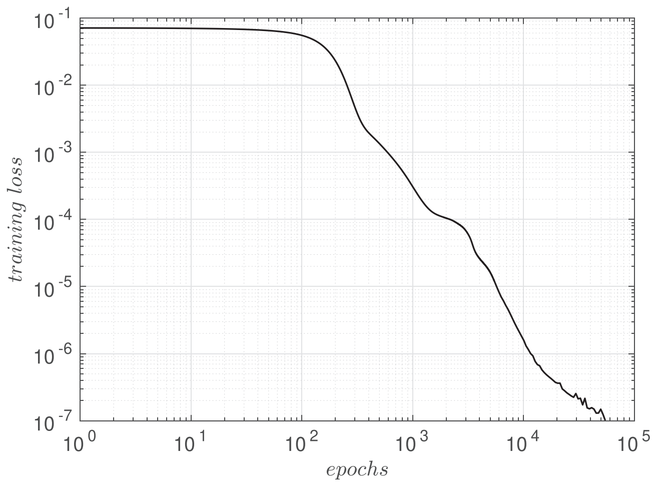


Fig. 13. Training loss history for DPINN. Total 54,652 epochs were required for convergence (loss ∼1e−7) and the time required for training was ∼ 91 minutes using ADAM optimizer with 5e−4 learning rate.

3. For the same amount of training data and number of parameters, each iteration of DPINN is ten times expensive as compared to DPINN with time-marching. The total time taken for convergence for the same amount of data points is three times for DPINN compared to DPINN with time-marching.

5.1.5. Solution of steady Navier–Stokes' equation at low Reynolds number

In this section, we solve the lid-driven cavity problem. The lid-driven cavity is a well-known benchmark problem for viscous incompressible fluid flow. It consists of a square cavity filled with fluid. At the top boundary, tangential velocity is applied to drive the fluid flow in the cavity. No-slip condition is applied to the remaining three walls. The computational domain and boundary conditions are shown in Fig. 16. The Navier–Stokes equation is the statement of mass and momentum conservation, and they govern the flow of viscous fluids. In this case, we solve the steady-state version of the planar Navier–Stokes equation, which is as follows.

$$u_x + v_y = 0 \tag{30}$$

$$uu_x + vu_y = -\frac{1}{\rho}p_x + \nu(u_{xx} + u_{yy}) \tag{31}$$

$$uv_x + vv_y = -\frac{1}{\rho}p_y + \nu(v_{xx} + v_{yy}) \tag{32}$$

where u, v denote the velocity components along x and y directions and p, ρ and ν denote the pressure, density and kinematic viscosity respectively. Eq. (30) is the mass conservation equation and the Eqs. (31) and (32) are momentum conservation equations in the horizontal and vertical directions respectively. The left-hand side of the momentum equations consists of inertia terms, and the right-hand side consists of pressure term and the viscous term. The relative strength of the inertia and viscous terms dictates the nature of the flow. The pressure term is primarily an adjustment term that

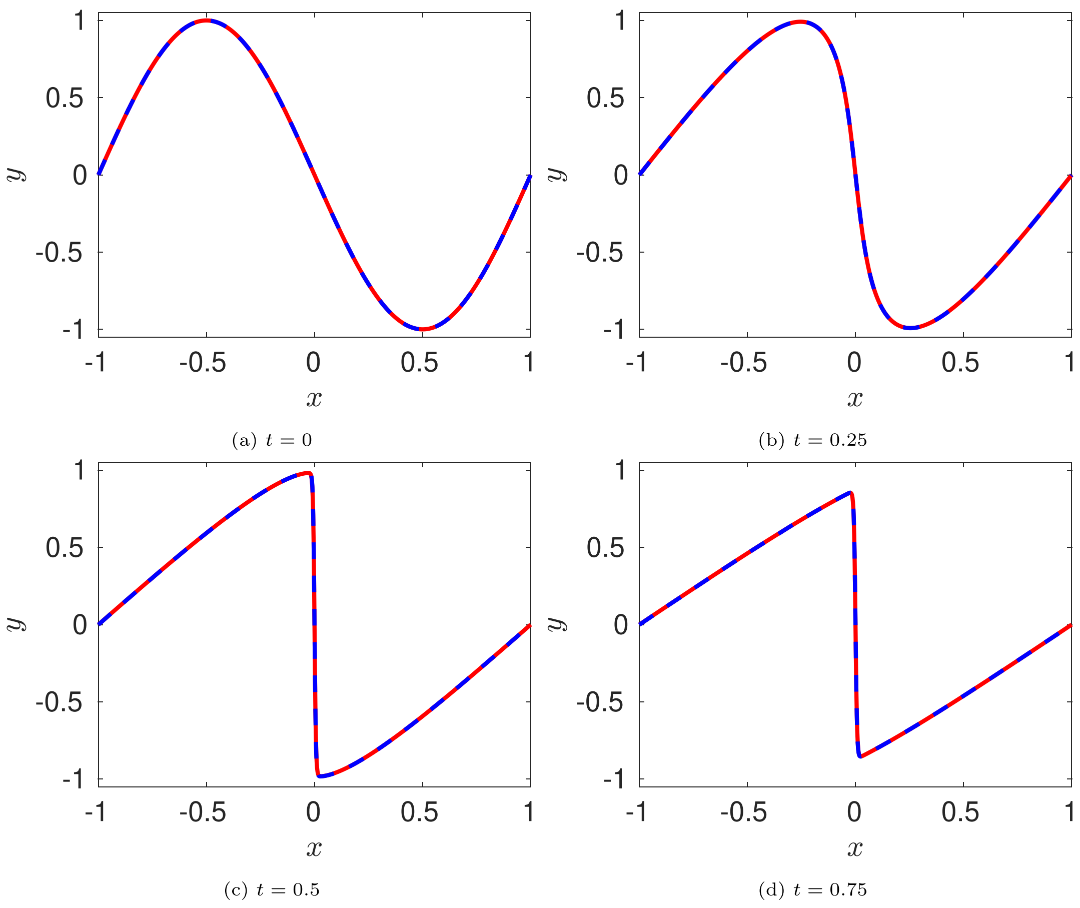


Fig. 14. Solution to the Burgers' equation obtained using DPINN with time-marching (Dashed line).

ensures that the mass conservation equation remains satisfied. The ratio of inertia and viscous terms is called the Reynolds number (*Re*). High values of *Re* indicate turbulent flows, and low values indicate slow viscous flow.

The Navier–Stokes' equations are a system of nonlinear equations that are difficult to solve. The computation of turbulent flows is complicated even for traditional computational methods. Therefore, we demonstrate the applicability of DPINN in solving this complicated nonlinear system of PDEs at low Reynolds numbers.

Since, for the problem of interest, the flow is incompressible, density can be assumed to be constant. Hence, we would train our proposed DPINN to retrieve the pressure and velocity field only. The problem domain and the boundary conditions are specified in Fig. 16. In this case, the Reynolds number ($Re = \frac{U_0 L}{\nu}$) for the flow is 10, where U_0 and L refer to the velocity of lid and length of cavity respectively. The training of the DPINN is performed using 961 collocation points. These points are distributed in ten equally spaced sub-domains along x-direction and y-direction.

We use a two-layer neural network for each distributed domain with five neurons in each layer. In Figs. 17–19, we show the predictions obtained using DPINN. In Fig. 17, we show the streamlines and in Fig. 18 we show velocity and pressure contours obtained using DPINN. These contours and streamlines effectively capture the physical flow pattern that is expected to form for this flow problem.

Further, the v-velocity obtained at the centerline $x = 0.5$ is presented in Fig. 19a and u-velocity obtained at the centerline $y = 0.5$ is presented in Fig. 19b. Both of these results are compared to the well-established results of Marchi et al. [26]. It can be observed that both the centerline velocities obtained using DPINN are in good agreement with the results of Marchi et al. [26].

Although, for the presented problem, DPINNs have been able to find the solution to the Navier–Stokes equation. We want to clarify that, the flow problem simulated in this work (lid-driven cavity) has a low Reynolds number. With increasing Reynolds number, the complexity of the solution increases since the flow becomes unsteady, turbulent, and three-dimensional. However, the current work sets the foundation for the use of DPINNs to directly solve the Navier–Stokes equations.

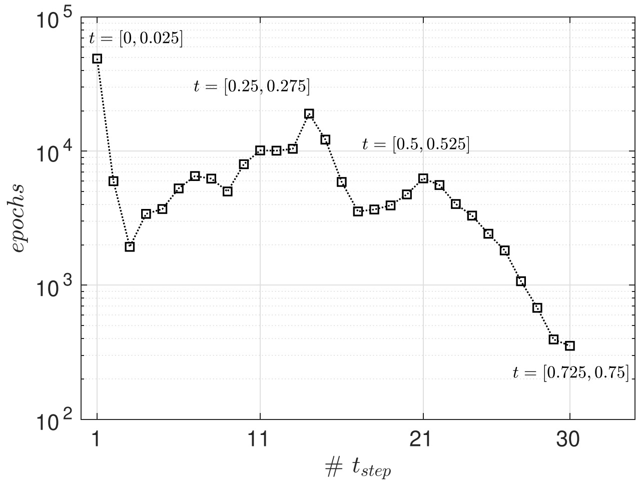
5.2. Inverse problems

We now show the details of the numerical experiments performed for solving the inverse problem using DPINN.

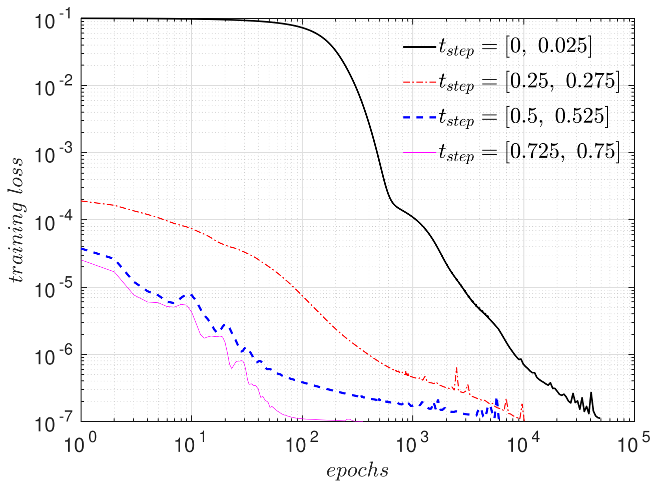
5.2.1. Prediction of thermal properties of heterogeneous media

Reconsider Eq. (21) that describes steady-state conduction in a composite slab. The problem statement for the inverse problem is as follows. Given training data, i.e., solution of Eq. (21) at few locations, and value of heat flux at the left end, find the value of model parameters (K_L, K_R) that best describes the available training data. In the real world, data acquisition is typically prone to errors like instrument errors, measurement errors, etc. To account for these errors in training data, we also added some random noise to the exact solution.

We solved this problem with PINN as well as DPINN. We found that similar to the forward problem, DPINN correctly predicts the value of model parameters, but PINN doesn't. The details of the



(a) Number of epochs required for convergence for each time step.



(b) Training loss history.

Fig. 15. Training loss history for DPINN with time marching. Total 204,595 Epochs were required for convergence (loss ~1e−7) using ADAM optimizer with 5e−4 learning rate. The time required for training was ~ 31 minutes.

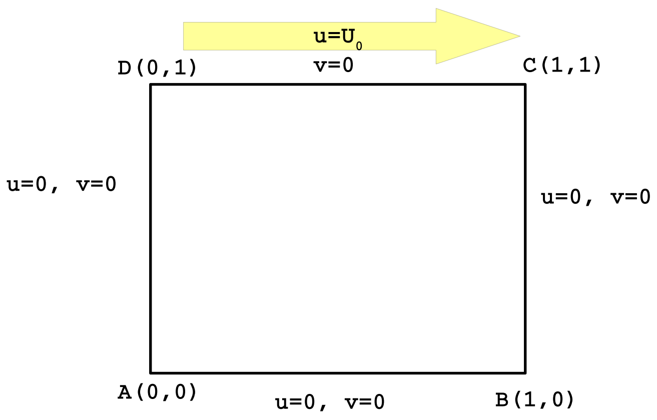


Fig. 16. Schematic of lid-driven cavity problem.

numerical experiment are as follows. A total of 41 equally spaced collocation with $NB_x = 2$ was used to train the DPINN network to infer the conductivity coefficient K. The results of the DPINN pre-

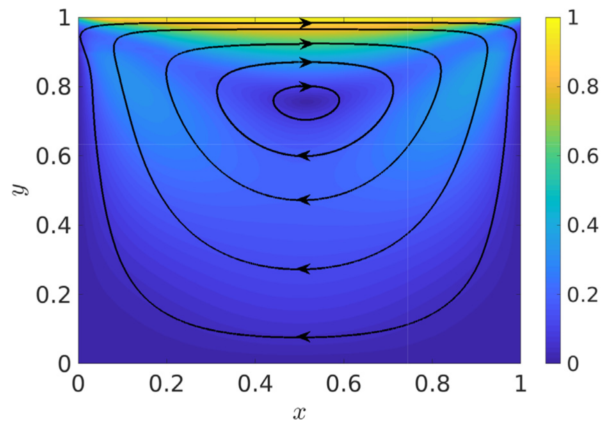


Fig. 17. Streamlines.

dictions for clean, as well as noisy data, are tabulated in Table 3. The predicted temperature profile based on learned parameters by DPINN is shown in Fig. 20. It is worth noting that despite a reasonable 5% noise in the training data, DPINN prediction (Fig. 20c) shows no tendency to overfit the noise in the data. This is a clear demonstration of the regularization effect produced by PDE loss, which restricts the space of possible solutions.

5.2.2. Prediction of wave speed in advection equation

Reconsider the linear advection equation (Eq. 1) that we described in Section 2. The statement of the inverse problem for Eq. (1) is as follows: given solution of PDE (with unknown model parameter λ) at a few sampling points, find the value of λ that best describe the available training data.

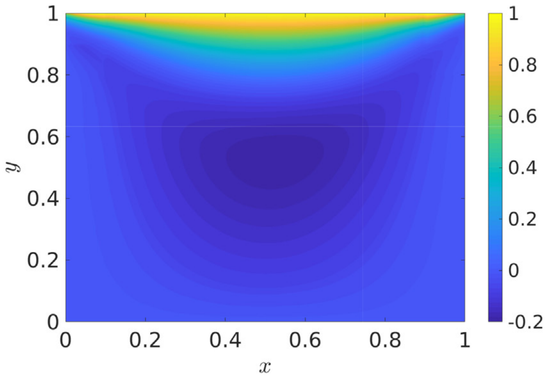
We solved this problem with DPINN. We found that DPINN correctly predicts the value of wave speed. To generate the synthetic dataset, the exact solution [25] of the linear advection equation is used. The network is trained with a cell configuration of $NB_x = 20$, $NB_t = 5$, and trained on the exact solution provided at 101x101 uniformly spaced collocation points. The results of the DPINN predictions for clean, as well as noisy data, are tabulated in Table 4. The predicted u profile at $t = 0.5$ on the basis of learned parameters by DPINN is shown in Fig. 21.

5.2.3. Prediction of coefficients in viscous Burgers' equation

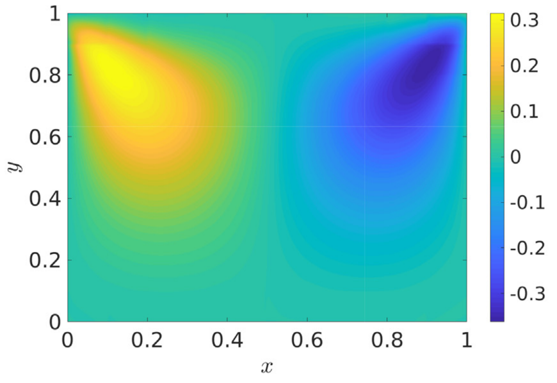
Reconsider the viscous Burgers' equation (Eq. 2) that we described in Section 3.In this problem, our aim was to predict the value of parameter ν using inverse DPINN. To generate the synthetic dataset, we solved the Burgers equation with the Roe scheme [25]. We found that DPINN correctly predicts the value of ν. The results of the DPINN predictions for clean, as well as noisy data, are tabulated in Table 5. The predicted u profile at $t = 0.5$ on the basis of learned parameters by DPINN is shown in Fig. 22. The numerical settings for the inverse problem are the same as the forward problem.

5.2.4. Prediction of viscosity coefficients in 2D steady-state Navier–Stokes equation

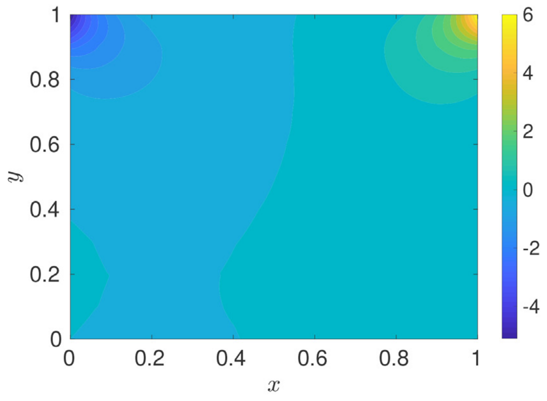
Finally, we end this section by solving the inverse problem of the 2D Navier–Stokes' equation that we solved in Section 5.1.5. Unlike previous inverse problems, the input to this problem are three separate datasets for u, v, and p. The desired output is the value of the parameter ν. To generate the dataset, we used the accurate solution of the Navier–Stokes equation. Keeping the same numerical settings for the inverse problem same as the forward problem, the results of the DPINN predictions for this problem are tabulated in Table 6.



(a) u contours

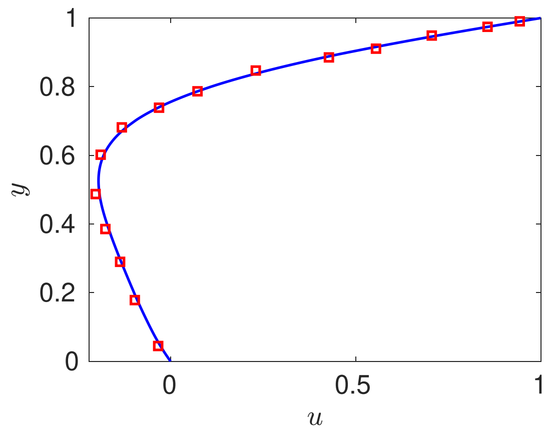


(b) v contours

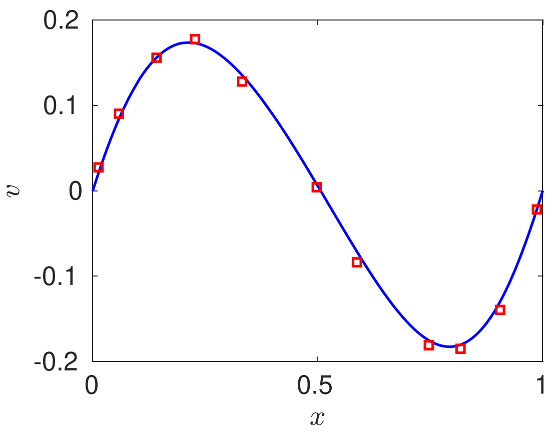


(c) p contours

Fig. 18. Contours of u, v and p.



(a) u center-line velocity component



(b) v center-line velocity component

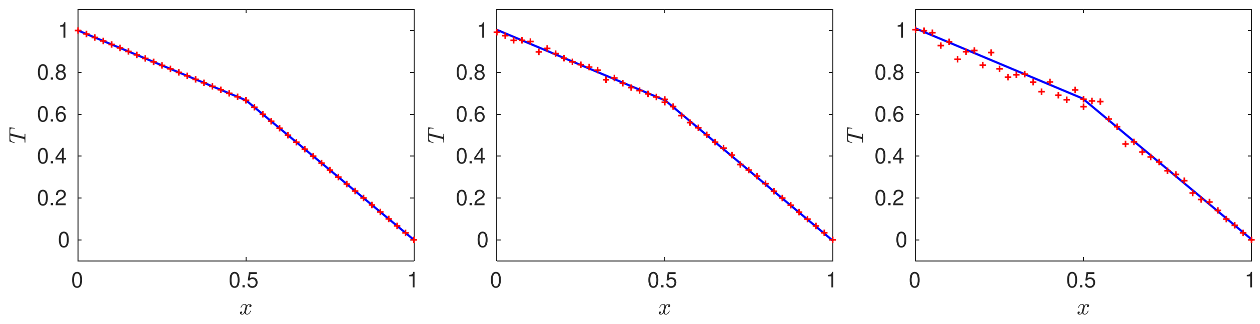
Fig. 19. Center-line velocity components. Blue line: DPINN prediction, Red squares: reference solution.

Table 3
Predicted conduction equation coefficients by inverse DPINN.

Correct PDE	$\frac{\partial}{\partial x}\left(K(x)\frac{\partial T(x)}{\partial x}\right) = 0, K(x) = \begin{cases} 1 & \text{if} x < 0.5 \\ 0.5 & \text{if} x > 0.5 \end{cases}$
Predicted PDE by PINN (noise-free data)	$\frac{\partial}{\partial x}\left(K(x)\frac{\partial T(x)}{\partial x}\right) = 0, K(x) = \begin{cases} 0.9645 & \text{if} x < 0.5 \\ -2.433e-9 & \text{if} x > 0.5 \end{cases}$
Predicted PDE by DPINN (noise-free data)	$\frac{\partial}{\partial x}\left(K(x)\frac{\partial T(x)}{\partial x}\right) = 0, K(x) = \begin{cases} 1.0000 & \text{if} x < 0.5 \\ 0.5000 & \text{if} x > 0.5 \end{cases}$
Predicted PDE by DPINN (1% noise in data)	$\frac{\partial}{\partial x}\left(K(x)\frac{\partial T(x)}{\partial x}\right) = 0, K(x) = \begin{cases} 1.0003 & \text{if} x < 0.5 \\ 0.5008 & \text{if} x > 0.5 \end{cases}$
Predicted PDE by DPINN (5% noise in data)	$\frac{\partial}{\partial x}\left(K(x)\frac{\partial T(x)}{\partial x}\right) = 0, K(x) = \begin{cases} 0.9984 & \text{if} x < 0.5 \\ 0.5038 & \text{if} x > 0.5 \end{cases}$

6. Conclusion and future work

In this work, we have proposed a methodology to develop novel learning machines called DLMs to solve both the forward and inverse problems in time-dependent nonlinear PDEs. Inspired by the traditional numerical methods like FEM and FVM, DLM involves the splitting of the computational domain into an analogous multi-domains structure and assigns each sub-domain to a local learning machine. To synchronize these learning machines, we have introduced additional interface regularization terms in the cost function, which can be physically interpreted as interface



(a) Noise-free data (b) 1% noise in data (c) 5% noise in data

Fig. 20. Predicted conduction equation solution by inverse DPINN (solid line). Symbol + represents the training data used to train DPINN.

Table 4
Predicted Advection equation coefficients by inverse DPINN.

Correct Advection equation	$\frac{\partial u}{\partial t} + 1.5000\frac{\partial u}{\partial x} = 0$
Predicted Advection equation (noise-free data)	$\frac{\partial u}{\partial t} + 1.5000u\frac{\partial u}{\partial x} = 0$
Predicted Advection equation (1% noise in data)	$\frac{\partial u}{\partial t} + 1.4999u\frac{\partial u}{\partial x} = 0$
Predicted Advection equation (5% noise in data)	$\frac{\partial u}{\partial t} + 1.4958u\frac{\partial u}{\partial x} = 0$

Table 5
Predicted Burgers' equation coefficients by inverse DPINN.

Correct Burgers equation	$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - 0.003183\frac{\partial^2 u}{\partial^2 x} = 0$
Predicted Burgers equation (noise-free data)	$\frac{\partial u}{\partial t} + 0.9994u\frac{\partial u}{\partial x} - 0.003188\frac{\partial^2 u}{\partial^2 x} = 0$
Predicted Burgers equation (1% noise in data)	$\frac{\partial u}{\partial t} + 0.9977u\frac{\partial u}{\partial x} - 0.003201\frac{\partial^2 u}{\partial^2 x} = 0$
Predicted Burgers equation (5% noise in data)	$\frac{\partial u}{\partial t} + 0.9932u\frac{\partial u}{\partial x} - 0.003211\frac{\partial^2 u}{\partial^2 x} = 0$

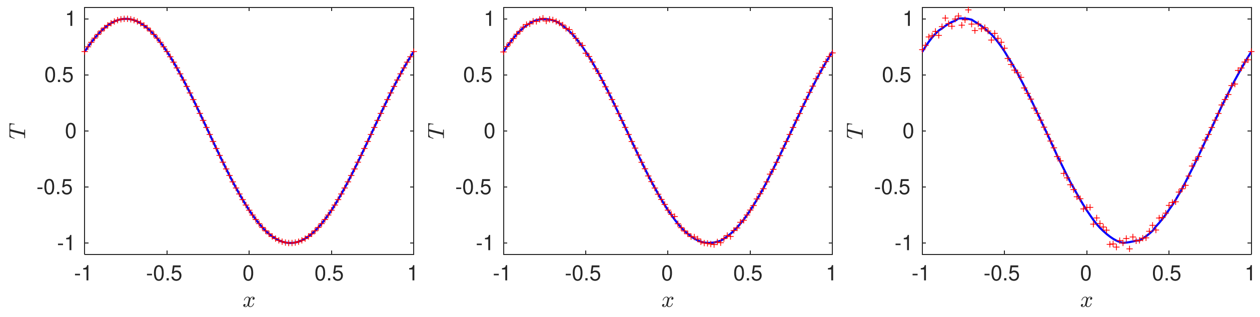
flux conservation conditions from the Finite Volume Method perspective or interface continuity and differentiability conditions from the FEM perspective. This paper presents three different variants of DLMs that combine various ideas from classical numerical methods with existing machine learning techniques.

The first DLM is a modification of DPIELM to solve the time-dependent nonlinear equation. The central idea of this approach is the linearization of the nonlinear PDE about the current time step to predict the solution in future time step. It is equivalent to a linearized, explicit time-stepping method in conventional time marching techniques. The advantage of this DLM is that it enables DPIELM, otherwise limited to linear PDEs, to solve nonlinear PDEs, e.g., Burgers' equation. However, this method cannot robustly capture jump discontinuities and is found to be computationally expensive. Further, the unique feature of ELMs that makes them extremely fast, i.e., independence of prediction from initial layer weights, also deprives them of taking the benefits of transfer learning.

The second DLM, called Distributed Physics Informed Neural Network (DPINN), is a distributed variant of the physics informed neural network (PINN). On one hand, it inherits the powerful approximation properties of PINN. On the other, it is free from the vanishing gradient problems typically found in deep PINNs, because each local PINN is just two layers deep. We showed that DPINN, like PINN, can efficiently solve forward and inverse problems for stationary as well as time-dependent nonlinear PDEs. We tested DPINN against two benchmark nonlinear PDEs–Burgers' equation and Navier–Stokes' equation at low Reynolds number for both forward and inverse problems. Furthermore, we demonstrated that the modality of specifying local flux-based regularization at PINN interfaces gives DPINN a clear edge over its parent PINN.

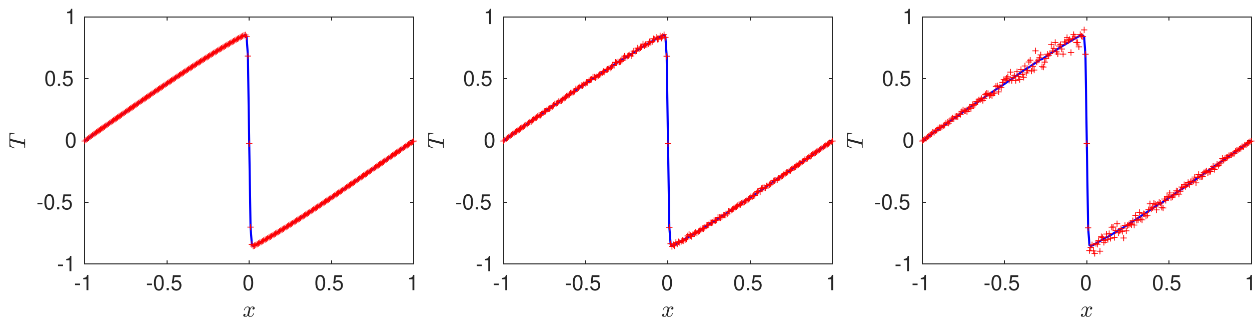
The third DLM is an explicit time-marching version of DPINN, which not only reduces the computational load (due to small explicit time steps) but also accelerates the training procedure by transfer learning, i.e., by taking advantage of learned weights in the present time step to expedite the training in the future time step. For Burgers' equation, we show that each iteration of time-marching DPINN is almost ten times faster than usual DPINN, and to solve the Burgers' equation; the time-marching DPINN takes only one-third of the time required by normal DPINN.

Due to the similarity of these learning machines with the traditional computer-aided engineering (CAE) solvers like ANSYS, COMSOL, etc., these approaches can be easily merged. It ultimately



(a) Noise-free data (b) 1% noise in data (c) 5% noise in data

Fig. 21. Predicted solution by inverse DPINN for advection equation at $t = 0.5$. Solid line represents the DPINN predictions and symbol + represents the training data.



(a) Noise-free data (b) 1% noise in data (c) 5% noise in data

Fig. 22. Predicted solution by inverse DPINN for Burgers' equation at $t = 0.75$. Solid line: DPINN and Symbol $+$: training data.

Table 6
Predicted viscosity coefficient in Navier Stokes equation by inverse DPINN.

Correct 2-D steady momentum equation	$\vec{V}.\vec{\nabla}\vec{V} = -\frac{1}{\rho}\vec{\nabla}p + 0.1000\nabla^2\vec{V}$
Predicted 2-D steady momentum equation	$\vec{V}.\vec{\nabla}\vec{V} = -\frac{1}{\rho}\vec{\nabla}p + 0.1005\nabla^2\vec{V}$

leads towards the promise of hybrid Neural Network-FEM or hybrid Neural Network-FVM schemes in the future.

CRediT authorship contribution statement

Vikas Dwivedi: Investigation, Methodology, Validation, Visualization, Writing - original draft, Writing - review & editing. **Nishant Parashar:** Investigation, Methodology, Validation, Visualization, Writing - original draft, Writing - review & editing. **Balaji Srinivasan:** Conceptualization, Supervision and Funding acquisition.

Declaration of Competing Interest

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.

Acknowledgements

This work was supported by Robert Bosch Centre for Data Science and Artificial Intelligence, Indian Institute of Technology-Madras, Chennai (Project No. CR1920ME615RBCX008832).

Appendix A. Mathematical formulation of PINN

Consider the following 1D unsteady problem

$$\frac{\partial}{\partial t}u(x,t) + \mathcal{N}u(x,t) = R(x,t), (x,t) \in \Omega \tag{A.1}$$

$$u(x,t) = B(x,t), (x,t) \in \partial\Omega, \tag{A.2}$$

$$u(x,0) = F(x), x \in (x_L, x_R), \tag{A.3}$$

where $\mathcal{N}$ is a potentially nonlinear differential operator and $\partial\Omega (= \partial\Omega_L \bigcup \partial\Omega_R)$ is the boundary of computational domain Ω as shown in Fig. 3. It also shows the PINN with collocation points $\left(\vec{x}_f, \vec{t}_f\right)$ at the interior (red triangles) and the boundary points $\left(\vec{x}_{bc}, \vec{t}_{bc}\right)$ at the four faces (blue rectangles).

We approximate $u(x,t)$ with the output $f(x,t)$ of the PINN. The network architecture may be shallow or deep depending upon the non-linearity $\mathcal{N}$. If we denote the errors in approximating the PDE, BCs and IC by $\vec{\xi}_f$, $\vec{\xi}_{bc}$ and $\vec{\xi}_{ic}$ respectively. Then, the expressions for these errors are as follows:

$$\vec{\xi}_f = \frac{\partial \vec{f}}{\partial t} + \mathcal{N}\vec{f} - \vec{R}, \text{ on } \left(\vec{x}_f, \vec{y}_f\right). \tag{A.4}$$

$$\vec{\xi}_{bc} = \vec{f} - \vec{B}, \left(\vec{x}_{bc}, \vec{t}_{bc}\right)_{side\,faces} \tag{A.5}$$

$$\vec{\xi}_{ic} = \vec{f}(.,0) - \vec{F}, \left(\vec{x}_{bc}, \vec{t}_{bc}\right)_{bottom\,face} \tag{A.6}$$

For shallow networks, $\frac{\partial \vec{f}}{\partial t}$ and $\mathcal{N}\vec{f}$ can be determined using hand calculations. However, for deep networks, we have to use finite difference methods or automatic differentiation [27]. The latter is preferred for its computational efficiency. We can recast the PDE, BC, IC system to an optimization problem by minimizing an appropriate loss function. The loss function J to be minimized for a PINN is given by

$$J = \frac{\vec{\xi}_f^T \vec{\xi}_f}{2N_f} + \frac{\vec{\xi}_{bc}^T \vec{\xi}_{bc}}{2N_{bc}} + \frac{\vec{\xi}_{ic}^T \vec{\xi}_{ic}}{2N_{ic}}, \tag{A.7}$$

where N_f, N_{bc} and N_{ic} refer to number of collocation points, boundary condition points in left and right faces and initial condition points at the bottom face respectively. We can see that we have chosen a least square loss function. Now any gradient based optimization routine may be used to minimize J. The weights of the network that result are in essence the parameters that decide the representation of the solution of the PDE. *Learning* therefore is the process of converging to these parameters.

Appendix B. Mathematical formulation of PIELM

Consider the 1D unsteady PDE problem given by Eqs. (A.1)–(A.3). The PIELM for 1D unsteady problem is schematically shown in Fig. B.23. The number of neurons in the hidden layers is N^*. If we define $\vec{\chi} = [x, t, 1]^T, \vec{m} = [m_1, m_2, \ldots, m \quad N^*]^T, \vec{n} = [n_1, n_2, \ldots, n_{N^*}]^T$, $\vec{b} = [b_1, b_2, \ldots, b_{N^*}]^T$ and $\vec{c} = [c_1, c_2, \ldots, c_{N^*}]^T$ then, the output of the k^{th} hidden neuron is

$$h_k = \varphi(z_k),$$

where $z_k = [m_k, n_k, b_k]\vec{\chi}$ and $\varphi = tanh$ is the nonlinear activation function. The PIELM output is given by

$$f\left(\vec{\chi}\right) = \vec{h}\,\vec{c}. \tag{B.1}$$

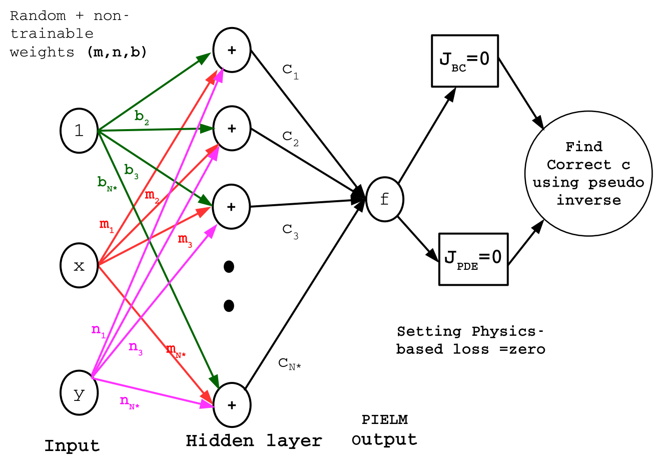

Random + non-trainable weights (m,n,b)
1
x
y
Input
Hidden layer
PIELM output
f
J_BC=0
J_PDE=0
Find Correct c using pseudo inverse
Setting Physics-based loss =zero

Fig. B.23. Overview of the proposed PIELM for 1D unsteady problems.

Given $f\left(\vec{\chi}\right)$, the exact formulae (without any discretization error) for $\frac{\partial^p f}{\partial x^p}$ and $\frac{\partial f}{\partial t}$ are given by

$$\frac{\partial^p f_k}{\partial x^p} = m_k^p \frac{\partial^p \varphi}{\partial z^p}, \tag{B.2}$$

$$\frac{\partial f_k}{\partial t} = n_k \frac{\partial \varphi}{\partial z}. \tag{B.3}$$

The expressions for the errors in approximating the PDE, BCs and IC are same as given in Eqs. (A.4)–(A.6) respectively. Now, if an ELM with N^* hidden nodes can solve this PDE with zero error, it then implies that there exist $\vec{c}, \vec{m}, \vec{n}$ and $\vec{b}$ such that

$$\vec{\xi}_f = \vec{0}, \tag{B.4}$$

$$\vec{\xi}_{bc} = \vec{0}, \tag{B.5}$$

$$\vec{\xi}_{ic} = \vec{0}. \tag{B.6}$$

Eqs. (B.4)–(B.6) can be collectively represented as

$$\boldsymbol{H}\vec{c} = \vec{K}. \tag{B.7}$$

The form of $\boldsymbol{H}$ and $\vec{K}$ depends on the $\mathcal{N}, B$ and F i.e. on the type of PDE, boundary condition and initial condition. In order to find $\vec{c}$, we use the pseudo-inverse as it works well for singular and non square $\boldsymbol{H}$ too.

Appendix C. Mathematical formulation of DPIELM

Referring to the Fig. 4, we write the DPIELM equations to be solved for a 1D unsteady problem. Firstly, we will write the equations arising from the original PIELM formulation. Next, we will write additional interface constraints.

Regular PIELM equations

1. $\vec{\xi}_f^{(i)} = \vec{0}$

$$\left(\frac{\partial \vec{f}^{(i)}}{\partial t} + \mathcal{N}\vec{f}^{(i)} - \vec{R}^{(i)}\right)_{\Omega_i} = \vec{0},\ on\ \left(\vec{x}_f, \vec{y}_f\right)^{(i)}, i = 1, 2, \ldots, N_c \tag{C.1}$$

2. $\vec{\xi}_{bc}^{(i)} = \vec{0}$

$$\left(\vec{f}^{(i)} - \vec{B}^{(i)}\right)_{I_4} = \vec{0},\ on\ \left(\vec{x}_{bc}, \vec{y}_{bc}\right)^{(i)}, i = 1, (1+NB_x), \ldots, (1+(NB_t-1)NB_x) \tag{C.2}$$

$$\left(\vec{f}^{(i)} - \vec{B}^{(i)}\right)_{I_2} = \vec{0},\ on\ \left(\vec{x}_{bc}, \vec{y}_{bc}\right)^{(i)}, i = NB_x, 2NB_x, \ldots, NB_t \times NB_x \tag{C.3}$$

3. $\vec{\xi}_{ic}^{(i)} = \vec{0}$

$$\left(\vec{f}^{(i)} - \vec{F}^{(i)}\right)_{I_1} = \vec{0},\ on\ \left(\vec{x}_f, \vec{y}_f\right)^{(i)}, i = 1, 2, \ldots, NB_x \tag{C.4}$$

Additional interface equations

1. Constraints for C^0 solutions i.e. $\vec{\xi}_{C^0}^{(i)} = \vec{0}$.
   - Continuity along x direction

$$\begin{Bmatrix} \vec{f}^{(\kappa(1)+i-1)} \\ \vec{f}^{(\kappa(2)+i-1)} \\ \cdots \\ \vec{f}^{(\kappa(NB_t)+i-1)} \end{Bmatrix}_{I_2} = \begin{Bmatrix} \vec{f}^{(\kappa(1)+i)} \\ \vec{f}^{(\kappa(2)+i)} \\ \cdots \\ \vec{f}^{(\kappa(NB_t)+i)} \end{Bmatrix}_{I_4}, \ i = 1, 2, \ldots, NB_x - 1 \tag{C.5}$$

where $\kappa = [1, (1+NB_x), \ldots, 1+(NB_t-1)NB_x]^T$.

   - Continuity along t direction

$$\begin{Bmatrix} \vec{f}^{(\kappa(1)+(i-1)NB_x)} \\ \vec{f}^{(\kappa(2)+(i-1)NB_x)} \\ \cdots \\ \vec{f}^{(\kappa(NB_x)+(i-1)NB_x)} \end{Bmatrix}_{I_3} = \begin{Bmatrix} \vec{f}^{(\kappa(1)+iNB_x)} \\ \vec{f}^{(\kappa(2)+iNB_x)} \\ \cdots \\ \vec{f}^{(\kappa(NB_x)+iNB_x)} \end{Bmatrix}_{I_1}, i = 1, 2, \ldots, NB_t - 1 \tag{C.6}$$

where $\kappa = [1, 2, \ldots, NB_x]^T$.

2. Constraints for C^1 solutions i.e. $\vec{\xi}_{C^1}^{(i)} = \vec{0}$.
   - Smooth solutions along x direction

$$\frac{\partial}{\partial x}\begin{Bmatrix} \vec{f}^{(\kappa(1)+i-1)} \\ \vec{f}^{(\kappa(2)+i-1)} \\ \cdots \\ \vec{f}^{(\kappa(NB_t)+i-1)} \end{Bmatrix}_{I_2} = \frac{\partial}{\partial x}\begin{Bmatrix} \vec{f}^{(\kappa(1)+i)} \\ \vec{f}^{(\kappa(2)+i)} \\ \cdots \\ \vec{f}^{(\kappa(NB_t)+i)} \end{Bmatrix}_{I_4}, i = 1, 2, \ldots, NB_x - 1 \tag{C.7}$$

where $\kappa = [1, (1+NB_x), \ldots, 1+(NB_t-1)NB_x]^T$.

Assembly of Eqs. (C.1)–(C.7) leads to a system of linear equations which can be represented as

$$\boldsymbol{H}\vec{c} = \vec{K}, \tag{C.8}$$

where $\vec{c} = \left[\vec{c}^{(1)}, \vec{c}^{(2)}, \ldots, \vec{c}^{(N_c)}\right]^T$. The form of $\boldsymbol{H}$ and $\vec{K}$ depends on the $\mathcal{N}, B$ and F. Finally $\vec{c}$ can be found using pseudo inverse.

References

[1] Y. Liu, E. Racah, Prabhat, J. Correa, A. Khosrowshahi, D. Lavers, K. Kunkel, M. Wehner, W. Collins, Application of deep convolutional neural networks for detecting extreme weather in climate datasets (2016). arXiv:1605.01156.

[2] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, Journal of Fluid Mechanics 807 (2016) 155–166, https://doi.org/10.1017/jfm.2016.615.

[3] T. Corcoran, R. Zamora-Resendiz, X. Liu, S. Crivelli, A spatial mapping algorithm with applications in deep learning-based structure classification (2018). arXiv:1802.02532.

[4] M. Jiang, B. Gallagher, N. Mandell, A. Maguire, K. Henderson, G. Weinert, A deep learning framework for mesh relaxation in arbitrary lagrangian-eulerian simulations, in: Applications of Machine Learning, vol. 11139, International Society for Optics and Photonics, 2019, p. 1113900.

[5] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, et al., Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence, Tech. rep., USDOE Office of Science (SC), Washington, DC (United States), 2019.

[6] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Transactions on Neural Networks 9 (5) (1998) 987–1000, https://doi.org/10.1109/72.712178.

[7] I.E. Lagaris, A.C. Likas, D.G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Transactions on Neural Networks 11 (5) (2000) 1041–1049, https://doi.org/10.1109/72.870037.

[8] W. E, B. Yu, The deep ritz method: A deep learning-based numerical algorithm for solving variational problems, Communications in Mathematics and Statistics 6 (1) (2018) 1–12. doi:10.1007/s40304-018-0127-z. doi: 10.1007/s40304-018-0127-z.

[9] J.H. sci, Relu deep neural networks and linear finite elements, Journal of Computational Mathematics 38 (3) (2020) 502–527. doi:10.4208/jcm.1901-m2018-0160. doi: 10.4208/jcm.1901-m2018-0160.

[10] J. Sirignano, K. Spiliopoulos, Dgm: A deep learning algorithm for solving partial differential equations, Journal of Computational Physics 375 (2018) 1339–1364, https://doi.org/10.1016/j.jcp.2018.08.029, http://www.sciencedirect.com/science/article/pii/S0021999118305527.

[11] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41, https://doi.org/10.1016/j.neucom.2018.06.056.

[12] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Computer Methods in Applied Mechanics and Engineering 361 (2020), https://doi.org/10.1016/j.cma.2019.112732, http://www.sciencedirect.com/science/article/pii/S004578251930622X 112732.

[13] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics 378 (2019) 686–707, https://doi.org/10.1016/j.jcp.2018.10.045.

[14] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, Journal of Computational Physics 404 (2020), https://doi.org/10.1016/j.jcp.2019.109136 109136.

[15] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, Computer Methods in Applied Mechanics and Engineering 360 (2020), https://doi.org/10.1016/j.cma.2019.112789, http://www.sciencedirect.com/science/article/pii/S0045782519306814 112789.

[16] X. Jin, S. Cai, H. Li, G.E. Karniadakis, Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes Eqs. (2020). arXiv:2003.06496.

[17] G. Pang, L. Lu, G.E. Karniadakis, fpinns: Fractional physics-informed neural networks, SIAM Journal on Scientific Computing 41 (4) (2019) A2603–A2626. arXiv:https://doi.org/10.1137/18M1229845, doi:10.1137/18M1229845. doi: 10.1137/18M1229845.

[18] S. Goswami, C. Anitescu, S. Chakraborty, T. Rabczuk, Transfer learning enhanced physics informed neural network for phase-field modeling of fracture, Theoretical and Applied Fracture Mechanics 106 (2020), https://doi.org/10.1016/j.tafmec.2019.102447, http://www.sciencedirect.com/science/article/pii/S016784421930357X 102447.

[19] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, Deepxde: A deep learning library for solving differential Eqs. (2019). arXiv:1907.04502.

[20] D. Zhang, L. Lu, L. Guo, G.E. Karniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, Journal of Computational Physics 397 (2019), https://doi.org/10.1016/j.jcp.2019.07.048, http://www.sciencedirect.com/science/article/pii/S0021999119305340 108850.

[21] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501, https://doi.org/10.1016/j.neucom.2005.12.126.

[22] V. Dwivedi, B. Srinivasan, Physics Informed Extreme Learning Machine (PIELM)–A rapid method for the numerical solution of partial differential equations, Neurocomputing 391 (2020) 96–118, https://doi.org/10.1016/j.neucom.2019.12.099, http://www.sciencedirect.com/science/article/pii/S0925231219318144.

[23] S.H. Strogatz, Nonlinear Dynamics and Chaos: with Applications to Physics, Biology, Chemistry, and Engineering. 1st ed Perseus Books.

[24] F. Charru, Hydrodynamic Instabilities, vol. 37, Cambridge University Press, 2011.

[25] R.J. LeVeque, Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems, vol. 98, SIAM, 2007.

[26] C.H. Marchi, R. Suero, L.K. Araki, The lid-driven square cavity flow: numerical solution with a 1024 x 1024 grid, Journal of the Brazilian Society of Mechanical Sciences and Engineering 31 (2009) 186–198, https://doi.org/10.1590/S1678-58782009000300004.

[27] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: A survey, Journal of Machine Learning Research 18 (1) (2017) 5595–5637, http://dl.acm.org/citation.cfm?id=3122009.3242010.

[28] Vikas Dwivedi, Balaji Srinivasan, Solution of Biharmonic Equation in Complicated Geometries With Physics Informed Extreme Learning Machine, Journal of Computing and Information Science in Engineering 20 (6) (2020), https://doi.org/10.1115/1.4046892, In this issue JCISE-19-1306.

Vikas Dwivedi received his B.E. in Mechanical Engineering from Devi Ahilya Vishwavidyalaya-Indore, India in 2011 and M.Tech in Applied Mechanics from Indian Institute of Technology (IIT)-Delhi in 2014. Currently, he is a Ph.D. student in the Mechanical Engineering Department at IIT-Madras. His research interests include Computational Fluid Dynamics and Applied Machine Learning.

Nishant received his M.Tech in Applied Mechanics from Indian Institute of Technology Delhi (IIT Delhi) in 2014. He is currently a PhD student in the Department of Applied Mechanics, IIT Delhi. His research interests include applications of machine learning for solving PDEs, modelling velocity gradient dynamics in turbulent flows and weather forecasting.

Balaji Srinivasan received his M.S in Aerospace Engineering from Purdue University in 2000 and Ph.D. from the Department of Aeronautics and Astronautics, Stanford University in 2006. He finished his B. Tech in Aerospace Engineering from Indian Institute of Technology (IIT)-Madras in 1998. From 2008-2016, he was serving as a faculty member in the Applied Mechanics Dept at IITDelhi. From 2017, he has been working as an Associate Professor in the Mechanical Engineering Department at IIT-Madras. His research interests include Numerical Analysis, Computational Methods for Partial Differential Equations, Turbulence and Applied Machine Learning.