IOP Publishing Mach. Learn.: Sci. Technol. 4 (2023) 025024 https://doi.org/10.1088/2632-2153/acd0a1

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
2 November 2022

REVISED
17 April 2023

ACCEPTED FOR PUBLICATION
26 April 2023

PUBLISHED
26 May 2023

PAPER

Spectrally adapted physics-informed neural networks for solving unbounded domain problems

Mingtao Xia[1], **Lucas Böttcher**[2] and **Tom Chou**[1,*]

[1] Department of Mathematics, UCLA, Los Angeles, CA 90095-1555, United States of America
[2] Department of Computational Science and Philosophy, Frankfurt School of Finance and Management, Frankfurt am Main 60322, Germany
* Author to whom any correspondence should be addressed.

E-mail: tomchou@ucla.edu

Keywords: physics-informed neural networks, spectral methods, adaptive methods, PDE models, unbounded domains

Abstract

Solving analytically intractable partial differential equations (PDEs) that involve at least one variable defined on an unbounded domain arises in numerous physical applications. Accurately solving unbounded domain PDEs requires efficient numerical methods that can resolve the dependence of the PDE on the unbounded variable over at least several orders of magnitude. We propose a solution to such problems by combining two classes of numerical methods: (i) adaptive spectral methods and (ii) physics-informed neural networks (PINNs). The numerical approach that we develop takes advantage of the ability of PINNs to easily implement high-order numerical schemes to efficiently solve PDEs and extrapolate numerical solutions at any point in space and time. We then show how recently introduced adaptive techniques for spectral methods can be integrated into PINN-based PDE solvers to obtain numerical solutions of unbounded domain problems that cannot be efficiently approximated by standard PINNs. Through a number of examples, we demonstrate the advantages of the proposed spectrally adapted PINNs in solving PDEs and estimating model parameters from noisy observations in unbounded domains.

1. Introduction

The use of neural networks as universal function approximators [1, 2] led to various applications in simulating [3, 4] and controlling [5–8] physical, biological, and engineering systems. Training neural networks in function-approximation tasks is typically realized in two steps. In the first step, an observable u_s associated with each distinct sample or measurement point $(x,t)_s \equiv (x_s,t_s)$, $s=1,2,\ldots,n$ is used to construct the corresponding loss function (e.g. the mean squared loss) in order to find representations for the constraint $u_s \equiv u(x_s,t_s)$ or infer the equation that the function $u(x,t)$ obeys. In many physical settings, the variables x and t denote the space and time variables, respectively. Thus, the data points $(x,t)_s$ in many cases can be classified in two groups, $\{x_s\}$ and $\{t_s\}$, and the information they contain may be manifested differently in an optimization process. In the second step, the loss function is minimized by backpropagating gradients to adjust neural network parameters Θ. If the number of observations n is limited, additional constraints may help to make the training process more effective [9].

To learn and represent the dynamics of physical systems, the constraints used in physics-informed neural networks (PINNs) [3, 4] provide one possible option of an inductive bias in the training process. The key idea underlying PINN-based training is that the constraints imposed by the known equations of motion for some parts of the system are embedded in the loss function. Terms in the loss function associated with the differential equation can be evaluated using a neural network, which could be trained via backpropagation and automatic differentiation. In accordance with the distinction between Lagrangian and Hamiltonian formulations of the equations of motion in classical mechanics, PINNs can be also divided into these two categories [10–12]. Another formulation of PINNs uses variational principles [13] in the loss function to further constrain the types of functions used. Such variational PINNs rely on finite element (FE) methods to discretize partial differential equation (PDE)-type constraints.

Many other PINN-based numerical algorithms have been recently proposed. A space-time domain decomposition PINN method was proposed for solving nonlinear PDEs [14]. In other variants, physics-informed Fourier neural operators have also been proposed to learn the underlying PDE models [15]. In general, PINNs link modern neural network methods with traditional complex physical models and allow algorithms to efficiently use higher-order numerical schemes to (i) solve complex physical problems with high accuracy, (ii) infer model parameters, and (iii) reconstruct physical models in data-driven inverse problems [3]. Therefore, PINNs have become increasingly popular as they can avoid certain computational difficulties encountered when using traditional FE/FD methods to find solutions to physics models.

The broad utility of PINNs is revealed by their numerous applications to problems in aerodynamics [16], surface physics [17], power systems [18], cardiology [19], and soft biological tissues [20]. PINNs have also been integrated into the multi-task learning [21] and meta-learning [22] frameworks. When implementing PINN algorithms to find functions in an unbounded system, the unbounded variables cannot be simply normalized, precluding reconstruction of solutions outside the range of data. Nonetheless, many problems in nature are associated with long-ranged potentials [23, 24] (i.e. unbounded spatial domains) and processes that are subject to algebraic damping [25] (i.e. unbounded temporal domains), and thus need to be solved in unbounded domains. For example, to capture the oscillatory and decaying behavior at infinity of the solution to Schrödinger's equation, efficient numerical methods are required in the unbounded domain $\mathbb{R}$ [26]. As another example, in structured cellular proliferation models in mathematical biology, efficient unbounded domain numerical methods are required to detect and better resolve possible blow-up in mean cell size [27, 28]. Finally, in solid-state physics, long-range interactions [29, 30] require algorithms tailored for unbounded domain problems to accurately simulate particle interactions over long distances.

Solving unbounded domain problems is thus a key challenge in various fields that cannot be addressed with standard PINN-based solvers. In static problems, if the solution's behavior at infinity is known, one can use boundary-layer methods to truncate the unbounded domain by discretizing space [31]. However, in spatiotemporal problems it is often the case that the solution's behavior is evolving over time or otherwise unknown. Solving a PDE in this situation requires proper detection and capturing of the function's long-range behavior over time. Thus, simply discretizing space or truncating the domain is usually not effective in spatiotemporal problems. To efficiently solve PDEs in unbounded domains, we will treat the information carried by the x_s data using spectral decompositions of the function $u(x,t)$ in the x variable. Typically, a spatial initial condition of the desired solution is given and some spatial regularity is assumed from the underlying physical process. As a consequence, we suppose that at time t, we can use a spectral expansion in x to record spatial information. On the other hand, a solution's behavior in time t is unknown and one still has to numerically step forward in time to obtain the solution. Thus, we combine PINNs with spectral methods and propose a spectrally adapted PINN (s-PINN) method that can utilize recently developed adaptive function expansions techniques [32, 33].

In contrast to traditional numerical spectral schemes that can only furnish solutions at discrete, predetermined timesteps, our approach uses time t as an input variable into the neural network combined with the PINN method to define a loss function, which enables (i) easy implementation of high-order Runge-Kutta schemes to relax the constraint on timesteps and (ii) easy extrapolation of the numerical solution at any time. However, our approach is distinct from that taken in standard PINNs, variational-PINNs, or physics-informed neural operator approaches. We do not input spatial positions x into the network or try to learn the x-dependence of $u(x,t)$; instead, we assume that the function $u(x,t)$ can be approximated by a spectral expansion in x with appropriate basis functions. Rather than learning the explicit spatial dependence directly, we train the neural network to learn the time-dependent expansion coefficients. Our main contributions include (i) integrating spectral methods into multi-output neural networks to approximate the spectral expansions of functions when partial information is available, (ii) incorporating recently developed adaptive spectral methods in our s-PINNs to allow accurate solutions of unbounded-domain spatiotemporal PDEs, and (iii) presenting explicit examples illustrating how s-PINNs can be used to solve unbounded domain problems, recover spectral convergence, and more easily solve inverse-type PDE inference problems. We show how s-PINNs provide a unified, easy-to-implement method for solving PDEs and performing parameter-inference given noisy observation data and how complementary adaptive spectral techniques can further improve efficiency, especially for solving problems in unbounded domains.

In section 2, we show how neural networks can be combined with modern adaptive spectral methods to outperform standard neural networks in function approximation tasks. As a first application, we show in section 3 how efficient PDE solvers can be derived from spectral PINN methods. In section 4, we discuss another application that focuses on reconstructing underlying physical models and inferring model parameters given observational data. In section 5, we summarize our work and discuss possible directions for

Table 1. Overview of variables. Definitions of the main variables and parameters used in this paper.

Symbol	Definition
n	Number of observations
N	Spectral expansion order
N_H	Number of intermediate layers in the neural network
H	Number of neurons per layer
η	Learning rate of stochastic gradient descent
Θ	Neural network parameters (weights and biases)
K	Order of the Runge–Kutta scheme
$\mathcal{L}$	Loss function, *e.g.*, sum of squared errors (SSEs)
β	Scaling factor in basis functions $\phi^{\beta}_{i,x_L}(x) := \phi_i(\beta(x-x_L))$
x_L	Translation of basis functions $\phi^{\beta}_{i,x_L} := \phi_i(\beta(x-x_L))$
u^{β}_{N,x_L}	Spectral expansion of order N generated by the neural network: $u^{\beta}_{N,x_L} = \sum_{i=0}^{N} w^{\beta}_{i,x_L}\phi_i(\beta(x-x_L))$
$\mathcal{F}(u^{\beta}_{N,x_L})$	Frequency indicator for the spectral expansion u^{β}_{N,x_L}
$\hat{\mathcal{H}}^{\beta}_{i,x_L}$	Generalized Hermite function of order i, scaling factor β, and translation x_L
P^{β}_{N,x_L}	Function space defined by the first $N+1$ generalized Hermite functions $P^{\beta}_{N,x_L} := \{\hat{\mathcal{H}}^{\beta}_{i,x_L}\}_{i=0}^{N}$
q	Scaling factor (β) adjustment ratio
ν	Threshold for adjusting the scaling factor β
ρ, ρ_0	Threshold for increasing, decreasing N
γ	Ratio for adjusting ρ

future research. A summary of the main variables and parameters used in this study is given in table 1. Our source codes are publicly available at https://gitlab.com/ComputationalScience/spectrally-adapted-pinns.

2. Combining spectral methods with neural networks

In this section, we first introduce the basic features of function approximators that rely on neural networks and spectral methods designed to handle variables that are defined in unbounded domains. In a dataset (x_s, t_s, u_s), $s \in \{1, \ldots, n\}$, x_s are values of the sampled 'spatial' variable x which can be defined in an unbounded domain. We will also assume that our problem is defined within a finite time horizon so that t_s are time points restricted to a bounded domain, and are thus normalizable. Our key assumption is that the solution's behavior in x can be represented by a spectral decomposition, while u's behavior in t remains unknown and is to be learned from the neural network. This is achieved by isolating the possibly unbounded spatial variables x from the bounded variables t by expressing u in terms of suitable basis functions in x with time-dependent weights. As indicated in figure 1(a), we approximate u_s using

$$u_s := u(x_s, t_s) \approx u_N(x_s, t_s) := \sum_{i=0}^{N} w_i(t_s)\phi_i(x_s), \tag{1}$$

where $\{\phi_i\}_{i=0}^{N}$ are suitable basis functions that can be used to approximate u in an unbounded domain (see figure 1(b) for a schematic of a basis function $\phi_i(x)$ that decays with x). Examples of such basis functions include, for example, the generalized Laguerre functions in $\mathbb{R}^+$ and the generalized Hermite functions in $\mathbb{R}$ [34]. In addition to being defined on an unbounded domain, spectral expansions allow high accuracy [35] calculations with errors that decay exponentially (spectral convergence) in space if the target function $u(x,t)$ is smooth.

Figure 1(c) shows a schematic of our proposed s-PINN algorithm. The variable x is directly fed into the basis functions ϕ_i instead of being used as an input in the neural network. If one wishes to connect the output $u_N(x,t;\Theta)$ of the neural network (here, Θ respresents the parameters of the neural network) to the solution of a PDE and perform backpropagation to minimize a loss functional $\mathcal{L}[u_N(x,t;\Theta), u_s(x,t)]$, it must contain spatial derivatives of u_N intrinsic to the underlying PDE. Derivatives that involve the variable x can be easily and explicitly calculated by taking derivatives of the basis functions with high accuracy while derivatives with respect to t can be obtained via automatic differentiation [36, 37].

If a function u can be written in terms of a spectral expansion in some dimensions (e.g. x in equation (1)) with appropriate spectral basis functions, we can approximate u using a multi-output neural network by solving the corresponding least squares optimization problem

$$\min_{\Theta}\left\{\sum_{s=1}^{n}\left|u_N(x_s,t_s;\Theta) - u_s\right|^2\right\},\ u_N(x,t;\Theta) = \sum_{i=0}^{N} w_i(t;\Theta)\phi_i(x), \tag{2}$$

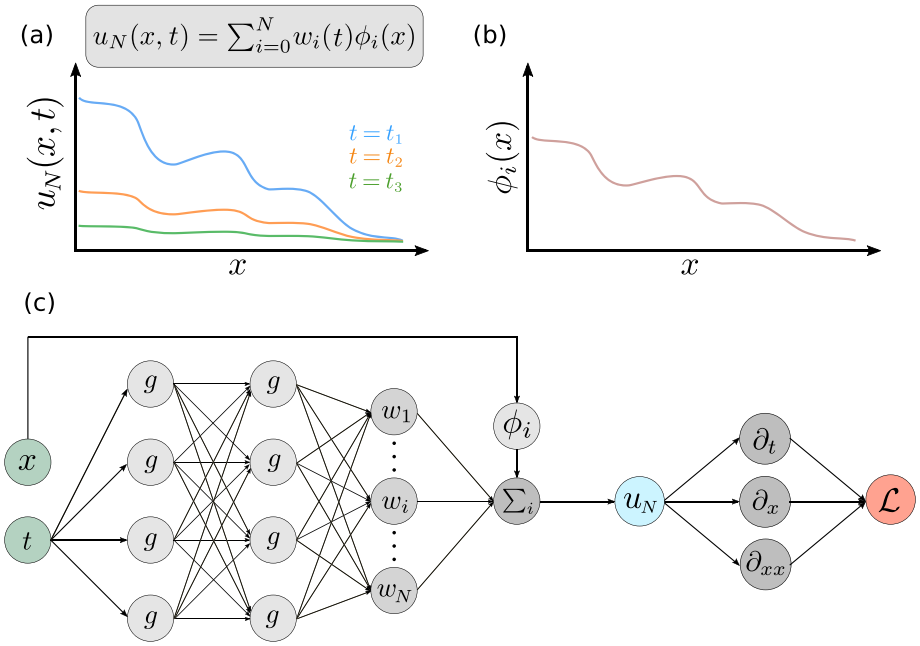

Figure 1. Solving unbounded domain problems with spectrally adapted physics-informed neural networks for functions $u_N(x,t)$ that can be expressed as a spectral expansion $u_N(x,t) = \sum_{i=0}^{N} w_i(t)\phi_i(x)$. (a) An example of a function $u_N(x,t)$ plotted at three different time points. (b) Decaying behavior of a corresponding basis function element $\phi_i(x)$. (c) PDEs in unbounded domains can be solved by combining a PINN with a neural network approximation of the spectral representation, $u_N(x,t,;\Theta) = \sum_{i=0}^{N} w_i(t;\Theta)\phi_i(x)$, and minimizing the loss function $\mathcal{L}$. Spatial derivatives of basis functions are explicitly defined and easily obtained. Here, g denotes an activation function such as the ReLU function.

where n is the number of sample points. The neural network outputs the t-dependent vector of coeffcients $w_i(t;\Theta)$. This representation will be used in the appropriate loss function depending on the application. The neural network can achieve arbitrarily high accuracy in the minimization of the loss function if it is deep enough and contains sufficiently many neurons in each layer [38]. Since the solution's spatial behavior has been approximated by the spectral expansion which could achieve high accuracy with proper ϕ_i, we shall show that solving equation (2) can be more accurate and efficient than directly fitting to u_s by a neural network without using a spectral expansion. The proper choice of basis function $\phi_i(x)$ usually depends on the domain and how the solution decays at infinity. Overviews of asymptotic properties of basis functions are given in [34, 39]. For instance, in bounded domains, using any set of basis functions in the Jacobi polynomial family leads to the same convergence order for smooth functions and usually similar performance; in a semi-unbounded domain $\mathbb{R}^+$, the generalized Laguerre functions are often used; in the whole unbounded domain $\mathbb{R}^+$, the generalized Hermite functions are a common choice if the function decays exponentially at infinity. If the solution is expected to decay algebraically at infinity, the mapped Jacobi functions, such as the modified mapped Gegenbauer functions (MMGFs) are to be used [34].

As a motivating example, we compare the approximation error of a neural network which is fed both x_s and t_s with that of the s-PINN method in which only t_s are inputted, but with the information contained in x_s imposed on the solution via the basis functions $\{\phi_i(x)\}_{i=0}^{N}$. We show that taking advantage of the prior knowledge on the x-data greatly improves training efficiency and accuracy. All neural networks that we use in our examples are based on fully connected linear layers with ReLU activation functions. Weights and biases in each layer are initially distributed according to a uniform distribution $\mathcal{U}(-\sqrt{a}, \sqrt{a})$, where a is the inverse of the number of input features. To normalize hidden-layer outputs, we apply the batch normalization technique [40]. Neural-network parameters are optimized using stochastic gradient descent.

Example 1 (Function approximation). Consider approximating the function

$$u(x,t) = \frac{8x\,\sin 3x}{(x^2+4)^2}\,t, \tag{3}$$

which decays algebraically as $u(x \to \infty, t) \sim t/|x|^3$ when $|x| \to \infty$. To numerically approximate equation (3), we choose the loss function to be the mean-squared error

$$\text{MSE} = \frac{1}{n}\sum_{s=1}^{n} \left|u_N(x_s,t_s) - u_s\right|^2. \tag{4}$$

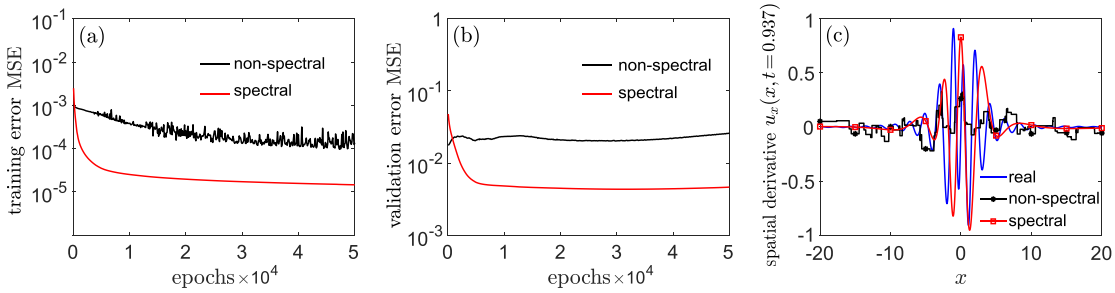

Figure 2. Example 1: function approximation. Approximation of the target function equation (3) using both standard feed-forward neural networks and a spectral multi-output neural network that learns the coefficients $w_i(t;\Theta)$ in the spectral expansion equation (1). Comparison of the approximation error using a spectral multi-output neural network (red) with the error incurred when using a standard neural-network function approximator (black). Here, both the spectral and non-spectral function approximators use the same number of parameters, but the spectral multi-output neural network converges much faster on the training set and has a smaller validation error than the standard feed-forward neural network. (a) The training curve of the spectral multi-output neural network decreases much faster than that of the standard feed-forward neural network. (b) Since the spectral multi-output neural network is better at fitting the data by taking advantage of the spectral expansion in x, its validation error is also much smaller and decreases faster. (c) Asymptotic behavior of the spatial derivatives of the analytic solution $\partial_x u(x,t)$, the feed-forward neural network $\partial_x \tilde{u}(x,t;\tilde{\Theta})$ (equation (5)), and the spectral neural network $\partial_x u_N(x,t;\Theta)$ (equation (7)). The feed-forward neural network fails to capture the function's behavior when $|x|$ is large because $\partial_x \tilde{u}(x,t;\tilde{\Theta})$ is not vanishing for large $|x|$, but the spectral approximation equation (7) leads to smaller errors because $\partial_x u_N(x,t;\Theta)$ better approximates $\partial_x u(x,t)$ especially when $|x|$ is large. Here, $t = 0.937$ is randomly chosen from one of the training samples.

A standard feed-forward neural network approach is applied by inputting *both* x_s and t_s into a five-layer, 15 neuron-per-layer network defined by the neural network parameters $\tilde{\Theta}$ to find a numerical approximation to

$$u_N(x_s,t_s) := \tilde{u}(x_s,t_s;\tilde{\Theta}) \tag{5}$$

by minimizing equation (4) with respect to $\tilde{\Theta}$.

To apply a multi-output neural network to this problem, we need to choose an appropriate spectral representation of the spatial dependence of equation (3), in the form of equation (2). To capture an algebraic decay at infinity as well as the oscillatory behavior resulting from the $\sin(3x)$ term, we start from the MMGFs [41]

$$R_i^{\lambda,\beta}(x) = (1+(\beta x)^2)^{-(\lambda+1)/2} C_i^{\lambda}\left(\beta x/\sqrt{1+(\beta x)^2}\right),\ x\in\mathbb{R}, \tag{6}$$

where $C_i^{\lambda}(\cdot)$ is the Gegenbauer polynomial of order i. At infinity, the MMGFs decay as $R_i^{\lambda,\beta}(x) \sim \mathrm{sign}(x)^i \frac{(2\lambda)^{(i)}}{i!}(1+(\beta x)^2)^{-(\lambda+1)/2}$, where $(2\lambda)^{(i)}$ is the ith rising factorial of 2λ. A suitable basis ϕ_i needs to include functions that decay more slowly than x^{-3}. If we choose $\beta = 1/4$ and the special case $\lambda = 0$, the basis function is defined as $\phi_i(x) = R_i^{0,\beta}(x) \equiv (1+(\beta x)^2)^{-1/2} T_i(\beta x/\sqrt{1+(\beta x)^2})$, where T_i are the Chebyshev polynomials. We thus use

$$u_N(x_s,t_s;\Theta) = \sum_{i=0}^{N=9} w_i(t_s;\Theta) R_i^{0,\beta}(x_s) \tag{7}$$

in equation (4) and use a four-layer neural network with 15 neurons per layer to learn the coefficients $\{w_i(t;\Theta)\}_{i=0}^{1}4$ by minimizing the MSE (equation (4)) with respect to Θ. The total numbers of parameters for both the four-layer spectral multi-output neural network and the normal five-layer neural network are the same. The training set and the validation set each contain $n = 200$ pairs of values $(x,t)_s = (x_s,t_s)$ where x_s are sampled from the Cauchy distribution, $x_s \sim \mathcal{C}(12,0)$, and $t_s \sim \mathcal{U}(0,1)$. For each pair (x_s,t_s), we find $u_s \equiv u(x_s,t_s)$ using equation (3). The positions x_s are sampled from the unbounded domain $\mathbb{R}$ and cannot be normalized (the expectation and variance of the Cauchy distribution do not exist). The minimum (maximum) value of x in the training set and the validation set are -18.65 (50.32) and -721.50 (120.01), respectively.

We set the learning rate $\eta = 5\times10^{-4}$ and plot the training and validation MSEs (equation (4)) as a function of the number of training epochs in figure 2. Figures 2(a) and (b) show that the spectral multi-output neural network yields smaller errors since it naturally and efficiently captures the oscillatory and decaying feature of the underlying function u from equation (3). Directly fitting $u \approx \tilde{u}$ leads to over-fitting on the training set which does nothing to reduce the validation error. We can see from figure 2(c) that using the feed-forward neural network, equation (5) results in a nonvanishing spatial derivative when $|x|$ is large. Such an approximation to the original function $u(x,t)$, which vanishes for large $|x|$, is thus inaccurate. On the other hand, the

spatial derivative of the spectral neural network equation (7) better fits $u(x,t)$ especially as $|x| \to \infty$. Therefore, it is important to take advantage of the data structure, in this case, using the spectral expansion to represent the function's known oscillations and decay as $x \to \infty$.

In this and subsequent examples, all computations are performed using Python 3.8.10 on a laptop with a 4-core Intel® i7-8550U CPU @ 1.80 GHz.

3. Application to solving PDEs

In this section, we show that spectrally adapted neural networks can be combined with PINNs which we shall call s-PINNs. We apply s-PINNs to numerically solve PDEs, and in particular, spatiotemporal PDEs in unbounded domains for which standard PINN approaches cannot be directly applied. Although we mainly focus on solving spatiotemporal problems, s-PINNs are also applicable to other types of PDEs.

Again, we assume that the problem is defined over a finite time horizon t while the spatial variable x may be defined in an unbounded domain. Assuming the solution's asymptotic behavior in x is known, we approximate it by a spectral expansion in x with suitable basis functions (e.g. MMGFs in example 1 for describing algebraic decay at infinity). Assuming $\mathcal{M}$ is an operator that only involves the spatial variable x (e.g. ∂_x, ∂_x^2, etc), we can represent the solution to the spatiotemporal PDE $\partial_t u = \mathcal{M}[u](x,t)$ by the spectral expansion in equation (2) with expansion coefficients $\{w_i(t;\Theta)\}$ to be learned by a neural network with parameters Θ. If the solution's behavior in both x and t are known and one can find proper basis functions in both the x and t directions, then one could use a spectral expansion in both x and t to solve the PDE directly without time-stepping. However, it is often the case that the time dependence is unknown and $u(x,t)$ needs to be solved step-by-step in time.

As in standard PINNs, we use a high-order Runge–Kutta scheme to advance time by uniform timesteps Δt. What distinguishes our s-PINNs from standard PINNs is that only the intermediate times t_s between timesteps are provided as inputs to the neural network, while the outputs contain global spatial information (the spectral expansion coefficients), as shown in figure 1(c). Over a longer time scale, the optimal basis functions in the spectral expansion equation (2) may change. Therefore, one can use new adaptive spectral methods proposed in [32, 33]. Using s-PINNs to solve PDEs has the advantages that they can (i) accurately represent spatial information via spectral decomposition, (ii) convert solving a PDE into an optimization and data fitting problem, (iii) easily implement high-order, implicit schemes to advance time with high accuracy, and (iv) allow the use of recently developed spectral-adaptive techniques that dynamically find the most suitable basis functions.

The approximated solution to the PDE $\partial_t u = \mathcal{M}[u](x,t)$ can be written at discrete timesteps $t_{j+1} - t_j = \Delta t$ as

$$u_N(x, t_{j+1}; \Theta_{j+1}) = \sum_{i=0}^{N} w_i(t_{j+1}; \Theta_{j+1}) \phi_i(x), \tag{8}$$

where $\Theta_{j+1}, j \geqslant 1$ is the parameter set of the neural network used in the time interval $(j\Delta t, (j+1)\Delta t)$. In order to forward time from $t_j = j\Delta t$ to $t_{j+1} = (j+1)\Delta t$, we can use, e.g. a Kth-order implicit Runge–Kutta scheme, with $0 < c_s < 1$ $(s = 1, \ldots, K)$ as parameters describing different collocation points in time and a_{rs}, b_r $(r = 1, \ldots, K)$ the associated coefficients.

Given $u(x, t_j)$, the Kth-order implicit Runge–Kutta scheme aims to approximate $u(x, t_j + c_s\Delta t)$ and $u(x, t_j + \Delta t)$ through

$$\begin{aligned} u_N(x, t_j + c_s\Delta t) &= u(x, t_j) + \sum_{r=1}^{K} a_{rs} \mathcal{M}\left[u_N(x, t_j + c_r\Delta t)\right], \\ u_N(x, t_j + \Delta t) &= u(x, t_j) + \sum_{r=1}^{K} b_r \mathcal{M}\left[u_N(x, t_j + c_r\Delta t)\right]. \end{aligned} \tag{9}$$

With the starting point $u_N(t_0, x; \Theta_0) := u_N(t_0, x)$ defined by the initial condition at t_0, we define the target function as the sum of squared errors

$$\begin{aligned} \mathrm{SSE}_j = \sum_{s=1}^{K} \left\| u_N(x, t_j + c_s\Delta t; \Theta_{j+1}) - u_N(x, t_j; \Theta_j) - \sum_{r=1}^{K} a_{sr} \mathcal{M}[u_N(x, t_j + c_r\Delta t; \Theta_{j+1})] \right\|_2^2 \\ + \left\| u_N(x, t_j + \Delta t; \Theta_{j+1}) - u_N(x, t_j; \Theta_j) - \sum_{r=1}^{K} b_r \mathcal{M}[u_N(x, t_j + c_r\Delta t; \Theta_{j+1})] \right\|_2^2, \end{aligned} \tag{10}$$

where the L^2 norm is taken over the spatial variable x. Minimization of equation (10) provides a numerical solution at t_{j+1} given its value at t_j. If coefficients in the PDE are sufficiently smooth, we can use the basis function expansion in equation (8) for u_N and find that the weights at the intermediate Runge–Kutta timesteps can be written as the Taylor expansion

$$w_i(t_j + c_r\Delta t; \Theta_{j+1}) = \sum_{\ell=0}^{\infty} \frac{w_i^{(\ell)}(t_j; \Theta_{j+1})}{\ell!} (c_r \Delta t)^{\ell}, \tag{11}$$

where $w_i^{(\ell)}(t_j)$ is the ℓth derivative of w_i with respect to time, evaluated at t_j. Therefore, the neural network is learning the mapping $t_j + c_s\Delta t \to \sum_{\ell=0}^{\infty} w_i^{(\ell)}(t_j)(c_s\Delta t)^{\ell}/\ell!$ for every i by minimizing the loss function equation (10).

Example 2 (Solving bounded domain PDEs). Before focusing on the application of s-PINNs to PDEs whose solution is defined in an unbounded domain, we first consider the numerical solution of a PDE in a bounded domain to compare the performance of the spectral PINN method (using recently developed adaptive methods) to that of the standard PINN.

Consider the following PDE:

$$\begin{aligned} \partial_t u &= \left(\frac{x+2}{t+1}\right) \partial_x u, \ x \in (-1, 1), \\ u(x, 0) &= \cos(x+2), \ u(1, t) = \cos(3(t+1)), \end{aligned} \tag{12}$$

which admits the analytical solution $u(x, t) = \cos((t+1)(x+2))$. In this example, we use Chebyshev polynomials $T_i(x)$ as basis functions and the corresponding Chebyshev-Gauss-Lobatto quadrature collocation points and weights such that the boundary $u(1, t) = \cos(3(t+1))$ can be directly imposed at a collocation point $x = 1$.

Since the solution becomes increasingly oscillatory in x over time, an ever-increasing expansion order (i.e. the number of basis functions) is needed to accurately capture this behavior. Between consecutive timesteps, we employ a recently developed p-adaptive technique for tuning the expansion order [33]. This method is based on monitoring and controlling a frequency indicator $\mathcal{F}(u_N)$ defined by

$$\mathcal{F}(u_N) = \left(\frac{\sum_{i=N-[\frac{N}{3}]+1}^{N} \gamma_i w_i^2}{\sum_{i=0}^{N} \gamma_i w_i^2} \right)^{\frac{1}{2}}, \tag{13}$$

where $\gamma_i := \int_{-1}^{1} T_i^2(x)(1-x^2)^{-1/2}\mathrm{d}x$. The frequency indicator $\mathcal{F}(u_N)$ measures the proportion of high-frequency waves and serves as a lower error bound of the numerical solution $u_N(x, t; \Theta) := \sum_{i=0}^{N} w_i(t; \Theta) T_i(x)$. When $\mathcal{F}(u_N)$ exceeds its previous value by more than a factor ρ, the expansion order is increased by one. The indicator is then updated and the factor ρ also is scaled by a parameter $\gamma \geqslant 1$.

We use a fourth-order implicit Runge–Kutta method to advance time in the loss function (10) and in order to adjust the expansion order in a timely way, we take $\Delta t = 0.01$. The initial expansion order $N = 8$, and the two parameters used to determine the threshold of adjusting the expansion order are set to $\rho = 1.5$ and $\gamma = 1.3$. A neural network with $N_H = 4$ layers and $H = 200$ neurons per layer is used in conjunction with the loss function (10) to approximate the solution of equation (12). We compare the results obtained using the s-PINN method with those obtained using a fourth-order implicit Runge–Kutta scheme with $\Delta x = \frac{1}{256}, \Delta t = 0.01$ in a standard PINN approach [3], also using $N_H = 4$ and $H = 200$.

Figure 3 shows that s-PINNs can be used to greatly improve accuracy because the spectral method can recover exponential convergence in space, and when combined with a high-order accurate implicit scheme in time, the overall error is small. In particular, the large error shown in figure 3 of the standard PINN suggests that the error of applying auto-differentiation to calculate the spatial derivative is significantly larger than the spatial derivatives calculated using spectral methods. Moreover, when equipping spectral PINNs with the p-adaptive technique to dynamically adjust the expansion order, the frequency indicator can be controlled, leading to even smaller errors as shown in figures 3(b) and (c).

Computationally, using our 4-core laptop on this example, the standard PINN method requires $\sim 10^6$ s while the s-PINN approach with and without adaptive spectral techniques (dynamically increasing the expansion order N) required 1711 and 1008 s, respectively. Thus, s-PINN methods can be computationally more efficient than the standard PINN approach. This advantage can be better understood by noting that training of standard PINNs requires time $\sim \mathcal{O}(\sum_{i=0}^{N_H} H_i H_{i+1})$ (H_i is the number of neurons in the ith layer) to calculate

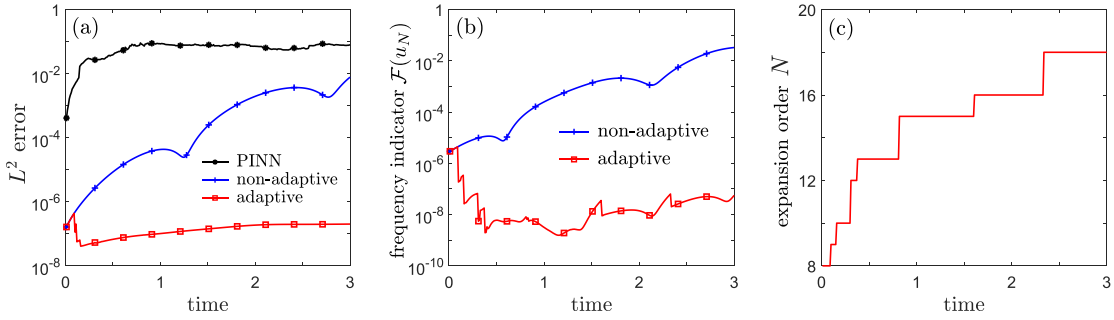

Figure 3. Example 2: solving equation (12) in a bounded domain. L^2 errors, frequency indicators, and expansion order associated with the numerical solution of equation (12) using the adaptive s-PINN method with a timestep $\Delta t = 0.01$. (a) In a bounded domain, the s-PINNs, with and without the adaptive spectral technique, have smaller errors than the standard PINN (black). Moreover, the s-PINN method combined with a *p*-adaptive technique that dynamically increases the number of basis functions (red) exhibits a smaller error than the non-adaptive s-PINN (blue). The higher accuracy of the adaptive s-PINN is a consequence of maintaining a small frequency indicator (13), as shown in (b). (c) Keeping the frequency indicator at small values is realized by increasing the spectral expansion order.

each spatial derivative (e.g. $\partial_x u, \partial_x^2 u, \ldots$) by autodifferentiation [42]. However, in an s-PINN, since a spectral decomposition $u_N(x,t;\Theta)$ has been imposed, the computational time to calculate derivatives of all orders is $\mathcal{O}(N)$, where N is the expansion order. Since $\sum_{i=0}^{N_H} H_i H_{i+1} \geqslant \sum_{i=0}^{N_H} H_i$ and the total number of neurons $\sum_{i=0}^{N_H} H_i$ is usually much larger than the expansion order N, using s-PINNs can substantially reduce computational cost.

In bounded-domain problems, there are many other good machine-learning-based PDE solvers against which we can compare, such as the DeepONet method [43], its PINN extension [44], and the Fourier neural operator method [45]. However, what distinguishes s-PINNs from the standard PINN framework is that the latter uses spatial and temporal variables as neural-network inputs, implicitly assuming that all variables are normalizable especially when batch-normalization techniques are applied while training the underlying neural network. Our s-PINN approach relies on spectral expansions to represent the dependence of a function $u(x,t)$ on the spatial variable x, which can then be defined in unbounded domains and does not need to be normalizable. Thus, our s-PINN method provides a novel machine-learning-based PDE solver for unbounded-domain spatiotemporal problems. In the following example, we shall explore how our s-PINN is applied to solving a PDE defined in $(x,t) \in \mathbb{R}^+ \times [0,T]$.

Example 3 (Solving unbounded domain PDEs). Consider the following PDE, which is similar to equation (12) but is defined in $(x,t) \in \mathbb{R}^+ \times [0,T]$:

$$\partial_t u = -\left(\frac{x}{t+1}\right)\partial_x u, \quad u(x,0) = e^{-x}, \ u(0,t) = 1. \tag{14}$$

Equation (14) admits the analytical solution $u(x,t) = \exp[-x/(t+1)]$. In this example, we use the basis functions $\{\hat{\mathcal{L}}_i^{\beta}(x)\} := \{\hat{\mathcal{L}}_i^{(0)}(\beta x)\}$ where $\hat{\mathcal{L}}_i^{(0)}(x)$ is the generalized Laguerre function of order i defined in [34]. Here, we use the Laguerre–Gauss quadrature collocation points and weights so that $x = 0$ is *not* included in the collocation node set. We use a fourth-order implicit Runge–Kutta method to minimize the SSE equation (10) by advancing time. In order to address the boundary condition, we augment the loss function in equation (10) with terms that represent the cost of deviating from the boundary condition:

$$\begin{aligned}\mathrm{SSE}_j = &\sum_{s=1}^{K}\left\| u_N(x,t_j+c_s\Delta t;\Theta_{j+1}) - u_N(x,t_j;\Theta_j) - \sum_{r=1}^{K} a_{sr}\mathcal{M}[u_N(x,t_j+c_r\Delta t;\Theta_{j+1})]\right\|_2^2 \\ &+ \left\| u_N(x,t_j+\Delta t;\Theta_{j+1}) - u_N(x,t_j;\Theta_j) - \sum_{r=1}^{K} b_r\mathcal{M}[u_N(x,t_j+c_r\Delta t;\Theta_{j+1})]\right\|_2^2 \\ &+ \sum_{s=1}^{K}\left[u_N(0,t_j+c_s\Delta t;\Theta_{j+1}) - u(0,t_j+c_s\Delta t)\right]^2 + \left[u_N(0,t_{j+1};\Theta_{j+1}) - u(0,t_{j+1})\right]^2,\end{aligned} \tag{15}$$

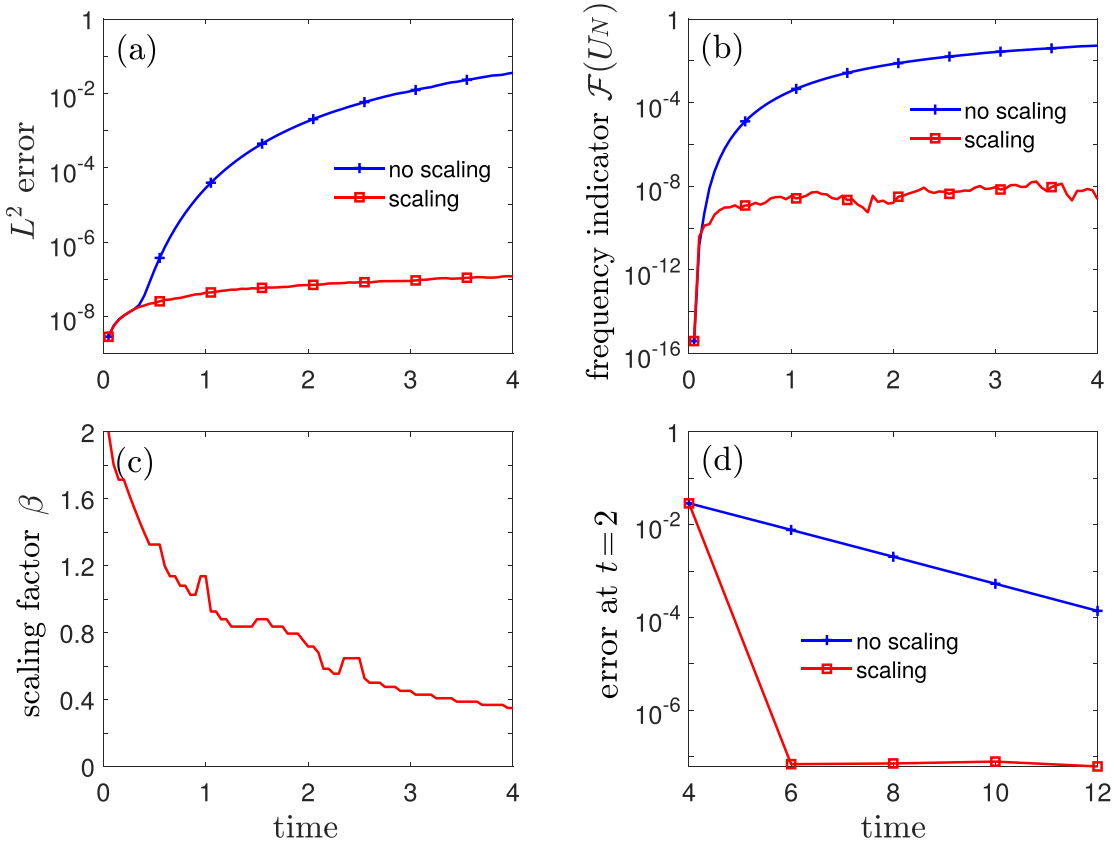

Figure 4. Example 3: solving equation (14) in an unbounded domain. L^2 error, frequency indicator, and expansion order associated with the numerical solution of equation (14) using the s-PINN method combined with the spectral scaling technique. (a) The s-PINN method with the scaling technique (red) has a smaller error than the s-PINN without scaling (blue). The higher accuracy of the adaptive s-PINN is a consequence of maintaining a smaller frequency indicator equation (13), as shown in (b). (c) Keeping the frequency indicator at small values is possible by reducing the scaling factor so that the basis functions decay more slowly at infinity. The timestep is $\Delta t = 0.05$. (d) The errors for the spectral method with and without scaling at $t = 2$. When the scaling factor is properly adjusted, very high accuracy can be obtained with only a few basis functions. Not dynamically adjusting the scaling factor leads to a much slower convergence.

where the last two terms push the constraints associated with the Dirichlet boundary condition at $x = 0$ at all time points:

$$u_N(0, t_j + c_s\Delta t; \Theta_{j+1}) = u(0, t_j + c_s\Delta t), \quad u_N(0, t_{j+1}; \Theta_{j+1}) = u(0, t_{j+1}), \tag{16}$$

where in this example, $u(0, t_j + c_s\Delta t) = u(0, t_{j+1}) \equiv 1$.

Because the solution of equation (14) becomes more diffusive with x (i.e. decays more slowly at infinity), it is necessary to decrease the scaling factor β to allow basis functions to decay more slowly at infinity. Between consecutive timesteps, we adjust the scaling factor by applying the scaling algorithm proposed in [32]. Thus, we dynamically adjust the basis functions in equation (1). As with the p-adaptive technique we used in example 2, the scaling technique also relies on monitoring and controlling the frequency indicator given in equation (13). In order to efficiently and dynamically tune the scaling factor, we set $\Delta t = 0.05$. The initial expansion order is $N = 8$, the initial scaling factor is $\beta = 2$, the scaling factor adjustment ratio is set to $q = 0.95$, and the threshold for tuning the scaling factor is set to $\nu = 1/(0.95)$. A neural network with 3 intermediate layers and 100 neurons per layer is used in conjunction with the loss function given in equation (10). Figure 4(a) shows that s-PINNs can achieve very high accuracy even when a relatively large timestep ($\Delta t = 0.05$) is used. Scaling techniques to dynamically control the frequency indicator are also successfully incorporated into s-PINNs, as shown in figures 14(b) and (c), and very high accuracy can be achieved with only a few basis functions, as shown in figure 14(d). Actually, such spatiotemporal diffusive behavior in unbounded domains distinguishes unbounded-domain problems from bounded-domain problems, as we have to dynamically adjust the scaling factor over time using the scaling technique in [32].

In equation (14), we imposed a Dirichlet boundary condition by modifying the SSE equation (15) to include boundary terms. Other types of boundary conditions can be applied in s-PINNs by including boundary constraints in the SSE as in standard PINN approaches.

In the next example, we focus on solving a PDE with two spatial variables, x and y, each defined on an unbounded domain.

Example 4 (Solving 2D unbounded domain PDEs). Consider the two-dimensional heat equation on $(x,y)\in\mathbb{R}^2$

$$\partial_t u(x,y,t)=\Delta u(x,y,t),\quad u(x,y,0)=\frac{1}{\sqrt{2}}e^{-x^2/12-y^2/8}, \tag{17}$$

which admits the analytical solution

$$u(x,y,t)=\frac{1}{\sqrt{(t+3)(t+2)}}\exp\left[-\frac{x^2}{4(t+3)}-\frac{y^2}{4(t+2)}\right]. \tag{18}$$

Note that the solution spreads out over time in both dimensions, i.e. it decays more slowly at infinity as time increases. Therefore, we apply the scaling technique to capture the increasing spread by adjusting the scaling factors β_x and β_y of the generalized Hermite basis functions. Generalized Hermite functions of orders $i=0,\ldots,N_x$ and $\ell=0,\ldots,N_y$ are used in the x and y directions, respectively.

In order to solve equation (17), we multiply it by any test function $v\in H^1(\mathbb{R})$ and integrate the resulting equation by parts to convert it to the weak form $(\partial_t u,v)=-(\nabla u,\nabla v)$. Solving the weak form of equation (17) ensures numerical stability. When implementing the spectral method, the goal is to find

$$u^{\beta_x,\beta_y}_{N_x,N_y}(x,y,t)=\sum_{i=0}^{N_x}\sum_{\ell=0}^{N_y}w_{i,\ell}(t)\hat{\mathcal{H}}^{\beta_x}_{i,0}(x)\hat{\mathcal{H}}^{\beta_y}_{\ell,0}(y), \tag{19}$$

where $\hat{\mathcal{H}}^{\beta_x}_{i,0}$, $\hat{\mathcal{H}}^{\beta_y}_{\ell,0}$ are generalized Hermite functions defined in table 1 such that $(\partial_t u,v)=-(\nabla u,\nabla v)\ t\in(t_j,t_{j+1})$ for all $v\in P^{\beta_x}_{N_x,0}\times P^{\beta_y}_{N_y,0}$, $t\in(t_j,t_{j+1})$. This allows one to advance time from t_j to t_{j+1} given $u^{\beta_x,\beta_y}_{N_x,N_y}(x,y,t_j)$.

Tuning the scaling factors β_x,β_y across different timesteps is achieved by monitoring the frequency indicators in the x- and y-directions, $\mathcal{F}_x$ and $\mathcal{F}_y$, as detailed in [32]. We use initial expansion orders $N_x=N_y=8$ and scaling factors $\beta_x=0.4,\beta_y=0.5$. The ratio and threshold for adjusting the scaling factors, are set to be $q=0.95$ and $\nu^{-1}=0.95$. The timestep $\Delta t=0.1$ is used to adjust both scaling factors in both dimensions in a timely manner and a fourth order implicit Runge–Kutta scheme is used for numerical integration. The neural network that we use to learn $w_{i,\ell}(t)$ has 5 intermediate layers with 150 neurons in each layer.

The results depicted in figure 5(a) show that an s-PINN using the scaling technique can achieve high accuracy by using high-order Runge–Kutta schemes in minimizing the SSE equation (10) and by properly adjusting β_x and β_y (shown in figure 5(b)) to control the frequency indicators $\mathcal{F}_x$ and $\mathcal{F}_y$ (shown in figures 5(c) and (d)). The s-PINNs can be extended to higher spatial dimensions by calculating the numerical solution expressed in tensor product form as in equation (19).

Since our method outputs spectral expansion coefficients, using the full tensor product in the spatial spectral decomposition leads to a number of outputs that increase exponentially with dimensionality. The very wide neural networks needed for such high-dimensional problems results in less efficient training. However, unlike other recent machine–learning–based PDE solvers or PDE learning methods [45, 46] that explicitly rely on a spatial discretization of grids or meshes, the curse of dimensionality can be partially mitigated in our s-PINN method. By using a hyperbolic cross space [47], we can effectively reduce the number of coefficients needed to accurately reconstruct the numerical solution. In the next example, we solve a 3D parabolic spatiotemporal PDE, similar to that in example 4, but we demonstrate how implementing a hyperbolic cross space can reduce the number of outputs and boost training efficiency.

Example 5 (Solving 3D unbounded domain PDEs). Consider the (3+1)-dimensional heat equation

$$\partial_t u(x,y,z,t)=\Delta u(x,y,z,t),\quad u(x,y,0)=\frac{1}{\sqrt{6}}e^{-x^2/12-y^2/8-z^2/4}, \tag{20}$$

which admits the analytical solution

$$u(x,y,z,t)=\frac{1}{\sqrt{(t+3)(t+2)(t+1)}}\exp\left[-\frac{x^2}{4(t+3)}-\frac{y^2}{4(t+2)}-\frac{z^2}{4(t+1)}\right] \tag{21}$$

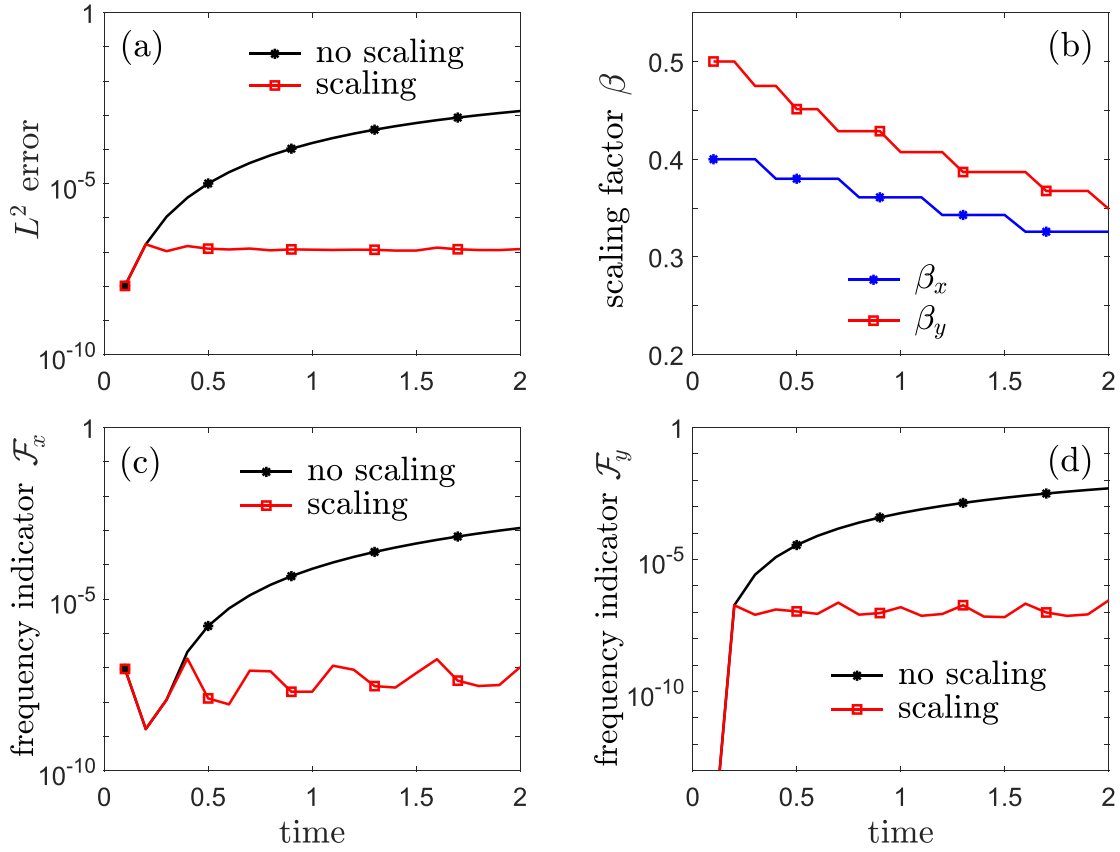

Figure 5. Example 4: Solving a higher-dimensional (2D spatial) unbounded domain PDE (equation (17)). L^2 error, scaling factor, and frequency indicators associated with the numerical solution of equation (17) using s-PINNs, with and without dynamic scaling. (a) L^2 error as a function of time. The s-PINNs that are equipped with the scaling technique (red) achieve higher accuracy than those without (black). (b) The scaling factors β_x (blue) and β_y (red) as functions of time. Both scaling factors are decreased to match the spread of the solution in both the x and y directions. Scaling factors are adjusted to maintain small frequency indicators in the x-direction (c), and in the y-direction (d). In all computations, the timestep is $\Delta t = 0.1$.

for $(x, y, z) \in \mathbb{R}^3$. If we use the full tensor product of spectral expansions with expansion orders $N_x = N_y = N_z = 9$, we will need to output $10^3 = 1000$ expansion coefficients, and in turn, a relatively wide neural network with many parameters will be needed to generate the corresponding weights as shown in figure 1(c). Training such wide networks can be inefficient. However, many of the spectral expansion coefficients are close to zero and can be eliminated without compromising accuracy. One way to select expansion coefficients is to use the hyperbolic cross space technique [47] to output coefficients of the generalized Hermite basis functions only in the space

$$V_{N,\gamma_\times}^{\vec{\beta},\vec{x}_0} := \text{span}\left\{\hat{\mathcal{H}}_{N_x}(\beta_x x)\hat{\mathcal{H}}_{N_y}(\beta_y y)\hat{\mathcal{H}}_{N_z}(\beta_z z) : |\vec{N}|_{\text{mix}}\|\vec{N}\|_\infty^{-\gamma_\times} \leqslant N^{1-\gamma_\times}\right\},$$
$$\vec{N} := (N_x, N_y, N_z), \quad |\vec{N}|_{\text{mix}} := \max\{N_x, 1\}\max\{N_y, 1\}\max\{N_z, 1\}, \tag{22}$$

where the hyperbolic space index $\gamma_\times \in (-\infty, 1)$. Taking $\gamma_\times = -\infty$ in equation (22) corresponds to the full tensor product with $N+1$ basis functions in each dimension. $\beta_x, \beta_y, \beta_x$ are the scaling factors for the basis functions in the x, y, z directions, and N_x, N_y, N_z are the orders of the basis function expansions in the x, y, z directions. For fixed N in equations (22), the number of total basis function tend to decrease with increasing $\gamma_\times$. We set $N = 9$ in equation (22) and use the initial scaling factors $\beta_x = 0.4, \beta_y = 0.5, \beta_z = 0.7$. Using a fourth-order implicit Runge–Kutta scheme with timestep $\Delta t = 0.2$, we set the ratio and threshold for adjusting the scaling factors are set to $q = 0.95$ and $\nu^{-1} = 0.95$ in each dimension.

To illustrate the potential numerical difficulties arising from outputting large numbers of coefficients when solving higher-dimensional spatiotemporal PDEs, we use a neural network with two hidden layers and different numbers of neurons in the intermediate layers. We also adjust $\gamma_\times$ to explore how decreasing the number of coefficients can improve training efficiency. Our results are listed in table 2.

The results shown in table 2 indicate that, compared to using the full tensor product $\gamma_\times = -\infty$, implementing the hyperbolic cross space with a moderate $\gamma_\times = -1$ or 0, the total number of outputs is significantly reduced, leading to faster training and better accuracy. However, increasing the hyperbolicity to $\gamma_\times = \frac{1}{2}$,

Table 2. Example 5: Applying hyperbolic cross space and s-PINNs to the (3+1) dimensional PDE equation (20). Applying the hyperbolic cross space (equation 22), we record the L^2 error as well as the training time (in seconds). The number of coefficients (outputs in the neural network) for $\gamma_\times = -\infty, -1, 0, \frac{1}{2}$ are 1000, 205, 141, 110, respectively. Using $\gamma_\times = -1$ or 0 leads to the most accurate results. The training time tends to increase with the number of outputs (a smaller $\gamma_\times$ corresponds to more outputs). By comparing the results in different rows for the same column, it can be seen that more outputs require a wide neural network for training.

H \ $\gamma_\times$	$-\infty$	-1	0	$\frac{1}{2}$
200	2.217×10^{-3}, (22911)	1.651×10^{-4}, (4309)	5.356×10^{-5}, (2886)	3.173×10^{-4}, (3956)
400	1.072×10^{-3}, (26725)	2.970×10^{-5}, (7014)	5.356×10^{-5}, (3309)	3.173×10^{-4}, (2356)
700	2.276×10^{-3}, (43923)	2.900×10^{-5}, (3133)	5.356×10^{-5}, (3229)	3.173×10^{-4}, (2098)
1000	7.871×10^{-5}, (55880)	2.901×10^{-5}, (3002)	5.356×10^{-5}, (2016)	3.173×10^{-4}, (1894)

the error increases relative to using $\gamma_\times = -1, 0$ because some useful, nonzero coefficients are excluded. Also, comparing the results across different rows, wider layers lead to both more accurate results and faster training speed. The sensitivity of our s-PINN method to the number of intermediate layers in the neural network and the number of neurons in each layer are further discussed in example 7. Overall, in higher-dimensional problems, there is a balance between computational cost and accuracy as the number of outputs needed will grow fast with dimensionality. Spectrally-adapted PINNs can easily incorporate a hyperbolic cross space so that the total number of outputs can be reduced to a manageable number for moderate-dimensional problems. Finding the optimal hyperbolicity index $\gamma_\times$ for the cross space equation (22) will be problem-specific.

In the next example, we explore how s-PINNs can be used to solve the Schrödinger equation in $x \in \mathbb{R}$. Solving this complex-valued equation poses substantial numerical difficulties as the solution exhibits diffusive, oscillatory, and convective behavior [26].

Example 6 (Solving an unbounded domain Schrödinger equation). We seek to numerically solve the following Schrödinger equation defined on $x \in \mathbb{R}$

$$\mathrm{i}\partial_t \psi(x,t) = -\partial_x^2 \psi(x,t), \quad \psi(x,0) = \frac{1}{\sqrt{\zeta}} \exp\left[\mathrm{i}kx - \frac{x^2}{4\zeta}\right]. \tag{23}$$

For reference, equation (23) admits the analytical solution

$$\psi(x,t) = \frac{1}{\sqrt{\zeta + \mathrm{i}t}} \exp\left[\mathrm{i}k(x-kt) - \frac{(x-2kt)^2}{4(\zeta+\mathrm{i}t)}\right]. \tag{24}$$

As in example 4, we shall numerically solve equation (23) in the weak form

$$(\partial_t \Psi(x,t), v) + \mathrm{i}(\partial_x \Psi(x,t), \partial_x v) = 0, \quad \forall v \in H^1(\mathbb{R}). \tag{25}$$

Since the solution to equation (23) decays as $\sim \exp[-x^2/(4\sqrt{(\zeta^2+t^2)})]$ at infinity, we shall use the generalized Hermite functions as basis functions. The solution is rightward-translating for $k > 0$ and increasingly oscillatory and spread out over time. Hence, as detailed in [33], we apply three additional adaptive spectral techniques to improve efficiency and accuracy: (i) a scaling technique to adjust the scaling factor β over time in order to capture diffusive behavior, (ii) a moving technique to adjust the center of the basis function x_L to capture convective behavior, and (iii) a p-adaptive technique to increase the number of basis functions N to better capture the oscillations. We set the initial parameters $\beta = 0.8, x_L = 0, N = 24$ at $t = 0$. The scaling factor adjustment ratio and the threshold for adjusting the scaling factor are $q = \nu^{-1} = 0.95$, the minimum and maximum change in displacements of the basis functions are 0.004 and 0.1 within each timestep, respectively, and the threshold for moving is 1.001. Finally, the thresholds of the p-adaptive technique are set to $\rho = \rho_0 = 2$ and $\gamma = 1.4$.

Generally speaking, it is desirable to set the adaptive spectral method scaling hyperparameters to $\nu \gtrsim 1 \gtrsim q$. When implementing adaptive moving, it is desirable to make the change in the basis functions' displacement as accurate as possible by setting a small minimum change in displacement per timestep, a large maximum change in displacement per timestep, and a threshold for moving which is slightly larger than 1. For the p-adaptive technique that adjusts the spectral expansion order, there is a cost-accuracy tradeoff; setting ρ and γ to small values but ρ_0 to a large value leads to smallest errors but higher computational costs. A more detailed and theoretical discussion of how the choices of those hyperparameters influence the results is given in [48].

To numerically solve equation (25), a fourth-order implicit Runge–Kutta scheme is applied to advance time with timestep $\Delta t = 0.1$. The neural network underlying the s-PINN that we use in this example contains

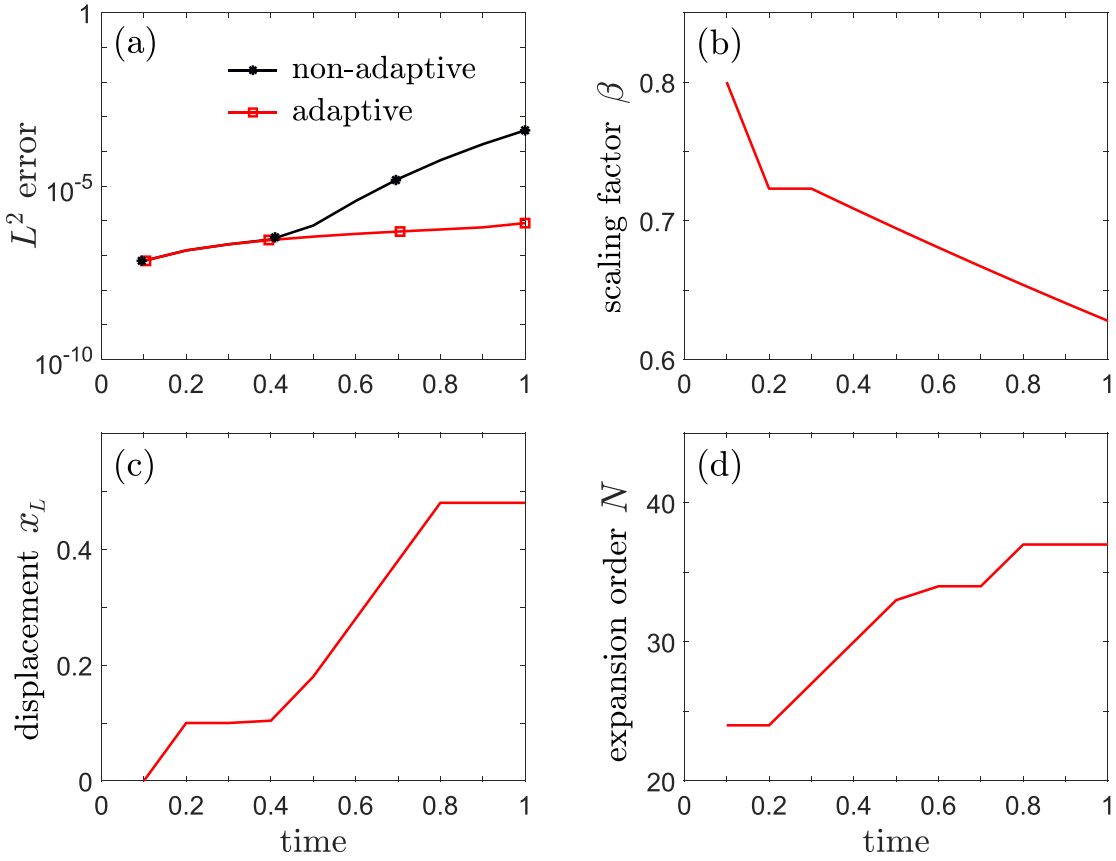

Figure 6. Example 6: Solving the Schrödinger equation (equation (23)) in an unbounded domain. Approximation error, scaling factor, displacement, and expansion order associated with the numerical solution of equation (23) using adaptive (red) and non-adaptive (black) s-PINNs. (a) Errors for numerically solving equation (23) with and without adaptive techniques. (b) The change of the scaling factor which decreases over time as the solution becomes more spread out. (c) The displacement of the basis functions x_L which is increased as the solution moves rightwards. (d) The expansion order N increases over time as the solution becomes more oscillatory. A timestep $\Delta t = 0.1$ was used.

13 layers with 100 neurons in each layer. Figure 6(a) shows that the s-PINN with adaptive spectral techniques leads to very high accuracy as it can properly adjust the basis functions over a longer timescale (across different timesteps), while not adapting the basis functions results in larger errors. Figures 6(b)–(d) show that the scaling factor β decreases over time to match the spread of the solution, the displacement of the basis function x_L increases in time to capture the rightward movement of the basis functions, and the expansion order N increases to capture the solution's increasing oscillatory behavior. Our results indicate that our s-PINN method can effectively utilize all three adaptive algorithms.

We now explore how the timestep and the order of the implicit Runge–Kutta method affect the approximation error, i.e. to what extent can we relax the constraint on the timestep and maintain the accuracy of the basis functions, or, if higher-order Runge–Kutta schemes are better. Another feature to explore is the neural network structure, such as the number of layers and neurons per layer, and how it affects the performance of s-PINNs. In the following example, we carry out a sensitivity analysis.

Example 7 (Sensitivity analysis of s-PINN). To explore how the performance of an s-PINN depends on algorithmic set-up and parameters, we apply it to solving the heat equation defined on $x \in \mathbb{R}$,

$$\partial_t u(x,t) = \partial_x^2 u(x,t) + f(x,t), \quad u(x,0) = e^{-x^2/4}\sin x \tag{26}$$

using generalized Hermite functions as basis functions. For the source $f(x,t) = [x\cos x + (t+1)\sin x](t+1)^{-3/2}\exp[-\frac{x^2}{4(t+1)}]$, equation (26) admits the analytical solution

$$u(x,t) = \frac{\sin x}{\sqrt{t+1}}\exp\left[-\frac{x^2}{4(t+1)}\right]. \tag{27}$$

Table 3. Example 7: Sensitivity analysis of s-PINN. Computational runtime (in seconds), error, and the final scaling factor for different timesteps Δt, different implicit order-K Runge–Kutta schemes, and the traditional Crank-Nicolson scheme. In each box, the runtime (in seconds) and the SSE are listed, with the final scaling factor given just below. The results associated with the smallest error are highlighted in italic while the results associated with the shortest run time for our s-PINN method are indicated in bold.

Δt \ K	C-N scheme	2	4	6	10
0.02	12, 8.252×10^{-6}, 0.545	27, 4.011×10^{-8}, 0.545	*54, 1.368* $\times$ *10*$^{-8}$, *0.545*	279, 2.545×10^{-7}, 0.545	7071, 6.358×10^{-5}, 0.695
0.05	5, 5.157×10^{-5}, 0.545	12, 2.799×10^{-8}, 0.545	23, 1.651×10^{-8}, 0.545	105, 2.566×10^{-7}, 0.545	3172, 1.052×10^{-6}, 0.545
0.1	3, 2.239×10^{-4}, 0.695	6, 1.331×10^{-6}, 0.695	10, 1.314×10^{-6}, 0.695	72, 1.346×10^{-6}, 0.695	1788, 2.782×10^{-6}, 0.695
0.2	2, 9.308×10^{-4}, 0.695	**3, 3.760** $\times$ **10**$^{-6}$, **0.695**	9, 2.087×10^{-6}, 0.695	317, 2.107×10^{-6}, 0.695	1310, 1.925×10^{-3}, 0.753

We solve equation (26) in the weak form by multiplying any test function $v \in H^1(\mathbb{R})$ on both sides and integrating by parts to obtain

$$(\partial_t u, v) = -(\partial_x u, \partial_x v) + (f, v), \ \forall v \in H^1(\mathbb{R}). \tag{28}$$

The solution diffusively spreads over time, requiring one to decrease the scaling factor β of the generalized Hermite functions $\{\hat{\mathcal{H}}_i^{\beta}(x)\}$. We shall first study how the timestep and the order of the implicit Runge–Kutta method associated with solving the minimization problem (10) affect our results. We use a neural network with five intermediate layers and 200 neurons per layer, and set the learning rate $\eta = 5 \times 10^{-4}$. The initial scaling factor is set to $\beta = 0.8$. The scaling factor adjustment ratio and threshold are set to $q = 0.98$, and $\nu = q^{-1}$, respectively. For comparison, we also apply a Crank-Nicolson scheme for numerically solving equation (28), i.e.

$$\frac{U_N^{\beta}(t_{j+1}) - U_N^{\beta}(t_j)}{\Delta t} = D_N^{\beta} \frac{\left[U_N^{\beta}(t_{j+1}) + U_N^{\beta}(t_j)\right]}{2} + \frac{F_N^{\beta}(t_{j+1}) + F_N^{\beta}(t_j)}{2} \tag{29}$$

where $U_N^{\beta}(t), F_N^{\beta}(t)$ are the $N+1$-dimensional vectors of spectral expansion coefficients of the numerical solution and of the source, respectively. $D_N^{\beta} \in \mathbb{R}^{(N+1)\times(N+1)}$ is the tridiagonal block matrix representing the discretized Laplacian operator ∂_x^2:

$$D_{i,i-2} = \beta^2 \frac{\sqrt{(i-2)(i-1)}}{2}, \ D_{i,i} = -\beta^2 \left(i - \frac{1}{2}\right), \ D_{i,i+2} = \beta^2 \frac{\sqrt{i(i+1)}}{2},$$

and $D_{i,j} = 0$, otherwise.

Table 3 shows that since the error from temporal discretization Δt^{2K} is already quite small for $K \geqslant 4$, using a higher-order Runge–Kutta method does not significantly improve accuracy for all choices of Δt. Using higher-order ($K \geqslant 4$) schemes tends to require longer run times. Higher orders require fitting over more data points (using the same number of parameters) leading to slower convergence when minimizing equation (10), which can result in larger errors. Compared to the second-order Crank-Nicolson scheme, whose error is $O(\Delta t^2)$, the errors of our s-PINN method do not grow significantly when Δt increases. In fact, the accuracy using the smallest timestep $\Delta t = 0.02$ in the Crank-Nicolson scheme was still inferior to that of the s-PINN method using the second order or fourth order Runge-Kutta scheme with $\Delta t = 0.2$. Moreover, the run time of our s-PINN method using a second or fourth-order implicit Runge–Kutta scheme for the loss function is not significantly larger than that of the Crank-Nicolson scheme. Thus, compared to traditional spectral methods for numerically solving PDEs, our s-PINN method, even when incorporating some lower-order Runge–Kutta schemes, can greatly improve accuracy without significantly increasing computational cost.

In table 3, the smallest run time of our s-PINN method, which occurs for $K = 2, \Delta t = 0.2$, is shown in bold. The smallest error case, which arises for $K = 4, \Delta t = 0.02$, is shown in italic. The run time always increases with the order K of the implicit Runge–Kutta scheme and always decreases with Δt due to fewer timesteps. Additionally, the error always increases with Δt regardless of the order of the Runge–Kutta scheme. However, the expected convergence order is not observed, implying that the increase in error results from increased lag in adjustment of the scaling factor β when Δt is too large, rather than from an insufficiently small time discretization error Δt^{2K}. Using a fourth-order implicit Runge–Kutta scheme with $\Delta t = 0.05$ to solve equation (28) seems to both achieve high accuracy and avoid large computational costs.

Table 4. Example 7: Sensitivity analysis of our s-PINN for different numbers of intermediate layers N_H and neurons per layer H. The first line gives the total computational runtime (seconds) and the runtime per epoch (in parentheses), while the second line lists the SSE (equation (10)) and the final scaling factor. Results associated with the smallest error are marked in italic while those associated with the shortest run time are highlighted in bold.

H \ N_H	3	5	8	13
50	1348 (0.0014) , 6.317×10^{-4}, 0.738	798 (0.0015), 9.984×10^{-5}, 0.695	995 (0.0020), 1.891×10^{-4}, 0.579	778 (0.0039), 4.022×10^{-4}, 0.695
80	784 (0.0015), 7.164×10^{-4}, 0.654	234 (0.0016), 1.349×10^{-6}, 0.695	216 (0.0023), 1.345×10^{-6}, 0.695	376 (0.0043), 1.982×10^{-6}, 0.695
100	1080 (0.0018), 8.804×10^{-5}, 0.695	*114 (0.0017), 1.344×10^{-6}, 0.695*	102 (0.0024), 1.346×10^{-6}, 0.695	145 (0.0043), 1.348×10^{-6}, 0.695
200	219 (0.0022), 1.349×10^{-6}, 0.695	72 (0.0035), 1.346×10^{-6}, 0.695	**43 (0.0048), 1.347×10^{-6}, 0.695**	64 (0.0057), 1.345×10^{-6}, 0.695

We also investigate how the total number of parameters in the neural network and the structure of the network affect efficiency and accuracy. We use a sixth-order implicit Runge–Kutta scheme with $\Delta t = 0.1$. The learning rate is set to $\eta = 5 \times 10^{-4}$ for all neural networks.

As shown in table 4, the computational cost tends to decrease with the number of neurons H in each layer as it takes fewer epochs to converge when minimizing equation (10). The run time tends to decrease with N_H due to a faster convergence rate, until about $N_H = 8$. The errors when $H = 50$ are significantly larger as the training terminates (after a maximum of 100 000 epochs) before it converges. For $N_H = 3$, the corresponding s-PINN always fails to achieve accuracy within 100 000 epochs unless $H \gtrsim 200$. Actually, the mean run time for training one epoch increases with H, N_H. Nonetheless, a neural network with 8 intermediate layers and 200 neurons in each layer performs the best with the smallest total run time. Therefore, overparametrization is indeed helpful in improving the neural network's performance, leading to faster convergence rates, in contrast to most traditional optimization methods that take longer to converge with more parameters. Similar observations have been made in other optimization tasks that involve deep neural networks [49, 50]. Consequently, our s-PINN method retains the advantages of deep and wide neural networks for improving accuracy and efficiency.

4. Parameter inference and source reconstruction

As with standard PINN approaches, s-PINNs can also be used for parameter inference in PDE models or reconstructing unknown sources in a physical model. Assuming observational data at uniform time intervals $t_j = j\Delta t$ associated with a partially known underlying PDE model, s-PINNs can be trained to infer model parameters θ by minimizing the sum of squared errors, weighted from both ends of the time interval (t_j, t_{j+1}),

$$\mathrm{SSE}_j = \mathrm{SSE}_j^{\mathrm{L}} + \mathrm{SSE}_j^{\mathrm{R}}, \tag{30}$$

where

$$\begin{aligned}
\mathrm{SSE}_j^{\mathrm{L}} &= \sum_{s=1}^{K} \left\| u(x, t_j + c_s\Delta t; \theta_{j+1}; \Theta_{j+1}) - u(x, t_j; \theta_j) - \sum_{r=1}^{K} a_{sr} \mathcal{M}\left[u(x, t_j + c_r\Delta t; \theta_{j+1}; \Theta_{j+1})\right] \right\|_2^2, \\
\mathrm{SSE}_j^{\mathrm{R}} &= \sum_{s=1}^{K} \left\| u(x, t_j + c_s\Delta t; \theta_{j+1}; \Theta_{j+1}) - u(x, t_{j+1}; \theta_{j+1}) - \sum_{r=1}^{K} (a_{sr} - b_r) \mathcal{M}\left[u(x, t_j + c_r\Delta t; \theta_{j+1}; \Theta_{j+1})\right] \right\|_2^2.
\end{aligned} \tag{31}$$

Here, θ_{j+1} are the set of model parameters to be found using the sample points $c_s\Delta t$ between t_j and t_{j+1}. The most obvious advantage of s-PINNs over standard PINN methods is that they can deal with models defined on unbounded domains, extending PINN-based methods that are typically applied to finite domains. Note that the revised loss function equation (30) differs from equation (10) because now the solutions at t_j and t_{j+1} are both known, while for equation (10) the solution at t_{j+1} is to be solved.

Given observations over a certain time interval, one may wish to both infer parameters θ_j in the underlying physical model and reconstruct the solution u at any given time. Here, we provide an example in which both a parameter and the numerical solution of a model are to be inferred.

Example 8 (Parameter (diffusivity) inference). As a starting point for a parameter-inference problem, we consider diffusion with a source defined on $x \in \mathbb{R}$

$$\partial_t u(x,t) = \kappa \partial_x^2 u(x,t) + f(x,t), \quad u(x,0) = e^{-x^2/4}\sin x, \tag{32}$$

where the constant parameter κ is the thermal conductivity (or diffusion coefficient) in the entire domain. In this example, we set $\kappa = 2$ as a reference and assume the source

$$f(x,t) = \left[\frac{2\,(x\cos x + (t+1)\sin x)}{(t+1)^{3/2}} - \frac{x^2}{4(t+1)^2} + \frac{\sin x}{2(t+1)^{3/2}}\right]\exp\left[-\frac{x^2}{4(t+1)}\right]. \tag{33}$$

In this case, the analytical solution to equation (32) is given by equation (27). We numerically solve equation (32) in the weak form of equation (28). If the form of the spatiotemporal heat equation is known (such as equation 32), but some parameters such as κ is unknown, reconstructing it from measurements is usually performed by defining and minimizing a loss function as was done in [51]. It can also be shown that $\kappa = \kappa(t)$ in equation (32) can be uniquely determined by the observed solution $u(x,t)$ [52–54] under certain conditions. Here, however, we assume that observations are taken at discrete time points $t_j = j\Delta t$ and seek to reconstruct both the parameter κ and the numerical solution at $t_j + c_s\Delta t$ (defined in equations (31)) by minimizing equation (30). We use a neural network with 13 layers and 100 neurons per layer with a sixth-order implicit Runge–Kutta scheme. The timestep Δt is 0.1. At each timestep, we draw the function values from

$$u(x,t_j) = \frac{\sin x}{\sqrt{t_j+1}}\exp\left[-\frac{x^2}{4(t_j+1)}\right] + \xi(x,t_j), \tag{34}$$

where $\xi(x,t)$ is the noise term that is both spatially and temporally uncorrelated, and $\xi(x,t) \sim \mathcal{N}(0,\sigma^2)$, where $\mathcal{N}(0,\sigma^2)$ is the normal distribution of mean 0 and variance σ^2 (i.e. $\langle \xi(x,t)\xi(y,s)\rangle = \sigma^2\delta_{x,y}\delta_{s,t}$). For different levels of noise σ, we take one trajectory of the measured solution with noise $u(x,t_j)$ to reconstruct the parameter κ, which is presumed to be a constant in $[t_j, t_{j+1})$, and simultaneously obtain the numerical solutions at the intermediate time points $t_j + c_s\Delta t$. We are interested in how different levels of noise and the increasing spread of the solution will affect the SSE and the reconstructed parameter $\hat{\kappa}$. Figure 7 shows the deviation of the reconstructed $\hat{\kappa}$ from its true value, $|\hat{\kappa} - 2|$, the SSE, the scaling factor, and the frequency indicator as functions of time for different noise levels. Figure 7(a) shows that the larger the noise, the less accurate the reconstructed κ. Moreover, as the function becomes more spread out (when $\sigma = 0$), the error in both the reconstructed diffusivity and the SSE increases across time, as shown in figure 7(b). This behavior suggests that a diffusive solution that decays more slowly at infinity can give rise to inaccuracies in the numerical computation of the intermediate timestep solutions and in reconstructing model parameters. Finally, as indicated in figures 7(c), (d), larger variances in the noise will impede the scaling process since the frequency indicator cannot be as easily controlled because larger variances in the noise usually corresponds to high-frequency and oscillatory components of a solution.

In example 8, both the parameter and the unknown solution were inferred. Apart from reconstructing the coefficients in a given physical model, in certain applications, we may also wish to reconstruct the underlying physical model by inferring, e.g. the heat source $f(x,t)$. Source recovery from observational data commonly arises and has been the subject of many previous studies [55–57]. We now discuss how the s-PINN methods presented here can also be used for this purpose. For example, in equation (26) or equation (32), we may wish to reconstruct an unknown source $f(x,t)$ by also approximating it with a spectral decomposition

$$f(x,t) \approx f_N(x,t) = \sum_{i=0}^{N} h_i(t)\phi^{\beta}_{i,x_L}(x), \tag{35}$$

and minimizing an SSE that is augmented by a penalty on the coefficients $h_i, i = 0,\ldots,N$.

We learn the expansion coefficients h_i within $[t_j, t_{j+1}]$ by minimizing

$$\begin{aligned}
\mathrm{SSE}_j =& \mathrm{SSE}_j^{\mathrm{L}} + \mathrm{SSE}_j^{\mathrm{R}} + \lambda\sum_{s=1}^{K}\left\|\mathbf{h}_N(t_j + c_s\Delta;\Theta_{j+1})\right\|_2^2, \quad \lambda \geqslant 0,\\
\mathrm{SSE}_j^{\mathrm{L}} =& \sum_{s=1}^{K}\left\|u(x,t_j+c_s\Delta t) - u(x,t_j) - \sum_{r=1}^{K} a_{sr}\left[\partial_{xx}u(x,t_j+c_r\Delta t) + f_N(x,t_j+c_r\Delta t;\Theta_{j+1})\right]\right\|_2^2,\\
\mathrm{SSE}_j^{\mathrm{R}} =& \sum_{s=1}^{K}\left\|u(x,t_j+c_s\Delta t) - u(x,t_{j+1}) - \sum_{r=1}^{K}(a_{sr}-b_r)\left[\partial_{xx}u(x,t_j+c_r\Delta t) + f_N(x,t_j+c_r\Delta t;\Theta_{j+1})\right]\right\|_2^2,
\end{aligned} \tag{36}$$

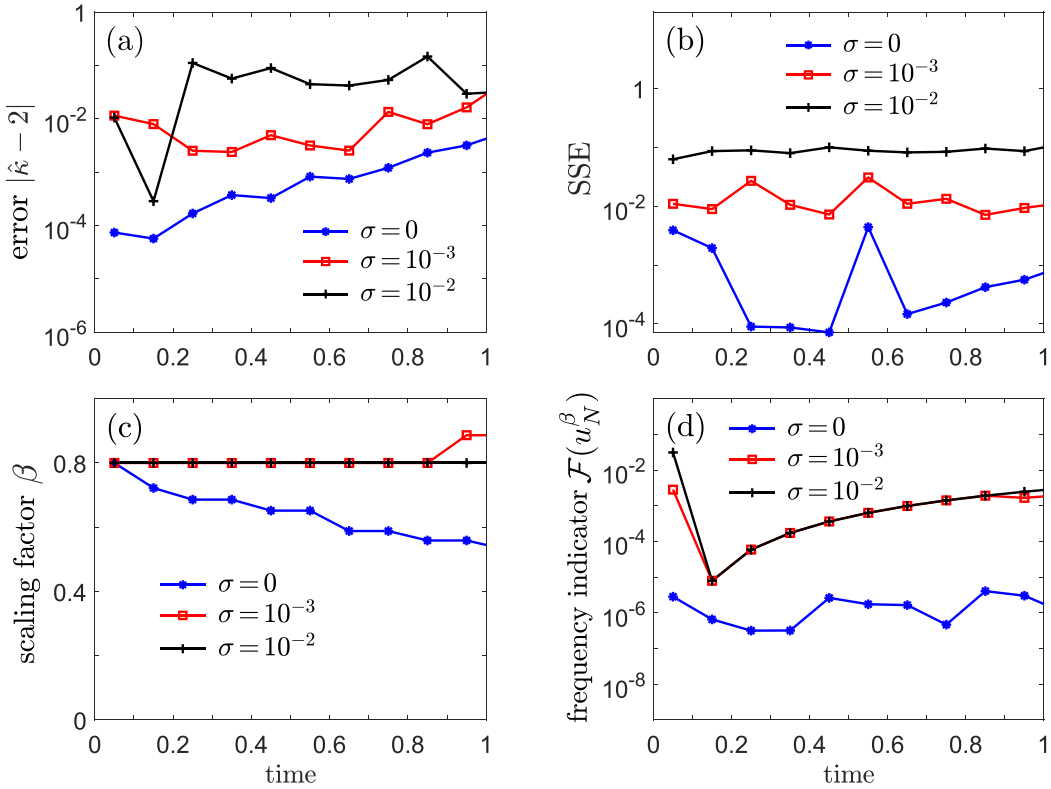

Figure 7. Example 8: parameter (diffusivity) inference. The parameter κ inferred within successive time windows of $\Delta t = 0.1$, the SSE error equation (30), the scaling factor, and the frequency indicators associated with solving equation (32), for different noise levels σ. Here, the SSE was minimized to find the estimate $\hat{\theta} \equiv \hat{\kappa}$ and the solutions u_N at intermediate timesteps $t_j + c_s\Delta t$. (a), (b) Smaller σ leads to smaller SSE equation (31) and a more accurate reconstruction of $\hat{\kappa}$. When the function has spread out significantly at long times, the reconstructed $\hat{\kappa}$ becomes less accurate, suggesting that unboundedness and small function values render the problem susceptible to numerical difficulties. (c), (d) Noisy data results in a larger proportion of high-frequency waves and thus a large frequency indicator, impeding proper scaling.

where $\mathbf{h}_N(t_j + c_s\Delta t; \Theta_{j+1}) \equiv (h_1(t_j + c_s\Delta t; \Theta_{j+1}), \ldots, h_N(t_j + c_s\Delta t; \Theta_{j+1}))$ and u (or the spectral expansion coefficients w_i of u) is assumed known at all intermediate time points $c_s\Delta t$ in (t_j, t_{j+1}).

The last term in equation (36) adds an L^2 penalty term on the coefficients of f which tends to reconstruct smoother and smaller-magnitude sources as λ is increased. Other forms of regularization such as L^1 can also be considered [58]. In the presence of noise, an L^1 regularization further drives small expansion weights to zero, yielding an inferred source f_N described by fewer nonzero weights.

Since the reconstructed heat source f_N is expressed in terms of a spectral expansion in equation (35), and minimizing the loss function equation (36) depends on the global information of the observation u, f at any location x also contains global information intrinsic to u. In other words, for such inverse problems, the s-PINN approach extracts global spatial information and is thus able to reconstruct global quantities. We consider an explicit case in the next example.

Example 9 (Source recovery). Consider the canonical source reconstruction problem [59–61] of finding $f(x,t)$ in the heat equation model in equation (26) for which observational data are given by equation (34) but evaluated at $t_j + c_s\Delta t$. A physical interpretation of the reconstruction problem is identifying the heat source $f(x,t)$ using measurement data in conjunction with equation (26). As in example 5, we numerically solve the weak form equation (28). To study how the L^2 penalty term in equation (36) affects source recovery and whether increasing the regularization λ will make the inference of f more robust against noise, we minimize equation (30) for different values of λ and σ.

We use a neural network with 13 layers and 100 neurons per layer to reconstruct $f_i(t)$ in the decomposition equation (35) with $N = 16$, i.e. the neural network outputs the coefficients h_i at the intermediate timesteps $t_j + c_s\Delta t$. The basis functions $\phi^\beta_{i,x_L}(x)$ are chosen to be Hermite functions $\hat{\mathcal{H}}^\beta_{i,x_L}(x)$. For simplicity, we consider the problem within only the first time interval $[0, 0.2]$ and a fixed scaling factor $\beta = 0.8$ as well as a fixed displacement $x_L = 0$.

Table 5. The error SSE_0 from equation (31) and the error of the reconstructed source equation (37), (in parentheses), under different strengths of data noise and regularization coefficients λ.

σ \ λ	0	10^{-3}	10^{-2}	10^{-1}
0	0.1370, (1.543×10^{-8})	0.1370, (1.368×10^{-5})	0.1477, (0.00132)	0.3228, (0.0888)
10^{-3}	0.1821, (2.837×10^{-6})	0.1818, (2.736×10^{-5})	0.1702, (1.387×10^{-3})	0.3222, (0.08964)
10^{-2}	1.0497, (0.001517)	1.0383 (1.579×10^{-3})	0.8031, (6.078×10^{-3})	0.3434, (0.1168)
10^{-1}	11.505, (0.2976)	11.458, (0.3032)	8.2961, (0.6905)	1.3018, (2.9330)

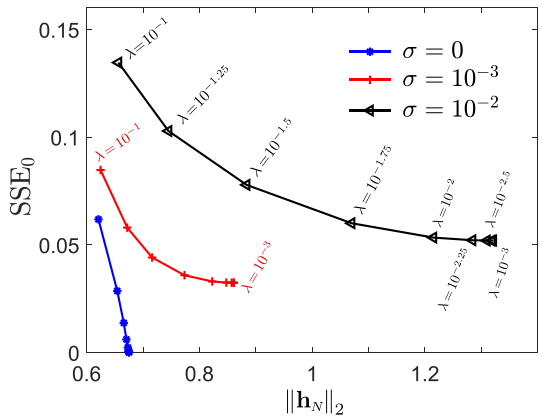

Figure 8. Example 9: source recovery. SSE_0 plotted against the reconstructed heat source $\|\mathbf{h}_N\|_2$ as given by equation (36), as a function of λ for various values of σ (an 'L-curve'). When λ is large, the norm of the reconstructed heat source $\|\mathbf{h}_N\|_2$ always tends to decrease while the 'error' SSE_0 tends to increase. When $\lambda = 10^{-1}$, $\|\mathbf{h}_N\|_2$ is small and the SSE_0 is large. A moderate $\lambda \in [10^{-2}, 10^{-3}]$ could reduce the error SSE_0, compared to using a large λ, while also generating a heat source with smaller $\|\mathbf{h}_N\|_2$.

In table 5, we record the L^2 error

$$\left\| f(x,t) - \sum_{i=0}^{16} h_i(t;\Theta)\hat{\mathcal{H}}^{\beta}_{i,x_L}(x) \right\|_2 \tag{37}$$

the lower-left of each entry and the SSE_0 in the upper-right. Observe that as the variance of the noise increases, the reconstruction of f via the spectral expansion becomes increasingly inaccurate. In the noise-free case, taking $\lambda = 0$ in equation (36) achieves the smallest SSE_0 and the smallest reconstruction error. However, with increasing noise σ, using an L^2 regularization term in equations (36) can prevent over-fitting of the data although SSE_0 increases with the regularization strength λ. When $\sigma = 10^{-3}$, taking $\lambda = 10^{-2}$ achieves the smallest reconstruction error equation (37); when $\sigma = 10^{-2}, 10^{-1}$, $\lambda = 10^{-1}$ achieves the smallest reconstruction error. However, if λ is too large, coefficients of the spectral approximation to f are pushed to zero. Thus, it is important to choose an intermediate λ so that the reconstruction of the source is robust to noise. In figure 8, we plot the norm of the reconstructed heat source $\|\mathbf{h}_N\|_2$ and the 'error' SSE_0 which varies as λ changes for different σ.

5. Summary and conclusions

In this paper, we propose an approach that blends standard PINN algorithms with adaptive spectral methods and show through examples that this hybrid approach can be applied to a wide variety of data-driven problems including function approximation, solving PDEs, parameter inference, and model selection. The underlying feature that we exploit is the physical differences across classes of data. For example, by understanding the difference between space and time variables in a PDE model, we can describe the spatial dependence in terms of basis functions, obviating the need to normalize spatial data. Thus, s-PINNs are ideal for solving problems in unbounded domains. The only additional 'prior' needed is an assumption on the asymptotic spatial behavior and an appropriate choice of basis functions. Additionally, adaptive techniques have been recently developed to further improve the efficiency and accuracy, making spectral decomposition especially suitable for unbounded-domain problems that the standard PINN cannot easily address.

Table 6. Advantages and disadvantages of traditional and PINN-based numerical solvers. This table provides an overview of the advantages ('+') and disadvantages ('–') associated with different methods and solvers. Finite difference (FD), finite-element (FE), and spectral methods can be used in a traditional sense without relying on neural networks.

Methods \ Solvers	Traditional	PINN
Non-spectral	+ Leverages existing numerical methods + Low-order FD/FE schemes easily implemented + Efficient evaluation of function and derivatives – Mainly restricted to bounded domains – Complicated time-extrapolation – Complicated implementation of higher-order schemes – Algebraic convergence, less accurate – More complicated inverse-type problems – More complicated temporal and spatial extrapolation – Requires understanding of problem to choose suitable discretization	+ Easy implementation + Efficient deep-neural-network training + Easy extrapolation + Easily handles inverse-type problems – Mainly restricted to bounded domains – Less accurate – Less interpretable spatial derivatives – Limited control of spatial discretization – Expensive evaluation of neural networks – Incompatible with existing numerical methods
Spectral	+ Suitable for bounded and unbounded domains + Spectral convergence in space, more accurate + Leverage existing numerical methods + Efficient evaluation of function and derivatives – Information required for choosing basis functions – More complicated inverse-type problems – More complicated implementation – More complicated temporal extrapolation in time – Usually requires a 'regular' domain e.g. rectangle, $\mathbb{R}^d$, a ball, etc	+ Suitable for both bounded and unbounded domains + Easy implementation + Spectral convergence in space, more accurate + Efficient deep-neural-network training + More interpretable derivatives of spatial variables + Easy extrapolation + Easily handles inverse-type problems + Compatible with existing adaptive techniques – Requires some information to choose basis functions – Expensive evaluation of neural networks – Usually requires a 'regular' domain

We applied s-PINNs (exploiting adaptive spectral methods) across a number of examples and showed that they can outperform simple feedforward neural networks for function approximation and existing PINNs for solving certain PDEs. Three major advantages are that s-PINNs can be applied to unbounded domain problems, more accurate by recovering spectral convergence in space, and more efficient as a result of faster evaluation of spatial derivatives of all orders compared to standard PINNs that use autodifferentiation. These advantages are rooted in separated data structures, allowing for spectral computation and high-accuracy numerics. Straightforward implementation of s-PINNs retains most of the advantageous features of deep PINN architectures, making s-PINNs ideal for data-driven inference problems. However, in the context of solving higher-dimensional PDEs, a tradeoff is necessary when using s-PINNs instead of PINNs. For s-PINNs, the network structure needs to be significantly widened to output an exponentially increasing (with dimensionality) number of expansion coefficients, while in standard PINNs, the network structure remains largely preserved but an exponentially larger number of trajectories are needed for sufficient training. We found that by restricting the spatial domain to a hyperbolic cross space, the number of outputs required for s-PINNs can be appreciably decreased for problems of moderate dimensions. While using a hyperbolic cross space cannot reduce the number of outputs sufficiently to allow s-PINNs to be effective for very high-dimensional problems, the standard PINNs approach to problems in very high dimensions could require an unattainable number of samples for sufficient training.

In table 6, we compare the advantages and disadvantages of the standard PINN and s-PINN methods. Potential improvements and extensions include applying techniques for selecting basis functions that best characterize the expected underlying process and inferring forms of the underlying model PDEs [62, 63]. While standard PINN methods deal with local information (e.g. $\partial_x u, \partial_x^2 u$), spectral decompositions capture global information making them a natural choice for also efficiently learning and approximating nonlocal terms such as convolutions and integral kernels. Potential future extensions of our s-PINN method may include adapting it to solve higher-dimensional problems by more systematically choosing a proper hyperbolic space or using other coefficient-reducing techniques, as well as using wavelets as activation functions [64] to solve nonlinear differential equations. Also, recent Gaussian–process–based smoothing techniques [65] can be considered to improve robustness of our s-PINN method against noise/errors in

measurements, and noise-aware physics-informed machine learning techniques [66] can be incorporated when applying our s-PINN for inverse-type PDE discovery problems. Finally, one can incorporate a recently proposed Bayesian-PINN (B-PINN) [67] method into our s-PINN method to quantify uncertainty when solving inverse problems under noisy data.

Data availability statement

No new data were created or analysed in this study.

Acknowledgments

L B acknowledges financial support from the Swiss National Fund (Grant Number P2EZP2_191888). The authors also acknowledge support from the US Army Research Office (W911NF-18-1-0345) and the National Science Foundation (DMS-1814364).

ORCID iDs

Mingtao Xia https://orcid.org/0000-0002-2116-4712
Lucas Böttcher https://orcid.org/0000-0003-1700-1897
Tom Chou https://orcid.org/0000-0003-0785-6349

References

[1] Hornik K 1991 Approximation capabilities of multilayer feedforward networks *Neural Netw.* **4** 251–7
[2] Park S, Yun C, Lee J and Shin J 2020 Minimum width for universal approximation *Int. Conf. on Learning Representations*
[3] Raissi M, Perdikaris P and Karniadakis G E 2019 Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations *J. Comput. Phys.* **378** 686–707
[4] Karniadakis G E, Kevrekidis I G, Lu L, Perdikaris P, Wang S and Yang L 2021 Physics-informed machine learning *Nat. Rev. Phys.* **3** 422–40
[5] Asikis T, Böttcher L and Antulov-Fantulin N 2022 Neural ordinary differential equation control of dynamics on graphs *Phys. Rev. Res.* **4** 013221
[6] Böttcher L, Antulov-Fantulin N and Asikis T 2022 AI Pontryagin or how artificial neural networks learn to control dynamical systems *Nat. Commun.* **13** 333
[7] Böttcher L and Asikis T 2022 Near-optimal control of dynamical systems with neural ordinary differential equations *Mach. Learn.: Sci. Technol.* **3** 045004
[8] Lewis F W, Jagannathan S and Yesildirak A 2020 *Neural Network Control of Robot Manipulators and Non-Linear Systems* (Boca Raton, FL: CRC Press)
[9] Kukačka J, Golkov V and Cremers D 2017 Regularization for deep learning: a taxonomy (arXiv:1710.10686)
[10] Lutter M, Ritter C and Peters J 2019 Deep Lagrangian networks: Using physics as model prior for deep learning *Int. Conf. on Learning Representations* (OpenReview.net)
[11] Roehrl M A, Runkler T A, Brandtstetter V, Tokic M and Obermayer S 2020 Modeling system dynamics with physics-informed neural networks based on Lagrangian mechanics *IFAC-PapersOnLine* **53** 9195–200
[12] Desmond Zhong Y, Dey B and Chakraborty A 2019 Symplectic ODE-net: learning Hamiltonian dynamics with control *Int. Conf. on Learning Representations*
[13] Kharazmi E, Zhang Z and Karniadakis G E 2019 Variational physics-informed neural networks for solving partial differential equations (arXiv:1912.00873)
[14] Jagtap A D and Karniadakis G E 2020 Extended physics-informed neural networks (xpinns): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations *Commun. Comput. Phys.* **28** 2002–41
[15] Li Z, Zheng H, Kovachki N, Jin D, Chen H, Liu B, Azizzadenesheli K and Anandkumar A 2021 Physics-informed neural operator for learning partial differential equations (arXiv:2111.03794)
[16] Mao Z, Jagtap A D and Karniadakis G E 2020 Physics-informed neural networks for high-speed flows *Comput. Methods Appl. Mech. Eng.* **360** 112789
[17] Fang Z and Zhan J 2019 A physics-informed neural network framework for PDEs on 3D surfaces: Time independent problems *IEEE Access* **8** 26328–35
[18] Misyris G S, Venzke A and Chatzivasileiadis S 2020 Physics-informed neural networks for power systems *2020 IEEE Power & Energy Society General Meeting (PESGM)* (IEEE) pp 1–5
[19] Sahli Costabal F, Yang Y, Perdikaris P, Hurtado D E and Kuhl E 2020 Physics-informed neural networks for cardiac activation mapping *Frontiers Phys.* **8** 42
[20] Liu M, Liang L and Sun W 2020 A generic physics-informed neural network-based constitutive model for soft biological tissues *Comput. Methods Appl. Mech. Eng.* **372** 113402
[21] Thanasutives P, Numao M and Fukui K-ichi 2021 Adversarial multi-task learning enhanced physics-informed neural networks for solving partial differential equations *2021 Int. Joint Conf. on Neural Networks (IJCNN)* (IEEE) pp 1–9
[22] Penwarden M, Zhe S, Narayan A and Kirby R M 2023 A Metalearning Approach for Physics-Informed Neural Networks (PINNs): application to Parameterized PDEs *J. Comput. Phys.* **477** 111912
[23] Böttcher L and Herrmann H J 2021 *Computational Statistical Physics* (Cambridge: Cambridge University Press)
[24] Strub S H and Böttcher L 2019 Modeling deformed transmission lines for continuous strain sensing applications *Meas. Sci. Technol.* **31** 035109
[25] Barré J, Olivetti A and Yamaguchi Y Y 2011 Algebraic damping in the one-dimensional Vlasov equation *J. Phys. A: Math. Theor.* **44** 405502

[26] Li B, Zhang J and Zheng C 2018 Stability and error analysis for a second-order fast approximation of the one-dimensional Schrödinger equation under absorbing boundary conditions *SIAM J. Sci. Comput.* **40** A4083–104
[27] Xia M, Greenman C D and Chou T 2020 PDE models of adder mechanisms in cellular proliferation *SIAM J. Appl. Math.* **80** 1307–35
[28] Xia M and Chou T 2021 Kinetic theory for structured populations: application to stochastic sizer-timer models of cell proliferation *J. Phys. A: Math. Theor.* **54** 385601
[29] Mengotti E, Heyderman L J, Fraile Rodríguez A, Nolting F, Hügli R V and Braun H-B 2011 Real-space observation of emergent magnetic monopoles and associated Dirac strings in artificial Kagomé spin ice *Nat. Phys.* **7** 68–74
[30] Hügli R V, Duff G, O'Conchuir B, Mengotti E, Fraile Rodríguez A, Nolting F, Heyderman L J and Braun H B 2012 Artificial Kagomé spin ice: dimensional reduction, avalanche control and emergent magnetic monopoles *Phil. Trans. R. Soc.* A **370** 5767–82
[31] Bararnia H and Esmaeilpour M 2022 On the application of physics informed neural networks (PINN) to solve boundary layer thermal-fluid problems *Int. Commun. Heat Mass Transfer* **132** 105890
[32] Xia M, Shao S and Chou T 2021 Efficient scaling and moving techniques for spectral methods in unbounded domains *SIAM J. Sci. Comput.* **43** A3244–68
[33] Xia M, Shao S and Chou T 2021 A frequency-dependent *p*-adaptive technique for spectral methods *J. Comput. Phys.* **446** 110627
[34] Shen J, Tang T and Wang Li-L 2011 *Spectral Methods: Algorithms, Analysis and Applications* vol 41 (New York: Springer)
[35] Trefethen L N 2000 *Spectral Methods in Matlab* (Philadelphia, PA: SIAM)
[36] Linnainmaa S 1976 Taylor expansion of the accumulated rounding error *BIT Numerical Mathematics* **16** 146–60
[37] Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L and Lerer A 2017 Automatic differentiation in PyTorch (available at: www.jmlr.org/papers/v18/17-468.html)
[38] Hornik K, Stinchcombe M and White H 1989 Multilayer feedforward networks are universal approximators *Neural Netw.* **2** 359–66
[39] Burns K J, Vasil G M, Oishi J S, Lecoanet D and Brown B P 2020 Dedalus: a flexible framework for numerical simulations with spectral methods *Phys. Rev. Res.* **2** 023068
[40] Ioffe S and Szegedy C 2015 Batch normalization: accelerating deep network training by reducing internal covariate shift *Int. Conf. on Machine Learning* (PMLR) pp 448–56
[41] Tang T, Wang Li-L, Yuan H and Zhou T 2020 Rational spectral methods for PDEs involving fractional Laplacian in unbounded domains *SIAM J. Sci. Comput.* **42** A585–611
[42] Baydin A G, Pearlmutter B A, Andreyevich Radul A and Mark Siskind J 2018 Automatic differentiation in machine learning: a survey *J. Mach. Learn. Res.* **18** 1–43
[43] Lu L, Jin P, Pang G, Zhang Z and Karniadakis G E 2021 Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators *Nat. Mach. Intell.* **3** 218–29
[44] Wang S, Wang H and Perdikaris P 2021 Learning the solution operator of parametric partial differential equations with physics-informed deeponets *Sci. Adv.* **7** eabi8605
[45] Li Z, Kovachki N B, Azizzadenesheli K, Bhattacharya K, Stuart A and Anandkumar A 2020 Fourier neural operator for parametric partial differential equations *Int. Conf. on Learning Representations*
[46] Brandstetter J, Worrall D E and Welling M 2021 Message passing neural PDE solvers *Int. Conf. on Learning Representations*
[47] Shen J and Wang Li-L 2010 Sparse spectral approximations of high-dimensional problems based on hyperbolic cross *SIAM J. Numer. Anal.* **48** 1087–109
[48] Chou T, Shao S and Xia M 2023 Adaptive Hermite spectral methods in unbounded domains *Appl. Numer. Math.* **183** 201–20
[49] Arora S, Cohen N and Hazan E 2018 On the optimization of deep networks: Implicit acceleration by overparameterization *Proc. 35th Int. Conf. on Machine Learning* (*Proc. Machine Learning Research* vol 80) ed J Dy and A Krause pp 244–53
[50] Chen Z, Cao Y, Zou D and Gu Q 2020 How much over-parameterization is sufficient to learn deep ReLU networks? *Int. Conf. on Learning Representations*
[51] Huntul M J 2021 Identification of the timewise thermal conductivity in a 2D heat equation from local heat flux conditions *Inverse Problems Sci. Eng.* **29** 903–19
[52] Ivanchov N I 1993 Inverse problems for the heat-conduction equation with nonlocal boundary conditions *Ukr. Math. J.* **45** 1186–92
[53] Jones B F Jr 1962 The determination of a coefficient in a parabolic differential equation: part I. Existence and uniqueness *J. Math. Mech.* **11** 907–18
[54] Beznoshchenko N Y 1974 On finding a coefficient in a parabolic equation *Differ. Equ.* **10** 24–35
[55] Yan L, Yang F-L and Fu C-Li 2009 A meshless method for solving an inverse spacewise-dependent heat source problem *J. Comput. Phys.* **228** 123–36
[56] Yang L, Dehghan M, Yu J-N and Luo G-W 2011 Inverse problem of time-dependent heat sources numerical reconstruction *Math. Comput. Simul.* **81** 1656–72
[57] Yang F and Fu C-Li 2010 A simplified Tikhonov regularization method for determining the heat source *Appl. Math. Model.* **34** 3286–99
[58] Wu T and Tegmark M 2019 Toward an artificial intelligence physicist for unsupervised learning *Phys. Rev.* E **100** 033311
[59] Rozier Cannon J 1968 Determination of an unknown heat source from overspecified boundary data *SIAM J. Numer. Anal.* **5** 275–86
[60] Tomas Johansson B and Lesnic D 2007 A variational method for identifying a spacewise-dependent heat source *IMA J. Appl. Math.* **72** 748–60
[61] Hasanov A and Pektaş B 2014 A unified approach to identifying an unknown spacewise dependent source in a variable coefficient parabolic equation from final and integral overdeterminations *Appl. Numer. Math.* **78** 49–67
[62] Long Z, Lu Y, Ma X and Dong B 2018 PDE-net: Learning PDEs from data *Int. Conf. on Machine Learning* (PMLR) pp 3208–16
[63] Raissi M 2018 Deep hidden physics models: deep learning of nonlinear partial differential equations *J. Mach. Learn. Res.* **19** 932–55
[64] Uddin Z, Ganga S, Asthana R and Ibrahim W 2023 Wavelets based physics informed neural networks to solve non-linear differential equations *Sci. Rep.* **13** 1–19
[65] Bajaj C, McLennan L, Andeen T and Roy A 2023 Recipes for when physics fails: recovering robust learning of physics informed neural networks *Mach. Learn.: Sci. Technol.* **4** 015013
[66] Thanasutives P, Morita T, Numao M and Fukui K-ichi 2023 Noise-aware physics-informed machine learning for robust PDE discovery *Mach. Learn.: Sci. Technol.* **4** 015009
[67] Yang L, Meng X and Karniadakis G E 2021 B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data *J. Comput. Phys.* **425** 109913