A Novel Sequential Method to Train Physics Informed Neural Networks for Allen Cahn and Cahn Hilliard Equations

Revanth Mattey [a], Susanta Ghosh [a,b,*]

[a] Department of Mechanical Engineering–Engineering Mechanics, Michigan Technological University, MI, USA
[b] The Center for Data Sciences, The Center for Applied Mathematics and Statistics Michigan Technological University, MI, USA

Abstract

A physics informed neural network (PINN) incorporates the physics of a system by satisfying its boundary value problem through a neural network's loss function. The PINN approach has shown great success in approximating the map between the solution of a partial differential equation (PDE) and its spatio-temporal coordinates. However, the PINN's accuracy suffers significantly for strongly non-linear and higher-order time–varying partial differential equations such as Allen Cahn and Cahn Hilliard equations. To resolve this problem, a novel PINN scheme is proposed that solves the PDE sequentially over successive time segments using a single neural network. The key idea is to re-train the same neural network for solving the PDE over successive time segments while satisfying the already obtained solution for all previous time segments. Thus it is named as backward compatible PINN (bc-PINN). To illustrate the advantages of bc-PINN, the Cahn Hilliard and Allen Cahn equations are solved. These equations are widely used to describe phase separation and reaction-diffusion systems. Additionally, two new techniques have been introduced to improve the proposed bc-PINN scheme. The first technique uses the initial condition of a time–segment to guide the neural network map closer to the true map over that segment. The second technique is a transfer learning approach where the features learned from the previous training are preserved. We have demonstrated that these two techniques improve the accuracy and efficiency of the bc-PINN scheme significantly. It has also been demonstrated that the convergence is improved by using a phase space representation for higher-order PDEs. It is shown that the proposed bc-PINN technique is significantly more accurate and efficient than PINN.

Keywords: Physics informed neural networks,, Partial differential equation (PDEs), Allen Cahn equation, Cahn Hilliard equation

1. Introduction

Traditional physics-based numerical methods for solving partial differential equations (PDEs) have found remarkable success in solving various science and engineering problems. These methods are accurate but computationally expensive for complex problems such as nonlinear PDEs and requires problem–specific techniques. In the last decade data driven methods have gained a lot of attention in almost all areas of science and engineering. Data driven methods for PDEs can help in identifying highly non-linear mappings (between the inputs and outputs) which can substitute or augment expensive physics based simulations. Due to their versatility and fast

*Corresponding author; Email: susantag@mtu.edu

evaluation capabilities, machine learning based models can be used as PDE solvers in situations when there is a requirement of a large number of simulations such as the inverse–problem and homogenization [1, 2].
Several data–driven techniques have been attempted to solve PDEs. For instance, the Gaussian Process based approaches are described in [3, 4, 5, 6, 7]. Despite the ease of training for Gaussian Process, this approach did not gain as much popularity as a neural network for solving PDEs due to its difficulties in handling high dimensional problems.

Among different data–driven techniques for PDEs the Physics Informed Neural Networks (PINN) has shown remarkable promise and versatility. PINN is a new class of machine learning technique where a neural network's loss function is designed to satisfy the Initial Boundary Value Problem (IBVP) [8]. A PINN "learns" the non linear map between the spatio–temporal input and the solution of the PDE in a given domain. Henceforth, throughout this paper the PINN described in [8] would be referred to as std-PINN. std-PINN utilizes the automatic-differentiation capability [9] to compute the derivatives of the field variables.

Different variants of std-PINN are shown to work effectively in solving many forward and inverse problems [10, 11, 12]. Recently in [13], PINN has been extended to satisfy various conservation laws while solving the PDEs. This approach is named as cPINNs. cPINNs solve the problem over several sub-domains and ensure flux continuity at the boundaries of the sub-domains. Another extension to cPINNs is XPINN known as extended PINN, where the authors propose a generalized space–time domain decomposition based deep learning framework [14]. The key idea in XPINN is to decompose the domain into multiple sub-domains and train the sub-domains using multiple neural networks (sub Net), while ensuring C^0 continuity along the interfaces. While most of the PINN approaches solve the strong form of a PDE, it can also be used to solve the weak (variational) form of a PDE. Since the weak form incorporates the natural boundary conditions, the neural network solution only needs to satisfy the essential boundary conditions *a priori*. This aspect is used in several numerical methods for PDEs such as the finite element method. Due to this advantage of weak form over strong form the application of PINN on the weak form has been investigated in [15]. In this study, the authors have considered the variational form for stochastic PDEs and applied the idea of PINNs to obtain the solution of the PDE. Also the corresponding uncertainty propagation through their model is presented in [15, 16]. Uncertainty quantification provides the variation associated with the prediction of the model. It is particularly useful for systems where there is a high cost of data acquisition or lack of high resolution data [17]. The authors in [18] proposed a Bayesian approach for physics informed neural network to solve forward and inverse problems.

The promise and versatility of PINN have been demonstrated through its application for a wide range of problems. PINN has been used in modeling subsurface transport phenomena [19], approximating Euler equations for high speed flows [1], constitutive modeling of stress–strain behavior in biological tissues [20], predicting arterial blood pressure from noisy MRI data of flow velocity [21], and cardiac activation mapping for diagnosing atrial fibrillation [22].

Recently many other data–driven techniques for solving time varying PDEs have been proposed. In [23] the authors proposed a framework called Neural-PDE which aims to learn the solution of the PDE by utilizing numerical techniques like FDM (Finite difference method) and LSTM (Long short term memory) networks. The method aims to predict the solution of the PDE at n future time steps by using the meshgrid data (solution) of all the previous time steps. Artificial neural network based approaches for solving parametric PDEs have been introduced by the authors in [24]. Operator learning is a new and emerging technique for solving PDEs. In operator learning a map between initial condition and solution of the PDE is learnt by using multiple instantiations [25, 26, 27].

In the present work, we demonstrate that the accuracy of the std-PINN [8] suffers in the presence of (i) Strong Non-linearity, and (ii) Higher order partial differential operators. In order to illustrate the above, we chose the Allen Cahn equation having *strong non-linearity* and Cahn Hilliard equation having *strong non-linearity and fourth order derivative.* These are the two most widely used PDEs to study diffusion separation and multi–phase flows [28, 29, 30, 31]. To overcome the drawbacks of the std-PINN, we have proposed an extension, which is named as backward compatible PINN (bc-PINN). The proposed bc-PINN solves the PDE over successive time segments by re-training the same neural network, where the key idea is:

To ensure that the neural network can reproduce the solution for all the prior time segments while solving the PDE for a particular time segment.

Henceforth, this idea is referred to as *backward compatibility.* Some of the main advantages of the proposed bc-PINN method are as follows:

1. It works for higher order and strongly nonlinear PDEs by using fewer iterations and collocation points while achieving significantly higher accuracy when compared to std-PINN.
2. A single neural network is used for the entire domain and continuity across the time segments is ensured for the predicted solution and its derivatives.

The rest of the paper is organized as follows: in section (2) the std-PINN method is briefly reviewed; in section (3) the proposed bc-PINN method is presented; in section (4) the Allen Cahn and Cahn Hilliard equations are described; in section (5) the bc-PINN method is analyzed and compared against the std-PINN and the XPINN method for the one dimensional (1D) Allen Cahn and Cahn Hilliard equations; in section (6) two new techniques initial condition guided learning and transfer learning based acceleration have been presented along with the results for the two dimensional (2D) Allen Cahn and Cahn Hilliard equations. Finally, the conclusions are presented in section (7).

2. A brief review of standard physics informed neural network (std-PINN) for partial differential equations

Physics informed neural network (PINN) is a class of machine learning model where the governing PDE is satisfied through the loss function of the neural network [8]. The efficient optimization and prediction capabilities of neural network are exploited in the std-PINN approach. In std-PINN a neural network is trained to predict the solution at any point in the entire spatial–temporal domain. Let's consider the general form of a m^{th} order partial differential equation (PDE):

$$h_t = F(h(\boldsymbol{x},t), h^{(1)}_{\boldsymbol{x}}(\boldsymbol{x},t)\,, h^{(2)}_{\boldsymbol{x}}(\boldsymbol{x},t), \cdots, h^{(m)}_{\boldsymbol{x}}(\boldsymbol{x},t))\,,\ \boldsymbol{x} \in \Omega \subset \mathbb{R}^D\,,\ t \in (0,T] \tag{1}$$

Here, Ω is an open set of $\mathbb{R}^D$ (D = 1,2,3). F is a non linear function of the solution $h(\boldsymbol{x},t)$ and it's spatial derivatives $(h^{(1)}_{\boldsymbol{x}}(\boldsymbol{x},t), h^{(2)}_{\boldsymbol{x}}(\boldsymbol{x},t), \cdots, h^{(m)}_{\boldsymbol{x}}(\boldsymbol{x},t))$ where $\boldsymbol{x}$ and t are the space and time coordinates respectively. The corresponding boundary conditions and initial conditions are

$$\begin{aligned} h(\boldsymbol{x},0) &= \phi(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \\ h(-\boldsymbol{x},t) &= h(\boldsymbol{x},t), \quad (\boldsymbol{x},t) \in \Gamma \times (0,T] \\ h^{(1)}_{\boldsymbol{x}}(-\boldsymbol{x},t) &= h^{(1)}_{\boldsymbol{x}}(\boldsymbol{x},t)\,, \quad (\boldsymbol{x},t) \in \Gamma \times (0,T] \end{aligned} \tag{2}$$

Where, Γ is the boundary of Ω. The PDE, the Initial and the Boundary Conditions (given by equation (1–2)) form a initial–boundary value problem (IBVP) considered in this study. The boundary conditions are taken as periodic and the initial condition is a real function.

std-PINN approximates the map between points in the spatio-temporal domain to the solution of the PDE. The parameters of the neural network are randomly initialized and iteratively updated by minimizing the loss function that enforces the PDE. The std-PINN's loss function consists of three error components, for the prediction of the neural network as in the following (i) Initial Condition, (ii) Boundary Condition, and (iii) PDE. Let $\hat{h}(\boldsymbol{x}, t)$ be the output of neural network. The three components of the std-PINN's loss function are given below:

- Mean squared error on the Initial Condition

$$\text{MSE}_I = \frac{1}{N_i}\sum_{k=1}^{N_i}\left(\hat{h}(\boldsymbol{x}_k^i, 0) - h_k^i\right)^2 \ , \quad \boldsymbol{x}_k^i \in \Omega \tag{3}$$

  where $\hat{h}(\boldsymbol{x}_k^i, 0)$ is the neural network output and h_k^i is the given initial condition at $(\boldsymbol{x}_k^i, 0)$. Here, the superscript, $(\bullet)^i$ stands for initial condition.

- Mean squared error on the Boundary Condition

$$\text{MSE}_B = \frac{1}{N_b}\sum_{k=1}^{N_b}\sum_{d=1}^{n_d}\left(\hat{h}^{(d-1)}(\boldsymbol{x}_k^b, t_k^b) - \hat{h}^{(d-1)}(-\boldsymbol{x}_k^b, t_k^b)\right)^2 \ , \qquad (\boldsymbol{x}_k^b, t_k^b) \in \Gamma \times (0, T] \tag{4}$$

  where n_d is the highest order of derivative to which the periodicity is enforced on the boundary, Γ. Here, the superscript, $(\bullet)^b$ stands for boundary condition.

- The Mean squared error due to Residual of the partial differential equation

$$\begin{gathered} R := \hat{h}_t - F(\hat{h}, \hat{h}_{\boldsymbol{x}}^{(1)}, \hat{h}_{\boldsymbol{x}}^{(2)}, ...\hat{h}_{\boldsymbol{x}}^{(m)}) \\ \text{MSE}_R = \frac{1}{N_r}\sum_{k=1}^{N_r}\left(R(\boldsymbol{x}_k^r, t_k^r)\right)^2 \ , \qquad (\boldsymbol{x}_k^r, t_k^r) \in \Omega \times (0, T] \end{gathered} \tag{5}$$

The superscript, $(\bullet)^r$ stands for residual of the PDE. $(\boldsymbol{x}_k^i)$ and $(\boldsymbol{x}_k^b, t_k^b)$, represent the set of points where the initial and boundary errors are computed. The residual/collocation error is computed at the collocation points $(\boldsymbol{x}_k^r, t_k^r)$. These points on the domain and the boundary are obtained using a latin hypercube sampling approach. Therefore, the total loss function of the neural network is given by adding all the aforementioned mean squared errors

$$\text{MSE} = \text{MSE}_I + \text{MSE}_B + \text{MSE}_R \tag{6}$$

Once the std-PINN is trained, the accuracy of the predicted solution is computed with respect to the reference solution at unknown points (called testing points). Highly accurate solution of the initial boundary value problem obtained by the Chebyshev polynomial based numerical algorithm [32] and is considered as the reference solution. The relative total error (ε_{total}) of the PINN's prediction over the entire domain is obtained by normalizing the error with respect to the reference solution as

$$\varepsilon_{total} = \frac{\left[\frac{1}{N}\sum_{k=1}^{N}\left(\hat{h}(\boldsymbol{x}_k, t_k) - h(\boldsymbol{x}_k, t_k)\right)^2\right]^{1/2}}{\left[\frac{1}{N}\sum_{k=1}^{N}\left(h(\boldsymbol{x}_k, t_k)\right)^2\right]^{1/2}} \tag{7}$$

The relative error (ε) of the PINN's prediction at each point is obtained by normalizing the absolute error with respect to the reference solution as

$$\varepsilon(\mathbf{x}_k, t_k) = \frac{\left|\hat{h}(\boldsymbol{x}_k, t_k) - h(\boldsymbol{x}_k, t_k)\right|}{\left[\sum_{k=1}^{N} \left(h(\boldsymbol{x}_k, t_k)\right)^2\right]^{1/2}} \tag{8}$$

Where $h(\boldsymbol{x}_k, t_k)$ is the reference solution and $\hat{h}(\boldsymbol{x}_k, t_k)$ is the neural network prediction for a set of testing points $\{(\boldsymbol{x}_k, t_k)\}_{k=1}^{N}$, $(\boldsymbol{x}_k, t_k) \in \Omega \times (0, T]$. For all comparisons between reference and predicted solutions the relative total error 'ε_{total}' and relative error 'ε' is used.

3. The proposed backward compatible sequential PINN method (bc-PINN)

In this section, we introduce an extension of the std-PINN technique that solves an initial-boundary value problem sequentially in time.

3.1. bc-PINN

In the proposed method the PDE is solved progressively in time by re-training a single neural network over successive time segments. The limitation of such retraining is that the network can predict only for the latest time segment and cannot predict for previous time segments for those it has been trained earlier. To overcome this limitation, the proposed model is designed to satisfy the solution of all the previous time segments while solving the PDE over a particular time segment. This scheme ensures backward compatibility of the solution by a single network. The proposed method is henceforth referred as backward compatible PINN (bc-PINN). The schematics of bc-PINN for a particular time segment is shown in figure (1) and the sequential scheme of proposed bc-PINN approach is shown in figure (2). In bc-PINN the time domain $[0, T]$ is discretized into n_{max} segments as

$$[T_0 = 0, T_1],\ [T_1, T_2], \cdots,\ [T_{n-1}, T_n],\ \cdots, [T_{n_{max}-1}, T_{n_{max}} = T] \tag{9}$$

where the n^{th} segment is denoted as $\Delta T_n = [T_{n-1}, T_n]$, $n = 1, \cdots, n_{max}$.

For the first time segment ΔT_1 the solution of the PDE is sought through the std-PINN by minimizing the following loss function

$$\begin{gathered} \text{MSE}_{\Delta T_1} = w_i\, \text{MSE}_I(\boldsymbol{x}_k^i, \mathit{0}) + w_b\, \text{MSE}_B(\boldsymbol{x}_k^b, t_k^b) + w_r\, \text{MSE}_R(\boldsymbol{x}_k^r, t_k^r) \\ \boldsymbol{x}_k^i \in \Omega, \qquad (\boldsymbol{x}_k^b, t_k^b) \in \Gamma \times (0, T_1] \qquad (\boldsymbol{x}_k^r, t_k^r) \in \Omega \times (0, T_1] \end{gathered} \tag{10}$$

Here, $(\boldsymbol{x}_k^i, t_k^i)$ represent the set of points where the error on initial condition is computed and $(\boldsymbol{x}_k^b, t_k^b)$ represent the set of points where the error on boundary condition is computed within the time segment $\Delta T_1 = (0, T_1]$. For all of the subsequent time segments (i.e. ΔT_n, $n = 2, \cdots, n_{max}$) we propose a novel loss function, which satisfies the solution of all previous time segments. The solution of all previous time segments is enforced by penalizing the departure from the already obtained solutions from the previous training, as given by

$$\begin{aligned} \text{MSE}_{\Delta T_n} = w_i\, \text{MSE}_I(\boldsymbol{x}_k^i,\, T_{n-\mathit{1}}) + w_b\, \text{MSE}_B(\boldsymbol{x}_k^b, t_k^b) + w_r\, \text{MSE}_R(\boldsymbol{x}_k^r, t_k^r) \\ + w_s\, \text{MSE}_S(\boldsymbol{x}_k^s, t_k^s)\,, \quad n = \mathit{2}, \cdots, n_{max} \\ x_k^i \in \Omega, \qquad (\boldsymbol{x}_k^b, t_k^b) \in \Gamma \times (T_{n-1}, T_n] \\ (\boldsymbol{x}_k^r, t_k^r) \in \Omega \times (T_{n-1}, T_n], \qquad (\boldsymbol{x}_k^s, t_k^s) \in \Omega \times [0, T_{n-1}] \end{aligned} \tag{11}$$

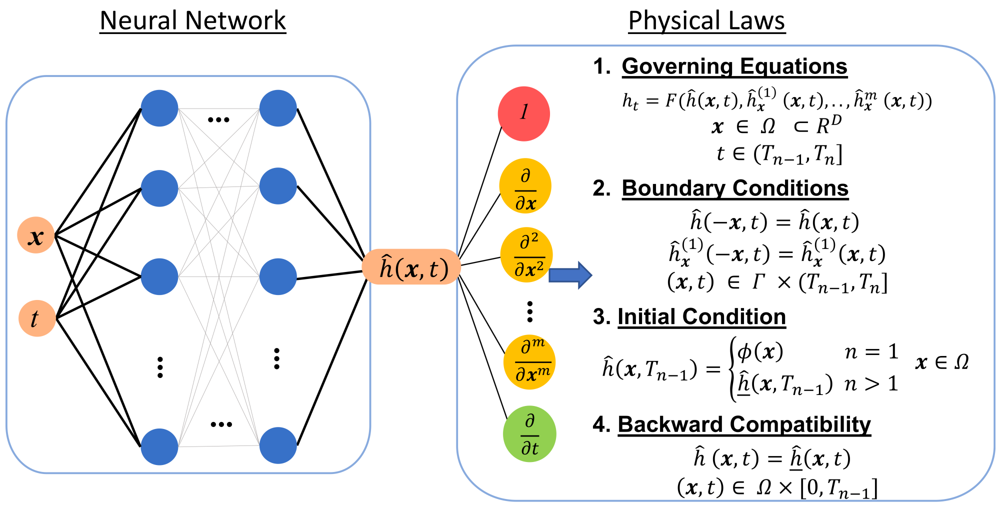

Neural Network
Physical Laws
x
t
$\hat{h}(\boldsymbol{x},t)$
I
$\frac{\partial}{\partial x}$
$\frac{\partial^2}{\partial x^2}$
$\frac{\partial^m}{\partial x^m}$
$\frac{\partial}{\partial t}$
1. Governing Equations
$h_t = F(\hat{h}(\boldsymbol{x},t), \hat{h}_{\boldsymbol{x}}^{(1)}(\boldsymbol{x},t), .., \hat{h}_{\boldsymbol{x}}^{m}(\boldsymbol{x},t))$
$\boldsymbol{x} \in \Omega \subset R^D$
$t \in (T_{n-1}, T_n]$
2. Boundary Conditions
$\hat{h}(-\boldsymbol{x},t) = \hat{h}(\boldsymbol{x},t)$
$\hat{h}_{\boldsymbol{x}}^{(1)}(-\boldsymbol{x},t) = \hat{h}_{\boldsymbol{x}}^{(1)}(\boldsymbol{x},t)$
$(\boldsymbol{x},t) \in \Gamma \times (T_{n-1}, T_n]$
3. Initial Condition
$\hat{h}(\boldsymbol{x},T_{n-1}) = \begin{cases} \phi(\boldsymbol{x}) & n=1 \\ \underline{\hat{h}}(\boldsymbol{x},T_{n-1}) & n>1 \end{cases} \quad \boldsymbol{x} \in \Omega$
4. Backward Compatibility
$\hat{h}(\boldsymbol{x},t) = \underline{\hat{h}}(\boldsymbol{x},t)$
$(\boldsymbol{x},t) \in \Omega \times [0, T_{n-1}]$

Figure 1: The schematics of the proposed backward compatible PINN (bc-PINN) approach for a time segment ($(T_{n-1}, T_n]$). The neural network re-trains the PDE over $(T_{n-1}, T_n]$ while satisfying the solution for all previous time segments. The error in the Initial Condition is computed at time $t = 0$ for the first time segment and at time $t = (T_{n-1}$ for the nth time segment.

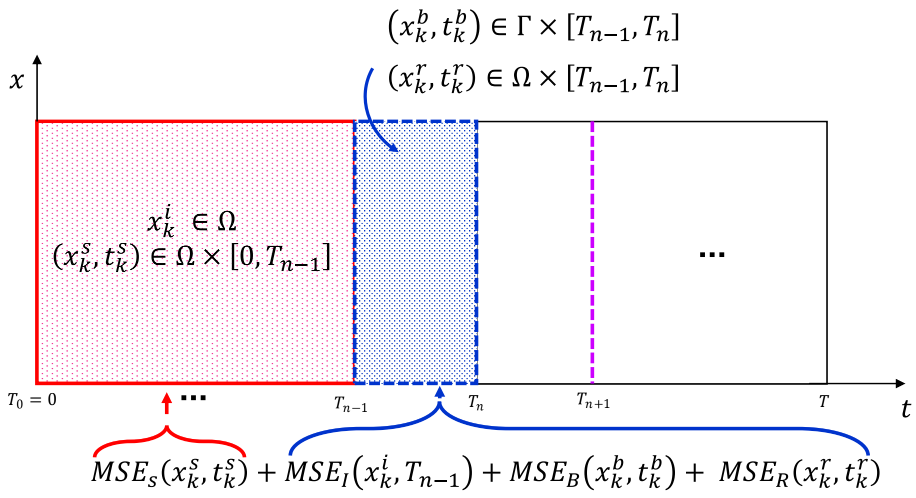

$(x_k^b, t_k^b) \in \Gamma \times [T_{n-1}, T_n]$
$(x_k^r, t_k^r) \in \Omega \times [T_{n-1}, T_n]$
x
$x_k^i \in \Omega$
$(x_k^s, t_k^s) \in \Omega \times [0, T_{n-1}]$
...
$T_0 = 0$
T_{n-1}
T_n
T_{n+1}
T
t
$MSE_s(x_k^s, t_k^s) + MSE_I(x_k^i, T_{n-1}) + MSE_B(x_k^b, t_k^b) + MSE_R(x_k^r, t_k^r)$

Figure 2: Illustration of the proposed backward compatibility scheme (over a 1D domain) that satisfies the solutions obtained on all previous time segments ($[0, T_{n-1}]$) while satisfying the PDE on the current time segment ($(T_{n-1}, T_n]$). The error in the Initial Condition is computed at time $t = 0$ for the first time segment and at time $t = (T_{n-1}$ for the nth time segment.

Here, $(\boldsymbol{x}_k^i, T_{n-1})$ represent the set of points where the error on initial condition is computed and $(\boldsymbol{x}_k^b, t_k^b)$ represent the set of points where the error on boundary conditions is computed within the time segment $(T_{n-1}, T_n]$. The residual/collocation error as given in equation (5) is computed at the collocation points $(\boldsymbol{x}_k^r, t_k^r)$. We also minimize the departure from the already obtained solution that were stored at the grid points $(\boldsymbol{x}_k^s, t_k^s)$. The solution obtained (on $(0, T_n]$) at the n^{th} segment is stored for using it in the $(n+1)^{\text{th}}$ segment.

The weights (w_i, w_b, w_r, w_s) given in equation (11) help in faster convergence to the true solution.

These weights are utilized to scale the difference in the magnitude of the errors. In the present work, the difference in the order of derivatives (spatial) and the number of spatial dimensions are used for scaling. From a mathematical perspective, the weighting of the loss function can be seen as a mechanism that forces the learning process to focus on the terms where high prediction accuracy is required.

In sections (3.2,3.3), two techniques have been proposed to further improve the accuracy and efficiency of the bc-PINN scheme.

3.2. Initial condition guided learning (ICGL)

In initial condition guided learning, the key idea is to perform the training in two stages for each time segment. In the first stage the neural network is trained to match only the initial condition of that time segment, using a small fraction of the total iterations. Therefore, the loss function for ICGL (MSE_{SI}) is as follows:

$$\mathrm{MSE}_{SI} = \frac{1}{N_{SI}} \sum_{k=1}^{N_{SI}} \left(\hat{h}(\boldsymbol{x}_k^{SI}, t_k^{SI}) - \underline{\hat{h}}_k^{SI}(\boldsymbol{x}_k^{SI}, T_{n-1}) \right)^2 \ , \ (\boldsymbol{x}_k^{SI}, t_k^{SI}) \in \Omega \times (T_{n-1}, T_n] \tag{12}$$

Here, $\hat{h}(\boldsymbol{x}, t)$ is the neural network prediction and $\underline{\hat{h}}(\boldsymbol{x}, t)$ is the known solution through the neural network at the previous time step, T_{n-1}. Whereas, in the second stage the weights obtained in the first stage are taken as the initial weights and the bc-PINN as described earlier is trained. This is motivated by the fact that the solution of a PDE in a time segment is expected to be close to the initial condition if the segment is small. Thus matching the initial condition brings the neural network map closer to the true map. Since in the first step there are no derivative calculations involved, it accelerates the training.

3.3. Transfer learning based acceleration (TL)

A transfer learning approach is implemented that uses the weights and biases from a bc-PINN that has been trained on a different initial condition or the previous segment. The transfer learning approach preserves the features from a previous training as reported in [33]. This helps in faster convergence for a new initial condition. A notable advantage of this technique is that the training time can be significantly reduced as the number of trainable parameters decreases.

The two proposed techniques proposed (*Initial condition guided learning* and *Transfer learning based acceleration*) shows significant improvement in the accuracy of the bc-PINN - solution for the Allen Cahn and Cahn Hilliard equations. These two techniques are independent of each other and can be used either in tandem or individually.

3.4. Details of the neural network of bc-PINN

We have used a standard (deep) neural network with two input neurons consisting of the spatial variables ($\boldsymbol{x}$) and temporal variable (t). The output of the neural network ($\hat{h}(\mathbf{x}, t)$) approximates the solution of the PDE ($h(\boldsymbol{x}, t)$). To avoid model bias due to input features of different scales we have performed “min-max” normalization to scale the data uniformly. For solving 1D Allen Cahn and Cahn Hilliard equations using bc-PINN the architecture of the neural network chosen has 6 hidden layers with 128 neurons in each layer. Whereas, while solving 2D Allen Cahn and Cahn Hilliard equations using the aforementioned architecture didn't yield a good performance. Thus, in order to choose the neural network architecture for solving 2D PDEs (section (6)) the following approach has been used. Preference has been given to increasing the number of neurons of a hidden layer rather than increasing the number of hidden layers. This is backed by the fact that increasing hidden layers is more computationally intensive than increasing the number of neurons. For both std-PINN and bc-PINN, *tanh* is chosen as the activation function. Even though it's well known that *tanh* activation function has a problem of vanishing gradient in very deep networks [34]. The advantages of *tanh* activation function are that its continuous (range [-1,1]) and differentiable. Since, *tanh* is a non-linear function it gives neural network the capability to learn non-linear maps [35]. The neural network has more than 100,000 learning parameters which have been initialized using the "xavier initialization" [36] technique. The optimization of the loss function and updating the learning parameters (weights and biases of the neural network) is performed using the ADAM and LBFGS optimizers. The learning rate for ADAM optimizer is considered as 0.001 with all other parameters as suggested in [37]. Following std-PINN, after training the neural network using the ADAM optimizer we again train it using the L–BFGS optimizer until one of the following stopping criteria is met: (i) Maximum iterations are equal to 50,000, (ii) Maximum number of function evaluations are equal to 50,000, (iii) Maximum number of line search steps (per iteration) equal to 50, (iv) The maximum number of variable metric corrections used to define the limited memory matrix are equal to 50, (v) The iteration stops when $\frac{f^k - f^{k+1}}{max(|f^k|,|f^{k+1}|,1)} <= 2.22044604925e - 16$, where f is the neural network objective function and k is the iteration number.

3.5. Details of the Computational Platform

All the neural networks are trained on Nvidia Tesla P100 (3584 CUDA cores and 16GB of HBM2 vRAM) and Nvidia Volta V100 GPU (5120 CUDA cores, 640 Tensor cores and 16GB of HBM2 vRAM). For inferencing and generating the reference solutions via chebfun, we have used Dell precision 3630 workstation with Intel core i7-9700k 8 core (4.9 GHz Turbo) and 32 GB RAM. The software packages used for all the computations are Tensorflow 1.15 and MATLAB R2020a. All the variables defined for computations in tensorflow are of float32 data type.

3.6. The reference solution

Accurate numerical solutions for the Allen Cahn and Cahn Hilliard equations are obtained using the chebfun package [32]. The chebfun approach provides a polynomial interpolant for smooth functions in Chebyshev points. To solve time varying PDEs an exponential time differencing with Runge–Kutta time stepping scheme [38] has been implemented in chebfun, which is used in the present work. Henceforth, these solutions are considered as the reference solutions. We have taken 512 points for spatial discretization and 201 points for discretization in time scale. A fourth order Runge–Kutta time integrator with time step $\Delta t = 10^{-5}$ is used.

The bc-PINN approach is applied to solve Allen Cahn equation and Cahn Hilliard equation in sections (5 and 6) to demonstrate its advantages for nonlinear and higher order PDEs in comparison to std-PINN method [8].

4. bc-PINN for Allen Cahn and Cahn Hilliard equations

The Allen Cahn and Cahn Hilliard [1] equations are two of the widely used partial differential equations for studying the phenomena of phase separation [39]. There are innumerous practical

[1]The Cahn Hilliard equation plays an essential role in the field of material science for describing the qualitative features in a phase separation process for two phase systems (assuming isotropy and constant temperature). The process of phase separation can be observed when a binary alloy is cooled down adequately. This leads to a state

applications of these equations in various fields such as material science [40, 41, 42, 43, 44, 45], biological systems [46, 47, 48, 49, 50, 51, 52], electro-chemical systems etc. [53, 54, 55].

4.1. Allen Cahn Equation

For every $\boldsymbol{x} \in \Omega$, ($\Omega$ is an open set of $\mathbb{R}^D$) the Allen Cahn equation can be written as:

$$\begin{gathered} h_t - c_1^2\, \nabla^2 h + f(h) = 0\,, \quad t \in (0,T], \;\; \boldsymbol{x} \in \Omega \subset \mathbb{R}^D \\ f(h) = c_2(h^3 - h) \end{gathered} \tag{13}$$

For a phase separation problem, the parameter h, represents the concentration of the individual component and the parameter 'c_1' represents the interfacial thickness. The solution progressively develops interfaces separating different phases. For a given initial condition, $h_0(\boldsymbol{x}) \in L^2(\Omega)$ and $T > 0$, we seek a function $h : \Omega \times (0,T] \to \mathbb{R}$ which satisfies the above equation. In order to implement the bc-PINN scheme for Allen Cahn equation, the following loss function has been used.

- Mean squared error on the initial Condition

$$\text{MSE}_I = \frac{1}{N_i} \sum_{k=1}^{N_i} \left(\hat{h}(\boldsymbol{x}_k^i, T_{n-1}) - h_k^i \right)^2, \quad \boldsymbol{x}_k^i \in \Omega \tag{14}$$

- Mean squared error on the boundary Condition

$$\text{MSE}_B = \frac{1}{N_b} \sum_{k=1}^{N_b} \sum_{d=1}^{n_d} \left(\hat{h}^{(d-1)}(\boldsymbol{x}_k^b, t_k^b) - \hat{h}^{(d-1)}(-\boldsymbol{x}_k^b, t_k^b) \right)^2 \qquad (\boldsymbol{x}_k^b, t_k^b) \in \Gamma \times (T_{n-1}, T_n] \tag{15}$$

  where n_d is the order to which periodicity is enforced on the boundary Γ. Here, the superscript, $(\bullet)^b$ stands for boundary condition.

- The Mean squared error due to residual of the partial differential equation

$$\begin{gathered} R := \hat{h}_t - c_1^2\, \nabla^2 \hat{h} + f(\hat{h}) \\ \text{MSE}_R = \frac{1}{N_r} \sum_{k=1}^{N_r} \left(R(\boldsymbol{x}_k^r, t_k^r) \right)^2, \qquad (\boldsymbol{x}_k^r, t_k^r) \in \Omega \times (T_{n-1}, T_n] \end{gathered} \tag{16}$$

  The superscript, $(\bullet)^r$ stands for residual of the PDE.

- Mean squared error for backward compatibility

$$\text{MSE}_S = \frac{1}{N_s} \sum_{k=1}^{N_s} \left(\hat{h}(\boldsymbol{x}_k^s, t_k^s) - \underline{\hat{h}}(\boldsymbol{x}_k^s, t_k^s) \right)^2, \qquad (\boldsymbol{x}_k^s, t_k^s) \in \Omega \times [0, T_{n-1}] \tag{17}$$

  where, $\hat{h}(\boldsymbol{x}, t)$ is the neural network prediction and $\underline{\hat{h}}(\boldsymbol{x}, t)$ is the known solution through the neural network from the previous time steps $\Omega \times [0, T_{n-1}]$. The superscript, $(\bullet)^s$ stands for the backward compatible solution.

- The total mean squared error is the same as given in equation (11)

of total nucleation which is mainly referred to as spinodal decomposition. In the subsequent stage coarsening occurs in the nucleated microstructure at a much slower rate. This whole phase separation phenomena affects the mechanical properties (eg. strength, hardness and fracture toughness) of the material.

4.2. Cahn Hilliard Equation

For every $\boldsymbol{x} \in \Omega$, ($\Omega$ is an open set of $\mathbb{R}^D$) the Cahn Hilliard equation can be written as:

$$h_t - \nabla^2 \left(\kappa f(h) - (\alpha\kappa)\nabla^2 h(\boldsymbol{x},t)\right) = 0\,, \quad t \in (0,T], \;\; \boldsymbol{x} \in \Omega \subset \mathbb{R}^D \tag{18}$$

To simplify the derivative calculation, a phase space representation of the Cahn Hilliard equation has been adopted. The phase space representation is widely used to represent a high order PDE into coupled multiple lower order PDEs. The phase space representation of the Cahn Hilliard equation (a fourth order PDE, equation (18)) yields two coupled second order PDEs.

$$\begin{gathered} h_t - \nabla^2(-(\alpha\,\kappa)\,\mu + \kappa f(h)) = 0, \quad \mu = \nabla^2 h \qquad t \in (0,T], \;\; \boldsymbol{x} \in \Omega \subset \mathbb{R}^D \\ f(h) = h^3 - h \end{gathered} \tag{19}$$

Since the entire process is governed by the Cahn Hilliard equation it is essential to understand the physical significance of each individual variable. The order parameter (h) in equation (18), refers to the rescaled density or concentration of one of the material components in the system and it takes values between (-1 and 1, which corresponds to their pure states). The density of second component is $(1-h)$, and this ensures that the total density over the simulation domain is a conserved quantity. The parameter κ is the mobility parameter and the parameter α is related to the surface tension at the interface. In order to implement the bc-PINN scheme for the Cahn Hilliard equation, the following loss function has been used.

- Mean squared error on the initial condition for $h(\boldsymbol{x},t)$ and $\mu(\boldsymbol{x},t)$

$$\text{MSE}_I = \frac{1}{N_i}\left\{\sum_{k=1}^{N_i}\left(\hat{h}(\boldsymbol{x}_k^i, T_{n-1}) - h_k^i\right)^2 + \sum_{k=1}^{N_i}\left(\hat{\mu}(\boldsymbol{x}_k^i, T_{n-1}) - \mu_k^i\right)^2\right\}\,, \quad \boldsymbol{x}_k^i \in \Omega \tag{20}$$

- Mean squared error on the boundary Condition

$$\begin{aligned} \text{MSE}_{B_h} &= \frac{1}{N_b}\left\{\sum_{k=1}^{N_b}\sum_{d=1}^{n_d}\left(\hat{h}^{(d-1)}(x_k^b, t_k^b) - \hat{h}^{(d-1)}(-x_k^b, t_k^b)\right)^2\right\} \\ \text{MSE}_{B_\mu} &= \frac{1}{N_b}\left\{\sum_{k=1}^{N_b}\sum_{d=1}^{n_d}\left(\hat{\mu}^{(d-1)}(x_k^b, t_k^b) - \hat{\mu}^{(d-1)}(-x_k^b, t_k^b)\right)^2\right\} \\ \text{MSE}_B &= \text{MSE}_{B_h} + \text{MSE}_{B_\mu}\,, \qquad (x_k^b, t_k^b) \in \Gamma \times (T_{n-1}, T_n] \end{aligned} \tag{21}$$

  Here, the superscript, $(\bullet)^b$ stands for boundary condition.

- The Mean squared error due to residual of the partial differential equation

$$\begin{gathered} R_1 := \hat{h}_t - \nabla^2\left(-(\alpha\kappa)\mu + \kappa f(\hat{h})\right) \\ R_2 := \hat{\mu} - \nabla^2\hat{h} \\ \text{MSE}_R = \frac{1}{N_r}\left\{\sum_{k=1}^{N_r} R_1\left(\boldsymbol{x}_k^r, t_k^r\right)^2 + \sum_{k=1}^{N_r} R_2\left(\boldsymbol{x}_k^r, t_k^r\right)^2\right\}\,, \qquad (\boldsymbol{x}_k^r, t_k^r) \in \Omega \times (T_{n-1}, T_n] \end{gathered} \tag{22}$$

  The superscript, $(\bullet)^r$ stands for residual of the PDE.

- Mean squared error for backward compatibility

$$\mathrm{MSE}_S = \frac{1}{N_s}\sum_{k=1}^{N_s}\left(\hat{h}(\boldsymbol{x}_k^s, t_k^s) - \underline{\hat{h}}(\boldsymbol{x}_k^s, t_k^s)\right)^2, \qquad (\boldsymbol{x}_k^s, t_k^s) \in \Omega \times [0, T_{n-1}] \tag{23}$$

  where, $\hat{h}(\boldsymbol{x}, t)$ is the neural network prediction and $\underline{\hat{h}}(\boldsymbol{x}, t)$ is the known solution through the neural network from the previous time steps $\Omega \times [0, T_{n-1}]$. The superscript, $(\bullet)^s$ stands for the backward compatible solution.

- The total mean squared error is the same as given in equation (11)

The boundary loss (equation (21) is applied for $n_d = 1$ on $\hat{h}$ and $\hat{\mu}$, to represent periodic boundary conditions. Equation (22) describes two components of residual for two PDEs in the phase space form of the Cahn Hilliard equation (equation (19)). In the following sections (5 and 6), the results of Allen Cahn and Cahn Hilliard equations obtained using bc-PINN are presented..

5. Results of Allen Cahn and Cahn Hilliard equations using bc-PINN in 1D

5.1. Allen Cahn equation in one dimension

In this section, a 1D time varying Allen Cahn equation has been considered. The PDE for Allen Cahn equation remains the same as described in section (4.1). Since, the domain considered is one dimensional, $\Omega \in [-1, 1]$ and $t \in (0, 1]$. The values of the parameters considered in equation (13) are, $c_1^2 = 0.0001$ and $c_2 = 5$.

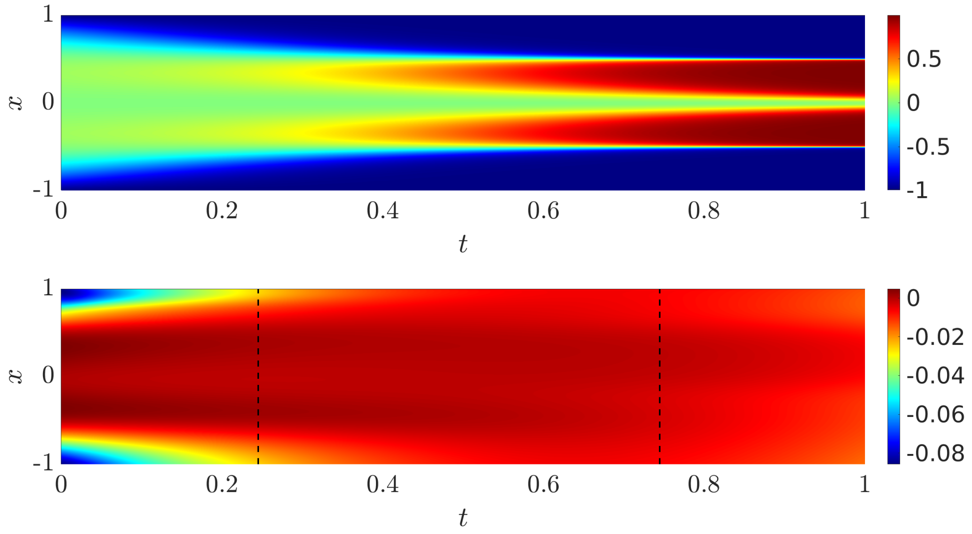
$$\begin{aligned} h(x, 0) &= x^2 \cos(\pi x) \\ h(x, t) &= h(-x, t), \quad (x, t) \in \Gamma \times (0, T] \\ h_x^{(1)}(x, t) &= h_x^{(1)}(-x, t), \quad (x, t) \in \Gamma \times (0, T] \end{aligned} \tag{24}$$

The above equation (24) gives details about the initial and boundary conditions (where Γ describes the boundary).

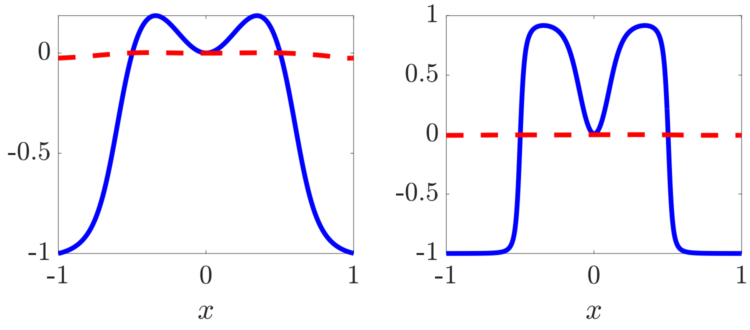
5.1.1. std-PINN for Allen Cahn equation

At first, the aforementioned Allen Cahn equation is solved using the std-PINN to demonstrate the challenge associated with non-linearity. The number of collocation points considered for training the std-PINN are 512 points for initial condition, 201 points for the boundary condition and 20,000 spatio-temporal points for computing the residual. The neural network architecture used has 4 hidden layers with 200 neurons in each layer and which is the same as given in [8]. For training the std-PINN, we have used both ADAM and L-BFGS optimizers. Training is performed using 100,000 ADAM iterations and the subsequent training has been performed using the L-BFGS optimization method until one of the stopping criteria is met: (i) Maximum iterations are equal to 50,000 (ii) Maximum number of function evaluations are equal to 50,000 (iii) Maximum number of line search steps (per iteration) equal to 50 (iv) The maximum number of variable metric corrections used to define the limited memory matrix are equal to 50 (v) The iteration stops when $\frac{f^k - f^{k+1}}{max(|f^k|, |f^{k+1}|, 1)} <= 2.22044604925e-16$,where f is the neural network objective function and k is the iteration number. The loss function for std-PINN is described in equation (3),(4) and (5). The solution of std-PINN is quite erroneous as shown in figure (3). In order to understand the reason for failure of the std-PINN, we analyze its prediction for the individual terms of the Allen Cahn equation. Figure (4) shows the individual terms of the Allen

Cahn equation obtained through the Chebfun method and the std-PINN. We observe that the std-PINN fail to predict the non-linear term $(5(h^3 - h))$ of the Allen Cahn equation. Therefore we have shown that the std-PINN [8] does not work for the Allen Cahn equation that consist of a strongly non-linear term.



(a)



(b)

Figure 3: (a): The reference solution (Top) and the std-PINN solution (Bottom) of the Allen Cahn equation for the entire spatio–temporal domain. (b): Time snapshots for the reference solution (——) and the std-PINN solution (- - -) at $t = 0.25$ and $t = 0.75$.

5.1.2. *bc-PINN for Allen Cahn equation*

To overcome this limitation of std-PINN, the backward compatible PINN approach has been used along with ICGL and TL techniques. Therefore for the proposed bc-PINN approach, the loss function given in equations (11,14-17) is used for any given time segment ΔT_n. The hyper-parameters associated with training the bc-PINN are number of ADAM iterations (N_{iter}), time steps per segment and number of residual collocation points (N_r) per segment.

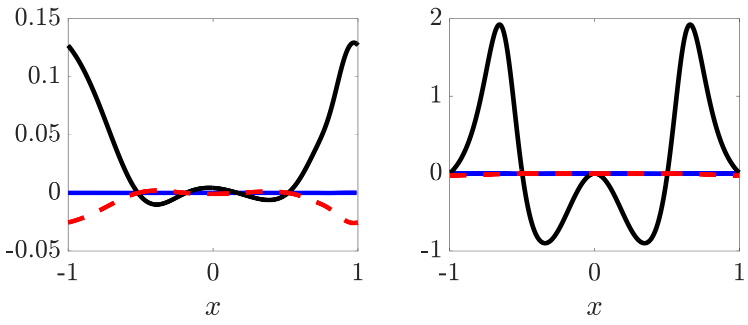


Figure 4: Individual terms of the Allen Cahn Equation obtained through the (Left): std-PINN method (Right): Chebfun method. $h(x, t)$, (- - -), $0.0001\nabla^2 h$ (——) and $5(h^3 - h)$ (——) at t = 0.25.

Variable	Description	Number
N_i	Initial collocation points	128
N_b	Boundary collocation points	50/segment
N_r	Residual collocation points	20000/segment
N_{iter}	Number of ADAM iterations	10000/segment

Table 1: Description of Training Data for Allen Cahn Equation. The segment considered here consists of 50 time steps.

In table (1), initial collocation points (N_i) refer to the input spatio-temporal points where the initial condition is prescribed. (N_b) and (N_r) refer to the number of boundary and residual collocation points respectively and the output of the neural network at these points is used to compute the loss function. For computing the backward compatibility loss the solution predicted by the neural network in all the previous time steps is utilized. To reiterate, the reference solution is only available at the initial condition whereas for all other points in the entire domain the solution of the PDE is obtained through minimizing the bc-PINN loss function.

The reference and predicted solution at time $t = 0.25$ obtained by the std-PINN and bc-PINN are shown in figure (5). While the std-PINN fails, the proposed bc-PINN predicts the solution quite accurately. The relative total errors (ε_{total}) and the total training time for std-PINN, XPINN, bc-PINN and bc-PINN with ICGL and TL approaches are shown in table 2. By using techniques like ICGL and TL we have been able to observe a significant improvement in accuracy and also reduction in training time. In order to implement the XPINN [14] for the time-varying Allen Cahn equation the entire domain has been decomposed into 5 sub-domains sequentially across time and each sub-domain is trained by a sub-net. Interfacial solution continuity across different sub-domains has also been implemented and trained using 50,000 ADAM iterations and L-BFGS optimizer with the same stopping criteria as bc-PINN. But the main drawback of XPINN in solving forward problems is that while training subnet-1 all other subsequent subnets are also trained which increases the computational cost. For example, before even the solution converges in subnet-1, subnet-2 is simultaneously being trained which searches for the solution of the PDE in an infinite-dimensional space in other words subnet-2 is trying to predict the solution with an incorrect initial condition, which has not yet converged to the correct solution.

The comparison between the predicted solution using bc-PINN and the reference solution is shown in figure (6). This shows that the bc-PINN can accurately predict the solution for the entire domain. The solutions and errors by the std-PINN and bc-PINN are compared in

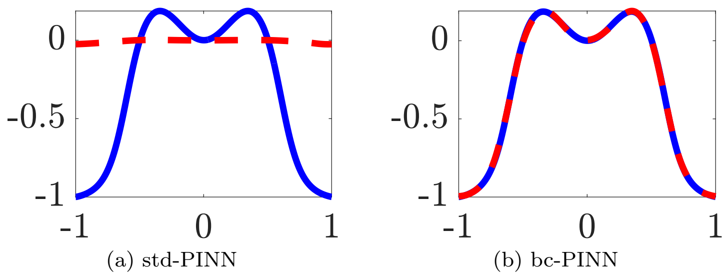


Figure 5: Reference (——) and Predicted (- - -) solution at time t = 0.25

Method	Error($\varepsilon_{\text{total}}$)	Training Time
std-PINN	0.9919	4.5hrs
XPINN	0.9612	4 hrs
bc-PINN	0.0701	2 hrs
bc-PINN with ICGL and TL	0.0168	0.75 hrs

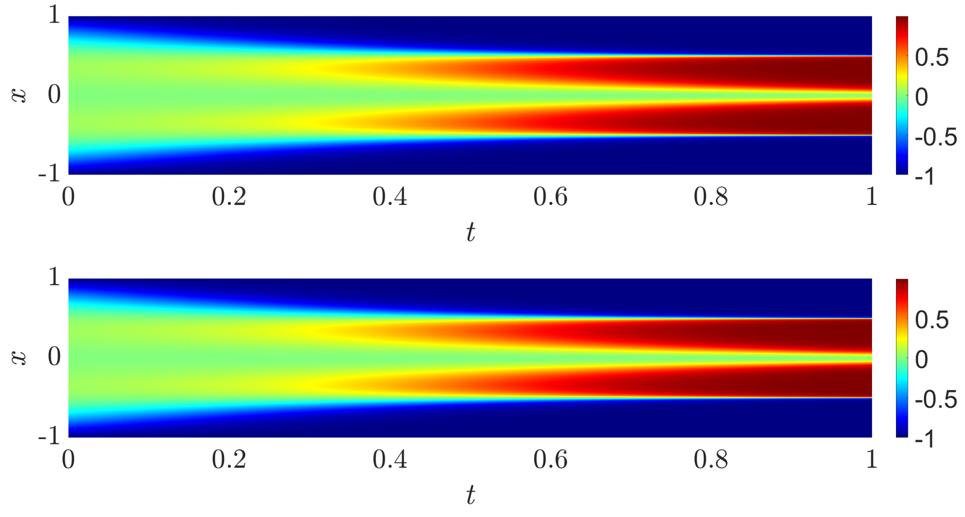
Table 2: Relative total errors (equation (7)) over the entire domain with respect to Chebfun solution for different methods.

figure (7), showing much higher accuracy by the bc-PINN. The error plots confirms high accuracy of bc-PINN. The error increases with time very slowly. This is due to two reasons: (i) the solution becomes progressively phase-separated (between zero and one) yielding greater curvatures and sharp phase-boundaries that are difficult to capture, and (ii) due to the sequential nature of the bc-PINN approach the error accumulates with time progression, which is similar to the time-integrators. To illustrate the high accuracy of the bc-PINN approach, solutions and errors for different values of the interfacial thickness (c_1) is plotted in figure (8). As we decrease the parameter c_1, it can be seen that the error in the prediction decreases. The parameter c_1 controls the effect of the double derivative of the solution ($\nabla^2 h$). Therefore, as we decrease c_1 the error due to the approximation in derivative reduces and thus the accuracy of the bc-PINN solution increases. In Appendix B, a new loss function including a logarithmic residual for the Allen Cahn equation is discussed. This new logarithmic residual bc-PINN approach and its results are presented in comparison with the simple bc-PINN approach without a logarithmic residual.

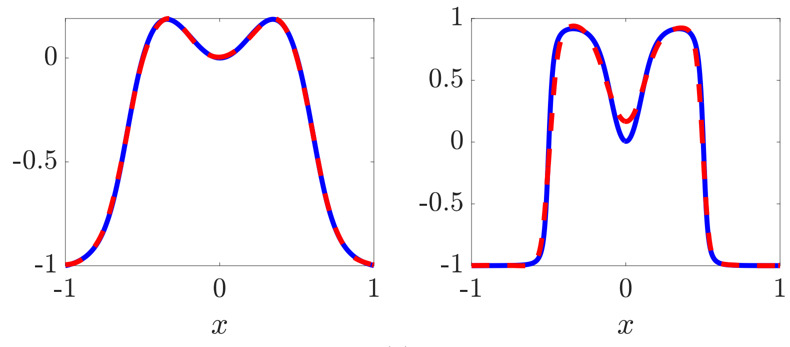
5.2. Cahn Hilliard equation in one dimension

In this section, a 1D time varying Cahn Hilliard equation has been considered. The PDE for Cahn Hilliard equation remains the same as described in section (4.2). Since, the domain considered is one dimensional, $\Omega \in [-1,1]$ and $t \in (0,1]$. The values of the parameters considered in equation (18) are, $\alpha = 0.02$ and $\kappa = 1$.

$$\begin{gathered}
h(x,0) = cos(\pi x) - \exp\left(-4(\pi x)^2\right) \\
h(-x,t) = h(x,t), \quad (x,t) \in \Gamma \times (0,T] \\
h_x^{(1)}(-x,t) = h_x^{(1)}(x,t)\,, \quad (x,t) \in \Gamma \times (0,T] \\
\mu(-x,t) = \mu(x,t), \quad (x,t) \in \Gamma \times (0,T] \\
\mu_x^{(1)}(-x,t) = \mu_x^{(1)}(x,t)\,, \quad (x,t) \in \Gamma \times (0,T]
\end{gathered} \tag{25}$$



(a)



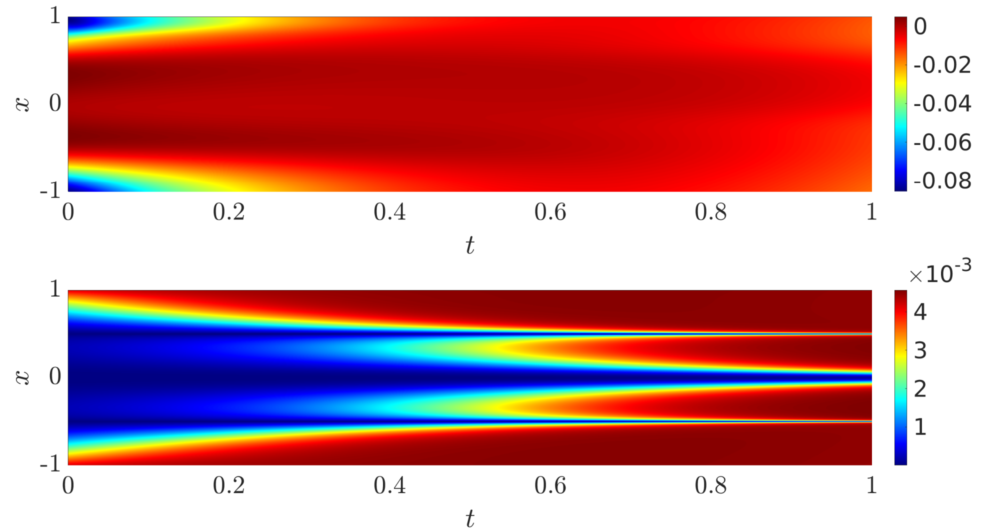
(b)

Figure 6: (a): Reference (Top) and bc-PINN (Bottom) solutions of the Allen Cahn equation for the entire spatio–temporal domain. (b): The Reference (——) and the bc-PINN (- - -) solutions at time $t = 0.25$ and $t = 0.75$.

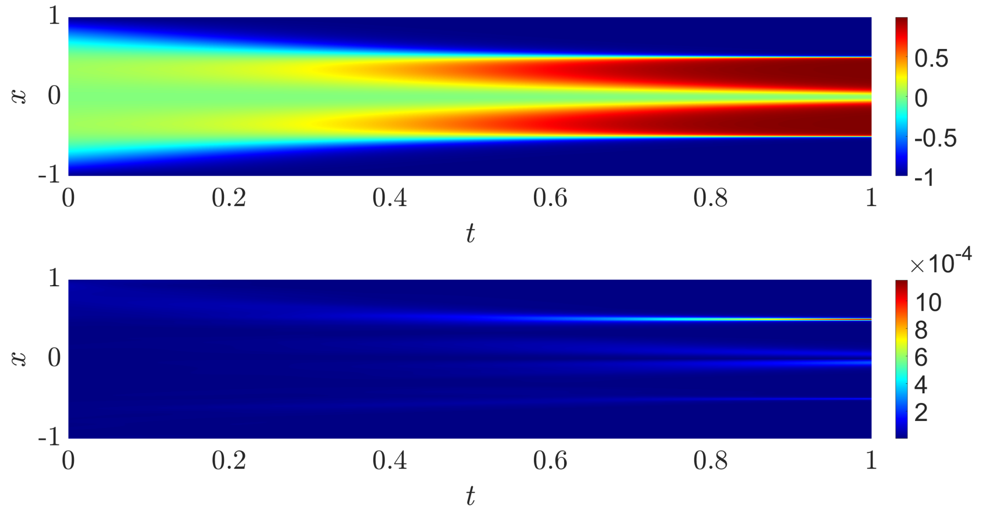
The above equation (25) gives details about the initial and boundary conditions (where Γ describes the boundary).

5.2.1. *std-PINN for Cahn Hilliard equation*

Initially, the Cahn Hilliard equation (18) (without phase space) is solved using the std-PINN to demonstrate the challenge associated with high order. The neural network architecture used has 4 hidden layers with 200 neurons in each layer and which is the same as given in [8]. For training the std-PINN, we have used 20,000 collocation points and the loss function is minimized by using 100,000 ADAM iterations and subsequently L-BFGS optimizer until one of the stopping criteria are met: (i) Maximum iterations are equal to 50,000 (ii) Maximum number of function evaluations are equal to 50,000 (iii) Maximum number of line search steps (per iteration) equal to 50 (iv) The maximum number of variable metric corrections used to define the limited memory matrix are equal to 50 (v) The iteration stops when $\frac{f^k - f^{k+1}}{max(|f^k|,|f^{k+1}|,1)} <= 2.22044604925e-16$,where f is the neural network objective function and k

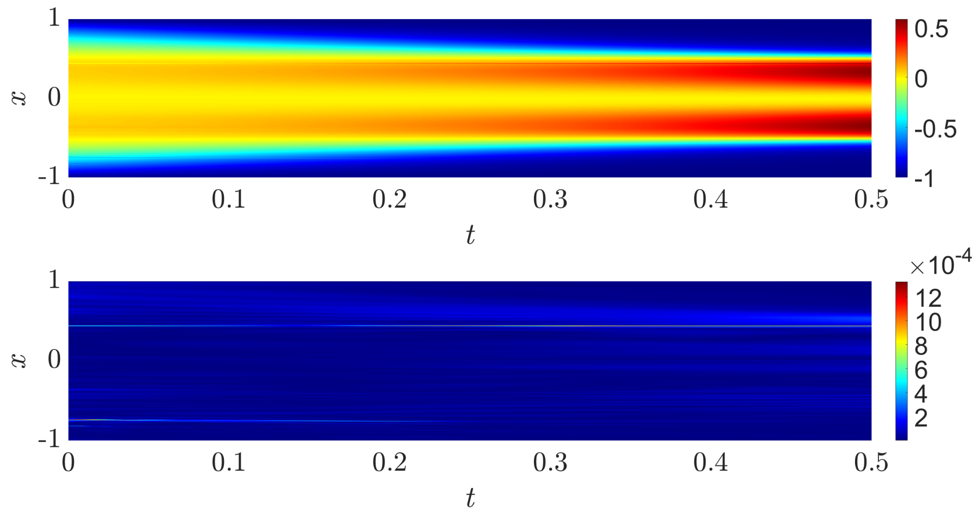


(a) Predicted solution (top) and relative error (bottom) obtained using std-PINN

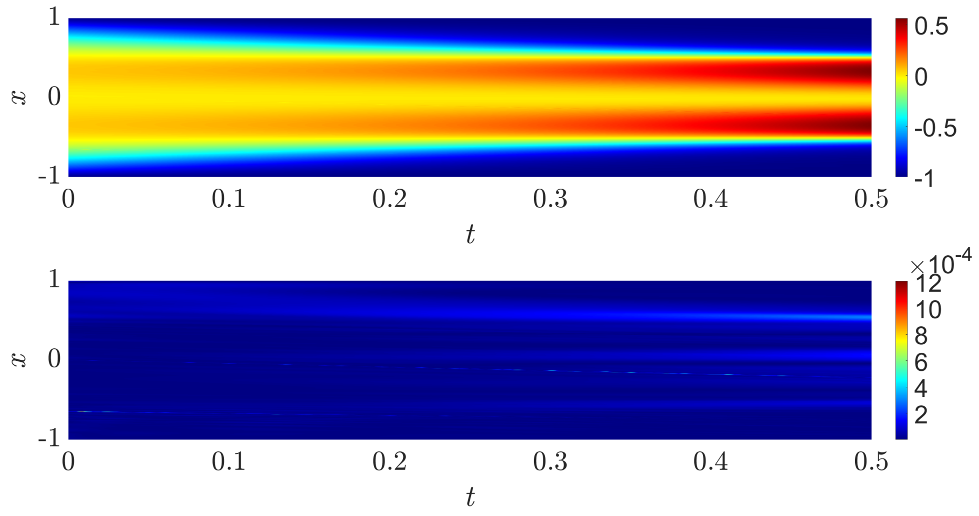


(b) Predicted solution (top) and relative error (bottom) obtained using bc-PINN

Figure 7: Solution and relative error associated with respect to the reference solution for Allen Cahn equation.

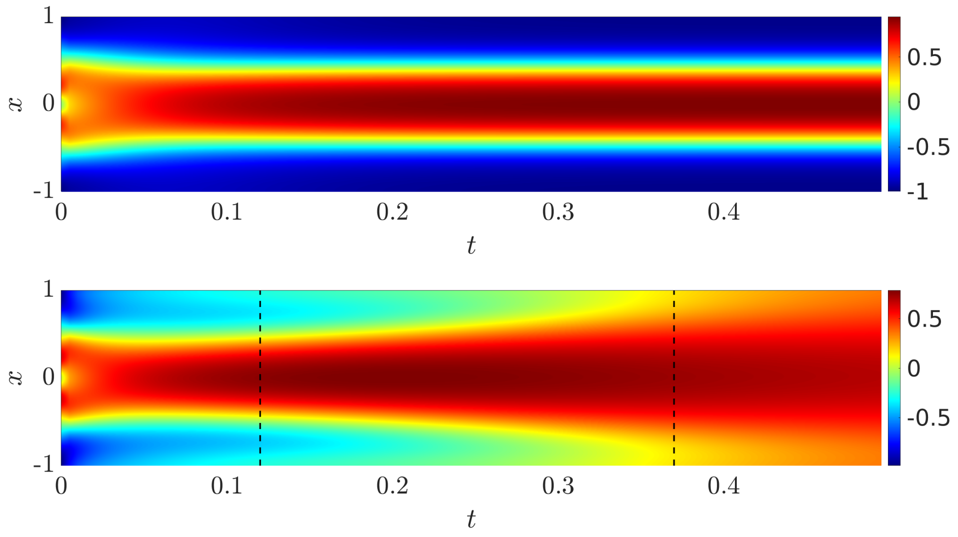


(a) Predicted solution (top) and relative error (bottom) obtained using bc-PINN for $c_1^2 = 0.00001$

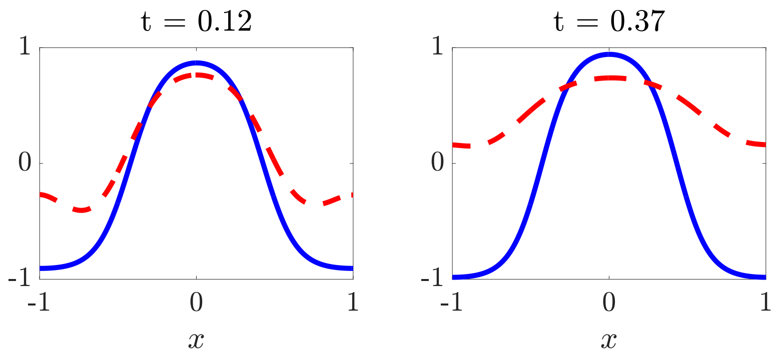


(b) Predicted solution (top) and relative error (bottom) obtained using bc-PINN for $c_1^2 = 0.00005$

Figure 8: Solutions and relative errors of the Allen Cahn equation for different c_1^2 (of equation (13)) obtained by the bc-PINN method.



(a)



(b)

Figure 9: (a): The reference solution (Top) and the std-PINN solution (Bottom) of the Cahn Hilliard equation for the entire spatio–temporal domain. (b): Time snapshots for the reference solution (—) and the std-PINN solution (- - -) at $t = 0.12$ and $t = 0.37$.

is the iteration number. The loss function for std-PINN is described in equation (3), (4) and (5). The solution predicted after training is shown in figure (9) and it can be observed that there is significant mismatch between the std-PINN prediction and the reference solution.

The two possible reasons for the inaccurate solution are strong non-linearity and the high order derivative terms (fourth order). In PINN the derivatives are approximated using automatic differentiation. It has been shown that as the order of the derivative increases the complexity in automatic differentiation increases and it becomes computationally expensive [9]. In order to overcome the difficulty in approximating the higher order derivative via automatic differentiation, we adopt the phase space representation in the proposed bc-PINN.

5.2.2. bc-PINN for Cahn Hilliard equation

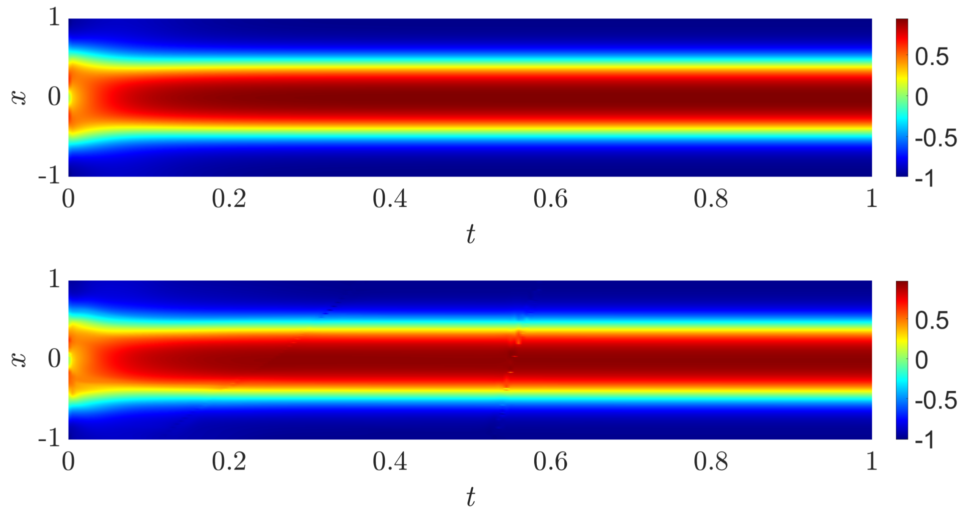
In this section, we introduce the bc-PINN approach with a phase space representation for solving the Cahn Hilliard Equation (equation (19). Additionally we have used the ICGL and TL techniques as described in sections (3.2, 3.3). Therefore, there are two outputs of the neural network $\hat{h}(x,t)$ and $\hat{\mu}(x,t)$ in the present method. The input features are the spatio–temporal variables (x,t). The modified loss function for the coupled phase space system includes an error on initial condition, error on the boundary conditions and error on the residual. In addition it will have the error for the backward compatibility. Therefore, the total loss function (equation (11)) for any time segment ΔT_n is sum of all the aforementioned errors given in equation (20-23).

Variable	Description	Number
N_i	Initial collocation points	128
N_b	Boundary collocation points	20/segment
N_r	Residual collocation points	10000/segment
N_{iter}	Number of ADAM iterations	10000/segment

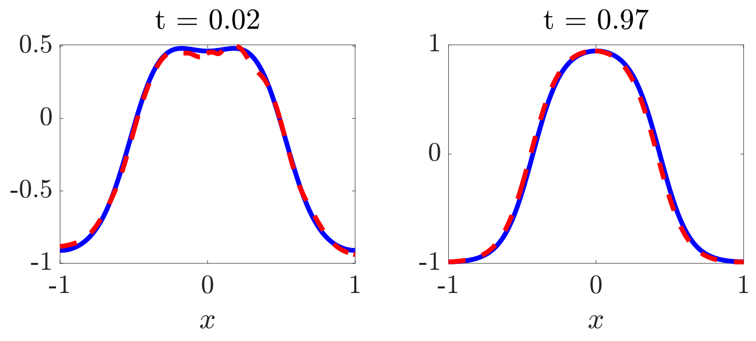
Table 3: Description of training data for Cahn Hilliard equation. 20 time steps/segment have been considered and the amount of collocation points generated remains same and doesn't increase as we progress through time.

Table 3 describes the values of the hyper-parameters used in bc-PINN. N_i and N_b refers to the number of points considered to enforce the initial and boundary condition respectively. N_r is the number of residual collocation points per time segment and N_{iter} is the number of ADAM iterations used to train the neural network per time segment. As described in section 5.1.2, only the reference solution at the initial points (N_i) is used to compute the initial loss and for computing the remaining terms in loss function (equation (20-23)) the output of the neural network (predicted solution of the PDE) is used.

The accuracy of the proposed bc-PINN approach is shown by comparing it against the reference solution obtained by the chebfun method in figure (10). This shows that the phase space representation with bc-PINN can closely match the reference solution for the Cahn Hilliard equation. The relative total error (ε_{total}) obtained for the bc-PINN solution is 0.0186 whereas for the std-PINN solution the error is 0.8594. It is evident from the error plots given in figure (11) that a more accurate solution is obtained by using the bc-PINN compared to std-PINN. The higher accuracy can be accredited to the fact that approximating lower order derivatives using automatic differentiation is much simpler. One key observation to note is that the solution in the n^{th} time segment takes the solution at time T_{n-1} from the $(n-1)^{\text{th}}$ time segment as initial condition. Thus, only the error at the end point in a time segment is propagated to the next time segment. For instance, only the error at the time T_{n-1} in $(n-1)^{\text{th}}$ time segment is propagated to the next time segment. Errors at all other time steps in $(n-1)^{\text{th}}$ time segment does not propagate to the n^{th} time segment. This can be observed in figure (11b), even though the error at time 0.01 is quite high but since this is not the end point of the time segment $[0, 0.05]$ it does not propagate with time. The error in the first time segment can be further reduced by using more iterations and the accuracy of the total solution can be improved. To further demonstrate the effectiveness of the current phase space backward compatible training approach, we have taken different values of the parameter $(\alpha\kappa)$ and compared the predicted solutions with the reference solutions generated using chebfun which is shown in figure (12). The proposed phase space representation with bc-PINN approach can be extended to any partial differential equation consisting higher order derivatives and non-linearity.

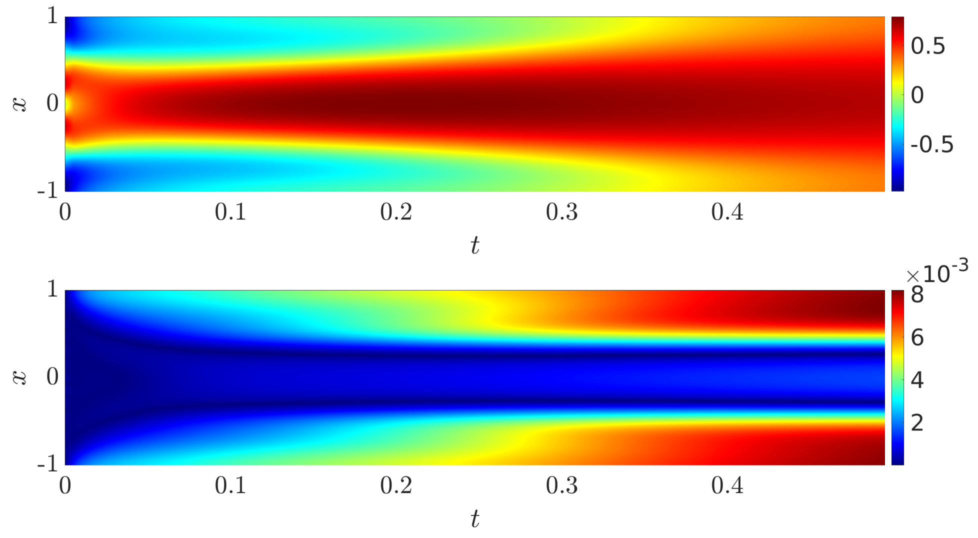


(a)

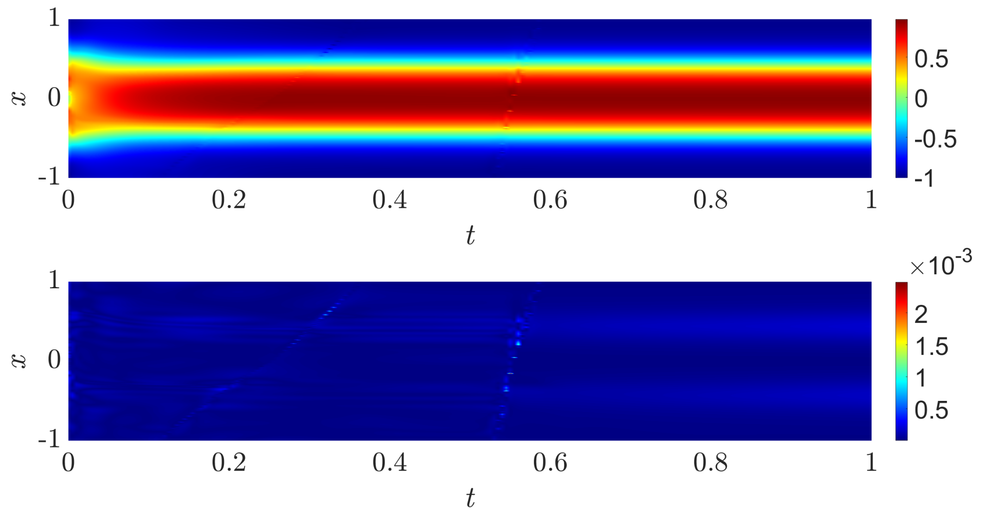


(b)

Figure 10: (a): Reference (Top) and bc-PINN (Bottom) solutions of the Cahn Hilliard equation for the entire spatio–temporal domain. (b): The Reference (—) and the bc-PINN (- - -) solutions at time $t = 0.02$ and $t = 0.97$.

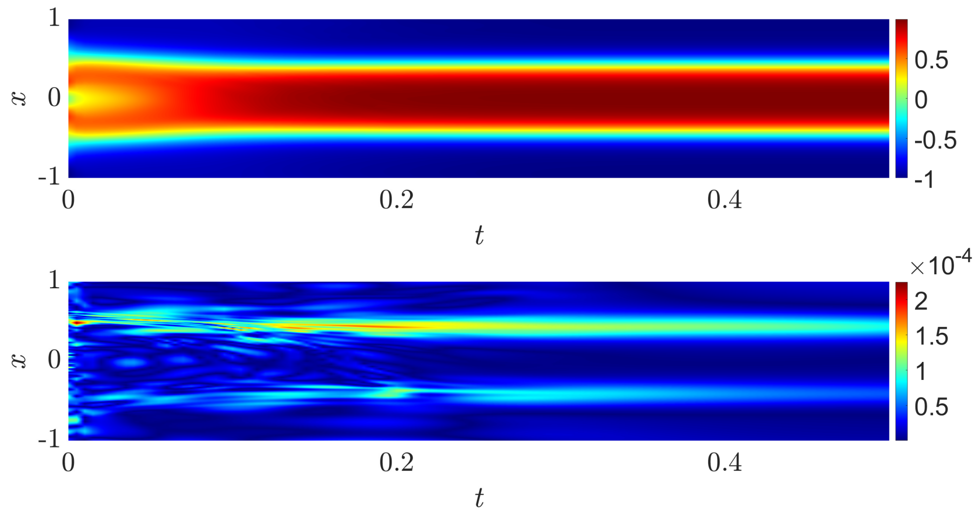


(a) Solution (top) and relative error (bottom) via std-PINN

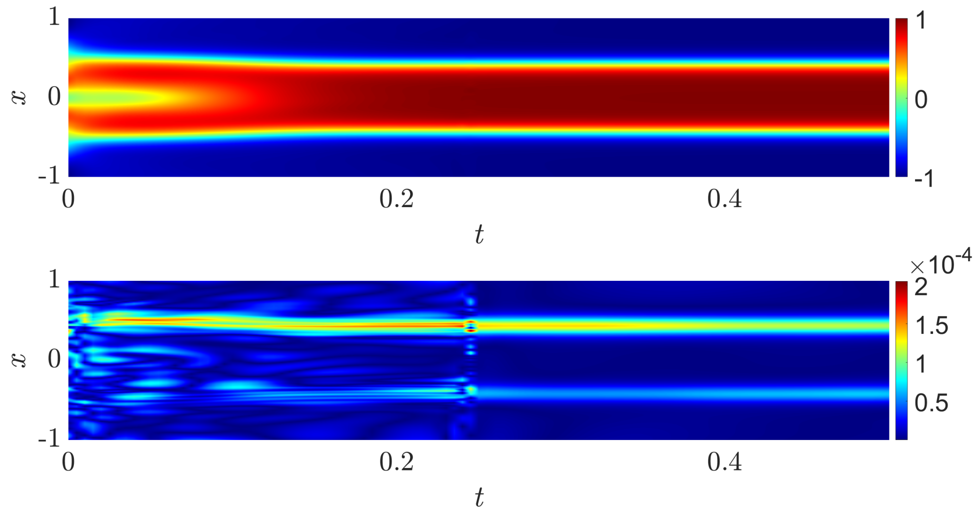


(b) Solution (top) and relative error (bottom) via bc-PINN

Figure 11: Solution and error associated with respect to the reference solution for Cahn Hilliard equation.



(a) Solution (top) and error (bottom) for $\alpha\,\kappa = 0.01$



(b) Solution (top) and error (bottom) $\alpha\,\kappa = 0.005$

Figure 12: Solution and relative errors of the Cahn Hilliard equation for different parameters ($\alpha\,\kappa$ of equation (18)) obtained by the bc-PINN method.

6. Results of Allen Cahn and Cahn Hilliard equations using bc-PINN in 2D

In this section, the Allen Cahn and Cahn Hilliard equations are solved in two dimensions using the proposed bc-PINN scheme. Solving the Allen Cahn and Cahn Hilliard equations in 2D is more computationally intensive than in 1D. Thus in order to solve the 2D equations, the bc-PINN technique has been implemented along with the ICGL and TL techniques as proposed in section (3).

6.1. Allen Cahn equation in two dimensions

In this section, a two dimensional time varying Allen Cahn equation has been considered. The PDE for Allen Cahn equation remains the same as described in section (4.1). The values of the parameters considered in equation (13) are, $c_1^2 = 0.0001$ and $c_2 = 1$. To demonstrate the proposed method following two IBVPs have been considered :

IBVP-1: The PDE used is the same as that given in equation (13). The domain for the IBVP is taken as $\boldsymbol{x} \times t \in [0,1]^2 \times (0,1]$. The initial condition chosen is, $\sin(4\pi x_1)\ \cos(4\pi x_2)$, where, (x_1, x_2) are points in the domain $[0,1]^2$.The boundary conditions have been considered to be periodic, $h^{(d-1)}(\boldsymbol{x}, t) = h^{(d-1)}(-\boldsymbol{x}, t)$, for $d = 1, 2$

IBVP-2: The PDE used is the same as that given in equation (13). The domain for the IBVP is taken as $\boldsymbol{x} \times t \in [0,1]^2 \times (0,2]$. The initial condition chosen is a random doubly periodic function where the maximum amplitude is 0.3. The boundary conditions have been considered to be periodic, $h^{(d-1)}(\boldsymbol{x}, t) = h^{(d-1)}(-\boldsymbol{x}, t)$, for $d = 1, 2$

6.1.1. bc-PINN for Allen Cahn equation in two dimensions

First in order to solve IBVP-1, the ICGL technique has been using along with bc-PINN. The loss function for ICGL and bc-PINN remains the same as described in equations (12, 14-17). Furthermore, as described in section (3.6), Chebfun is used to obtain the reference solution. Here, the spatial domain has been discretized into a grid containing 64 points along each axis and the temporal domain has been discretized into 101 grid points. The neural network architecture has 3 input neurons and consists of 6 hidden layers with 128 neurons in each layer. The output layer contains only one neuron for the output/solution of the PDE (h). Following, to solve the IBVP-1, bc-PINN is used along with with the ICGL technique. The loss function has been optimized using both ADAM and L-BFGS optimizers. 15,000 ADAM iterations (20% for MSE_{SI} and 80% for $MSE_{\Delta T_n}$) and 10,000 LBFGS iterations are used for every time segment. Apart from the maximum iterations other stopping criteria for L-BFGS remain the same as those mentioned in section (3.4) (criteria ii - v). The total number of collocation points used to train the bc-PINN for IBVP-1 are given in table 4. The total error in prediction (ε_{total}), using bc-PINN with ICGL is 2.5%. Whereas, the total error in prediction using bc-PINN without ICGL is more than 95%. Figure (13) shows the evolution of solution as time increases.

N_i	Initial collocation points	4096
N_b	Boundary collocation points	640 per segment
N_r	Residual collocation points	20,000 per segment

Table 4: Description of training data for 2D Allen Cahn equation (IBVP - 1). 20 time steps/segment have been considered and the amount of collocation points generated remains same and doesn't increase as we progress through time.

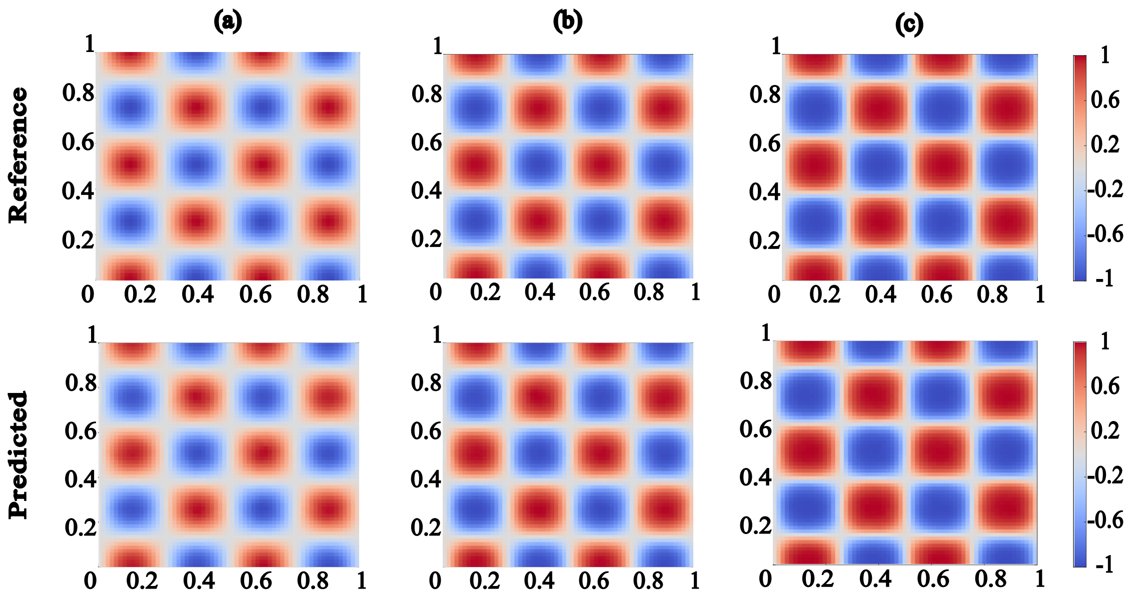


Figure 13: Reference and bc-PINN predicted solution of the 2D Allen Cahn equation (IBVP-1) at different time snapshots (a) t = 0 , (b) t = 0.5 , (c) t = 1

To demonstrate bc-PINN with (*ICGL* and *TL*), IBVP-2 has been considered. The loss function for ICGL, initial condition, and boundary condition remains the same as described in equations (12, 14-17). The total loss function also remains the same as given in equation (11). The optimization of the loss function is performed using both ADAM and L-BFGS optimizers. For minimizing the loss function in the first segment 30,000 ADAM iterations and 10,000 LBFGS iterations are used. From the second segment 40,000 ADAM iterations and 10,000 LBFGS iterations are used in optimizing the bc-PINN loss function. Apart from the maximum iterations other stopping criteria for L-BFGS are the same as mentioned in section (3.4) (criteria ii - v). Also, during the minimization process in each time segment, 20% of the total ADAM iterations are used to implement the ICGL technique. The neural network architecture remains the same as that used for solving IBVP - 1 (3 input neurons, 6 hidden layers with 128 neurons, and 1 output). Following, the TL technique has been implemented by freezing the parameters obtained after solving the IBVP-1. Initially, only a single layer is fixed. As the training progresses through different time segments the number of layers frozen has been slowly incremented. Table (5) gives details about the number of collocation points and the time segment details. From figure (14) it can be observed that the bc-PINN prediction matches well with the reference solution, whereas the std-PINN cannot solve this using the same number of collocation points.

N_i	Initial collocation points	4096
N_b	Boundary collocation points	1600 per segment
N_r	Residual collocation points	30,000 per segment

Table 5: Description of training data for 2D Allen Cahn equation (IBVP - 2). 50 time steps/segment have been considered and the amount of collocation points generated remains same and doesn't increase as we progress through time.

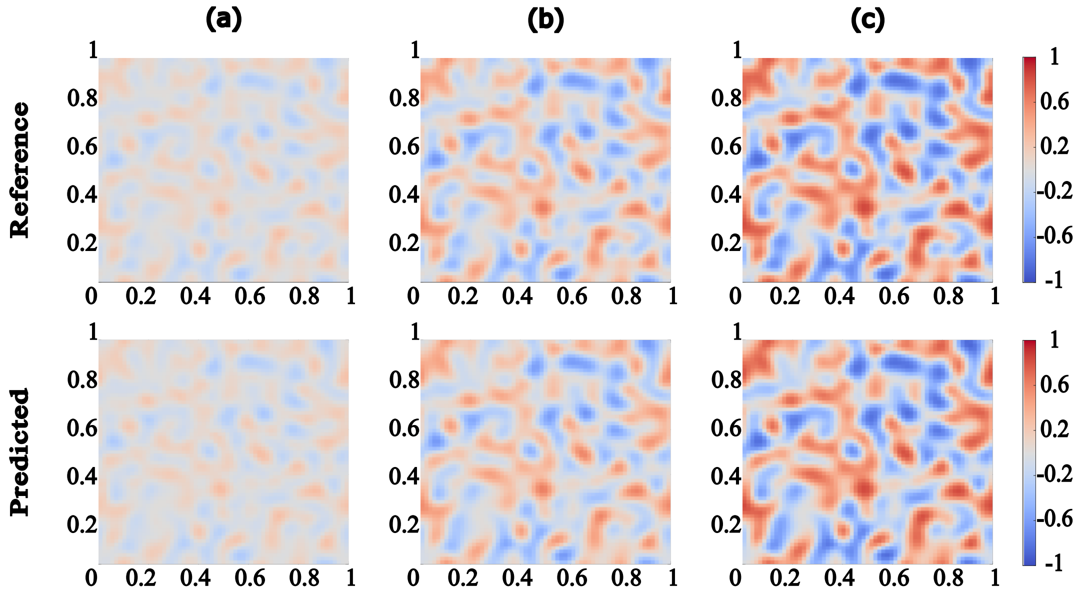


Figure 14: Reference and bc-PINN predicted solution of the 2D Allen Cahn equation (IBVP-2) at different time snapshots (a) t = 0 , (b) t = 1.2 , (c) t = 2

6.2. Cahn Hilliard equation in two dimensions

In this section, a two dimensional time varying Cahn Hilliard equation has been considered. The PDE for the Cahn Hilliard equation remains the same as described in section (4.2). The values of the parameters considered in equation (19) are, $(\alpha, \kappa) = (0.02, 1)$. To demonstrate the applicability of bc-PINN for 2D Cahn Hilliard, the following two IBVP's has been considered:

IBVP-1: The PDE used is the same as that given in equation (19). The domain for the IBVP is taken as $\boldsymbol{x} \times t \in [0,1]^2 \times (0, 0.005]$. The initial condition chosen is, $0.4\,\cos(3\pi x_1)\,\cos(3\pi x_2)$, where, (x_1, x_2) are points in the domain $[0,1]^2$. Homogeneous Neumann boundary conditions have been considered.

IBVP-2: The PDE used is the same as that given in equation (19). The domain for the IBVP is taken as $\boldsymbol{x} \times t \in [0,1]^2 \times (0, 0.005]$. The initial condition chosen is, $0.4\,\cos(4\pi x_1)\,\cos(4\pi(x_1 + x_2))$, where, (x_1, x_2) are points in the domain $[0,1]^2$. Periodic boundary conditions have been considered.

IBVP-3: The PDE used is the same as that given in equation (19). The domain for the IBVP is taken as $\boldsymbol{x} \times t \in [0,1]^2 \times (0, 0.00375]$. The initial condition chosen is a random doubly periodic function where the maximum amplitude is 0.5. Periodic boundary conditions have been considered.

6.2.1. bc-PINN for Cahn Hilliard equation in two dimensions

As described in section (3), the ICGL technique has been used along with bc-PINN to solve IBVP-1. A key point to be noted here is that the loss function for ICGL remains the same as that in equation (12). Therefore, the total loss function (equation (11)) for any time segment ΔT_n is a sum of all the aforementioned errors given in equation (20–23).

Further in order to generate the reference solution for IBVP-1, a cosine transform has been utilized as given in [56]. Here, the spatial domain is discretized into a grid containing 64 points along each axis. The temporal domain is discretized into 201 grid points. The neural network architecture has 3 input neurons and consists of 5 hidden layers with 128 neurons in each layer. Two output neurons have been taken to represent the output/solution of the PDE and the phase space term (h, μ). The minimization of bc-PINN loss function has been performed using ADAM and L-BFGS optimizers. The number of ADAM iterations are 30,000 for the first segment and for all the later segments 50,000 iterations have been used. The maximum number of L-BFGS iterations are 15,000 per segment and the other stopping criteria are the same as those mentioned in section (3.4) (criteria ii - v). The details of collocation points used to train the bc-PINN for IBVP1 are given in table (6). Figure (15), shows the evolution of $h(\boldsymbol{x}, t)$ with time and there's a good match between the predicted and reference solution.

N_i	Initial collocation points	4096
N_b	Boundary collocation points	1600 per segment
N_r	Residual collocation points	50,000 per segment

Table 6: Description of training data for 2D Cahn Hilliard equation (IBVP-1). 50 time steps/segment have been considered and the amount of collocation points generated remains same and doesn't increase as we progress through time.

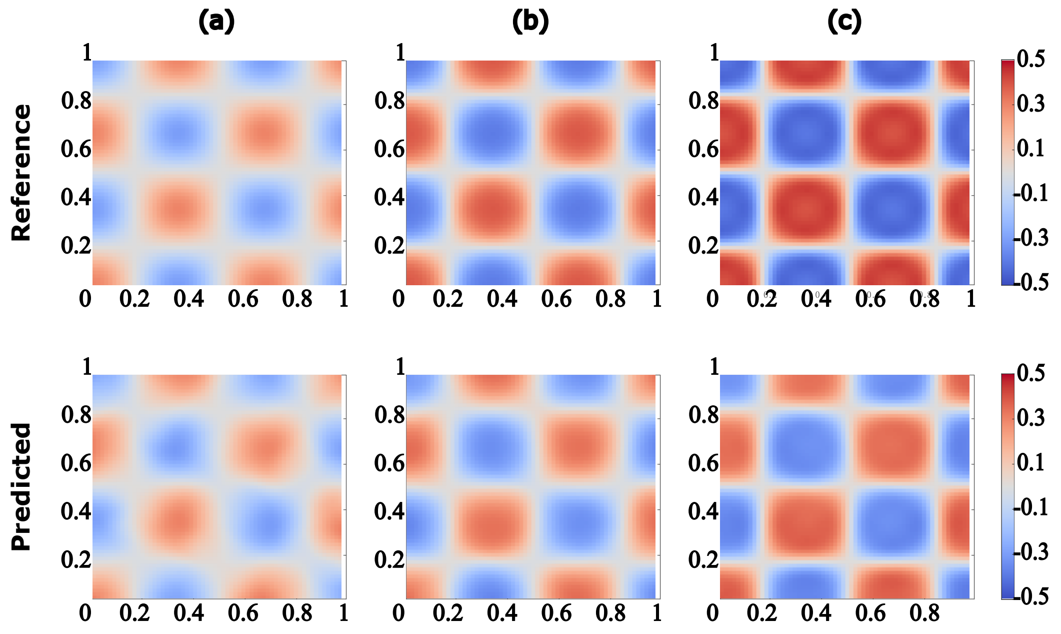


Figure 15: Reference and bc-PINN predicted solution of the 2D Cahn Hilliard equation (IBVP-1) at different time snapshots (a) t = 0 , (b) t = 0.003 , (c) t = 0.005

In order to demonstrate bc-PINN with (*ICGL* and *TL*), IBVP-2 has been considered. To solve the IBVP-2, the loss function for ICGL, initial condition and boundary condition remain the same as described in equations (12, 20-23, 11). To generate the reference solution for IBVP-2 and IBVP-3, an explicit time stepping scheme is used for time integration and a 9-stencil finite difference method for computing the spatial derivatives. Here, the spatial domain is discretized

N_i	Initial collocation points	4096
N_b	Boundary collocation points	1600 per segment
N_r	Residual collocation points	50,000 per segment

Table 7: Description of training data for 2D Cahn Hilliard equation (IBVP-2,IBVP-3). 25 time steps/segment have been considered and the amount of collocation points generated remains same and doesn't increase as we progress through time.

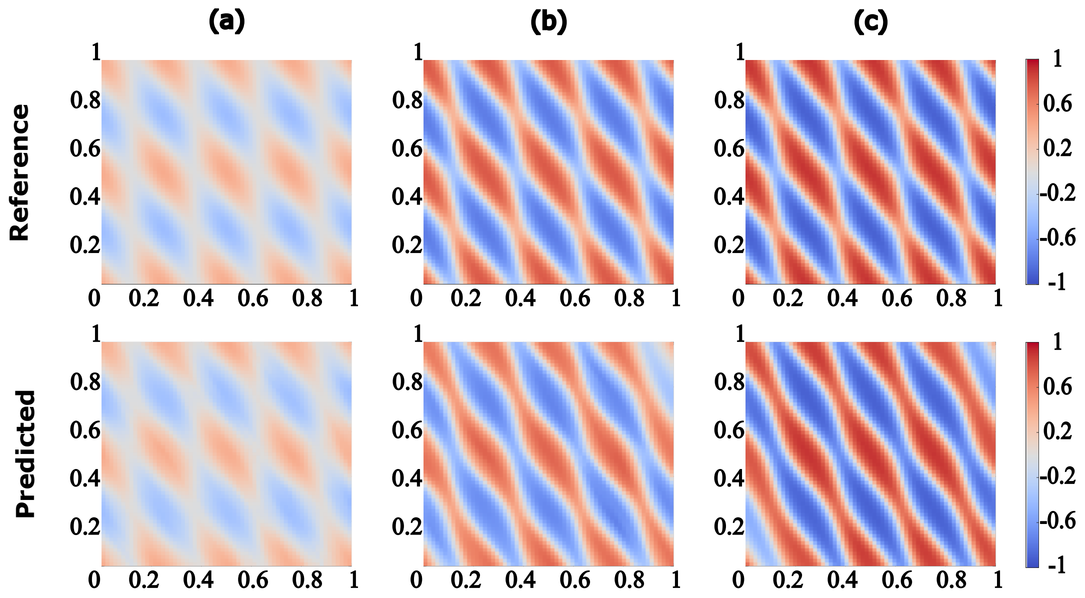


Figure 16: Reference and bc-PINN predicted solution of the 2D Cahn Hilliard equation (IBVP-2) at different time snapshots (a) t = 0 , (b) t = 0.003 , (c) t = 0.005

into a grid containing 64 points along each axis, whereas, the temporal domain is discretized into 101 grid points. The neural network architecture has 3 input neurons and consists 5 hidden layers with 128 neurons in each layer. Two output neurons have been taken to represent the output/solution of the PDE and the phase space term (h, μ). The neural network architecture remains the same as that used for solving IBVP - 1 (3 input neurons, 5 hidden layers with 128 neurons and 2 outputs). Following, the TL technique has been implemented for IBVP2 by freezing the parameters obtained after solving the first segment of IBVP-1. Initially, only a single layer is fixed and as the training progresses through different time segments the number of layers frozen has been incremented by one after every time segment. The minimization of bc-PINN loss function for IBVP-2 has been performed using ADAM and L-BFGS optimizers. The number of ADAM iterations are 50,000 for all the segments. The maximum number of L-BFGS iterations are 75,000 per segment and the other stopping criteria are same as those mentioned in in section (3.4) (criteria ii - v). The details of collocation points used to train the bc-PINN for IBVP-2 are given in table (7). Figure (16) shows the evolution of $h(\boldsymbol{x}, t)$ with time.

Despite using all the aforementioned techniques (ICGL, TL and weighting of the loss function) the solution doesn't converge efficiently for IBVP-3. This is because the initial condition in this case is random and the evolution is complex. Therefore, we have used a sum of $\mathcal{L}^1$ norm and $\mathcal{L}^2$ norm of the individual error terms in the total loss function. The reason behind using such a

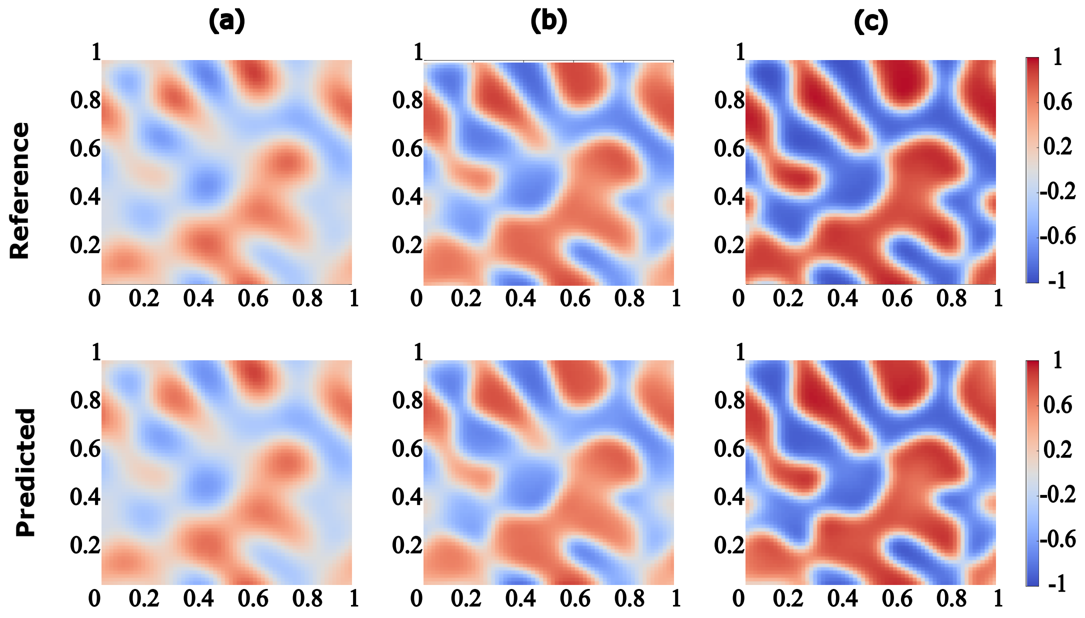


Figure 17: Reference and bc-PINN predicted solution of the 2D Cahn Hilliard equation (IBVP-3) at different time snapshots (a) t = 0 , (b) t = 0.002 , (c) t = 0.005

loss function is that, when the error is small the penalization due to $\mathcal{L}^1$ norm is more than $\mathcal{L}^2$ norm. Thus, as given in equation (26) a combination of $\mathcal{L}^1$ norm + $\mathcal{L}^2$ norm has been used to solve the IBVP-3.

$$\begin{aligned} \text{MSE}_{\Delta T_n} = w_i\left(\text{MSE}_I + \text{MAE}_I\right) + w_b\left(\text{MSE}_B + \text{MAE}_B\right) + \\ w_r\left(\text{MSE}_R + \text{MAE}_R\right) + w_s\left(\text{MSE}_S + \text{MAE}_S\right) \end{aligned} \tag{26}$$

Here, MAE_I, is the mean absolute error on the initial condition ; MAE_B, is the mean absolute error on the boundary condition; MAE_R, is the mean absolute error on the residual (R_1, R_2) as given in equations (22); MSE_S; is the mean absolute error on backward compatibility;

Other hyper–parameters like neural network architecture, number of training iterations, number of collocation points etc. remain the same as those used for solving IBVP-2. The only difference is that the TL technique has been implemented for IBVP-3, by using the parameters obtained after solving the first segment of IBVP-2. Considering the complexity of the Cahn Hilliard equation from figure (17) it can be observed that the bc-PINN prediction matches the reference solution remarkably well.

7. Conclusions

A new PINN approach (named as bc-PINN) has been proposed for solving the Allen Cahn and Cahn Hilliard equations, however, the methodology in general should be applicable to any PDE. The bc-PINN re-trains the neural network over successive time segments while satisfying the solution for all previous time segments. Additionally, bc-PINN incorporates two new techniques to improve the accuracy and efficiency of training. Firstly, while solving a boundary value problem using bc-PINN, the initial condition of that segment is used to bring the neural network map closer to the true map. Secondly, a transfer learning approach is implemented to accelerate training, where the parameters learned from previous training are used to train a subsequent segment or a new initial condition. In addition by using a phase space representation for the Cahn Hilliard equation, better convergence has been achieved.

The key advantages of bc-PINN are summarized below. The proposed bc-PINN method can provide an accurate solution for nonlinear or higher–order PDEs such as the Cahn Hilliard and Allen Cahn equations in 1D and 2D. Moreover, the proposed method can achieve high accuracy by using fewer collocation points compared to std-PINN. Despite the segmentation of the time domain, it uses only one neural network and provides a continuous solution for the entire spatio–temporal domain. The proposed backward compatibility scheme may enhance many other machine learning approaches applied to complex systems represented by time dependent PDEs.

The code accompanying this manuscript would be available on Github repository:
`https://github.com/vmattey/bc-PINN`

Acknowledgments: SG acknowledges the financial support by NSF (CMMI MoMS) under grant number 1937983. We acknowledge Superior, a high-performance computing facility at MTU and Google Colab, a cloud service hosted by Google. This work used the Extreme Science and Engineering Discovery Environment (XSEDE) (allocation number MSS200004), which is supported by the NSF grant number ACI-1548562. RM acknowledges Ponkrshnan Thiagarajan and Shashank Pathrudkar for their valuable discussions and insights on the topic.

Appendix A. Hyper-parameter selection for bc-PINN

As discussed in section 5 and 6, the proposed method has a number of hyper-parameters like number of ADAM iterations (N_{iter} per segment), time steps per segment, number of collocation points (N_r) etc. The accuracy of the bc-PINN's solution depends on proper choice of these hyper-parameters. To optimize each of the hyper-parameters, various cases and metrics like computational time and accuracy have been considered. In the current section, we chose the Cahn Hilliard equation as the canonical example for all the analysis performed. Table (A.8) describes the optimum parameters required to train 100 steps. The optimum parameters are chosen to achieve an accurate solution while balancing the computational cost as shown in figure (A.18). It can be also seen that as the number of collocation points and number of iterations are increased the accuracy increases. Therefore, a segment size of 10 steps with 5000 collocation points and 10000 iterations per time segment has been chosen.

In general the bc-PINN framework is robust enough to handle any arbitrary length of time segments and any number of time segments. One effective way for choosing a minimum time segment is to try training the bc-PINN initially for a specific length of time segment where the accuracy doesn't get affected. Based on this minimum length of time segment the total number of time segments can be chosen depending on the length of the total time domain.

Model	**Time steps/segment**	N_r	N_{iter}
A	10	5000	10000
B	10	5000	20000
C	10	10000	10000
D	10	10000	20000
E	25	5000	10000
F	25	5000	20000
G	25	10000	10000
H	25	10000	20000

Table A.8: Parameter combinations for choosing the optimum segment size, collocation points and number of ADAM iterations to apply the bc-PINN technique for Cahn Hilliard equation.

Appendix B. bc-PINN with a logarithmic residual for Allen Cahn Equation

In this section we show how the bc-PINN with logarithmic residual compares against the std-PINN and bc-PINN without the logarithmic residual. The loss function for the bc-PINN with a logarithmic residual is same as the bc-PINN except the equation(16) is replaced by the following loss term for the residual of the PDE:

$$R := \hat{h}_t - c_1^2\,\nabla^2\hat{h} + f(\hat{h})$$
$$\mathrm{MSE}_R^{(\ln)} = \frac{1}{N_r}\sum_{k=1}^{N_r} \ln\left(1 + (R(x_k^r, t_k^r))^2\right), \qquad (x_k^r, t_k^r) \in \Omega \times (T_{n-1}, T_n] \tag{B.1}$$

It turns out that the bc-PINN with a logarithmic residual is more accurate than the bc-PINN without the logarithmic residual. A possible explanation is that the logarithmic function reduces the relative weight on the MSE_R, which would have larger inaccuracy due to its derivative and nonlinear terms. Thus the initial and boundary terms are satisfied more accurately, which in

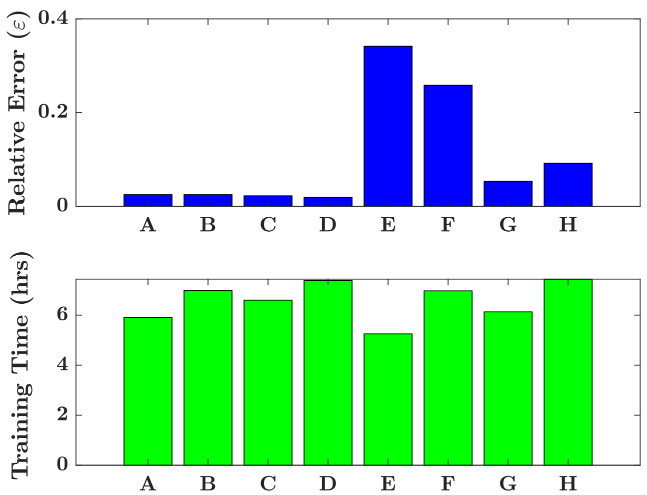
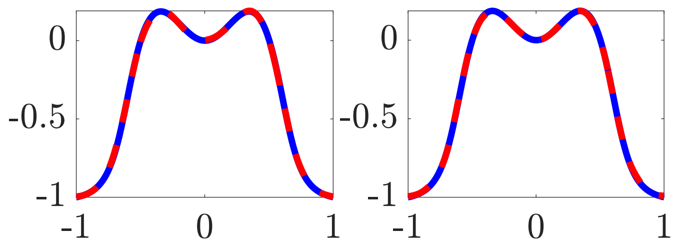


Figure A.18: Relative error (ε) and time taken for various models given in table (A.8).



(a) bc-PINN (b) bc-PINN with log residual

Figure B.19: Reference (—) and Predicted (- - -) solution at time t = 0.25

Method	**Error**(ε)
std-PINN	0.9919
bc-PINN	0.0701
bc-PINN with logresidual	0.03

Table B.9: Relative errors (equation (7)) over the entire domain with respect to Chebfun solution for different methods.

turn yields a more accurate solution. This explanation is substantiated by the fact that when the logarithmic function is used on all of the four terms in the loss function then the accuracy decreases.

Appendix C. Minimization of the bc-PINN loss function

In section 3.4, we have described the learning rates and stopping criteria for the ADAM and LBFGS optimizer that are utilized to train the bc-PINN. Figure (C.20) shows the minimization of the loss function (equation (11)) while training the bc-PINN for 1D Cahn Hilliard equation in time segment $[0.45, 0.5]$.

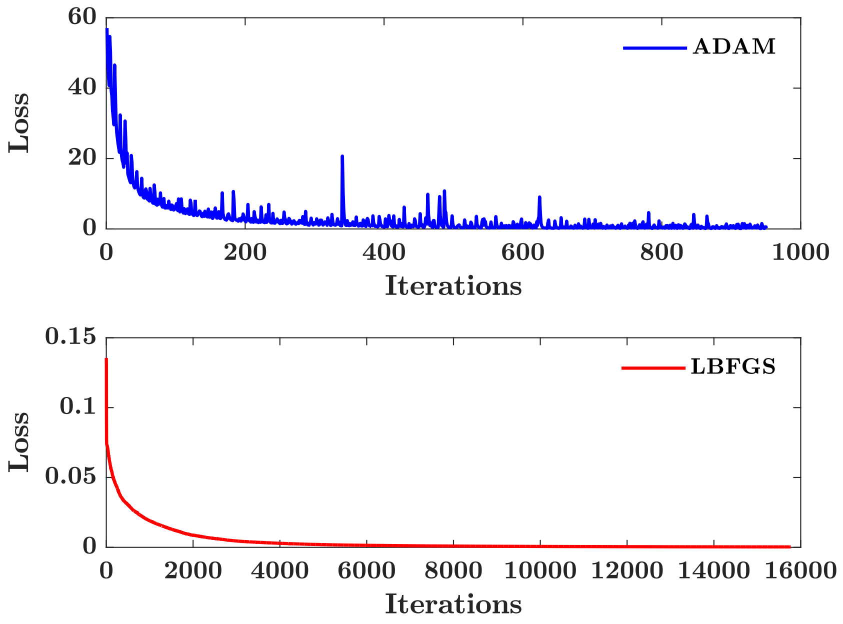


Figure C.20: Loss vs Iterations (Top): using ADAM optimizer (Bottom): using the LBFGS optimizer for training the time segment $[0.45, 0.5]$ of the Cahn Hilliard equation.

Appendix D. Data driven identification of parameters for PDEs using bc-PINN

The framework of bc-PINN is versatile and can be applied to both forward and inverse problems. Data driven methods for inverse problems have been shown to be extremely effective [57, 24, 58, 59]. Especially in situations where only partial physics and noisy data are present physics informed neural networks perform well. In this section, the performance of std-PINN [8] and bc-PINN for non-parametric PDEs and parametric PDEs has been showcased. For the non-parametric PDE case, Allen Cahn equation which has been described in section 4 will be considered as the canonical example. Finally for the parametric PDE, burgers equation with a time-varying parameter has been considered.

Consider a general, m^{th} order partial differential equation with parameters λ_1, λ_2 etc. as given in equation (D.1).

$$h_t = F(h(\boldsymbol{x},t), h^{(1)}_{\boldsymbol{x}}(\boldsymbol{x},t)\,, h^{(2)}_{\boldsymbol{x}}(\boldsymbol{x},t), \cdots, h^{(m)}_{\boldsymbol{x}}(\boldsymbol{x},t), \lambda_1, \lambda_2, \cdots, \lambda_n)\,,\ \boldsymbol{x} \in \Omega \subset \mathbb{R}^D\,,\ t\ \in\ (0,T] \tag{D.1}$$

In equation (D.1), we are interested in finding the parameters $\lambda_1, \lambda_2 \cdots \lambda_n$. These parameters are learnt by defining them as trainable parameters to the physics informed neural networks. Therefore the loss function can be written as

- Mean squared error on the observed data

$$\text{MSE}_o = \frac{1}{N_o}\sum_{k=1}^{N_o}\left(\hat{h}(\boldsymbol{x}_k^o, t_k^o) - h_k^o\right)^2, \quad (\boldsymbol{x}_k^o, t_k^o) \in \Omega \times (0, T] \tag{D.2}$$

- Mean squared error of the residual at observed data points

$$\begin{aligned} R &:= h_t - F(h(\boldsymbol{x},t), h_{\boldsymbol{x}}^{(1)}(\boldsymbol{x},t), h_{\boldsymbol{x}}^{(2)}(\boldsymbol{x},t), \cdots, h_{\boldsymbol{x}}^{(m)}(\boldsymbol{x},t), \lambda_1, \lambda_2, \cdots, \lambda_n) \\ \text{MSE}_{oR} &= \frac{1}{N_o}\sum_{k=1}^{N_o}\left(R(\boldsymbol{x}_k^o, t_k^o)\right)^2, \qquad (\boldsymbol{x}_k^o, t_k^o) \in \Omega \times (0, T] \end{aligned} \tag{D.3}$$

  where $\hat{h}(\boldsymbol{x}_k^o, t_k^o)$ is the neural network output and h_k^o is the observed data at $(\boldsymbol{x}_k^o, t_k^o)$. Here, the superscript, $(\bullet)^o$ stands for observed data.

- The total mean squared error for the inverse std-PINN is given as

$$\text{MSE} = \text{MSE}_o + \text{MSE}_{oR} \tag{D.4}$$

The equations (D.2,D.3,D.4) represent the total loss function of std-PINN for inverse identification of parameters. In order to implement the bc-PINN scheme for inverse problems, the domain has been divided into multiple segments as shown in figure (D.21). To identify the parameters in a particular time segment, the residual has been minimized only on the observed data in that particular segment. Additionally, to implement the backward compatibility scheme the solution of all the previous segments is satisfied simultaneously. Moreover, the parameters for n^{th} time segment are initialized from the parameters learned in the $(n-1)^{th}$ time segment. This framework helps in identifying the parameters of the PDE in a given segment and also learning the solution in the entire domain. The loss function for parameter identification using the bc-PINN framework is as follows:

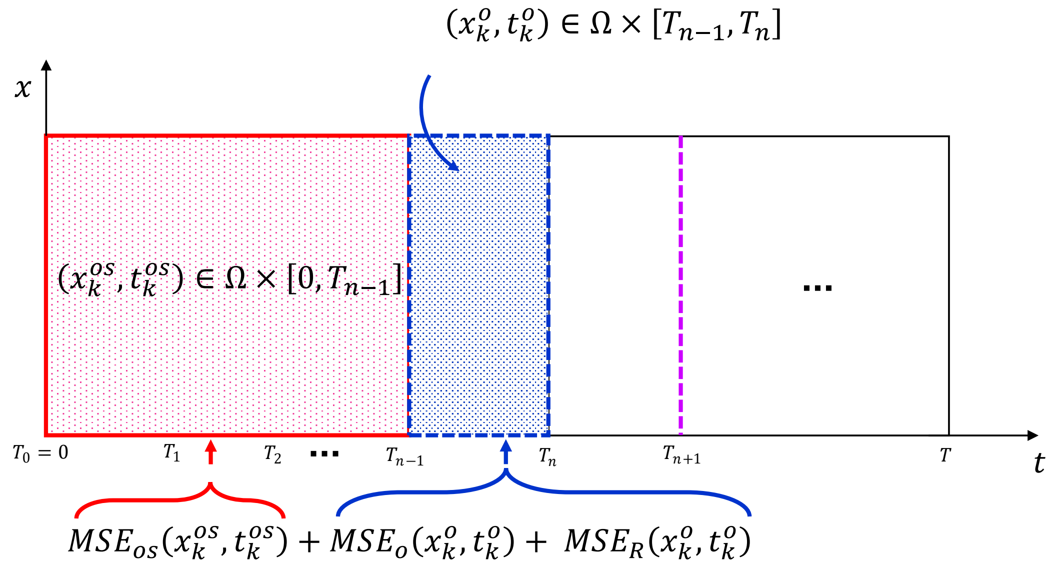

$(x_k^o, t_k^o) \in \Omega \times [T_{n-1}, T_n]$
x
$(x_k^{os}, t_k^{os}) \in \Omega \times [0, T_{n-1}]$
...
$T_0 = 0$
T_1
T_2
...
T_{n-1}
T_n
T_{n+1}
T
t
$MSE_{os}(x_k^{os}, t_k^{os}) + MSE_o(x_k^o, t_k^o) + MSE_R(x_k^o, t_k^o)$

Figure D.21: Illustration of the proposed backward compatibility scheme that satisfies the data observed in the previous time segments at the same time learning the observed data and parameters in a particular segment.

- Mean squared error on the observed data

$$\text{MSE}_o = \frac{1}{N_o}\sum_{k=1}^{N_o}\left(\hat{h}(\boldsymbol{x}_k^o, t_k^o) - h_k^o\right)^2 , \quad (\boldsymbol{x}_k^o, t_k^o) \in \Omega \times [T_{n-1}, T_n] \tag{D.5}$$

- Mean squared error of the residual at observed data points

$$\begin{aligned} R &:= h_t - F(h(\boldsymbol{x},t), h_{\boldsymbol{x}}^{(1)}(\boldsymbol{x},t), h_{\boldsymbol{x}}^{(2)}(\boldsymbol{x},t), \cdots, h_{\boldsymbol{x}}^{(m)}(\boldsymbol{x},t), \lambda_1, \lambda_2, \cdots, \lambda_n) \\ \text{MSE}_{oR} &= \frac{1}{N_o}\sum_{k=1}^{N_o}\left(R(\boldsymbol{x}_k^o, t_k^o)\right)^2 , \qquad (\boldsymbol{x}_k^o, t_k^o) \in \Omega \times [T_{n-1}, T_n] \end{aligned} \tag{D.6}$$

- Mean squared error on the previous segments observed data

$$\text{MSE}_{os} = \frac{1}{N_{os}}\sum_{k=1}^{N_{os}}\left(\hat{h}(\boldsymbol{x}_k^{os}, t_k^{os}) - h_k^{os}\right)^2 , \quad (\boldsymbol{x}_k^{os}, t_k^{os}) \in \Omega \times [0, T_{n-1}] \tag{D.7}$$

  where $\hat{h}(\boldsymbol{x}_k^{os}, t_k^{os})$ is the neural network output and h_k^{os} is the observed data at $(\boldsymbol{x}_k^{os}, t_k^{os})$. Here, the superscript, $(\bullet)^{os}$ stands for observed data in the previous time segments.

- Total mean squared error for inverse bc-PINN is given as

$$\text{MSE}_{\Delta T_n} = \text{MSE}_o + \text{MSE}_{oR} + \text{MSE}_{os} \tag{D.8}$$

Appendix D.1. Parameter identification of Allen Cahn equation 1D

The Allen Cahn as described in earlier sections is a very widely used PDE in material science for studying the diffusion separation process. Therefore inverse identification of the Allen Cahn equation is essential to understand the governing physics of a process. Let us consider the explicit form of Allen Cahn equation

$$h_t = \lambda_1 \nabla^2 h + \lambda_2 (h^3 - h), \qquad t \in (0, T], \quad x \in \Omega \subset \mathbb{R} \tag{D.9}$$

Using the proposed framework of bc-PINN for inverse problems, the parameters learned are compared against std-PINN. To generate the required data set, random points have been sampled from the reference solution of Allen Cahn equation. The reference solution is computed as given in section (3.6). Also to test the effectiveness of both the methods noise has been added to the reference solution. Table (D.10) shows the parameter values obtained using std-PINN and bc-PINN.

	Predicted λ_1 (True value : 1e-04)		Predicted λ_2 (True value : 5)		Total relative error in the predicted solution	
Noise (%)	bc-PINN	PINN	bc-PINN	PINN	bc-PINN	PINN
0	1.504e-04	1.522e-04	5.03237	5.01081	0.0068	0.0062
2	1.375e-04	7.592e-05	4.98174	4.98214	0.0078	0.0070

Table D.10: True and Predicted parameters using bc-PINN and std-PINN for the 1D Allen Cahn equation.

Appendix D.2. Parameter identification of Burgers equation with a time-varying parameter

This section highlights the ability of the proposed bc-PINN framework for solving parametric PDEs. Most of the real world systems are governed by parametric PDEs. The explicit parametric form of burgers equation is as follows:

$$h_t = \lambda_1(t)\, hh_x + \lambda_2 h_{xx} \quad (x,t) \in [-8,8] \times (0,10] \tag{D.10}$$

where, $\lambda_1(t)$ is a time varying parameter and, λ_2 is a constant value equal to 0.1. The reference solution has been computed as given in [57].

$$\lambda_1(t) = \begin{cases} 1.00 & 0 \leq t < 2 \\ 0.75 & 2 \leq t < 4 \\ 0.50 & 4 \leq t < 6 \\ 0.75 & 6 \leq t < 8 \\ 1.00 & 8 \leq t < 10 \end{cases} \tag{D.11}$$

Using the proposed framework of bc-PINN for inverse problems, the parameters of the parametric burgers equation are learnt using an arbitrary length of time segment. In the present example the total time domain has been discretized into 256 time steps. Therefore, in order to identify the time varying parameter (λ_1) an arbitrary time segment of 10 steps has been chosen. In order to identify the parameter λ_1, all the observed data within the 10 steps segment has been considered. A similar procedure has also been adopted for identifying the parameter λ_2.

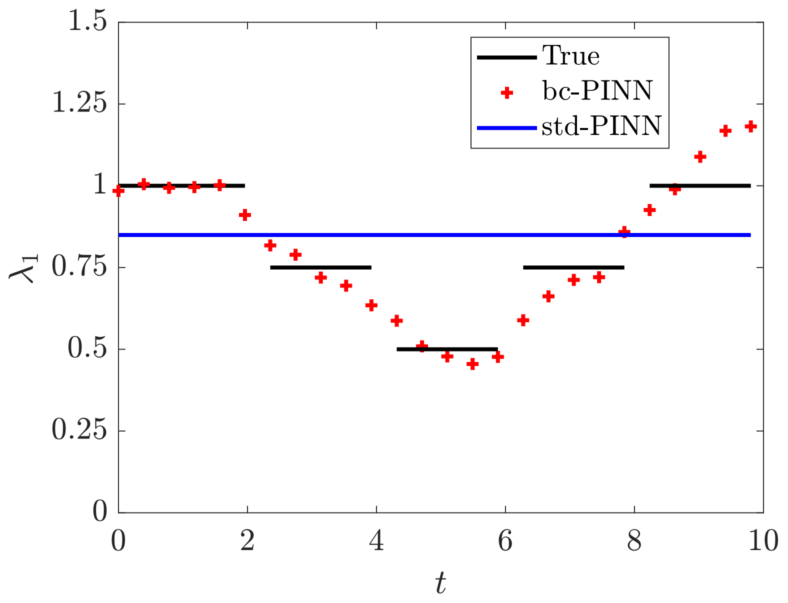


Figure D.22: 'λ_1' learned using the bc-PINN and std-PINN approach for the parametric burgers equation

From figure (D.22,D.23), it can be observed that the predicted parameters follow a trend similar to the true values. The main advantage of inverse bc-PINN scheme is that without any prior knowledge, the nature of a PDE (contant or time-varying) can be identified. Moreover, the values of the time varying parameters can be obtained without any prior information about the time segments over which its constant.

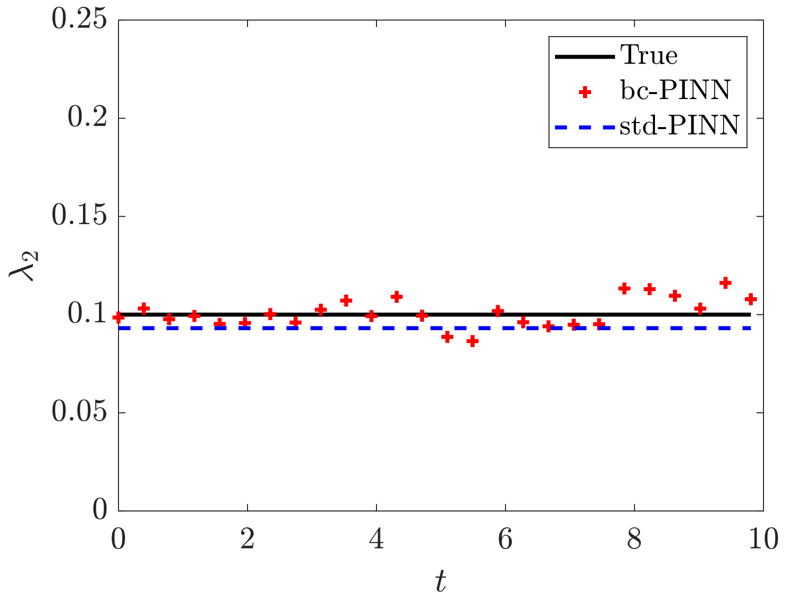


Figure D.23: 'λ_2' learned using the bc-PINN and std-PINN approach for the parametric burgers equation

References

[1] 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) 112789.

[2] H. Arbabi, J. E. Bunder, G. Samaey, A. J. Roberts, I. G. Kevrekidis, Linking machine learning with multiscale numerics: Data-driven discovery of homogenized equations, JOM 72 (12) (2020) 4444–4457.

[3] M. Raissi, P. Perdikaris, G. E. Karniadakis, Inferring solutions of differential equations using noisy multi-fidelity data, Journal of Computational Physics 335 (2017) 736–746.

[4] M. Raissi, P. Perdikaris, G. E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, Journal of Computational Physics 348 (2017) 683–693.

[5] M. Raissi, G. E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, Journal of Computational Physics 357 (2018) 125–141. arXiv:1708.00588.

[6] S. Atkinson, N. Zabaras, Structured bayesian gaussian process latent variable model: Applications to data-driven dimensionality reduction and high-dimensional inversion, Journal of Computational Physics 383 (2019) 166 – 195.

[7] I. Bilionis, N. Zabaras, B. A. Konomi, G. Lin, Multi-output separable gaussian process: Towards an efficient, fully bayesian paradigm for uncertainty quantification, Journal of Computational Physics 241 (2013) 212 – 239.

[8] 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.

[9] 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 (2018).

[10] 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) 108850.

[11] L. Yang, D. Zhang, G. E. Karniadakis, Physics-informed generative adversarial networks for stochastic differential equations, SIAM Journal on Scientific Computing 42 (1) (2020) A292–A317.

[12] X. Meng, G. E. Karniadakis, A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse pde problems, Journal of Computational Physics 401 (2020) 109020.

[13] A. Jagtap, E. Kharazmi, G. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Computer Methods in Applied Mechanics and Engineering 365 (2020) 113028.

[14] A. D. Jagtap, G. Em Karniadakis, Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Communications in Computational Physics 28 (5) (2020) 2002–2041.

[15] S. Karumuri, R. Tripathy, I. Bilionis, J. Panchal, Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks, Journal of Computational Physics 404 (2020) 109120. arXiv:1902.05200.

[16] R. K. Tripathy, I. Bilionis, Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification, Journal of Computational Physics 375 (2018) 565–588.

[17] Y. Yang, P. Perdikaris, Adversarial uncertainty quantification in physics-informed neural networks, Journal of Computational Physics 394 (2019) 136–152.

[18] L. Yang, X. Meng, G. E. Karniadakis, B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data (2020). arXiv:2003.06097.

[19] Q. He, D. Barajas-Solano, G. Tartakovsky, A. M. Tartakovsky, Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport, Advances in Water Resources 141 (2020) 103610.

[20] M. Liu, L. Liang, W. Sun, A generic physics-informed neural network-based constitutive model for soft biological tissues, Computer Methods in Applied Mechanics and Engineering 372 (2020) 113402.

[21] G. Kissas, Y. Yang, E. Hwuang, W. R. Witschey, J. A. Detre, P. Perdikaris, Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering 358 (2020) 112623.

[22] F. Sahli Costabal, Y. Yang, P. Perdikaris, D. E. Hurtado, E. Kuhl, Physics-informed neural networks for cardiac activation mapping, Frontiers in Physics 8 (2020) 42.

[23] Y. Hu, T. Zhao, Z. Xu, L. Lin, Neural time-dependent partial differential equation (2020). arXiv:2009.03892.

[24] Y. Khoo, J. Lu, L. Ying, Solving parametric pde problems with artificial neural networks, European Journal of Applied Mathematics 32 (3) (2020) 421–435.

[25] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Graph kernel network for partial differential equations (2020). arXiv:2003.03485.

[26] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations (2021). arXiv:2010.08895.

[27] L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nature Machine Intelligence 3 (3) (2021) 218–229.

[28] H. Abels, H. Garcke, G. Grün, Thermodynamically consistent, frame indifferent diffuse interface models for incompressible two-phase flows with different densities, Mathematical Models and Methods in Applied Sciences 22 (03) (2012) 1150013.

[29] K. Deckelnick, G. Dziuk, C. M. Elliott, Computation of geometric partial differential equations and mean curvature flow, Acta Numerica 14 (2005) 139–232.

[30] J. Lowengrub, L. Truskinovsky, Quasi incompressible cahn hilliard fluids and topological transitions, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences 454 (1978) (1998) 2617–2654.

[31] B. Li, J. Lowengrub, A. Ratz, A. Voigt, Geometric evolution laws for thin crystalline films: modeling and numerics, Communications in Computational Physics 6 (3) (2009) 433.

[32] L. N. Trefethen, N. Hale, T. A. Driscoll, Chebfun Guide, Pafnuty Publications, Oxford, 2014.

[33] S. J. Pan, Q. Yang, A survey on transfer learning, IEEE Transactions on Knowledge and Data Engineering 22 (10) (2010) 1345–1359. doi:10.1109/TKDE.2009.191.

[34] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Transactions on Neural Networks 5 (2) (1994) 157–166.

[35] Y. Lecun, L. Bottou, G. Orr, K.-R. Müller, Efficient backprop (08 2000).

[36] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[37] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2017). arXiv:1412.6980.

[38] S. Cox, P. Matthews, Exponential time differencing for stiff systems, Journal of Computational Physics 176 (2) (2002) 430–455.

[39] M. Z. Bazant, Thermodynamic stability of driven open systems and control of phase separation by electro-autocatalysis, Faraday discussions 199 (2017) 423–463.

[40] S. M. Allen, J. W. Cahn, A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening, Acta Metallurgica 27 (6) (1979) 1085–1095.

[41] S. Bartels, Numerical methods for nonlinear partial differential equations, Vol. 47, Springer, 2015.

[42] J. Shen, X. Yang, Numerical approximations of allen-cahn and cahn-hilliard equations, Discrete & Continuous Dynamical Systems-A 28 (4) (2010) 1669.

[43] J. W. Cahn, J. E. Hilliard, Free energy of a nonuniform system. i. interfacial free energy, The Journal of chemical physics 28 (2) (1958) 258–267.

[44] A. Miranville, The cahn-hilliard equation and some of its variants, AIMS Mathematics 2 (2017) 479–544.

[45] J. Kim, S. Lee, Y. Choi, S.-M. Lee, D. Jeong, Basic principles and practical applications of the cahn–hilliard equation, Mathematical Problems in Engineering 2016 (2016).

[46] S. C. Takatori, J. F. Brady, Towards a thermodynamics of active matter, Phys. Rev. E 91 (2015) 032117.

[47] T. Speck, J. Bialké, A. M. Menzel, H. Löwen, Effective cahn-hilliard equation for the phase separation of active brownian particles, Phys. Rev. Lett. 112 (2014) 218304.

[48] S. C. Takatori, W. Yan, J. F. Brady, Swim pressure: Stress generation in active matter, Phys. Rev. Lett. 113 (2014) 028103.

[49] A. A. Hyman, C. A. Weber, F. Jülicher, Liquid-liquid phase separation in biology, Annual Review of Cell and Developmental Biology 30 (1) (2014) 39–58.

[50] D. Zwicker, A. A. Hyman, F. Jülicher, Suppression of ostwald ripening in active emulsions, Physical Review E 92 (1) (2015) 012317.

[51] C. P. Brangwynne, C. R. Eckmann, D. S. Courson, A. Rybarska, C. Hoege, J. Gharakhani, F. Jülicher, A. A. Hyman, Germline p granules are liquid droplets that localize by controlled dissolution/condensation, Science 324 (5935) (2009) 1729–1732.

[52] C. P. Brangwynne, P. Tompa, R. V. Pappu, Polymer physics of intracellular phase transitions, Nature Physics 11 (11) (2015) 899–904.

[53] B. Horstmann, T. Danner, W. G. Bessler, Precipitation in aqueous lithium–oxygen batteries: a model-based analysis, Energy & Environmental Science 6 (4) (2013) 1299–1314.

[54] J. Erlebacher, M. J. Aziz, A. Karma, N. Dimitrov, K. Sieradzki, Evolution of nanoporosity in dealloying, Nature 410 (6827) (2001) 450–453.

[55] W. Tian, X. Mao, P. Brown, G. C. Rutledge, T. A. Hatton, Electrochemically nanostructured polyvinylferrocene/polypyrrole hybrids with synergy for energy storage, Advanced Functional Materials 25 (30) (2015) 4803–4813.

[56] D. Lee, J.-Y. Huh, D. Jeong, J. Shin, A. Yun, J. Kim, Physical, mathematical, and numerical derivations of the cahn–hilliard equation, Computational Materials Science 81 (2014) 216–225.

[57] S. Rudy, A. Alla, S. L. Brunton, J. N. Kutz, Data-driven identification of parametric partial differential equations (2018). `arXiv:1806.00732`.

[58] V. Dwivedi, N. Parashar, B. Srinivasan, Distributed learning machines for solving forward and inverse problems in partial differential equations, Neurocomputing 420 (2021) 299–316.

[59] M. Cheng, T. Y. Hou, M. Yan, Z. Zhang, A data-driven stochastic method for elliptic pdes with random coefficients, SIAM/ASA Journal on Uncertainty Quantification 1 (1) (2013) 452–493.