Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 365 (2020) 113028
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

# Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems

Ameya D. Jagtap[a], Ehsan Kharazmi[a], George Em Karniadakis[a,b,*]

[a] *Division of Applied Mathematics, Brown University, 182 George Street, Providence, RI 02912, USA*
[b] *Pacific Northwest National Laboratory, Richland, WA 99354, USA*

Received 13 December 2019; received in revised form 18 March 2020; accepted 21 March 2020
Available online 11 April 2020

**Abstract**

We propose a conservative physics-informed neural network (cPINN) on discrete domains for nonlinear conservation laws. Here, the term discrete domain represents the discrete sub-domains obtained after division of the computational domain, where PINN is applied and the conservation property of cPINN is obtained by enforcing the flux continuity in the strong form along the sub-domain interfaces. In case of hyperbolic conservation laws, the convective flux contributes at the interfaces, whereas in case of viscous conservation laws, both convective and diffusive fluxes contribute. Apart from the flux continuity condition, an average solution (given by two different neural networks) is also enforced at the common interface between two sub-domains. One can also employ a deep neural network in the domain, where the solution may have complex structure, whereas a shallow neural network can be used in the sub-domains with relatively simple and smooth solutions. Another advantage of the proposed method is the additional freedom it gives in terms of the choice of optimization algorithm and the various training parameters like residual points, activation function, width and depth of the network etc. Various forms of errors involved in cPINN such as optimization, generalization and approximation errors and their sources are discussed briefly. In cPINN, locally adaptive activation functions are used, hence training the model faster compared to its fixed counterparts. Both, forward and inverse problems are solved using the proposed method. Various test cases ranging from scalar nonlinear conservation laws like Burgers, Korteweg–de Vries (KdV) equations to systems of conservation laws, like compressible Euler equations are solved. The lid-driven cavity test case governed by incompressible Navier–Stokes equation is also solved and the results are compared against a benchmark solution. The proposed method enjoys the property of domain decomposition with separate neural networks in each sub-domain, and it efficiently lends itself to parallelized computation, where each sub-domain can be assigned to a different computational node.
Published by Elsevier B.V.

*Keywords:* cPINN; Mortar PINN; Domain decomposition; Machine learning; Conservation laws; Inverse problems

## 1. Introduction

In the last ten years, deep learning in the form of deep neural networks has been extensively used in many diverse disciplines ranging from the classification problem including speech recognition [1], natural language translation [2],

* Corresponding author at: Division of Applied Mathematics, Brown University, 182 George Street, Providence, RI 02912, USA.
*E-mail addresses:* ameyadjagtap@gmail.com, ameya_jagtap@brown.edu (A.D. Jagtap), ehsan_kharazmi@brown.edu (E. Kharazmi), george_karniadakis@brown.edu (G.E. Karniadakis).

https://doi.org/10.1016/j.cma.2020.113028
0045-7825/Published by Elsevier B.V.

and computer vision [3], to complex regression problems like approximation of nonlinear/discontinuous functions. However, their applications are less explored in the area of scientific computing. Recently, solving the underlying governing partial differential equations (PDEs) of physical phenomena using deep learning has emerged as a new field of *scientific machine learning* thanks to the universal approximation and high expressivity of neural networks (NNs) as an ansatz for the solution space of governing PDEs; yet such networks in general are *ignorant* of the underlying physics. The idea of constructing physics-informed learning machines, which makes use of systematically structured prior information about the solution goes back to earlier study of Owhadi [4], which showed the promising approach of exploiting such prior information. Later, Raissi et al. [5,6] employed Gaussian process regression to obtain representation of functionals of linear operators, inferring the solution accurately and also providing uncertainty estimates for many physical problems. They further extended this method to nonlinear problems, solution inference and system identification [7,8]. More recently in [9], they developed the physics-informed neural networks (PINNs) that can accurately solve both forward problems of approximating the solutions of governing mathematical model, as well as inverse problems, where the model parameters are inferred from the observed data. The PINN algorithm infuses the governing equation into the network and thus enriches the loss function by adding a *residual* term from that equation, which essentially acts as a penalizing term to constrain the space of admissible solutions. In this setting, the problem of inferring solutions of PDEs is transformed into an optimization problem of the loss function. A major advantage of this method is providing a mesh-free algorithm as the differential operators in the governing PDEs are approximated by *automatic differentiation* [10]. Therefore, PINNs are fundamentally different from the traditional numerical methods like the finite element method and finite difference method, where the governing PDEs are eventually discretized over the computational domain. However, there are two limitations of PINNs. The first one is the accuracy of the solution, *i.e*, the absolute error does not go below levels of about $10^{-5}$ due to the inaccuracy involved in solving high-dimensional non-convex optimization problem that may result in local minima. The second limitation is the large training cost associated with deep neural networks and long time-integration of the governing PDEs.

The performance of this algorithm, however, is not yet fully investigated in the literature in solving hyperbolic conservation laws such as the compressible Euler equations that can admit discontinuous solutions like shock and contact waves even with sufficiently smooth initial conditions. Such discontinuities lead to excessive complications in choosing the hyper-parameters of PINN and thus obtaining accurate solution close to steep gradients. In [11], the authors showed that the proper clustering of training data points near high gradient region can improve the solution accuracy in that region and thus prevent the propagation of error to the whole domain. This improvement suggests the idea of decomposing the computational domain into several sub-domains and employing a separate localized powerful network in the region of high gradient solution, and thus developing a set of individual local PINNs with different characteristics complied to the known prior knowledge of solution in each sub-domain. The *hp*-refinement via domain decomposition and projection onto space of high-order polynomials has been recently formulated in [12,13], where the authors develop a general framework, namely *hp*-variational physics-informed neural networks. Such idea is also used by Li et al. in [14] where they employed a local neural network on discrete sub-domains and solved PDE using the variational principle. In [15], the authors used PINN to solve conservation laws in graph topologies where domains are connected by interface boundary conditions, and they employed a single network to solve the complete system of conservation laws. In this work, we present a conservative physics-informed neural network (cPINN) on discrete sub-domains where we use a separate PINN in each sub-domains, and then stitch back all sub-domains through the corresponding conservative quantity like flux, which makes this approach unique. Given the sub-domains of a finite, manageable size, a massively parallel computations can be performed using clusters of CPUs and GPUs. This is the best strategy to tackle the problem of *curse of dimensionality*, which is related to the numerical approximation of a high-dimensional problems, both in terms of computational cost and memory requirements. Thus, the proposed cPINN method can efficiently handle high-dimensional problems. The advantages of the cPINN are multi-fold:

- **Domain decomposition**: PINNs do not require the discretization of PDE, yet it can benefit from decomposing the computational domain. In this setting, a tailored network is employed to locally obtain the solution in each sub-domain while preserving global requirements via proper *interface conditions*. This will address the second limitation of PINN as it is a promising approach for parallelization of PINN codes.

- **Reduction of error propagation in the domain**: Single network used in PINN can pollute the solution in the domain, especially during early training period due to only training data points available from initial or boundary conditions (in the absence of any experimental data). But in cPINN, different networks in each sub-domain provide additional information about the fluxes at the interfaces along with initial or boundary training data points. This results in reduction of error propagation in the neighbouring sub-domains as well as faster convergence.
- **Flux continuity at the interfaces**: The underlying conservation law of the physical phenomena controls the required continuity conditions of local solution of individual PINNs at the interface of sub-domains. Additional term(s) are added to the loss function of cPINN to account for such conditions.
- **Efficient hyper-parameter adjustment**: Based on prior (and sparse) knowledge of the solution regularity in each sub-domain, the hyper-parameter set of corresponding PINN is properly adjusted, *i.e.*, the choice of depth, width, activation function, network structure, number and distribution (clustering) of training/penalizing points *etc*.
- **Parallelization capacity**: The partial independence of individual PINNs in decomposed domains can be further employed to implement cPINN in a parallelized algorithm, in which each sub-domain is assigned to a computational node that interacts with other computational nodes through interface conditions.
- **Representation capacity**: Due to deployment of individual network in each sub-domain by the proposed cPINN method, the representation capacity of the network increases. This is important for accurately predicting the complex solution of given PDEs.
- **Efficient handling of inverse problems**: Unlike PINN, cPINN can also handle inverse problem with piecewise constant coefficients efficiently by assigning the different neural networks in each-subdomain.

The rest of the paper is organized as follows: In Section 2 we discussed about the problem setup. Section 3 gives the methodology of the proposed method including a brief discussion on sub-domain loss function, interface conditions, optimization methods and the various forms of errors involved in the cPINN method. Numerical experiments are performed in Section 4, where various comparisons are made between PINN and cPINN methods. One- and two-dimensional test cases like Burgers equation, KdV equation, coupled two-dimensional Burgers equations, incompressible Navier–Stokes equations and compressible Euler equations are solved to verify our claim. Inverse problems are also solved using the proposed methodology. All the test cases are chosen according to the degree of complexity, for example, compressible Euler equations admit discontinuous solution, which are difficult to capture accurately for any numerical method. Finally, in Section 5 we summarize our findings.

## 2. Problem setup

In general, the parameterized conservation law is given by

$$u_t + f(u, u_x, u_{xx}, \ldots; \lambda_1, \lambda_2, \ldots)_x = 0 \ , x \in \Omega \subset \mathbb{R}^d, \ t > 0, \tag{1}$$

subject to proper boundary conditions and initial condition $u(x, 0) = u_0$. $u(t, x)$ denotes the solution of the governing conservation law and $f(\ .\ ; \lambda_1, \lambda_2, \ldots)$ contains the inviscid and higher-order fluxes, parameterized by $\lambda_i$'s. As an example, the one-dimensional viscous Burgers equation flux is given as $f = u^2/2 - \lambda u_x$, where the parameter $\lambda$ plays the role of viscosity coefficient.

In this paper, we shall solve both forward problems where solutions of partial differential equations are inferred given fixed model parameters $\lambda_i$'s as well as inverse problems, where the unknown parameters $\lambda_i$'s are learned from the observed data. The given mathematical model is converted into a surrogate model, more specifically, a given problem of solving PDE is converted into a minimization problem where global minima of loss function correspond to the solution of the PDE. The loss function can be defined using training data points like initial and boundary conditions and the residual of the given PDE. For Eq. (1), the residual $\mathcal{F}$ is given by

$$\mathcal{F} := u_t + f(u, u_x, u_{xx}, \ldots; \lambda_1, \lambda_2, \ldots)_x.$$

## 3. Methodology

Let $\mathcal{N}^L : \mathbb{R}^{D_i} \to \mathbb{R}^{D_o}$ be a feed-forward neural network of $L$ layers and $N_k$ neurons in $k$th layer ($N_0 = D_i$, and $N_L = D_o$). The weight matrix and bias vector in the $k$th layer ($1 \leq k \leq L$) are denoted by $\mathbf{W}^k \in \mathbb{R}^{N_k \times N_{k-1}}$

and $\mathbf{b}^k \in \mathbb{R}^{N_k}$ respectively. The input vector is denoted by $\mathbf{z} \in \mathbb{R}^{D_i}$ and the output vector at $k$th layer is denoted by $\mathcal{N}^k(\mathbf{z})$ and $\mathcal{N}^0(\mathbf{z}) = \mathbf{z}$. We denote the activation function by $\Phi$ which is applied layer-wise along with the scalable parameters $na^k$, where $n$ is the scaling factor. Layer-wise introduction of the additional parameters $a^k$ changes the slope of activation function in each hidden-layer thereby increasing the training speed. Moreover, these activation slopes can also contribute to the loss function through the slope recovery term, see [16,17] for more details. The slope recovery term based on the layer-wise activation slope $a^k$ [17] in the loss function is defined as

$$\mathcal{S}(a) := \frac{1}{\frac{1}{(L-1)}\sum_{k=1}^{L-1} \exp(a^k)}.$$

Such locally adaptive activation functions enhance the learning capacity of the network, especially during the early training period. Mathematically, one can prove this by comparing the gradient dynamics of the adaptive activation function method against that of the fixed activation method. The gradient dynamics of the adaptive activation modifies the standard dynamics (fixed activation) by multiplying a conditioning matrix by the gradient and by adding the approximate second-order term. In this paper, we used scaling factor $n = 5$ for all hidden-layers and initialize $na^k = 1, \ \forall k$.

The $(L-1)$-hidden layer feed-forward neural network is defined as

$$\mathcal{N}^k(\mathbf{z}) = \mathbf{W}^k \Phi(a^{k-1}\mathcal{N}^{k-1}(\mathbf{z})) + \mathbf{b}^k \in \mathbb{R}^{N_k}, \qquad 2 \leq k \leq L \tag{2}$$

and $\mathcal{N}^1(\mathbf{z}) = \mathbf{W}^1\mathbf{z} + \mathbf{b}^1$, where in the last hidden-layer, the activation function is identity. By letting $\tilde{\mathbf{\Theta}} = \{\mathbf{W}^k, \mathbf{b}^k, a^k\}$ as the collection of all weights, biases, and slopes, we can write the output of the neural network by

$$\mathbf{u}_{\tilde{\mathbf{\Theta}}}(\mathbf{z}) = \mathcal{N}^L(\mathbf{z}; \tilde{\mathbf{\Theta}}),$$

where $\mathcal{N}^L(\mathbf{z}; \tilde{\mathbf{\Theta}})$ emphasizes the dependence of the neural network output $\mathcal{N}^L(\mathbf{z})$ on $\tilde{\mathbf{\Theta}}$. In general, weights and biases are initialized from known probability distributions. Xavier initialization [18] is one of the most widely used initialization methods, which initializes the weights in the network by drawing them from a distribution with zero mean and a finite variance. The optimal value of variance is $1/N$, where $N$ is the number of nodes feeding into that layer.

Fig. 1 shows the schematic of cPINN algorithm, where along with NN and PDE part, additional interface conditions are also contributing to the loss function. The interface condition includes flux continuity conditions in strong form as well as enforcing the average solution given by two NNs along the common interface. Although it is not necessary to enforce the average solution along the common interface, our numerical experiments reveal that it will drastically speed-up the convergence rate. Fig. 2 shows the schematic representation of PINN and cPINN methods. Unlike PINN, cPINN divides the domain into number of small sub-domains, in which we can employ completely different neural networks (here we refer to as sub-PINN networks) with different architecture to obtain the solution of the same underlying PDE. Such domain decomposition also offers easy parallelization of the network, which is quite important in terms of achieving computational efficiency. Another aspect of the proposed algorithm is that we can freely choose network hyper-parameters like optimization method, activation function, depth or width of the network depending on some intuitive knowledge of the solution regularity in each sub-domain. In case of smooth zones, we can use a shallow network, whereas a deep neural network can be employed in a region where a complex nature of the solution is expected.

The cPINN formulation in fact considers the flux continuity at the interfaces of each sub-domain, which is inspired by the conservation laws. The total solution in this setting is reconstructed by stitching all the solutions in each sub-domain by using the proper interface conditions. This approach can be extended to a more general case, hereafter called as *Mortar PINN* for joining the non-overlapping decomposed domains, where the interface conditions are not necessarily coming from the conservation laws and rather depending on the corresponding governing equation in each particular problem. Here, we only focus on the conservation laws and thus, develop cPINNs. In cPINN, the output of neural network in the $p$th sub-domain is given by

$$\mathbf{u}_{\tilde{\mathbf{\Theta}}_p}(\mathbf{z}) = \mathcal{N}^L(\mathbf{z}; \tilde{\mathbf{\Theta}}_p), \quad p = 1, 2, \ldots, N_{sd},$$

where $N_{sd}$ represent the total number of sub-domains. The final solution will be obtained as

$$\mathbf{u}_{\tilde{\mathbf{\Theta}}}(\mathbf{z}) = \bigcup_{p=1}^{N_{sd}} \mathbf{u}_{\tilde{\mathbf{\Theta}}_p}(\mathbf{z}).$$

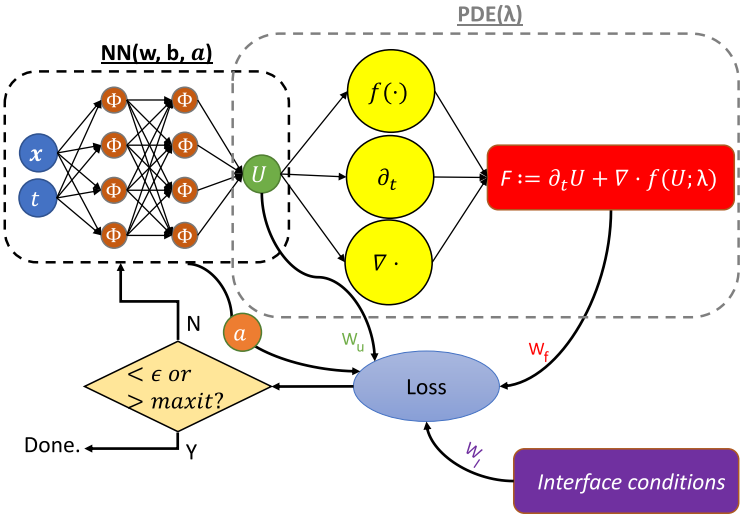


**Fig. 1.** Schematic of cPINN with an adaptive activation function. Along with neural network and PDE parts, cPINN has additional interface conditions that contribute to the loss function.

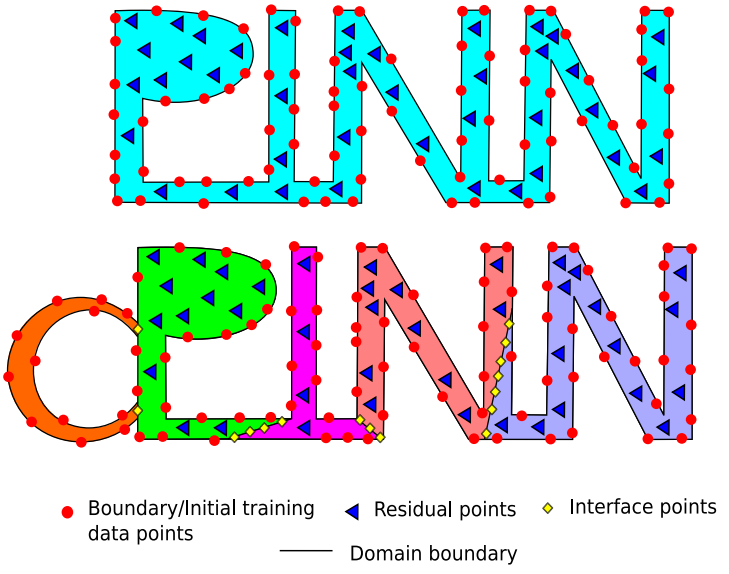


**Fig. 2.** Training, residual points in PINN (top) and cPINN (bottom). The training data points are red circles, residual points are blue triangle, and interface data points are yellow diamonds.

In many applications, the goal is to train a neural network using an available set of training data. In general, the training data set for solving PDEs can be obtained from the initial and boundary conditions. In particular, we can also obtain such data at certain locations in the domain from carefully performed experiments and define a low- or high-fidelity data as per the possible error involved in the measurements. A high resolution numerical simulation can also render a good training data set.

### *3.1. Sub-domain loss function and optimization algorithm*

Let $\{\mathbf{x}_u^i\}_{i=1}^{N_u}$, $\{\mathbf{x}_F^i\}_{i=1}^{N_F}$ and $\{\mathbf{x}_I^i\}_{i=1}^{N_I}$ be the set of randomly selected training, residual, and interface points, respectively. These points are usually drawn from a distribution, which is usually not known a priori and often

need to be chosen from the given input training data. The cPINN algorithm aims to learn a surrogate $u = u_{\tilde{\Theta}}$ for predicting the solution $u$ of the given PDE. The loss function of cPINN is defined sub-domain wise, which has a similar structure as the PINN loss function in each sub-domain but is endowed with the interface conditions. For the forward problem, we define the loss function in the $p$th sub-domain as

$$\mathcal{L}(\tilde{\Theta}_p) = W_{u_p} \times \text{MSE}_{u_p} + W_{\mathcal{F}_p} \times \text{MSE}_{\mathcal{F}_p} + W_{I_p} \times \underbrace{(\text{MSE}_{flux} + \text{MSE}_{u_{avg}})}_{\text{Interface conditions}}, \quad p = 1, 2, \ldots, N_{sd}, \tag{3}$$

where $W_{u_p}$, $W_{\mathcal{F}_p}$ and $W_{I_p}$ are the boundary/initial, residual and interface weights. The MSE stands for mean squared error and is given for each term by

$$\text{MSE}_{u_p} = \frac{1}{N_{u_p}} \sum_{i=1}^{N_{u_p}} \left| u^i - u(\mathbf{x}^i_{u_p}) \right|^2,$$

$$\text{MSE}_{\mathcal{F}_p} = \frac{1}{N_{F_p}} \sum_{i=1}^{N_{F_p}} \left| \mathcal{F}(\mathbf{x}^i_{F_p}) \right|^2,$$

$$\text{MSE}_{flux} = \frac{1}{N_{I_p}} \sum_{i=1}^{N_{I_p}} \left| f_p(u(\mathbf{x}^i_{I_p}))\cdot \mathbf{n} - f_{p^+}(u(\mathbf{x}^i_{I_p}))\cdot \mathbf{n} \right|^2,$$

$$\text{MSE}_{u_{avg}} = \frac{1}{N_{I_p}} \sum_{i=1}^{N_{I_p}} \left| u_p(\mathbf{x}^i_{I_p}) - \left\{\!\!\left\{ u(\mathbf{x}^i_{I_p}) \right\}\!\!\right\} \right|^2,$$

where $\mathcal{F}$ is the residual of the governing PDEs, $f_p \cdot \mathbf{n}$ and $f_{p^+} \cdot \mathbf{n}$ are the interface fluxes normal to the common interfaces given by two different neural networks on sub-domains $p$ and $p^+$, respectively; the superscript $+$ represents the neighbouring sub-domain. $N_{u_p}$, $N_{F_p}$, and $N_{I_p}$ represent the number of boundary/initial training data points, the number of residual points, and the number of points on the common interface in the $p$th sub-domain, respectively. $\mathbf{x}^i_{u_p}$ and $\mathbf{x}^i_{F_p}$ represent the coordinates of boundary/initial training data points and residual points in the $p$th sub-domain, respectively, whereas $\mathbf{x}^i_{I_p}$ represents the coordinates of common interface points between two sub-domains $p$ and $p^+$. The average value of $u$ along the common interface is given by

$$\{\!\{u\}\!\} = u_{avg} := \frac{u_p + u_{p^+}}{2}.$$

**Remark 1.** The interface weight $W_{I_p}$ plays an important role in the convergence of cPINN algorithm due to additional information provided in terms of fluxes/average solution at the interfaces. It can be seen from the numerical experiments that by fine tuning these weights we can obtain a faster convergence.

**Remark 2.** In case of smooth solution along the interface, the enforcement of average solution along these interfaces is similar to the solution continuity condition. However, in case of discontinuous solution (which can be expected in hyperbolic conservation laws), such enforcement forces the adjacent networks to satisfy the average value of the discontinuous solution along the interface.

**Remark 3.** The role of interface points is to stitch the sub-domains together, which is essential for the loss function to converge faster. Hence, a sufficient number of interface points must be taken while stitching the sub-domains together, which are densely distributed along the interface line. We can also use the knowledge of solution on the interface lines obtained from each sub-network to properly select the interface points.

Next, we shall focus our attention on inverse problems, in particular, problem of data-driven discovery of partial differential equation from the known unique solution. Inverse problems are some of the most important problems in science, and, in general, they are highly ill-posed in nature. In the inverse problem where constant coefficients are identified from the known solution, the loss function in the $p$th sub-domain is given as

$$\mathcal{L}(\tilde{\Theta}_p) = W_{u_p} \times \text{MSE}_{u_p} + W_{\mathcal{F}_p} \times \text{MSE}_{\mathcal{F}_p} + W_{I_p} \times \underbrace{(\text{MSE}_{flux} + \text{MSE}_{u_{avg}} + \text{MSE}_{\lambda})}_{\text{Interface conditions}}, \quad p = 1, 2, \ldots, N_{sd}, \tag{4}$$

where $\text{MSE}_{\mathcal{F}_p}$ and $\text{MSE}_\lambda$ are

$$\text{MSE}_{\mathcal{F}_p} = \frac{1}{N_{u_p}} \sum_{i=1}^{N_{u_p}} \left| \mathcal{F}(\mathbf{x}^i_{u_p}) \right|^2,$$

$$\text{MSE}_\lambda = \frac{1}{N_{I_p}} \sum_{i=1}^{N_{I_p}} \left| \lambda_p(\mathbf{x}^i_{I_p}) - \lambda_{p^+}(\mathbf{x}^i_{I_p}) \right|^2.$$

The $\text{MSE}_\lambda$ term gives the continuity condition for the value of constant coefficients on the interface. The main advantage of cPINN method over PINN for solving inverse problems is, cPINN can handle piecewise constant coefficients by assigning different network in each sub-domain.

We seek to find $\tilde{\mathbf{\Theta}}^*_p$ that minimizes the loss function $\mathcal{L}(\tilde{\mathbf{\Theta}}_p)$ in each sub-domain. Even though there is no theoretical guarantee that the above mentioned procedure converges to a global minimum, our numerical solution indicates that as long as the given PDE is well-posed and has a unique solution, the cPINN formulation is capable of achieving an accurate solution provided sufficiently expressive network and sufficient number of residual points are used. There are several optimization algorithms available to minimize the loss function. The stochastic gradient descent (SGD) method is the widely used optimization methods in the machine learning community. A brief survey on SGD is given by [19]. In particular, we shall use the ADAM optimizer which is one version of SGD [20].

**Algorithm 1:** cPINN algorithm

**Step 1** : Specify the training set over all sub-domains

*Training data* : $\mathbf{u}_{\tilde{\mathbf{\Theta}}_p}$ network $\{\mathbf{x}^i_{u_p}\}_{i=1}^{N_{u_p}}, \quad p = 1, 2, \cdots, N_{sd}.$

*Residual training points* : $\mathcal{F}$ network $\{\mathbf{x}^i_{F_p}\}_{i=1}^{N_{F_p}}, \quad p = 1, 2, \cdots, N_{sd}.$

**Step 2** : Specify the interface points

*Interface points* : $\{\mathbf{x}^i_{I_p}\}_{i=1}^{N_{I_p}}$.

**Step 3**: Assign the common interface points to the neighbouring sub-domains.

**Step 4** : Construct the neural network $\mathbf{u}_{\tilde{\mathbf{\Theta}}_p}$ with random initialization of parameters $\tilde{\mathbf{\Theta}}_p$ in each sub-domain.

**Step 5** : Construct the residual neural network $\mathcal{F}$ in each sub-domain by substituting surrogate $\mathbf{u}_{\tilde{\mathbf{\Theta}}_p}$ into the governing equations using automatic differentiation and other arithmetic operations.

**Step 6**: Specify the loss function in the $p^{th}$ sub-domain as

$$\mathcal{L}(\tilde{\mathbf{\Theta}}_p) = \frac{W_{u_p}}{N_{u_p}} \sum_{i=1}^{N_{u_p}} \left| u^i - u(\mathbf{x}^i_{u_p}) \right|^2 + \frac{W_{\mathcal{F}_p}}{N_{F_p}} \sum_{i=1}^{N_{F_p}} \left| \mathcal{F}(\mathbf{x}^i_{F_p}) \right|^2 + W_{I_p} \times \text{Interface Conditions}, \quad p = 1, 2, \cdots, N_{sd}.$$

**Step 7**: Find the best parameters using suitable optimization method for minimizing the loss function in each sub-domain

$$\tilde{\mathbf{\Theta}}^*_p = \arg\min_{\tilde{\mathbf{\Theta}}_p} (\mathcal{L}(\tilde{\mathbf{\Theta}}_p)), \quad p = 1, 2, \cdots, N_{sd}. \tag{5}$$

### 3.2. Error analysis for cPINN

Neural network is a universal function approximator [21]. In [22], Pinkus showed that using a single hidden-layer neural network can approximate any function. In [23], Chen and Chen extended this result by showing that the neural networks can in fact approximate functionals and even nonlinear operators with arbitrarily good accuracy. The rigorous numerical analysis of (deep) neural networks in solving differential equations, however, is still an open problem in the literature. Here, we discuss different sources of error associated with the cPINN formulation in solving differential equations.

Let $F_i$ and $u_i$ be the family of functions that can be represented by the chosen neural network and the exact solution of PDE, respectively in each sub-domain $i = 1, 2, \ldots, N_{sd}$. Then, we define $u_{a_i} = \arg\min_{f \in F_i} \|f - u_i\|$ as the best approximation to the exact solution $u_i$. Let $u_{t_i} = \arg\min_{\tilde{\mathbf{\Theta}}_i} \mathcal{L}(\tilde{\mathbf{\Theta}}_i)$ be the solution obtained by training the

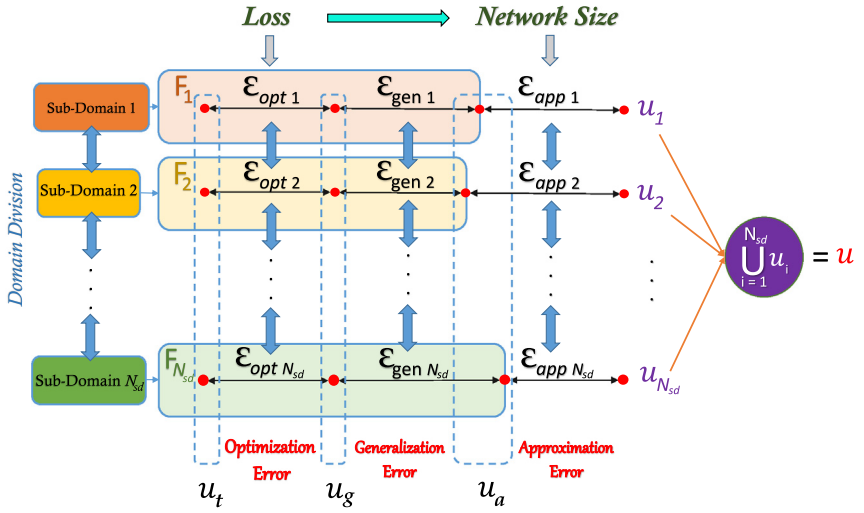


**Fig. 3.** Illustrating the errors involved in cPINN. The total error in each sub-domain consists of the optimization error, the generalization error and the approximation error. $u$ represents the solution of the governing equation, $u_t$ is the solution obtained by training neural network, $u_g$ is the solution of the neural network at global minimum, and $u_a$ is the best function close to $u$ in the corresponding function space $F_i$, $i = 1, 2, \ldots, N_{sd}$.

$i$th sub-net and $u_{g_i} = \arg\min_{\tilde{\Theta}_i} \mathcal{L}(\tilde{\Theta}_i)$ be the solution of the $i$th sub-net at global minimum. Therefore, the total error in each sub-domain $i = 1, 2, \ldots, N_{sd}$ with its own sub-neural network consists of an approximation error $\mathcal{E}_{app\ i} = \|u_i - u_{a_i}\|$, the generalization error $\mathcal{E}_{gen\ i} = \|u_{g_i} - u_{a_i}\|$, and the optimization error $\mathcal{E}_{opt\ i} = \|u_{t_i} - u_{g_i}\|$. Fig. 3 shows a sketch of all three errors involved in cPINN. A deep neural network may reduce the approximation error by increasing the network expressivity, however, it may yield a large generalization error, which is known as the *bias–variance* trade-off. In PINNs, two important factors in the generalization error are the number and location (distribution) of residual points as these factors can adversely alter the configuration of loss function; an example is a sparse set of residual points close to steep changes in the solution. The optimization error is introduced due to the complexity of loss function. The structure of the network, i.e, depth, width, and connections strongly affect the optimization error. Other hyper-parameters of the network such as learning rate, number of iterations etc. can be tuned to further control and improve this error. We see in Fig. 3 that the solution of PDE, $u$, is finally given by the union of solutions given by each sub-network in the corresponding sub-domain, where all these errors in each sub-domain are inter-connected through the interface conditions as shown by the double-headed arrows. Thus, we define the total error in cPINN as

$$\mathcal{E}_{cPINN} := \|u_t - u\| \ \leq \ \|u_t - u_g\| \ + \ \|u_g - u_a\| \ + \ \|u_a - u\|, \tag{6}$$

where $(u, u_t, u_g, u_a) = \bigcup_{i=1}^{N_{sd}} (u_i, u_{t_i}, u_{g_i}, u_{a_i})$, respectively.

### 3.2.1. Advantages of cPINN over PINN

Domain decomposition in cPINN along with employment of individual neural network gives the following advantages for cPINN over PINN algorithm.

- cPINN can reduce the generalization error by selecting the number/location of penalizing points carefully. This can be done during division of sub-domains with some knowledge about the solution behaviour such that the localized powerful neural network can be employed in that sub-domain.
- cPINN can also reduce the approximation error by selecting the network size, and other hyper-parameters such as activation function, optimization algorithm, learning rate etc. in each sub-domain depending on the complexity of the solution.
- The optimization error in cPINN can be reduced by using the different network size as well as a function space that belongs to each sub-domain, which results in a simplified optimization problem.

**Table 1**
Two different neural network architectures used to solve one-dimensional Burgers equation.

| Domain number | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| # Layers | 6 | 6 | 6 | 6 |
| # Neurons | 20 | 20 | 20 | 20 |
| # Residual points | 6000 | 8000 | 4000 | 4000 |
| Adaptive activation function | tanh | tanh | tanh | tanh |
| Rel. $L_2$ error | 6.591226e−3 | 5.855430e−2 | 7.903244e−3 | 4.530096e−3 |
| Domain number | 1 | 2 | 3 | 4 |
| # Layers | 2 | 6 | 3 | 2 |
| # Neurons | 20 | 30 | 25 | 20 |
| # Residual points | 6000 | 8000 | 4000 | 4000 |
| Adaptive activation function | cos | sin | tanh | sin |
| Rel. $L_2$ error | 7.475894e−3 | 3.904935e−2 | 7.174811e−3 | 5.429780e−3 |

## 4. Numerical experiments

This section provides detailed numerical experiments on various nonlinear conservation laws including Burgers equation (scalar and coupled vector equations), KdV equation, incompressible Navier–Stokes equations and compressible Euler equations. Both forward and inverse problems are solved using the proposed cPINN algorithm. All the test cases are solved with double precision arithmetic.

### 4.1. Viscous Burgers equation

The one-dimensional viscous Burgers equation is given by

$$u_t + uu_x = \nu u_{xx}, \quad x \in [-1,\ 1],\ t > 0 \tag{7}$$

with initial condition $u(x, 0) = -\sin(\pi x)$, boundary conditions $u(-1, t) = u(1, t) = 0$ and $\nu = 0.01/\pi$. The analytical solution can be obtained using the Hopf–Cole transformation, see Basdevant et al. [24] for more details. The nonlinearity in the convection term develops very steep solution due to small value of diffusion coefficient $\nu$. For this problem, the domain is divided into four sub-domains with interface locations $x = [-0.2,\ 0.1,\ 0.5]$. Table 1 shows the two different neural network architecture used to solve the one-dimensional Burgers equation. In the top table, the number of hidden layers and the number of neurons in each layer are same, also the same activation function '*tanh*' is used in all sub-domains. In the bottom table, the number of hidden layers as well as the number of neurons are different. Moreover, different activation functions are used in the subdomains. For both architectures, the residual points are the same in all the sub-domains. Both architectures give comparable results, even their relative $L_2$ error (as shown in Table 1) is of the same order in each sub-domains. The cPINN simulation is performed for 15 000 iterations. Fig. 4 shows the contour plots of solution of the Burgers equation, where the top left figure shows the sub-domain numbering system. The solution predicted by cPINN and point-wise error are given by top middle, and top right figure, respectively. The maximum point-wise error given by cPINN occurs near the high gradient region. White horizontal lines in the predicted solution show the interface positions. The bottom row shows the comparison of exact and cPINN solution at $t = 0.14, 0.69$ and $0.99$ locations. The predicted solution agrees well with the exact one. Fig. 5 (left) shows the loss history over the number of iterations, where all sub-domain losses are converging and still going down after 15 000 iterations. In order to see the effect of enforcement of average solution on the common interface, we plot the relative $L_2$ error for full computational domain with and without average solution term. Fig. 5 (right) shows this comparison, where it is clearly visible that with enforcement of the average solution on the common interface, the error decreases faster. Table 2 shows the relative $L_2$ error in the cPINN solution with varying width and depth of the neural network. All simulations are performed till 15 000 iterations. Other network hyper-parameters are given in the bottom Table 1. We observed that with increase in the depth as well as width of the network we can obtain an accurate solution.

In summary, in this example we tried to demonstrate the flexibility of cPINNs with respect to domain decomposition, architecture, and hyper-parameters but we did not attempt to optimize any of these. This can be done in a broader sense by a meta-learning approach, which employs an outside optimization loop of the parameters.

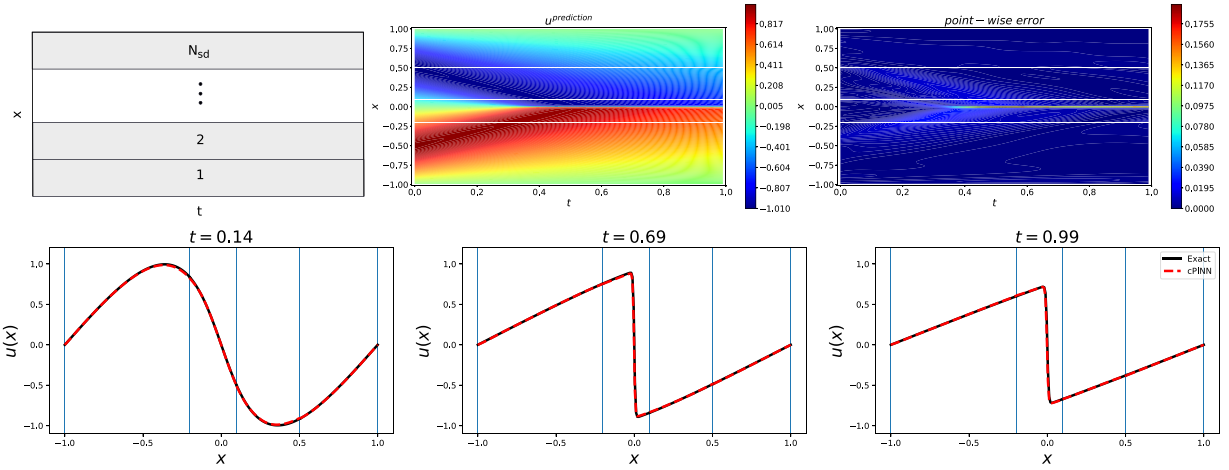


**Fig. 4.** $N_{sd}$ sub-domain numbering, cPINN solution and point-wise error (top row, left to right), and comparison of exact and predicted solution at $t = 0.14, 0.69$ and $0.99$ locations (bottom row) for the one-dimensional viscous Burgers equation. The vertical blue lines give the location of interface positions.

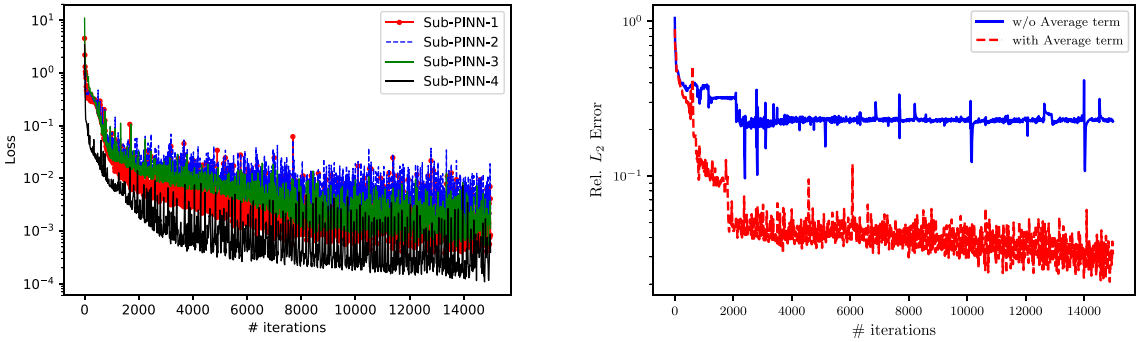


**Fig. 5.** Loss history for one-dimensional viscous Burgers equation (left) and variation of relative $L_2$ error with and without average term over number of iterations (right).

**Table 2**
Viscous Burgers equation: Relative $L_2$ error in the solution with varying width and depth of the fully connected feed forward neural networks. These errors represent the average of 5 different runs corresponding to different initialization.

| Depth | Width → | | | |
|---|---|---|---|---|
| ↓ | 4 | 8 | 16 | 32 |
| 1 | 3.53219e−01 | 3.40286e−01 | 2.50904e−01 | 2.65794e−01 |
| 2 | 2.59507e−01 | 2.04275e−01 | 1.41734e−01 | 1.03863e−01 |
| 4 | 1.42021e−01 | 1.84809e−01 | 2.00492e−02 | 2.24178e−02 |
| 8 | 9.72168e−02 | 1.46037e−02 | 3.08018e−02 | 4.19013e−02 |

### *4.2. Korteweg–de Vries equation*

KdV is a mathematical model of shallow water surface waves. It arises in many physical problems like ion acoustic waves in a plasma, acoustic waves on a crystal lattice *etc*. It also possess a soliton solution which does

**Table 3**
cPINN architecture for the KdV equation.

| Domain number | 1 | 2 |
|---|---|---|
| # Layers | 10 | 10 |
| # Neurons | 20 | 20 |
| # Residual points | 4000 | 14 000 |
| Adaptive activation function | sin | sin |

not change its shape and size while moving. For more details, see [25,26]. The one-dimensional KdV equation is given by

$$u_t + uu_x = \nu u_{xxx}, \qquad x \in [-1,\ 1],\ t > 0 \tag{8}$$

where $\nu = 0.0025$. The initial condition is $u(0, x) = cos(\pi x)$ and boundary conditions are periodic.

The training data set is obtained by a high-resolution numerical method, where the conventional Fourier spectral method is employed for space discretization, whereas a fourth-order Runge–Kutta method is used for temporal discretization. For comparison of PINN and cPINN algorithms, we used 18 000 residual points, sin activation function and 10 hidden-layers with 20 neurons in each layer. In case of cPINN, the domain is divided into two sub-domains. In sub-domain 1 (bottom), we used 4000 residual points and in sub-domain 2 (top), we used 14 000 residual points. Thus, the total number of residual points are the same in both cases. Table 3 gives the cPINN architecture for the KdV equation. Fig. 6 shows the exact solution (top), predicted solution as well as point-wise error by PINN (middle row) and by cPINN (bottom row). In cPINN contour plots, the horizontal white line shows the interface location at $x = -0.74$, which divides the domain into two-subdomains. The cPINN solution accurately captures all the dispersive waves in the solution and the point-wise error is very low in the entire domain, whereas PINN fails to predict the solution and the point-wise error is very high compared to cPINN. Fig. 7 shows the comparison of solution of PINN and cPINN at various $t$ locations, where the accuracy of cPINN over PINN is clearly visible. Finally, Fig. 8 gives the loss history over the number of iterations (in particular, 300 000 iteration in both cases) for cPINN (left) and for PINN (right), where one can see that the loss is decreasing in both cases.

For more fair comparison, we keep the number of parameters (weights and biases) same in both PINN and cPINN methods by increasing the network depth in PINN but, this does not improve the solution of PINN further. One of the reason for the better performance of cPINN algorithm is the additional information provided by the algorithm in terms of interface condition like continuity of fluxes inside the domain. Moreover, domain decomposition approach and the deployment of a separate network in each sub-domain increases the representation capacity of the network.

### *4.3. Two-dimensional Burgers equation*

The two-dimensional viscous Burgers equation in conservation form is given by

$$u_t + f(u)_x + g(u)_y = \nu \left(u_{xx} + u_{yy}\right), \quad (x, y) \in [0,\ 1] \otimes [0,\ 1] \ \text{ and } t > 0, \tag{9}$$

where the inviscid fluxes in $x$ and $y$ directions are $f(u) = u^2/2, g(u) = u^2/2$, respectively and viscosity coefficient $\nu = 0.004$. The initial and boundary conditions are obtained from the exact solution $u(x, y, t) = 1/(1 + exp[(x + y - t)/(2\nu)])$. We divide the domain into 12 sub-domains, where a PINN is applied in each domain. Flux continuity and the average solution given by two different networks are enforced at the common interface.

The location of interface along the $x$ and $y$ axis are $[0.4,\ 0.8]$ and $[0.3,\ 0.6,\ 0.9]$, respectively and the number of interface points on each interface is 80. The adaptive activation function is *tanh* and the learning rate is 0.0008. The boundary/initial, residual and interface weights are chosen as $W_{u_p} = 20$, $W_{\mathcal{F}_p} = 1.0$ and $W_{I_p} = 20$ respectively. Table 4 shows the neural network architecture in each sub-domain. Fig. 9 shows the sub-domain numbering, exact and predicted contour plots of the solution along with point-wise absolute error in the whole computational domain (left to right). The maximum error is 1.7e−2 near the high gradient region. We observed that the large values of error occurs near high gradient region, near the interfaces. This can be mitigated by taking a sufficiently large number of interface and residual points. Fig. 10 shows the comparison of exact and predicted solution at different

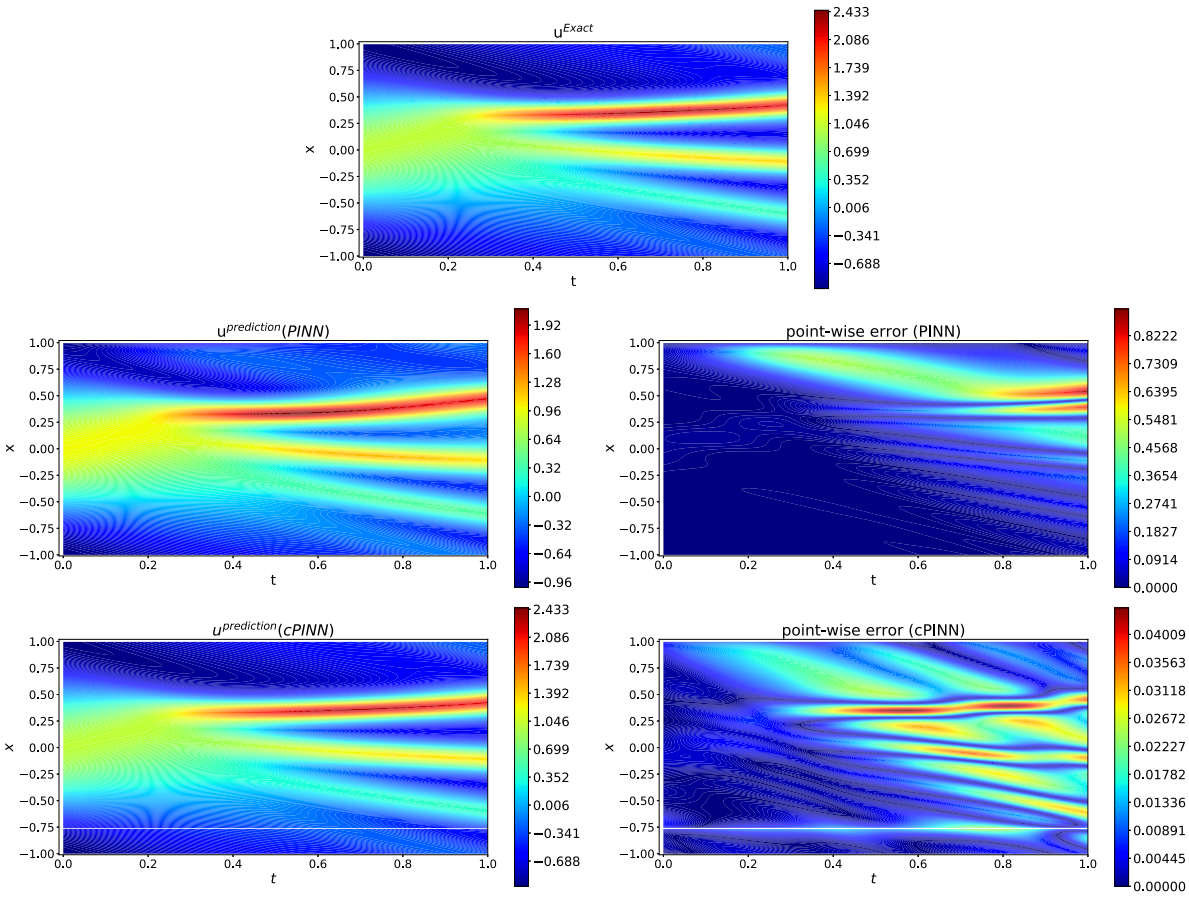


**Fig. 6.** KdV equation: Exact solution (top), and predicted solution as well as point-wise error by PINN (middle row) and by cPINN (bottom row).

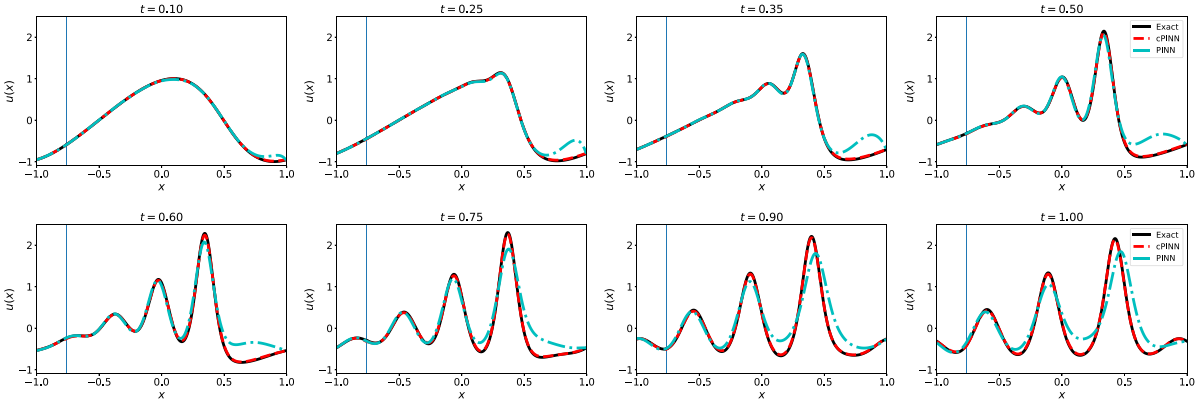


**Fig. 7.** KdV equation: comparison of exact, PINN and cPINN solutions at various $t$ locations. Exact solution is given in black continuous line, cPINN solution is in red colour (dash–dash line) and PINN solution is in cyan colour (dash–dot line). The vertical blue line represents the location of interface for cPINN.

$x$ and $y$ locations. cPINN accurately predicts the solution in all the regions of the domain. Finally, Fig. 11 shows the loss versus the number of iterations for all the 12 sub-PINNs for 50 000 iterations.

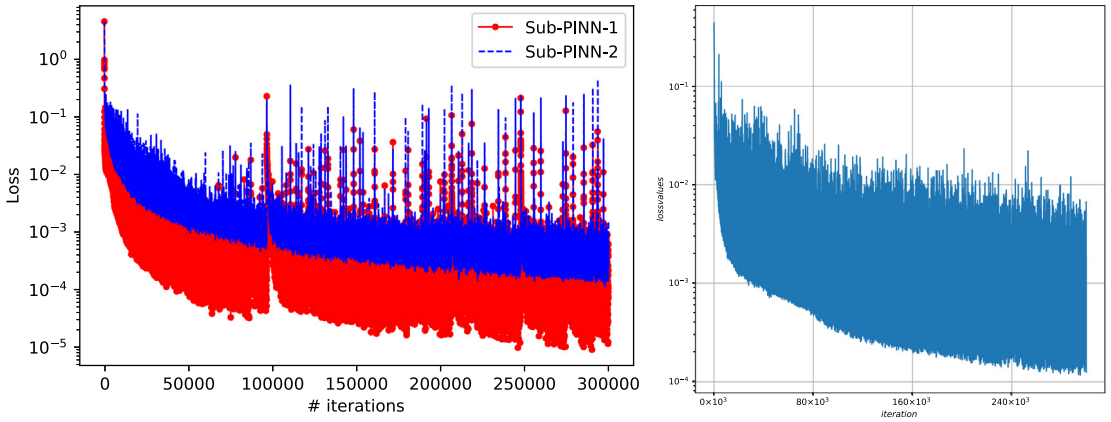


**Fig. 8.** KdV equation: (Left) cPINN and (right) PINN loss history with 300 000 number of iterations.

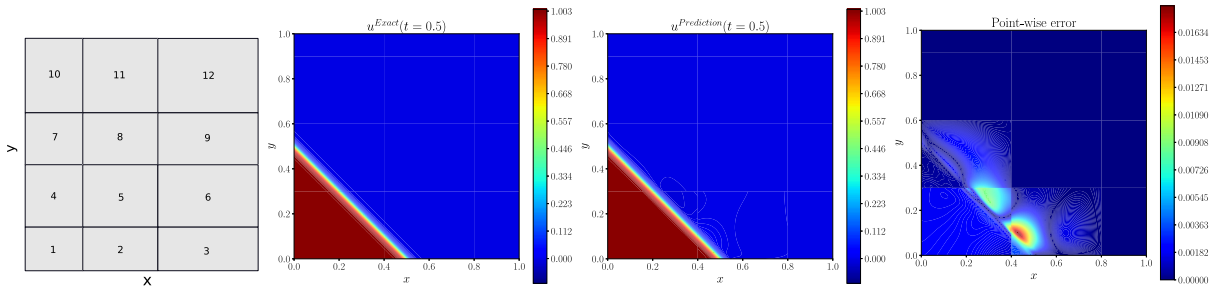


**Fig. 9.** (Left to right) 12 sub-domains numbering, exact solution, cPINN solution, and point-wise error of two-dimensional Burgers equation.

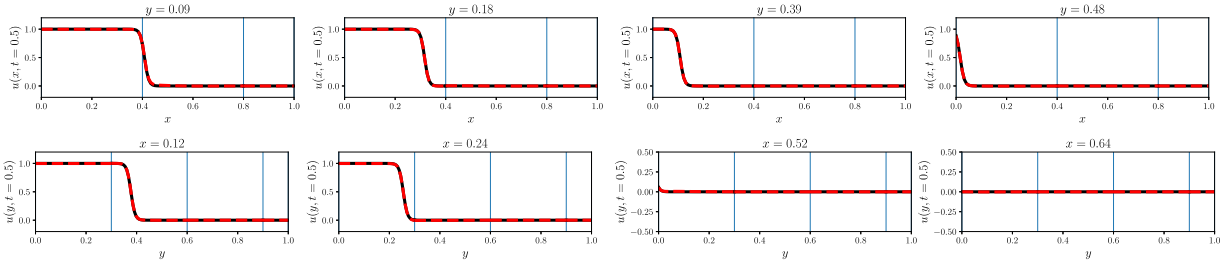


**Fig. 10.** Exact (black solid line) and predicted (red dashed line) solution comparison at various $x$ and $y$ locations. The vertical blue lines show the location of interface positions.

**Table 4**
Neural network architecture in each sub-domains for two-dimensional Burgers equation.

| Domain number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # Layers | 6 | 6 | 3 | 6 | 6 | 3 | 6 | 3 | 3 | 3 | 3 | 3 |
| # Neurons | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| # Residual points | 3000 | 3000 | 1000 | 3000 | 1000 | 200 | 400 | 200 | 200 | 200 | 200 | 200 |

Next, we consider irregular sub-domains by dividing the domain into three different parts as shown in Fig. 12. Both residual points (left) as well as training data points (right) are shown in three different sub-domains. The interface points are shown in green, see right figure. The number of interface points on interface one (between

**Table 5**
Neural network architecture in three sub-domains for the two-dimensional Burgers equation.

| Domain number | 1 | 2 | 3 |
|---|---|---|---|
| # Layers | 3 | 6 | 3 |
| # Neurons | 20 | 20 | 20 |
| # Residual points | 500 | 10 000 | 1000 |
| Adaptive activation function | tanh | sin | cos |

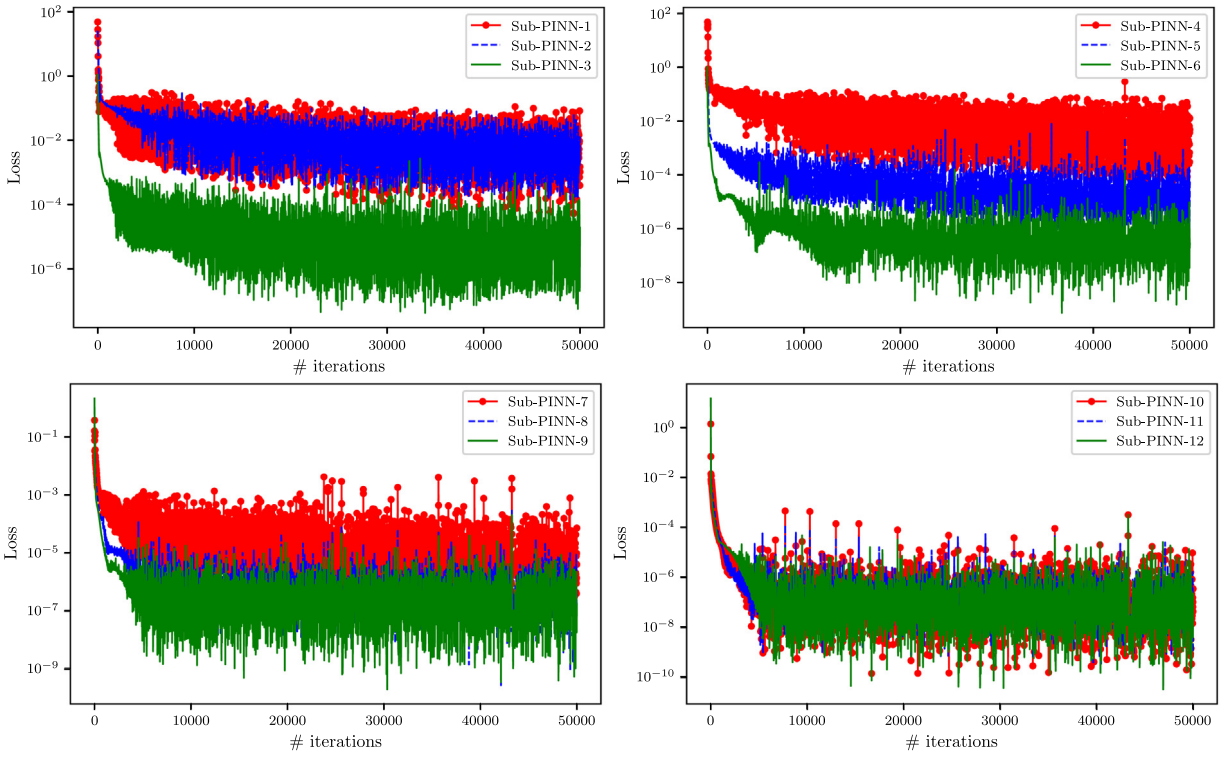


**Fig. 11.** Loss versus number of iterations for all 12 sub-domains.

region 1 and 2) is 102 points, whereas on interface two (between region 2 and 3) 89 points are used. Table 5 shows the neural network architecture in these three sub-domains. Fig. 13 shows the exact and predicted solution along with the point-wise error. The predicted solution matches well with the exact solution; also, the maximum point-wise error is of the order of 1e−02, which is less than 12 sub-domain case. We can also observe the large value of error near the second interface, which is due to the small number of interface points. Such behaviour can be mitigated by using adequately large number of interface points. Finally, Fig. 14 shows the loss function variation for the three different sub-domains for 40 000 iterations.

### *4.4. Two-dimensional coupled viscous Burgers equation*

The coupled viscous Burgers equation is an appropriate form of incompressible Navier–Stokes equation, which has exact solution. It contains similar advection and diffusion terms as in incompressible Navier–Stokes equations. This model governs various physical flows like shock wave propagation, vorticity transport, hydrodynamic turbulence *etc*., see [27,28] for details. Thus, it is natural to verify new computational methods for fluid problems

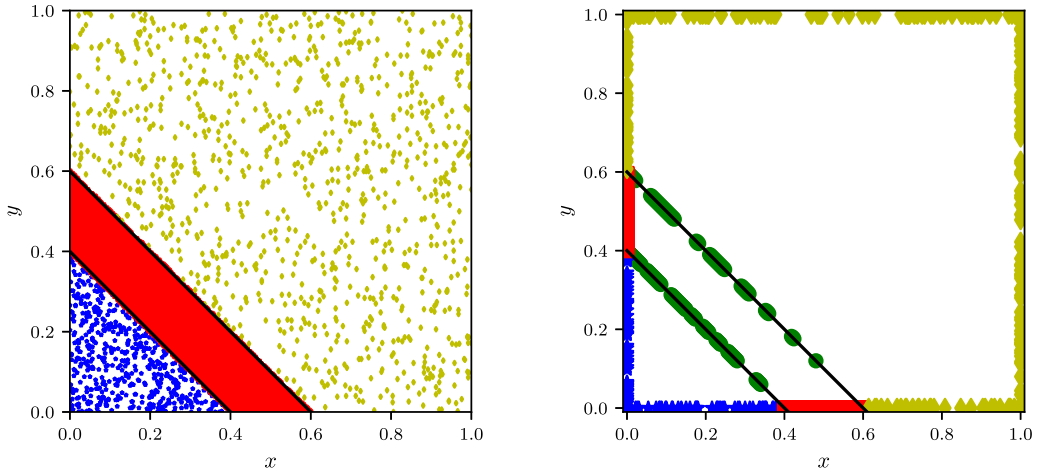


**Fig. 12.** Residual points (left) and the corresponding training data points (right) in the three regions. Residual points in region 1 are blue asterisks $*$, in region 2 are red squares and in region 3 are yellow diamonds. Interface points are shown in green colour (right). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

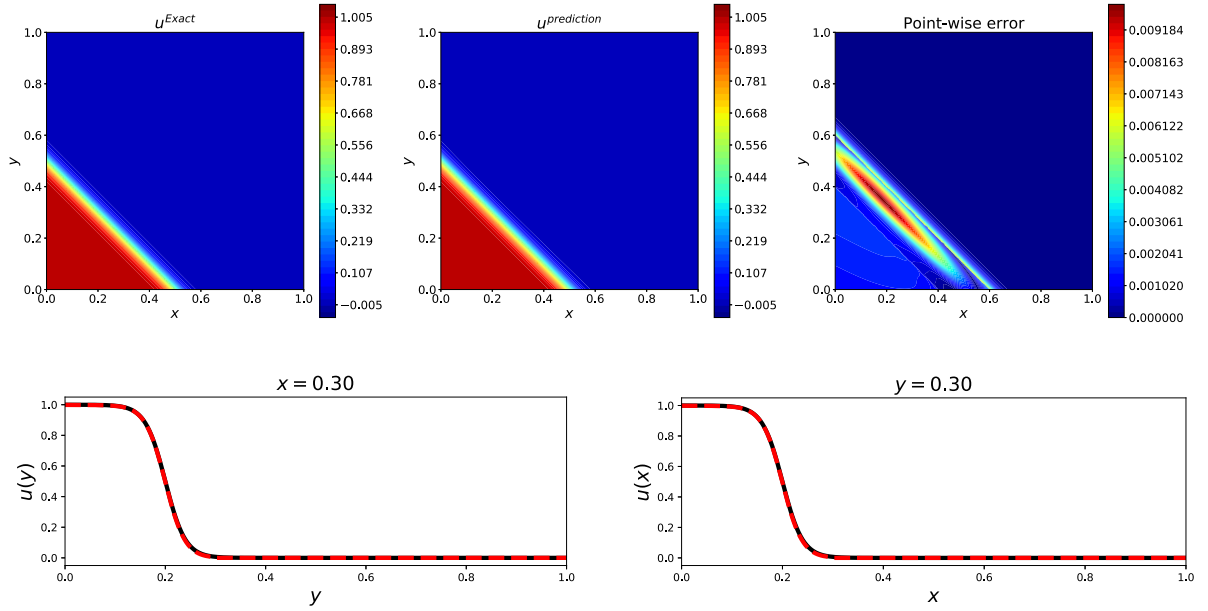


**Fig. 13.** Two-dimensional Burgers equation: First row shows the exact and cPINN solutions, and point-wise error, whereas the second row shows solution slices at $x = y = 0.3$ locations.

using such equations. The two-dimensional coupled viscous Burgers equation is given by

$$u_t + uu_x + vu_y = \nu\left(u_{xx} + u_{yy}\right),$$
$$v_t + uv_x + vv_y = \nu\left(v_{xx} + v_{yy}\right),$$

where $(x, y) \in [0,\ 1] \otimes [0,\ 1]$ and $t > 0$ and $\nu = 0.001$. The initial and boundary conditions are obtained from the exact solution

$$u(x, y, t) = \frac{3}{4} + \frac{1}{4(1 + exp[(-4x + 4y - t)/(32\nu)])},$$

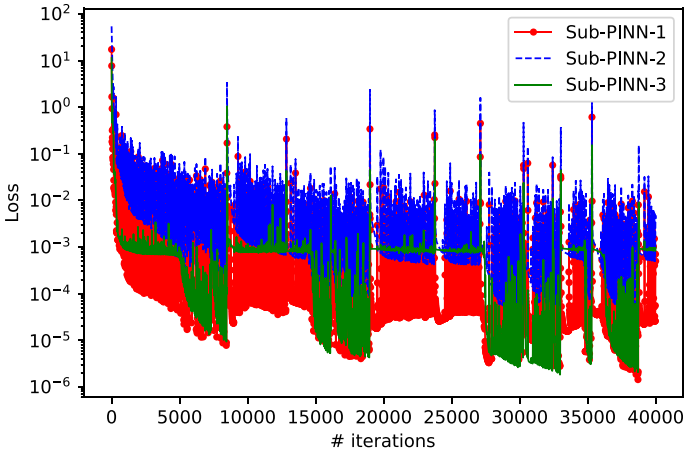


**Fig. 14.** Loss versus number of iterations for all 3 sub-domains.

**Table 6**
Neural network architecture in each sub-domains for two-dimensional vector Burgers equation.

| Domain number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # Layers | 6 | 4 | 2 | 6 | 6 | 2 | 4 | 6 | 4 | 2 | 6 | 6 |
| # Neurons | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| # Residual points | 1000 | 600 | 400 | 1000 | 1000 | 600 | 400 | 1000 | 600 | 400 | 1000 | 1000 |

$$v(x, y, t) = \frac{3}{4} - \frac{1}{4(1 + exp[(-4x + 4y - t)/(32v)])}.$$

We divide the domain into 12 sub-domains, where PINN is applied in each sub-domain. For both equations, the flux continuity and average solution given by two different networks are enforced at the interfaces.

The residual terms are obtained as

$$\mathcal{F}_1 := u_t + uu_x + vu_y - \nu\left(u_{xx} + u_{yy}\right),$$
$$\mathcal{F}_2 := v_t + uv_x + vv_y - \nu\left(v_{xx} + v_{yy}\right),$$

and the MSE for residual term in the $p$th sub-domain becomes

$$\text{MSE}_{\mathcal{F}_p} = \frac{1}{N_{F_p}} \sum_{i=1}^{N_{F_p}} \left( \left| \mathcal{F}_1(\mathbf{x}^i_{F_p}) \right|^2 + \left| \mathcal{F}_2(\mathbf{x}^i_{F_p}) \right|^2 \right),$$

The location of interface along the $x$ and $y$ axis are [0.33, 0.66] and [0.25, 0.5, 0.75], respectively and the number of interface points on each interface is 80. The boundary/initial, residual and interface weights are chosen as $W_{u_p} = 20$, $W_{\mathcal{F}_p} = 1.0$ and $W_{I_p} = 20$ respectively. Again, the adaptive activation function is *tanh* and the learning rate is 0.0008. Table 6 gives the neural network architecture in all 12 sub-domains. Fig. 15 shows the exact and predicted contour plot of the solution $u$ and $v$ along with point-wise absolute error. The maximum error is 1.4e−2, which is near the high gradient region. Despite being cutting across several sub-domains, the cPINN is able to capture the narrow high gradient region in the solutions. The chessboard type of pattern observed in the error plots shows that the loss function in each sub-domain converges to a different minimum, resulting in a slight mismatch in the solution along the interface line. Such mismatch can be reduced by choosing adequate number of interface points, which results in smooth stitching of sub-domains. Another source of mismatch in the solution may also come from different approximation error (due to different network architecture) and different generalization error (due to different number/locations of residual points) in each sub-domain. In such cases, the mismatch in the solution becomes inevitable.

Fig. 16 shows the comparison of exact and predicted solution at different $x$ and $y$ locations, where exact $u$ (black solid line), exact $v$ (black dash-dot line), predicted $u$ (red dashed line), and predicted $v$ (cyan dotted line) solutions

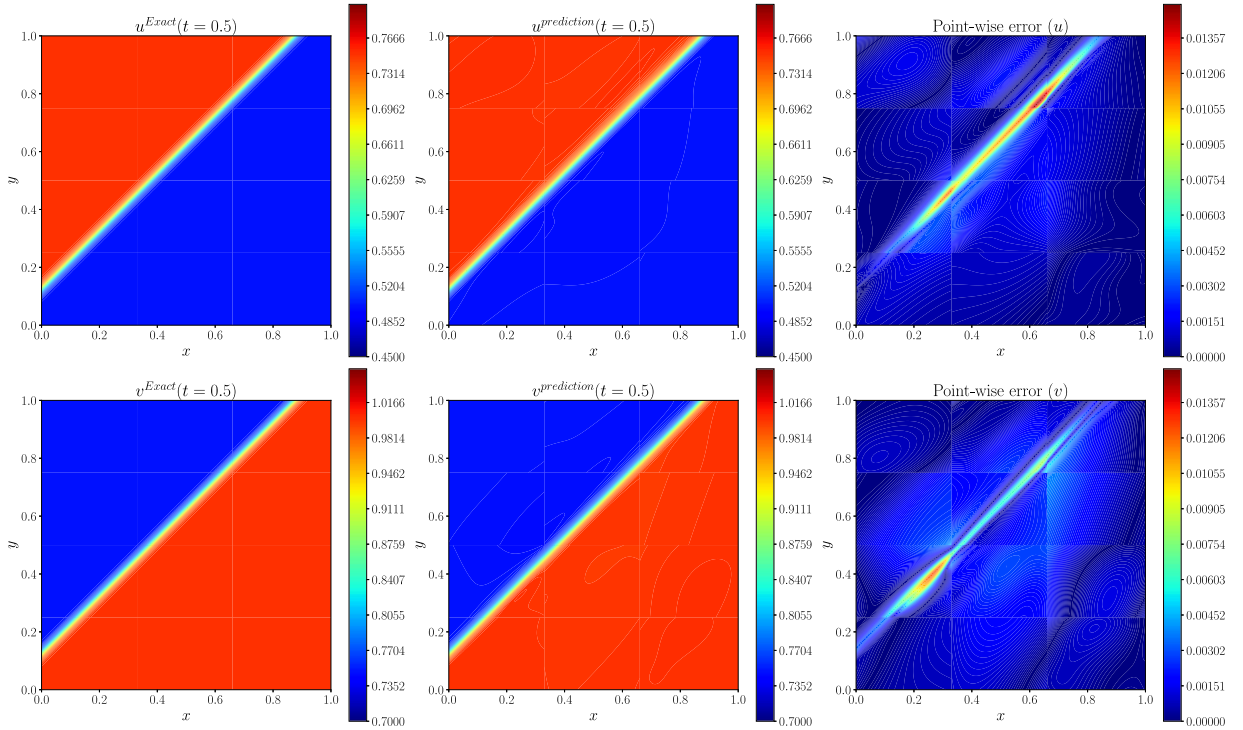


**Fig. 15.** Exact solution (first column), cPINN solution (second column) and point-wise error (third column) for the two-dimensional coupled Burgers equation.

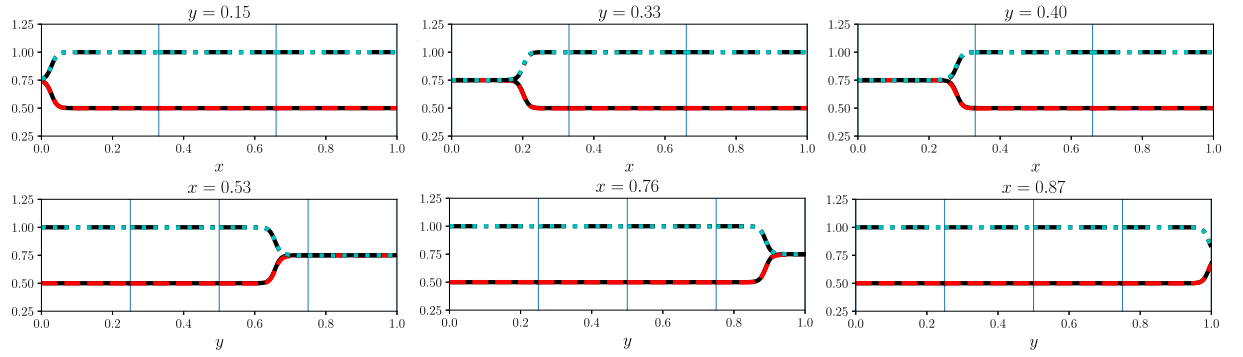


**Fig. 16.** Exact $u$ (black solid line), exact $v$ (black dash-dot line) and predicted $u$ (red dashed line), predicted $v$ (cyan dotted line) solution comparison at various $x$ and $y$ locations. The vertical blue lines show the location of interface positions.

are plotted. cPINN accurately predicts the solution in all the regions of the domain. Finally, Fig. 17 shows the loss versus number of iterations for all the 12 sub-PINNs.

### *4.5. Incompressible Navier–Stokes equations*

The lid-driven cavity is a standard test case for verifying the accuracy of new computational methods for incompressible Navier–Stokes equations. Although, there are many papers in the literature that present results of lid-driven cavity with different formulations, grids and numerical methods, we shall compare the results with Ghia et al. [29], where they used a finite-difference formulation of the stream function-vorticity formulation using uniform

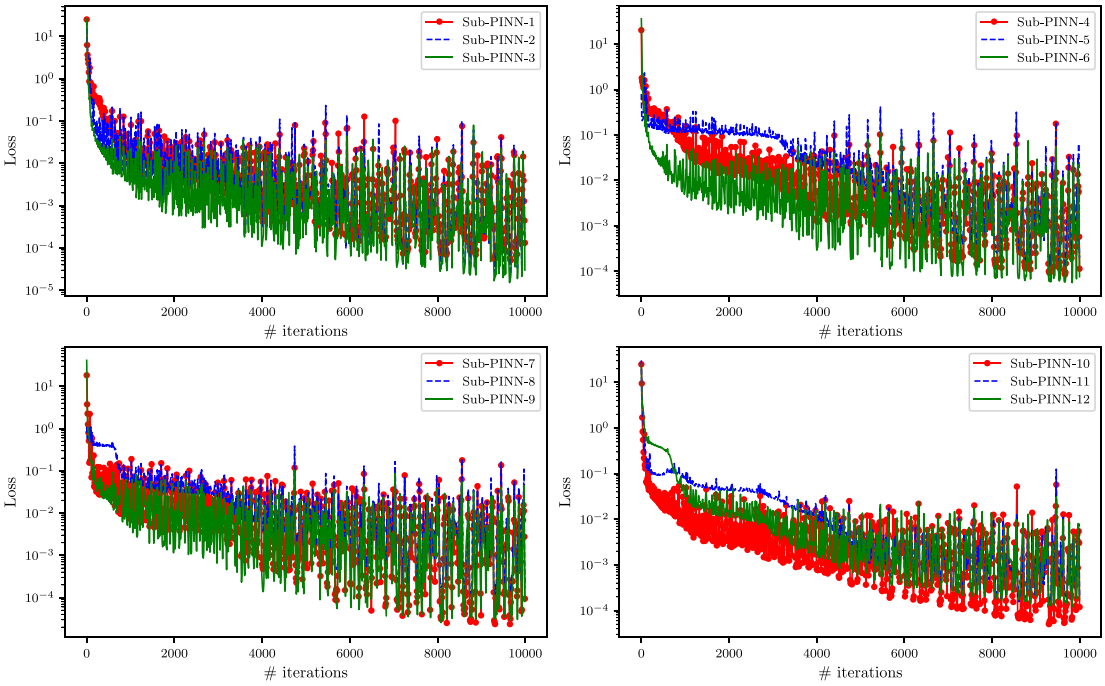


**Fig. 17.** Loss versus number of iterations for all 12 sub-domains.

**Table 7**
Neural network architecture in each sub-domains for the two-dimensional incompressible Navier–Stokes equations.

| Domain number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # Layers | 6 | 6 | 6 | 6 | 4 | 6 | 6 | 4 | 6 | 6 | 6 | 6 |
| # Neurons | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| # Residual points | 2500 | 5000 | 2500 | 2500 | 5000 | 2500 | 2500 | 5000 | 2500 | 2500 | 5000 | 2500 |

grids. The domain is $[0,\ 1]\otimes[0,\ 1]$ and no-slip boundary conditions are applied at left, bottom and right boundaries. The top boundary is moving with constant velocity in the positive $x$ direction. The governing two-dimensional incompressible Navier–Stokes equations are given by

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

with appropriate Dirichlet and Neumann boundary conditions. $\mathbf{u} = \{u, v\}$, $p$ are velocity components in the $x$, $y$ directions and pressure respectively, and $Re$ is the Reynolds number.

Table 7 shows the neural network architecture used for the simulation in each sub-domain. The adaptive activation function used in the simulation is *tanh* and the learning rate is 0.0006. The location of the interface along the $x$ and $y$ axis are $[0.2,\ 0.8]$ and $[0.2,\ 0.45,\ 0.8]$, respectively and the number of interface points on each interface is 100. Fig. 18 shows the schematic of the lid-driven cavity test case with boundary conditions (top left). Contour plots of velocities $u$ and $v$ (top middle and right, respectively) for $Re$ = 100 are shown. Velocity slices are compared at locations $x = y = 0.5$, which are in good agreement with the results given in [29]. Finally, the loss function variations in all 12 sub-domains are shown in Fig. 19, up to 450 000 iterations.

### *4.6. Compressible Euler equations*

The compressible Euler equations are hyperbolic conservation laws, which can admit discontinuous solutions like shock and contact waves even when the initial conditions are sufficiently smooth. Such equations govern a

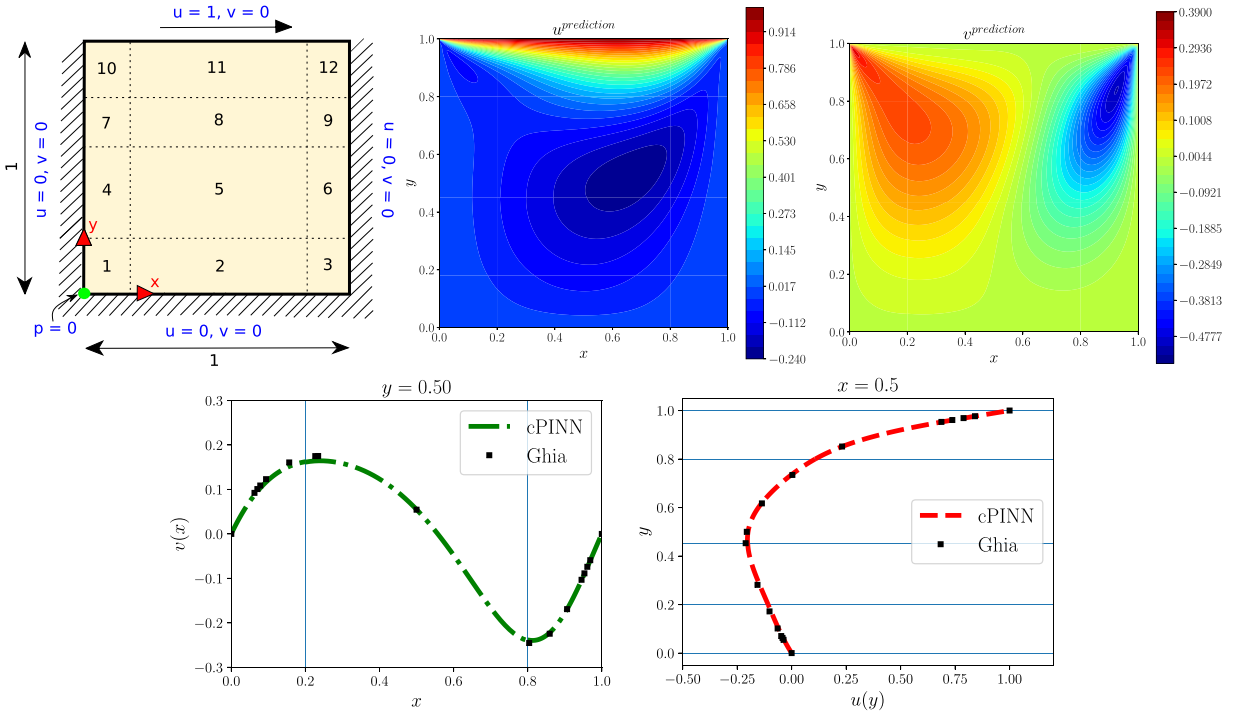


**Fig. 18.** Schematic of lid-driven cavity test case with boundary conditions (top left). $Re = 100$: Contour plots of velocities $u$ and $v$ (top middle and right), respectively. Finally, velocity slices of cPINN are compared with the standard results given by Ghia et al. [29], at locations $x = y = 0.5$.

wide range of high speed fluid flows such as transonic, supersonic and hypersonic flows. Over the past few decades various numerical methods have been developed for hyperbolic conservation laws, which are tailored to capture such discontinuities. PINN [11] is shown to capture such discontinuous solution exactly for one-dimensional Euler equations, which is otherwise a difficult task for existing numerical methods. In this section, we shall solve Euler equations using cPINN, demonstrating the capability of the proposed method.

The governing compressible Euler equations in two dimensions are given as

$$U_t + (G_1(U))_x + (G_2(U))_y = 0, \quad (\mathbf{x}, t) \in \Omega \times (0, T] \subset \mathbb{R}^2 \times \mathbb{R}_+ \tag{10}$$

with appropriate initial and boundary conditions. The fluxes $G_i(U), i = 1, 2$ are functions of the conserved variable $U$. For the two-dimensional Euler equations, $U$ and $G_i$'s are given as

$$U = \begin{pmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho E \end{pmatrix}, \quad G_i = \begin{pmatrix} \rho u_i \\ \delta_{i1} p + \rho u_1 u_i \\ \delta_{i2} p + \rho u_2 u_i \\ p u_i + \rho u_i E \end{pmatrix}, \quad i = 1, 2$$

where $\rho, u_1, u_2, E, p$ are density, velocity components in the $x$ and $y$ directions, total energy and pressure respectively and $\delta_{ij}$ is Kronecker delta. The total energy is given by

$$E = \frac{p}{\rho(\gamma - 1)} + \frac{1}{2}\|\mathbf{u}\|^2_{\mathcal{L}_2}$$

The eigenvalues of flux Jacobian matrices $\frac{\partial G_i}{\partial U}$, $i = 1, 2$ are

$$u_i \pm a, \; u_i, \; u_i$$

where $a = \sqrt{\gamma p/\rho}$ is the speed of sound.

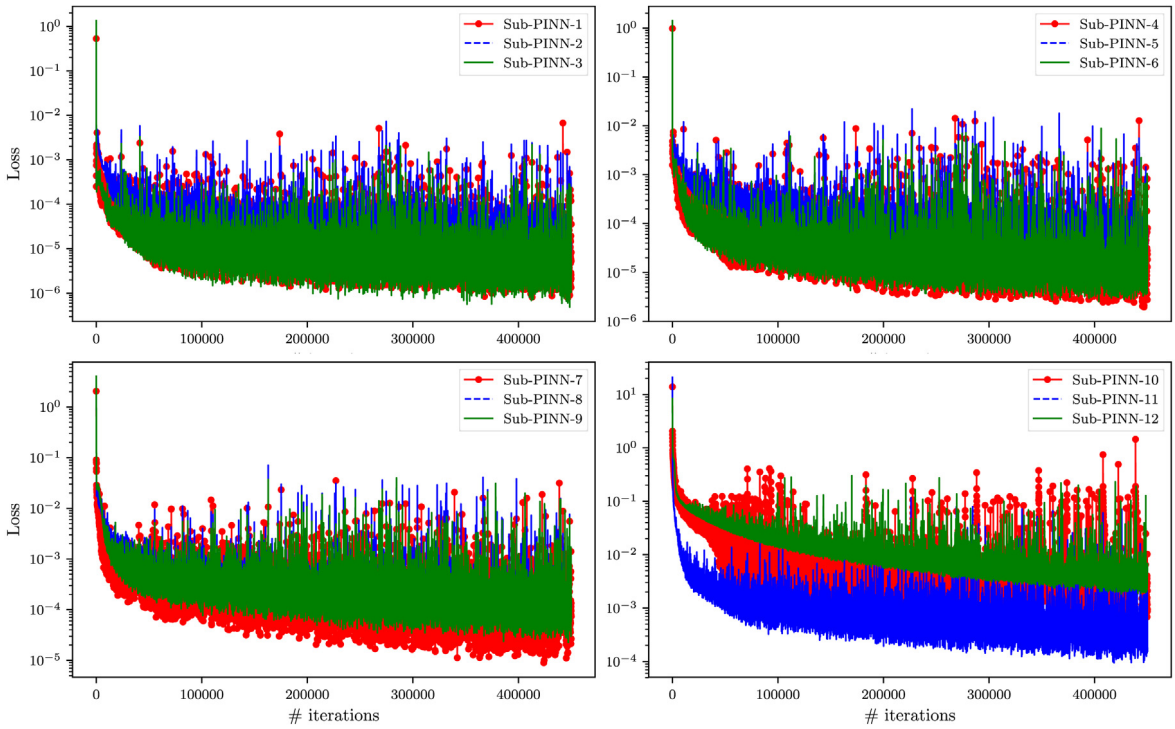


**Fig. 19.** Loss function variation in various sub-domains for the lid-driven cavity test case.

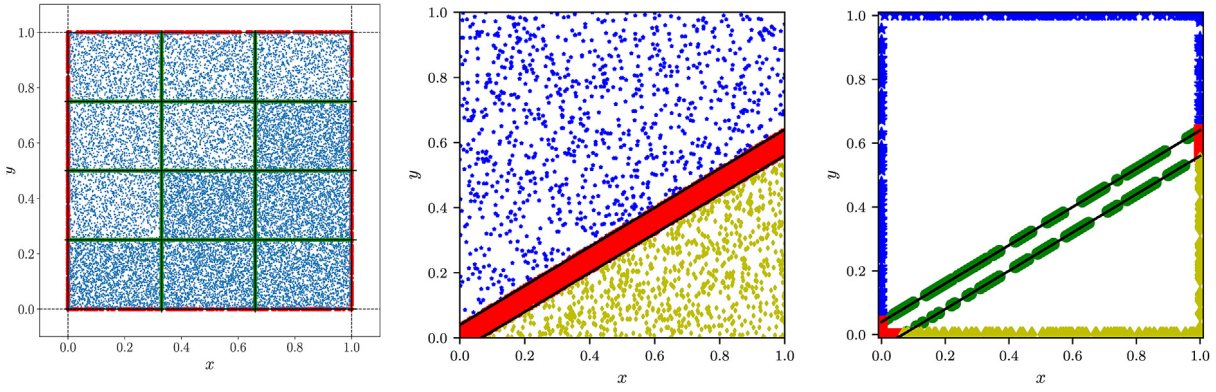


**Fig. 20.** 12 sub-domains (left) where boundary training points are red, interface points are green and residual points are blue. 3 sub-domains (middle and right) show the residual and boundary training data points in blue (*), red (square) and yellow (diamond) colours in region 1, 2 and 3, respectively. Green circles are the interface points on two inclined interfaces. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

We shall consider the oblique shock wave problem, where the computational domain is $[0,\ 1]^2$. An inviscid Mach 2 flow is at an angle of -10° with respect to the horizontal wall which generates an oblique shock with shock angle 29.3° with the wall. Dirichlet boundary conditions are applied at the left and top boundaries, whereas all primitive variables are extrapolated at the right outgoing boundary. Wall boundary conditions are prescribed at the bottom boundary. The exact solution is given by

$$(M,\ \rho,\ u_1,\ u_2,\ p) = \begin{cases} (2.0,\ 1.0,\ \cos 10^\circ,\ -\sin 10^\circ,\ 0.17857) & \text{before shock} \\ (1.6405,\ 1.4584,\ 0.8873,\ 0.0,\ 0.3047) & \text{after shock.} \end{cases} \tag{11}$$

**Table 8**
Neural network architecture in 12 sub-domains for the two-dimensional compressible Euler equations.

| Domain number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # Layers | 6 | 6 | 6 | 4 | 6 | 6 | 4 | 4 | 6 | 4 | 4 | 4 |
| # Neurons | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| # Residual points | 2200 | 2200 | 2200 | 1200 | 2200 | 2200 | 1200 | 1200 | 2200 | 1200 | 1200 | 1200 |

The $p$th sub-domain loss function is given by

$$\begin{aligned}\mathcal{L}(\tilde{\boldsymbol{\Theta}}_p) =& W_{\mathcal{F}_p} \times (\mathrm{MSE}_G^{Mass} + \mathrm{MSE}_G^{MomX} + \mathrm{MSE}_G^{MomY} + \mathrm{MSE}_G^{Ene}) \\ &+ W_{u_p} \times (\mathrm{MSE}_\rho^D + \mathrm{MSE}_p^D + \mathrm{MSE}_{u_1}^D + \mathrm{MSE}_{u_2}^D + \mathrm{MSE}_\rho^N + \mathrm{MSE}_p^N + \mathrm{MSE}_{u_1}^N + \mathrm{MSE}_{u_2}^N) \\ &+ W_{I_p} \times \text{Interface conditions},\end{aligned} \tag{12}$$

where the MSE for mass, momentum ($x$ and $y$ directions), and energy conservation laws are given as

$$\mathrm{MSE}_G^{Mass} = \frac{1}{N_{F_p}} \sum_{i=1}^{N_{F_p}} |(G_1^{Mass}(\mathbf{x}_{F_p}^i))_x + (G_2^{Mass}(\mathbf{x}_{F_p}^i))_y|^2,$$

$$\mathrm{MSE}_G^{MomX} = \frac{1}{N_{F_p}} \sum_{i=1}^{N_{F_p}} |(G_1^{MomX}(\mathbf{x}_{F_p}^i))_x + (G_2^{MomX}(\mathbf{x}_{F_p}^i))_y|^2,$$

$$\mathrm{MSE}_G^{MomY} = \frac{1}{N_{F_p}} \sum_{i=1}^{N_{F_p}} |(G_1^{MomY}(\mathbf{x}_{F_p}^i))_x + (G_2^{MomY}(\mathbf{x}_{F_p}^i))_y|^2,$$

$$\mathrm{MSE}_G^{Ene} = \frac{1}{N_{F_p}} \sum_{i=1}^{N_{F_p}} |(G_1^{Ene}(\mathbf{x}_{F_p}^i))_x + (G_2^{Ene}(\mathbf{x}_{F_p}^i))_y|^2,$$

whereas MSE for Dirichlet and Neumann boundary conditions are given as

$$\mathrm{MSE}_\rho^D = \frac{1}{N_{u_p}} \sum_{i=1}^{N_{u_p}} |\rho^i - \rho(\mathbf{x}_{u_p}^i)|^2; \qquad \mathrm{MSE}_\rho^N = \frac{1}{N_{u_p}} \sum_{i=1}^{N_{u_p}} |\nabla\rho(\mathbf{x}_{u_p}^i)|^2,$$

$$\mathrm{MSE}_p^D = \frac{1}{N_{u_p}} \sum_{i=1}^{N_{u_p}} |p^i - p(\mathbf{x}_{u_p}^i)|^2; \qquad \mathrm{MSE}_p^N = \frac{1}{N_{u_p}} \sum_{i=1}^{N_{u_p}} |\nabla p(\mathbf{x}_{u_p}^i)|^2,$$

$$\mathrm{MSE}_{u_1}^D = \frac{1}{N_{u_p}} \sum_{i=1}^{N_{u_p}} |u_1^i - u(\mathbf{x}_{u_p}^i)|^2; \qquad \mathrm{MSE}_{u_1}^N = \frac{1}{N_{u_p}} \sum_{i=1}^{N_{u_p}} |\nabla u(\mathbf{x}_{u_p}^i)|^2,$$

$$\mathrm{MSE}_{u_2}^D = \frac{1}{N_{u_p}} \sum_{i=1}^{N_{u_p}} |u_2^i - u(\mathbf{x}_{u_p}^i)|^2; \qquad \mathrm{MSE}_{u_2}^N = \frac{1}{N_{u_p}} \sum_{i=1}^{N_{u_p}} |\nabla u(\mathbf{x}_{u_p}^i)|^2.$$

The interface conditions include continuity of the mass, momentum and the energy fluxes as well as enforcing the average values of the primitive variables like density, velocities in $x$ and $y$ directions, and pressure along the interfaces. The weights for initial, residual and interface terms are taken as $W_{u_p} = 40$, $W_{\mathcal{F}_p} = 1$ and $W_{I_p} = 40$, respectively. Again, the activation function is *tanh* and the learning rate is 0.0008. Table 8 gives the architecture of the neural network in all 12 sub-domains.

For 12 sub-domains, the location of interface along the $x$ and $y$ axis are [0.33, 0.66] and [0.25, 0.5, 0.75], respectively and the number of interface points on each interface is 100. Fig. 20 shows the residual points (blue), interface points (green) and boundary points (red) randomly chosen over 12 subdomains as well as the clustered points randomly chosen over three subdomains. Table 9 shows the cPINN architecture for three inclined sub-domains, where a different neural network is used. Three different adaptive activation functions are used with

**Table 9**
Neural network architecture in three sub-domains for the two-dimensional compressible Euler equations.

| Domain number | 1 | 2 | 3 |
|---|---|---|---|
| # Layers | 2 | 6 | 2 |
| # Neurons | 20 | 25 | 20 |
| # Residual points | 2000 | 18 000 | 1500 |
| Adaptive activation function | sin | tanh | cos |

**Table 10**
Relative $L_2$ error in primitive variables using cPINN for 12 and 3 sub-domains (SD).

| | Density | Pressure | $u_1$ | $u_2$ |
|---|---|---|---|---|
| cPINN (12 SD) | 7.49323e−02 | 3.23695e−02 | 8.21709e−02 | 6.04430e−02 |
| cPINN (3 SD) | 8.34527e−03 | 7.05783e−03 | 9.02445e−03 | 8.13452e−03 |

**Table 11**
Neural network architectures used to solve the inverse problem of one-dimensional Burgers equation.

| Domain number | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| # Layers | 2 | 6 | 3 | 2 |
| # Neurons | 20 | 30 | 25 | 20 |
| Adaptive activation function | cos | sin | tanh | sin |

different number of hidden-layers and neurons in each layer. Fig. 21 gives the slices of density, $u_1$, $u_2$ and pressure for 12 sub-domains (top row) and for three inclined sub-domains (bottom row) at various $x$ locations. For clustered data points (3 sub-domain case), the cPINN captures the shock wave more accurately than randomly distributed data points (12 sub-domain case). Table 10 shows the relative $L_2$ error in density, $u_1$, $u_2$ and pressure over a complete domain for 12 and 3 sub-domains cases. Finally, Fig. 22 shows the loss value variation for three sub-domains, which is still decreasing after half million iterations.

### *4.7. Inverse problem: One-dimensional viscous Burgers equation*

In this section, we shall consider a problem of data-driven discovery of partial differential equations, which is an inverse problem. The parameterized viscous Burgers equation is written as

$$u_t + f(u, u_x; \lambda)_x = 0, \quad x \in \Omega \subset \mathbb{R}, \ t > 0,$$

where $u(x, t)$ is the hidden solution and $f(u, u_x; \lambda) = u^2/2 - \lambda u_x$ is the nonlinear parameterized flux with parameter $\lambda$. In particular, given sparse and potentially noisy observations of the solution $u(x, t)$, what is the parameter $\lambda$ that accurately describes the observed data?

In this example, $\lambda = \nu = 0.003184713$ is the viscosity coefficient. The computational domain is divided into 4 sub-domains as discussed in the forward problem. The architecture of the network is given in Table 11. Fig. 23 shows the number of training points in two different cases. The total number of training points in each sub-domain for the case 1 (left) and for the case 2 (right) are [770, 130, 270 390] and [3850, 650, 1350, 1950], respectively. The initial values of $\nu$ in all four sub-domains are $[1, \ 2, \ 3, \ 4]e{-}3$ for both cases. In all sub-domain losses, the continuity condition on diffusion coefficient $\nu$ is enforced along the interfaces. Fig. 24 shows the variations of $\nu$ history, $L_2$ error and MSE history over number of iterations, respectively (left to right), whereas top and bottom rows show case 1 and 2, respectively. From the history of $\nu$, we observed that by increasing the number of training points, the predicted $\nu$ converges faster towards its exact value. In the first case, the predicted $\nu = 0.003263753$ and in second case $\nu = 0.00321328$. Fig. 25 shows the converged value of $\nu = 0.003191094$ for case 2 after 250 000 iterations.

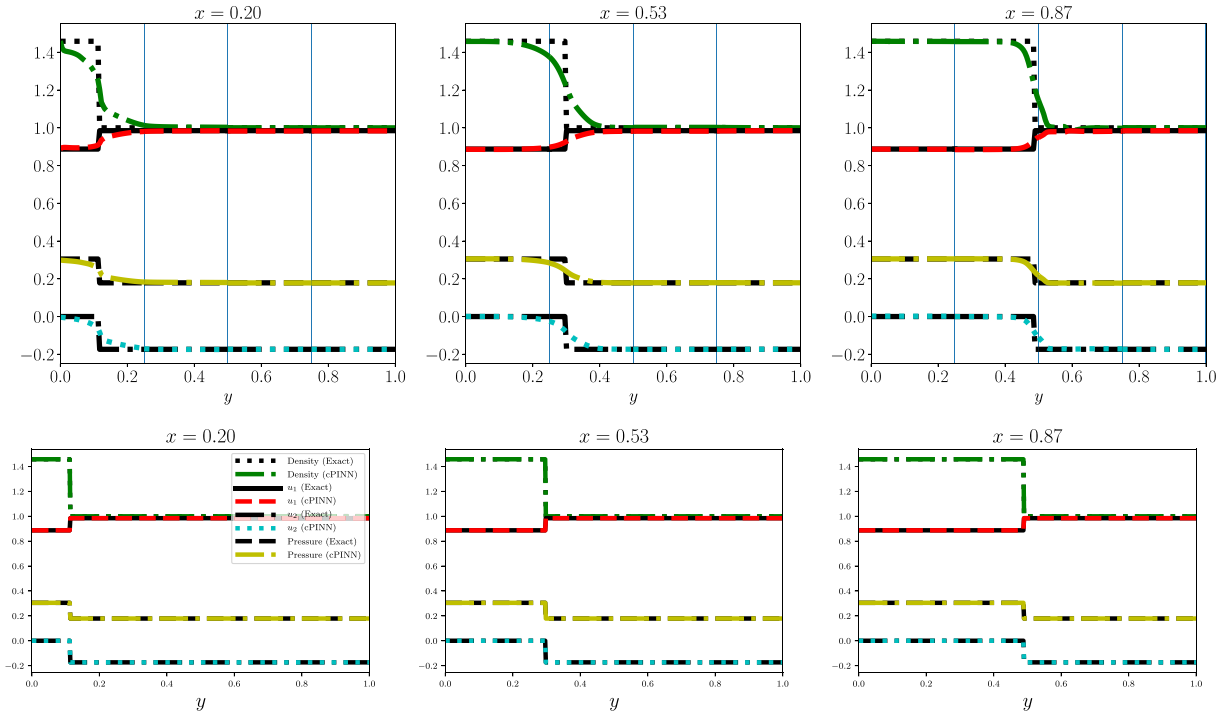


**Fig. 21.** Density (green), $u_1$ (red), $u_2$(cyan) and pressure (yellow) slices at $x = 0.2, 0.53$ and 0.87 locations. The black lines indicate the corresponding exact solutions. Top row shows solutions using 12 sub-domains, whereas bottom row gives solutions using 3 inclined sub-domains. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

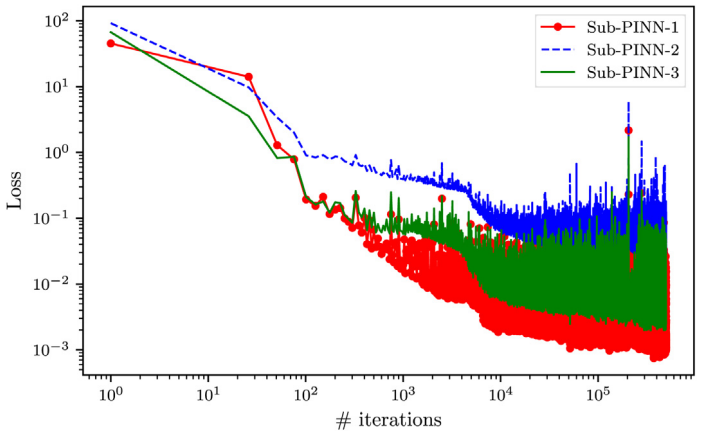


**Fig. 22.** Loss function vs number of iterations for three inclined sub-domains.

Next, we shall consider the generalization of this problem. Instead of assuming the exact differential equation, we are supplying the library of differential terms given in the nonlinear operator $\mathcal{N}$ of the following differential equation

$$u_t = \mathcal{N}\left(x, t, u, u^2, \ldots, u_x, uu_x, u^2u_x, \ldots, u_{xx}, uu_{xx}, \ldots\right)$$

where

$$\mathcal{N} = c_0 + c_1u + c_2u^2 + \cdots + c_ku_x + c_{k+1}uu_x + c_{k+2}u^2u_x + \cdots + c_{k+m}u_{xx} + c_{k+m+1}uu_{xx} + \cdots$$

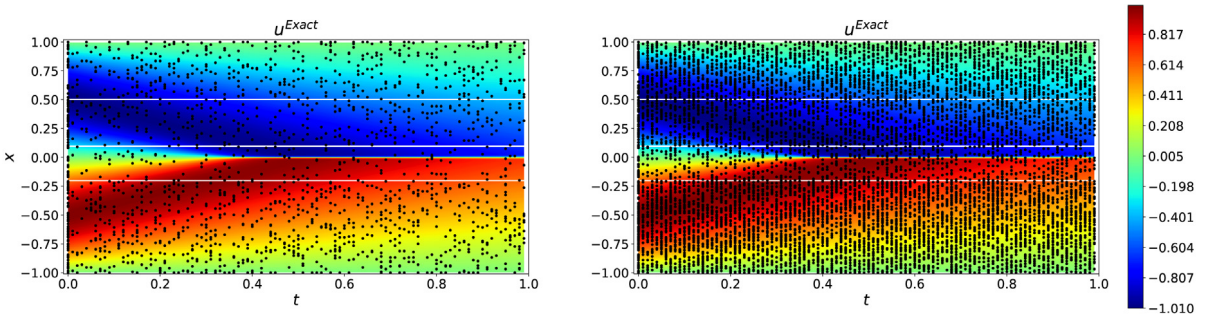


**Fig. 23.** Inverse problem: Black dot represents the training points in case 1 (left) and 2 (right).

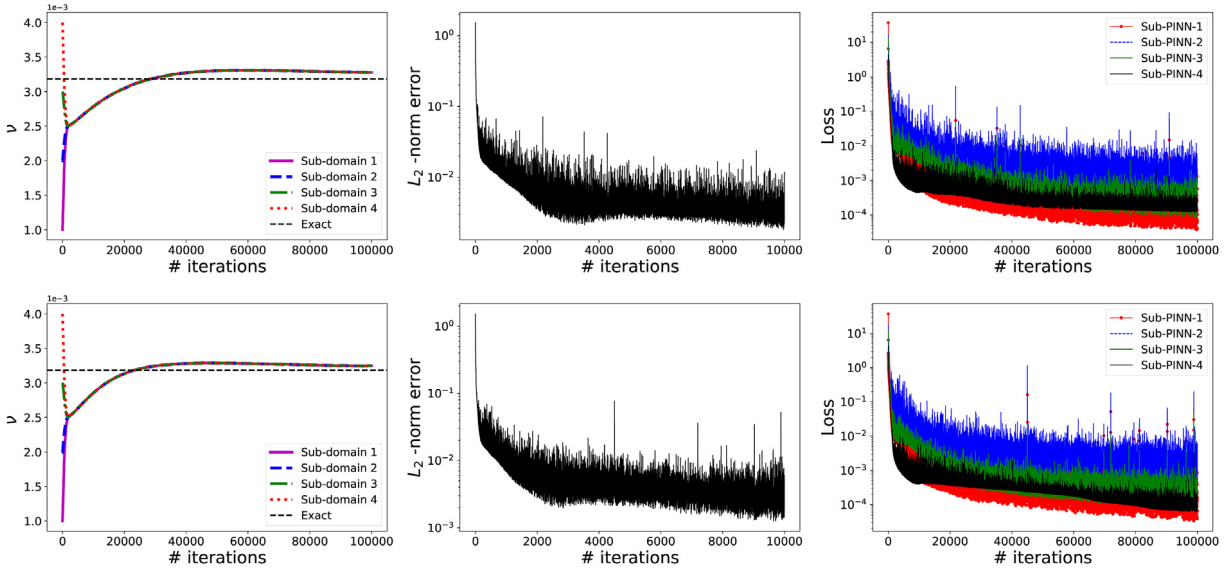


**Fig. 24.** Inverse problem: From left to right, $\nu$ history, $L_2$ error and MSE history over number of iterations, respectively. Top and bottom rows show case 1 and 2, respectively.

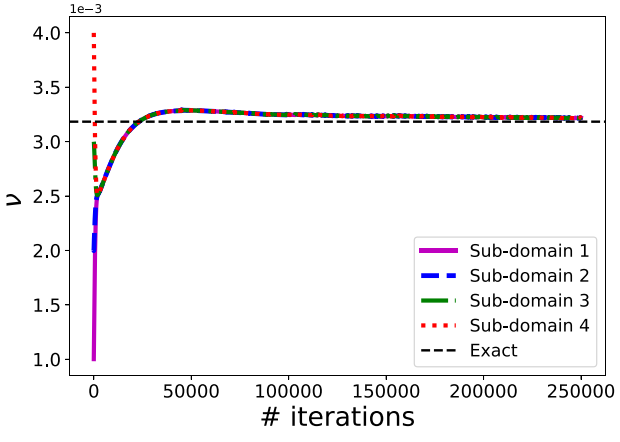


**Fig. 25.** Inverse problem: Convergence of $\nu$ towards its exact value over 250 000 iterations.

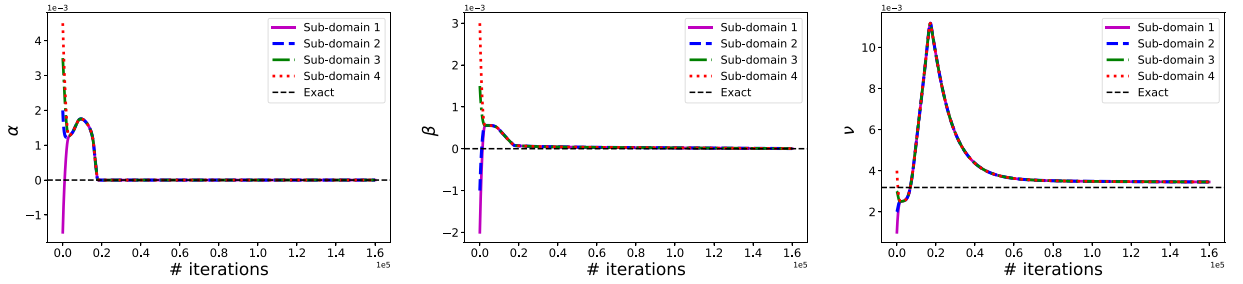


**Fig. 26.** Inverse problem: From left to right, variation of $\alpha, \beta, \nu$ with number of iterations for viscous Burgers equation.

Here, the network's job is to identify all the coefficients $c_i$'s present in the given differential equation. As discussed in [19], we could use sparse regression technique to determine the coefficients $c_i$'s. For viscous Burgers equation, we are supplying the additional dispersive term and the linear convection term, i.e.,

$$u_t + uu_x - \beta u_x - \nu u_{xx} + \alpha u_{xxx} = 0, \quad x \in \Omega \subset \mathbb{R}, \ t > 0,$$

and now, the aim is to identify all the terms $\alpha, \beta, \nu$, which represent wave speed, diffusion coefficient and dispersion coefficient, respectively in the given differential equation. The network architecture is the same as before and the total number of training points in each sub-domain is [770, 1800, 1700, 390].

Fig. 26 shows the variation of these terms with number of iterations. The initial values of $\alpha, \beta, \nu$ in all four sub-domains are $[-1.5,\ 2,\ 3.5,\ 4.5]e{-}3, [-2, -1,\ 1.5,\ 3]e{-}3, [1,\ 2,\ 3,\ 4]e{-}3$, respectively. In all sub-domain losses, the continuity condition on $\alpha, \beta, \nu$ is enforced along all the interfaces. After 20 000 iterations, both $\alpha, \beta$ converge to their exact value, which is zero and after that the viscosity $\nu$ started converging towards the exact value. After 160 000 iterations, the value of constants $\alpha = 1.4392e{-}6, \beta = 4.1053e{-}7, \nu = 0.003422561$. More accurate solution can be obtained by increasing the number of training data points.

### *4.8. Inverse problem: Two-dimensional inviscid Burgers equation*

The two-dimensional inviscid Burgers equation is given as

$$u_t + uu_x + u_y = 0, \quad x \in [-0.2,\ 1], y \in [0,\ 1], \text{ and } t > 0,$$

subject to boundary conditions

$$u(x, 0) = \begin{cases} a & \text{if } x < 0, \\ b & \text{otherwise,} \end{cases}$$

$u(-0.2, y) = a$ and $u(1, y) = b, \ \forall y$. The exact solution for the case of $a = 1$ and $b = -1$ is given in [30], which has a steady discontinuity at $x = 0$. In cPINN algorithm, instead of supplying the original equation, the following parameterized differential equation is supplied

$$u_t + uu_x + u_y = \nu(u_{xx} + u_{yy}),$$

and the job is to identify the value of the viscosity coefficient $\nu$.

The number of hidden layers in 12 sub-domains, starting from 1 to 12 are [6, 3, 3, 6, 3, 3, 6, 3, 3, 6, 3, 3] with 20 neurons in each layer, and the corresponding initial values of $\nu$ in each sub-domain are [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]/3. Fig. 27 shows the variation of $\nu$ in all 12 sub-domains versus number of iterations. Left figure is for clean data and right figure is for the data with 1% noise. The corresponding relative $L_2$ error in the whole domain for both cases is given in Table 12. The network correctly identifies the viscosity $\nu$ value in both the cases. Also, it can be observed that the presence of noise in the training data reduces the convergence of $\nu$ towards its exact value. Clean data takes approximately 9000 iterations whereas the data with 1% noise takes approximately 13 000 iterations for convergence.

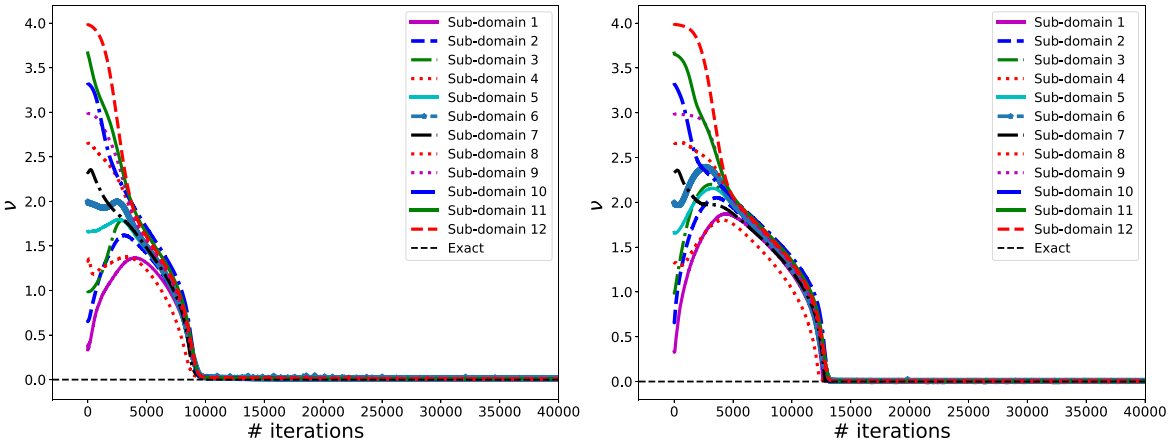


**Fig. 27.** Variation of $\nu$ in all 12 sub-domains versus number of iterations. Left figure is for the clean data and right figure is for the data with 1% noise.

**Table 12**
Relative $L_2$ error with clean data and data with 1% noise.

| | Clean data | 1% noise |
|---|---|---|
| Rel. $L_2$ error | 1.0248e−2 | 4.3812e−2 |

## 5. Summary

To summarize, a conservative physics-informed neural network (cPINN) is proposed on discrete domains for nonlinear conservation laws. This approach is an extension of the PINN, but has the flexibility of domain decomposition into several non-overlapping sub-domains. The conserved quantities like fluxes, are preserved by enforcing their continuity in the strong form at the common interfaces of neighbouring sub-domains. Various forms of errors involved in cPINN such as optimization, generalization and approximation errors and their sources are discussed. We examined our proposed method for solving both forward and inverse problems of high-dimensional nonlinear high order differential equations, including one- and two-dimensional Burgers equation, the KdV equations, the coupled Burgers equation, incompressible Navier– Stokes equations, and compressible Euler equations. In these cases, we showed that in addition to conservation of fluxes at interfaces, enforcing the average solution drastically increases the convergence rate; this additional constraint is important as it reduces the training cost while maintaining the accuracy of approximation. We discussed that cPINN belongs to a bigger class that we called *Mortar PINN*, where any interface conditions, and not necessarily the conservative quantities, corresponding to the problem at hand can be used for joining the decomposed domains. A key factor in the cPINN formulation is the flexibility of domain decomposition. In the case of the two-dimensional Burgers equation, we showed that by having a rough a priori knowledge of the location of shock, one can properly decompose the domain (even in an unstructured manner) to more accurately capture the steep descents in solution. Moreover, having separate networks in each sub-domain offers the flexibility of adopting a properly chosen network in the corresponding sub-domain, e.g. a deep network in a sub-domain with complex solution. Each sub-domain is an individual PINN that can have its own set of parameters, including width and depth of the network, activation function, optimization method, other training parameters, and penalizing points. This will further open up the possibility of formulating a meta-learning algorithm, which optimizes the overall structure and parameters of networks in cPINNs. Another major advantage of cPINNs in solving complex problems is its parallelization capacity, which can efficiently decrease the training cost. The implementation of parallel computation, however, was not carried in this paper, and we intend to systematically parallelize our method in future work.

## 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 the Department of Energy, USA PhILMs grant DE-SC0019453, the DARPA-AIRA, USA grant HR00111990025, and by AFOSR, USA Grant FA9550-17-1-0013.

## References

[1] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, et al., Deep neural networks for acoustic modeling in speech recognition, IEEE Signal Process. Mag. 29 (2012).
[2] Y. Wu, M. Schuster, Z. Chen, Q.V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., Google's neural machine translation system: Bridging the gap between human and machine translation, 2016, arXiv preprint arXiv:1609.08144.
[3] A. Krizhevsky, I. Sutskever, G. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[4] H. Owhadi, Bayesian numerical homogenization, Multiscale Model. Simul. 13 (2015) 812–828.
[5] M. Raissi, P. Perdikaris, G.E. Karniadakis, Inferring solutions of differential equations using noisy multi-fidelity data, J. Comput. Phys. 335 (2017) 736–746.
[6] M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, J. Comput. Phys. 348 (2017) 683–693.
[7] M. Raissi, G.E. Karniadakis, Hidden physics models: machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.
[8] M. Raissi, P. Perdikaris, G.E. Karniadakis, Numerical Gaussian processes for time-dependent and nonlinear partial differential equations, SIAM J. Sci. Comput. 40 (2018) A172–A198.
[9] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural network: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[10] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2018) 1–43.
[11] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural network for high-speed flows, Comput. Methods Appl. Mech. Engrg. 360 (2020) 112789.
[12] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Variational Physics-Informed Neural Networks For Solving Partial Differential Equations, arXiv:1912.00873.
[13] E. Kharazmi, Z. Zhang, G.E. Karniadakis, hp-VPINNs: Variational Physics-Informed Neural Networks With Domain Decomposition, arXiv:2003.05385.
[14] K. Li, K. Tang, T. Wu, Q. Liao, D3M: A deep domain decomposition method for partial differential equations, IEEE Access 8 (2020) 5283–5294.
[15] Georgios Kissas, et al., Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 358 (1) (2020) 112623.
[16] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. 404 (2020) 109136.
[17] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Locally adaptive activation functions with slope recovery term for deep and physics-informed neural networks, 2019, arXiv preprint arXiv:1909.12228.
[18] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforwardneural networks, in: Aistats, Vol. 9, 2010, pp. 249–256.
[19] S. Ruder, An overview of gradient descent optimization algorithms, 2017, arXiv:1609.04747v2.
[20] D.P. Kingma, J.L. Ba, ADAM: A method for stochastic optimization, 2017, arXiv:1412.6980v9.
[21] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control. Signals Syst. (MCSS) 2 (1989) 303–314.
[22] A. Pinkus, Approximation theory of the MLP model in neural networsk, Acta Numer. 8 (1999) 143–195.
[23] T. Chen, H. Chen, Universal approximation by nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Trans. Neural Netw. 6 (4) (1995) 911–917.
[24] C. Basdevant, et al., Spectral and finite difference solution of the Burgers equation, Comput. Fluids 14 (1986) 23–41.
[25] M.J. Ablowitz, Nonlinear Dispersive Waves: Asymptotic Analysis and Solitons, Cambridge University Press, 2012.
[26] P.G. Drazin, R.S. Johnson, Solitons: An Introduction, Cambridge University Press, 1989.
[27] S.E. Esipov, Coupled Burgers equation: a model of poly-dispersive sedimentation, Phys. Rev. 52 (1995) 3711.
[28] J.D. Logan, An Introduction to Nonlinear Partial Differential Equations, Wiley-Interscience, New York, 1994.
[29] U. Ghia, K.N. Ghia, C.T. Shin, High-Re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method, J. Comput. Phys. 48 (1982) 387–411.
[30] A.D. Jagtap, Method of relaxed streamline upwinding for hyperbolic conservation laws, Wave Motion 78 (2018) 132–161.