Integral Regularization PINNs for Evolution Equations

Xiaodong Feng[1], Haojiong Shangguan[1,*], Tao Tang[2,1,3] and Xiaoliang Wan[4]

[1] *Faculty of Science and Technology, Beijing Normal-Hong Kong Baptist University, Zhuhai 519087, China.*
[2] *School of Mathematics and Statistics, Guangzhou Nanfang College, Guangzhou 510970, China.*
[3] *Zhuhai SimArk Technology Co., LTD, Zhuhai 519085, China.*
[4] *Department of Mathematics and Center for Computation and Technology, Louisiana State University, Baton Rouge 70803, USA.*

Abstract. Evolution equations, including both ordinary differential equations (ODEs) and partial differential equations (PDEs), play a pivotal role in modeling dynamic systems. However, achieving accurate long-time integration for these equations remains a significant challenge. While physics-informed neural networks (PINNs) provide a mesh-free framework for solving PDEs, they often suffer from temporal error accumulation, which limits their effectiveness in capturing long-time behaviors. To alleviate this issue, we propose integral regularization PINNs (IR-PINNs), a novel approach that enhances temporal accuracy by incorporating an integral-based residual term into the loss function. This method divides the entire temporal interval into smaller subintervals and enforces integral constraints either within each subinterval or across intervals extending from the initial moment to the current one, thereby improving the resolution and correlation of temporal dynamics. Furthermore, IR-PINNs leverage adaptive sampling to dynamically refine the distribution of collocation points based on the evolving solution, ensuring higher accuracy in regions with sharp gradients or rapid variations. Numerical experiments on benchmark problems demonstrate that IR-PINNs outperform original PINNs and other state-of-the-art methods in capturing long-time behaviors, offering a robust and accurate solution for evolution equations.

AMS subject classifications: 65L04, 65M99, 68T07

Key words: Deep learning, evolution equation, long-time integration, adaptive sampling.

---

*Corresponding author. *Email addresses:* `xiaodongfeng@uic.edu.cn` (X. Feng), `shangguanhaojiong@uic.edu.cn` (H. Shangguan), `ttang@nfu.edu.cn` (T. Tang), `xlwan@lsu.edu` (X. Wan)

356

1   Introduction

Evolution equations are fundamental in modeling a wide range of physical, biological, and engineering phenomena, spanning from fluid dynamics to material science [5]. Among these, evolution equations, characterized by their dependence on both temporal and spatial variables, play a crucial role in describing dynamic systems. However, achieving accurate and efficient solutions to these equations remains a significant challenge, particularly for problems requiring long-time integration or involving complex and high-dimensional domains.

In recent years, physics-informed neural networks (PINNs) [24] have emerged as a promising alternative for solving evolution equations. By embedding the governing equations into the loss function and leveraging the expressive power of neural networks, PINNs can approximate solutions without relying on predefined grids or explicit discretization schemes. Despite their versatility, challenges remain in applying PINNs to evolution equations [6, 17, 25, 28, 30]. One of the most pressing issues is the accumulation of temporal errors during long-time integration, prompting significant efforts to address this limitation. Several training strategies have been proposed to improve temporal accuracy, including sequential learning [20, 31], causal training [14, 22, 27] and operator learning [18, 26, 29, 32]. Additionally, hybrid strategies have been developed to combine classical numerical methods with deep learning techniques. These approaches either adapt neural networks to augment classical PDE solvers [2, 8] or incorporate classical numerical methods to enhance the performance of PINNs [4, 10, 13].

For evolution equations, the solution at any given time is inherently dependent on its state at previous times, reflecting strong temporal correlation. However, original PINNs treat temporal collocation points in isolation, failing to explicitly account for these correlations. This limitation often leads to challenges in capturing long-time dynamics and results in temporal error accumulation. Inspired by the integral form of evolution equations, we introduce a regularization term into the training process, proposing a novel framework termed integral regularization PINNs (IR-PINNs). Our main contributions can be summarized as:

- We propose integral regularization PINNs (IR-PINNs) for evolution equations by dividing the entire temporal interval into smaller subintervals, reformulating the evolution equation into an integral form, and incorporating an integral-based residual term into the loss function. This approach enhances temporal accuracy by enforcing constraints over specific temporal subintervals, thereby improving the resolution and correlation of temporal dynamics.

- We extend IR-PINNs with an adaptive sampling strategy, which dynamically refines the distribution of spatial collocation points, ensuring higher accuracy in regions with sharp gradients or rapid variations.

- We conduct numerical experiments on benchmark problems, including both linear

and nonlinear evolution equations, to demonstrate the effectiveness of IR-PINNs in capturing long-time behaviors and outperforming original PINNs and other state-of-the-art methods.

This paper is structured as follows. In Section 2, we provide a brief overview of PINNs, followed by a critical analysis of their limitations in solving evolution equations through a simple case study. In Section 3, we introduce the proposed IR-PINNs framework and further enhance it by incorporating an adaptive sampling strategy to improve computational efficiency. In Section 4, we demonstrate comprehensive numerical experiments with detailed discussions, highlighting the advantages of the proposed method. Finally, Section 5 summarizes the paper and discusses potential future research directions.

2 Physics-informed neural networks

We begin with a brief overview of physics-informed neural networks (PINNs) [24]. On an open domain $\Omega \subset \mathbb{R}^d$, consider a general time-dependent PDE

$$u_t(t,\boldsymbol{x})+\mathcal{N}[u](t,\boldsymbol{x})=f(t,\boldsymbol{x}), \quad t\in[0,T], \ \boldsymbol{x}\in\Omega, \tag{2.1}$$

subject to the initial and boundary conditions

$$\begin{aligned} u(0,\boldsymbol{x})&=g(\boldsymbol{x}), \quad \boldsymbol{x}\in\Omega, \\ \mathcal{B}[u](t,\boldsymbol{x})&=b(t,\boldsymbol{x}), \quad t\in[0,T], \ \boldsymbol{x}\in\partial\Omega, \end{aligned} \tag{2.2}$$

where $\mathcal{N}[\cdot]$ is a linear or nonlinear differential operator, and $\mathcal{B}[\cdot]$ is a boundary operator corresponding to Dirichlet, Neumann, Robin or periodic boundary conditions.

Our goal is to seek a neural network $u_N(t,\boldsymbol{x};\theta)$, where θ denotes trainable parameters, to approximate the solution $u(t,\boldsymbol{x})$ by minimizing the following composite loss function

$$\begin{aligned} \mathcal{L}(\theta) &= \mathcal{L}_r(\theta)+\lambda_1\mathcal{L}_{ic}(\theta)+\lambda_2\mathcal{L}_{bc}(\theta) \\ &= \|r(t,\boldsymbol{x};\theta)\|_{L^2([0,T]\times\Omega)}^2+\lambda_1\|u_N(0,\boldsymbol{x};\theta)-g(\boldsymbol{x})\|_{L^2(\Omega)}^2 \\ &\quad +\lambda_2\|\mathcal{B}[u_N](t,\boldsymbol{x};\theta)-b(t,\boldsymbol{x})\|_{L^2([0,T]\times\partial\Omega)}^2, \end{aligned} \tag{2.3}$$

where λ_1 and λ_2 are hyperparameters to balance the interplay between different loss terms during the training process, $\|\cdot\|_{L^2}$ represents L^2 norm and

$$r(t,\boldsymbol{x};\theta)=(u_N)_t(t,\boldsymbol{x};\theta)+\mathcal{N}[u_N](t,\boldsymbol{x};\theta)-f(t,\boldsymbol{x}). \tag{2.4}$$

In practice, we often choose three sets of uniformly distributed collocation points $S_r=\{(t_r^{(i)},\boldsymbol{x}_r^{(i)})\}_{i=1}^{N_r}$ on $[0,T]\times\Omega$, $S_{ic}=\{\boldsymbol{x}_{ic}^{(i)}\}_{i=1}^{N_{ic}}$ on Ω and $S_{bc}=\{(t_{bc}^{(i)},\boldsymbol{x}_{bc}^{(i)})\}_{i=1}^{N_{bc}}$ on $[0,T]\times\partial\Omega$ and the loss function is discretized numerically as

$$\begin{aligned} \widehat{\mathcal{L}}(\theta) &= \widehat{\mathcal{L}}_r(\theta)+\hat{\lambda}_1\widehat{\mathcal{L}}_{ic}(\theta)+\hat{\lambda}_2\widehat{\mathcal{L}}_{bc}(\theta) \\ &= \|r(t,\boldsymbol{x};\theta)\|_{S_r}^2+\hat{\lambda}_1\|u_N(0,\boldsymbol{x};\theta)-g(\boldsymbol{x})\|_{S_{ic}}^2+\hat{\lambda}_2\|\mathcal{B}[u_N](t,\boldsymbol{x};\theta)-b(t,\boldsymbol{x})\|_{S_{bc}}^2, \end{aligned} \tag{2.5}$$

where

$$\|r(t,\boldsymbol{x};\theta)\|^2_{S_r} = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|(u_N)_t(t_r^{(i)},\boldsymbol{x}_r^{(i)};\theta)+\mathcal{N}[u_N](t_r^{(i)},\boldsymbol{x}_r^{(i)};\theta)-f(t_r^{(i)},\boldsymbol{x}_r^{(i)})\right|^2,$$

$$\|u_N(0,\boldsymbol{x};\theta)-g(\boldsymbol{x})\|^2_{S_{ic}} = \frac{1}{N_{ic}}\sum_{i=1}^{N_{ic}}\left|u_N(0,\boldsymbol{x}_{ic}^{(i)};\theta)-g(\boldsymbol{x}_{ic}^{(i)})\right|^2, \tag{2.6}$$

$$\|\mathcal{B}[u_N](t,\boldsymbol{x};\theta)-b(t,\boldsymbol{x})\|^2_{S_{bc}} = \frac{1}{N_{bc}}\sum_{i=1}^{N_{bc}}\left|\mathcal{B}[u_N](t_{bc}^{(i)},\boldsymbol{x}_{bc}^{(i)};\theta)-b(t_{bc}^{(i)},\boldsymbol{x}_{bc}^{(i)})\right|^2,$$

and $\hat{\lambda}_1 = \frac{\lambda_1}{T}$, $\hat{\lambda}_2 = \frac{\lambda_2|\partial\Omega|}{|\Omega|}$ such that $\widehat{\mathcal{L}}(\theta)$ is a Monte Carlo approximation of $\mathcal{L}(\theta)$ up to a constant scaling factor $T|\Omega|$.

To provide intuition for the key ideas behind our method, we first analyze a simple case by considering the following ordinary differential equation (ODE):

$$\begin{aligned} u'(t) &= \lambda u(t), \quad t \in [0,T], \\ u(0) &= u_0, \end{aligned} \tag{2.7}$$

where the exact solution is $u(t) = u_0 e^{\lambda t}$. Conventional PINNs are known to struggle with large values of λT since the ODE system will become stiff. Here we set $\lambda = 1$, $T = 5$ and $u_0 = 1$. In the framework of PINNs, we represent the latent variable $u(t)$ by a fully-connected neural network $u_N(t;\theta)$ with Tanh activation function, 3 hidden layers, and 64 neurons per hidden layer, and exactly impose the initial condition by

$$\tilde{u}_N(t;\theta) = tu_N(t;\theta)+u_0. \tag{2.8}$$

Then the discretized loss function simplifies to

$$\widehat{\mathcal{L}}(\theta) = \widehat{\mathcal{L}}_r(\theta) = \|r(t;\theta)\|^2_{S_r} = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|\tilde{u}'_N(t_r^{(i)};\theta)-\lambda\tilde{u}_N(t_r^{(i)};\theta)\right|^2, \tag{2.9}$$

where $S_r = \{t_r^{(i)}\}_{i=1}^{N_r}$ are uniform mesh points on $[0,T]$ and $N_r = 40$. The Adam algorithm is used to optimize the loss function with an initial learning rate 10^{-3}, which decays every 1,000 epochs at a rate of 0.9. The maximum number of training epochs is 80,000.

As shown in Fig. 1, although the residuals on the training set are sufficiently small, conventional PINNs fail to capture the exponential growth of the solution. This suggests that the model tends to overfit when the temporal collocation points are insufficiently dense.

To investigate the failure mode of PINNs in this scenario, we analyze the role of the automatic differentiation technique. While automatic differentiation can accurately compute the derivative at individual temporal collocation points, it does not account for the information in the small neighborhood around each collocation point. This may lead to
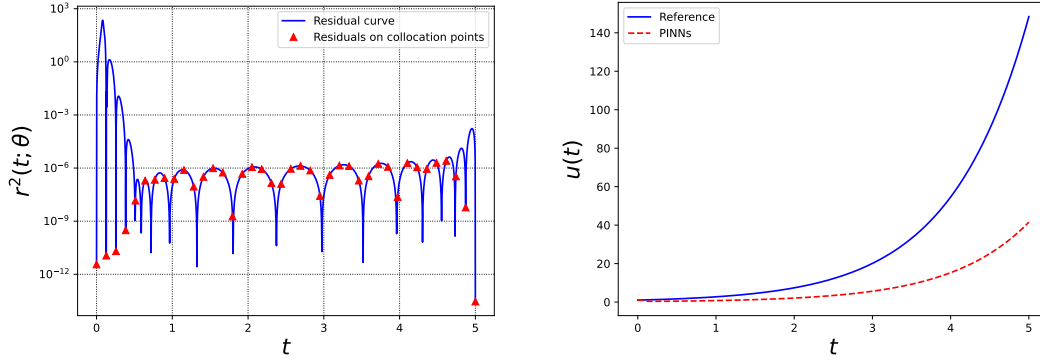
Figure 1: *Simple ODE*: Left: Residual curve. Right: Reference solution versus numerical solution given by PINNs. Relative L_2 error: 7.2033×10^{-1}.

relatively large residuals in regions outside the training points. From the residual curve, it is evident that when $t < 0.5$, the overall residual is large. Given the propagation property of the evolution equation, PINNs cannot converge to the correct solution, even if the residual later becomes small. This analysis suggests that when solving long-time integration problems, it is crucial to consider the correlations between collocation points to reduce the residuals across the entire temporal interval, a principle that traditional numerical schemes often emphasize. In the next section, we propose a simple and effective strategy based on integral form of the evolution equation to strengthen the temporal correlation and mitigate the overfitting problem.

3   Methodology

3.1   Integral regularization PINNs (IR-PINNs)

We propose a novel method for solving evolution equations, termed integral regularization PINNs (IR-PINNs), which enhances the standard loss function with a regularization term designed to inherently capture temporal correlation in the solution.

More precisely, we consider the integral form of the general time-dependent PDE (2.1) over a temporal interval $[a,b]$ $(a < b)$:

$$u(b,\boldsymbol{x}) - u(a,\boldsymbol{x}) + \int_a^b \mathcal{N}[u](t,\boldsymbol{x})\mathrm{d}t = \int_a^b f(t,\boldsymbol{x})\mathrm{d}t, \quad \forall a,b \in [0,T], \ \boldsymbol{x} \in \Omega. \quad (3.1)$$

Then we can define an integral-based residual function corresponding to the above equation (3.1) as follows:

$$r_{\mathrm{int}}(a,b,\boldsymbol{x}) = \frac{u(b,\boldsymbol{x}) - u(a,\boldsymbol{x}) + \int_a^b \left(\mathcal{N}[u](t,\boldsymbol{x}) - f(t,\boldsymbol{x})\right)\mathrm{d}t}{b-a}. \quad (3.2)$$

Given a numerical quadrature scheme defined by a set of nodes $Q = \{s_i\}_{i=1}^M$ and their associated weights $\{w_i\}_{i=1}^M$, the integral term can be approximated as follows

$$I_Q[\mathcal{N}[u](t,\boldsymbol{x}) - f(t,\boldsymbol{x})] = \sum_{i=1}^M w_i \left(\mathcal{N}[u](s_i,\boldsymbol{x}) - f(s_i,\boldsymbol{x})\right). \tag{3.3}$$

The most straightforward approach involves imposing an integral-based residual constraint over the entire temporal interval. However, when the temporal interval is long or the solution exhibits complex behaviors, a single integral-based residual constraint may lead to insufficient accuracy of numerical integration and a relatively weak impact of the regularization term during the training process. This observation leads to our first regularization strategy, which divides the entire temporal interval $[0,T]$ into several subintervals, denoted by $0 = t_0 < t_1 < t_2 < \cdots < t_N = T$, and applies independent integral-based residual constraints to each subinterval:

$$\text{(IR-PINNs1)} \quad \mathcal{L}_{\text{int}}^{(\text{R1})}(\theta) = \frac{1}{N} \sum_{k=0}^{N-1} \|r_{\text{int}}(t_k, t_{k+1}, \boldsymbol{x};\theta)\|_{L^2(\Omega)}^2.$$

Building upon this foundation, we further develop the second regularization term to incorporate temporal causality:

$$\text{(IR-PINNs2)} \quad \mathcal{L}_{\text{int}}^{(\text{R2})}(\theta) = \frac{1}{N} \sum_{k=0}^{N-1} \|r_{\text{int}}(t_0, t_{k+1}, \boldsymbol{x};\theta)\|_{L^2(\Omega)}^2.$$

This formulation features accumulative integral-based residual constraints propagating from the initial moment, thereby implicitly emphasizing the importance of earlier temporal subintervals in the regularization scheme.

Given numerical quadrature points $Q_r^{(k)} = \{t_r^{(k,i)}\}_{i=1}^M$ for each subinterval $[t_k, t_{k+1}]$, the total numerical quadrature points set is $Q = \cup_{k=0}^{N-1} Q_r^{(k)}$ and the new residual loss function can be written as

$$\mathcal{L}_{\text{couple}}(\theta) = \mathcal{L}_r(\theta) + \alpha \mathcal{L}_{\text{int}}(\theta), \tag{3.4}$$

where $\mathcal{L}_r(\theta)$ is modified to

$$\mathcal{L}_r(\theta) = \frac{1}{NM} \sum_{k=0}^{N-1} \sum_{i=1}^M \|r(t_r^{(k,i)}, \boldsymbol{x};\theta)\|_{L^2(\Omega)}^2, \tag{3.5}$$

$\mathcal{L}_{\text{int}}(\theta)$ stands for $\mathcal{L}_{\text{int}}^{(\text{R1})}(\theta)$ or $\mathcal{L}_{\text{int}}^{(\text{R2})}(\theta)$ and α is a hyperparameter to balance the two residual loss terms.

In practice, we generate uniformly distributed points $\{\boldsymbol{x}_r^{(j)}\}_{j=1}^{N_x}$ in spatial direction and obtain two sets of collocation points $S_r = \cup_{k=0}^{N-1} \cup_{i=1}^M S_r^{(k,i)} = \cup_{k=0}^{N-1} \cup_{i=1}^M \{(t_r^{(k,i)}, \boldsymbol{x}_r^{(j)})\}_{j=1}^{N_x}$ and $S_{\text{int}} = \cup_{k=0}^{N-1} S_{\text{int}}^{(k)} = \cup_{k=0}^{N-1} \{\boldsymbol{x}_r^{(j)}\}_{j=1}^{N_x}$, the discretized residual loss function is

$$\widehat{\mathcal{L}}_{\text{couple}}(\theta) = \widehat{\mathcal{L}}_r(\theta) + \hat{\alpha} \widehat{\mathcal{L}}_{\text{int}}(\theta), \tag{3.6}$$

where

$$\widehat{\mathcal{L}}_r(\theta) = \frac{1}{NM} \sum_{k=0}^{N-1} \sum_{i=1}^{M} \|r(t_r^{(k,i)}, x; \theta)\|_{S_r^{(k,i)}}^2, \tag{3.7}$$

$$\widehat{\mathcal{L}}_{\text{int}}^{(\text{R1})}(\theta) = \frac{1}{N} \sum_{k=0}^{N-1} \|r_{\text{int}}(t_k, t_{k+1}, x; \theta)\|_{S_{\text{int}}^{(k)}}^2, \quad \widehat{\mathcal{L}}_{\text{int}}^{(\text{R2})}(\theta) = \frac{1}{N} \sum_{k=0}^{N-1} \|r_{\text{int}}(t_0, t_{k+1}, x; \theta)\|_{S_{\text{int}}^{(k)}}^2. \tag{3.8}$$

More precisely,

$$\|r(t_r^{(k,i)}, x; \theta)\|_{S_r^{(k,i)}}^2 = \frac{1}{N_x} \sum_{j=1}^{N_x} \left| (u_N)_t(t_r^{(k,i)}, x_r^{(j)}; \theta) + \mathcal{N}[u_N](t_r^{(k,i)}, x_r^{(j)}; \theta) - f(t_r^{(k,i)}, x_r^{(j)}) \right|^2, \tag{3.9}$$

for all $0 \le k \le N-1$, $1 \le i \le M$,

$$\|r_{\text{int}}(t_m, t_n, x; \theta)\|_{S_{\text{int}}^{(k)}}^2$$

$$= \frac{1}{N_x} \sum_{j=1}^{N_x} \left| \frac{u_N(t_n, x_r^{(j)}; \theta) - u_N(t_m, x_r^{(j)}; \theta) + \sum_{k=m}^{n-1} I_{Q_r^{(k)}}[\mathcal{N}[u_N](t, x_r^{(j)}; \theta) - f(t, x_r^{(j)})]}{t_n - t_m} \right|^2, \tag{3.10}$$

for all $0 \le m < n \le N$ and $u_N(t_0, x; \theta)$ can be calculated by the initial condition. Here $\hat{\alpha} = \alpha$ such that $\widehat{\mathcal{L}}_{\text{couple}}(\theta)$ is a Monte Carlo approximation of $\mathcal{L}_{\text{couple}}(\theta)$ up to a constant scaling factor $|\Omega|$. Finally we obtain an estimator $\theta^\star$ by solving the following optimization problem

$$\theta^\star = \underset{\theta}{\arg\min} \, \widehat{\mathcal{L}}(\theta) = \underset{\theta}{\arg\min} \left(\widehat{\mathcal{L}}_{\text{couple}}(\theta) + \hat{\lambda}_1 \widehat{\mathcal{L}}_{ic}(\theta) + \hat{\lambda}_2 \widehat{\mathcal{L}}_{bc}(\theta) \right). \tag{3.11}$$

We now proceed to implement IR-PINNs to solve the simple ODE problem described in the preceding section. The network architecture, number of collocation points, training parameters, maximum number of iterations, and optimizer remain unchanged. The principle enhancement involves the incorporation of an integral-based regularization term into the loss function. Here we set $\alpha = 1$ and employ the trapezoidal rule to approximate the integral in the regularization term.

The results of IR-PINNs1 and IR-PINNs2 are summarized in Fig. 2 and Fig. 3, respectively. One can see that the predicted solutions achieve an excellent agreement with the reference solution, yielding a reduction in relative L_2 error of two orders of magnitude compared to PINNs. This improvement can be primarily attributed to the incorporation of the regularization term, which inherits the benefits of traditional methods by strengthening the correlations among collocation points, effectively reducing residuals, particularly in the earlier time periods. This ensures proper temporal propagation of the governing equation's information during the training process. It is noteworthy that, due to the exponential growth nature of the solution, the residual range of the numerical solution gradually increases over time, which is entirely expected.
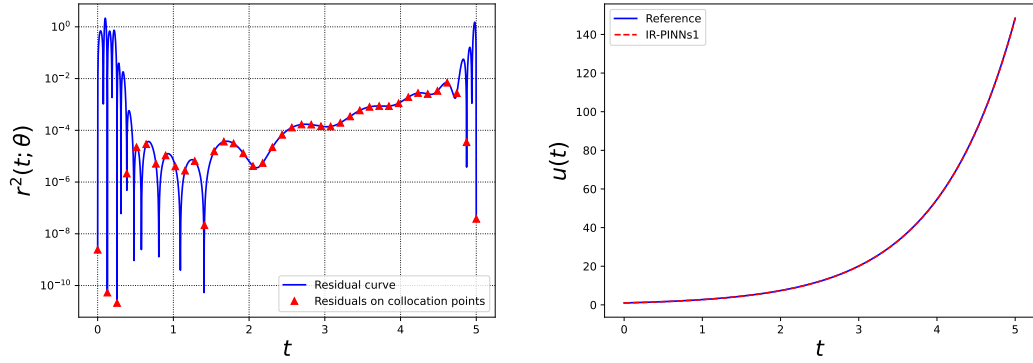
Figure 2: *Simple ODE*: Left: Residual curve. Right: Reference solution versus numerical solution given by IR-PINNs1. Relative L_2 error: 1.3156×10^{-3}.
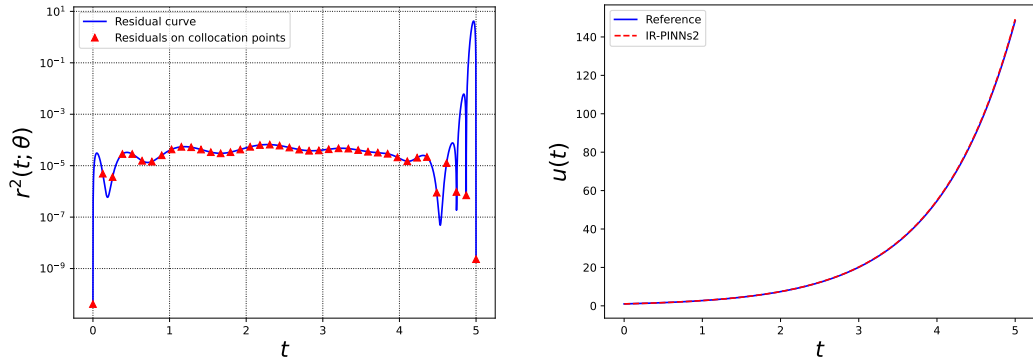

Figure 3: *Simple ODE*: Left: Residual curve. Right: Reference solution versus numerical solution given by IR-PINNs2. Relative L_2 error: 4.9012×10^{-3}.

The choice of number of subintervals N: The selection of N significantly influences both the numerical precision and the efficacy of the regularization process. With a fixed number of quadrature points for each subinterval, a larger N is essential for accurate numerical integration, particularly when the temporal interval is long or the solution exhibits complex behaviors. Additionally, N determines the number of integral-based residual constraints, and increasing it appropriately enhances the impact of regularization term on the training process, thereby improving the ability of model to capture temporal correlation.

The choice of hyperparameter α: The hyperparameter α determines the balance between the two residual loss terms $\mathcal{L}_r(\theta)$ and $\mathcal{L}_{\text{int}}(\theta)$ and plays a critical role in the training process of the neural network. If α is too small, the method degenerates to the original PINNs, where the regularization term has almost no effect on the training process. On

the other hand, if α is too large, the regularization term dominates the training process. However, due to limitations in integral accuracy and optimization difficulty, this can lead to unstable training and hinder the achievement of desired accuracy. Therefore, selecting an appropriate α is crucial. Here we propose a simple strategy for selecting α. From the mean value theorem, there exists some $\xi \in [a,b]$ such that

$$
\begin{aligned}
r_{\text{int}}(a,b,\boldsymbol{x}) &= \frac{u(b,\boldsymbol{x}) - u(a,\boldsymbol{x}) + \int_a^b (\mathcal{N}[u](t,\boldsymbol{x}) - f(t,\boldsymbol{x}))\mathrm{d}t}{b-a} \\
&= \frac{\int_a^b (u_t(t,\boldsymbol{x}) + \mathcal{N}[u](t,\boldsymbol{x}) - f(t,\boldsymbol{x}))\mathrm{d}t}{b-a} \\
&= u_t(\xi,\boldsymbol{x}) + \mathcal{N}[u](\xi,\boldsymbol{x}) - f(\xi,\boldsymbol{x}) \\
&= r(\xi,\boldsymbol{x}).
\end{aligned}
\tag{3.12}
$$

This implies that the two residuals r and r_{int} are of similar order of magnitude, suggesting $\alpha = 1$ as a natural choice. Certainly, a more effective approach would be to adaptively adjust the weight α, as discussed in [28, 30]. However, this is not the primary focus of the current work. Therefore, for all numerical experiments conducted in this study, we consistently set $\alpha = 1$ as a fixed value.

Computational cost of regularization term: While introducing a new regularization term, it is essential to consider its computational cost, particularly in the context of numerical quadrature. From Eqs. (3.9) and (3.10), we observe that the set of residual collocation points S_r is used for discretizing the two residual loss terms. Before each update of the loss function, we can compute and store the following values:

$$
\mathcal{N}[u_N](t_r^{(i)}, \boldsymbol{x}_r^{(j)}; \theta) - f(t_r^{(i)}, \boldsymbol{x}_r^{(j)}), \quad \forall i = 1, \cdots, N_t, \ j = 1, \cdots, N_x.
\tag{3.13}
$$

These precomputed values can then be reused to evaluate the two residual loss terms, minimizing redundant computations. As a result, the additional computational cost is negligible and remains within acceptable limits.

The schematic diagram of the proposed approach is shown in Fig. 4, and the corresponding algorithm is summarized in Algorithm 1.

3.2 Adaptive sampling strategy for IR-PINNs

In this section, we present an extension of the integral regularization PINNs (IR-PINNs) by incorporating the adaptive sampling strategy proposed in [10]. This extension further enhances the performance of IR-PINNs when addressing more complex and challenging scenarios, such as low-regularity problems or unbounded domain problems.

The basic idea is to construct a continuous joint probability density model $p(t,\boldsymbol{x};\theta_f)$ to approximate the distribution induced by the residual $r^2(t,\boldsymbol{x};\theta)$ in (2.4) as

$$
p(t,\boldsymbol{x};\theta_f) = p_{\text{poly}}(t;\theta_{f,1}) p_{\text{B-KRnet}}(\boldsymbol{x}|t;\theta_{f,2}),
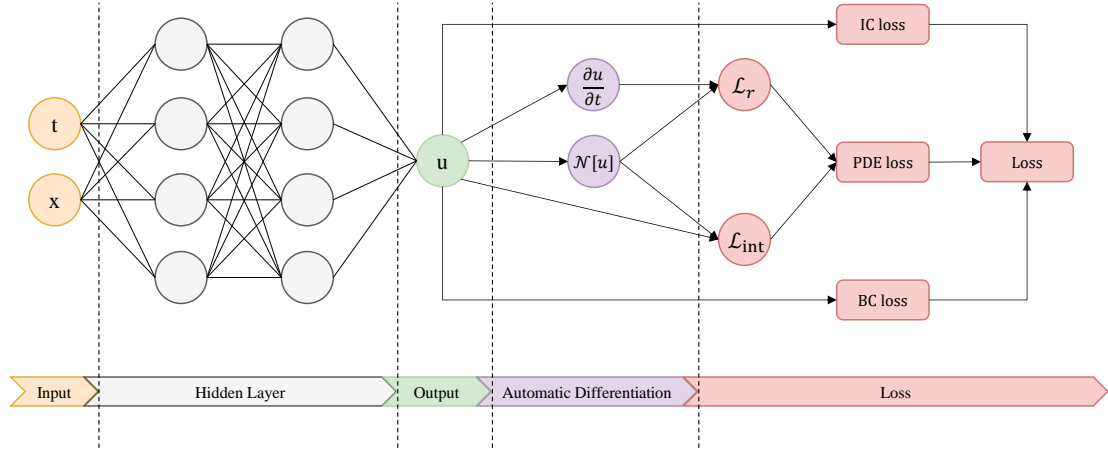\tag{3.14}
$$

Figure 4: Schematic diagram of integral regularization PINNs.

Algorithm 1 Solving evolution equations via integral regularization PINNs

---

Input: Terminal time T, number of subintervals N, numerical quadrature points set Q, number of training epochs N_e, initial learning rate $l_r^{(0)}$, decay rate η, step size n_s.

Generate the training datasets S_r, S_{int}, S_{ic} and S_{bc}.

Set l_r to $l_r^{(0)}$.

for $e = 1, \cdots, N_e$ **do**

    if $e\%n_s == 0$ **then**

        $l_r = \eta * l_r$.

    end if

    Compute the shared values in (3.13) using the datasets S_r and S_{int}.

    Compute the residual loss function $\widehat{\mathcal{L}}_{r,\text{new}}(\theta)$ using the datasets S_r, S_{int} and shared values.

    Compute the initial and boundary loss functions $\widehat{\mathcal{L}}_{ic}(\theta)$ and $\widehat{\mathcal{L}}_{bc}(\theta)$ using the dataset S_{ic} and S_{bc}.

    Compute the total loss function $\widehat{\mathcal{L}}(\theta)$ in Eq. (3.11).

    Update θ by stochastic gradient descent method.

end for

Output: The predicted solutions $u_N(t,\boldsymbol{x};\theta^*)$.

---

where $p_{\text{poly}}(t;\theta_{f,1})$ is parameterized by the polynomial spline layer [21], $p_{\text{B-KRnet}}(\boldsymbol{x}|t;\theta_{f,2})$ is parameterized by the bounded KRnet [33] and $\theta_f = \{\theta_{f,1}, \theta_{f,2}\}$. We give a brief introduction for the polynomial spline layer and the bounded KRnet in Appendix A.

To seek the optimal parameter θ_f, we can minimize the following objective

$$
\begin{aligned}
&D_{\mathrm{KL}}(\hat{r}(t,\boldsymbol{x};\theta)||p(t,\boldsymbol{x};\theta_f)) \\
&=D_{\mathrm{KL}}\big(\hat{r}(t,\boldsymbol{x};\theta)||p_{\mathrm{poly}}(t;\theta_{f,1})p_{\mathrm{B-KRnet}}(\boldsymbol{x}|t;\theta_{f,2})\big) \\
&=\iint \hat{r}(t,\boldsymbol{x};\theta)\log(\hat{r}(t,\boldsymbol{x};\theta))\,\mathrm{d}\boldsymbol{x}\mathrm{d}t-\iint \hat{r}(t,\boldsymbol{x};\theta)\log\big(p_{\mathrm{poly}}(t;\theta_{f,1})p_{\mathrm{B-KRnet}}(\boldsymbol{x}|t;\theta_{f,2})\big)\,\mathrm{d}\boldsymbol{x}\mathrm{d}t,
\end{aligned}
\tag{3.15}
$$

where D_{KL} indicates the Kullback-Leibler (KL) divergence and $\hat{r}(t,\boldsymbol{x};\theta)$ is the distribution induced by $r^2(t,\boldsymbol{x};\theta)$ and the well-trained parameters $\theta_f^* = \{\theta_{f,1}^*,\theta_{f,2}^*\}$ is obtained. The first term on the right-hand side in (3.15) represents the negative differential entropy of $\hat{r}(t,\boldsymbol{x};\theta)$, which does not affect the optimization process with respect to θ_f. So minimizing the KL divergence is equivalent to minimizing the cross entropy between $\hat{r}(t,\boldsymbol{x};\theta)$ and $p(t,\boldsymbol{x};\theta_f)$

$$
H(\hat{r}(t,\boldsymbol{x};\theta),p(t,\boldsymbol{x};\theta_f))=-\iint \hat{r}(t,\boldsymbol{x};\theta)\log\big(p_{\mathrm{poly}}(t;\theta_{f,1})p_{\mathrm{B-KRnet}}(\boldsymbol{x}|t;\theta_{f,2})\big)\,\mathrm{d}\boldsymbol{x}\mathrm{d}t.
\tag{3.16}
$$

Since the samples from $\hat{r}(t,\boldsymbol{x};\theta)$ are not available, we approximate the cross entropy up to a constant scaling factor $T|\Omega|$ using the importance sampling technique:

$$
\begin{aligned}
&H(\hat{r}(t,\boldsymbol{x};\theta),p(t,\boldsymbol{x};\theta_f)) \\
&\approx-\frac{1}{N_r}\sum_{i=1}^{N_r}\frac{\hat{r}(t_i,\boldsymbol{x}_i;\theta)}{p_{\mathrm{poly}}(t_i;\theta_{f,1})p_{\mathrm{B-KRnet}}(\boldsymbol{x}_i|t_i;\theta_{f,2})}\big(\log p_{\mathrm{poly}}(t_i;\theta_{f,1})+\log p_{\mathrm{B-KRnet}}(\boldsymbol{x}_i|t_i;\theta_{f,2})\big),
\end{aligned}
\tag{3.17}
$$

where

$$
t_i\sim p_{\mathrm{poly}}(t;\theta_{f,1}),\quad \boldsymbol{x}_i\sim p_{\mathrm{B-KRnet}}(\boldsymbol{x}|t_i;\theta_{f,2}).
\tag{3.18}
$$

More details about the adaptive sampling approach can be found in [10].

Considering that the residuals $r^2(t,\boldsymbol{x};\theta)$ are computed only at the discrete temporal points set $Q=\cup_{k=0}^{N-1}Q_r^{(k)}$, we will employ a discrete distribution $p_{\mathrm{dis}}(t;\theta_{f,1})$ based on the continuous distribution $p_{\mathrm{poly}}(t;\theta_{f,1})$:

$$
p_{\mathrm{dis}}(t;\theta_{f,1})=\begin{cases} \int_0^{(s_0+s_1)/2}p_{\mathrm{poly}}(s;\theta_{f,1})\mathrm{d}s, & t=s_0, \\ \int_{(s_{i-1}+s_i)/2}^{(s_i+s_{i+1})/2}p_{\mathrm{poly}}(s;\theta_{f,1})\mathrm{d}s, & t=s_i,\ 0<i<N, \\ \int_{(s_{N-1}+s_N)/2}^1 p_{\mathrm{poly}}(s;\theta_{f,1})\mathrm{d}s, & t=s_N, \\ 0, & \text{otherwise}, \end{cases}
\tag{3.19}
$$

where $s_0,s_1,\cdots,s_N$ are the ordered arrangement of all points in the set $\mathcal{Q}$. Then N_{new} samples $\{t^{(j)}\}_{j=1}^{N_{\mathrm{new}}}$ are generated from the discrete distribution $p_{\mathrm{dis}}(t;\theta_{f,1})$, along with their corresponding spatial points $\boldsymbol{x}^{(j)}\sim p_{\mathrm{B-KRnet}}(\boldsymbol{x}|t^{(j)};\theta_{f,2})$ for each $t^{(j)}$. Finally, we reorder
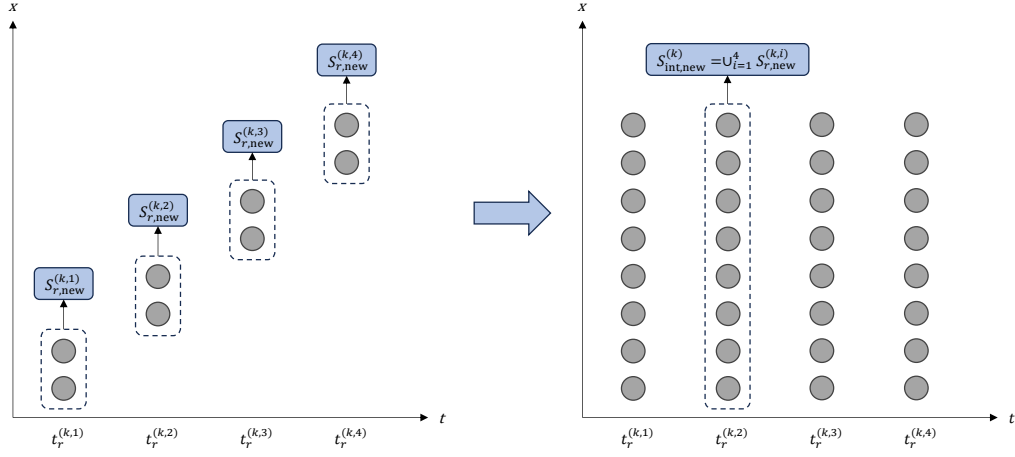
Figure 5: Schematic diagram of adaptive sampling.

the newly generated data into the original spatial domain

$$\boldsymbol{x}^{(j)} \in S_{r,\text{new}}^{(k,i)}, \quad \text{if } t^{(j)} = t_r^{(k,i)}, \; \forall j = 1, \cdots, N_{\text{new}}, \; k = 0, \cdots, N-1, \; i = 1, \cdots, M, \tag{3.20}$$

$$\boldsymbol{x}^{(j)} \in S_{\text{int,new}}^{(k)}, \quad \text{if } t^{(j)} \in T^{(k)}, \; \forall j = 1, \cdots, N_{\text{new}}, \; k = 0, \cdots, N-1, \tag{3.21}$$

where $T^{(k)} = Q_r^{(k)}$ for IR-PINNs1 and $T^{(k)} = \cup_{p=0}^k Q_r^{(p)}$ for IR-PINNs2. Now we refine the training set for the residual loss function $\widehat{\mathcal{L}}_{r,\text{new}}(\theta)$ as follows:

$$S_r = \cup_{k=0}^{N-1} \cup_{i=1}^{M} (S_r^{(k,i)} \cup S_{r,\text{new}}^{(k,i)}), \tag{3.22}$$

$$S_{\text{int}} = \cup_{k=0}^{N-1} S_{\text{int,new}}^{(k)}. \tag{3.23}$$

Then we continue training with the updated training datasets S_r and S_{int} until the training is terminated and the final solution is obtained. A schematic diagram representing adaptive sampling is shown in Fig. 5. The detailed algorithm is summarized in Algorithm 2.

4 Numerical experiments

In this section, we conduct some numerical experiments to demonstrate the effectiveness of the proposed method. Specifically, we will approximate the Lorenz system, the Kuramoto-Sivashinsky equation, the Boussinesq-Burgers equations, and the nonlinear time-dependent Fokker-Planck equation. Throughout all benchmarks, we will employ the fully-connected neural network equipped with hyperbolic tangent activation functions (Tanh) and initialized using the Glorot normal scheme [12]. All neural networks

---

Algorithm 2 Adaptive sampling strategy for IR-PINNs

Input: Number of adaptive iteration N_{adaptive}, number of adaptive training epochs N_a, number of newly added points N_{new}, initial training datasets $S_r^{(0)}$, $S_{\text{int}}^{(0)}$, S_{ic} and S_{bc}, initial probability density model $p(t,\boldsymbol{x};\theta_f^{*,(0)})$.

Solve evolution equation via Algorithm 1 to obtain $u(t,\boldsymbol{x};\theta^{*,(0)})$.

for $k=0,\cdots,N_{\text{adaptive}}-1$ **do**
    // Train probability density model.
    for $j=1,\cdots,N_a$ **do**
        Generate samples from $p(t,\boldsymbol{x};\theta_f^{*,(k)})$.
        Update $p(t,\boldsymbol{x};\theta_f^{*,(k+1)})$ by descending the stochastic gradient of (3.17).
    end for
    // Refine training datasets.
    Generate new training samples $S_{r,\text{new}}$ through $p(t,\boldsymbol{x};\theta_f^{*,(k+1)})$.
    Update training datasets $S_r^{(k+1)}$ and $S_{\text{int}}^{(k+1)}$ based on (3.22) and (3.23).
    Solve evolution equation via Algorithm 1 to obtain $u(t,\boldsymbol{x};\theta^{*,(k+1)})$.
end for
Output: The predicted solution $u_N(t,\boldsymbol{x};\theta^*)$.

---

are trained via full-batch gradient descent using the Adam optimizer with default settings [15]. For the approximation of the integral in the regularization term, we employ Gaussian quadrature formula. Considering the highly chaotic behavior of the solutions in the first two examples, we utilize 64-bit double-precision floating-point numbers in our numerical simulations. All experiments are implemented by JAX [1].

In order to test the validity of the method, we use the following relative L_2 error:

$$err_{L_2} = \frac{\sqrt{\sum_{i=1}^{N_{\text{test}}}|u_N(t_i,\boldsymbol{x}_i;\theta)-u(t_i,\boldsymbol{x}_i)|^2}}{\sqrt{\sum_{i=1}^{N_{\text{test}}}|u(t_i,\boldsymbol{x}_i)|^2}}, \tag{4.1}$$

where N_{test} represents the total number of test points chosen randomly in the domain, and $u_N(t_i,\boldsymbol{x}_i;\theta)$ and $u(t_i,\boldsymbol{x}_i)$ represent the numerical and the reference solution values, respectively.

4.1 Lorenz system

We start with the chaotic Lorenz system, an ODE system. It is well known that this system exhibits strong sensitivity to its initial conditions, which can trigger divergent trajectories in finite time if the numerical predictions sought are not sufficiently accurate. The system

is described by the following ordinary differential equations

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y-x),$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = x(\rho-z)-y, \qquad (4.2)$$
$$\frac{\mathrm{d}z}{\mathrm{d}t} = xy-\beta z.$$

Here, we consider a classical setting with $\sigma=3$, $\rho=28$ and $\beta=8/3$. The temporal domain is $[0,20]$ and initial conditions are $[x(0),y(0),z(0)]=[1,1,1]$.

In this example, we employ the time-marching strategy and partition the temporal domain $[0,20]$ into 40 subdomains. The details of the time-marching strategy can be found in Appendix B. For each subdomain, we construct a 5-layer fully-connected neural network with 512 neurons per hidden layer. The initial learning rate is 10^{-3}, with an exponential decay rate of 0.9 applied every 2,000 training epochs. The maximum number of training epochs is 150,000. We divide the subdomain into $N=64$ subintervals and in each subinterval, the number of Gaussian quadrature points is $M=4$. Since Lorenz system is highly sensitive to the initial condition, we exactly impose the initial condition by

$$\tilde{x}_N(t;\theta) = tx_N(t;\theta)+x(0), \qquad (4.3)$$
$$\tilde{y}_N(t;\theta) = ty_N(t;\theta)+y(0), \qquad (4.4)$$
$$\tilde{z}_N(t;\theta) = tz_N(t;\theta)+z(0). \qquad (4.5)$$

For a fair comparison of our method with PINNs, we employ the same hyperparameter settings. Fig. 6 and Fig. 7 present the predicted trajectories in comparison with the reference trajectories, which are obtained using `scipy.integrate.odeint` with default settings. In Table 1, we provide the relative L_2 errors and running time, demonstrating that IR-PINNs1 and IR-PINNs2 achieve superior accuracy at an acceptable computational cost.

We also compare the performance and computational cost of the proposed method with causal PINNs. In [27], causal PINNs employ the same hyperparameter settings, time-marching strategy, and embedding of initial conditions. The only difference is that causal PINNs generate the same number of equidistant points instead of Gaussian quadrature points in the temporal direction. We can see that the proposed method shows better performance compared to causal PINNs in Table 2. Additionally, casual PINNs require a total of more than 6,000,000 training epochs, indicating that the running time is at least greater than that of PINNs, and similar to or even more than that of the proposed method.
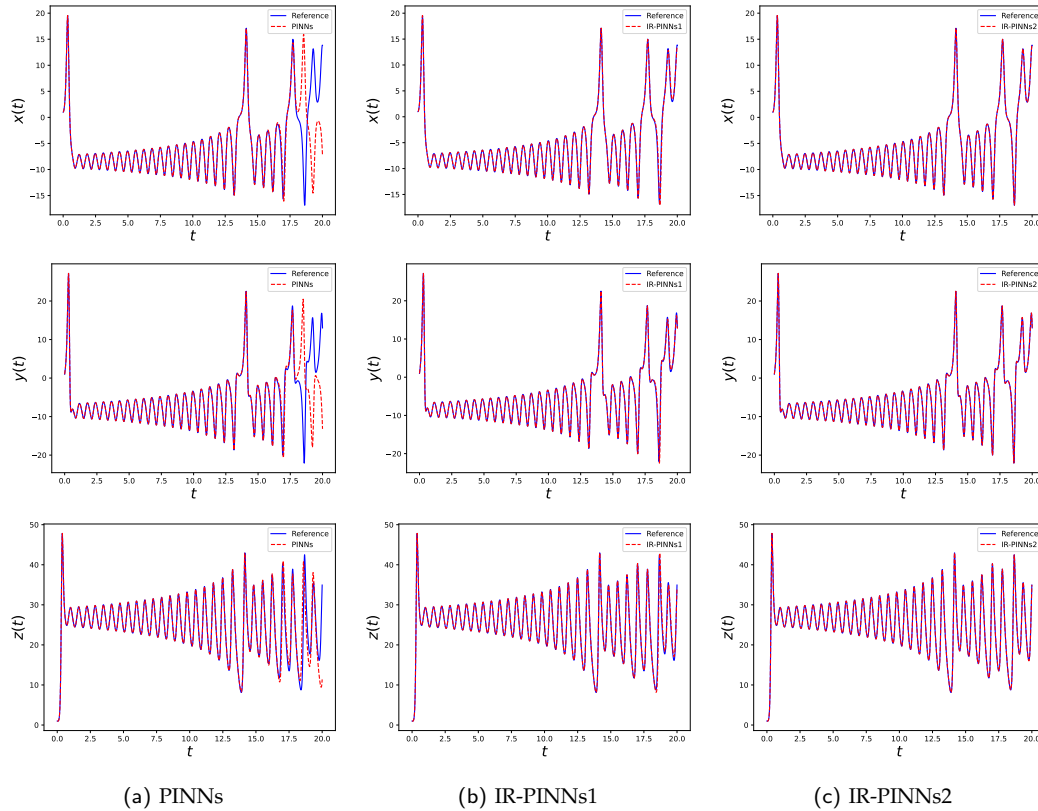
(a) PINNs   (b) IR-PINNs1   (c) IR-PINNs2

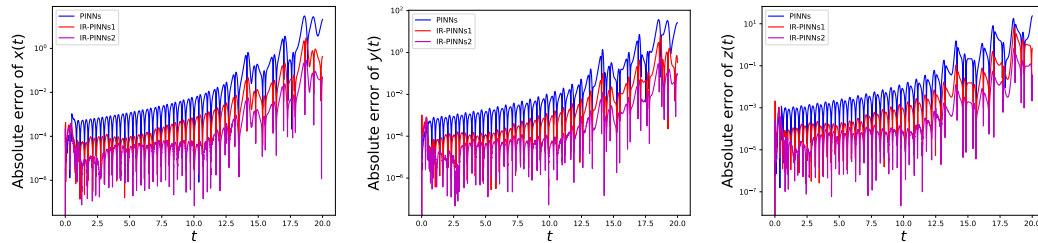Figure 6: *Lorenz system:* Comparison between the reference and numerical solutions.

Figure 7: *Lorenz system:* Absolute errors of $x(t)$, $y(t)$, and $z(t)$.

4.2 Kuramoto-Sivashinsky equation

The next example aims to demonstrate the effectiveness of the proposed method in tackling spatial-temporal chaotic systems. Consider one-dimensional Kuramoto-Sivashinsky equation, which has been independently derived in the context of reaction-diffusion systems and flame front propagation. The Kuramoto-Sivashinsky equation is a classic model describing spatial-temporal chaotic dynamical behavior. The equation is expressed as fol-

Table 1: *Lorenz system:* Relative L_2 errors and running time of different methods.

Relative L_2 error	PINNs	IR-PINNs1	IR-PINNs2
x	5.5445×10^{-1}	3.8330×10^{-2}	3.5086×10^{-3}
y	6.0381×10^{-1}	5.5684×10^{-2}	5.1053×10^{-3}
z	9.1354×10^{-2}	2.3450×10^{-2}	2.1651×10^{-3}
Running time (hours)	1.557	1.961	1.931

Table 2: *Lorenz system:* Relative L_2 errors between the proposed method and causal PINNs.

Relative L_2 error	IR-PINNs2	Causal PINNs
x	3.5086×10^{-3}	1.139×10^{-2}
y	5.1053×10^{-3}	1.656×10^{-2}
z	2.1651×10^{-3}	7.038×10^{-3}

lows:

$$u_t + \alpha u u_x + \beta u_{xx} + \gamma u_{xxxx} = 0, \tag{4.6}$$

subject to periodic boundary conditions and an initial condition

$$u(0,x) = u_0(x). \tag{4.7}$$

Here we set $\alpha = 100/16$, $\beta = 100/16^2, \gamma = 100/16^4$, for a fixed spatial domain in $[0, 2\pi]$. For validation, we use the Chebfun package [9] with a spectral Fourier discretization with 512 modes and a fourth-order stiff time-stepping scheme (ETDRK4) with time-step size 10^{-5}. Starting from the initial condition $u_0(x) = \cos(x)(1 + \sin(x))$, we select the numerical solution at $t = 0.4$ as our initial condition in the chaotic regime. Our goal is to learn the associated solution from initial time $t_0 = 0.4$ to terminal time $T = 0.9$. The reference solution is shown in Fig. 8.

In this example, we employ the time-marching strategy and partition the temporal domain $[0, 0.5]$ into 5 subdomains. For each subdomain, we construct a 5-layer fully-connected neural network with 512 neurons per hidden layer. The initial learning rate is 10^{-3}, with an exponential decay rate of 0.9 applied every 5,000 training epochs. The maximum number of training epochs is 600,000. We divide the subdomain into $N = 8$ subintervals and in each subinterval, the number of Gaussian quadrature points is $M = 4$. Spatially we choose 256 equidistant points for enforcing the PDE residual and 512 equidistant initial points. To further simplify the training objective, we also strictly impose the periodic boundary conditions by embedding the input coordinates into Fourier expansion (see Appendix C). The initial loss weight coefficient is set to $\lambda_2 = 10,000$.

The numerical results are summarized in Fig. 9, from which one can see that the numerical solutions of IR-PINNs1 and IR-PINNs2 exhibit good agreements with the reference solution. In Fig. 10, we can clearly observe that for PINNs, the relative L_2 error experiences a sudden and significant increase within the temporal subdomain $[0.1, 0.2]$,
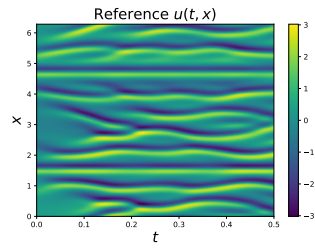
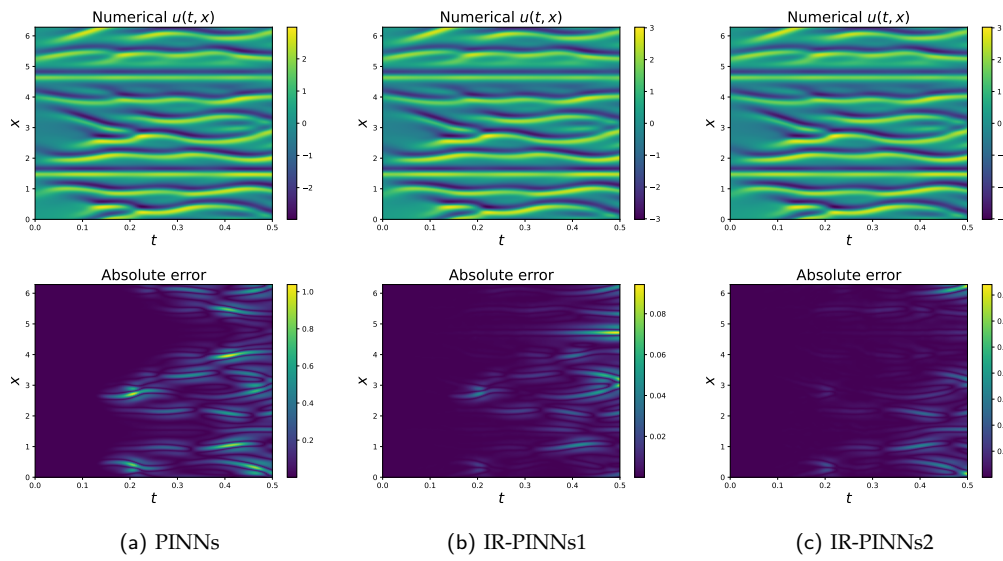Figure 8: *Kuramoto-Sivashinsky equation:* Reference solution.

(a) PINNs   (b) IR-PINNs1   (c) IR-PINNs2

Figure 9: *Kuramoto-Sivashinsky equation:* Numerical solutions and absolute errors.
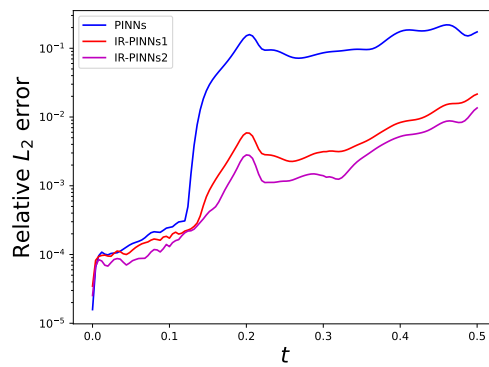

Figure 10: *Kuramoto-Sivashinsky equation:* Relative L_2 errors over time.

Table 3: *Kuramoto-Sivashinsky equation:* Relative L_2 errors and running time of different methods.

Relative L_2 error	PINNs	IR-PINNs1	IR-PINNs2
u	1.1278×10^{-1}	6.9356×10^{-3}	3.9945×10^{-3}
Running time (hours)	46.91	49.36	48.23

Table 4: *Kuramoto-Sivashinsky equation:* Relative L_2 errors between the proposed method and causal PINNs.

Relative L_2 error	IR-PINNs2	Causal PINNs
u	3.9945×10^{-3}	2.2600×10^{-2}

resulting in a substantial deviation between the initial values and the true values in subsequent temporal subdomains, ultimately leading to the failure of the simulation. In contrast, both IR-PINNs1 and IR-PINNs2 effectively control the rise of the error. Table 3 provides the relative L_2 errors and running time, demonstrating that our proposed method is equally effective in describing the PDEs of complex chaotic systems.

We also compare the performance and computational cost of the proposed method with causal PINNs. Similar to the previous example, under the premise of employing the same hyperparameter settings, time-marching strategy, and embedding of periodic boundary conditions, the proposed method demonstrates a more significant improvement in accuracy compared to causal PINNs in Table 4. Additionally, casual PINNs require a total of more than 3,000,000 training epochs, indicating that the running time is at least greater than that of PINNs, and similar to or even more than that of the proposed method.

4.3 Boussinesq-Burgers equations

In order to demonstrate the effectiveness of the adaptive sampling strategy, we consider the Boussinesq-Burgers equations, a coupled system of nonlinear partial differential equations that model the interaction of nonlinearity and dispersion in shallow water wave dynamics, with Dirichlet boundary conditions,

$$u_t + 2uu_x - \frac{1}{2}v_x = 0,$$
$$v_t + 2(uv)_x - \frac{1}{2}u_{xxx} = 0, \quad t \in [t_0, t_1], \ x \in [x_0, x_1]. \tag{4.8}$$

The one-soliton solution of the Boussinesq-Burgers equations is as follows

$$u(t,x) = -\frac{\exp(-x + \frac{7}{2}t - 7)}{2(1 + \exp(-x + \frac{7}{2}t - 7))} + 2,$$
$$v(t,x) = -\frac{\exp(-x + \frac{7}{2}t - 7)}{2(1 + \exp(-x + \frac{7}{2}t - 7))^2}, \tag{4.9}$$

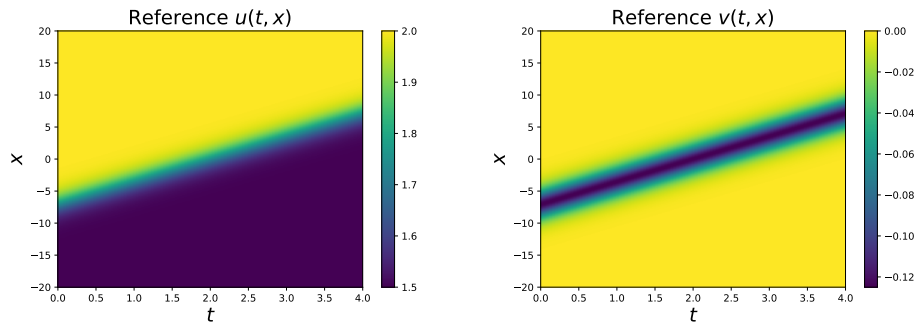Figure 11: *Boussinesq-Burgers equations:* Reference solutions of $u(t,x)$ and $v(t,x)$.

where the spatial-temporal domain is $[t_0,t_1] \times [x_0,x_1] = [0,4] \times [-20,20]$, as shown in Fig. 11. Initial and boundary conditions are obtained from the exact solution.

For the training process, we construct two 4-layer fully-connected neural networks with 64 neurons per hidden layer. The initial learning rate is 10^{-3}, with an exponential decay rate of 0.9 applied every 500 training epochs. The maximum number of training epochs is 20,000. We divide the subdomain into $N=8$ subintervals and in each subinterval, the number of Gaussian quadrature points is $M=4$. Before the first training, we select $8 \times 4 \times 16$ collocation points for enforcing the PDE residual, 256 equidistant initial points and 256 equidistant boundary points. The boundary and initial loss weight coefficients are both set to $\lambda_2 = \lambda_3 = 10$.

The number of adaptive iterations is 2. For each adaptive iteration, we take 20 bins for polynomial spline layer and 8 CDF coupling layers for bounded KRnet, with each layer parameterized by a 2-layer fully-connected neural network with 64 neurons. We randomly generate $8 \times 4 \times 100$ samples for evaluating PDE residual as our initial training set and the maximum number of adaptive training epochs for training $p(t,x;\theta_f)$ is 3,000. To refine the training set, we generate $8 \times 4 \times 8$ new samples from the density model $p(t,x;\theta_f)$.

In Fig. 12 and Fig. 13, the numerical solutions and absolute errors of different methods are presented. We observe that IR-PINNs1 and IR-PINNs2 are still capable of enhancing the accuracy of the solutions when all methods employ the adaptive sampling strategy, as evidenced by Fig. 14 and Table 5. However, it should be noted that if the evolution pattern of the solution in the temporal direction is relatively simple, the dominant source of error may stem from spatial discretization, such that the introduction of an integral-based regularization term results in modest improvement in accuracy. Additionally, the adaptive algorithm generates different spatial points at different times. Consequently, it is not feasible to discretize both residual terms simultaneously using the same set of points during the computation, which leads to an increase in the overall computation time.

In Fig. 15 and Fig. 16, we present the relative L_2 errors and newly generated points at different adaptive iterations. We can observe that the new training points are pri-

(a) PINNs   (b) IR-PINNs1   (c) IR-PINNs2
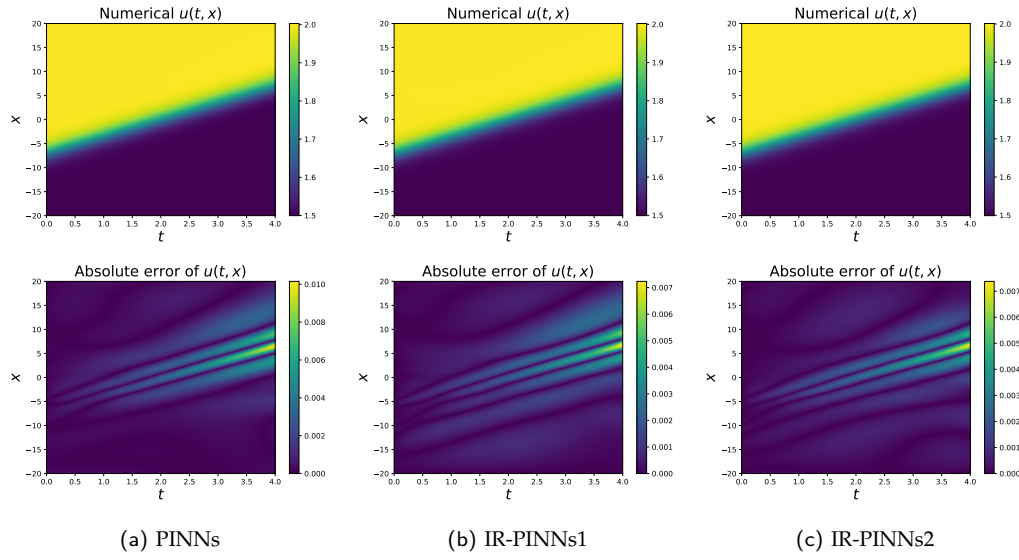
Figure 12: *Boussinesq-Burgers equations:* Numerical solutions and absolute errors of $u(t,x)$.

(a) PINNs   (b) IR-PINNs1   (c) IR-PINNs2
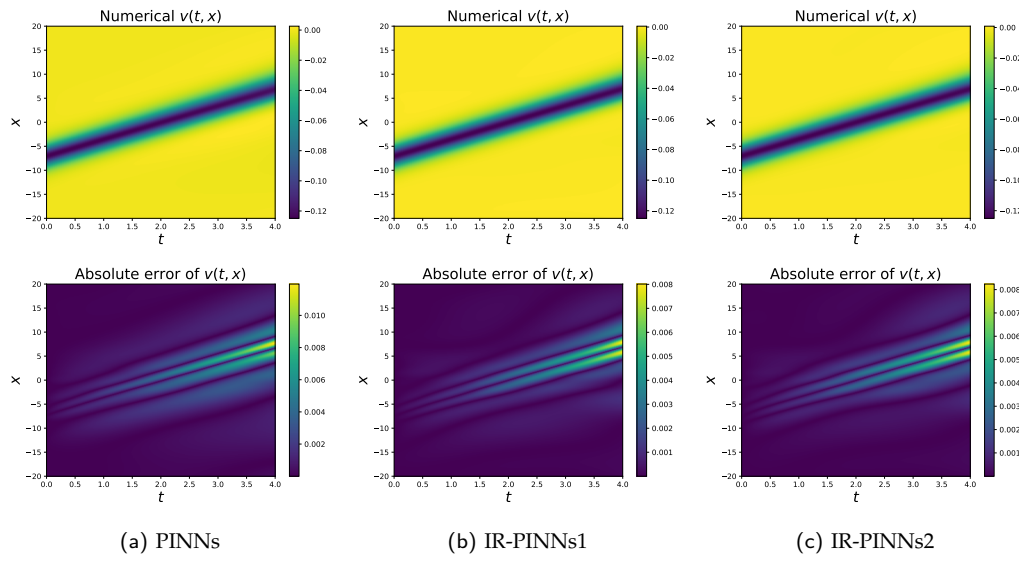
Figure 13: *Boussinesq-Burgers equations:* Numerical solutions and absolute errors of $v(t,x)$.

Table 5: *Boussinesq-Burgers equations:* Relative L_2 errors and running time of different methods.

Relative L_2 error	PINNs	IR-PINNs1	IR-PINNs2
u	8.1022×10^{-4}	5.9748×10^{-4}	5.3934×10^{-4}
v	5.0101×10^{-2}	3.0561×10^{-2}	3.2439×10^{-2}
Running time (hours)	0.3288	0.3693	0.4173
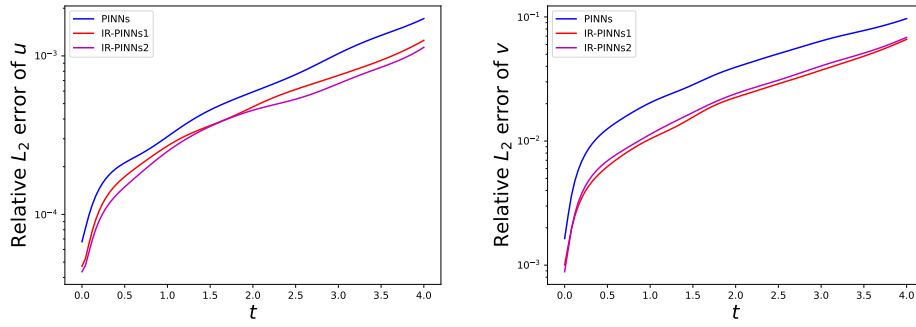
Figure 14: *Boussinesq-Burgers equations:* Relative L_2 errors over time.
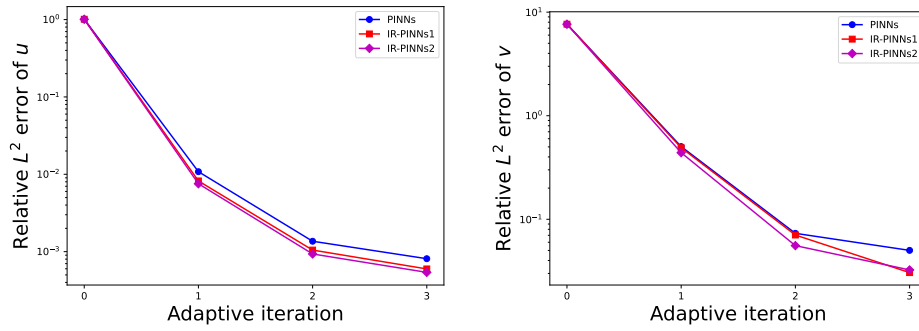
Figure 15: *Boussinesq-Burgers equations:* Relative L_2 errors at different adaptive iterations.

(a) PINNs   (b) IR-PINNs1   (c) IR-PINNs2
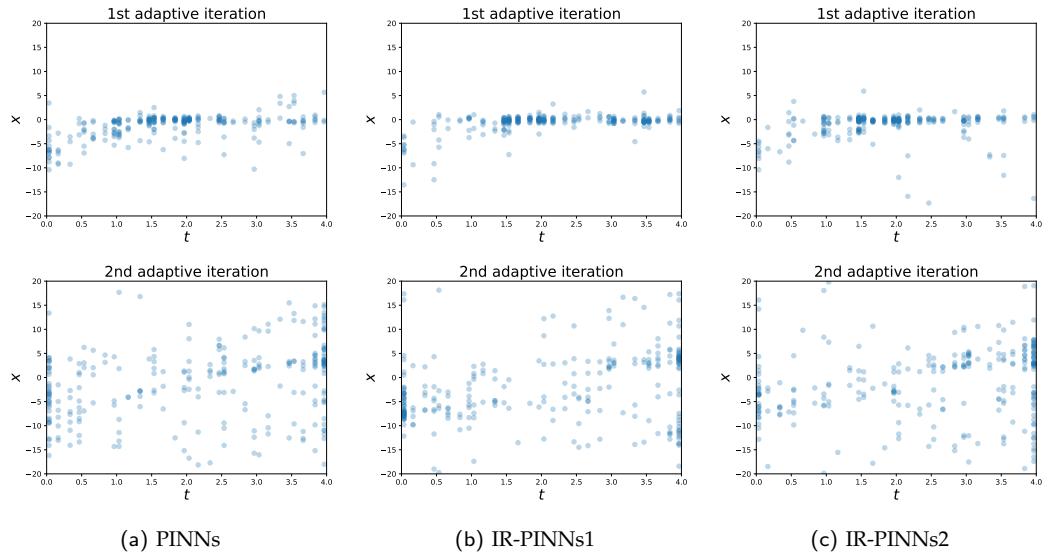
Figure 16: *Boussinesq-Burgers equations:* New samples generated by probability density model $p(t,\boldsymbol{x};\theta_f)$ of two adaptive iterations.

marily concentrated in the middle region where the gradient is larger. As the training progresses, the residuals become more evenly distributed, resulting in a more uniform spatial distribution of the points.

4.4   Time-dependent Fokker-Planck equation

In this example, we consider an unbounded domain problem, time-dependent Fokker-Planck equation, which is a fundamental model in statistical physics and describes the evolution of the probability density function of a stochastic process. We start from the following stochastic differential equation

$$\mathrm{d}\boldsymbol{X}_t = -\boldsymbol{\mu}(t,\boldsymbol{X}_t)\mathrm{d}t + \sigma(t,\boldsymbol{X}_t)\mathrm{d}W_t, \tag{4.10}$$

where $\boldsymbol{X}_t$ and $\boldsymbol{\mu}(t,\boldsymbol{X}_t)$ are d-dimensional random vectors, $\sigma(t,\boldsymbol{X}_t)$ is a $d \times M$ matrix and W_t is a M-dimensional standard Wiener process. The corresponding Fokker-Planck equation is given by

$$\frac{\partial p(t,\boldsymbol{x})}{\partial t} = -\nabla \cdot (\boldsymbol{\mu}(t,\boldsymbol{x})p(t,\boldsymbol{x})) + \frac{1}{2}\nabla \cdot \left(\nabla \cdot \left(\sigma(t,\boldsymbol{x})\sigma(t,\boldsymbol{x})^T p(t,\boldsymbol{x}) \right) \right), \tag{4.11}$$

where $p(t,\boldsymbol{x})$ is the probability density function of $\boldsymbol{X}_t$, $\boldsymbol{\mu}$ is the drift term and σ is the diffusion term. Generally, the solution of the time-dependent Fokker-Planck equation is defined over an unbounded domain with the following boundary condition:

$$p(t,\boldsymbol{x}) \to 0, \quad \text{as } \|\boldsymbol{x}\| \to \infty. \tag{4.12}$$

Moreover, since the solution is a probability density function, it must satisfy the non-negativity and normalization constraints:

$$p(t,\boldsymbol{x}) \geq 0, \quad \int_{\mathbb{R}^d} p(t,\boldsymbol{x})\mathrm{d}\boldsymbol{x} \equiv 1. \tag{4.13}$$

We consider a two-dimensional nonlinear Fokker-Planck equation with the following drift and diffusion terms

$$\boldsymbol{\mu} = (x_2, x_1 - 0.4x_2 - 0.1x_1^3), \quad \sigma = \mathrm{diag}(0,\sqrt{0.8}) \tag{4.14}$$

with the initial condition

$$p(0,\boldsymbol{x}) = \frac{1}{2\pi|\boldsymbol{\Sigma}_0|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}_0^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_0) \right), \tag{4.15}$$

where $\boldsymbol{\mu}_0 = (0,5)$ and $\boldsymbol{\Sigma}_0 = I_2$. The temporal domain is $[0,10]$ and we obtain the reference solution by the ADI scheme [23] in a truncated domain $[-10,10] \times [-10,10]$, with mesh size $\delta t = 0.005$ and $\delta h = 0.01$.

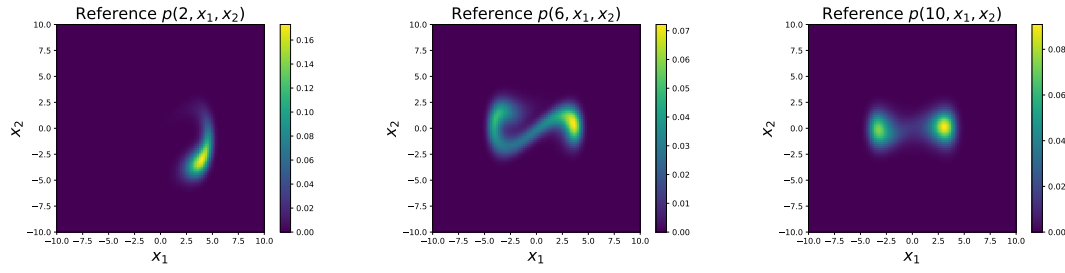Figure 17: *Time-dependent Fokker-Planck equation:* Reference solutions at $t=2,6,10$.

Here we directly approximate the solution of time-dependent Fokker-Planck equation by temporal normalizing flow [11], a probability density model, which satisfies the properties in (4.12) and (4.13) automatically. Since it is a long-time simulation, we employ the time-marching strategy and partition the temporal domain $[0,10]$ into 10 subdomains. For each subdomain, we use the actnorm layer and 8 affine coupling layers with each layer parameterized by a 2-layer fully-connected neural network with 64 neurons. The initial learning rate is 10^{-3}, with an exponential decay rate of 0.9 applied every 500 training epochs. The maximum number of training epochs is 20,000. We divide the subdomain into $N=10$ subintervals and in each subinterval, the number of Gaussian quadrature points is $M=4$. The initial loss weight coefficient is set to $\lambda_2=100$.

To further improve the accuracy of the solution, we first generate 1,000 sample paths by the Euler-Maruyama method [16] with $\delta t = 0.01$ and pre-train the neural network model for 2,000 epochs with the data to obtain a good initialization. The details of the pre-training approach are provided in the Appendix D. Subsequently, we generate 100 spatial points for each temporal collocation point by the pre-trained model and 1,000 initial points from the initial condition. Then we proceed with adaptive training. The number of adaptive iterations is 2 for each subdomain and for each adaptive iteration, we newly generate $10\times4\times100$ adaptive sample points. Note that temporal normalizing flow is inherently a probability density model, so we can directly perform adaptive sampling using this model.

We present the reference solutions, numerical solutions and absolute errors of different methods at $t=2,6,10$ in Figs. 17-20. The results demonstrate that our method can be directly combined with an adaptive sampling strategy for solving time-dependent Fokker-Planck equation using temporal normalized flows. Similar to the previous example, due to the difficulties caused by the evolving physical localization, the improvements from temporal discretization are limited, as shown in Fig. 21 and Table 6 in terms of the relative L_2 error and relative KL divergence. In Fig. 22, we present the adaptively generated samples of the last adaptive iteration at $t=2,6,10$.
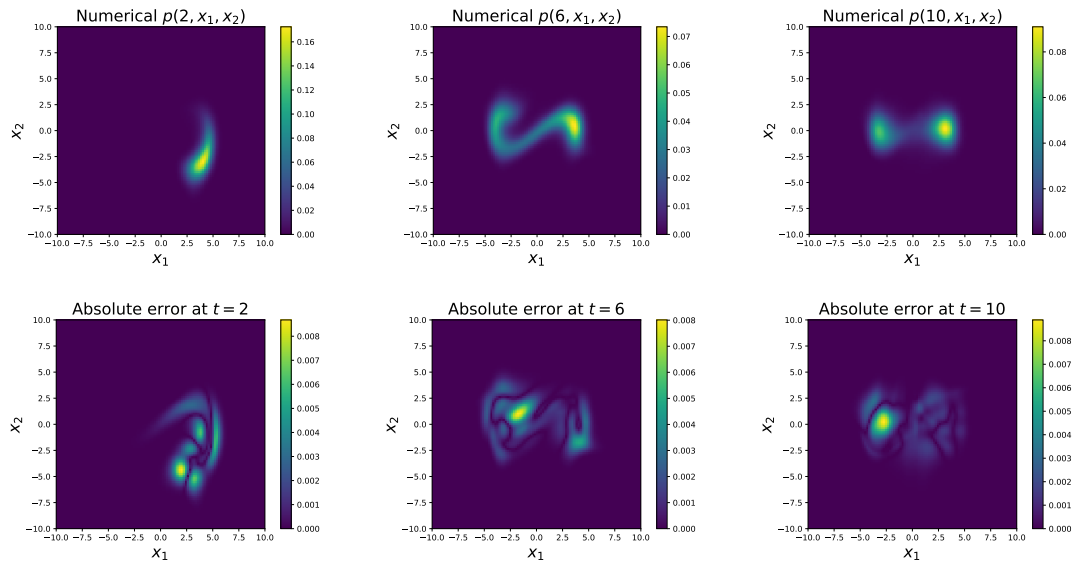
Figure 18: *Time-dependent Fokker-Planck equation:* Numerical solutions and absolute errors of PINNs at $t=$ 2,6,10.
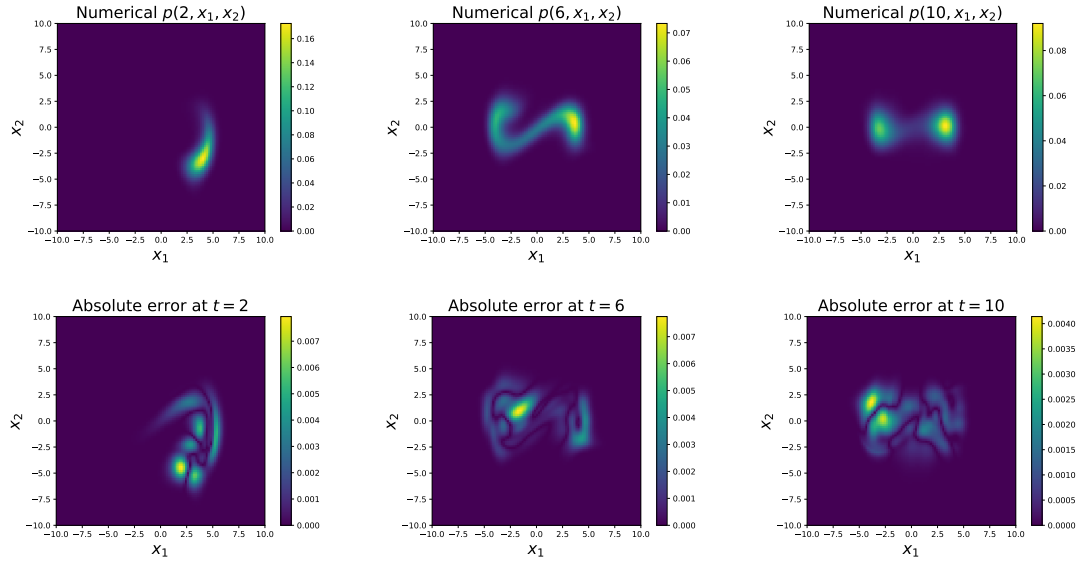

Figure 19: *Time-dependent Fokker-Planck equation:* Numerical solutions and absolute errors of IR-PINNs1 at $t=$2,6,10.
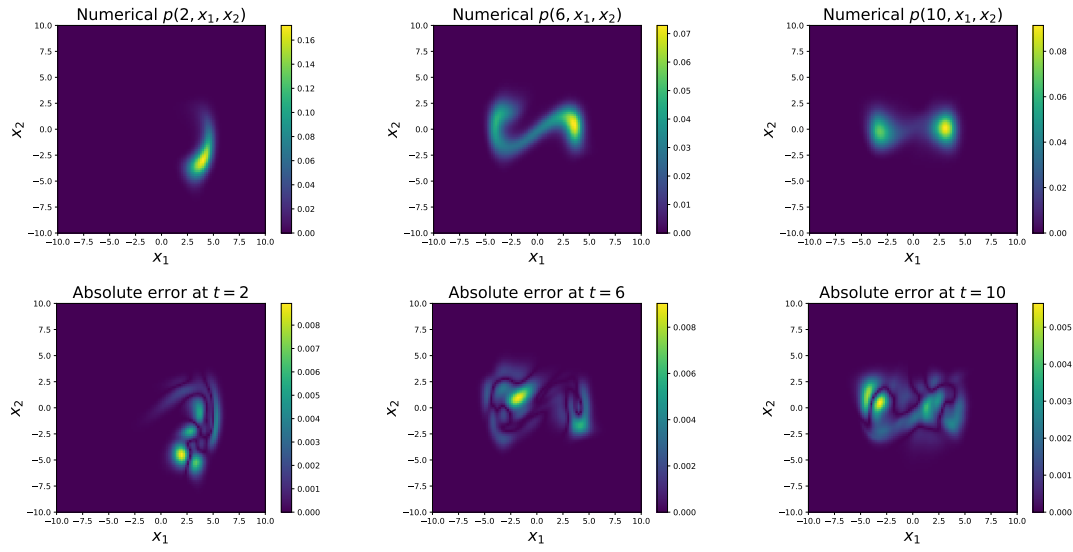
Figure 20: *Time-dependent Fokker-Planck equation:* Numerical solutions and absolute errors of IR-PINNs2 at $t=2,6,10$.
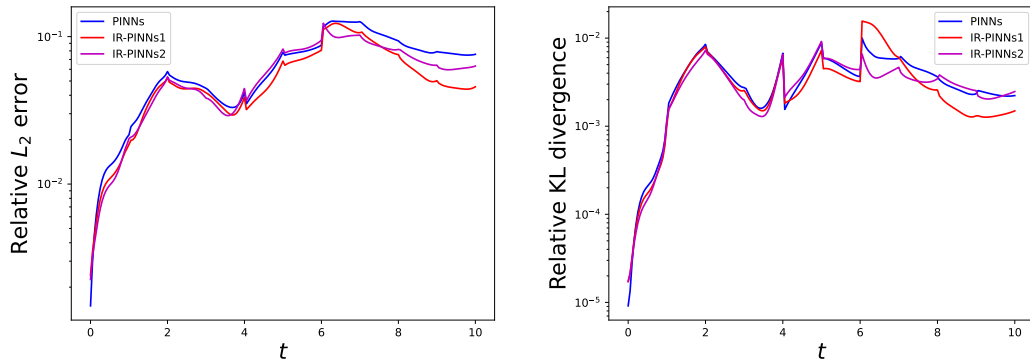

Figure 21: *Time-dependent Fokker-Planck equation:* Relative L_2 errors and relative KL divergences over time.

Table 6: *Time-dependent Fokker-Planck equation:* Relative L_2 errors and running time of different methods.

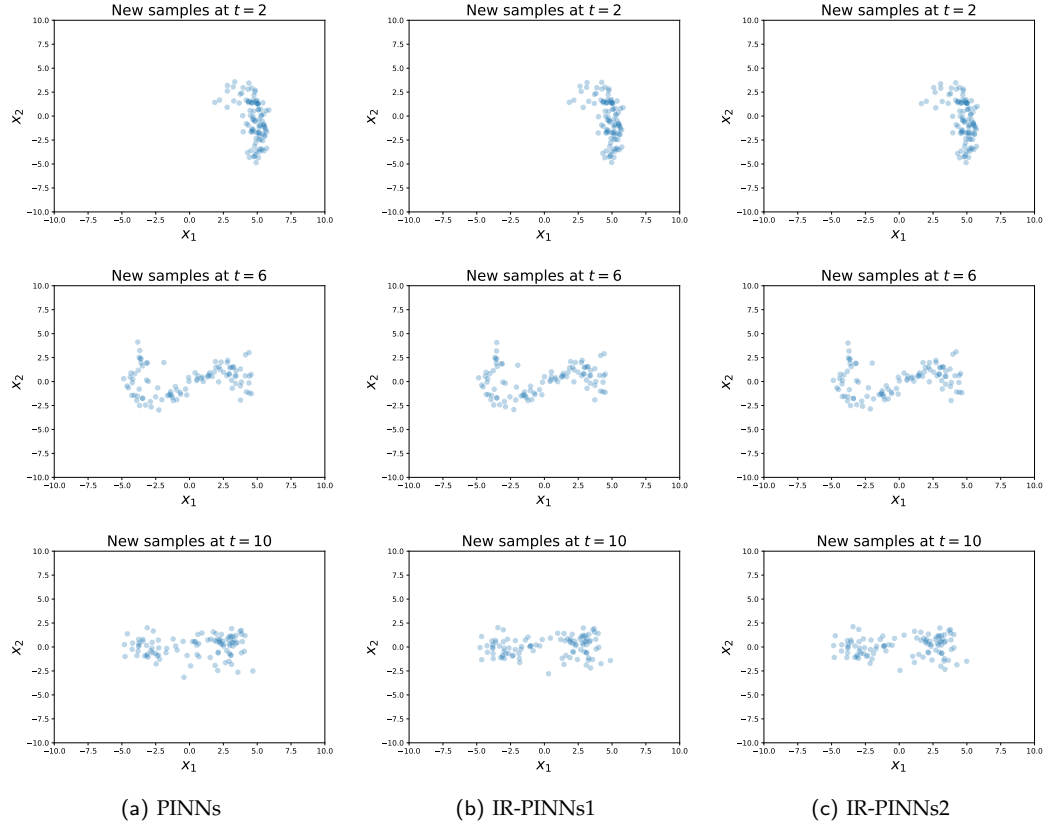|	PINNs	IR-PINNs1	IR-PINNs2
Relative L_2 error	6.3784×10^{-2}	5.3724×10^{-2}	5.6185×10^{-2}
Relative KL divergence	2.0582×10^{-3}	2.0117×10^{-3}	1.8202×10^{-3}
Running time (hours)	9.612	21.06	24.47

Figure 22: *Time-dependent Fokker-Planck equation:* New samples generated by probability density model $p(t,x;\theta_f)$ of the last adaptive iteration at $t=2,6,10$.

5 Conclusion

In this paper, we have proposed integral regularization physics-informed neural networks (IR-PINNs) to address the challenges of solving evolution equations, particularly in capturing long-time dynamics and reducing temporal error accumulation. By reformulating the equations into an integral form and incorporating an integral-based residual term, IR-PINNs enhance temporal accuracy and improve the resolution of temporal correlations. The addition of an adaptive sampling strategy further ensures higher accuracy in regions with sharp gradients or rapid variations. Numerical experiments on benchmark problems have demonstrated the effectiveness of IR-PINNs. Future research directions include investigating the underlying mechanisms of how the integral-based regularization term improves accuracy and determining its applicability to a broader range of equations. Additionally, exploring more effective adaptive sampling strategies could further enhance computational efficiency and solution precision. In summary, IR-

PINNs offer a robust and accurate framework for solving evolution equations, advancing the field of physics-informed machine learning and its applications to dynamic systems.

Acknowledgments

A Normalizing flow

For the polynomial spline layer, without loss of generality, we assume that $t \in [0,1]$. We aim to find an invertible transformation $z = f_{\text{poly}}(t)$ such that

$$p_{\text{poly}}(t) = p_Z(f_{\text{poly}}(t))|\det\nabla_t f_{\text{poly}}(t)|, \quad Z \sim \mathcal{U}[0,1], \tag{A.1}$$

where $\mathcal{U}$ denotes the uniform distribution. To parameterize f_{poly}, we let $0 = l_0 < l_1 < \cdots < l_{m-1} < l_m = 1$ be a given partition of the unit interval and $\{k_j\}_{j=0}^m$ be the corresponding weights satisfying $\sum_j k_j = 1$. A piecewise linear polynomial can be defined as follows:

$$p_{\text{poly}}(t) = \frac{k_{j+1}-k_j}{l_{j+1}-l_j}(t-l_j) + k_j, \quad \forall t \in [l_j, l_{j+1}], \tag{A.2}$$

and the corresponding cumulative probability function f_{poly} is

$$f_{\text{poly}}(t) = \frac{k_{j+1}-k_j}{2(l_{j+1}-l_j)}(t-l_j)^2 + k_j(t-l_j) + \sum_{i=0}^{j-1}\frac{k_i+k_{i+1}}{2}(l_{i+1}-l_i), \quad \forall t \in [l_j, l_{j+1}]. \tag{A.3}$$

To satisfy $\int_0^1 p(t)\mathrm{d}t = 1$, we can model $\{k_j\}_{j=0}^m$ as

$$k_j = \frac{\exp(\tilde{k}_j)}{C}, \quad \forall j = 0, \cdots, m, \tag{A.4}$$

where $\theta_{f,1} = \{\tilde{k}_j\}_{j=0}^m$ are trainable parameters of $f_{\text{poly}}(t;\theta_{f,1})$ and C is a normalization constant:

$$C = \sum_{i=0}^{m-1}\frac{(\exp(\tilde{k}_i)+\exp(\tilde{k}_{i+1}))(l_{i+1}-l_i)}{2}. \tag{A.5}$$

For the bounded KRnet, given $\boldsymbol{x} \in \mathbb{R}^d$, we seek an invertible transformation $\boldsymbol{y} = f_{\text{B-KRnet}}(t,\boldsymbol{x}) \in \mathbb{R}^d$ for any given t such that

$$p_{\text{B-KRnet}}(\boldsymbol{x}|t) = p_{\boldsymbol{Y}}(\boldsymbol{y})|\det\nabla_{\boldsymbol{x}} f_{\text{B-KRnet}}(t,\boldsymbol{x})|, \quad \boldsymbol{Y} \sim \mathcal{U}[-1,1]^d, \quad \forall s. \tag{A.6}$$

Here we employ a conditional bounded KR-net to parameterize $f_{\mathrm{B-KRnet}}(t,\boldsymbol{x})$. The basic idea is to define the structure of $f_{\mathrm{B-KRnet}}(t,\boldsymbol{x})$ in terms of the Knothe-Rosenblatt rearrangement [3], which has a block low-triangular structure

$$\boldsymbol{y} = f_{\mathrm{B-KRnet}}(t,\boldsymbol{x};\theta_{f,2}) = \begin{bmatrix} f_1(t,x_1;\theta_{f_1,2}) \\ f_2(t,x_1,x_2;\theta_{f_2,2}) \\ \vdots \\ f_m(t,x_1,\cdots,x_m;\theta_{f_m,2}) \end{bmatrix}, \tag{A.7}$$

where $\theta_{f,2} = \{\theta_{f_1,2},\theta_{f_2,2},\cdots,\theta_{f_m,2}\}$ includes all the model parameters and $[x_1,x_2,\cdots,x_m]$ corresponds to a partition of $\boldsymbol{x}$. The sub-transformations $f_1,\cdots,f_m$ consist of affine linear layers, squeezing layers and CDF coupling layers and more details can be found in [33].

B   Time-marching strategy

Following the methodology proposed in [31], we implement a time-marching strategy to enhance the convergence of IR-PINNs when applied to long-time integration problems. Specifically, we partition the temporal domain $[0,T]$ into subdomains

$$[0,\Delta t],[\Delta t,2\Delta t],\cdots,[(N-1)\Delta t,N\Delta t], \quad \Delta t = T/N. \tag{B.1}$$

We build a single neural network for each subdomain, training consecutively by using the final prediction of the current subdomain as the initial condition for the next, until the entire temporal domain is fully trained.

C   Exact periodic boundary conditions

Following the work from [7,27], we can enforce exact C^{∞} periodic boundary conditions by constructing a Fourier feature embedding of the form

$$v(x) = \{1,\cos(\omega x),\sin(\omega x),\cdots,\cos(M\omega x),\sin(M\omega x)\} \tag{C.1}$$

as the spatial input to the neural network, where $\omega = \frac{2\pi}{L}, L = x_{\max} - x_{\min}$, and M is a nonnegative integer representing the frequency of the input. In this work, we take $M = 5$.

D   Pre-training approach for the time-dependent Fokker-Planck equation

Assuming we can obtain some sample paths directly from the SDE (4.10) using Euler-Maruyama method [16], the accuracy of the numerical solution will be further improved. Specifically, for a given set of N_t temporal points $\{t_j\}_{j=1}^{N_t}$, we generate N_p paths $\{X_t^{(i)}\}_{i=1}^{N_p}$:

$$X_t^{(i)} = \left((t_1,\boldsymbol{x}_1^{(i)}),(t_2,\boldsymbol{x}_2^{(i)}),\cdots,(t_{N_t},\boldsymbol{x}_{N_t}^{(i)})\right), \quad i = 1,2,\cdots,N_p. \tag{D.1}$$

Subsequently, the neural network can be pre-trained by minimizing the negative log-likelihood loss [19], formulated as:

$$\mathcal{L}_{\text{data}} = -\frac{1}{N_p N_t} \sum_{i=1}^{N_p} \sum_{j=1}^{N_t} \log p(t_j, \boldsymbol{x}_j^{(i)}). \tag{D.2}$$

References

[1] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. JAX: Composable transformations of Python+ Numpy programs. 2018.

[2] Oscar P Bruno, Jan S Hesthaven, and Daniel V Leibovici. FC-based shock-dynamics solver with neural-network localized artificial-viscosity assignment. *Journal of Computational Physics: X*, 15:100110, 2022.

[3] Guillaume Carlier, Alfred Galichon, and Filippo Santambrogio. From Knothe's transport to Brenier's map and a continuation method for optimal transport. *SIAM Journal on Mathematical Analysis*, 41(6):2554–2576, 2010.

[4] Pao-Hsiung Chiu, Jian Cheng Wong, Chinchun Ooi, My Ha Dao, and Yew-Soon Ong. CAN-PINN: A fast physics-informed neural network based on coupled-automatic-numerical differentiation method. *Computer Methods in Applied Mechanics and Engineering*, 395:114909, 2022.

[5] Richard Courant and David Hilbert. *Methods of Mathematical Physics: Partial Differential Equations*. John Wiley & Sons, 2008.

[6] Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. Rethinking the importance of sampling in physics-informed neural networks. *arXiv preprint arXiv:2207.02338*, 2022.

[7] Suchuan Dong and Naxian Ni. A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. *Journal of Computational Physics*, 435:110242, 2021.

[8] Gideon Dresdner, Dmitrii Kochkov, Peter Norgaard, Leonardo Zepeda-Núñez, Jamie A Smith, Michael P Brenner, and Stephan Hoyer. Learning to correct spectral methods for simulating turbulent flows. *arXiv preprint arXiv:2207.00556*, 2022.

[9] Tobin A Driscoll, Nicholas Hale, and Lloyd N Trefethen. Chebfun Guide, 2014.

[10] Xiaodong Feng, Haojiong Shangguan, Tao Tang, Xiaoliang Wan, and Tao Zhou. A hybrid FEM-PINN method for time-dependent partial differential equations. *arXiv preprint arXiv:2409.02810*, 2024.

[11] Xiaodong Feng, Li Zeng, and Tao Zhou. Solving time dependent Fokker-Planck equations via temporal normalizing flow. *Communications in Computational Physics*, 32(2):401–423, 2022.

[12] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[13] Yiqi Gu and Michael K Ng. Deep adaptive basis Galerkin method for high-dimensional evolution equations with oscillatory solutions. *SIAM Journal on Scientific Computing*, 44(5):A3130–A3157, 2022.

[14] Jeahan Jung, Heechang Kim, Hyomin Shin, and Minseok Choi. CEENs: Causality-enforced evolutional networks for solving time-dependent partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 427:117036, 2024.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Peter E Kloeden, Eckhard Platen, Peter E Kloeden, and Eckhard Platen. *Stochastic Differential Equations*. Springer, 1992.

[17] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.

[18] Phillip Lippe, Bas Veeling, Paris Perdikaris, Richard Turner, and Johannes Brandstetter. PDE-Refiner: Achieving accurate long rollouts with neural PDE solvers. *Advances in Neural Information Processing Systems*, 36:67398–67433, 2023.

[19] Yubin Lu, Romit Maulik, Ting Gao, Felix Dietrich, Ioannis G Kevrekidis, and Jinqiao Duan. Learning the temporal evolution of multivariate densities via normalizing flows. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 32(3), 2022.

[20] Revanth Mattey and Susanta 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*, 390:114474, 2022.

[21] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Transactions on Graphics (ToG)*, 38(5):1–19, 2019.

[22] Michael Penwarden, Ameya D Jagtap, Shandian Zhe, George Em Karniadakis, and Robert M Kirby. A unified scalable framework for causal sweeping strategies for physics-informed neural networks (PINNs) and their temporal decompositions. *Journal of Computational Physics*, 493:112464, 2023.

[23] Lukas Pichler, Arif Masud, and Lawrence A Bergman. Numerical solution of the Fokker–Planck equation by finite difference and finite element methods—a comparative study. *Computational Methods in Stochastic Dynamics: Volume 2*, pages 69–85, 2013.

[24] Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.

[25] Pratik Rathore, Weimu Lei, Zachary Frangella, Lu Lu, and Madeleine Udell. Challenges in training PINNs: A loss landscape perspective. *arXiv preprint arXiv:2402.01868*, 2024.

[26] Sifan Wang and Paris Perdikaris. Long-time integration of parametric evolution equations with physics-informed deeponets. *Journal of Computational Physics*, 475:111855, 2023.

[27] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality for training physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 421:116813, 2024.

[28] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[29] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science Advances*, 7(40):eabi8605, 2021.

[30] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[31] Colby L Wight and Jia Zhao. Solving Allen-Cahn and Cahn-Hilliard equations using

the adaptive physics informed neural networks. *Communications in Computational Physics*, 29(3):930–954, 2021.

[32] Zhikai Wu, Shiyang Zhang, Sizhuang He, Sifan Wang, Min Zhu, Anran Jiao, Lu Lu, and David van Dijk. Coast: Intelligent time-adaptive neural operators. *arXiv preprint arXiv:2502.08574*, 2025.

[33] Li Zeng, Xiaoliang Wan, and Tao Zhou. Bounded KRnet and its applications to density estimation and approximation. *arXiv preprint arXiv:2305.09063*, 2023.