PINN-TI: Physical Information embedded in Neural Networks for solving ordinary differential equations with Time-varying Inputs

Xiaotong Zhao
Xidian University
jingli du (jldu@mail.xidian.edu.cn)
Xidian University
YeTong Shi
Xidian University
Kunpeng Zhao
Xidian University

Research Article

Keywords: Solving differential equations, Time-varying input, Physical information

Posted Date: January 11th, 2024

DOI: https://doi.org/10.21203/rs.3.rs-3830736/v1

Additional Declarations: No competing interests reported.

PINN-TI: Physical Information embedded in Neural Networks for solving ordinary differential equations with Time-varying Inputs

Xiaotong Zhao[a] (1015502154@qq.com), Jingli Du[a*](jldu@mail.xidian.edu.cn), Yetong Shi[a], Kunpeng Zhao[a]

[a] *Key Laboratory of Electronic Equipment Structure Design of Ministry of Education, Xidian University, Xi'an, Shaanxi, 710071, China*

Abstract:

Modeling and predicting the dynamics of multiphysics and multiscale systems with hidden physics methods is often costly, requiring different formulations and complex computer codes. There has been significant progress in solving differential equations using physical information embedded in neural networks, but solving and generalizing for differential equations with time-varying inputs is still an open problem. In this paper, we design a PINN-TI network structure and a Step-by-Step Forward (SSF) algorithm. It enables the network to predict the results of ordinary differential equations under different initial values or boundary conditions and different time-varying inputs (including segmented functions) after one training. Our approach can extend the time domain of the solution to a larger range. We validated our model using a variant of Vander Pol's equation, which contains one or two external inputs. The results show that our network can handle the prediction and generalization problem of ordinary differential equations under time-varying inputs well. Our method provides a new solution for solving and generalizing differential equations under time-varying inputs.

Keywords: Solving differential equations, Time-varying input, Physical information

1. Introduction

Differential equations are essential for modeling the dynamics of complex systems in both the spatial and time domains [1, 2]. Methods for solving Ordinary Differential Equations (ODEs) or Partial Differential Equations (PDEs) using classical analytical or computational tools, such as finite difference, finite elements, spectral, and even meshless methods, are well established. However, there are still many problems in modeling and predicting the evolution of nonlinear multiscale systems with inhomogeneous cascades-of-scales, such as high computational cost, difficulty in solving the noise problem, inability to fuse the observed data, and poor solution results under higher-order problems [3, 4]. In contrast to conventional numerical algebraic solvers, learning-based neural solvers (physics-driven deep learning methods) can integrate observational data and physical information. These neural solvers combine physics with the flexibility of deep learning by building neural networks in domain knowledge. Neural solvers have been applied to solve differential equations in various fields, including computational fluid dynamics [5, 6], thermodynamic prediction [7, 8], rigid chemical kinetics, Eikonal equations, and so on [9-11]. In the modeling of mechanical dynamics, continuous time can also be modeled by neural ordinary differential equations [10, 12].

Physics-based learning integrates noisy data and mathematical models and is implemented through neural networks or other kernel-based regression networks [13, 14]. Relevant outstanding research results mainly include Physics-informed neural networks (PINNs) [3], Deep operator networks (DeepONet) [15], and Fourier Neural Operator (FNO) [16, 17]. There are some significant common features of these models, i.e., they incorporate a lot of existing and correct prior knowledge (e.g., physical principles, constraints, symmetries, computational simulations), which makes the models more accurate, robust, reliable, and explanatory [18]. These methods seamlessly integrate data and abstract mathematical operators, including ODEs or PDEs with or without missing physical information [19]. With this prior knowledge constraint, deep learning is more interpretable and can be robust in the presence of imperfect data (e.g., omissions or noisy values, outliers), even for extrapolation or generalization tasks. Other relevant studies include RK-NET [20], Vandermonde Neural Operators (VNO) [21], and Deep CORAL [22].

Although the above methods have made a great contribution to the solution of differential equations, they still have significant limitations in specific applications. The PINN method requires retraining after replacing initial values or

boundary conditions and cannot handle the generalization problem of solving differential equations with time-varying inputs [23]. DeepONet belongs to the category of operator learning, and its branch network achieves generalization under different inputs by sampling time-varying inputs at fixed sensor locations. However, for input fluctuations under non-sensor locations do not affect the final result output of DeepONet. The FNO method implements operator learning via Fourier transform and Fourier inverse transform, but it is also incapable of handling the problem of stable generalization to systems containing time-varying inputs.

We named our model as PINN-TI, signifying the use of the idea of Physical Information embedded in Neural Networks to solve ordinary differential equations with Time-varying Inputs. PINN-TI extends the application of neural networks to solve differential equations with time-varying external inputs. It is not only able to fit the solution of differential equations with different initial values for any time-invariant input, but also able to solve differential equations with time-varying external inputs using only the training data of differential equations with time-invariant inputs. The advantage of our model is that the trained network can solve differential equations with arbitrary initial values, boundary conditions, and arbitrary time-varying input, which are not possible with existing PINNs and DeepONet. Compared to DeepONet, our model is equally sensitive to segmented time-varying inputs. This property helps to apply the model to control areas such as vibration suppression control, model predictive control, etc.

The loss functions of the PINN-TI model include differential equation structure loss, boundary or initial value condition loss, recursive loss, and measurement data loss. Compared to the loss function structure of PINN, our model adds a recursive loss, which is set for our proposed Step-by-Step Forward (SSF) algorithm, which solves the differential equation with time-varying inputs by step-by-step iteration. The SSF method is inspired by traditional numerical iterative solution methods. The SSF method can predict the results of differential equations over a time range much larger than the time range set during training. We verified our algorithm using two ordinary differential equations. The second example of an ordinary differential equation contains two external time-varying inputs. The results show that our network can well handle the problem of differential equation prediction and generalization under time-varying inputs.

The main structure of the article is as follows, in section 2, we introduce the fundamentals of PINN and DeepONet. In section 3, we design a network structure and loss function which can be generalized to the solution of differential equations under different initial conditions, different constant values of external inputs or arbitrary time-varying external inputs after a single training session. Section 4 is the experimental section, we used two examples to validate our model and algorithm. Section 5 and 6 present the results and discussion, respectively.

2. Preliminary

2.1 Original PINNs

PINNs can integrate information from measurements and the partial differential equations themselves by embedding partial differential equations into the loss function of neural networks using automatic differentiation [2, 24, 25]. This method can be applied to integer-order PDEs, integro-differential equations, fractional PDEs or stochastic PDEs [19]. More information on PINNs can be found in Ref. [3]。PINN is mainly used for data-driven solutions of general form PDEs (1). We will only introduce the continuous time model here.

$$y_t + \mathrm{N}[y] = 0, x \in \Omega, t \in [0, T], \tag{1}$$

where, $y(t,x)$ denotes the hidden solution, $\mathrm{N}(\cdot)$ is a nonlinear differential operator, and Ω is a subset of R^D. In a simpler case, y can be simply a function of time t, independent of x, represented as $y(t)$, which is simulated in *Example 1* of Section 4.

Construct $f(t,x)$ by the equation:

$$f := y_t + \mathrm{N}[y], \tag{2}$$

where, $y(t,x)$ is approximated through multilayer perceptrons or other forms of deep neural networks, $\mathrm{N}[y]$ can be derived by applying the chain rule for differentiating compositions of functions using automatic differentiation.

$y(t,x)$ and $\mathrm{N}[y]$ share network parameters, such as weights and biases, but have different activation functions due to the presence of differential operators N. The update of network parameters is achieved by minimizing mean square error (MSE) loss,

$$MSE = MSE_y + MSE_f\,, \tag{3}$$

where,

$$MSE_y = \frac{1}{N_y}\sum_{i=1}^{N_y} |\, y(t_y^i, x_y^i) - y^i \,|^2, \tag{4}$$

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |\, f(t_f^i, x_f^i) \,|^2\,, \tag{5}$$

where, the calculation of the loss MSE_y depends on the initial or boundary training data $\{t_u^i, x_u^i, y^i\}_{i=1}^{N_u}$, the loss MSE_f enforces the structure imposed by equation (1) at a finite set of collocation points $\{t_f^i, x_f^i\}_{i=1}^{N_f}$, which is specify the collocations points for $f(t,x)$. Some variants of PINNs have increased the fusion of measurement data, which adds MSE_m to the total loss,

$$MSE_m = \frac{1}{N_m}\sum_{i=1}^{N_m} |\, y(t_y^i, x_y^i) - y_m(t_y^i, x_y^i) \,|^2 \tag{6}$$

where, $y_m(t_y^i, x_y^i)$ represents the actual measured value at $\{t_y^i, x_y^i\}$. The loss of MSE_m represents the loss between the measurement data and the neural network fitting data. Introducing this loss can help neural networks learn random errors, or parts of the system that cannot be modeled or are not accurately modeled.

2.2 DeepONet

DeepONet can learn parameterized PDEs, ODEs, convective equations, diffusion reaction equations, and a host of other physics-driven Maxwell equation solvers. DeepONets have been demonstrated as a powerful tool to learn nonlinear operators in a supervised data-driven manner [26-28]. DeepONet define that G represents the operator and $G(u)$ represents the output function of the input function u. The output $G(u)(y)$ is a real number, where y is any point in the domain of $G(u)$. The idea of DeepONet is similar to the Universal Approximation Theorem for Operator

$$\left| G(u)(y) - \sum_{k=1}^{p}\sum_{i=1}^{n} c_i^k \sigma\Bigg(\underbrace{\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k}_{branch}\Bigg) \underbrace{\sigma(\omega_k \cdot y + \zeta_k)}_{trunk} \right| < \varepsilon\,, \tag{7}$$

where, $\varepsilon > 0$, σ is a continues non-polynomial function, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathrm{R}$, $\omega_k \in \mathrm{R}^d$.

The generalized universal approximation theorem for operators generalizes Eq. (7) to

$$\left| G(u)(y) - \Big\langle \underbrace{\mathbf{g}(u(x_1), u(x_2), \cdots, u(x_m))}_{branch}, \underbrace{\mathbf{f}(y)}_{trunk} \Big\rangle \right| < \varepsilon\,, \tag{8}$$

where, $\mathbf{g}(\cdot)$ and $\mathbf{f}(\cdot)$ represents continues vector functions, which can be represented by multilayer perceptron, convolutional neural network, etc. From Eq. (8), we can see that the architecture of DeepONet can be divided into two parts: branch and trunk. A large number of different input functions u are required as training inputs in branch section, which chooses two function spaces Gaussian random field (GRF)

$$u:\ \mathrm{G}(0, k_l(x_1, x_2))\,, \tag{9}$$

where the convariance kernel $k_l(x_1, x_2) = \exp(-\| x_1 - x_2 \|^2 / 2l^2)$ is the radial-basis function (RBF) kernel with a length-scale parameter $l > 0$, and Chebyshev polynomials

$$u:\ V_{poly}=\left\{\sum_{i=0}^{N-1}a_iT_i(x):|\ a_i\ |\le M\right\}, \tag{10}$$

where, T_i are Chebyshev polynomials of the first kind, and $M>0$. Sensors $[x_1,x_2,...,x_m]$ represent the sampling point in the input function u, i.e., using $[u(x_1),u(x_2),...,u(x_m)]$ instead of u. More detailed explanations can be found in [15].

2.3 Deficiencies

The original PINNs algorithm can only fit the solution process of a differential equation under certain initial conditions, and it trains the network by minimizing the loss through gradient-based optimizers, such as Adam [29] and L-BFGS [30], until the loss is less than a threshold ε. If the initial conditions of the differential equation change, the network needs to be retrained, and the PINNs do not have the ability to generalize. For a constant input function u or constant U, it is possible to solve the differential equation by encoding it equivalently to the initial conditions of the differential equation with only a few adjustments to the structure of the original PINNs. At the same time, by encoding the initial conditions into the inputs of the PINNs network, it is possible to solve the problem of achieving generalization with different initial conditions. The external inputs to systems in a large number of physical control domains are often time-varying, and PINNs are not able to generalize time-varying external inputs, or rather, they can only be solved for some definite time-varying external input.

DeepONet is an operator learning that can solve differential equations under different time-varying inputs. The training process requires collecting results under different basis functions, which are represented by sensor information under a fixed position. One fatal flaw in this network structure is its inability to handle input function variations at non-sensor locations. Although the input function domain can be covered as much as possible by encrypting the location of the sensor, it increases the computer arithmetic consumption super-linearly and does not fundamentally address the insensitivity to input fluctuations at non-sensors. DeepONet works ineffectively when dealing with external inputs as a segmented function, because sampling based on sensor position tends to lose a lot of information. One of the fatal drawbacks of DeepONet is that the positions of the sensors in its training and usage phases cannot be varied, which results in the inability of the DeepONet network to scale the results to a longer duration of the input function.

3. PINN-TI

3.1 PINN-TI Network structure

The ordinary differential equation with continuous time-varying input can be expressed as:

$$y_t+\mathrm{N}[y]=u(t),t\in[0,T], \tag{11}$$

where, $u(t)$ represents the time-varying external input. In contrast to the DeepONet method where the generalization problem of solving differential equations with time-varying inputs is achieved by arranging a large number of sensors to sample under multiple input basis functions, our network only needs to sample the results of differential equations under constant external inputs U during the training phase. The differential equation with constant external input can be expressed as:

$$y_t+\mathrm{N}[y]=U,t\in[0,T], \tag{12}$$

We first start by constructing a fully-connected neural network with an output $\hat{y}$ that will be used as a proxy model for solving differential equations. In PINN, the ODE or PDE information is encoded as constraints into the loss function of the neural network for training, and the initial and boundary conditions do not appear in the input data of the neural network. In contrast to PINN, boundary conditions, initial conditions, and constant inputs U are added to the inputs of our network, which is equivalent to expanding the input dimension of the network. The input to our network can be expressed as $\{Y_0,t\}$, Y_0 is the set of initial conditions, boundary conditions.

$$Y_0 = \{y_0, y_0', y_0'', ...\} \tag{13}$$

where, y_0 represents the initial or boundary condition for each x-dimension at $t = 0$, y_0' and y_0'' represent the set of initial conditions for all first and second order derivatives at $t = 0$, respectively. The network structure of PINN-TI is shown in Fig. 1.

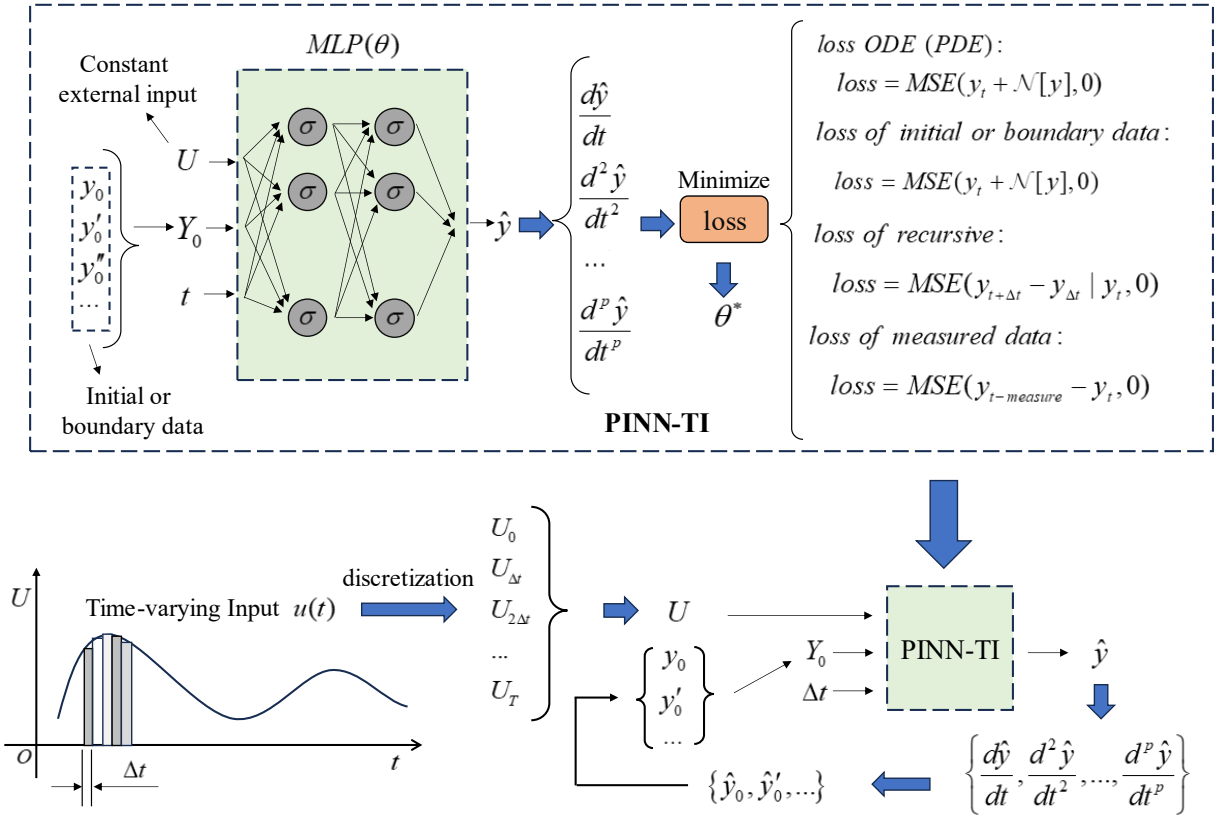

Fig. 1. The network structure of PINN-TI. The external input U to the neural network during the training phase is a constant. When the external input is time-varying $u(t)$, the solution of the differential equation is realized by calling the trained net PINN-TI.

By introducing Y_0 into the input part of the neural network, the trained neural network can fit solutions to differential equations with constant external time-invariant input U for arbitrary initial values or boundary conditions. Derivatives of compositions involving differentiable functions can be found using the chain rule. Therefore, when we assume that the current moment is the $t = 0$ moment, based on the existing network structure and network parameters, the corresponding initial values and boundary conditions in the current state can be derived. In contrast to the DeepONet method, our proposed solution to the problem of solving differential equations with arbitrary time-varying inputs is to discretize the time-varying inputs and use multiple constant inputs for equivalence. Using Δt as the time window or sampling step, the corresponding initial values and initial boundary conditions are updated sequentially, while the time-invariant input U is updated using the results of discretization of the time-varying input $u(t)$ at the global time. This enables the prediction of the results of the time-varying input $u(t)$ using a network trained on the constant external input U. A schematic of the process is shown in Fig. 2.

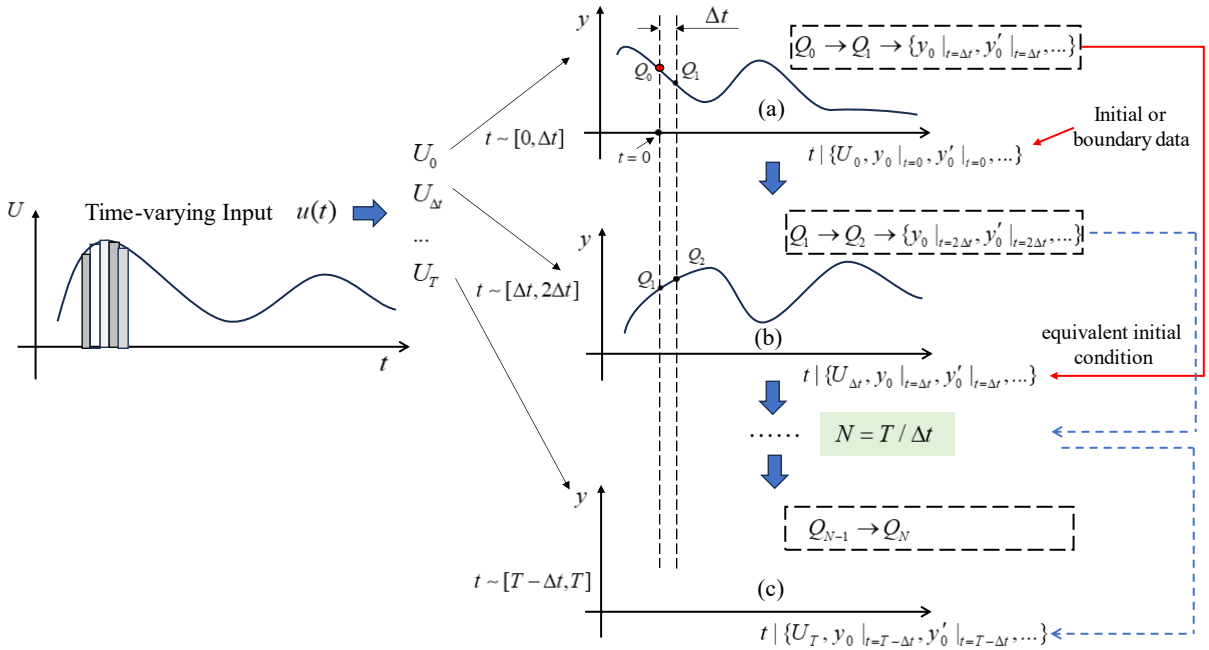

Fig. 2. Solution procedure for differential equations with time-varying external inputs.

In Fig. 2(a), the Q_0 point represents the y-value at the moment $t=0$. The curve represents the relationship of y with time t under $\{U_0, Y_0|_{t=0}\}$, and point Q_0 is transferred to point Q_1 after Δt moment. Fig. 2(b) represents the relationship between y and time t for the initial value of point Q_1 under the action of $U_{\Delta t}$ and $\{U_{\Delta t}, Y_0|_{t=\Delta t}\}$. Continue the above operation until the end of the whole simulation cycle. For ease of description, we name this method as the Step-by-Step Forward (SSF) algorithm. The pseudo-code of the SSF algorithm is shown in Algorithm 1. The SSF method is also applicable to the solution of differential equations under time-invariant inputs. Although it is possible to directly solve differential equations with time-invariant inputs by changing the value of t, the prediction range of t is often limited due to the presence of the activation function. The SSF method predicts at any moment with a smaller prediction step size, which can significantly expand the prediction range.

Algorithm 1: Step-by-Step Forward

Initialize Δt, Y_0, T, and a trained PINN-TI network *NET*

Generate external inputs $u(t)$

Discretize $u(t)$ as $[U_0, U_{\Delta t}, ..., U_T]$

$Y_0 \leftarrow Y_0$

for $i = 1:N$

$y = NET(U_{i\Delta t}, Y_0, \Delta t)$

$Y_0 \leftarrow [\frac{dy}{dt}, \frac{d^2y}{dt^2}, ...]$

$y_{out}(i) = y$

end for

Our approach remains applicable to the problem of solving ordinary differential equations when the external inputs to the system are high-dimensional. Consider a more general form for ordinary differential equations with high-dimensional external time-varying inputs:

$$G\left(y_t, \mathrm{N}[y], u_p(t)\right) = 0, t \in [0, T], \tag{14}$$

where, $u_p(t)$ represents the *p*-th external time-varying input. In the training phase, the U to be sampled contains $(U^1, U^2, ..., U^p)$. When solving with the SSF algorithm, it is only necessary to discretize $u_p(t)$ separately for each dimension.

3.2 Data generation

The data generation phase is similar to PINN data generation in that only U, Y_0, and t need to be sampled and no computation of y is required. To make the neural network more responsive to the differential equations themselves, Δt is also performed using sampling. In this way, the final trained network can be adapted to the prediction problem at different time steps.

3.3 Loss function of PINN-TI

The loss function of PINN-TI consists of four main parts, the loss of the structure of the differential equation $loss_d$, the loss of the boundary or initial value condition $loss_b$, the recursive loss $loss_r$ and the loss of the measured data $loss_m$.

$$loss = \lambda_1 loss_d + \lambda_2 loss_b + \lambda_3 loss_r + \lambda_4 loss_m \tag{15}$$

where, $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ represents the weights between the loss functions.

The structure of the differential equation can be expressed as

$$f_d(y, t, U; \frac{d\hat{y}}{dt}, \frac{d^2\hat{y}}{dt^2}, ...) = 0 \tag{16}$$

where, y is a function of t, and U is the input function.

Then $loss_d$ is calculated as follows:

$$loss_d = \frac{1}{n}\sum_{i=1}^{n}\left\| f_d^i - \mathbf{0} \right\|_2^2 \tag{17}$$

where, n is the number of sampled data.

The boundary or initial value condition constraints can be expressed as:

$$f_b(y, t_c, U; \frac{d\hat{y}}{dt}, \frac{d^2\hat{y}}{dt^2}, ...) = 0, x \in \Omega \tag{18}$$

where, $t_c = 0$ represents the boundary or initial condition constraints.

Then $loss_b$ is calculated as follows:

$$loss_b = \frac{1}{n_b}\sum_{i=1}^{n}\left\| f_b^i - \mathbf{0} \right\|_2^2 \tag{19}$$

where, n_b is the number of data sampled at the initial value and boundary conditions.

The recursive constraint is the difference between using the neural network to directly predict the outcome at $t + \Delta t$ and using the neural network first to predict the outcome at moment t, updating the initial values and boundary conditions and then further predicting the output after the Δt moment.

$$y \mid (U, Y_0, t + \Delta t) = y \mid (U, Y_t, \Delta t) \tag{20}$$

Fig. 3. illustrates the computation of recursive loss.

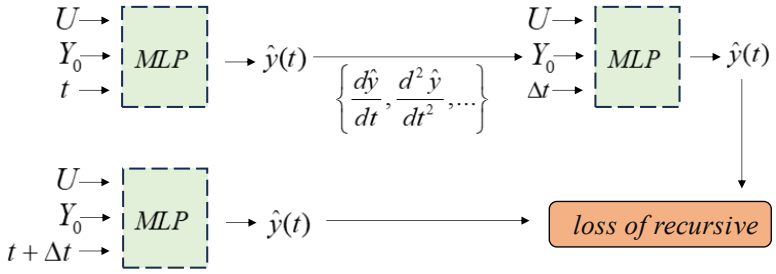

Fig. 3. Calculation Process of the recursive loss.

Then $loss_r$ is calculated by:

$$loss_r = \frac{1}{n_r}\sum_{i=1}^{n_r}\left\|\hat{y}_i(U, Y_0^i, t_i + \Delta t_i) - \hat{y}_i(U, Y_{t_i}^i, \Delta t_i)\right\|_2^2 \tag{21}$$

where, n_r is the number of sampled data, $Y_{t_i}^i$ represents the corresponding state at time t_i, including the equivalent initial value, boundary conditions, derivatives of each order, etc. The parameter Δt_i is not a constant value, its also sampled randomly within a certain range.

Measurement data loss $loss_m$ is the MSE between the real measurement data and the neural network fitted data.

$$loss_m = \frac{1}{n_m}\sum_{i=1}^{n_m}\left\|\hat{y}_i(U, Y_0^i, t_i) - y_i(U, Y_0^i, t_i)\right\|_2^2 \tag{22}$$

where, $\hat{y}_i(U, Y_0^i, t_i)$ represents the computation of the neural network, $y_i(U, Y_0^i, t_i)$ represents actual measurement, n_m is the number of sampled data. Considering that the cumulative error will be gradually aggravated with time, we added time-based weights to the data on the basis of the above loss function. It makes the weight of the data close to the initial moment high and the weight of the data far from the initial moment low. In this paper, data weights ω_t are assigned using a simple linear variation, i.e.

$$\omega_t = \omega_{\max} - at + 1 \tag{23}$$

4. Experimental

In the experimental part, we choose a variant of the Vander Pol's equation as a case study to analyze the fitting effect of our method under fixed inputs, time-varying inputs, and segmented function inputs. The effectiveness of the model's solution fluctuates as the difficulty of the ordinary differential equation changes, but it is applicable to arbitrary ordinary differential equations of similar form.

Example 1: *A variant of the Vander Pol's equation*

Consider a second-order system:

$$\ddot{y}(t) - (1 - y^2(t))\dot{y}(t) + y(t) = 0 \tag{24}$$

with

$$y(0) = y_0, \dot{y}(0) = v_0 \tag{25}$$

where $0 \le t \le T$. The differential equation (24) has no external input, we modify the equation to add an external input u. Here the external input can be a constant value U, or a time-varying value $u(t)$. For descriptive convenience, $u(t)$ is used throughout the following. The modified equation is as follows

$$\ddot{y}(t) - (1 - y^2(t))\dot{y}(t) + y(t) = -u(t) \tag{26}$$

where, $u(t)$ is time-varying is an external input to the system. In the control domain, $u(t)$ can be an external force or torque. The differential equation (24) can be regarded as a special case of the Eq. (26), i.e. $u(t) \equiv 0$. This differential equation is a simple example without *x*-dimension, and its inputs are only Y_0 and t.

The activation function chosen in PINN-TI is *tanh*, and the properties of the activation function limit the input *t*.

During training, we sample time t within 0-2 seconds, too large $t(t>5)$ will result in an activation that is infinitely close to 1 and thus fail. The learning rate is 0.001 and the number of iterations is 500000. The exact solution of the differential equation is taken to be computed by the 4th order Runge-Kutta method with the step size set to 0.0001.

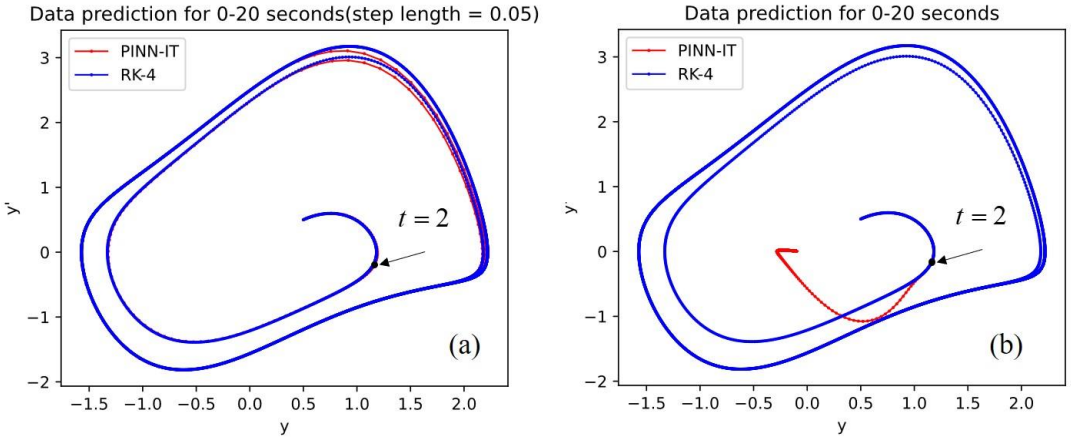

Fig. 4 Comparison of results for solving differential equations with time-invariant input. Fig. 4(a) shows the results solved using the PINN-TI model, and fig. 4(b) shows the results solved using the original PINN. The initial conditions are set to $y(0)=0.5$, $\dot{y}(0)=0.5$, $u(t)\equiv 0.5$.

Table. 1. MSE of prediction results with time-invariant external inputs

Δt	0-1s	0-2s	0-5s	0-10s
0.05	1.9565e-5	3.9754e-5	1.6168e-3	3.3053e-3
	8.2221e-6	1.5541e-5	1.7807e-3	5.4552e-3
0.10	5.9063e-6	9.3016e-6	3.6125e-4	7.4362e-4
	1.0271e-6	2.5844e-6	3.9845e-4	1.1757e-3
0.20	1.8133e-6	1.8485e-6	7.3896e-5	1.8441e-4
	2.8208e-8	1.8912e-7	7.9226e-5	2.3864e-4
0.40	3.7711e-7	3.4812e-7	1.3716e-5	4.0757e-5
	2.5422e-7	2.1081e-7	7.9170e-6	6.0483e-5

In Fig. 4 and Table 1. we show only the solution results of our model under a single initial value condition and a single constant input. Our model can fit the results under any initial value and any constant input (within a predefined range) after one training. Our model can extend the prediction range from 0-2 seconds to 0-20 seconds, while the original PINN, see Fig. 4(b), can fit the solution of differential equations within 0-2 seconds with high accuracy, but the results are poor when extended to 2-20s. The original PINN network needs to be retrained when the initial conditions and external inputs are changed.

PINN-TI model can fit the results of differential equations with arbitrary time-varying external inputs. Assuming that the time-varying external input is

$$u(t)=0.8*\sin(t)+0.57\,. \tag{27}$$

The initial or boundary conditions are set to $y(0)=0.5$, $\dot{y}(0)=0.5$. Based on the previously trained network, the solution of the differential equation under this time-varying input is predicted using the SSF method.

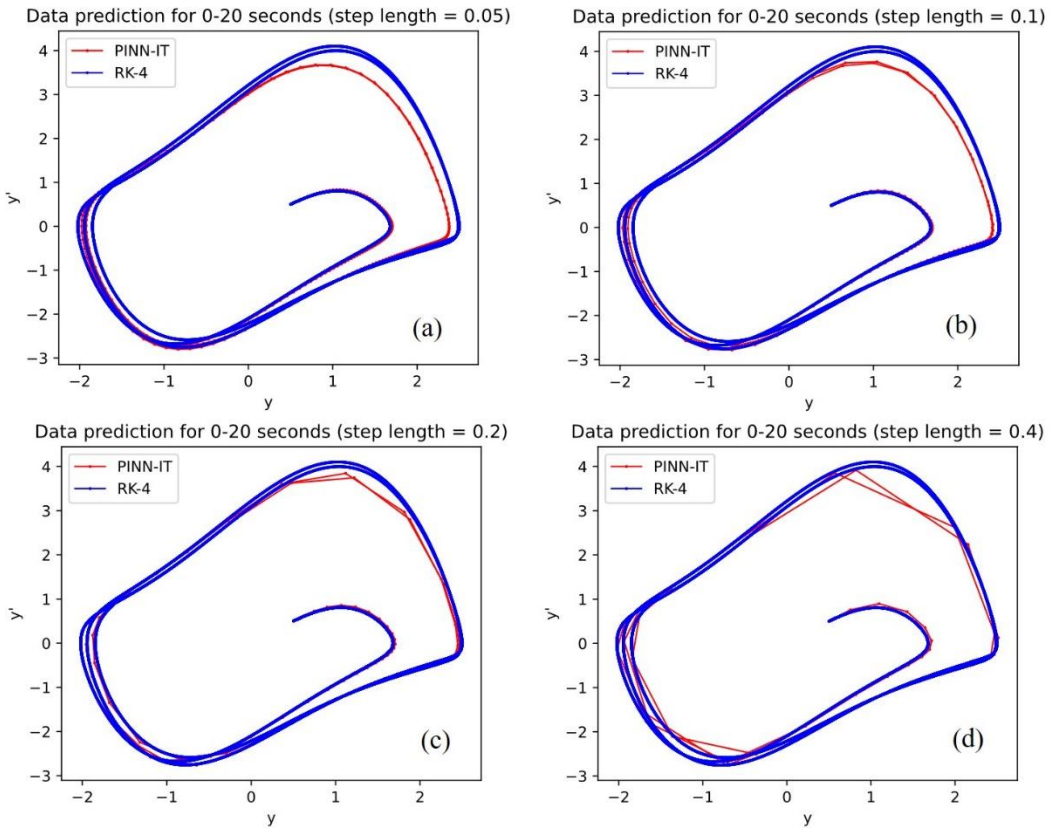

Fig. 5. Comparison of results for solving differential equations with time-varying input. The steps in fig. 5(a), fig. 5(b), fig. 5(c) and fig. 5(d) are 0.05, 0.10, 0.20, and 0.40, respectively. The blue line represents the results of the RK-4 solution and the red line represents the results of the PINN-TI solution.

Table. 2. MSE of prediction results with time-variant external inputs

Δt	0-1s	0-2s	0-5s	0-10s
0.05	3.6989e-6	3.3250e-4	1.5172e-3	5.8392e-3
	3.7238e-4	3.5658e-4	3.8928e-4	7.6342e-3
0.10	3.7689e-5	3.5190e-4	3.2095e-4	2.0487e-3
	4.9394e-4	4.1039e-4	7.2399e-4	4.4076e-3
0.20	1.6872e-4	8.9548e-4	3.1608e-3	9.0212e-3
	2.1361e-4	9.7549e-4	5.0838e-3	2.4878e-2
0.40	1.4598e-4	3.2161e-3	9.2460e-3	4.6327e-2
	2.2246e-3	3.0276e-3	1.6812e-2	1.2358e-1

Fig. 5 and Table 2 illustrates the results of the PINN-TI network fitting under time-varying inputs at different step sizes. As the time horizon of the prediction expands, the cumulative error accumulates and the accuracy of the prediction decreases. The MSE of the fitted results and exact solutions of the PINN-TI network under time-varying inputs with different step sizes and different time scales are shown in Table 2. The increase of the time step leads to the loss of more information about the time-varying inputs, and the error shows an increasing trend as the time step increases. In order to verify the results of our method under different time-varying inputs, we sampled 100 different time-varying inputs $u(t)$, i.e.

$$u(t) = a * \sin(t) + b * \cos(t) + c \tag{28}$$

where, the coefficients a, b and c are all constants, $a, b, c \in [0,1]$. Different combinations of coefficients represent different time-varying inputs. The results are shown in Fig. 6. The MSE of the prediction results for all four Δt in

the range of 0-2 seconds is less than 0.001. The MSE tends to increase as the prediction time range and the prediction step size increase.

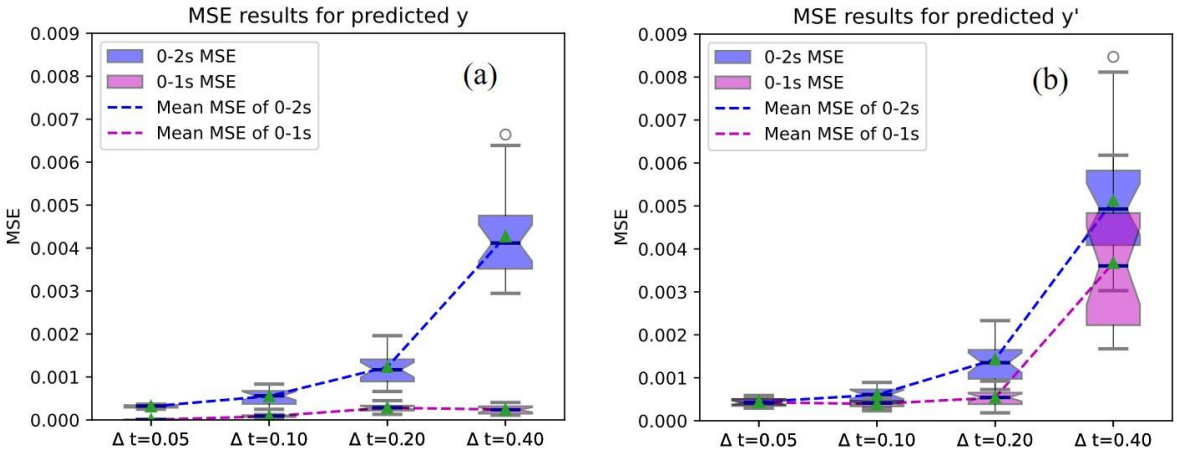

Fig. 6 Box plots of MSE of prediction results for multiple time-varying input. (a) is the box plot of the MSE between the predicted y and the true value. (b) is the box plot of the MSE between the predicted y' and the true value.

PINN-TI model is also applicable to the process of solving differential equations with arbitrary segmented function inputs. Assume that the external input for the required solution is

$$u(t)=\begin{cases}0.5*\sin(t)+0.57, \sin(t)>0\\ 0.5*\cos(t)-0.57, \sin(t)<0\end{cases}. \tag{29}$$

The initial or boundary conditions in Eq. (26) are set to $y(0)=0.45$, $y'(0)=0.45$.

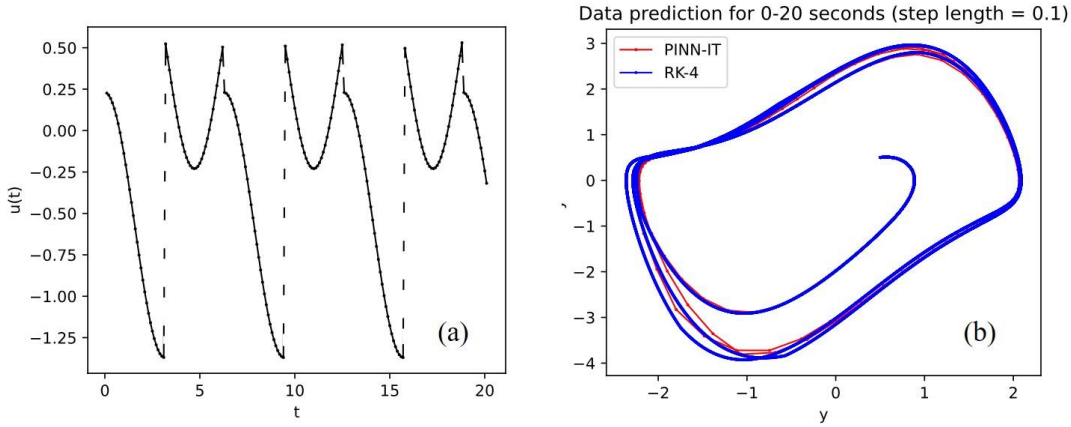

Fig. 7. Solution results under external inputs in the form of segmented functions.

Table. 3. MSE of prediction results with segmented functions inputs

Δt	0-1s	0-2s	0-5s	0-10s
0.05	3.6509e-5	2.1417e-5	1.5029e-3	8.2687e-3
	4.5375e-6	1.0376e-4	1.6717e-3	2.4337e-2
0.10	6.0028e-6	1.7867e-4	1.6999e-3	8.2221e-3
	5.2618e-5	8.0780e-4	4.2223e-3	2.5952e-2
0.20	9.2288e-6	9.8015e-4	7.4652e-3	3.2853e-2
	2.1068e-4	3.2419e-3	1.7701e-2	8.4991e-2

Table. 4. MSE of predicted results using DeepONet

	0-1s	0-2s	0-5s	0-10s
MSE	5.7133	6.5323	6.2422	9.3231
	4.2936	6.4292	9.4711	8.4421

Fig. 7 and Table 3 show the prediction results of our model under the segmented function input. We also used the DeepONet model and trained the DeepONet model with 100 sensors selected uniformly over a period of 0-10 seconds. The trained network predicts the results under this segmented function input and the MSE between the predicted results and the exact solution is shown in Table 4. The DeepONet model is unable to handle similar model prediction problems under segmented function inputs, and input fluctuations at non-sensor locations do not affect the model's prediction results.

Example 2: *Multiple external inputs*

This example is used to simulate the applicability of our method when there are multiple external inputs. We modify Eq. (26) to the following form

$$\ddot{y}(t) - (1 - y^2(t) - u_2(t))\dot{y}(t) + y(t) = -u_1(t) \tag{30}$$

where, $u_1(t)$ and $u_2(t)$ are both external inputs. Corresponding to our PINN-TI model, the model input U needs to be sampled in two dimensions, u_1 and u_2. Such systems with multiple external inputs are more common in practical applications such as rope-driven parallel robots [31], multi-stage vibration suppression platforms, etc.

Assume that the time-varying inputs u_1 and u_2 are respectively:

$$\begin{aligned} u_1(t) &= 0.8 * \sin(2t) + 0.57 \\ u_2(t) &= 0.9 * \cos(t/2) + 0.15 \end{aligned} \tag{31}$$

The hyperparameters for model training are set as in example 1. The initial or boundary conditions are set to $y(0) = 0.37$, $\dot{y}(0) = 0.48$. The range of sampling time set during network training is 0-1 second.

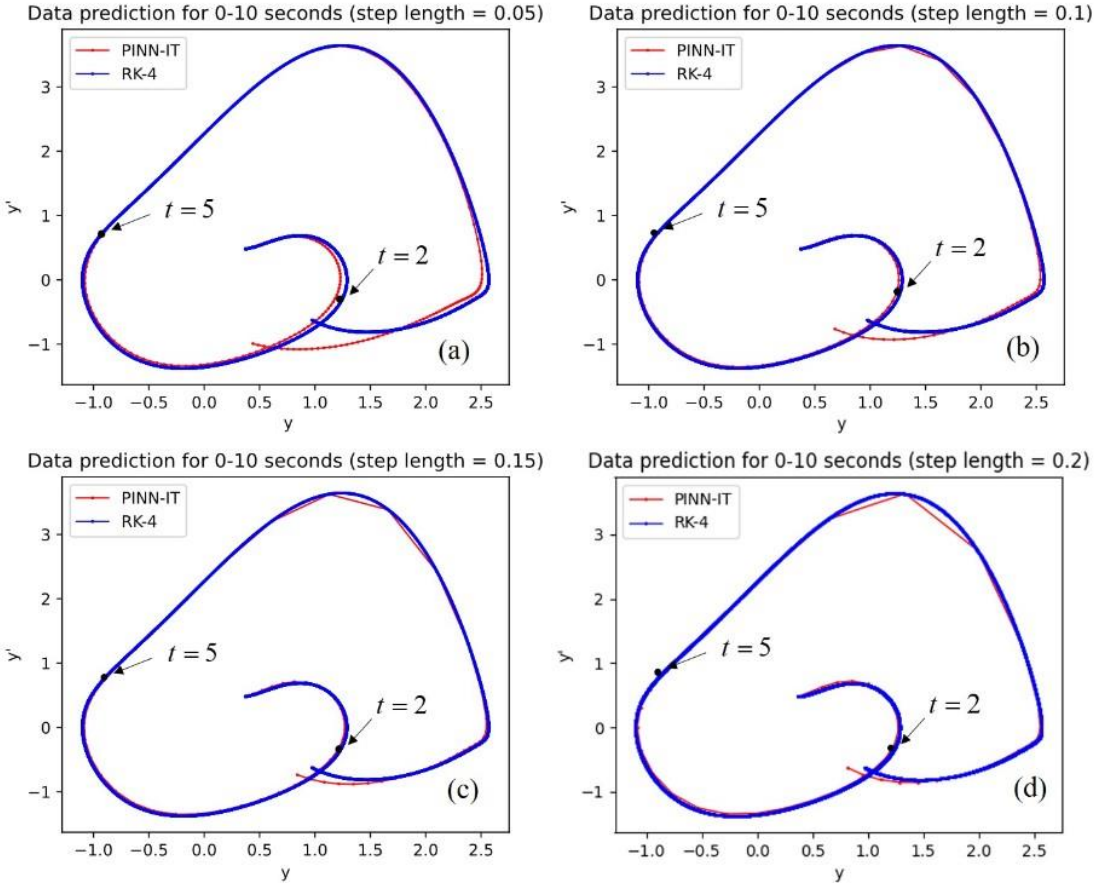

Fig. 8 Comparison of results for solving differential equations with two time-varying inputs.

Table. 5. MSE of prediction results with two external inputs

Δt	0-1s	0-2s	0-5s
0.05	5.6228e-4	1.5000e-3	2.6575e-3
	5.2391e-5	2.6295e-4	1.7052e-3
0.10	1.5432e-5	2.5624e-4	1.9450e-3
	1.6366e-4	1.5274e-4	2.0812e-3
0.15	2.7886e-5	7.3140e-5	2.5833e-3
	6.6643e-4	7.4457e-4	3.5791e-3
0.20	1.6506e-4	2.1720e-4	3.6601e-3
	1.2766e-3	1.6776e-3	5.7699e-3

Fig. 8 and Table 5 show the results of solving the ordinary differential equation (30) with an external input of (31). Our prediction time is set to 0-10 seconds, which is 10 times the range of the training time. The results of the MSE of the prediction results at different sampling steps within 0-5 seconds are shown in Table 5, in which the MSE is minimized at a sampling step of 0.1 within 0-1 seconds, and in the range of 0-2 seconds, the MSE is minimized at a sampling step of 0.15. Sampling $u_1(t)$ and $u_2(t)$ 100 times using Eq. (28) verifies the effectiveness of our trained model under different time-varying inputs. The results of the MSE are shown in Fig. 9.

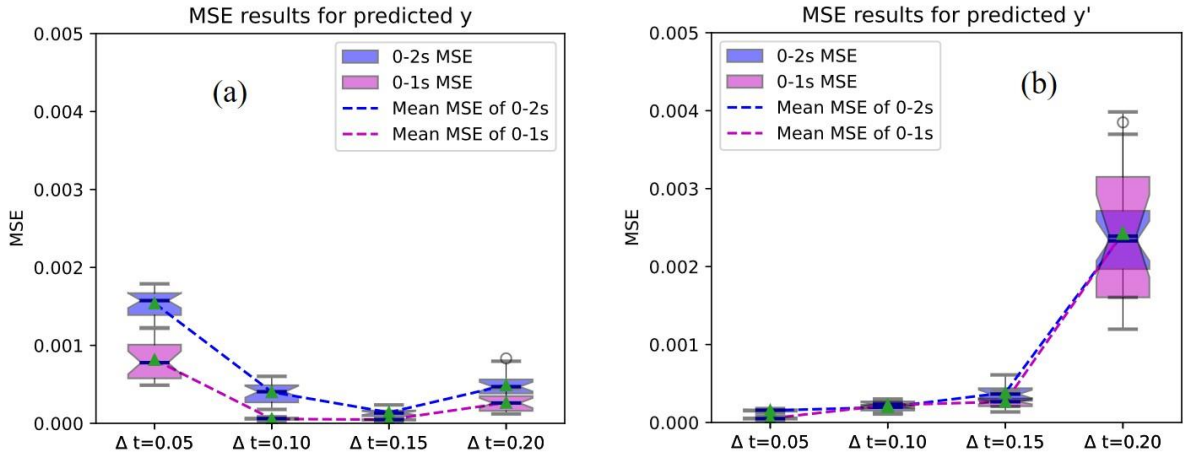

Fig. 9. Box plots of MSE of prediction results. (a) is the box plot of the MSE between the predicted y and the true value. (b) is the box plot of the MSE between the predicted y' and the true value.

As shown in Fig. 9, our model predicts y-value in 0-1 seconds with MSEs of the order of 1e-4 at a sampling step of 0.1. When the sampling step size is 0.2, its prediction result for y' has a large error, due to the loss of time-varying input information for a large sampling step size.

5. Conclusion

We propose a PINN-TI model and SSF algorithm which can be trained once to realize the problem of solving ordinary differential equations under different initial values or boundary conditions and different time-varying inputs. Our method can handle the solution of differential equations under segmented function inputs, which is not possible with PINN and DeepONet networks. Also our network can make predictions for time-varying inputs of arbitrary duration, whereas DeepONet is affected by the location of the sensors and its required external inputs must be of fixed duration. Our approach provides a new solution and idea for the problem of solving and generalizing differential equations containing time-varying inputs.

6. Discussion

Our solution scheme is based on the same idea of discretizing the time-varying inputs as in traditional numerical solution methods, while embedding the physical information into the neural network during the training process. We have only considered the solution process for arbitrary variable inputs in a continuous time model and have not extended it to discrete environments. In the future, we will extend the network to discrete environments. This discretization idea to solve time-varying input differential equations can also be combined with DeepONet and FNO, and is one of the main research directions in the future. In terms of network structure, MLP can be extended to CNN or RNN in order to be applicable to different problems and is a major future research direction. Our model has some challenges in applying it to partial differential equations and is a viable research direction.

Acknowledgement

The authors would like to thank the project "Basic Theory and Key Technology of High-Performance Rigid-Flexible Coupling Driven Spraying Robot for Large and Complex Components", Project No. 52335002, for the support of this work.

Data availability

Data, models and/or codes that support the findings of this study are available from the corresponding author upon reasonable request.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Reference：

[1] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Inferring solutions of differential equations using noisy multi-fidelity data," *Journal of Computational Physics,* vol. 335, pp. 736-746, 2017/04/15/ 2017.

[2] J. Cheng, M. R. Sayeh, M. R. Zargham, and Q. Cheng, "Real-Time Vector Quantization and Clustering Based on Ordinary Differential Equations," *IEEE Transactions on Neural Networks,* vol. 22, no. 12, pp. 2143-2148, 2011.

[3] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics,* vol. 378, pp. 686-707, 2019/02/01/ 2019.

[4] X. Zhao, J. Du, and Z. Wang, "HCS-R-HER: Hierarchical reinforcement learning based on cross subtasks rainbow hindsight experience replay," *Journal of Computational Science,* vol. 72, p. 102113, 2023/09/01/ 2023.

[5] S. Mohammad, P. Nick, and X. G. Chen, "Modeling the Effects of Cu Content and Deformation Variables on the High-Temperature Flow Behavior of Dilute Al-Fe-Si Alloys Using an Artificial Neural Network," *Materials,* vol. 9, no. 7, 2016.

[6] A. Usman, M. Rafiq, M. Saeed, A. Nauman, A. Almqvist, and M. Liwicki, "Machine Learning Computational Fluid Dynamics," in *2021 Swedish Artificial Intelligence Society Workshop (SAIS)*, 2021, pp. 1-4.

[7] S. Chickerur and P. Ashish, "A Convolutional Neural Network Based Approach for Computational Fluid Dynamics," in *2021 Second International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, 2021, pp. 1-5.

[8] E. N. Irawan, S. Sitompul, K. I. Yamashita, and G. Fujita, "Computational Fluid Dynamics Analysis on the Improvement of Hybrid Savonius-Darrieus NREL S809 at Various Fluid Flows," in *2023 4th International Conference on High Voltage Engineering and Power Systems (ICHVEPS)*, 2023, pp. 389-394.

[9] J. Baiges, R. Codina, I. Castañar, and E. Castillo, "A finite element reduced-order model based on adaptive mesh refinement and artificial neural networks," *International Journal for Numerical Methods in Engineering,* vol. 121, no. 4, pp. 588-601, 2019.

[10] H. F. Parapari and M. B. Menhaj, "Solving nonlinear ordinary differential equations using neural networks," in *2016 4th International Conference on Control, Instrumentation, and Automation (ICCIA)*, 2016, pp. 351-355.

[11] R. Kaplan, J. Klobušický, S. Pandey, D. H. Gracias, and G. Menon, "Building polyhedra by self-assembly: Theory and experiment," *Artificial Life,* vol. 20, no. 4, pp. 409-439, 2014.

[12] Z. Luo, S. i. Kamata, Z. Sun, and W. Zhou, "Deep Neural Networks with Flexible Complexity While Training Based on Neural Ordinary Differential Equations," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 1690-1694.

[13] S. Cai, Z. Wang, L. Lu, T. A. Zaki, and G. E. Karniadakis, "DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks," *Journal of Computational Physics,* vol. 436, p. 110296, 2021/07/01/ 2021.

[14] Z. Mao, L. Lu, O. Marxen, T. A. Zaki, and G. E. Karniadakis, "DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators," *Journal of Computational Physics,* vol. 447, p. 110698, 2021/12/15/ 2021.

[15] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," *Nature Machine Intelligence,* vol. 3, no. 3, pp. 218-229, 2021/03/01 2021.

[16] Z.-Y. Li *et al.*, "Fourier Neural Operator for Parametric Partial Differential Equations," *ArXiv,* vol. abs/2010.08895, 2020.

[17] Z.-Y. Li *et al.*, "Neural Operator: Graph Kernel Network for Partial Differential Equations," *ArXiv,* vol. abs/2003.03485, 2020.

[18] Z. Ma, J. Hou, W. Zhu, Y. Peng, and Y. Li, "PMNN: Physical model-driven neural network for solving time-fractional differential equations," *Chaos, Solitons & Fractals,* vol. 177, p. 114238, 2023/12/01/ 2023.

[19] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics,* vol. 3, no. 6, pp. 422-440, 2021/06/01 2021.

[20] Y.-J. Wang and C.-T. Lin, "Runge-Kutta neural network for identification of dynamical systems in high accuracy," *IEEE Transactions on Neural Networks,* vol. 9, no. 2, pp. 294-307, 1998.

[21] L. Lingsch, M. Y. Michelis, S. M. Perera, R. K. Katzschmann, and S. Mishra, "Vandermonde Neural Operators," *ArXiv,* vol. abs/2305.19663, 2023.

[22] L. Serrano *et al.*, "Operator Learning with Neural Fields: Tackling PDEs on General Geometries," *ArXiv,* vol. abs/2306.07266, 2023.

[23] U. B. Waheed, "Kronecker Neural Networks Overcome Spectral Bias for PINN-Based Wavefield Computation," *IEEE Geoscience and Remote Sensing Letters,* vol. 19, pp. 1-5, 2022.

[24] Y. Wang, S. Alkhadhr, and M. Almekkawy, "PINN Simulation of the Temperature Rise Due to Ultrasound Wave Propagation," in *2021 IEEE International Ultrasonics Symposium (IUS)*, 2021, pp. 1-4.
[25] R. Liu and P. Gerstoft, "SD-PINN: Physics Informed Neural Networks for Spatially Dependent PDES," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1-5.
[26] S. L. N. Dhulipala and R. C. Hruska, "Efficient Interdependent Systems Recovery Modeling with DeepONets," in *2022 Resilience Week (RWS)*, 2022, pp. 1-6.
[27] K. Ye, J. Zhao, X. Liu, C. Moya, and G. Lin, "DeepONet Based Uncertainty Quantification for Power System Dynamics with Stochastic Loads," in *2023 IEEE Power & Energy Society General Meeting (PESGM)*, 2023, pp. 1-6.
[28] X. Zhang, M. Zhang, Y. Song, X. Jiang, F. Zhang, and D. Wang, "DeepONet-Based Waveform-Level Simulation for a Wideband Nonlinear WDM System," *Journal of Lightwave Technology,* pp. 1-15, 2023.
[29] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *Computer Science,* 2014.
[30] D. R. S. Saputro and P. Widyaningsih, "Limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method for the parameter estimation on geographically weighted ordinal logistic regression model (GWOLR)," *AIP Conference Proceedings,* vol. 1868, no. 1, 2017.
[31] M. Seo, S. Yoo, M. Choi, J. Oh, H. S. Kim, and T. Seo, "Vibration Reduction of Flexible Rope-Driven Mobile Robot for Safe Façade Operation," *IEEE/ASME Transactions on Mechatronics,* vol. 26, no. 4, pp. 1812-1819, 2021.