arXiv:2403.03223v2 [cs.LG] 7 Mar 2024

Exact Enforcement of Temporal Continuity in Sequential Physics-Informed Neural Networks

Pratanu Roy[a,*], Stephen Castonguay[b,*]

[a] *Atmospheric, Earth and Energy Division, Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, CA 94550, USA*
[b] *Computational Engineering Division, Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, CA 94550, USA*

Abstract

The use of deep learning methods in scientific computing represents a potential paradigm shift in engineering problem solving. One of the most prominent developments is Physics-Informed Neural Networks (PINNs), in which neural networks are trained to satisfy partial differential equations (PDEs). While this method shows promise, the standard version has been shown to struggle in accurately predicting the dynamic behavior of time-dependent problems. To address this challenge, methods have been proposed that decompose the time domain into multiple segments, employing a distinct neural network in each segment and directly incorporating continuity between them in the loss function of the minimization problem. In this work we introduce a method to exactly enforce continuity between successive time segments via a solution ansatz. This hard constrained sequential PINN (HCS-PINN) method is simple to implement and eliminates the need for any loss terms associated with temporal continuity. The method is tested for a number of benchmark problems involving both linear and non-linear PDEs. Examples include various first order time dependent problems in which traditional PINNs struggle, namely advection, Allen-Cahn, and Korteweg-de Vries equations. Furthermore, second and third order time-dependent problems are demonstrated via wave and Jerky dynamics examples, respectively. Notably, the Jerky dynamics problem is chaotic, making the problem especially sen-

*Corresponding authors. These authors contributed equally to this work.
Email addresses: roy23@llnl.gov (Pratanu Roy), castonguay1@llnl.gov (Stephen Castonguay)

Preprint submitted to Journal of Computational Physics

sitive to temporal accuracy. The numerical experiments conducted with the proposed method demonstrated superior convergence and accuracy over both traditional PINNs and the soft-constrained counterparts.

Keywords: Deep learning, Physics-informed neural networks (PINNs), Causality, Temporal continuity, Chaotic equations

---

1. Introduction

Physics-Informed Neural Networks, introduced by Raissi et al. [1], encompass a general framework that can incorporate physical constraints via differential equations. In general, it involves using a deep neural network as a trial solution to a PDE-constrained optimization problem whose loss function includes residuals from governing equations, boundary and initial conditions, as well as observed data, if present. It is meshless, and the recent progress in deep learning tools has made its implementation through automatic differentiation a straightforward exercise. Indeed, it has been used in modeling a variety of problems, ranging from fluid mechanics [2, 3, 4, 5], heat transfer [6], solid mechanics [7] to subsurface systems [8] and chemical processes [9]. Additionally, it shows promise for inverse problems as it can readily accommodate parameters and data into its structure [10, 11, 12, 13, 14, 15].

However, it has not seen widespread adoption in the computational science community. For one, it generally is not as accurate as traditional numerical methods such as the finite element and finite volume methods [16]. Additionally, the standard PINN method is not robust as it often fails to converge due to the complexity of the loss landscape [17]. Moreover, time-dependent problems can suffer from a causality violation when the temporal variable is treated the same way as spatial variables [18]. Therefore, various improvements must be made before it can be an acceptable alternative for traditional numerical methods [19]. One common approach to simplify the process is to directly enforce initial and/or boundary conditions via specific solution ansatz or modified neural network architecture. This leaves only the PDE residuals (and possibly observable data) as terms in the loss function. For time-dependent problems, an additional approach is to divide the temporal domain into segments and use different neural networks for each one. This takes advantage of the inherent unidirectional nature of time and helps avoid spurious solutions by maintaining causality.

In this work, we propose a novel method for exactly imposing continuity between successive time segments, leading to hard constrained sequential PINNs (HCS-PINNs). We demonstrate its effectiveness for a set of carefully chosen time-dependent problems with increasing difficulty. The problem set encompasses both linear and non-linear equations involving first, second and third order derivatives in time. As demonstrated in the results, the HCS-PINN method can accurately predict the solution of time-dependent problems for which traditional PINNs struggle.

The outline of the paper is as follows. Section 2 presents the numerical background of PINNs, describing various temporal strategies and their advantages and limitations. Section 3 describes our proposed method as a hard constrained strategy that preserves temporal continuity. In Section 4, a number of time-dependent benchmark problems are solved using the proposed method and their solution accuracy is compared with the soft constrained counterpart. We conclude the paper by summarizing the findings in Section 5.

2. Background

Consider the general representation of a partial differential equation:

$$\mathcal{F}[u(x,t)] = 0, \quad x \in \Omega,\ t \in [0,T], \tag{1}$$

where $\mathcal{F}$ is an operator which may consist of time and spatial derivatives. This is accompanied by

$$\mathcal{I}[u(x,0)] = 0, \quad x \in \Omega, \tag{2}$$
$$\mathcal{B}[u(x,t)] = 0, \quad x \in \partial\Omega,\ t \in [0,T], \tag{3}$$

where $\mathcal{I}$ and $\mathcal{B}$ are initial and boundary condition operators, respectively. The vanilla version of PINNs involves a minimization problem with both initial and boundary conditions included in the loss function in addition to the PDE residual, i.e.

$$u(x,t) \approx f(x,t,\theta); \quad \theta = \arg\min_{\theta^*} \mathcal{L}(\theta^*), \tag{4}$$

where $f(x, t, \theta)$ is the output of a neural network with x and t as input neurons and θ as weights and biases. The loss terms are given by

$$\mathcal{L}(\theta) = \lambda_P \mathcal{L}_{\mathrm{P}}(\theta) + \lambda_I \mathcal{L}_{\mathrm{I}}(\theta) + \lambda_B \mathcal{L}_{\mathrm{B}}(\theta), \tag{5}$$

$$\mathcal{L}_{\mathrm{P}}(\theta) = \frac{1}{N_P} \sum_{i=1}^{N_P} (\mathcal{F}[f(x_i, t_i, \theta)])^2, \quad x_i \in \Omega,\ t_i \in [0, T], \tag{6}$$

$$\mathcal{L}_{\mathrm{I}}(\theta) = \frac{1}{N_I} \sum_{i=1}^{N_I} (\mathcal{I}[f(x_i, 0, \theta)])^2, \quad x_i \in \Omega, \tag{7}$$

$$\mathcal{L}_{\mathrm{B}}(\theta) = \frac{1}{N_B} \sum_{i=1}^{N_B} (\mathcal{B}[f(x_i, t_i, \theta)])^2, \quad x_i \in \partial\Omega,\ t_i \in [0, T]. \tag{8}$$

N_P, N_I, and N_B are the number of collocation points for the PDE, initial and boundary conditions, respectively. Additionally, λ_P, λ_I, and λ_B are parameters which weight the individual impact of each loss term. Figure 1 shows a sample distribution of random collocation points for a two-dimensional spatio-temporal domain.

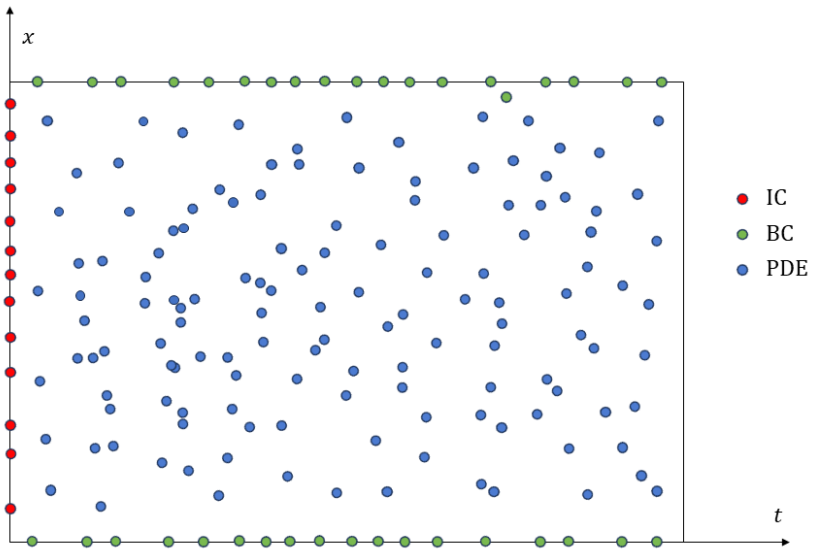

Figure 1: Collocation points for PDE, initial and boundary conditions.

2.1. Soft Versus Hard Constraints

The above formulation enforces the initial and boundary conditions in a soft manner, which leads to a competition between driving down the PDE residual and satisfying the boundary and initial conditions. To obtain satisfactory solutions, the weights must often be adjusted to balance the losses among these three terms. To circumvent this, various strategies have been proposed that exactly enforce these conditions (i.e. in a hard manner), leaving only the PDE residual to be minimized.

Periodic boundary conditions can be enforced exactly by introducing an intermediate layer between the input neurons and the first hidden layer [20]. Initial, Dirichlet, Neumann, and Robin boundary conditions can be exactly enforced via solution ansatz that includes the output of the neural network [12]. For example, the following simple form exactly satisfies the initial condition for first order problems in time:

$$u(x,t) = g_0(x) + f(x,t,\theta)t, \tag{9}$$

where $u(x,0) = g_0(x)$ is the given initial condition. For simple geometric domains, analytic ansatz can generally be found to enforce boundary conditions as well. However, the situation becomes much more complicated for arbitrarily complex geometries, although some recent progress has been made in this area. For instance, Sukumar and Srivastava [21] generated approximate composite distance functions to exactly enforce both Dirichlet and Neumann conditions.

2.2. Temporal Strategies

For time-dependent problems, vanilla PINNs often suffer from error propagation due to the unidirectional nature of time, leading to nonphysical results. Because of inherent causality, time should not be treated similarly as other space variables. Various methods have been proposed to address this issue.

One class of methods involve decomposing the domain into multiple time segments while using a single global neural network as the trial solution. For instance, Wight and Zhao [22] introduced the Adaptive Time Sampling approach. The neural network is first trained on a small initial subdomain. After convergence, the training domain is extended by a time increment, and the network is retrained for the new extended time interval. This continues until the neural network is trained over the whole domain. A similar method

was proposed by Mattey and Ghosh [23], in which each time slab is trained sequentially, with an additional term added to the loss function so that the solution remains compatible with all previous times steps. This method is known as backward compatible PINNs (bc-PINNs). Wang et al. [18] developed the Causal Training approach, in which time steps are used to weight each subdomain separately. The weights are defined in terms of the loss from all prior time steps, ensuring that for any time step the solution up to that point is obtained with adequate accuracy.

Time stepping schemes have also been used in which a separate neural network is defined for each time slab [22, 17, 24, 25]. In this case, the initial condition for each segment depends on the PINN solution at the end of the previous time step. Penwarden et al. [26] unified many of these ideas under the XPINNs framework, which allows multiple NNs to be trained either sequentially, all at once, or through a moving window strategy in which multiple are trained at the same time.

Each of the previous methods that utilize multiple neural networks enforce temporal compatibility conditions via loss terms in the minimization problem, leading to only approximate continuity between time steps. In this sense, they can be categorized as Soft Constrained Sequential PINNs (SCS-PINNs). As shown in the next section, temporal continuity can be imposed exactly via a solution ansatz leading to Hard Constrained Sequential PINNs.

3. Hard Constrained Sequential PINNs (HCS-PINNs)

First, the temporal domain is decomposed into time windows, as shown in Figure 2. The beginning and end points of a given time window are denoted as t_N and t_{N+1}, respectively. The solution in this sub-domain is denoted as u_{N+1}. A straightforward approach to enforcing C^0 continuity would be to use

$$u_1(x,t,\theta_1) = g(x,t) + f(x,t,\theta_1)(t), \tag{10}$$

$$u_{N+1}(x,t,\theta_{N+1}) = u_N(x,t,\theta_N) + f(x,t,\theta_{N+1})(t-t_N) \quad \forall N \geq 1 \tag{11}$$

Where $g(x,t)$ satisfies $\mathcal{I}[g(x,0)] = 0$. However, this is a recursive form, where neural network solutions for all previous time windows must be evaluated to compute the solution for the current time window. In order to avoid a

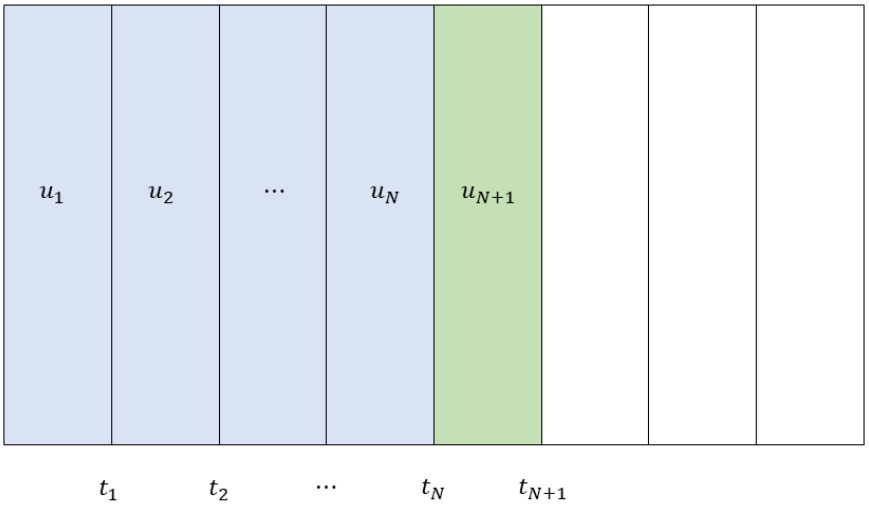

Figure 2: Neural network solution in multiple time sub-domains.

recursive definition for the trial solution, we propose the following:

$$u_1(x,t,\theta_1) = h_N(t)g(x,t) + h_{N+1}(t)f_1(x,t,\theta_1), \tag{12}$$

$$\begin{aligned} u_{N+1}(x,t,\theta_{N+1}) &= h_N(t)f_N(x,t,\theta_N) \\ &\quad + h_{N+1}(t)f_{N+1}(x,t,\theta_{N+1}) \quad \forall N \geq 1, \end{aligned} \tag{13}$$

Here, h_N and h_{N+1} are auxiliary functions chosen to obey certain conditions depending on the desired continuity of time derivatives. In order to break recursion and maintain C^0 continuity, we define $\tau = \frac{t-t_N}{t_{N+1}-t_N}$ and require the following:

$$h_N(\tau = 0) = 1,\ h_N(\tau = 1) = 0, \tag{14}$$

$$h_{N+1}(\tau = 0) = 0,\ h_{N+1}(\tau = 1) = 1. \tag{15}$$

C^m continuity can be obtained by additionally enforcing that the derivatives up to order m vanish at $\tau = 0$ and $\tau = 1$. The auxiliary functions can be chosen by including polynomials from order 0 up to order $2m+1$ while enforcing these boundary conditions. Generally speaking, the continuity requirement between time segments is one less than the highest order time derivative in the problem, and choosing a value greater than this will unnecessarily over-constrain the problem. In this work, we will consider first, second, and third order problems with respect to time, which therefore leads to the choices for

$h(\tau)$ listed in Table 1. Note that these functions satisfy a partition of unity, and can be interpreted as interpolation functions (see Figure 3).

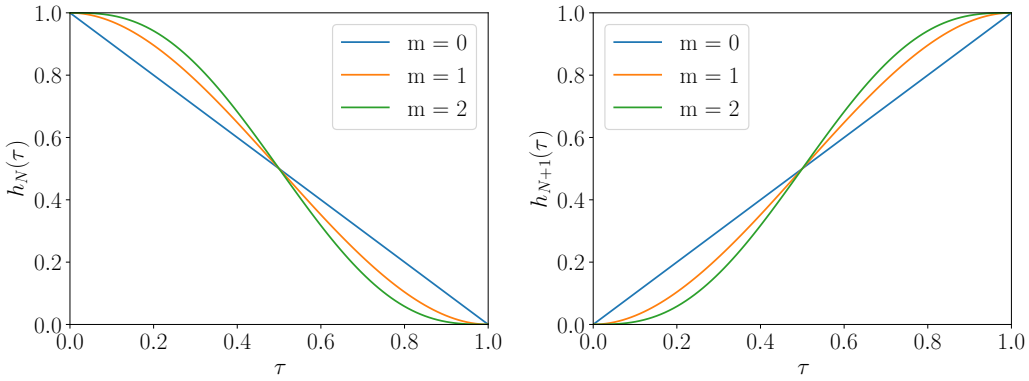

Figure 3: Interpolation functions for C^0, C^1, and C^2 continuity.

Table 1: Auxiliary or Interpolation Functions

C^m	$h_N(\tau)$	$h_{N+1}(\tau)$
$m=0$	$1-\tau$	τ
$m=1$	$1-3\tau^2+2\tau^3$	$3\tau^2-2\tau^3$
$m=2$	$1-6\tau^5+15\tau^4-10\tau^3$	$6\tau^5-15\tau^4+10\tau^3$

While these functions enforce continuity between time steps, the initial condition must be satisfied as well. Consider initial conditions given in the form:

$$\begin{aligned} u(x,0) &= g_0(x) \\ u_t(x,0) &= g_1(x) \\ u_{tt}(x,0) &= g_2(x) \\ &\vdots \end{aligned} \tag{16}$$

The initial conditions for M^{th}-order problems in time (requiring C^{M-1} continuity) are satisfied by taking the first M terms of the following series:

$$g(x,t) = g_0(x) + tg_1(x) + \frac{t^2}{2}g_2(x) + \ldots \frac{t^n}{n!}g_n(x) + \ldots \tag{17}$$

4. Results

A framework was developed to test the accuracy and robustness of physics informed neural network algorithms for dynamic problems. The Python based Jax library [27] was used to develop the code, which employs feed forward neural networks. For all problems, boundary conditions are enforced directly in the solution ansatz or encoded in the neural network architecture (see Appendix A). Therefore, the loss terms for HCS-PINNs solely consist of PDE residuals, whereas for SCS-PINNs, they include both initial and temporal continuity conditions. A dense distribution of random collocation points is used to sample each time segment, with mini-batching employed to speed up the training time, except for the Jerk equation where a full batch gradient descent is used. The optimization procedure starts with a fixed number of Adam [28] iterations. Then the L-BFGS [29] iterations are performed until a convergence criterion is met or until it reaches a maximum number of iterations. Due to the stochastic nature of the mini-batching procedure, a simple five term moving average of the change in loss function from iteration to iteration is used as the convergence criterion. Additionally, a different (larger) mini-batch is used to evaluate the loss criterion to prevent false convergence. The activation function is the hyperbolic tangent ($tanh$), with all parameters initialized using the Glorot scheme [30]. An empirical linear loss weighting scheme is used to further enforce temporal causality (see Appendix B). For any problem without an analytical solution, reference solutions were generated with the Chebfun package [31] in Matlab. In each scenario, we analyze the relative L^2 error and training duration between HCS-PINN and SCS-PINN. The simulations were conducted using an Apple M1 Max processor, featuring a clock speed of 3.2 GHz and a memory of 32 GB.

We conduct numerical experiments by solving five dynamic problems with increasing level of complexity. For each problem, three scenarios are presented in which the temporal decomposition varies by the number of time steps (nt). The first two problems, the advection equation and wave equation, are linear and exhibit hyperbolic behavior. Despite their linear nature, the presence of a high characteristic velocity poses challenges for predicting solution over extended time intervals. The subsequent two problems, namely Allen-Cahn and Korteweg-de Vries (KdV) equations, are non-linear and commonly used in the PINN literature to benchmark algorithms. Lastly, we address a Jerky dynamics equation, known for its chaotic behavior.

4.1. Advection Equation

The advection equation describes transport of a quantity that moves with a fluid. The periodic, one-dimensional version that is used in many PINN papers is given by

$$u_t + cu_x = 0, x \in [0, 2\pi], t \in [0, 1] \tag{18}$$

$$u(0, x) = sin(x) \tag{19}$$

$$u(t, 0) = u(t, 2\pi) \tag{20}$$

where c is the prescribed fluid velocity. The exact solution to this problem is given by

$$u(x, t) = sin(x - ct) \tag{21}$$

As the advection equation is first order in time, C^0 continuity needs to be enforced between adjacent time segments. In our implementation, the PDE has been normalized by the fluid velocity. We found that this normalization sufficiently enforced the initial condition for SCS-PINNs. Alternatively, the penalty parameter λ_I could be increased to achieve the same effect.

When the fluid velocity is relatively large, vanilla PINNs struggle to capture the correct solution. In particular, PINN solutions often falsely converge to the zero solution, as seen in Figures 4 and 5. For SCS-PINN, taking 10 time segments is sufficient to avoid this for $c = 50$, but not for $c = 100$. However, HCS-PINN captures the correct solution for all fluid speeds when taking 10 time steps, and captures up to $c = 50$ when taking only 4.

Various other efforts have been made to mitigate this issue, including aforementioned temporal techniques as well as methods that encode characteristics into the architecture [32]. Wang et al. [25] suggested a range of techniques for obtaining optimal accuracy which include temporal and spatial Fourier feature embedding and grad norm weighting, among others. Indeed a relative L^2 error as low as $\mathrm{O}(10^{-4})$ was achieved for high transport velocity ($c = 80$). In addition to the above techniques, they used a deep neural network (4 layers, 256 neurons in each layer, batch size of 8192) with a modified multi-layer perceptron (MLP). Here, we demonstrate that even with a relatively smaller network and a batch size (4 layers, 32 neurons in each layer, batch size of 128) using a standard MLP, a relative L^2 error of $\mathrm{O}(10^{-3})$ can be reached for high advection velocity cases (i.e. $c = 100$) with HCS-PINN.

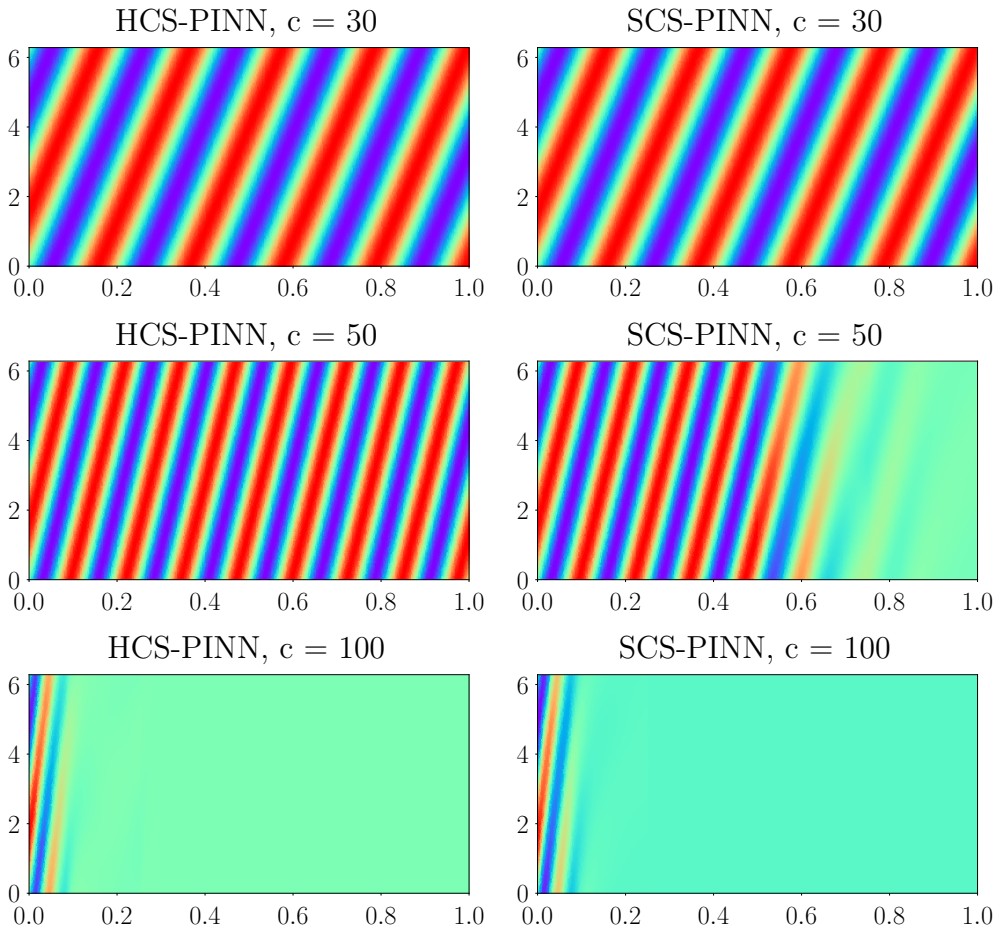

Figure 4: HCS-PINN and SCS-PINN solution of advection equation for $nt = 4$.

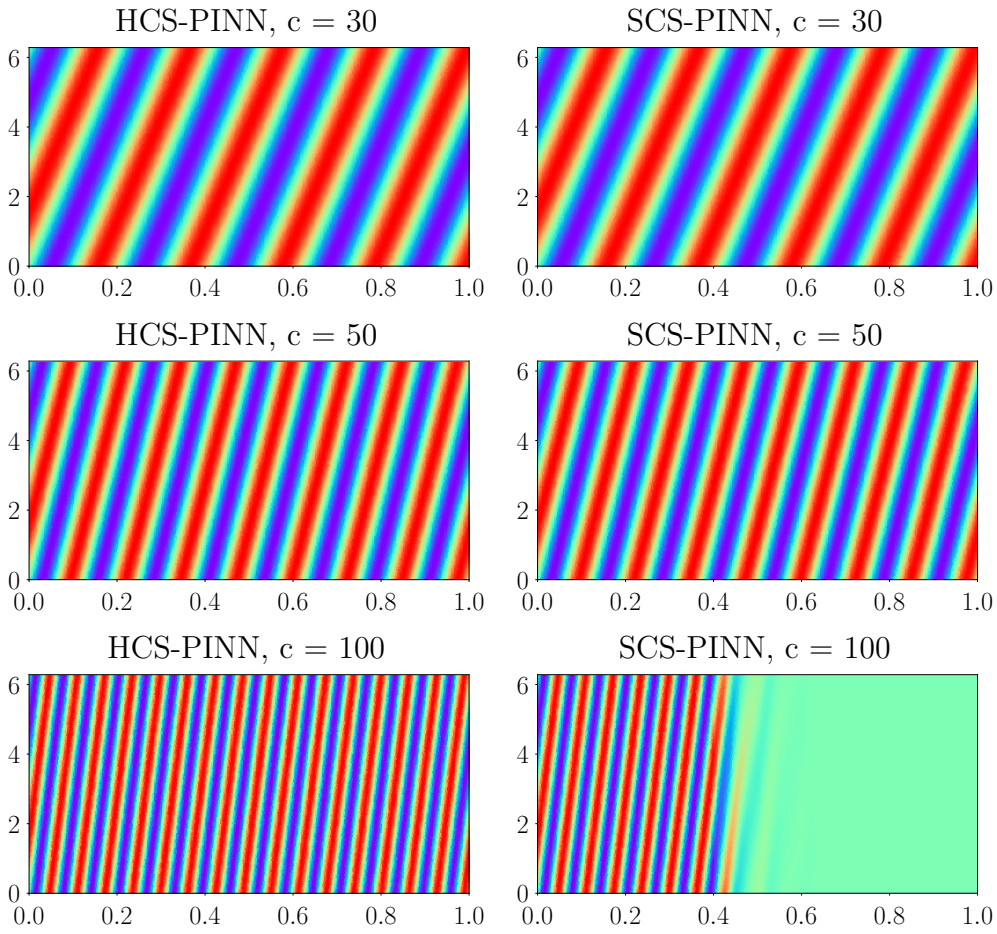

Figure 5: HCS-PINN and SCS-PINN solution of advection equation for $nt = 10$.

Table 2: Relative L^2 Error for advection velocity c = 30 (10,000 Adam iterations + 1,000 L-BFGS iterations, loss tolerance = 1×10^{-6})

nt	HCS-PINN	HCS-PINN Time (secs)	SCS-PINN	SCS-PINN Time (secs)
1	8.6700×10^{-1}	76	8.3647×10^{-1}	47
4	3.4179×10^{-3}	601	5.0206×10^{-3}	395
10	2.1304×10^{-3}	590	3.4125×10^{-3}	502

Table 3: Relative L^2 Error for advection velocity c = 50 (10,000 Adam iterations + 1,000 L-BFGS iterations, loss tolerance = 1×10^{-6})

nt	HCS-PINN	HCS-PINN Time (secs)	SCS-PINN	SCS-PINN Time (secs)
1	9.5444×10^{-1}	230	1.6658×10^{0}	57
4	5.1898×10^{-3}	1314	5.7477×10^{-1}	981
10	3.7520×10^{-3}	1127	4.8585×10^{-3}	993

Table 4: Relative L^2 Error for advection velocity c = 100 (10,000 Adam iterations + 1,000 L-BFGS iterations, loss tolerance = 1×10^{-6})

nt	HCS-PINN	HCS-PINN Time (secs)	SCS-PINN	SCS-PINN Time (secs)
1	1.02236×10^{0}	56	9.9963×10^{-1}	48
4	9.6380×10^{-1}	619	9.7280×10^{-1}	146
10	6.2772×10^{-3}	3812	7.3534×10^{-1}	1969

4.2. Wave Equation

Another hyperbolic system for which PINNs sometimes struggle is the following 1D wave equation:

$$u_{tt} = c^2 u_{xx},\ x \in [0, \pi],\ t \in [0, 2\pi] \tag{22}$$

$$u(0, t) = 0,\ u(\pi, t) = 0, \tag{23}$$

$$u(x, 0) = sin(x),\ u_t(x, 0) = c\, sin(x), \tag{24}$$

where c is the wave propagation speed. The analytical solution of the above problem is given by:

$$u(x, t) = sin(x)(sin(ct) + cos(ct)) \tag{25}$$

The second order derivative in time requires that we enforce C^1 continuity at time sub-domain interfaces. In addition to the temporal interpolation functions, the neural network is multiplied by the term $x(\pi - x)$ to strongly enforce zero Dirichlet boundary conditions. Similar to the advection equation, the PDE was scaled by the square of the wave speed in lieu of altering the weighting parameter λ_I for SCS-PINNs. For low wave speeds, such as $c = 1$ or less, it is relatively easy to get accurate solutions (see Figure 6). However, as the wave speed increases, this becomes increasingly difficult with vanilla PINNs. For example, when $c = 10$, neither the HCS-PINN or SCS-PINN converges when only a single time segment is used, as shown in Figure 7. For the cases of 4 and 10 time windows, HCS-PINN gives less error than the corresponding SCS-PINN. Additionally, the 10 time step case actually takes less time than the 4 time step case, as it continually reaches the tolerance criterion earlier. These results are tabulated in Table 5.

While the SCS-PINN solution may seem continuous over time, the presence of discontinuities becomes evident in the error contour depicted in Figure 6. For both low and high fluid velocities $c = 1$ and $c = 10$, relative L^2 errors of $O(10^{-4})$ were achieved using the HCS-PINN method.

4.3. Allen-Cahn Equation

The Allen-Cahn equation is used for the evolution of phase transitions as well as more general interface motion [33]. A particular form used in benchmarking PINNs is given by

$$u_t - \lambda_1 u_{xx} + \lambda_2 u^3 - \lambda_2 u = 0,\ x \in [-1, 1],\ t \in [0, 1] \tag{26}$$

$$u(0, x) = x^2 cos(\pi x) \tag{27}$$

$$u(t, -1) = u(t, 1),\ u_x(t, -1) = u_x(t, 1) \tag{28}$$

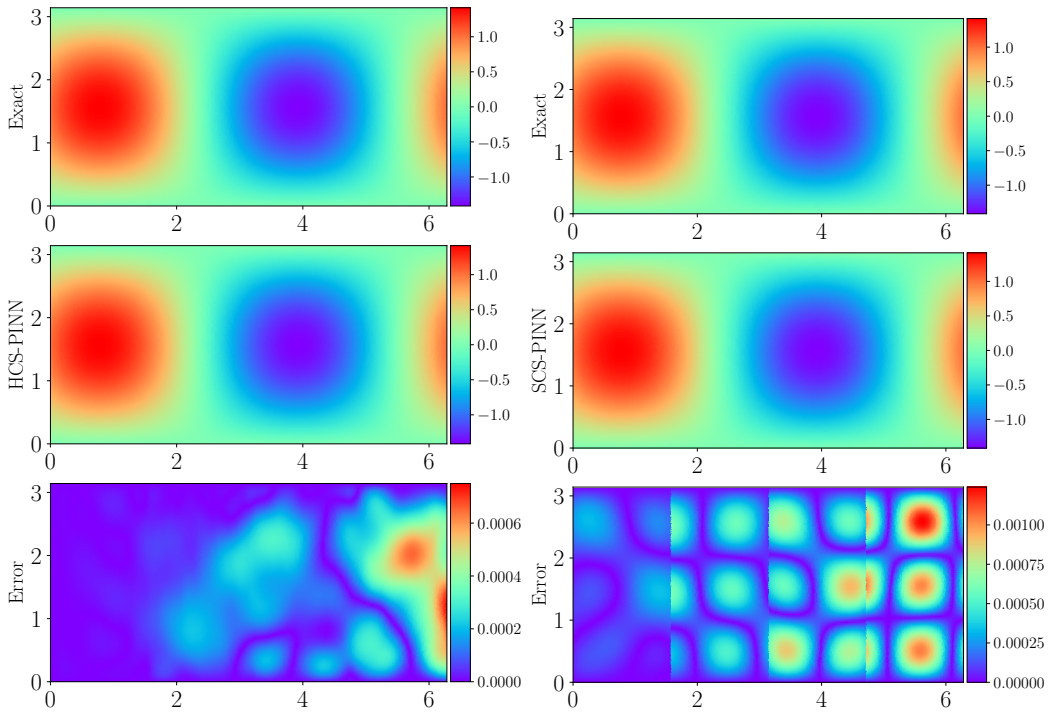

Figure 6: 1D Wave equation contour for wave speed $c = 1$ with HCS-PINN (left) and SCS-PINN (right). Even with 4 time segments, HCS-PINN produced a relative L^2 error of 2.6525×10^{-4}. The corresponding SCS-PINN error is 5.1211×10^{-4}.

where $\lambda_1 = 0.0001$ and $\lambda_2 = 5$. The Allen-Cahn equation requires C^0 continuity between sequential neural networks. The interplay between the diffusion term and non-linear source terms makes it a particularly challenging problem, posing difficulties even for traditional numerical methods. The stiffness of the equation is dictated by the relative magnitude of the diffusion and reaction coefficients, i.e. λ_1 and λ_2, respectively. A number of methods, including Adaptive Time Sampling and bc-PINNs, were developed with this specific problem in mind.

Here, all cases ($nt = 1, 4, 10$) converge as indicated by Table 6, with HCS-PINN reaching a relative L^2 error of $\mathrm{O}(10^{-4})$ and SCS-PINN reaching $\mathrm{O}(10^{-3})$. While both methods take approximately the same amount of time in these simulations, HCS-PINN produced more accurate results for a given number of time steps. Note that we have used a penalty of 100 for λ_I in the SCS-PINN loss terms associated with temporal continuity and initial

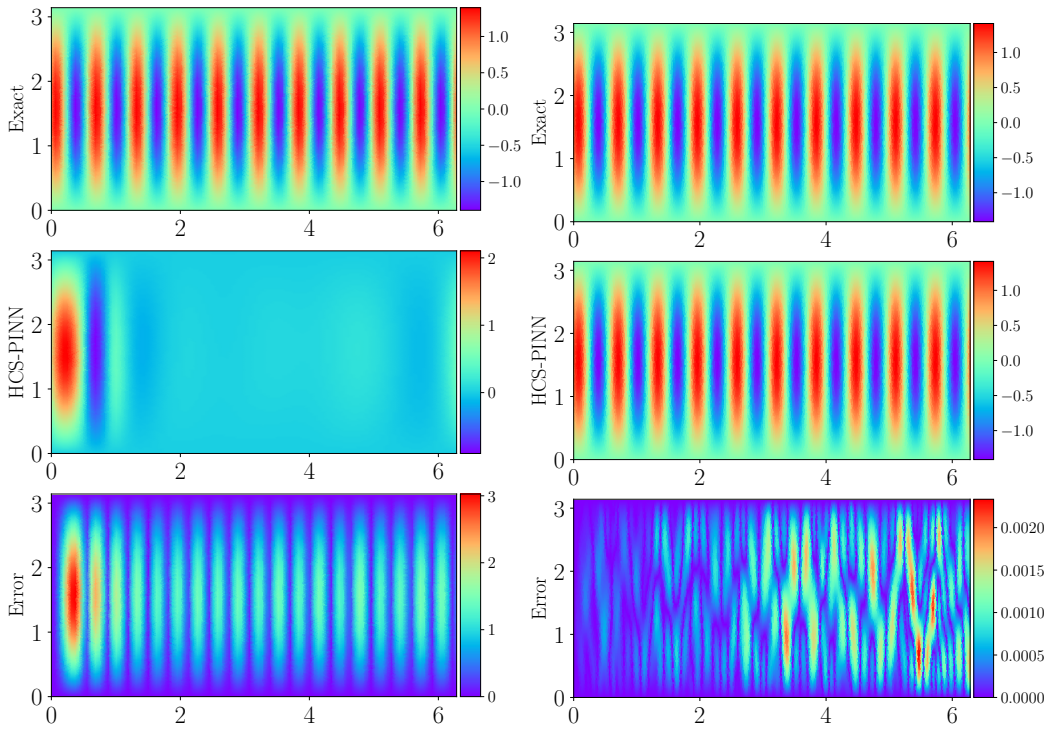

Figure 7: 1D Wave equation contour for fluid velocity $c = 10$ with HCS-PINN: $nt = 1$(left), $nt = 10$ (right)

Table 5: Relative L^2 Error for wave equation with wave speed $c = 10$ (20,000 iterations of Adam + 1,000 L-BFGS iterations, loss tolerance $= 1 \times 10^{-6}$)

nt	HCS-PINN	HCS-PINN Time (secs)	SCS-PINN	SCS-PINN Time (secs)
1	1.1642×10^{0}	718	9.8927×10^{-1}	598
4	1.5629×10^{-3}	2760	3.1508×10^{-2}	3564
10	8.2517×10^{-4}	2188	1.9872×10^{-3}	7348

conditions. Without this penalty, the accuracy of SCS-PINN was even lower.

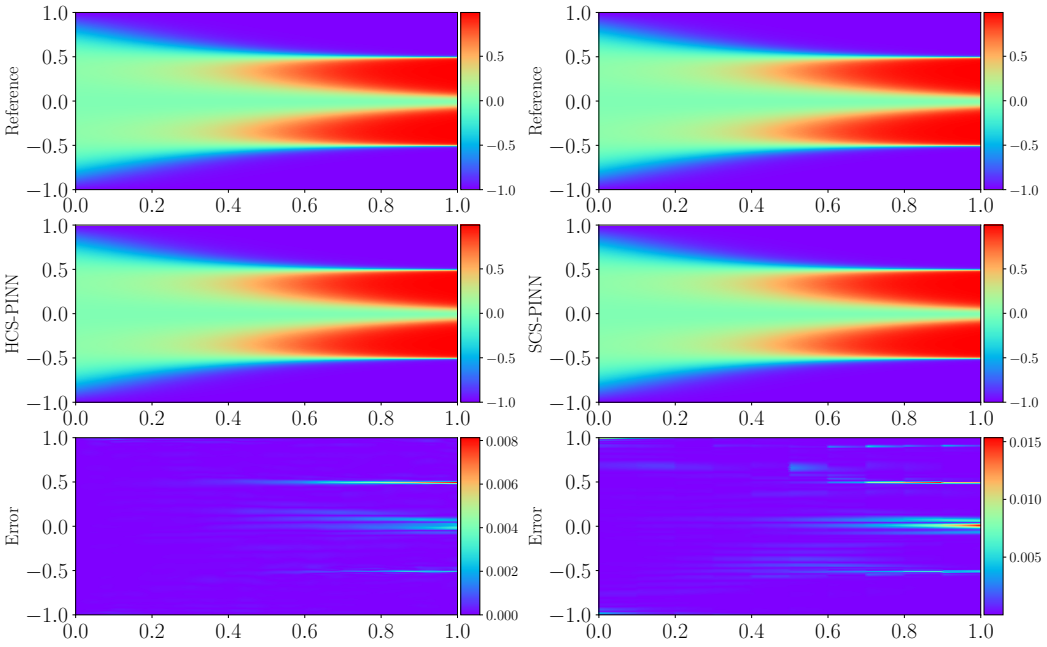

Figure 8: Contours of solution and errors of Allen-Cahn equation for $nt = 10$ with HCS-PINN (left), and SCS-PINN (right).

Table 6: Relative L^2 Error for Allen-Cahn equation (20,000 Adam iterations + 200 L-BFGS iterations, batch size = 256, loss tolerance = 1×10^{-7})

nt	HCS-PINN	HCS-PINN Time (secs)	SCS-PINN	SCS-PINN Time (secs)
1	1.0455×10^{-2}	348	4.3057×10^{-2}	368
4	1.0650×10^{-3}	1495	2.8939×10^{-3}	1426
10	5.4117×10^{-4}	3222	1.4205×10^{-3}	3639

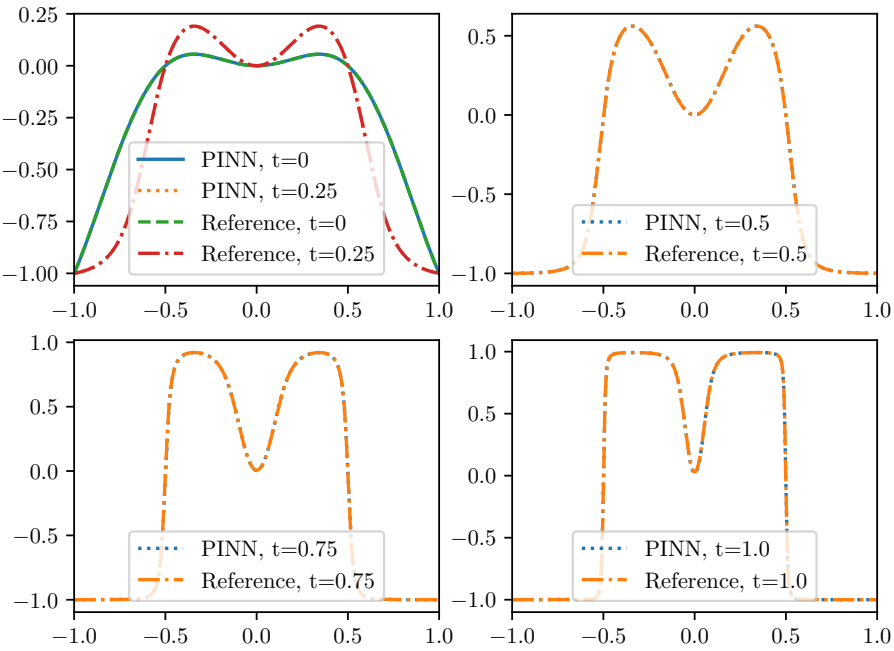

Figure 9: Comparison of Allen-Cahn equation with reference solution [31] for different time snapshots with HCSPINN ($nt = 4$).

4.4. Korteweg–de Vries (KdV) Equation

The Korteweg–de Vries (KdV) equation, which depicts the evolution of waves on shallow water surfaces [34], is another common benchmark problem

for PINNs. The particular problem is given by

$$u_t + \lambda_1 u u_x + \lambda_2 u_{xxx} = 0,\ x \in [-1, 1],\ t \in [0, 1] \tag{29}$$

$$u(0, x) = cos(\pi x) \tag{30}$$

$$u(t, -1) = u(t, 1),\ u_x(t, -1) = u_x(t, 1) \tag{31}$$

where $\lambda_1 = 1$ and $\lambda_2 = 0.0025$. The KdV equation contains three distinct terms. The unsteady term u_t requires C^0 continuity, while the uu_x term is a non-linear advection term and u_{xxx} is a dispersion term. The initial cosine wave therefore undergoes both advection and dispersion as it evolves, resulting in the generation of higher frequency modes over time. This evolution poses a challenge for PINN algorithms in accurately predicting the long-term behavior of the KdV equation.

The solution is well captured by the SCS-PINN and especially the HCS-PINN within the time range [0,1], as shown in Figures 10 and 11. Again, the discontinuities at the time window interfaces in the error contour for SCS-PINN makes apparent the soft nature of continuity enforcement.

A comparison of the errors from the two approaches is listed in Table 7. As with the Allen-Cahn equation, the HCS-PINN shows better accuracy than the SCS-PINN using the same amount of training time. Again, we have also tuned the penalty parameter λ_I to 100 to improve the SCS-PINN solution.

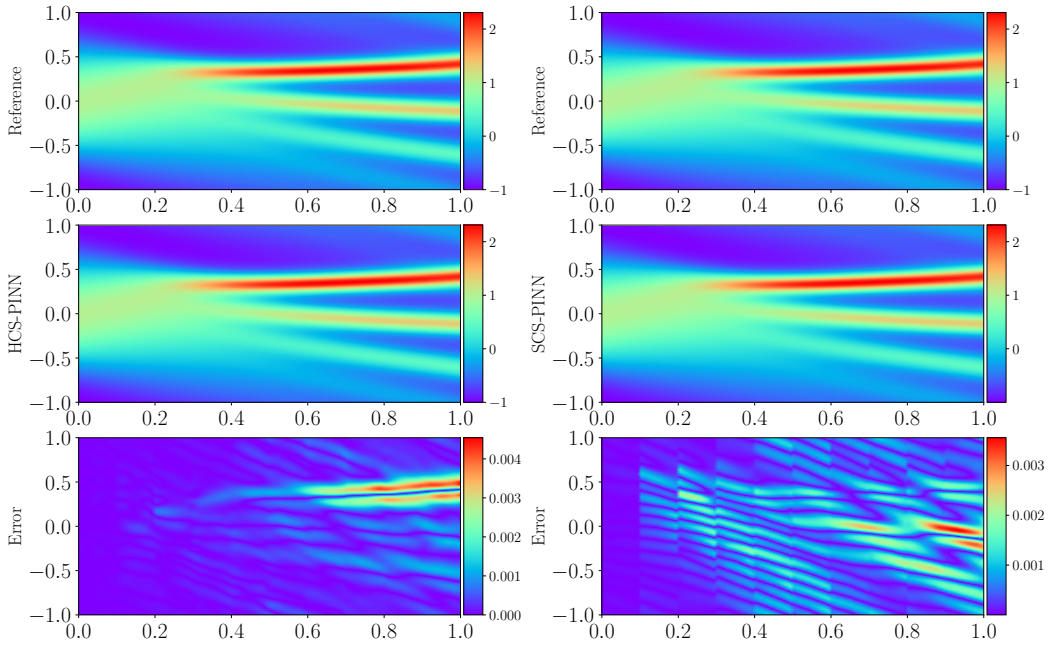

Figure 10: Contours of solution and errors of KdV equation for $nt = 10$ with HCS-PINN (left), and SCS-PINN (right).

Table 7: Relative L^2 Error for KdV equation (20,000 Adam iterations + 200 L-BFGS iterations, loss tolerance $= 1 \times 10^{-6}$)

nt	HCS-PINN	HCS-PINN Time (secs)	SCS-PINN	SCS-PINN Time (secs)
1	1.1506×10^{-2}	701	3.2753×10^{-2}	684
4	1.3264×10^{-3}	2764	2.6425×10^{-3}	2738
10	9.7535×10^{-4}	7128	1.0520×10^{-3}	6477

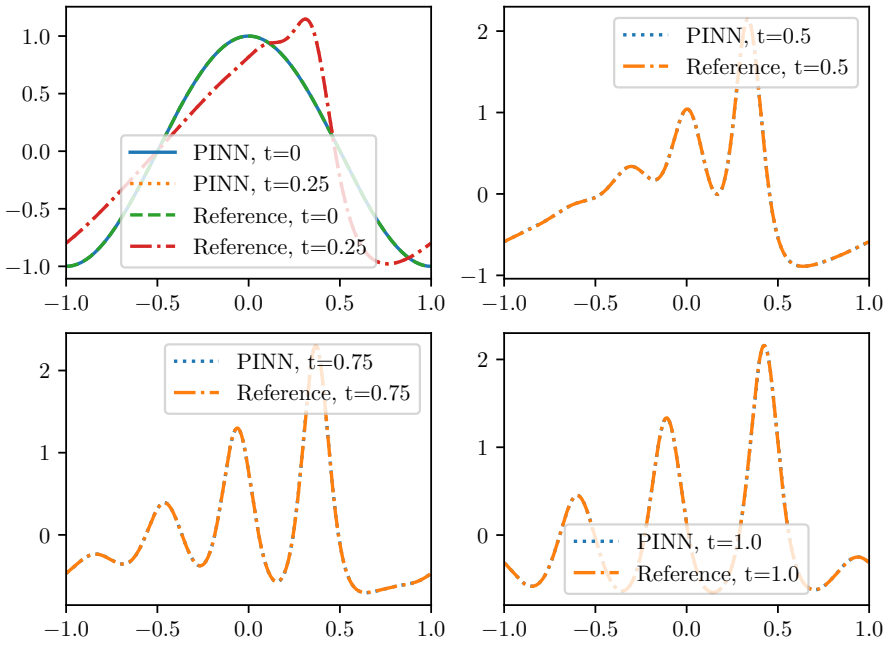

Figure 11: Comparison of KdV equation with reference solution [31] for different time snapshots with HCS-PINN ($nt = 4$).

4.5. Chaotic Dynamics

Many dynamical systems which exhibit chaotic behavior, such as the famous Lorenz equation [35], are cast in terms of three first order autonomous ODEs, i.e.

$$\begin{aligned} x_t &= f_1(x, y, z) \\ y_t &= f_2(x, y, z) \\ z_t &= f_3(x, y, z) \end{aligned} \tag{32}$$

.

In fact, this is a minimal setting for chaotic solutions, as one or two first order continuous autonomous equations cannot produce chaos according to Poincare-Bendixon theorem [36]. In the years that followed Lorenz's pioneering work, various efforts were made to develop simple chaotic models of

this form, of which Rossler [37] and Sprott [38] were at the forefront. Many of these models can be recast into a single, third order Jerk equation of the form:

$$x_{ttt} = J(x, x_t, x_{tt}) \tag{33}$$

In particular, Eichhorn et al. [39] systematically showed that Rossler's prototype-4 model and many of Sprott's equations can undergo this type of transformation. Additionally, they outlined seven different classes of dissipative, chaotic Jerk systems. One of the simplest is the JD_2 class, which is given by

$$x_{ttt} = k_1 x_{tt} + k_2 x_t + x^2 + k_3 \,. \tag{34}$$

This is used here as an example problem, with $k_1 = -0.4$, $k_2 = -2.1$, and $k_3 = -1.0$ as suggested by Sprott [40]. Additionally we take $x(0) = 0$, $x_t(0) = 1.0$, $x_{tt}(0) = 1.0$ as initial conditions.

The third order derivative in time requires that C^2 continuity should be enforced for the Jerk equation. A penalty parameter of $\lambda_I = 10$ was used for SCS-PINN to improve convergence. Both HCS-PINN and SCS-PINN methods fail to converge for $nt = 1$. The HCS-PINN iterations diverge and the line search method fails for L-BFGS optimizer resulting in a very high value of L^2 error for $nt = 1$. The SCS-PINN iterations converges to a constant (or zero) solution even for $nt = 5$, as shown in Figure 12. Using the HCS-PINN method, we were able to obtain $\mathrm{O}(10^{-3})$ accurate results for $nt = 5$ and $nt = 10$ up to $t = 50$. Figure 13 shows the solution in a 3D phase space for $nt = 10$, which depicts the chaotic nature of its evolution.

Table 8: Relative L^2 Error for Jerk equation (10,000 Adam iterations + 300 L-BFGS iterations, loss tolerance $= 1 \times 10^{-7}$)

nt	HCS-PINN	HCS-PINN Time (secs)	SCS-PINN	SCS-PINN Time (secs)
1	8.4944×10^{1}	299	1.7552×10^{0}	1584
5	4.3303×10^{-3}	1979	1.7543×10^{0}	1494
10	1.0087×10^{-3}	1791	2.8439×10^{-2}	1257

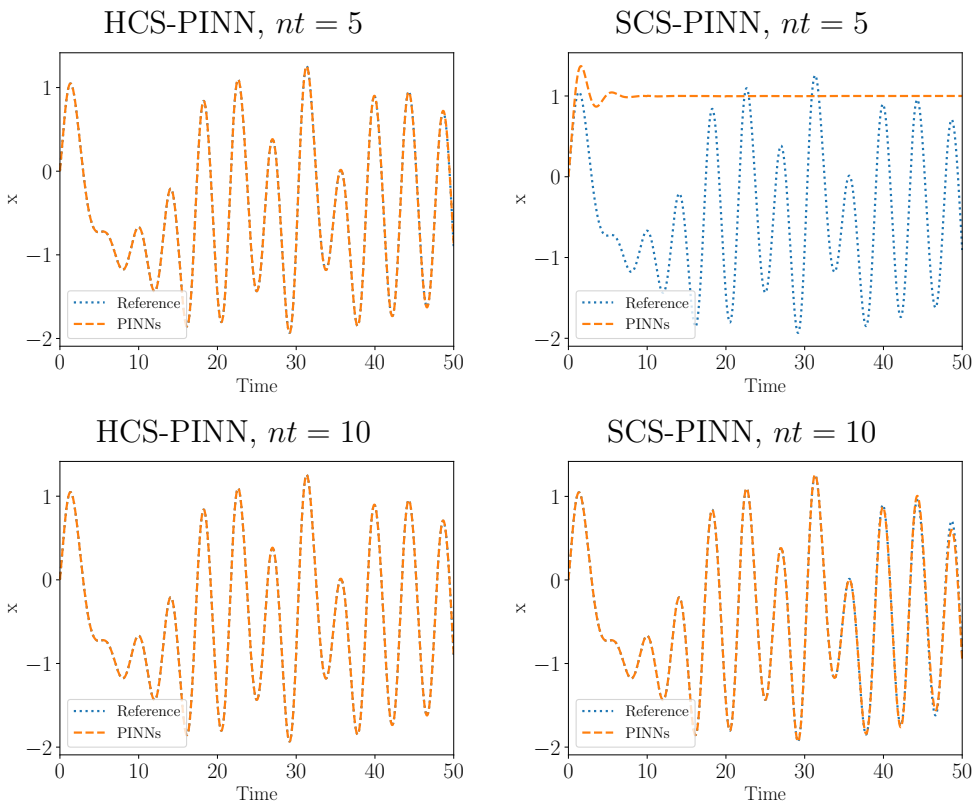

Figure 12: Comparison with HCS-PINN and SCS-PINN solutions of Jerky dynamics equation with chebfun solution [31].

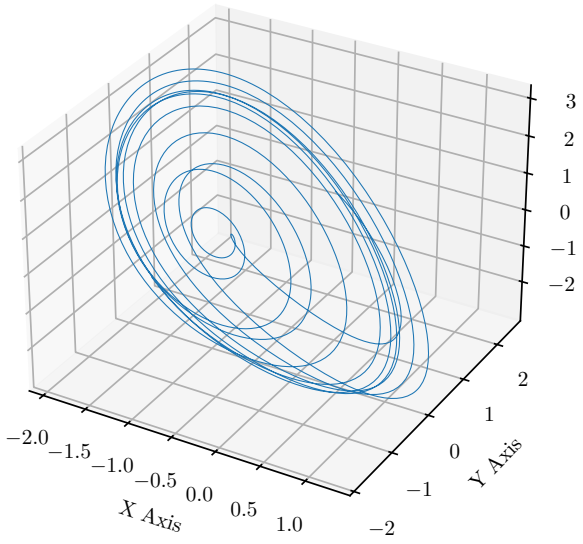

Figure 13: Evolution of Jerky dynamics in 3D phase space with HCSPINN ($nt = 10$).

While some hyper-parameter tuning of the penalty parameter of the initial condition (λ_I) was conducted to ensure a fair comparison between HCS-PINN and SCS-PINN for all problems, it is feasible to manually adjust weights to achieve similar or better accuracy for SCS-PINN. We note that there are approaches to adaptively adjust weights for optimally balancing the loss [41, 42]. However, this comes with additional computational expense. In this aspect, HCS-PINN is advantageous as it obviates the need to manually or automatically adjust the weights of time continuity constraints.

5. Summary

PINNs often struggle for time-dependent problems due to issues with temporal causality. This has previously been addressed by performing a time-domain decomposition, employing a distinct neural network for each time segment, and ensuring temporal continuity through terms incorporated into the loss function. In this work we have introduced the HCS-PINN method, which eliminates the need to include these terms by strictly enforcing temporal continuity between successive neural networks.

Both methods were demonstrated on first, second, and third order problems in time. The advection and wave equations are linear, hyperbolic prob-

lems for which PINNs often erroneously converge to the zero solution if the specified characteristic speeds are large. HCS-PINN has shown to be a superior approach in avoiding this spurious solution. Allen-Cahn and Korteweg-de Vries are both nonlinear problems which are widely used as benchmarks in the literature. In all cases, the hard constrained PINNs outperformed the soft constrained counterparts in terms of accuracy. Finally, a jerky dynamics problem was presented that is chaotic in nature, making it very sensitive to initial conditions. HCS-PINN was able to obtain the correct solution with only 5 time steps while SCS-PINN was not, and for 10 time steps its accuracy was better by an order of magnitude.

Acknowledgement

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under the Contract DE-AC52-07NA27344.

Appendix A. Periodic Boundary Condition Enforcement

Periodic boundary conditions are enforced via a method introduced by Dong and Ni [20]. This architecture involves passing the spatial input through a layer consisting of C^m functions. For instance, for C^∞ continuity, the following layer can be used:

$$\text{Layer} = \{sin(\omega x), cos(\omega x), ..., sin(k\omega x), cos(k\omega x)\} \tag{A.1}$$

While using higher order terms can help capture higher modes [25, 26], only the first two terms are used throughout this work, i.e. $k = 1$.

Appendix B. Empirical Causal Weighting

The temporal causality in enforced in this work by using multiple time segments and solving the problem sequentially. However, this does not preclude the use of other schemes to enforce causality within a time step. To this end, an empirical weighting scheme is used in which the loss function is multiplied by term that linearly decreases in time. In particular, the following form proposed by NVIDIA Modulus [43] is used:

$$\lambda_P = C_T \left(1 - \frac{t}{t_{\max}}\right) + 1 \tag{B.1}$$

With the exception of the Jerky dynamics problem, $t_{\max}$ is taken to be the maximum time in the problem, not in the time step. For all problems, $C_T = 10$ was used as causal weighting parameter.

Appendix C. Hyper-Parameters

A list of hyper-parameters used in various problems is found in Table C.9.

Table C.9: Hyper-parameters for numerical experiments. NN denotes Neural network, MB denotes Mini-batch, and FB denotes Full-batch.

Case	NN depth	NN width	Batch type and size	Adam step size	Adam iteration	L-BFGS iteration	λ_I
Advection	4	32	MB, 128	5×10^{-3}	10,000	1,000	1
Wave	4	32	MB, 128	5×10^{-3}	20,000	1,000	1
Allen-Cahn	4	32	MB, 256	5×10^{-3}	20,000	200	100
KdV	4	32	MB, 256	2×10^{-3}	20,000	200	100
Jerk	3	16	FB, 2001	2×10^{-3}	10,000	300	10

References

[1] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[2] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (PINNs) for fluid mechanics: A review," *Acta Mechanica Sinica*, vol. 37, no. 12, pp. 1727–1738, 2021.

[3] M. Mahmoudabadbozchelou, G. E. Karniadakis, and S. Jamali, "nn-PINNs: Non-newtonian physics-informed neural networks for complex fluid modeling," *Soft Matter*, vol. 18, no. 1, pp. 172–185, 2022.

[4] H. Eivazi, M. Tahani, P. Schlatter, and R. Vinuesa, "Physics-informed neural networks for solving reynolds-averaged navier–stokes equations," *Physics of Fluids*, vol. 34, no. 7, 2022.

[5] S. K. Biswas and N. Anand, "Three-dimensional laminar flow using physics informed deep neural networks," *Physics of Fluids*, vol. 35, no. 12, 2023.

[6] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks for heat transfer problems," *Journal of Heat Transfer*, vol. 143, no. 6, p. 060801, 2021.

[7] Y. Ghaffari Motlagh, P. K. Jimack, and R. de Borst, "Deep learning phase-field model for brittle fractures," *International Journal for Numerical Methods in Engineering*, vol. 124, no. 3, pp. 620–638, 2023.

[8] A. Sarma, C. Annavarapu, P. Roy, S. Jagannathan, and D. Valiveti, "Variational interface physics informed neural networks (VI-PINNs) for heterogeneous subsurface systems," in *ARMA US Rock Mechanics/Geomechanics Symposium*, pp. ARMA–2023, ARMA, 2023.

[9] S. A. Niaki, E. Haghighat, T. Campbell, A. Poursartip, and R. Vaziri, "Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture," *Computer Methods in Applied Mechanics and Engineering*, vol. 384, p. 113959, 2021.

[10] D. N. Tanyu, J. Ning, T. Freudenberg, N. Heilenkötter, A. Rademacher, U. Iben, and P. Maass, "Deep learning methods for partial differential equations and related parameter identification problems," *Inverse Problems*, vol. 39, no. 10, p. 103001, 2023.

[11] L. Yang, X. Meng, and G. E. Karniadakis, "B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data," *Journal of Computational Physics*, vol. 425, p. 109913, 2021.

[12] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson, "Physics-informed neural networks with hard constraints for inverse design," *SIAM Journal on Scientific Computing*, vol. 43, no. 6, pp. B1105–B1132, 2021.

[13] A. D. Jagtap, D. Mitsotakis, and G. E. Karniadakis, "Deep learning of inverse water waves problems using multi-fidelity data: Application to

serre–green–naghdi equations," *Ocean Engineering*, vol. 248, p. 110775, 2022.

[14] Y. Chen, L. Lu, G. E. Karniadakis, and L. Dal Negro, "Physics-informed neural networks for inverse problems in nano-optics and metamaterials," *Optics express*, vol. 28, no. 8, pp. 11618–11633, 2020.

[15] A. Serebrennikova, R. Teubler, L. Hoffellner, E. Leitner, U. Hirn, and K. Zojer, "Physics informed neural networks reveal valid models for reactive diffusion of volatiles through paper," *Chemical Engineering Science*, vol. 285, p. 119636, 2024.

[16] J. N. Reddy, N. Anand, and P. Roy, *Finite element and finite volume methods for heat transfer and fluid dynamics*. Cambridge University Press, 2022.

[17] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney, "Characterizing possible failure modes in physics-informed neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 26548–26560, 2021.

[18] S. Wang, S. Sankaran, and P. Perdikaris, "Respecting causality is all you need for training physics-informed neural networks," *arXiv preprint arXiv:2203.07404*, 2022.

[19] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics–informed neural networks: Where we are and what's next," *Journal of Scientific Computing*, vol. 92, no. 3, p. 88, 2022.

[20] S. Dong and N. Ni, "A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks," *Journal of Computational Physics*, vol. 435, p. 110242, 2021.

[21] N. Sukumar and A. Srivastava, "Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 389, p. 114333, 2022.

[22] C. L. Wight and J. Zhao, "Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks," *arXiv preprint arXiv:2007.04542*, 2020.

[23] R. Mattey and S. Ghosh, "A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations," *Computer Methods in Applied Mechanics and Engineering*, vol. 390, p. 114474, 2022.

[24] A. Bihlo and R. O. Popovych, "Physics-informed neural networks for the shallow-water equations on the sphere," *Journal of Computational Physics*, vol. 456, p. 111024, 2022.

[25] S. Wang, S. Sankaran, H. Wang, and P. Perdikaris, "An expert's guide to training physics-informed neural networks," *arXiv preprint arXiv:2308.08468*, 2023.

[26] M. Penwarden, A. D. Jagtap, S. Zhe, G. E. Karniadakis, and R. M. Kirby, "A unified scalable framework for causal sweeping strategies for physics-informed neural networks (PINNs) and their temporal decompositions," *arXiv preprint arXiv:2302.14227*, 2023.

[27] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, *JAX: composable transformations of Python+NumPy programs*, 2018.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[29] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

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

[31] T. A. Driscoll, N. Hale, and L. N. Trefethen, "Chebfun guide," 2014.

[32] U. Braga-Neto, "Characteristics-informed neural networks for forward and inverse hyperbolic problems," *arXiv preprint arXiv:2212.14012*, 2022.

[33] S. M. Allen and J. W. Cahn, "A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening," *Acta metallurgica*, vol. 27, no. 6, pp. 1085–1095, 1979.

[34] R. M. Miura, "The Korteweg–deVries equation: a survey of results," *SIAM review*, vol. 18, no. 3, pp. 412–459, 1976.

[35] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of atmospheric sciences*, vol. 20, no. 2, pp. 130–141, 1963.

[36] S. Smale and M. W. Hirsch, *Differential equations, dynamical systems, and linear algebra*, vol. 60. Elsevier, 1974.

[37] O. E. Rössler, "Continuous chaos - four prototype equations," *Annals of the New York Academy of Sciences*, vol. 316, no. 1, pp. 376–392, 1979.

[38] J. Sprott, "Some simple chaotic jerk functions," *American Journal of Physics*, vol. 65, no. 6, pp. 537–543, 1997.

[39] R. Eichhorn, S. J. Linz, and P. Hänggi, "Transformations of nonlinear dynamical systems to jerky motion and its application to minimal chaotic flows," *Physical Review E*, vol. 58, no. 6, p. 7151, 1998.

[40] J. Sprott, "Simplifications of the Lorenz attractor," *Nonlinear dynamics, psychology, and life sciences*, vol. 13, no. 3, p. 271, 2009.

[41] S. Wang, X. Yu, and P. Perdikaris, "When and why PINNs fail to train: A neural tangent kernel perspective," *Journal of Computational Physics*, vol. 449, p. 110768, 2022.

[42] Z. Xiang, W. Peng, X. Liu, and W. Yao, "Self-adaptive loss balanced physics-informed neural networks," *Neurocomputing*, vol. 496, pp. 11–34, 2022.

[43] NVIDIA, Santa Clara, CA, *Nvidia Modulus*, 2022.