Journal of Computational Physics 462 (2022) 111260

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A-PINN: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations

Lei Yuan [a,b], Yi-Qing Ni [a,b,*], Xiang-Yun Deng [a,b], Shuo Hao [a,b]

[a] Department of Civil and Environmental Engineering, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong Special Administrative Region

[b] Hong Kong Branch of the National Engineering Research Center on Rail Transit Electrification and Automation, Hung Hom, Kowloon, Hong Kong Special Administrative Region

ARTICLE INFO

Article history:
Received 6 December 2021
Received in revised form 23 April 2022
Accepted 24 April 2022
Available online 29 April 2022

Keywords:
Physics informed neural network (PINN)
Auxiliary physics informed neural network (A-PINN)
Integro-differential equations (IDEs)
Deep learning
Multi-output neural network

ABSTRACT

Physics informed neural networks (PINNs) are a novel deep learning paradigm primed for solving forward and inverse problems of nonlinear partial differential equations (PDEs). By embedding physical information delineated by PDEs in feedforward neural networks, PINNs are trained as surrogate models for approximate solution to the PDEs without need of label data. Due to the excellent capability of neural networks in describing complex relationships, a variety of PINN-based methods have been developed to solve different kinds of problems such as integer-order PDEs, fractional PDEs, stochastic PDEs and integro-differential equations (IDEs). However, for the state-of-the-art PINN methods in application to IDEs, integral discretization is a key prerequisite in order that IDEs can be transformed into ordinary differential equations (ODEs). However, integral discretization inevitably introduces discretization error and truncation error to the solution. In this study, we propose an auxiliary physics informed neural network (A-PINN) framework for solving forward and inverse problems of nonlinear IDEs. By defining auxiliary output variable(s) to represent the integral(s) in the governing equation and employing automatic differentiation of the auxiliary output to replace integral operator, the proposed A-PINN bypasses the limitation of integral discretization. Distinct from the neural network in the original PINN which only approximates the variables in the governing equation, in the proposed A-PINN framework, a multi-output neural network is constructed to simultaneously calculate the primary outputs and auxiliary outputs which respectively approximate the variables and integrals in the governing equation. Subsequently, the relationship between the primary outputs and auxiliary outputs is constrained by new output conditions in compliance with physical laws. By pursuing the first-order nonlinear Volterra IDE benchmark problem, we validate that the proposed A-PINN can obtain more accurate solution than the conventional PINN. We further demonstrate the good performance of A-PINN in solving the forward problems involving nonlinear Volterra IDEs system, nonlinear 2-dimensional Volterra IDE, nonlinear 10-dimensional Volterra IDE, and nonlinear Fredholm IDE. Finally, the A-PINN framework is implemented to solve the inverse problem of nonlinear IDEs and the results show that the unknown parameters can be satisfactorily discovered even with heavily noisy data.

* Corresponding author at: Department of Civil and Environmental Engineering, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong Special Administrative Region.

E-mail address: ceyqni@polyu.edu.hk (Y.-Q. Ni).

https://doi.org/10.1016/j.jcp.2022.111260

1. Introduction

With the explosive growth of computing resources over the past decade, deep learning, especially deep neural networks (DNNs), has undergone revolutionary development. In recent years, a novel DNN framework named physics informed neural networks (PINNs) [1,2] has been developed, which successfully integrates the underlying physical information described by partial differential equations (PDEs) with neural networks. Making the full use of physical information as prior knowledge, PINNs can be trained with few or even no label data as surrogate models for accurate solution of PDEs. In the PINN framework, fully connected feedforward neural networks are employed as the core of the surrogate models to predict the outputs in the PDE domain, and automatic differentiation of the neural networks is utilized to calculate the differential operator in the governing PDEs. The automatic differentiation [3], a continuous grid-free differentiation manipulation procedure, can calculate the derivative analytically by stepping the backward chain in the neural networks. It is completely different from traditional numerical methods such as finite difference since it doesn't involve discretization and approximation that bring discretization and truncation errors to the solution. By employing an optimizer to minimize the sum of residuals of the initial conditions, boundary conditions, governing equations, and measurement data of the PDEs, PINNs are trained to accurately approximate the exact solutions. Due to the excellent capability of neural networks in describing complex relationship between inputs and outputs, PINNs create a new path to solve the forward and inverse problems involving nonlinear PDEs [1]. Successful applications of PINNs have been made which involve diverse physical, engineering and medical areas such as fluid mechanics [4–10], medical diagnosis [11,12], heat transfer analysis [13–15], materials science [10,16–19], and engineering mechanics [20–24].

For surrogate models, the accurate expression of exact solutions and the adaptability to complex problems are two key indicators in regard to the model performance. A great deal of research efforts has been made to improve the adaptability and accuracy of PINN-enabled models. With the intent to enhance the adaptability of PINNs in handling irregular domains, cPINN [25], PPINN [26], and XPINN [27] have been proposed by decomposing the equation domain into multiple subdomains and pursuing parallel computing. In the original PINN, the boundary conditions are imposed by 'soft' penalty functions; better embedding methods of the essential boundary conditions [27–29] have recently been proposed which are proven to reduce the error on boundary conditions effectively.

Other efforts have been devoted to enhancing the solution accuracy and convergence efficiency through synthesis with numerical methods [30], defining problem-specific activation functions [31,32], configuring special neural network frameworks [33–35], optimizing the sampling of collocation points [9,35,36], and compromising residuals with adaptive weights [30,35,37–43]. The quantification of uncertainty in the solution of PINNs has also been addressed by considering different origins of uncertainty: (i) uncertainty in the equations [44–46], (ii) uncertainty in the data [47–50], and (iii) uncertainty in the neural network models [45].

Working in synergy with traditional numerical methods or other machine learning methods, the state-of-the-art PINN-based methods can be extended to address the problems of fractional differential equations, stochastic differential equations, and integro-differential equations (IDEs). In [44], a new class of physics-informed generative adversarial networks (PI-GANs) was put forth to solve forward, inverse, and mixed stochastic problems. In [51], the spectrally dynamically orthogonal (DO) and borthogonal (BO) constraints were introduced into the loss function to develop neural network (NN)-DO/BO methods for solving time-dependent stochastic PDEs. In [45], the arbitrary polynomial chaos (aPC) was introduced to develop the so-called NN-aPC for solving stochastic PDEs, where DNNs were used to learn each individual mode of the aPC expansion. In [52], normalization of field flows for learning random fields from scattered measurements was proposed to solve data-driven forward, inverse, and mixed stochastic PDEs. For fractional PDEs, standard numerical methods such as finite difference have been used to discretize the fractional operators, thereby bypassing the difficulty stemming from the fact that automatic differentiation is not applicable to fractional operators [37]. To reduce the computational cost of PINN methods, a novel sampling-based Monte Carlo physics informed neural network (MC-PINN) has been proposed and successfully applied to solve high-dimensional fractional PDEs problems in [53].

The IDEs, that involve both partial differential and integral operators, are widely used in a variety of disciplines of science and engineering, such as economic mathematics [54,55], dynamics of nanobeams [56], population growth [57], glass forming [58], and renewable energy [59]. For IDEs, in particular nonlinear IDEs, it is usually difficult to solve analytically; therefore, numerical methods such as finite element and finite difference are often used to obtain the approximate solutions of IDEs. However, since neural networks are convenient for calculating automatic differentiation but difficult to calculate the integral operator, only a few attempts have been made to solve nonlinear IDEs by the PINN method. Specifically, in [37] a discretization integration scheme working with numerical methods such as finite element, finite difference, and spectral methods was proposed to solve the IDE problem. This method is schematically illustrated in Fig. 1. In [60], the discretization scheme shown in Fig. 1 was applied to solve the first-order Volterra IDE, where Gauss-Legendre quadrature was adopted to discretize the integral operator and transform this IDE into an ODE problem. However, in this discretization scheme, integral discretization is prerequisite yet it brings two shortcomings: (i) with the approximation of integral by numerical methods such as Gaussian Legendre quadrature, it is inevitable to generate discretization error to the solution; (ii) a high-order discretization is usually required in order to achieve satisfactory accuracy. For example, in [60], the 20th-order Gaussian Legendre quadrature was employed. Since the integral operator at each discrete point needs to be calculated with high-order numerical methods, the computational cost for each iteration will increase significantly as the discrete points and the order of discretization increase, diminishing the training efficiency of PINNs.

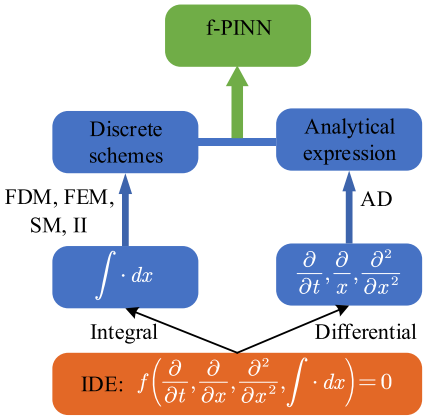

Fig. 1. Discretization scheme to solve integro-differential equations (IDEs) [37]. 'FDM' – finite difference method; 'FEM' – finite element method; 'SM' – spectral method; 'AD' – automatic differentiation.

In this paper, we propose an auxiliary physics informed neural network (A-PINN) framework to solve the forward and inverse problems of IDEs. A-PINN is a kind of DNNs with the same intended target as PINN, which will be trained as surrogate models approximating the exact solutions of IDEs. Specifically, in the A-PINN framework, the DNN is configured to calculate not only the primary outputs which approximate the primary variables, but also the auxiliary outputs which approximate the integral(s) in the governing equation. And the automatic differentiation of the auxiliary outputs is utilized to replace the integral operation involved in IDEs. The relationship between the primary outputs and auxiliary outputs is imposed by embedding additional output conditions in compliance with physical laws. Compared with the numerical discretization scheme adopted in the current PINN paradigm as shown in Fig. 1, the A-PINN framework affords several appealing features: (i) By replacing the integral operation with automatic differentiation of the auxiliary outputs, A-PINN avoids the need for integral discretization and thus eliminates discretization and truncation errors. Results of simulation studies show that A-PINN can obtain solutions with higher accuracy than the PINN adopting numerical discretization scheme; (ii) A-PINN does not rely on any fixed grids or nodes. It is a mesh-free method that can predict the solution at any point in the domain of solution without interpolation; and (iii) A-PINN is also amenable to the inverse problem of IDEs to which the conventional PINN is intractable because of integral discretization. By training the unknown parameters as hyper-parameters together with the parameters of DNN and adding new residuals of the measurement data to the loss function, the A-PINN framework for solving the forward problem can be easily adapted to solve the inverse problem of IDEs. Results of numerical simulations show that A-PINN can accurately discover the unknown parameters in IDEs even with noisy data.

The remainder of this paper is organized as follows. In Section 2, we describe the general form of the forward and inverse problems of nonlinear IDEs. In Section 3, we brief the standard PINN and then propose A-PINN. In Section 4, we demonstrate the effectiveness of the A-PINN framework with numerous examples for solving the forward and inverse problems of various nonlinear IDEs. More specifically, we first address a benchmark problem of 1-dimensional (1D) nonlinear Volterra IDE and explore the impact of the parameters of A-PINN on solution accuracy. Afterwards, the forward problems of nonlinear Volterra IDEs system, 2-dimensional (2D) nonlinear Volterra IDE, 10-dimensional (10D) nonlinear Volterra IDE, and nonlinear Fredholm IDE are pursued. To demonstrate the capability of A-PINN to solve inverse problems of nonlinear IDEs, the unknown parameters in both nonlinear Volterra IDE and nonlinear Volterra IDEs system are discovered. Finally, we conclude the study in Section 5.

2. Problem statement

The general form of IDEs can be expressed as

$$u^{(n)}(x) = f(x) + \lambda \int_{g(x)}^{h(x)} K(x,t)\, u(t) dt \tag{1}$$

where $u^{(n)}(x)$ is the n-order ordinary derivative; $g(x)$ and $h(x)$ are the limits of integration; λ is a constant parameter; and $K(x,t)$ is the kernel function. When $g(x)$ and $h(x)$ are fixed, the form represents a Fredholm IDE; otherwise, when at least one of $g(x)$ and $h(x)$ is variable, the form represents a Volterra IDE. On the other hand, if the index n about the unknown function $u(x)$ is equal to one, the IDE is linear; otherwise, if the unknown function $u(x)$ has an index other than one, such as e^u, $\sinh(u)$, $\cos(u)$, the IDE is nonlinear. In this paper, we focus on nonlinear IDEs of both Volterra type and Fredholm type, which are more complex than linear IDEs.

We will consider both forward and inverse problems of nonlinear IDEs. In the forward problem, where the governing IDEs are definite, the parameters λ and boundary and initial conditions which ensure a unique solution are all known. The

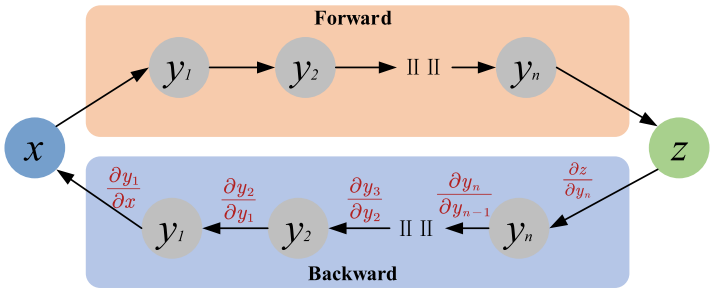

Fig. 2. Backward chain for derivative calculation in fully connected feedforward neural networks.

solution of the forward problem is the value of $u(x)$ at any x in the equation domain. In the inverse problem, where the IDEs are definite and boundary and initial conditions may be known or not, the parameters λ in the governing equation(s) are unknown. We will use some measurement data $u_m(x)$ to discover the unknown parameters λ.

3. Method

In this section, the proposed A-PINN for solving the forward and inverse problems of nonlinear IDEs is presented. First, we brief DNNs and automatic differentiation, which are the basic knowledge of PINN. Then, the original PINN framework and the PINN adopting integral discretization for solving IDEs are introduced. Finally, the A-PINN framework is presented in detail.

3.1. DNNs and automatic differentiation

In both PINN and A-PINN frameworks, a fully connected feedforward neural network is used as the core of the training model, which generally includes an input layer, an output layer and n hidden layers. The connection between layers is expressed as follows:

$$\begin{aligned} &\boldsymbol{y}_i = \sigma\left(\boldsymbol{w}_i \cdot \boldsymbol{y}_{i-1} + \boldsymbol{b}_i\right) \quad 1 \le i \le n \\ &\boldsymbol{z} = \boldsymbol{w}_{n+1} \cdot \boldsymbol{y}_n + \boldsymbol{b}_{n+1} \end{aligned} \tag{2}$$

where $i = [1, n]$ represents the hidden layers, and $n+1$ is the output layer. $\boldsymbol{y}_i$ denotes the output of the ith layer. $\boldsymbol{z}$ is the output of the neural network. $\sigma(\cdot)$ represents the activation function, which allows a neural network to map nonlinear relationship. $[\boldsymbol{\omega}_i, \boldsymbol{b}_i]$ represents the weight and bias of the ith layer, which will be updated during training. The fully connected feedforward neural network is a combination of linear summations and activation functions. Therefore, for a neural network with differentiable activation functions such as $tanh(\cdot)$ and $sin(\cdot)$, it is feasible to calculate the derivative of the output $\boldsymbol{z}$ with respect to the input $\boldsymbol{x}$ following the backward chain as shown in Fig. 2. The backward chain for calculating derivatives is called *automatic differentiation* [3]. Unlike the common numerical methods such as finite difference that suffers from truncation error, the automatic differentiation can elicit analytical derivatives by following the backward chain. Even better, in the popular machine learning platforms such as Pytorch [61,62] and Tensorflow [63], automatic differentiation is a built-in function that can be directly called.

3.2. Physics informed neural network (PINN)

PINN is a machine learning framework based on DNNs which successfully integrates physical information with neural networks. It has been proven that PINN can solve the forward and inverse problems of both linear and nonlinear PDEs [1]. Consider the forward problem of a PDE with Dirichlet boundary condition, given by

$$\begin{aligned} &\lambda \frac{\partial^n u(x,t)}{\partial x^n} + f(x,t) = 0 \\ &u(0,t) = \varphi(0,t) \\ &u(x,0) = g(x,0) \end{aligned} \tag{3}$$

where $\frac{\partial^n u(x,t)}{\partial x^n}$ is a n-order partial differential operator; λ is the parameter in the governing equation which is a known constant in the forward problem. $\varphi(0,t)$ is the Dirichlet boundary condition at $x = 0$; $g(x,0)$ is the initial condition at $t = 0$. A typical PINN framework for solving this problem is shown in Fig. 3.

In the PINN framework, the input (x,t) of the neural network is the coordinates of the training points which consist of three parts: the sampling points $(x^{ini}, 0)$ on the initial condition, the sampling points (x^b, t^b) on the boundary condition, and collocation points (x^f, t^f) in the equation domain. A fully connected feedforward DNN is employed to calculate the

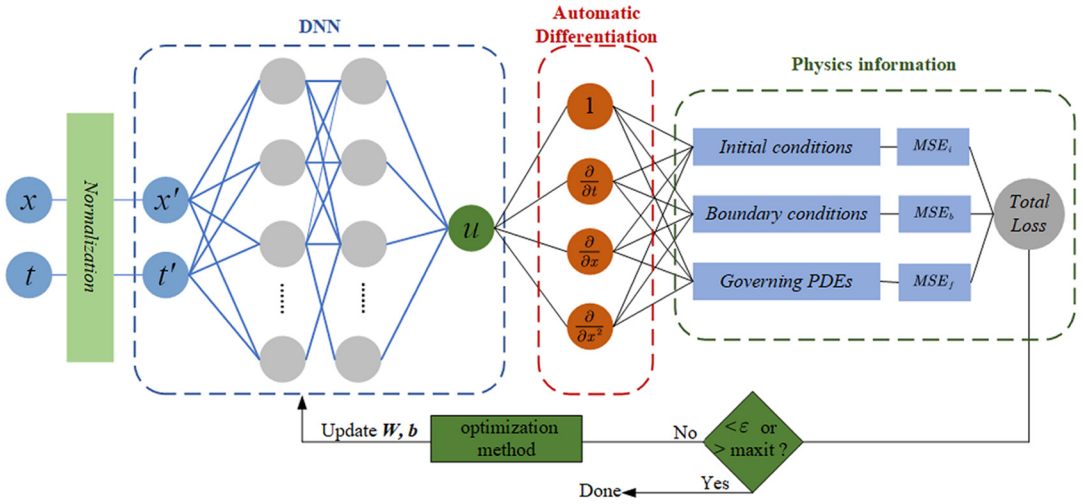

Fig. 3. A typical PINN framework to solve the forward problem of nonlinear PDEs with Dirichlet boundary condition.

predicted value $u_{pred}(x_i, t_i; \theta)$ corresponding to the input point (x_i, t_i) which approximates the exact value $u_{exact}(x_i, t_i)$, where θ is the parameters of the DNN including weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$. The automatic differentiation of the DNN is utilized to calculate the partial derivatives of $u_{pred}(x_i, t_i; \theta)$ with respect to x_i and t_i, which can be substituted into the initial condition, boundary condition, and governing equation to calculate the mean square errors of residuals of the above components. The linear sum of the mean square errors is regarded as the loss function MSE_{total}. Subsequently, through minimizing MSE_{total} with an optimizer to update the parameters θ of the DNN, $u_{pred}(x_i, t_i; \theta)$ would converge to the exact value $u_{exact}(x_i, t_i)$. For the PDE shown in Eq. (3), the loss function MSE_{total} is determined by

$$MSE_{total} = MSE_i + MSE_b + MSE_f \tag{4}$$

where

$$MSE_i = \frac{1}{N_i}\sum_{i=1}^{N_i}\left|u_{pred}\left(x_i^{ini}, 0; \theta\right) - g\left(x_i^{ini}, 0\right)\right|^2$$

$$MSE_b = \frac{1}{N_b}\sum_{i=1}^{N_b}\left|u_{pred}\left(0, t_i^b; \theta\right) - \varphi\left(0, t_i^b\right)\right|^2$$

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\lambda\frac{\partial^n u_{pred}(x_i^f, t_i^f; \theta)}{\partial x^n} + f\left(x_i^f, t_i^f\right)\right|^2$$

in which MSE_i, MSE_b and MSE_f are the mean square errors of residuals of the initial condition, boundary condition, and governing equation, respectively. N_i and N_b are the numbers of sampling points on the initial and boundary conditions, respectively. N_f is the number of collocation points randomly sampled in the equation domain. $g\left(x_i^{ini}, 0\right)$ are the exact values at sampling points $\left(x_i^{ini}, 0\right)$ on the initial condition, and $\varphi(0, t_i^b)$ are the exact values at sampling points $(0, t_i^b)$ on the Dirichlet boundary condition.

With the property of automatic differentiation and continuous prediction, PINN is a grid-free method that doesn't rely on any fixed grids or nodes and can predict the value at any point (x, t) in the equation domain without interpolation, thus effectively avoiding the truncation and discretization errors inevitable in traditional numerical methods.

For solution to the inverse problem of PDEs, i.e., to discover the unknown parameters λ in the PDE with measurement data, only minor modifications to the PINN framework shown in Fig. 3 are needed. The PINN framework for solving the inverse problem of PDEs is illustrated in Fig. 4. Now the unknown parameters λ as hyper-parameters are trained together with the DNN. In the inverse problem of PDEs, sometimes it is difficult to obtain the exact initial and boundary conditions. Most often, all we know are the governing equation and measurement data. In such cases, the physical information consists of only two parts, i.e., residuals of the governing equation and residuals of the measurement data. The loss function MSE_{total} is thus expressed as

$$MSE_{total} = MSE_m + MSE_f \tag{5}$$

where

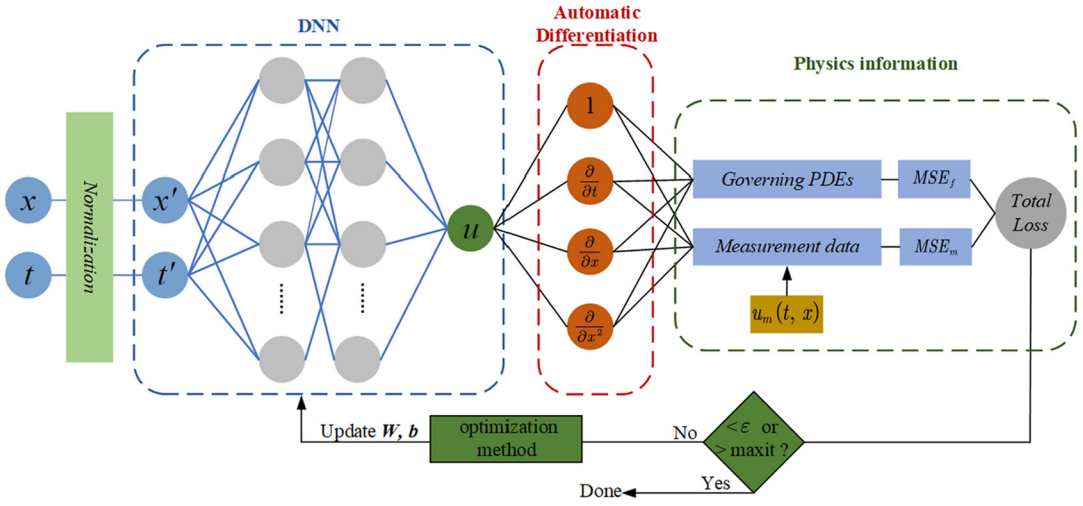

Fig. 4. The framework of PINN to solve the inverse problem of PDEs.

$$MSE_m = \frac{1}{N_m}\sum_{i=1}^{N_m}\left|u_{pred}\left(x_i^m, t_i^m; \theta, \lambda\right) - u_m\left(x_i^m, t_i^m\right)\right|^2$$

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\lambda\frac{\partial^n u_{pred}(x_i^f, t_i^f; \theta, \lambda)}{\partial x^n} + f\left(x_i^f, t_i^f\right)\right|^2$$

in which MSE_m and MSE_f are the mean square error of residuals of the measurement data and the mean square error of residuals of the governing equation, respectively; and N_m is the number of the measurement data. $u_{pred}\left(x_i^m, t_i^m; \theta, \lambda\right)$ and $u_m(x_i^m, t_i^m)$ are the predicted and measured values at the measurement points $\left(x_i^m, t_i^m\right)$. θ is the parameters of the DNN and λ is the unknown parameters in the PDE.

3.3. Integral discretization in PINN for IDEs

In the state-of-the-art PINN method for IDEs, automatic differentiation is adopted to calculate the ordinary differential operator, but numerical methods (e.g., finite element method, finite difference method, or spectral method) must be applied to approximate the integral operator. With the integral discretization by numerical methods, IDEs can be transformed into ODEs. This treatment however inevitably brings discretization and truncation errors to the solution by PINN. For instance, consider the following IDE:

$$\frac{dy}{dx} + y(x) = \int_0^x e^{t-x}y(t)dt \tag{6}$$

Discretization of the integral by the n-order Gauss Legendre yields

$$\int_0^x e^{t-x}y(t)dt \approx \sum_{i=1}^{n}\alpha_i e^{t_i(x)-x}y(t_i(x)) \tag{7}$$

where $t_i(x)$ and α_i are the known integral nodes and weighting factors, respectively. After integral discretization, the IDE is transformed into an ODE, which can be solved by PINN, as

$$\frac{dy}{dx} + y(x) \approx \sum_{i=1}^{n}\alpha_i e^{t_i(x)-x}y(t_i(x)) \tag{8}$$

Although this seems a simple and straightforward way to treat IDEs, we note that for calculating MSE_f in Eq. (4), it is necessary to calculate the Gauss Legendre quadrature at each collocation point which involves $y(t_i(x))$ at n integration nodes $t_i(x)$ and all $y(t_i(x))$ should be calculated by PINN in each iteration. For example, for a 20th-order Gauss Legendre quadrature, the PINN should calculate $y(t_i(x))$ at 20 different integration nodes $t_i(x)$ for each collocation point. When 100 collocation points are involved, in each iteration we need to use PINN to predict $y(x)$ at 100 collocation points as well as to calculate $y(t_i(x))$ at $20 \times 100 = 2000$ integration nodes. With the increase of the order for discretization integration

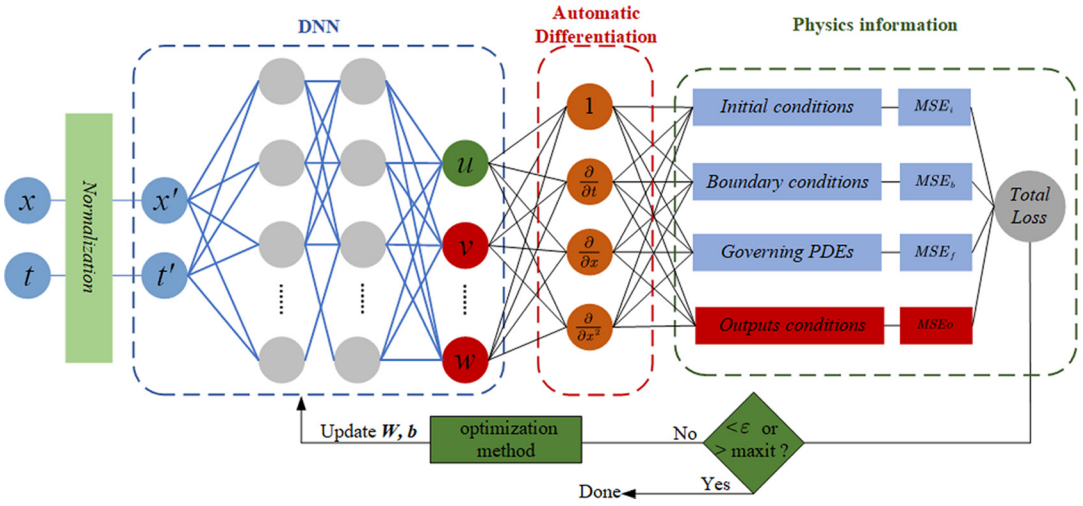

Fig. 5. The framework of A-PINN for solving the forward problem of IDEs.

and the number of collocation points, the computational amount is significantly increased, which diminishes the efficiency in training. Also, as indicated in [60], the discretization error is unavoidable in the solution due to the approximation of integral by Gaussian quadrature.

3.4. A-PINN: auxiliary physics informed neural networks

In this section, we propose A-PINN to solve the forward and inverse problems of nonlinear IDEs. The overall framework of A-PINN is illustrated in Fig. 5. Unlike PINN that only approximates primary variables in the governing equation, a multi-output DNN is utilized in the A-PINN framework to simultaneously calculate the primary outputs and auxiliary outputs which respectively represent the variables and integrals involved in the governing equation. Subsequently, the relationship between the primary outputs and auxiliary outputs is imposed by enforcing additional output conditions in compliance with physical laws.

We first consider the first-order nonlinear Volterra IDE:

$$u^{(n)}(x) = f(x) + \lambda \int_0^x K(t)u(t)dt, \quad u(0) = a \tag{9}$$

By defining an auxiliary output $v(x)$ to represent the integral in Eq. (9), we can transform Eq. (9) into

$$\begin{cases} u^{(n)}(x) = f(x) + \lambda \cdot v(x) \\ v(x) = \int_0^x K(t)u(t)dt \\ u(0) = a \end{cases} \tag{10}$$

For this one-dimensional problem where the independent variable is x only, the input of the DNN is the coordinates of x at training points which consist of two parts: the sampling points x^{ini} on the initial condition and collocation points x^f in the equation domain. A fully connected feedforward DNN is employed to calculate the predicted value $u_{pred}(x_i;\theta)$ and auxiliary output $v_{pred}(x_i;\theta)$ which approximate the exact value $u_{exact}(x_i)$ and the integral $\int_0^{x_i} K(t)u(t)dt$, respectively, where θ is the parameters of the DNN. To avoid integral manipulation, the relationship between $u(x)$ and $v(x)$ is re-expressed as

$$\begin{cases} \frac{dv(x)}{dx} = K(x)u(x) \\ v(0) = 0 \end{cases} \tag{11}$$

The automatic differentiation of the DNN is executed to calculate the ordinary derivative on the left-hand side of Eq. (10), which together with the predicted value $u_{pred}(x_i;\theta)$ can be substituted into the initial condition and governing equation to calculate the mean square errors of residuals of the two components. In addition, a new output condition (i.e. Eq. (11)) is added as a physical law to constrain the relationship between the auxiliary output v and the primary output u. The mean square error of residuals of this new output condition is calculated by

$$MSE_o = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial v_{pred}(x_i^f;\theta)}{\partial x} - K\left(x_i^f\right)u_{pred}\left(x_i^f;\theta\right)\right|^2 \tag{12}$$

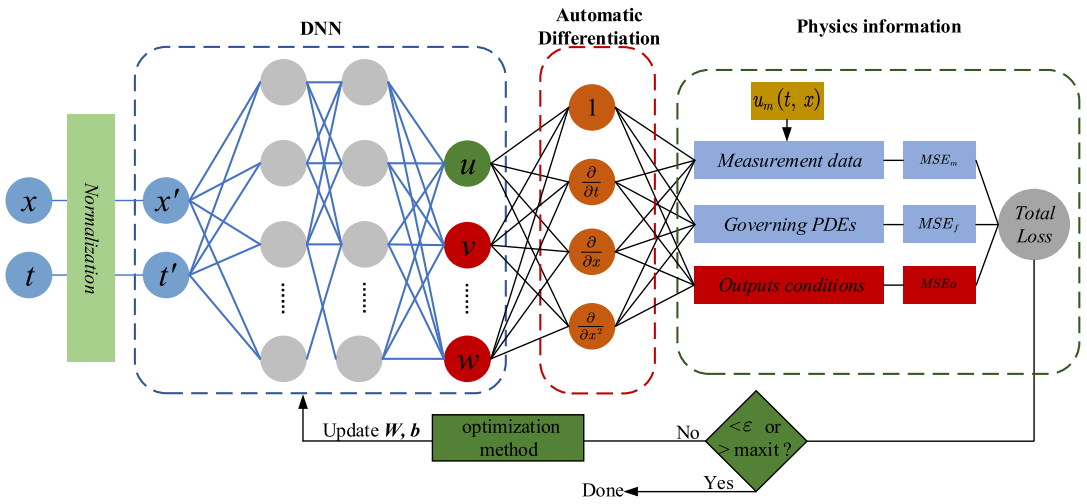

Fig. 6. The framework of A-PINN for solving the inverse problem of IDEs.

where N_f is the number of collocation points evenly (or randomly) sampled in the equation domain. For example, if 11 collocation points are evenly sampled in the domain of $[0, 1]$, the interval between the collocation points is equal to 0.1. The mean square error of residuals of the initial conditions MSE_i is calculated by

$$MSE_i = \left|u_{pred}\left(0;\theta\right) - a\right|^2 + \left|v_{pred}\left(0;\theta\right) - 0\right|^2 \tag{13}$$

The mean square error of residuals of the governing equation is calculated by

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial^n u_{pred}(x_i^f;\theta)}{\partial x^n} - \left[f\left(x_i^f\right) + \lambda \cdot v_{pred}\left(x_i^f;\theta\right)\right]\right|^2 \tag{14}$$

The MSE_{total} in the A-PINN framework is considered as the weighted sum of all the mean square errors, which is expressed as

$$MSE_{total} = w_i \cdot MSE_i + w_f \cdot MSE_f + w_o \cdot MSE_o \tag{15}$$

An accurate solution to the IDE can be achieved when all the residuals approach 0. If there is an imbalance among the residuals, for example, when MSE_f and MSE_o approach 0 but MSE_i is non-zero, we are unable to achieve an accurate solution. An adaptive weight strategy is therefore used in this study that can automatically balance various residuals to achieve a balanced convergence. In this strategy, the weights of various residual components are proportional to the values of residuals while normalizing the weight of the smallest residual component to be 1 in each iteration. As such, we give greater weights to the larger components to speed up their convergence. The weight for each component is determined by

$$\left[w_i, w_f, w_o\right] = \frac{\left[MSE_i, MSE_f, MSE_o\right]}{\min(MSE_i, MSE_f, MSE_o)} \tag{16}$$

In each iteration, we will calculate the adaptive weights of all residual components and take the weighted sum of all MSE as MSE_{total}. Converged solution of the IDE is then obtained by using an optimizer to minimize MSE_{total}. In this study, we choose L-BFGS, a quasi-Newton full-batch gradient-based optimizer, to optimize the loss function. Although there is no theoretical guarantee that the optimizer converges to globally optimal solutions, numerical experiments show that making use of adaptive weights and L-BFGS optimizer, converged solutions with good accuracy can be obtained by A-PINN.

The inverse problem of IDEs is intended to discover unknown parameters in governing equations with measurement data, e.g., to discover the parameter λ in Eq. (9). With few modifications to the A-PINN framework for solving the forward problem of IDEs, the proposed A-PINN can be conveniently applied to solve the inverse problem of IDEs with the procedure framework shown in Fig. 6. For the inverse problem of Eq. (9), a fully connected feedforward DNN is configured to calculate the predicted value $u_{pred}(x_i;\theta,\lambda)$ and auxiliary output $v_{pred}(x_i;\theta,\lambda)$ which approximate the exact value $u_{exact}(x_i)$ and the integral $\int_0^{x_i} K(t)u(t)dt$, respectively, where θ is the parameters of the DNN and λ is the unknown parameter in the IDE.

In addressing inverse problems, it may be difficult to obtain the exact initial and boundary conditions of IDEs. Most often, all we know may only be governing equations and sparse measurement data. Of course, if we have more information such as boundary and initial conditions, such information can be easily incorporated into the loss function, same as the A-PINN framework for solving forward problems. We first consider the case with the least amount of information, i.e., only

governing equations and measurement data. For the IDE described by Eq. (9), the loss function MSE_{total} with A-PINN can be determined by

$$MSE_{total} = w_m \cdot MSE_m + w_f \cdot MSE_f + w_o \cdot MSE_o \tag{17}$$

where

$$MSE_m = \frac{1}{N_m}\sum_{i=1}^{N_m}\left|u_{pred}\left(x_i^m;\theta,\lambda\right) - u_m\left(x_i^m\right)\right|^2$$

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial^n u_{pred}(x_i^f;\theta,\lambda)}{\partial x^n} - \left[f\left(x_i^f\right) + \lambda\cdot v_{pred}\left(x_i^f;\theta,\lambda\right)\right]\right|^2$$

$$MSE_o = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial v_{pred}(x_i^f;\theta,\lambda)}{\partial x} - K\left(x_i^f\right)u_{pred}\left(x_i^f;\theta,\lambda\right)\right|^2$$

where N_m is the number of measurement data. The weights still be adaptive to balance the convergence of various residual components, but the difference from the A-PINN framework for solving forward problems is that we keep the weight of MSE_m as 1.0. It is because when there is noise in the measurement data, the optimal solution of MSE_m will no longer be 0, while the optimal solution of MSE_f and MSE_o is still 0. If we adjust the adaptive weights for all MSE simultaneously, A-PINN tends to overfit the noisy measurement data by conferring MSE_m an excessive weight. Therefore, the adaptive weight is now determined by

$$\left[w_m, w_f, w_o\right] = [1.0, \frac{MSE_f}{\min\left(MSE_f, MSE_o\right)}, \frac{MSE_o}{\min\left(MSE_f, MSE_o\right)}] \tag{18}$$

For inverse problems, we still choose L-BFGS optimizer to minimize MSE_{total} so that converged solution of the unknown parameters can be attained.

In comparison with traditional pure numerical analysis methods such as finite difference method and the method based on shifted Jacobi polynomials [64], the proposed A-PINN framework is more intuitive and simple for solving IDEs in that it does not require complex matrix manipulations and integral discretization. Different from the PINN adopting integral discretization [37,60], A-PINN is independent of any fixed grids and nodes. As pointed out before, integral discretization brings about extra computational cost as well as discretization and truncation errors. A-PINN avoids the above shortcomings and is convenient for application to solve the inverse problem of IDEs with only few modifications to the framework for the forward problem.

4. Numerical experiments

In this section, we conduct numerical experiments of solving various nonlinear IDEs by A-PINN to illustrate its capability and efficiency. The forward problems of 1D Volterra IDE, Volterra IDEs system, 2D Volterra IDE, 10D Volterra IDE, and Fredholm IDE will be addressed first. Then the inverse problems of 1D Volterra IDE and Volterra IDEs system will be pursued. In the numerical experiments, the activation function in all neural networks is $tanh(\cdot)$, a widely used differentiable nonlinear activation function. The optimizer used in all problems is L-BFGS with 0.01 learning rate. All algorithms are coded in Python with PyTorch, an open-source machine learning framework. The codes and trained models of the experiments will be available on GitHub.

4.1. The forward problem of 1D nonlinear Volterra IDE

Our first experiment aims to demonstrate the ability of A-PINN for solving 1D nonlinear Volterra IDE. Here we consider a benchmark problem, which was solved in [60] with the PINN adopting integral discretization as delineated in Section 3.4. The 1D nonlinear Volterra IDE is expressed as

$$\frac{du(x)}{dx} + u(x) = \lambda\int_0^x e^{t-x}u(t)dt, \quad x\in[0,5] \tag{19}$$

Considering the initial condition as $u(0)=1$ and taking $\lambda=1$, the equation has an exact solution $u(x)=e^{-x}\cosh x$. Defining an auxiliary output $v(x)$ to represent the integral in the equation, Eq. (19) can be re-expressed as

Table 1
The relative L2 error between the predicted and exact solutions $u(x)$ for different numbers of collocation points N_f. Here we use a neural network architecture with 4 hidden layers and 100 neurons in each hidden layer.

N_f	10	20	50	100	200	500	1000
Error	9.06E-04	4.13E-04	2.59E-04	8.31E-04	6.93E-04	9.29E-04	5.20E-04

$$\begin{cases} \frac{du(x)}{dx} + u(x) = \lambda v(x), \quad x \in [0,5] \\ v(x) = \int_0^x e^{t-x} u(t) dt \\ u(0) = 1 \\ v(0) = 0 \end{cases} \tag{20}$$

We represent the derivative of $v(x)$ as $\frac{dv(x)}{dx} = u(x) - \int_0^x e^{t-x} u(t) dt = u(x) - v(x)$. A two-output DNN is constructed to approximate $u(x)$ and $v(x)$ in Eq. (20). Thus, in the A-PINN framework, the mean square error of residuals of the governing equation MSE_f is calculated by

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| \frac{\partial u_{pred}(x_i^f;\theta)}{\partial x} + u_{pred}\left(x_i^f;\theta\right) - v_{pred}\left(x_i^f;\theta\right) \right|^2 \tag{21}$$

where N_f is the number of collocation points. The mean square error of residuals of the initial conditions MSE_i is calculated by

$$MSE_i = \left|u_{pred}(0;\theta) - 1\right|^2 + \left|v_{pred}(0;\theta) - 0\right|^2 \tag{22}$$

The relationship between the primary output $u(x)$ and the auxiliary output $v(x)$ is constrained by a new output condition, and the mean square error of residuals of this output condition is calculated by

$$MSE_o = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| \frac{\partial v_{pred}(x_i^f;\theta)}{\partial x} - \left[u_{pred}\left(x_i^f;\theta\right) - v_{pred}\left(x_i^f;\theta\right)\right] \right|^2 \tag{23}$$

The MSE_{total} in the A-PINN framework is the weighted sum of all the mean square errors, which is expressed as

$$MSE_{total} = w_i \cdot MSE_i + w_f \cdot MSE_f + w_o \cdot MSE_o \tag{24}$$

The weights of the three MSE are determined by the adaptive weighting strategy described in Section 3.4. First, we use a DNN with 4 hidden layers and 100 neurons in each layer to solve this IDE. The training data set consists of two parts: the initial point $x^{ini} = 0$ and 50 collocation points evenly sampled in the equation domain. To evaluate the accuracy of the solution, we calculate the relative L2 error between the solution by A-PINN and the exact solution. The relative L2 error is obtained by

$$L_2\ error = \frac{\sqrt{\sum_{i=1}^{M} \left|u_{pred}^i - u_{exact}^i\right|^2}}{\sqrt{\sum_{i=1}^{M} \left|u_{exact}^i\right|^2}} \tag{25}$$

The solution by A-PINN after 2000 iterations and the exact solution are presented in Fig. 7. The relative L2 error between the A-PINN solution and the exact value is 0.0259%. In [60], Gauss-Legendre quadrature for integral discretization was employed to transform the same IDE into an ODE, then the ODE was solved with PINN. The relative L2 error of the result obtained in [60] is 0.2%. This benchmark study shows that A-PINN can achieve more accurate solution for the IDE than the PINN adopting integral discretization.

To further evaluate the performance of the proposed A-PINN, we obtain its predictive accuracy for scenarios with different numbers of collocation points and different neural network architectures. Table 1 provides the relative L2 errors of the solutions by A-PINN, where the number of collocation points changes from 10 to 1000 and the neural network is fixed to 4 hidden layers with 100 neurons in each hidden layer. The results show that the relative L2 errors in all cases are less than 0.1%, and the accuracy of the solutions does not alter significantly with different numbers of collocation points N_f. Table 2 shows the results of relative L2 errors for different neural network architectures, where the number of collocation points N_f is fixed to 50. The influences of the number of hidden layers and the number of neurons on the accuracy of the solution are fluctuating. In general, the accuracy of the solution slightly increases with the increase of the number of neurons in each hidden layer, while it slightly decreases with the increase of the number of hidden layers.

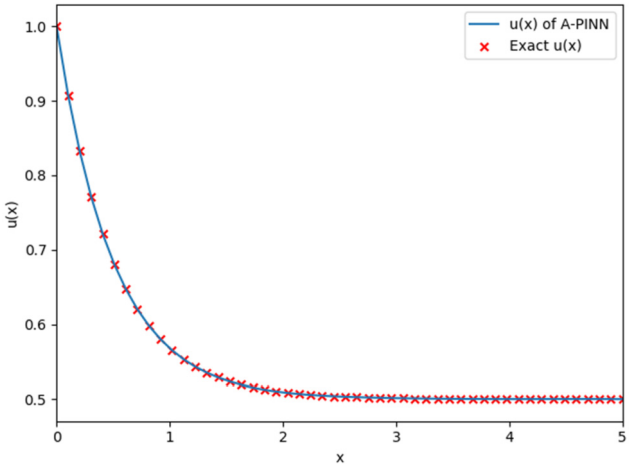

Fig. 7. The solution by A-PINN and the exact solution for 1D Volterra IDE. Blue line – the A-PINN solution; Red mark – the exact solution. (For interpretation of the colors in the figures, the reader is referred to the web version of this article.)

Table 2
The relative L2 error between the predicted and exact solutions $u(x)$ for different neural network architectures. Here we set the number of collocation points $N_f = 50$.

Layers	Neurons				
	20	40	60	80	100
2	5.14E-04	9.11E-04	6.32E-04	1.04E-03	5.06E-04
4	6.93E-04	4.54E-04	3.38E-04	9.17E-04	2.59E-04
6	8.55E-04	3.22E-04	2.65E-04	1.71E-03	1.20E-03
8	1.11E-03	1.98E-03	1.27E-03	5.42E-04	1.32E-03

4.2. The forward problem of nonlinear Volterra IDEs system

Consider the following nonlinear Volterra IDEs system:

$$\begin{cases} \frac{d^2u(x)}{dx^2} = 1 - \frac{x^3}{3} - \frac{1}{2}\frac{dv(x)}{dx} + \lambda_1 \int_0^x [u^2(t) + v^2(t)]dt \\ \frac{d^2v(x)}{dx^2} = -1 + x^2 - xu(x) + \lambda_2 \int_0^x [u^2(t) - v^2(t)]dt \\ u(0) = 1;\ v(0) = -1;\ \frac{du(0)}{dx} = 2;\ \frac{dv(0)}{dx} = 0;\ x \in [0, 1] \end{cases} \tag{26}$$

We set $\lambda_1 = \frac{1}{2}$ and $\lambda_2 = \frac{1}{4}$, then the IDEs system has its exact solution as $u(x) = x + e^x$ and $v(x) = x - e^x$. By defining auxiliary outputs $w(x)$ and $p(x)$ to represent the integrals, the governing equations can be re-expressed as

$$\begin{cases} \frac{d^2u(x)}{dx^2} = 1 - \frac{x^3}{3} - \frac{1}{2}\frac{dv(x)}{dx} + \lambda_1 w(x) \\ \frac{d^2v(x)}{dx^2} = -1 + x^2 - xu(x) + \lambda_2 p(x) \\ w(x) = \int_0^x [u^2(t) + v^2(t)]dt \\ p(x) = \int_0^x [u^2(t) - v^2(t)]dt \end{cases} \tag{27}$$

To constrain the relationships between $u(x)$, $v(x)$, $w(x)$ and $p(x)$, we use automatic differentiation in the DNN to explore the following physical laws:

$$\begin{cases} \frac{dw(x)}{dx} = u^2(x) + v^2(x) \\ \frac{dp(x)}{dx} = u^2(x) - v^2(x) \\ w(0) = 0;\ p(0) = 0 \end{cases} \tag{28}$$

We use a four-output neural network to approximate $u(x)$, $v(x)$, $w(x)$ and $p(x)$ in Eq. (27). Thus, in the A-PINN framework, the mean square error of residuals of the governing equations MSE_f is composed of

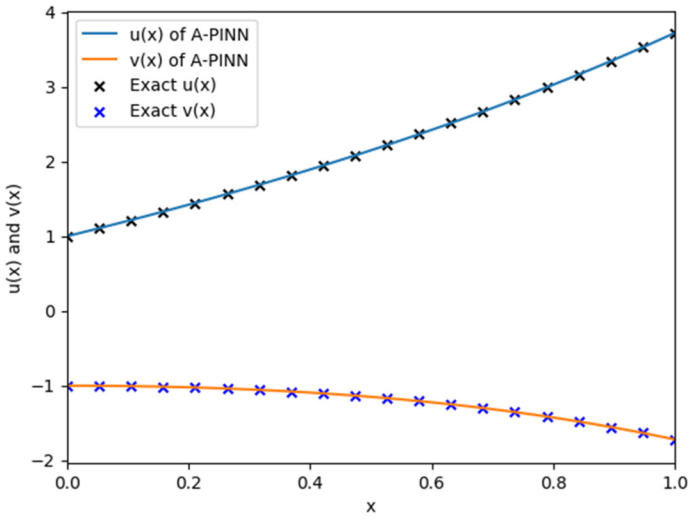

Fig. 8. The solution by A-PINN and the exact solution for Volterra IDEs system.

$$MSE_{f1} = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial^2 u_{pred}(x_i^f;\theta)}{\partial x^2} - \left[1 - \frac{{x_i^f}^3}{3} - \frac{1}{2}\frac{\partial v_{pred}\left(x_i^f;\theta\right)}{\partial x} + \lambda_1 w_{pred}\left(x_i^f;\theta\right)\right]\right|^2 \tag{29}$$

$$MSE_{f2} = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial^2 v_{pred}(x_i^f;\theta)}{\partial x^2} - \left[-1 + {x_i^f}^2 - x_i^f u_{pred}\left(x_i^f;\theta\right) + \lambda_2 p_{pred}\left(x_i^f;\theta\right)\right]\right|^2 \tag{30}$$

The mean square error of residuals of the initial conditions MSE_i comprises

$$MSE_{i1} = \left|u_{pred}(0;\theta) - 1\right|^2 + \left|v_{pred}(0;\theta) + 1\right|^2 + \left|\frac{\partial u_{pred}(0;\theta)}{\partial x} - 2\right|^2 + \left|\frac{\partial v_{pred}(0;\theta)}{\partial x} - 0\right|^2 \tag{31}$$

$$MSE_{i2} = \left|w_{pred}(0;\theta) - 0\right|^2 + \left|p_{pred}(0;\theta) - 0\right|^2 \tag{32}$$

The relationships between the primary outputs $u(x)$, $v(x)$ and the auxiliary outputs $w(x)$, $p(x)$ are constrained by two new output conditions, and the mean square errors of residuals of these two output conditions are

$$MSE_{o1} = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial w_{pred}\left(x_i^f;\theta\right)}{\partial x} - \left[{u_{pred}}^2\left(x_i^f;\theta\right) + {v_{pred}}^2\left(x_i^f;\theta\right)\right]\right|^2 \tag{33}$$

$$MSE_{o2} = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial p_{pred}\left(x_i^f;\theta\right)}{\partial x} - \left[{u_{pred}}^2\left(x_i^f;\theta\right) - {v_{pred}}^2\left(x_i^f;\theta\right)\right]\right|^2 \tag{34}$$

The loss function MSE_{total} in A-PINN is the weighted sum of all the mean square errors, which is determined by

$$MSE_{total} = w_i \cdot (MSE_{i1} + MSE_{i2}) + w_f \cdot (MSE_{f1} + MSE_{f2}) + w_o \cdot (MSE_{o1} + MSE_{o2}) \tag{35}$$

The weights of various MSE components are calculated automatically by the adaptive weighting strategy given in Section 3.4. We choose a 'shallow' yet 'wide' neural network model to address this problem: it consists of 2 hidden layers with 40 neurons in each hidden layer. The training data set includes two parts: the initial value $x^{ini} = 0$ and 50 collocation points evenly sampled in the equation domain. The solution by A-PINN after 500 iterations and the exact solution are presented in Fig. 8. The relative L2 error between the A-PINN solution and the exact value is 0.00559%.

4.3. The forward problem of nonlinear 2D Volterra IDE

Consider the following nonlinear 2D Volterra IDE:

$$\frac{\partial^2 u(t,x)}{\partial t^2} = \frac{\partial u(t,x)}{\partial x} - \frac{\partial u(t,x)}{\partial t} - u(t,x) + g(t,x) + \lambda\int_0^x\int_0^t t\cos(y_1 - y_2)\,u(y_1,y_2)\,dy_1 dy_2 \tag{36}$$

with the initial and boundary conditions:

$$u(0,x)=x;\ \frac{\partial u(0,x)}{\partial t}=\sin(x);\ u(t,0)=tsin(t);\ 0\le t,x\le 1 \tag{37}$$

Here, the parameter λ is set as 1 and $g(t,x)$ is such chosen that the exact solution is $u(t,x)=x+tsin(t+x)$. By defining auxiliary outputs $v(t,x)$ and $w(t,x)$ to represent the integrals in the equation, the governing equation can be re-expressed as

$$\begin{cases} \frac{\partial^2 u(t,x)}{\partial t^2}=\frac{\partial u(t,x)}{\partial x}-\frac{\partial u(t,x)}{\partial t}-u(t,x)+g(t,x)+\lambda v(t,x) \\ \frac{\partial v(t,x)}{\partial x}=t\int_0^t cos(y_1-x)\,u(y_1,x)\,dy_1=t\cdot w(t,x) \\ \frac{\partial w(t,x)}{\partial t}=cos(t-x)\,u(t,x) \end{cases} \tag{38}$$

The new initial and boundary conditions are $v(t,0)=0$ and $w(0,x)=0$. In the A-PINN framework, the mean square error of residuals of the governing equation MSE_f is represented as

$$MSE_f=\frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial^2 u_{pred}\left(t_i^f,x_i^f;\theta\right)}{\partial t^2}-[\frac{\partial u_{pred}\left(t_i^f,x_i^f;\theta\right)}{\partial x}-\frac{\partial u_{pred}\left(t_i^f,x_i^f;\theta\right)}{\partial t}-u_{pred}\left(t_i^f,x_i^f;\theta\right)+g\left(t_i^f,x_i^f\right)+\lambda v_{pred}\left(t_i^f,x_i^f;\theta\right)]\right|^2 \tag{39}$$

where N_f is the number of collocation points which are randomly sampled in the equation domain. The mean square error of residuals of the initial conditions MSE_i is

$$MSE_i=\frac{1}{N_i}\sum_{i=1}^{N_i}\left|u_{pred}\left(0,x_i^{ini};\theta\right)-x_i^{ini}\right|^2+\frac{1}{N_i}\sum_{i=1}^{N_i}\left|w_{pred}\left(0,x_i^{ini};\theta\right)-0\right|^2+\frac{1}{N_i}\sum_{i=1}^{N_i}\left|\frac{\partial u_{pred}\left(0,x_i^{ini};\theta\right)}{\partial x}-\sin\left(x_i^{ini}\right)\right|^2 \tag{40}$$

where N_i is the number of training points which are randomly sampled on the initial conditions. The mean square error of residuals of the boundary conditions MSE_b is

$$MSE_b=\frac{1}{N_b}\sum_{i=1}^{N_b}\left|u_{pred}\left(t_i^b,0;\theta\right)-t_i^b sin\left(t_i^b\right)\right|^2+\frac{1}{N_b}\sum_{i=1}^{N_b}\left|v_{pred}\left(t_i^b,0;\theta\right)-0\right|^2 \tag{41}$$

where N_b is the number of training points which are randomly sampled on the boundary conditions. The relationships between the primary output $u(t,x)$ and the auxiliary outputs $v(t,x)$, $w(t,x)$ are constrained by two new output conditions, and the mean square errors of residuals of these two output conditions are

$$MSE_{o1}=\frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial v_{pred}\left(t_i^f,x_i^f;\theta\right)}{\partial x}-t_i^f\cdot w_{pred}\left(t_i^f,x_i^f;\theta\right)\right|^2 \tag{42}$$

$$MSE_{o2}=\frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial w_{pred}\left(t_i^f,x_i^f;\theta\right)}{\partial t}-cos\left(t_i^f-x_i^f\right)\cdot u_{pred}\left(t_i^f,x_i^f;\theta\right)\right|^2 \tag{43}$$

The loss function MSE_{total} in the A-PINN framework is the weighted sum of all the mean square errors, which is determined by

$$MSE_{total}=w_i\cdot MSE_i+w_b\cdot MSE_b+w_f\cdot MSE_f+w_o\cdot(MSE_{o1}+MSE_{o2}) \tag{44}$$

The weights of multiple MSE components are calculated by the adaptive weighting strategy. We consider again a neural network with 2 hidden layers and 40 neurons in each hidden layer. The training data set consists of three parts: 100 initial points $\left(0,x_i^{ini}\right)$ randomly sampled at $t=0$, 100 boundary points $\left(t_i^b,0\right)$ randomly sampled at $x=0$, and 5000 collocation points $\left(t_i^f,x_i^f\right)$ randomly sampled in the equation domain. The solution by A-PINN after 500 iterations and the error are presented in Fig. 9. The relative L2 error between the A-PINN solution and the exact solution is 0.0399%.

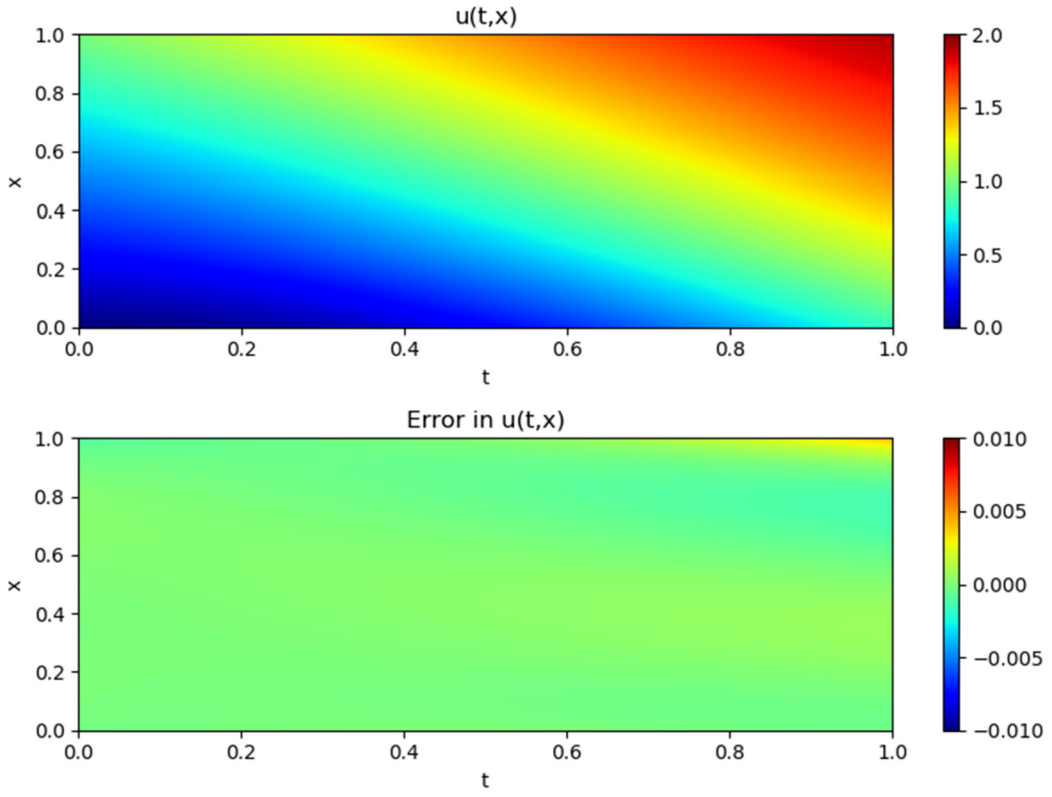

Fig. 9. The solution by A-PINN for 2D Volterra IDE (upper) and the error of the solution (bottom).

4.4. The forward problem of nonlinear 10D Volterra IDE

Consider the following nonlinear 10D Volterra IDE:

$$\begin{cases} \frac{\partial u(t,x_1,x_2,\cdots,x_9)}{\partial t} + \frac{\partial u(t,x_1,x_2,\cdots,x_9)}{\partial x_1} + \frac{\partial u(t,x_1,x_2,\cdots,x_9)}{\partial x_2} + \cdots + \frac{\partial u(t,x_1,x_2,\cdots,x_9)}{\partial x_9} = f\left(t,x_1,x_2,\cdots,x_9\right) \\ f\left(t,x_1,x_2,\cdots,x_9\right) = u(t,x_1,x_2,\cdots,x_9) + g(t,x_1,x_2,\cdots,x_9) \\ + \int_o^{x_9}\cdots\int_0^{x_1}\int_0^t t\cdot u(t,x_1,x_2,\cdots,x_9)dy_1dy_2\cdots dy_{10} \\ 0 \le t,x_1,x_2,\cdots,x_9 \le 1 \end{cases} \tag{45}$$

Here $g\left(t,x_1,x_2,\cdots,x_9\right)$ is such chosen that the exact solution is $u\left(t,x_1,x_2,\cdots,x_9\right) = t\cdot(x_1+x_2+x_3)\cdot\sin\left(x_4+x_5+x_6\right)\cdot\cos\left(x_7+x_8+x_9\right)$. The boundary conditions of the Volterra IDE are Dirichlet boundary conditions whose value can be obtained from the exact solution. Similar to the above nonlinear 2D Volterra IDE example, we define 10 auxiliary outputs $v_1\left(t,x_1,x_2,\cdots,x_9\right), v_2\left(t,x_1,x_2,\cdots,x_9\right),\cdots,v_{10}\left(t,x_1,x_2,\cdots,x_9\right)$ to represent the integrals in the equation respectively, thus the first term in Eq. (45) can be re-expressed as

$$f\left(t,x_1,x_2,\cdots,x_9\right) = u(t,x_1,x_2,\cdots,x_9) + g(t,x_1,x_2,\cdots,x_9) + v_1(t,x_1,x_2,\cdots,x_9) \tag{46}$$

We define the 10 outputs' conditions to constrain the relationship between the 10 auxiliary outputs and the primary output $u\left(t,x_1,x_2,\cdots,x_9\right)$. At the same time, 10 boundary conditions belonging to the auxiliary outputs are defined. Similarly, we calculate the mean square error of residuals of the governing equation MSE_f, the mean square error of residuals of the boundary conditions MSE_b, and the mean square errors of residuals of output conditions MSE_o. There are totally 20 terms in MSE_b and 10 terms in MSE_o. The loss function MSE_{total} in the A-PINN framework is the weighted sum of all the mean square errors, which is determined by

$$MSE_{total} = w_b\cdot MSE_b + w_f\cdot MSE_f + w_o\cdot MSE_o \tag{47}$$

The weights of multiple MSE components are calculated by the adaptive weighting strategy. We train a neural network with 4 hidden layers and 40 neurons in each hidden layer to solve this forward problem. The training data set consists of two parts: 1000 boundary points randomly sampled on the 10 boundary conditions, and 10000 collocation points $\left(t_i^f, x_i^f\right)$ randomly sampled in the equation domain. To illustrate the solution of A-PINN, we define two different planes in the

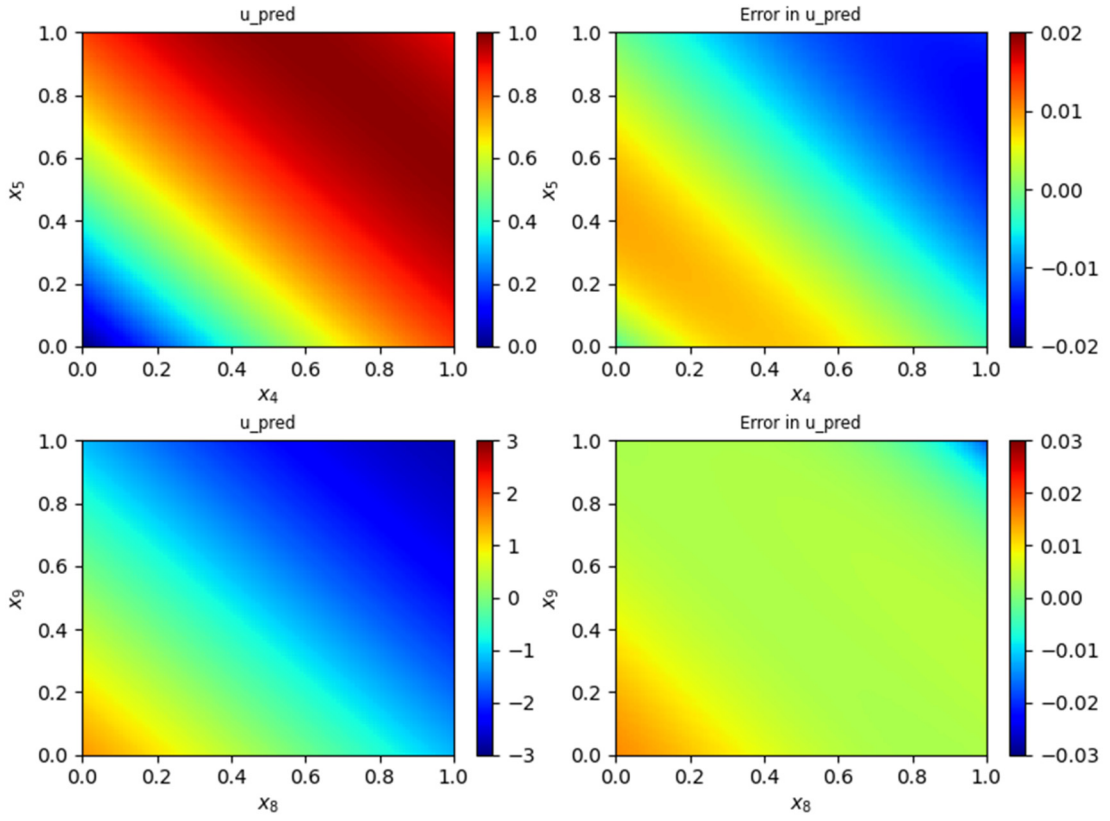

Fig. 10. A-PINN solution and error for 10D Volterra IDE. Top: solution by A-PINN on Plane 1 (left), and the corresponding absolute error (right). Bottom: solution by A-PINN on Plane 2 (left), and the corresponding absolute error (right).

equation domain. The exact solution on Plane 1 is $u\,(1,1,0,0,x_4,x_5,0,0,0,0) = \sin(x_4+x_5)$, and the exact solution on Plane 2 is $u\,(1,1,1,1,0,1,1,1,x_8,x_9) = 3\cdot\sin(2)\cdot\cos(x_8+x_9+1)$. The solution by A-PINN after 1000 iterations on these two planes is illustrated in Fig. 10. In addition, to estimate the accuracy of the A-PINN solution in the whole equation domain, we randomly sample 100000 test points in the equation domain and calculate the relative L2 error with Eq. (25). The relative L2 error of the solution by A-PINN is 0.519%.

4.5. The forward problem of nonlinear 1D Fredholm IDE

Consider the following nonlinear 1D Fredholm IDE:

$$\frac{du(x)}{dx} = cosx - x + \frac{1}{4}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} xtu^2(t)dt, \quad u\left(-\frac{\pi}{2}\right) = 0 \tag{48}$$

The exact solution is $u(x) = 1 + sinx$. By defining an auxiliary output $v(x)$ to represent the integral, Eq. (48) can be re-expressed as

$$\begin{cases} \frac{du(x)}{dx} = cosx - x + \frac{x}{4}\cdot v(\frac{\pi}{2}) \\ v(x) = \int_{-\frac{\pi}{2}}^{x} tu^2(t)dt \\ u\left(-\frac{\pi}{2}\right) = 0 \end{cases} \tag{49}$$

Thus we can use $\frac{dv(x)}{dx} = xu^2(x)$, $v\left(-\frac{\pi}{2}\right) = 0$ to constrain the relationship between $u(x)$ and $v(x)$. In the A-PINN framework, the mean square error of residuals of the governing equation MSE_f is represented as

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial u_{pred}(x_i^f;\theta)}{\partial x} - \left[\cos(x_i^f) - x_i^f + \frac{1}{4}x_i^f\, v_{pred}\left(\frac{\pi}{2};\theta\right)\right]\right|^2 \tag{50}$$

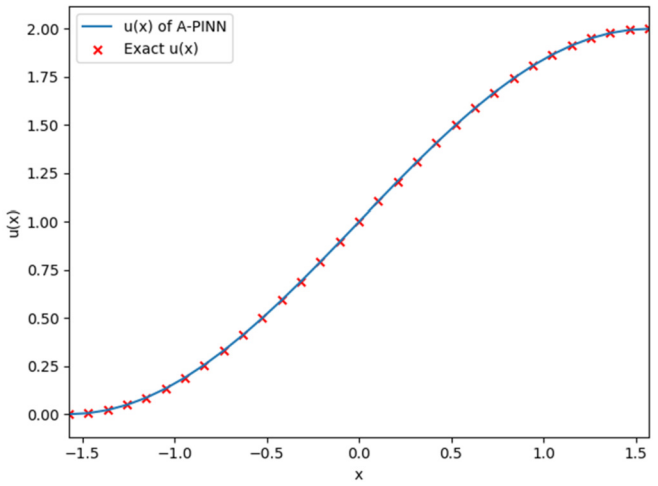

Fig. 11. The solution by A-PINN and the exact solution for 1D Fredholm IDE.

The mean square error of residuals of the initial conditions MSE_i is

$$MSE_i = \left|u_{pred}\left(-\frac{\pi}{2};\theta\right) - 0\right|^2 + \left|v_{pred}\left(-\frac{\pi}{2};\theta\right) - 0\right|^2 \tag{51}$$

The relationship between the primary output $u_{pred}\left(x_i^f;\theta\right)$ and the auxiliary output $v_{pred}\left(x_i^f;\theta\right)$ is constrained by the new output condition as

$$MSE_o = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial v_{pred}\left(x_i^f;\theta\right)}{\partial x} - x_i^f \cdot {u_{pred}}^2\left(x_i^f;\theta\right)\right|^2 \tag{52}$$

The loss function MSE_{total} in the A-PINN framework is the weighted sum of all the mean square errors. The neural network architecture considered consists of 2 hidden layers, each having 20 neurons. The training data set consists of the initial values at $x = -\frac{\pi}{2}$ and 50 collocation points evenly sampled in the equation domain. The solution by A-PINN after 500 iterations and the exact solution are presented in Fig. 11. The relative L2 error between the A-PINN solution and the exact value is 0.0477%.

4.6. The inverse problem of nonlinear 1D Volterra IDE

In this section, we will show the capability of the A-PINN framework to pursue the inverse problem of IDEs. To the best of the authors' knowledge, this is the first attempt to use physics informed machine learning to address the inverse problem of IDEs. We first consider the nonlinear 1D Volterra IDE defined in Eq. (19), where λ is now an unknown parameter. With some measurement data randomly acquired in the exact solution, we will apply A-PINN to discover the value of λ. Akin to the forward problem described in Section 4.1, we define an auxiliary output $v(x) = \int_0^x e^{t-x}u(t)dt$ to transform Eq. (19) into Eq. (20).

In the inverse problem, it is often difficult to know the exact initial condition of the equation. Most often, all we know are the governing equation and discrete measurement data. As such, we have four residual components in the A-PINN framework for the nonlinear 1D Volterra IDE. The first one is the mean square error of residuals of the measurement data, which is represented as

$$MSE_m = \frac{1}{N_m}\sum_{i=1}^{N_m}\left|u_{pred}\left(x_i^m;\theta,\lambda\right) - u_m\left(x_i^m\right)\right|^2 \tag{53}$$

where N_m is the number of measurement data, $u_{pred}\left(x_i^m;\theta,\lambda\right)$ and $u_m(x_i^m)$ are the predicted and measured values at the measurement point x_i^m. The second residual component is the mean square error of residuals of the governing equation, which is represented as (here we simply assume that the collocation points are exactly the same as the measurement points)

$$MSE_f = \frac{1}{N_m}\sum_{i=1}^{N_m}\left|\frac{\partial u_{pred}(x_i^m;\theta,\lambda)}{\partial x} + u_{pred}\left(x_i^m;\theta,\lambda\right) - v_{pred}\left(x_i^m;\theta,\lambda\right)\right|^2 \tag{54}$$

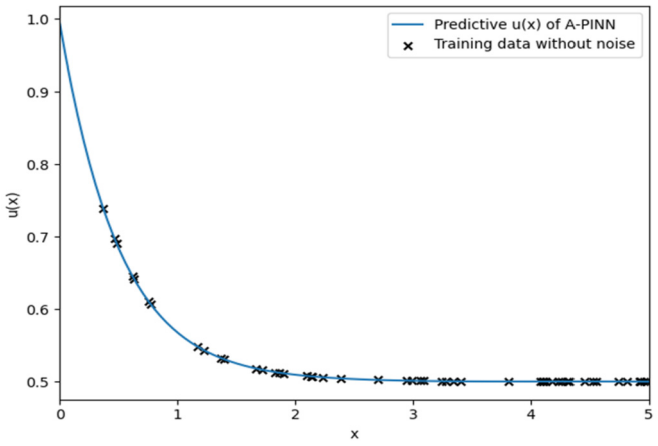

Fig. 12. The training data and the predicted values of $u(x)$ for the inverse problem of 1D Volterra IDE.

The third residual component is the mean square error of residuals of the constraint between the primary output and the auxiliary output, which is represented as (we simply assume that the collocation points are exactly the same as the measurement points)

$$MSE_o = \frac{1}{N_m}\sum_{i=1}^{N_m}\left|\frac{\partial v_{pred}(x_i^m;\theta,\lambda)}{\partial x} - u_{pred}\left(x_i^m;\theta,\lambda\right) + v_{pred}\left(x_i^m;\theta,\lambda\right)\right|^2 \tag{55}$$

The last residual component is mean square error of residuals of the initial condition of the auxiliary output, which is represented as

$$MSE_i = \left|v_{pred}\left(0;\theta\right) - 0\right|^2 \tag{56}$$

The loss function MSE_{total} is the weighted sum of all the residual components, which is expressed as

$$MSE_{total} = w_m \cdot MSE_m + w_i \cdot MSE_i + w_f \cdot MSE_f + w_o \cdot MSE_o \tag{57}$$

As explained in Section 3.4, in the case of noisy measurement data, the optimal value of MSE_m would not be 0. In this regard, we control $w_m = 1$ and calculate w_i, w_f and w_o in line with the adaptive weighting strategy. The neural network architecture for this problem consists of 2 hidden layers, each having 40 neurons. We randomly sample 50 measurement data on the exact solution of the equation as noise-free training data. In the noise-free scenario, the discovered λ is 0.999837 by A-PINN after 500 iterations in comparison with the exact value 1.0. The relative error is -0.0163%. The training data and the predicted values of $u(x)$ with the discovered λ are shown in Fig. 12.

In order to further analyze the influence of the number and the noise level of measurement data on the solution of the inverse problem, we consider different numbers of measurement data with different noise levels as training data. The noisy data is generated by adding Gaussian noise to the exact value, which is obtained by

$$u_m^{noisy}(x_i) = u_m^{exact}(x_i) \cdot (1 + noise \cdot randn) \tag{58}$$

where $u_m^{noisy}(x_i)$ represents the measurement data with noise, $u_m^{exact}(x_i)$ represents the noise-free measurement data, *noise* represents the percentage level of noise, and *randn* is a random variable with standard normal distribution. We test the number of measurement data from 5 to 500 and the noise level from 0% to 10%. The relative errors of the identified parameter λ in different scenarios are shown in Table 3. It is observed that as the number of measurement data increases, the relative error in the identified parameter decreases and more accurate value of λ is achieved. However, the increase of noise level in the measurement data results in an increase in the error of the identified parameter. Nevertheless, the following two observations are worthy of mention: (i) When using only 5 measurement data with noise level equal to or lower than 1%, the parameter λ can be accurately discovered with the relative error less than 1%. It indicates that A-PINN for inverse problems does not require a large number of measurement data; (ii) Even if the noise level is as heavy as 10%, A-PINN can still accurately identify the value of the parameter λ when 20 or more measurement data are collected. It indicates that A-PINN for parameter discovery is insensitive to noise level in the presence of sufficient measurement data.

4.7. The inverse problem of nonlinear Volterra IDEs system

Consider the nonlinear Volterra IDEs system described by Eq. (26). A-PINN is applied to discover the unknown parameters λ_1 and λ_2 with measurement data which are generated by randomly sampling the exact solution $u(x) = x + e^x$ and $v(x) =$

Table 3
Relative error in the identified λ for different numbers of measurement data and noise levels.

N_m	*noise*			
	0%	1%	5%	10%
5	−1.53E-03	3.65E-03	−2.72E-02	−3.15E-02
10	3.59E-05	7.51E-03	−2.11E-02	−2.29E-02
20	−3.62E-04	1.69E-03	2.46E-03	7.32E-03
50	−1.63E-04	2.70E-03	−1.31E-02	−2.99E-03
100	1.98E-04	−3.12E-04	−3.17E-03	6.21E-03
200	−1.07E-04	−1.75E-04	6.08E-03	−5.55E-03
500	7.72E-05	7.00E-04	3.27E-03	5.10E-03

$x - e^x$. Similar to the forward problem, we introduce auxiliary outputs $w(x)$ and $p(x)$ to represent the integrals in Eq. (26). A four-output neural network is configured to approximate $u(x)$, $v(x)$, $w(x)$, and $p(x)$. In the A-PINN framework, the mean square errors of residuals of the governing equations MSE_f are represented as (again we simply assume that the collocation points are exactly the same as the measurement points)

$$MSE_{f1} = \frac{1}{N_m}\sum_{i=1}^{N_m}\left|\frac{\partial^2 u_{pred}(x_i^m;\theta,\lambda)}{\partial x^2} - 1 + \frac{{x_i^m}^3}{3} + \frac{1}{2}\frac{\partial v_{pred}\left(x_i^m;\theta,\lambda\right)}{\partial x} - \lambda_1 w_{pred}\left(x_i^m;\theta,\lambda\right)\right|^2 \tag{59}$$

$$MSE_{f2} = \frac{1}{N_m}\sum_{i=1}^{N_m}\left|\frac{\partial^2 v_{pred}(x_i^m;\theta,\lambda)}{\partial x^2} + 1 - {x_i^m}^2 + x_i^m u_{pred}\left(x_i^m;\theta,\lambda\right) - \lambda_2 p_{pred}\left(x_i^m;\theta,\lambda\right)\right|^2 \tag{60}$$

The mean square error of residuals of the measurement data is represented as

$$MSE_m = \frac{1}{N_m}\sum_{i=1}^{N_m}\left|u_{pred}\left(x_i^m;\theta,\lambda\right) - u_m\left(x_i^m\right)\right|^2 + \frac{1}{N_m}\sum_{i=1}^{N_m}\left|v_{pred}\left(x_i^m;\theta,\lambda\right) - v_m\left(x_i^m\right)\right|^2 \tag{61}$$

The mean square error of residuals of the initial conditions of the auxiliary outputs is represented as

$$MSE_i = \left|w_{pred}\left(0;\theta,\lambda\right) - 0\right|^2 + \left|p_{pred}\left(0;\theta,\lambda\right) - 0\right|^2 \tag{62}$$

The relationships between the primary outputs $u(x)$, $v(x)$ and the auxiliary outputs $w(x)$, $p(x)$ are constrained by two new output conditions, and the mean square errors of residuals of these two output conditions are (we simply assume that the collocation points are exactly the same as the measurement points)

$$MSE_{o1} = \frac{1}{N_m}\sum_{i=1}^{N_m}\left|\frac{\partial w_{pred}\left(x_i^m;\theta,\lambda\right)}{\partial x} - {u_{pred}}^2\left(x_i^m;\theta,\lambda\right) - {v_{pred}}^2\left(x_i^m;\theta,\lambda\right)\right|^2 \tag{63}$$

$$MSE_{o2} = \frac{1}{N_m}\sum_{i=1}^{N_m}\left|\frac{\partial p_{pred}\left(x_i^m;\theta,\lambda\right)}{\partial x} - {u_{pred}}^2\left(x_i^m;\theta,\lambda\right) + {v_{pred}}^2\left(x_i^m;\theta,\lambda\right)\right|^2 \tag{64}$$

The loss function MSE_{total} is the weighted sum of all the residual components, which is expressed as

$$MSE_{total} = w_m \cdot MSE_m + w_i \cdot MSE_i + w_f \cdot (MSE_{f1} + MSE_{f2}) + w_o \cdot (MSE_{o1} + MSE_{o2}) \tag{65}$$

We set $w_m = 1$ and calculate w_i, w_f and w_o according to the adaptive weighting strategy. A DNN with 2 hidden layers and 40 neurons in each hidden layer is configured to solve this inverse problem. We randomly sampled 50 measurement data on the exact solution of the equations. By A-PINN, the discovered values of the parameters $[\lambda_1, \lambda_2]$ are $[0.503, 0.245]$ after 500 iterations, with the relative errors with the exact values $[0.5, 0.25]$ being $[0.6\%, 2\%]$. The training data and the predicted values of $u(x)$ and $v(x)$ are shown in Fig. 13.

5. Conclusions

In this paper, we presented the auxiliary physics informed neural network (A-PINN) framework for solving the forward and inverse problems of nonlinear IDEs. In the proposed A-PINN, a multi-output neural network is configured to simultaneously represent the primary variables and integrals in the governing equations. By pursuing automatic differentiation of the auxiliary outputs in lieu of the integral operators in IDEs, we bypass the limitation of neural networks in dealing with integral manipulation. As integral discretization is avoided, A-PINN doesn't suffer from discretization and truncation errors

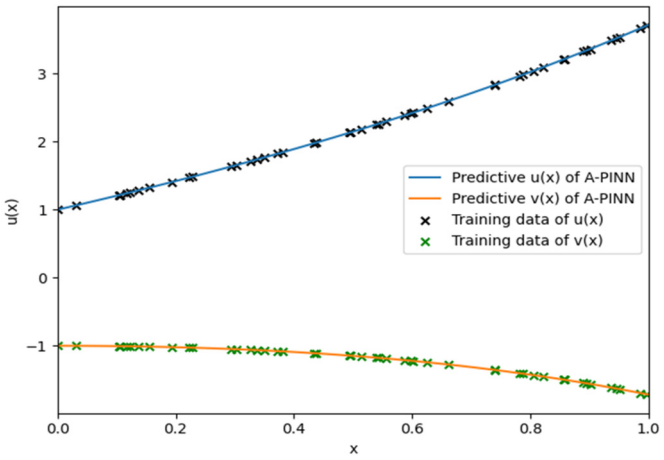

Fig. 13. Training data and predicted values of $u(x)$ and $v(x)$ for the inverse problem of Volterra IDEs system.

for forward and inverse solution of IDEs. Because of being devoid of fixed grids or nodes, A-PINN is a mesh-free method that can calculate/predict the solution at any point in the equation domain without interpolation.

By pursuing a benchmark problem of 1D Volterra IDE, we validated that the proposed A-PINN can achieve more accurate solution than the PINN adopting integral discretization. Through several numerical experiments, we further demonstrated that A-PINN can accurately solve forward problems involving nonlinear 1D IDEs system, nonlinear 2D IDE, high-dimensional (10D) nonlinear IDE, and Fredholm IDE. With few modifications to the A-PINN framework for forward problems, we successfully applied A-PINN to discover unknown parameters in IDEs, and showed that A-PINN has the capability to accurately identify unknown parameters in IDEs even when the measurement data are heavily polluted by noise.

CRediT authorship contribution statement

Lei Yuan: Conceptualization, Methodology, Software, Investigation, Coding, Data curation, Writing – original draft. **Yi-Qing Ni**: Conceptualization, Methodology, Supervision, Project administration, Funding acquisition, Writing – review & editing. **Xiang-Yun Deng**: Investigation, Supervision, Data curation, Writing – review & editing. **Shuo Hao**: Investigation, Coding, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The work described in this paper was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (SAR), China (Grant No. R5020-18) and a grant from The Hong Kong Polytechnic University (Grant No. 1-YW5H). The authors also appreciate the funding support by the Innovation and Technology Commission of Hong Kong SAR Government to the Hong Kong Branch of National Engineering Research Center on Rail Transit Electrification and Automation (Grant No. K-BBY1).

References

[1] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[2] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations, arXiv preprint, arXiv:1711.10561, 2017.
[3] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2018) 1–43.
[4] S. Cai, Z. Mao, Z. Wang, M. Yin, G.E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: a review, Acta Mech. Sin. (2022), https://doi.org/10.1007/s10409-021-01148-1.
[5] Q. He, D. Barajas Solano, G. Tartakovsky, A.M. Tartakovsky, Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport, Adv. Water Resour. 141 (2020) 103610.
[6] S. Falas, C. Konstantinou, M.K. Michael, Physics-informed neural networks for securing water distribution systems, arXiv preprint, arXiv:2009.08842, 2020.
[7] J.C. Wong, C. Ooi, P.H. Chiu, M.H. Dao, Improved surrogate modeling of fluid dynamics with physics-informed neural networks, arXiv preprint, arXiv: 2105.01838, 2021.

[8] T. Kadeethum, T.M. Jørgensen, H.M. Nick, Physics-informed neural networks for solving inverse problems of nonlinear Biot's equations: batch training, in: 54th US Rock Mechanics/Geomechanics Symposium, American Rock Mechanics Association, 2020.
[9] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, Comput. Methods Appl. Mech. Eng. 360 (2020) 112789.
[10] Q. Zhu, Z. Liu, J. Yan, Machine learning for metal additive manufacturing: predicting temperature and melt pool fluid dynamics using physics-informed neural networks, Comput. Mech. 67 (2021) 619–635.
[11] A. Arzani, J.X. Wang, R.M. D'Souza, Uncovering near-wall blood flow from sparse data with physics-informed neural networks, Phys. Fluids 33 (2021) 071905.
[12] F. Sahli Costabal, Y. Yang, P. Perdikaris, D.E. Hurtado, E. Kuhl, Physics-informed neural networks for cardiac activation mapping, Front. Phys. 8 (2020) 42.
[13] H. He, J. Pathak, An unsupervised learning approach to solving heat equations on chip based on auto encoder and image gradient, arXiv preprint, arXiv:2007.09684, 2020.
[14] S. Cai, Z. Wang, S. Wang, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks for heat transfer problems, J. Heat Transf. 143 (2021) 060801.
[15] R. Laubscher, Simulation of multi-species flow and heat transfer using physics-informed neural networks, Phys. Fluids 33 (2021) 087101.
[16] Y. Chen, L. Lu, G.E. Karniadakis, L. Dal Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, Opt. Express 28 (2020) 11618–11633.
[17] S. Goswami, C. Anitescu, S. Chakraborty, T. Rabczuk, Transfer learning enhanced physics informed neural network for phase-field modeling of fracture, Theor. Appl. Fract. Mech. 106 (2020) 102447.
[18] E. Zhang, M. Yin, G.E. Karniadakis, Physics-informed neural networks for nonhomogeneous material identification in elasticity imaging, arXiv preprint, arXiv:2009.04525, 2020.
[19] M. Yin, X. Zheng, J.D. Humphrey, G.E. Karniadakis, Non-invasive inference of thrombus material properties with physics-informed neural networks, Comput. Methods Appl. Mech. Eng. 375 (2021) 113603.
[20] R. Zhang, Y. Liu, H. Sun, Physics-informed multi-LSTM networks for metamodeling of nonlinear structures, Comput. Methods Appl. Mech. Eng. 369 (2020) 113226.
[21] E. Haghighat, A.C. Bekar, E. Madenci, R. Juanes, Deep learning for solution and inversion of structural mechanics and vibrations, arXiv preprint, arXiv: 2105.09477, 2021.
[22] E. Haghighat, M. Raissi, A. Moure, H. Gomez, R. Juanes, A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics, Comput. Methods Appl. Mech. Eng. 379 (2021) 113741.
[23] U. bin Waheed, E. Haghighat, T. Alkhalifah, C. Song, Q. Hao, Eikonal solution using physics-informed neural networks, arXiv preprint, arXiv:2007.08330, 2020.
[24] Q. Zhang, Y. Chen, Z. Yang, Data-driven solutions and discoveries in mechanics using physics informed neural network, Preprints, 2020060258, https://doi.org/10.20944/preprints202006.0258.v1, 2020.
[25] A.D. Jagtap, E. Kharazmi, G.E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: applications to forward and inverse problems, Comput. Methods Appl. Mech. Eng. 365 (2020) 113028.
[26] X. Meng, Z. Li, D. Zhang, G.E. Karniadakis, PPINN: parareal physics-informed neural network for time-dependent PDEs, Comput. Methods Appl. Mech. Eng. 370 (2020) 113250.
[27] A. Jagtap, G. Karniadakis, 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 (2020) 2002–2041.
[28] J. Huang, H. Wang, T. Zhou, An augmented Lagrangian deep learning method for variational problems with essential boundary conditions, Commun. Comput. Phys. 31 (2022) 966–986.
[29] Y. Liao, P. Ming, Deep Nitsche method: deep Ritz method with essential boundary conditions, Commun. Comput. Phys. 29 (2021) 1365–1384.
[30] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, arXiv preprint, arXiv: 1912.00873, 2019.
[31] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, G. Wetzstein, Implicit neural representations with periodic activation functions, in: NeurIPS 2020, in: Advances in Neural Information Processing Systems, vol. 33, 2020.
[32] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. 404 (2020) 109136.
[33] H. Gao, L. Sun, J.X. Wang, PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain, J. Comput. Phys. 428 (2021) 110079.
[34] R. Rodriguez-Torrado, P. Ruiz, L. Cueto-Felgueroso, M.C. Green, T. Friesen, S. Matringe, J. Togelius, Physics-informed attention-based neural network for solving non-linear partial differential equations, arXiv preprint, arXiv:2105.07898, 2021.
[35] C.L. Wight, J. Zhao, Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks, Commun. Comput. Phys. 29 (2021) 930–954.
[36] M.A. Nabian, R.J. Gladstone, H. Meidani, Efficient training of physics-informed neural networks via importance sampling, Comput.-Aided Civ. Infrastruct. Eng. 36 (2021) 962–977.
[37] G. Pang, L. Lu, G.E. Karniadakis, fPINNs: fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (2019) A2603–A2626.
[38] E. Kharazmi, Z. Zhang, G.E.M. Karniadakis, hp-VPINNs: variational physics-informed neural networks with domain decomposition, Comput. Methods Appl. Mech. Eng. 374 (2021) 113547.
[39] A.F. Psaros, K. Kawaguchi, G.E. Karniadakis, Meta-learning PINN loss functions, J. Comput. Phys. 458 (2022) 111121.
[40] L. McClenny, U. Braga-Neto, Self-adaptive physics-informed neural networks using a soft attention mechanism, arXiv preprint, arXiv:2009.04544, 2020.
[41] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (2021) A3055–A3081.
[42] S. Wang, X. Yu, P. Perdikaris, When and why PINNs fail to train: a neural tangent kernel perspective, J. Comput. Phys. 449 (2022) 110768.
[43] Z. Xiang, W. Peng, X. Zheng, X. Zhao, W. Yao, Self-adaptive loss balanced physics-informed neural networks for the incompressible Navier-Stokes equations, arXiv preprint, arXiv:2104.06217, 2021.
[44] L. Yang, D. Zhang, G.E. Karniadakis, Physics-informed generative adversarial networks for stochastic differential equations, SIAM J. Sci. Comput. 42 (2020) A292–A317.
[45] D. Zhang, L. Lu, L. Guo, G.E. Karniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, J. Comput. Phys. 397 (2019) 108850.
[46] X. Chen, L. Yang, J. Duan, G.E. Karniadakis, Solving inverse stochastic problems from discrete particle observations using the Fokker-Planck equation and physics-informed neural networks, arXiv preprint, arXiv:2008.10653, 2020.
[47] X. Meng, L. Yang, Z. Mao, J.A. Ferrandis, G.E. Karniadakis, Learning functional priors and posteriors from data and physics, J. Comput. Phys. 457 (2022) 111073.
[48] L. Yang, X. Meng, G.E. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, J. Comput. Phys. 425 (2021) 109913.

[49] X. Meng, H. Babaee, G.E. Karniadakis, Multi-fidelity Bayesian neural networks: algorithms and applications, J. Comput. Phys. 438 (2021) 110361.
[50] Y. Yang, P. Perdikaris, Adversarial uncertainty quantification in physics-informed neural networks, J. Comput. Phys. 394 (2019) 136–152.
[51] D. Zhang, L. Guo, G.E. Karniadakis, Learning in modal space: solving time-dependent stochastic PDEs using physics-informed neural networks, SIAM J. Sci. Comput. 42 (2020) A639–A665.
[52] L. Guo, H. Wu, T. Zhou, Normalizing field flows: solving forward and inverse stochastic differential equations using physics-informed flow models, arXiv preprint, arXiv:2108.12956, 2021.
[53] L. Guo, H. Wu, X. Yu, T. Zhou, Monte Carlo PINNs: deep learning approach for forward and inverse problems involving high dimensional fractional partial differential equations, arXiv preprint, arXiv:2203.08501, 2022.
[54] J. Bélair, M.C. Mackey, Consumer memory and price fluctuations in commodity markets: an integrodifferential model, J. Dyn. Differ. Equ. 1 (1989) 299–325.
[55] E. Voltchkova, R. Cont, Integro-differential equations for option prices in exponential Lévy models, Finance Stoch. 9 (2005) 299–325.
[56] R. Ansari, K. Hosseini, A. Darvizeh, B. Daneshian, A sixth-order compact finite difference method for non-classical vibration analysis of nanobeams including surface stress effects, Appl. Math. Comput. 219 (2013) 4977–4991.
[57] N. Apreutesei, A. Ducrot, V. Volpert, Travelling waves for integro-differential equations in population dynamics, Discrete Contin. Dyn. Syst., Ser. B 11 (2009) 541.
[58] A.A. Minakov, C. Schick, Integro-differential equation for the non-equilibrium thermal response of glass-forming materials: analytical solutions, Symmetry 13 (2021) 256.
[59] D. Sidorov, I. Muftahov, N. Tomin, D. Karamov, D. Panasetsky, A. Dreglea, F. Liu, A. Foley, A dynamic analysis of energy storage with renewable and diesel generation using Volterra equations, IEEE Trans. Ind. Inform. 16 (2019) 3451–3459.
[60] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: a deep learning library for solving differential equations, SIAM Rev. 63 (2021) 208–228.
[61] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, Pytorch: an imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, vol. 32, 2019, pp. 8026–8037.
[62] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in Pytorch, in: 31st Conference on Neural Information Processing Systems, 2017.
[63] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, Tensorflow: a system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation, 2016, pp. 265–283.
[64] Y. Wang, S.S. Ezz-Eldien, A.A. Aldraiweesh, A new algorithm for the solution of nonlinear two-dimensional Volterra integro-differential equations of high-order, J. Comput. Appl. Math. 364 (2020) 112301.