*Electronic Research Archive*

https://www.aimspress.com/journal/era

ERA, 34(6): 4080–4106.
DOI: 10.3934/era.2026183
Received: 25 December 2025
Revised: 16 April 2026
Accepted: 07 May 2026
Published: 15 May 2026

*Research article*

# Physics-informed neural networks utilizing the Legendre-Gauss-Lobatto collocation method for solving differential-algebraic equation with discrete event

**Canyi Che, Qingli Zhao*, Funing Yang and Xintong Zhang**

School of Science, Shandong Jianzhu University, Jinan 250101, China

* **Correspondence:** Email: shizilu@126.com.

**Abstract:** Differential-algebraic equation (DAE) is widely used in engineering domains, such as fluid dynamics, multi-body dynamics, mechanical systems, and control theory, owing to their ability to effectively characterize dynamic variations and inherent constraints. In recent years, physics-informed neural networks (PINNs) have manifested remarkable advantages in solving both the forward and inverse problems of DAE by integrating physical prior knowledge into neural network models. Presently, PINNs-based approaches still encounter challenges of inadequate solution accuracy and limited generalization performance when dealing with DAE involving discrete events. This paper presented a physics-informed neural network that integrates the Legendre-Gauss-Lobatto (LGL) collocation method from spectral methods to solve the aforementioned DAE with discrete event. To further augment the accuracy and continuity of the solution, the model employed a time-domain decomposition strategy to construct the network architecture, thereby enabling high-precision continuous-time prediction of DAE. Numerical examples illustrated that the LGL-PINN can attain high-precision solutions of DAE. In comparison with the PINNs, the error between the predicted solution and the exact solution of the LGL-PINN was substantially reduced, with the accuracy improved by one to two orders of magnitude. Therefore, the proposed solution model demonstrated excellent computational accuracy for solving DAE problems involving discrete event.

**Keywords:** differential-algebraic equation; discrete event; physics-informed neural network; LGL collocation method; time-domain decomposition

## 1. Introduction

Differential-algebraic equation (DAE) constitute a prevalent and crucial category of equations in the realms of science and engineering, integrating both differential equations and algebraic equations. This class of equations finds extensive applications in various fields, such as fluid dynamics, multi-body

dynamics [1], power system simulation [2], mechanical system modeling, chemical reaction dynamics, circuit simulation, control system design, and biological system modeling [3, 4]. Consequently, the precise resolution of DAE carries substantial theoretical and practical significance.

In distinct developmental stages and across diverse research realms, DAE frequently manifest different structural configurations, encompassing linear DAE, nonlinear DAE, semi-explicit DAE, implicit DAE, and Hessenberg-type DAE. Fortuitously, in actual physical modeling, the majority of system models are either low-index DAE or high-index Hessenberg DAE [5]. The conventional numerical approaches commonly employed to solve DAE systems involve the implicit Runge-Kutta method [6], the backward differentiation formula (BDF) method [7], the pseudo-spectral method [8], the domain decomposition method [9], the exponential integrator method [10], the Generalized-$\alpha$ method [11], and the Lie group method [12–14]. Nevertheless, for high-index and discontinuous event DAE systems, these methods are only applicable to specific DAE types and may lead to varying extents of numerical accuracy deterioration. Moreover, when addressing high-dimensional, discontinuous event, and nonlinear DAE, they might encounter problems such as substantial computational burden or poor stability.

In recent years, owing to the significant progress in computing power, neural networks have witnessed a qualitative improvement in computational capacity and have been extensively utilized in numerous domains. In 2019, Raissi, Perdikaris, and Karniadakis put forward the physics-informed neural network (PINN), a crucial numerical approximation approach for addressing partial differential equations problems [15]. Since its introduction, PINNs have experienced rapid development and sparked extensive discussions and research in the realm of scientific computing. In addition to demonstrating significant advantages in the field of numerical solution for ordinary differential equations (ODEs) and partial differential equations (PDEs) [16–18], the application boundary of PINNs has been further extended to the solution of DAE.

Leake and Mortari [19] put forward the theory of functional connections (TFC), which integrates the constraints of differential equations into the TFC trial solution expressions that contain free functions. By integrating neural networks with TFC, the method transforms differential equations into unconstrained optimization problems for resolution. Liu et al. [20] proposed a Jacobi neural network approach for solving linear DAE with variable coefficients. This approach attains satisfactory precision in solving DAE with an index of 1. Moya and Lin introduced the DAE-PINN framework based on the implicit Runge-Kutta method and PINN [21]. This framework resolves the index-1 stiff DAE problems that describe power network simulations and facilitates the long-term simulation of DAE solutions. Liang [22] proposed an improved physics-informed neural network solution framework named PDAE-PINN, which can directly solve several types of typical linear (algebraic index-1, algebraic index-2, and variable-coefficient) and nonlinear partial differential algebraic equation systems. Yin and Hu [23] put forward the PINN with temporal attention (PINN-TA), a parameter identification model for time-delay chaotic systems based on the temporal attention mechanism. This model offers a new solution strategy for solving time-delay DAE. Yang et al. [24] solved several types of Heisenberg-type DAE based on artificial neural networks and derived approximate analytical solutions to the equations. Chen [25] proposed a PINN computational framework integrated with the high-precision Radau-PINN numerical method and an improved fully connected neural network architecture, which enables the direct forward solution of high-index DAE. In addition, the numerical solutions for extended forms of DAE, such as fractional-order and

stochastic types, have also been studied in depth [26–32]. Although these works have promoted the application of PINNs in continuous DAEs, almost all are designed for smooth, continuous, and time-domain-free systems. When dealing with non-smooth DAEs with discrete events such as switching, collisions, and pulses, problems such as inaccurate capture of solution jumps, local precision collapse, and accumulated errors over time often occur, making it difficult to balance global stability and local high accuracy.

To overcome the limitations of traditional PINNs, the academic community has proposed numerous improved PINNs frameworks in directions such as time-domain decomposition, adaptive strategies, kernel function embedding, and numerical method integration, significantly enhancing the efficiency and accuracy of solving time-varying problems and dynamic systems. Guo et al. [33] proposed a curriculum-transfer-learning based physics-informed neural network (CTL-PINN), which decomposes long-term dynamic problems into a series of short-term sub-problems, combining curriculum learning and transfer learning to achieve sequential time-domain information transmission and efficient reuse, effectively alleviating the defects of standard PINN such as being prone to getting stuck in local optima, long-term error accumulation, and low computational efficiency. It has demonstrated excellent long-term simulation capabilities in nonlinear wave propagation, thin plate dynamic response, and large-scale hydrodynamic models. Fu et al. [34] proposed the physical information kernel function neural network (PIKFNN), embedding PDE prior information into the activation function and using a single hidden layer shallow network to reduce computational complexity, achieving rapid convergence in time-varying problems such as heat conduction and wave propagation; Huang et al. [35] proposed enhanced PINNs (EPINNs), dynamically balancing multiple physical constraints through adaptive loss weights and resampling strategies, significantly improving stability and convergence speed in strongly coupled nonlinear dynamic systems; Li et al. respectively proposed the basic solution neural network (FSNNs) [36] and adaptive basic solution method (AMFS) [37], integrating PINNs with the basic solution method (MFS), intelligently optimizing the distribution of points and source points, achieving high-precision solutions in gridless, inverse problems, and non-smooth boundary scenarios. These advancements collectively indicate that combining PINNs with high-precision numerical methods, introducing adaptive configuration and domain decomposition strategies, is the key path to enhancing the solution capabilities of dynamic systems and non-smooth problems, and also provides important insights for solving DAE systems with discrete event.

However, the existing improved PINNs framework still has clear shortcomings: CTL-PINN, PIKFNN, EPINNs, etc. are designed based on pure differential equations and do not consider the unique differential-algebraic coupling constraints of DAEs; FSNNs and AMFS focus on boundary-type problems and source point optimization, but do not adapt the architecture to time-domain discontinuous events and rigid algebraic constraints; the existing DAE-PINNs methods lack high-precision point distribution support at the spectral method level, making it difficult to achieve exponential convergence in non-smooth regions. In summary, there is still a lack of a unified PINN solution framework that can strictly satisfy the differential-algebraic coupling constraints of DAEs, accurately capture discrete events and non-smooth solutions, and simultaneously suppress the accumulation of time-domain errors. Based on this, this paper integrates the Legendre-Geuss-Lobatto (LGL) high-precision point distribution method with the PINN, introduces time-domain decomposition and adaptive point distribution encryption strategies, and constructs the LGL-PINN

method, aiming to make up for the accuracy and stability deficiencies of existing methods on discrete event DAEs, and provide a new high-precision and high-robustness solution approach for discrete DAE problems.

## 2. DAE and PINN

### *2.1. Differential-algebraic equation*

DAE constitutes a category of mixed equations that incorporate both differential equations and algebraic equations. Variables within an equation that do not entail derivatives are denoted as algebraic variables, and the presence of algebraic variables indicates that the equation cannot be presented in the explicit form. Generally, a differential-algebraic equation can be formulated as follows

$$\begin{cases} \mathbf{F}(t, y(t), y^{(1)}(t), \ldots, y^{(k)}(t), z_1(t), z_2(t), \ldots, z_m(t)) = \mathbf{0}, \\ \mathbf{G}(t, y(t), z_1(t), z_2(t), \ldots, z_m(t)) = \mathbf{0}. \end{cases} \tag{2.1}$$

Here, $\mathbf{F}$ represents the differential equations, $\mathbf{G}$ represents the algebraic function equations, $y(t) \in \mathbb{R}$ serves as the differential function variable, $\mathbf{z}(t) = (z_1(t), z_2(t), \ldots, z_m(t)) \in \mathbb{R}^m$ denotes the algebraic function variable, $t_0$ denotes the initial time point, and $y(t_0) = y_0$ and $\mathbf{z}(t_0) = \mathbf{z_0}$ are the initial values. The DAE with discrete event studied in this article includes two cases: discontinuous differential equations and discontinuous analytical solutions.

### *2.2. PINN*

#### 2.2.1. PINN method

PINN incorporates physical laws (such as differential equations, algebraic equations, initial conditions, and boundary conditions) into the loss function of neural networks. This incorporation ensures that the network adheres to both data distribution and physical laws throughout the training process. PINN typically employs fully connected neural networks (MLPs), in which the inputs consist of spatial coordinates and time, and the outputs are the physical quantities to be determined.

The loss function of the PINN consists of four components and realizes the dual constraints of data and physics through the minimization of the total loss. For instance, the heat conduction equation is given as

$$\frac{\partial u(x,t)}{\partial t} = \alpha \frac{\partial^2 u(x,t)}{\partial x^2}, \tag{2.2}$$

where $u(x,t)$ represents the physical quantity of the temperature field at the spatial position $x$ and time $t$, $\alpha$ represents the thermal diffusivity of the medium, which is a constant scalar. This equation is the core physical constraint that needs to be satisfied during PINN training, and it is used to construct the physical residual loss term.

Data fitting term: compute the error between the network output data and the true value

$$L_{\text{data}} = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left(u_N(x_i, t_i) - u_i^{\text{obs}}\right)^2. \tag{2.3}$$

Here, $N_{\text{data}}$ represents the aggregate quantity of observed data, $u_N(x_i, t_i)$ denotes the predicted value of the PINN, and $u_i^{\text{obs}}$ signifies the actual observed value.

Regarding the physical equation term: the output of the network is substituted into the physical equation to compute the residual error. The residual is presented as

$$R(x_i, t_i) = \frac{\partial u_N(x_i, t_i)}{\partial t} - \alpha \frac{\partial^2 u_N(x_i, t_i)}{\partial x^2}. \tag{2.4}$$

Loss is defined as

$$L_{physics} = \frac{1}{N_{physics}} \sum_{i=1}^{N_{physics}} R(x_i, t_i)^2, \tag{2.5}$$

where $N_{physics}$ represents the number of sampling points for the physical residual constraints, which is used to control the strength and computational load of the physical constraints.

Boundary condition term ensures that the network output at the boundary satisfies the boundary condition.

$$L_{\text{BC}} = \frac{1}{N_{\text{BC}}} \sum_{i=1}^{N_{\text{BC}}} \left( u_N(x_i^{\text{BC}}, t_i) - u_i^{\text{BC}} \right)^2, \tag{2.6}$$

where $N_{\text{BC}}$ represents the total number of boundary condition sampling points, $x_i^{\text{BC}}$ represents the spatial coordinates of the $i$-th boundary point, and $x_i^{\text{BC}}$ represents the preset true value at the $i$-th boundary point.

Initial condition term ensures that the network output at the initial moment satisfies the initial condition

$$L_{\text{IC}} = \frac{1}{N_{\text{IC}}} \sum_{i=1}^{N_{\text{IC}}} (u_N(x_i^{\text{IC}}, t_i) - u_i^{\text{IC}})^2, \tag{2.7}$$

$N_{\text{IC}}$ represents the total number of initial condition sampling points, $x_i^{\text{IC}}$ represents the spatial coordinate point corresponding to the initial time, and $u_i^{\text{IC}}$ represents the preset true value at the coordinate point at the initial time.

The total loss of PINN is the weighted sum of the above four items, and it achieves the dual constraints of data and physics by minimizing the total loss. The total loss is calculated as

$$L_{\text{PINN}} = \lambda_{\text{data}} L_{\text{data}} + \lambda_{\text{physics}} L_{\text{physics}} + \lambda_{\text{BC}} L_{\text{BC}} + \lambda_{\text{IC}} L_{\text{IC}}, \tag{2.8}$$

where $\lambda_{\text{data}}$, $\lambda_{\text{physics}}$, $\lambda_{\text{BC}}$, and $\lambda_{\text{IC}}$ are weighting coefficients, which are used to balance the contributions of the respective terms.

In the training process, the automatic differentiation technique of the neural network can be employed to effectively compute the arbitrary-order derivatives of the network output. Consequently, the residuals of the physical equation can be integrated into the loss function. During the training phase, the network parameters are optimized through back propagation to minimize the total loss. The most frequently utilized optimization algorithms for training neural networks encompass Adam and stochastic gradient descent (SGD), as well as a multitude of other variants. Although there is a lack of theoretical proof that this training process converges to the global optimal solution, empirical evidence suggests that when the differential equation is well-posed, the solution is unique, the neural network structure possesses sufficient expressive capability, and the quantity of training data points is

adequate, PINN can generally produce prediction results with a certain degree of accuracy. It is worth noting that for finite difference or finite element methods, numerical solutions are only defined at selected grid points, and numerical solutions at other points need to be acquired through interpolation. In contrast, neural network functions can conveniently offer values at any point within the domain.

#### 2.2.2. Activation function

An activation function is generally defined as a functional association between the outputs of upper-layer nodes and the inputs of lower-layer nodes within a neural network. In contemporary neural networks, the majority of activation functions employed are nonlinear. This is because they introduce nonlinear relationships into the network, thereby endowing the network with an improved expressive capability and the theoretical potential to approximate any functional relationship. The subsequent section presents four frequently utilized activation functions: the Sigmoid function, the rectified linear unit (ReLU) function, the Leaky-ReLU function, and the tanh function, which are elaborated as follows.

(1) Sigmoid function

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{2.9}$$

The Sigmoid function is capable of mapping a continuous input value to an output within the interval $[0, 1]$. Nevertheless, it is susceptible to the problem of gradient vanishing during the process of gradient back propagation, which may cause the training to cease. Moreover, exponential operations augment the computational load in deep neural networks, resulting in its decreasing utilization in recent years.

(2) ReLU function

$$\text{ReLU}(x) = \max(0, x). \tag{2.10}$$

Here, symbol $x$ represents the weighted input of the neuron, which is the result obtained by multiplying the upper-level output by weights and adding biases. When the input z is greater than or equal to 0, the output is equal to the input and the gradient is always 1; when the input $x$ is less than 0, the output is 0 and the gradient is 0. ReLU function is extensively employed as an activation function within neural networks. This is attributed to its rapid computational speed, straightforward derivative, and its capacity to facilitate faster training of neural networks compared to the sigmoid and tanh functions. Nevertheless, the ReLU function is susceptible to the dying ReLU issue. Specifically, when the input assumes a negative value, the derivative of the ReLU function becomes zero at that particular point, thereby impeding the update of the neuron.

(3) Leaky-ReLU function

$$\text{Leaky-ReLU}(x) = \begin{cases} x, & x \geq 0, \\ \alpha x, & x < 0. \end{cases} \tag{2.11}$$

Leaky-ReLU represents an enhanced variant of the ReLU activation function. Its primary aim is to address the issue of neuron death, which is induced by a zero gradient within the negative input range of the ReLU. This is achieved by maintaining a small nonzero gradient for negative inputs, thereby guaranteeing the gradient's continuity during the back propagation process.

(4) Tanh function

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.12}$$

Although the hyperbolic tangent function also encounters issues related to gradient vanishing and exponential operations, it exhibits superior performance compared to other activation functions in certain practical scenarios.

Numerical experiments show that Leaky-ReLU can effectively avoid the phenomenon of gradient disappearance in negative intervals, and can still maintain relatively stable differential characteristics near the mutation points of the solution, making it most suitable for high-precision solution of discontinuous DAE problems; Tanh is more suitable for scenarios with relatively smooth solution structures, but it converges slowly in long time domains and high gradient regions; ReLU is prone to neuron death in discontinuous regions, resulting in a significant decrease in solution accuracy; Sigmoid is prone to the problem of gradient disappearance and is not suitable for such neural network solutions with strong physical constraints. Therefore, in both numerical examples of this article, Leaky-ReLU was used as the activation function.

## 3. Method introduced in this article

### *3.1. LGL collocation method*

The LGL collocation method is a highly representative high-precision numerical solution approach within the framework of spectral methods. It is specifically devised for differential equations, integral equations, and dynamical systems featuring discontinuities or high-gradient characteristics. Its primary advantage lies in the fact that it can attain extremely high precision using significantly fewer points compared to traditional numerical methods, such as the finite difference method and the finite element method. The fundamental concept of the collocation method is to convert the continuous problem of solving differential equations into ensuring that the equation residuals are zero at discrete collocation points and subsequently obtain numerical solutions by solving algebraic equations. The selection of collocation points is founded on the LGL points of Legendre polynomials, and the approximation method employed is global spectral approximation with exponential convergence.

LGL points serve as the core essence of the LGL collocation method, and their distribution characteristics directly influence the accuracy and efficacy of the method. The definition of LGL points is founded upon the distinctive properties of Legendre polynomials. LGL points within the standard interval [-1, 1] for an $n$-order approximation $x_0, x_1, \cdots, x_N$. The collocation points are required to encompass the endpoints of the interval, specifically, $x_0 = -1$ and $x_N = 1$. This characteristic renders it inherently appropriate for both initial value problems and boundary value problems, obviating the necessity for additional treatment of boundary conditions. The internal collocation points $x_1, x_2, \ldots, x_{N-1}$ adhere to the internal point constraint: each internal collocation point is a root of the first derivative of the $n$th-order Legendre polynomial, namely, $P'_N(x) = 1$. Where the nth-order Legendre polynomial $P_N(x)$ is an orthogonal polynomial satisfying the following orthogonality condition

$$\int_{-1}^{1} P_m(x)P_n(x)\,dx = \frac{2}{2n+1}\delta_{mn}, \tag{3.1}$$

here $P_m(x)$ and $P_n(x)$ represent the $m$-th and $n$-th Legendre polynomials; $\delta_{mn}$ represents the Kronecker function, with $\delta_{mn} = 1$ when $m = n$, and $\delta_{mn} = 0$ when $m \neq$n. Orthogonality serves as the mathematical foundation for the high precision of the LGL collocation method. By leveraging the arbitrary series expansion property of orthogonal polynomials, any function can be expanded into a polynomial series,

which can conveniently approximate the derivatives, integrals, and inner products of functions. It particularly excels at error minimizing and is applicable to various function approximation scenarios. It exhibits high computational stability, is not easily affected by numerical issues such as rounding errors, and features fast convergence. It can achieve high approximation accuracy with lower-order polynomials, thereby helping to reduce computational complexity. Compared with other collocation methods, the LGL rule is a high-order integral approximation method that achieves higher accuracy with a larger integral step size, significantly reducing problem complexity.

### *3.2. Definition and calculation of LGL points*

The collocation method is an enhanced approach combining region-specific LGL collocation, adaptive residue-driven densification, and subdomains-weighted loss optimization. Its core is to decompose the time domain into subdomains, first generate the initial LGL collocation points in each subdomain, then densify high-error regions based on the residuals, and finally minimize the sum of DAE residuals and constraint losses at the collocation points. The steps are as follows:

(1) Generate initial collocation points. The LGL collocation points are the roots of the derivative of the Legendre polynomial over the interval [-1, 1]. For the k-th subdomain $\mathcal{T}_k$, we first solve for the roots of the derivative of the $N_k^{(0)}$-th order Legendre polynomial, which satisfy

$$(1 - x^2)P'_{N_k^{(0)}}(x) = 0.$$

These roots are listed as

$$\tilde{x}_i \in [-1, 1], \quad i = 1, \ldots, N_k^{(0)}.$$

Coordinate transformation to the $k$-th subdomain is given by

$$x_{k,i}^{(0)} = \frac{1}{2}\left[\tilde{x}_i \cdot (t_{k,\text{end}} - t_{k,\text{start}}) + t_{k,\text{start}} + t_{k,\text{end}}\right], \tag{3.2}$$

where $x_{k,i}^{(0)}$ represents the $i$-th initial LGL node in the $k$-th subdomain; $\tilde{x}_i$ represents the LGL node within the standard interval [-1, 1]; and $t_{k,\text{start}}, t_{k,\text{end}}$ represents the start and end times of the $k$-th subdomain.

The final initial collocation point set for the $k$-th subdomain is given by

$$\mathcal{X}_k^{(0)} = \left\{x_{k,i}^{(0)} \mid i = 1, \ldots, N_k^{(0)}\right\},$$

for example, when $k = 2, N_2^{(0)} = 40$, this means 40 initial LGL collocation points are generated.

(2) Sub-domain adaptive collocation densification. First, calculate the combined residuals for the initial collocation points of the $k$-th subdomain, which integrate DAE residuals and exact solution errors, and densify encrypt high-error regions based on collocation residuals. The combined residual is defined as

$$\text{res}(t; \theta_k) = \sqrt{\text{res}_{\text{dae}}(t)^2 + \omega_{\text{data}}\text{res}_{\text{data}}^2}, \tag{3.3}$$

where $\omega_{\text{data}}$ represents the weighting coefficient of the data residuals, which is used to balance the proportion of the residuals of the DAE equation and the data fitting residuals; $\text{res}_{\text{dae}}(t)$ represents the residuals of the differential algebraic equation at time $t$; $\text{res}_{\text{data}}$ represents the data fitting residuals; and $\theta_k$ represents the network parameters of the $k$-th sub-domain.

Next, set a residual threshold $\tau_k$ and select high-error collocation points that satisfied $res(t;\theta_k) > \tau_k$, insert new collocation points to the intervals adjacent to these high-error points in both normal and critical sub-domain; merge the initial collocation points with the new ones, remove duplicates to obtain the final collocation point set, and limit the maximum number of collocation points.

(3) Construct a weighted loss function for the final collocation point set in the k-th subdomain, which enforces the collocation points to satisfy DAE constraints, initial conditions, boundary conditions, and exact solution supervision, with the goal of minimizing this loss function. The total loss function is defined as

$$\mathcal{L}_k(\theta_k) = \begin{cases} \mathcal{L}_{\text{dae},a} + \mathcal{L}_{\text{data},a} + \mathcal{L}_{\text{IC},a} + \mathcal{L}_{\text{BC},a} + \mathcal{L}_{\text{local},a}, \\ \mathcal{L}_{\text{dae},k} + \mathcal{L}_{\text{data},k} + \mathcal{L}_{\text{IC},k} + \mathcal{L}_{\text{BC},k}, \end{cases} \tag{3.4}$$

where $\mathcal{L}_k(\theta_k)$ represents the total loss of the kth subdomains. $a$ denotes critical sub-domain, $k$ denotes other sub-domains. $\mathcal{L}_{\text{local},a}$ represents additional constraints for critical sub-domains, including boundary continuity loss, physical constraint loss, and supervision of local high-mutation regions. The total loss is minimizing using gradient descent to solve for the optimal network parameters

$$\theta_k^* = \arg\min_{\theta_k} \mathcal{L}_k(\theta_k). \tag{3.5}$$

(4) After training each subdomain, concatenate the prediction results of the collocation points from all subdomains to obtain the global solution.

LGL points are automatically densely distributed at interval endpoints and in discontinuity regions. Through spatial adaptation and loss weighting, we increase the initial number of points, enhance residual weights, densify local points, and extend training iterations to improve the fitting accuracy of the discontinuous regions. This is also the core reason why the LGL collocation method can accurately capture mutations.

Finally, the framework of the LGL-PINN method is shown as Figure 1.

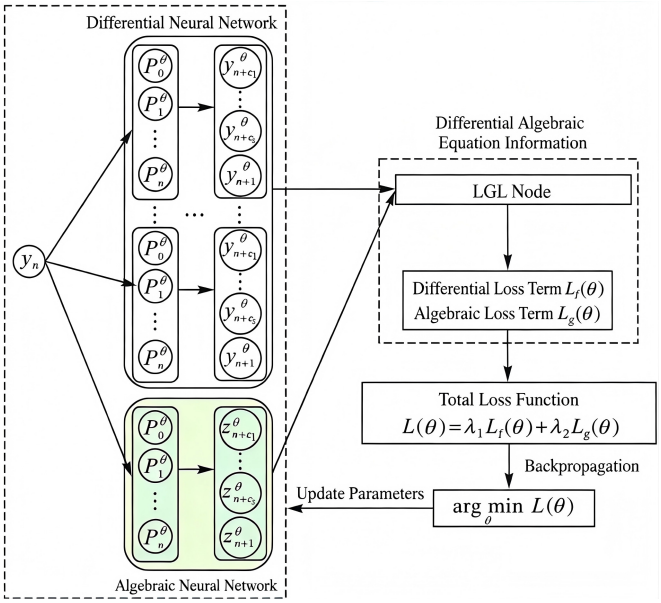


**Figure 1.** Framework of LGL-PINN.

### *3.3. PINN-based optimization algorithm for solving DAE*

#### 3.3.1. Time region decomposition

This section adopts the time domain decomposition scheme to partition the time domain to be solved into multiple sub-regions. This method renders the neural network more adaptable to the step-size iterative solution of the LGL collocation method and resolves issues such as the inadequate model capacity of the original PINN framework when predicting coupled DAE systems. By partitioning the solution time domain into multiple sub-regions and assigning each sub-domain network to handle a specific sub-time region after partitioning, the neural network can more precisely capture the local features and nonlinear relationships within the DAE systems. Consequently, it enhances the method's expressive ability and approximation accuracy, enabling more accurate simulation and solution of complex problems and improving the solution accuracy and efficiency.

Specifically, the solution time domain $t$ is partitioned into $M$ equal-time-step regions according to the time step $\Delta t$ employed. Subsequently, each time-step region is assigned a parallel differential and algebraic neural network. Ultimately, all the neural networks within the solution time domain $t$ are concatenated to acquire the overall LGL-PINN, as depicted in Figure 2. All neural networks in the time domain share identical adjustable parameters. This methodology not only elevates the accuracy of the solution but also augments the method's flexibility and adaptability, thereby enabling the neural network to be more effectively applied to the task of resolving the DAE.

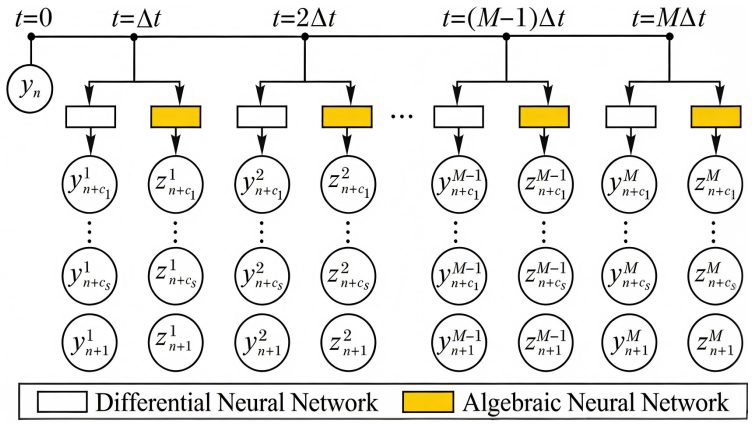


**Figure 2.** Shows the structure of the time domain decomposition scheme [38].

#### 3.3.2. High-precision continuous-time prediction

The error in predicting and solving DAE systems via neural networks is influenced not only by the local truncation error inherent in the numerical method itself but also by the approximation ability of the neural network, and disparities in the training data. These errors accumulate as the sampling time progresses, leading to an augmentation of error across the entire time domain. To fulfill the requirement for high-precision solutions, this research employs a high-precision continuous-time prediction strategy during the prediction process to mitigate the accumulation of errors induced by time, thus enhancing the accuracy of the entire solution time domain.

The fundamental concept of this strategy is to utilize the output values derived from the training

of the preceding sub-time domain network as the input for training the subsequent sub-time domain network. These are then solved sequentially in accordance with the time domain order to acquire the predicted approximate solution for the entire time span. This strategy alleviates the accumulation of time integration errors and meets the solution requirements of various time ranges. It enables the predicted approximate solution to more accurately depict the dynamic behavior of the DAE system and improves the accuracy of the solution procedure. Furthermore, this strategy can fully exploit the features of the LGL collocation method, demonstrating greater flexibility and high controllability in the selection of time steps, and providing a more reliable and precise solution for DAE solving.

## 4. Numerical experiments

This chapter presents numerical examples of differential-algebraic equations to verify the effectiveness of the LGL-PINN in solving DAE systems. The $L_2$ relative error is introduced to quantify the overall discrepancy between the method's predicted solution and the exact solution.

### *4.1. Example 1*

The following simulates a DAE system for a circuit consisting of resistors (R), inductors (L), and capacitors (C) connected in series (series RLC circuit), with a switch

$$\begin{cases} \dfrac{\mathrm{d}i_L(t)}{\mathrm{d}t} = \dfrac{U_s - R(t)i_L(t) - u_C(t)}{L}, \\ \dfrac{\mathrm{d}u_C(t)}{\mathrm{d}t} = \dfrac{i_C(t)}{C}, \\ i_C(t) = i_L(t) - \dfrac{u_C(t)}{R(t)}, \end{cases} \tag{4.1}$$

where $L = 1.0H$ is the inductance value, $C = 0.01F$ is the capacitance value, $U_s = 10.0V$ is the supply voltage, $i_L(t)$ is the inductor current with unit $A$, $u_C(t)$ is the capacitor voltage with unit $V$, $i_C(t)$ is the capacitor current with unit $A$, and $R(t)$ is the resistive value that switches with time with unit $\Omega$.

The first equation represents the kirchhoff's voltage law (KVL) constraint, and the third equation represents the kirchhoff's current law (KCL) constraint. The resistance $R(t)$ changes with time as follows

$$R(t) = \begin{cases} R_1 = 4.0, & t < 1.0, \\ R_2 = 8.0, & t \geq 1.0. \end{cases}$$

Inductor current $i_L(t)$, capacitor voltage $u_C(t)$, and capacitor current $i_C(t)$ repsent the state variables. Supply voltage $U_s = 10.0$, inductance $L = 1.0$, and capacitance $C = 0.01$. Given $t \in [0, 3]$, the initial conditions are employed as

$$i_L(0) = 0, u_C(0) = 0, i_C(0) = 0.$$

This research compares the PINN and the LGL-PINN. The PINN adopts a deep neural network with 3–9 hidden layers, and uses the LeakyReLU activation function. The total loss function is expressed as

$$\mathcal{L} = \lambda_{\text{data}}\mathcal{L}_{\text{data}} + \lambda_{\text{dae}}\mathcal{L}_{\text{dae}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}}. \tag{4.2}$$

Data loss is defined as

$$\begin{aligned}\mathcal{L}_{\text{data}} = &\frac{1}{|\Omega_{\text{mask}}|}\sum_{t\in\Omega_{\text{mask}}}\left(\hat{i}_L(t) - i_L^*(t)\right)^2 \\ &+ \frac{1}{|\Omega_{\text{mask}}|}\sum_{t\in\Omega_{\text{mask}}}\left(\hat{u}_C(t) - u_C^*(t)\right)^2 + \frac{1}{|\Omega_{\text{mask}}|}\sum_{t\in\Omega_{\text{mask}}}\left(\hat{i}_C(t) - i_C^*(t)\right)^2,\end{aligned} \tag{4.3}$$

where $\Omega_{\text{mask}}$ denotes 10% of randomly selected training points. Symbols $i_L^*(t)$, $u_C^*(t)$, and $i_C^*(t)$ are the high-precision reference solutions obtained using the fourth-order Runge-Kutta method. DAE residual loss is defined as

$$\mathcal{L}_{\text{dae}} = \mathcal{L}_{\text{res,d}_1} + \mathcal{L}_{\text{res,d}_2} + \mathcal{L}_{\text{res,a}}. \tag{4.4}$$

Inductance differential equation residual is given as

$$\mathcal{L}_{\text{res,d}_1} = L\frac{d\hat{i}_L}{dt} + R(t)\hat{i}_L + \hat{u}_C - U_s. \tag{4.5}$$

Capacitance differential equation residual is defined as

$$\mathcal{L}_{\text{res,d}_2} = C\frac{d\hat{u}_C}{dt} - iC. \tag{4.6}$$

The definition of current conservation algebraic residual is as follows

$$\mathcal{L}_{\text{res},a} = \hat{i}_C - \left(\hat{i}_L - \frac{\hat{u}_C}{R(t)}\right), \tag{4.7}$$

where $\hat{i}_L$ is the inductor current predicted by PINN, $\hat{u}_C$ represents the capacitor voltage predicted by PINN, $\hat{i}_C$ denotes the capacitor current predicted by PINN, $\mathrm{d}^2\hat{i}_L/\mathrm{d}t^2$ is the time derivative of the predicted inductor current, and $\mathrm{d}\hat{u}_C/\mathrm{d}t$ is the time derivative of the predicted capacitance voltage. Initial condition loss is defined as

$$\mathcal{L}_{\text{IC}} = (\hat{i}_L(0) - 0)^2 + (\hat{u}_C(0) - 0)^2. \tag{4.8}$$

It is noteworthy that this research employs the four-order Runge-Kutta method, which is a classic and extensively utilized explicit single-step numerical integration algorithm. This choice is made because there exists no precise analytical solution for the series RLC circuit system equipped with a switch.

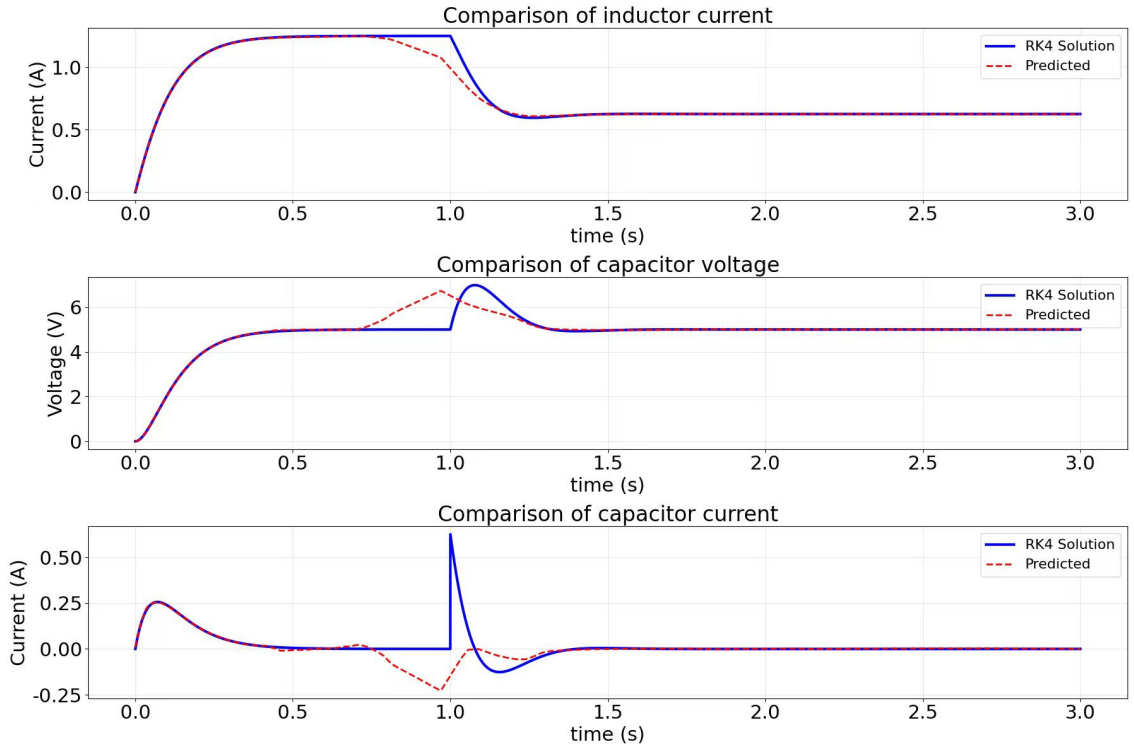


**Figure 3.** Comparison between the reference solution and the predicted solution of the PINN for Example 1.

From the Figure 3, it can be discerned that for both the differential variables and the algebraic variable, the PINN demonstrates extremely large errors at the switching point, with the magnitudes of the absolute errors ranging from $10^{0}$ to $10^{-1}$. During the stationary phase, the magnitudes of the absolute errors range from $10^{-2}$ to $10^{-3}$.

Subsequently, the LGL-PINN method has been used. The primary challenge in the switching RLC circuit lies in the solution discontinuity induced by the resistance switch at $t = 1.0$. We employ time-domain decomposition, fitting each sub-domain with an independent PINN. This method emphasizes optimizing the abrupt region while maintaining a balance between global efficiency and local accuracy. We concentrate on the abrupt region and allocate the majority of computational resources to the critical interval (1.00–1.25) surrounding the switch. The learning rate was set to 5e–5, the number of iterations was 120,000, the weight decay was 3e–6, and the scheduler was selected as MultiStepLR. The transition zones (0.50–1.00 and 1.25–1.50) serve as intermediaries to guarantee continuity between the abrupt region and the stationary segments. The learning rate was set to 6e–5, the number of iterations was 60,000, the weight decay was 2e–6, and the scheduler was selected as MultiStepLR. Stationary segments distant from the abrupt point (e.g., 0.00–0.50 and 1.50–3.00) are partitioned into larger sub-domains to decrease the number of models and enhance efficiency. The learning rate was set to 6e–5, the number of iterations was 30,000, the weight decay was 1e–6, and the scheduler was selected as StepLR. In this way, the training error of a single sub-domain does not propagate globally. Even when the training accuracy of a stationary segment is

slightly lower, it does not affect the core results of the abrupt region.

The Leaky-ReLU activation function is chosen to compute derivatives efficiently and precisely through automatic differentiation. To conduct a more in-depth analysis, the following systematic investigations are carried out to quantify its predictive accuracy for diverse neural network architectures.

**Table 1.** Comparison of $L_2$ errors of predicted solutions of LGL-PINN with different numbers of hidden layers and different numbers of neurons per layer for Example 1.

| Layers \ Neurons | 128 | 256 | 512 |
|---|---|---|---|
| 3 | 1.12E-02 | 9.44E-03 | 6.39E-03 |
| 5 | 1.34E-02 | 3.54E-03 | 1.30E-03 |
| 7 | 5.09E-03 | 5.14E-03 | 1.23E-03 |
| 9 | 3.53E-03 | 1.17E-03 | 1.10E-03 |

Table 1 presents the resultant relative $L_2$ errors corresponding to varying numbers of hidden layers and distinct numbers of neurons per layer. As anticipated, it is observed that with an increase in the number of layers and neurons consequently enhancing the neural network's capacity to approximate more intricate functions, the predictive accuracy is augmented.

**Table 2.** Comparison of training times of predicted solutions of LGL-PINN with different numbers of hidden layers and different numbers of neurons per layer for Example 1.

| Layers \ Neurons | 128 | 256 | 512 |
|---|---|---|---|
| 3 | 38 min | 41 min | 50 min |
| 5 | 45 min | 50 min | 68 min |
| 7 | 53 min | 60 min | 88 min |
| 9 | 59 min | 70 min | 88 min |

As presented in Table 2, it can be seen that as the complexity of the network architecture increases, the corresponding training time shows an increasing trend. The acquired solutions are compared with the predictions of the optimal LGL-PINN, and the comparison between the reference solution and the predicted solution is presented in Figure 4.

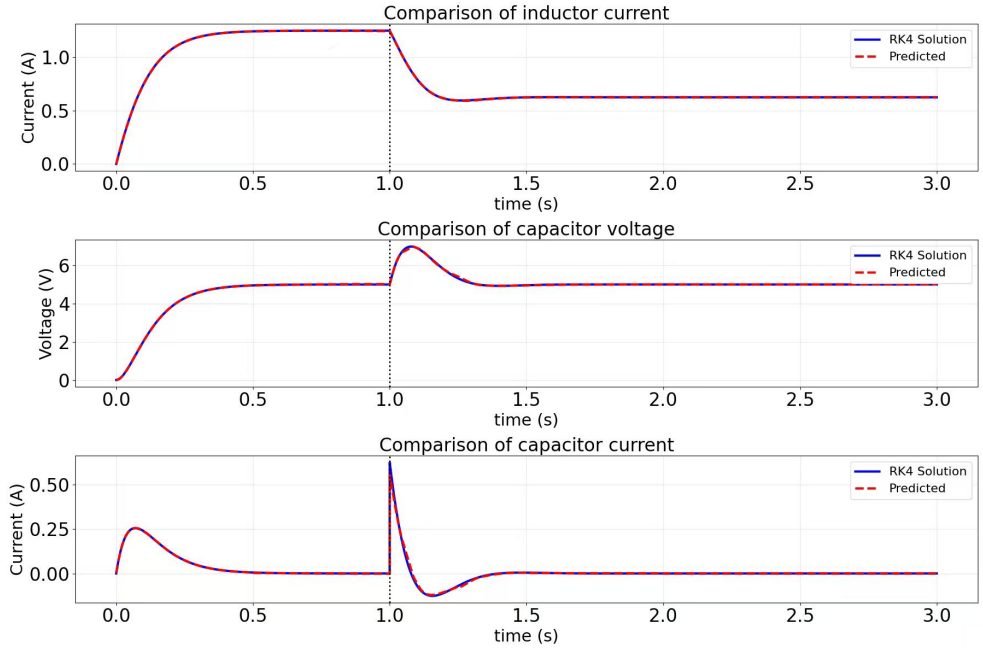


**Figure 4.** Comparison between the reference solution and the predicted solution of the LGL-PINN for Example 1.

It can be observed from Figure 4 that the predicted solution of the LGL-PINN is highly coincident with the reference solution image. The absolute errors of the three predicted variables remain at a low level, with magnitudes ranging from $10^{-2}$ to $10^{-4}$. As the number of iterations increases, the predicted values converge to the true values, and the absolute error curves stabilizes within one order of magnitude.

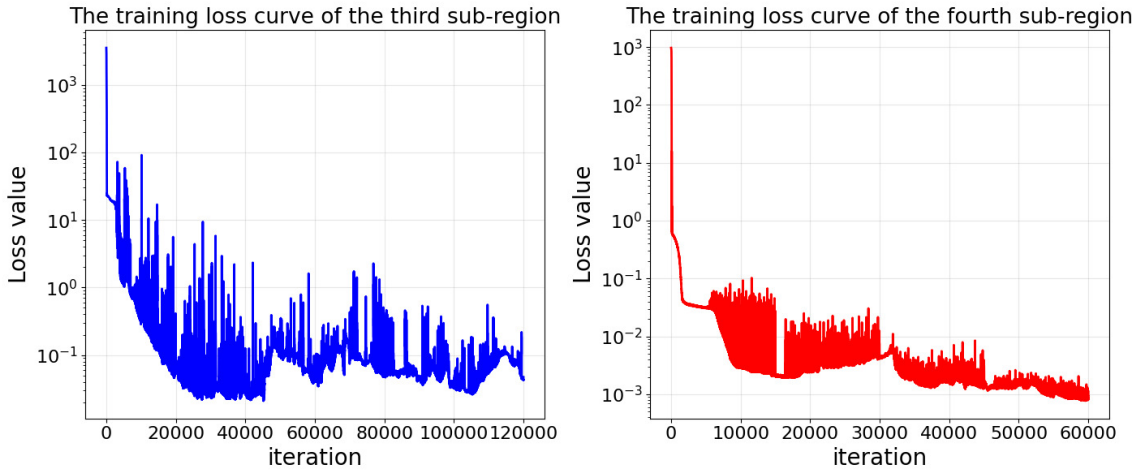


**Figure 5.** Training iteration performance in regions 3 and 4 for Example 1.

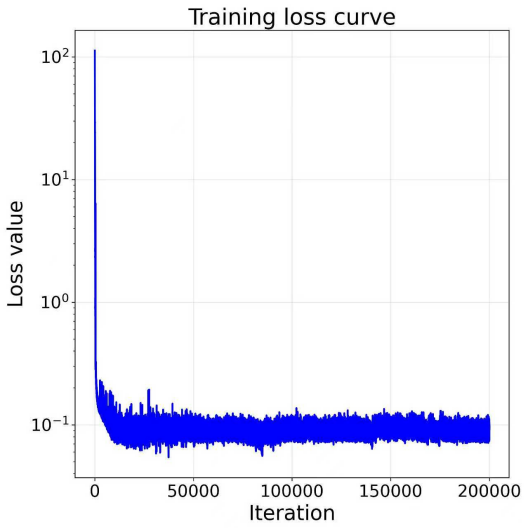


**Figure 6.** Training iteration performance of the PINN for Example 1.

From the Figure 5, it is evident that the loss of the third sub-region near the switching point of the LGL-PINN experiences a rapid decline during the early training stage. This phenomenon suggests that the method promptly captures the data features and physical constraints of this sub-region in the initial phase. Subsequently, the loss shows continuous fluctuations in the later training phase. In the early training stage, the loss of the fourth sub-region near the switching point decreases extremely rapidly, indicating that the method attains a higher learning efficiency in this sub-region. Moreover, the subsequent convergence process of the model in this sub-region is notably smoother. From the Figure 6, the PINN demonstrates rapid learning in the early training stage and stable convergence in the later stage. Under the current training settings, it has achieved satisfactory fitting results.

**Table 3.** $L_2$ Relative error comparison between exact solutions and predicted solutions of the two methods for Example 1.

| Method | $i_L(t)$ | $u_C(t)$ | $i_C(t)$ |
|---|---|---|---|
| PINN | 4.48E-02 | 3.81E-01 | 7.49E-02 |
| LGL-PINN | 2.92E-04 | 1.37E-03 | 1.62E-03 |

As presented in Table 3, the LGL-PINN attains lower absolute and relative errors for both differential and algebraic variables within the entire solution domain. The prediction accuracy of the LGL-PINN is at least one order of magnitude superior to that of the PINN. This phenomenon primarily stems from the fact that the LGL-PINN not only integrates physical constraints through the PINN framework but also combines LGL spectral methods and region-adaptive strategies. Meanwhile, it adopts time-domain decomposition and continuous-time prediction schemes capable of simultaneously capturing diverse attributes of the system. This empowers the method to more effectively capture the inter-regional coupling relationships and alleviate the error accumulation

problem resulting from time evolution.

We also discussed the impact of the number of LGL nodes on the error and training time.

**Table 4.** Influence of different LGL node numbers on $L_2$ error and training time for Example 1.

| Total number of nodes | $L_2$ error | Training time |
|---|---|---|
| 4652 | 4.2E-03 | 48 min |
| 6321 | 7.1E-03 | 48 min |
| 7822 | 2.3E-03 | 50 min |
| 9461 | 2.6E-03 | 51 min |
| 11,082 | 4.5E-03 | 53 min |
| 12,715 | 3.6E-03 | 55 min |

Due to the complexity of the circuit system, when the initial distribution points are different, the total number of nodes will not be regular. We conducted experiments using a network structure with 5 hidden layers and 256 neurons per layer. It can be observed from Table 4 that as the number of LGL nodes increases, the training time of the model will rise accordingly; however, the prediction error does not continue to decrease with the increase in the number of nodes. Instead, there exists a relatively optimal node configuration.

### *4.2. Example 2*

The subsequent presents the DAE equation set for the bouncing ball model featuring rigid constraints.

$$\begin{cases} \dfrac{\mathrm{d}y(t)}{\mathrm{d}t} = v(t), \\ \dfrac{\mathrm{d}v(t)}{\mathrm{d}t} = -g + \lambda(t), \\ y(t) \geq 0, \\ \lambda(t) \geq 0, \\ \lambda(t)y(t) = 0, \end{cases} \tag{4.9}$$

where $y(t)$ represents the vertical displacement of the ball, with the unit being meters. $v(t)$ represents the vertical velocity of the ball, with units of m/s, $g = 9.8$ m/s represents the gravitational acceleration, $\lambda(t)$ represents the normal ground constraint force, and the complementary constraint ensures that the ball does not penetrate the ground.

The initial conditions of the model are set as follows

$$y(0) = y_0 = 1, \ v(0) = 0.$$

The initial stage of the ball's motion is characterized by free-fall. The ball is released from its initial height with an initial velocity of zero and is solely under the influence of gravitational force, undergoing uniformly accelerated linear motion. The displacement of the ball exhibits a quadratic decrease with respect to time, which is in accordance with the free-fall displacement formula. The magnitude of the

ball's velocity increases linearly with time in the downward direction. $\lambda$ manifests as an impulse-type instantaneous force that functions to ensure the conservation of momentum during elastic collisions. Its physical significance pertains to the normal supporting force exerted by the ground on the object, at which moment $\lambda = 0$.

The second stage pertains to the instantaneous motion during collision. The object arrives at the ground at time $t_c$, where

$$y(t_c) = 0.$$

At this specific instant

$$\lambda(t) = \delta(t - t_c) \cdot (1 + e)\sqrt{2gy_0},$$

in which $\delta(t - t_c)$ represents the Dirac delta function, the coefficient of restitution $e$ is set to 0.9.

Upon colliding with the ground and rebounding, the ball ascends and may subsequently descend once more. In the code, the vertical displacement of the ball subsequent to the rebound is approximated as a quadratic function, with the function coefficients ($e^2$) representing the energy dissipation resulting from the collision, at which juncture $\lambda = 0$. Following the collision, the direction of the ball's velocity is reversed, and the magnitude of the velocity becomes e times that prior to the collision

$$v(t^+) = -e * v(t^-).$$

The ball's velocity then changes linearly with time decelerating upward under the action of gravitational force. The collision time of the ball is calculated as follows

$$t_c = \sqrt{\frac{2y_0}{g}}.$$

During the free-fall phase,

$$\begin{cases} y(t) = y_0 - \frac{1}{2}gt^2, & 0 < t < t_c, \\ v(t) = -gt, & 0 < t < t_c. \end{cases} \tag{4.10}$$

After the collision,

$$\begin{cases} y(t) = \dfrac{1}{2}gt^2(t - t_c)^2, & t > t_c, \\ v(t) = g \cdot e(t - t_c), & t > t_c. \end{cases} \tag{4.11}$$

In the context of inequality constraints, the PINN employs the core concept of the penalty method in conjunction with gradient descent. This approach transforms the fulfillment of inequality constraints into the minimization of penalty losses. For each inequality constraint ($c_i(x) \geq 0$), the degree of constraint violation is quantified as optimizable scalar loss, and the penalty function $P_i(x)$ is defined as

$$P_i(x) = \omega_i \frac{1}{N} \sum_{k=1}^{N} \left[\text{Softplus}\left(-c_i(x_k) - \epsilon\right)\right]^2. \tag{4.12}$$

Among them,

$$Softplus(z) = \frac{1}{\beta} \ln\left(1 + e^{\beta z}\right),$$

which is a smooth approximation of ReLU. $\omega_i$ is the penalty weight, $x_k$ is the method output of the k-th training point, and $\epsilon$ is the numerical stability term, tolerating minor floating-point errors. Inequality constraints must be integrated with complementary constraints to establish a comprehensive constraint system. The complementary penalty function is defined as

$$P_{\text{comp}} = \omega_{\text{comp}} \frac{1}{N} \sum_{k=1}^{N} (y(t_k)\lambda(t_k))^2. \tag{4.13}$$

The total loss is composed of the inequality penalty, complementary penalty, differential equation loss, and initial condition loss. It can be expressed as

$$\mathcal{L} = \lambda_{\text{data}}\mathcal{L}_{\text{data}} + \lambda_{\text{dae}}\mathcal{L}_{\text{dae}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \sum_{i=1}^{M} P_i(x) + P_{\text{comp}}. \tag{4.14}$$

The data loss term of the loss function is defined as

$$\mathcal{L}_{\text{data}} = \frac{1}{N} \sum_{i=1}^{N} \left[ (\hat{y}(t_i) - y(t_i))^2 + (\hat{v}(t_i) - v(t_i))^2 \right]. \tag{4.15}$$

Among these terms, $\hat{y}(t_i)$ and $\hat{v}(t_i)$ denote the predicted position and velocity of the network, while $y(t_i)$ and $v(t_i)$ represent the position and velocity of the exact solution.

The DAE equation residual loss term is presented as

$$\mathcal{L}_{\text{dae}} = \frac{1}{N} \sum_{i=1}^{N} \left[ \left( \frac{\mathrm{d}y}{\mathrm{d}t} - v(t) \right)^2 + \left( \frac{\mathrm{d}v}{\mathrm{d}t} + g - \lambda(t) \right)^2 \right]. \tag{4.16}$$

The initial condition loss term is defined as

$$\mathcal{L}_{\text{ic}} = (\hat{y}(0) - y_0)^2 + (\hat{v}(0) - v_0)^2 + \lambda(0)^2. \tag{4.17}$$

First, the PINN is adopted for prediction, which employs a fully connected neural network with 2 hidden layers and 100 neurons per hidden layer. The tanh activation function is selected for the network, the Adam optimizer is used for parameter updating, and the ReduceLR On Plateau strategy is adopted for learning rate scheduling. The comparison between the predicted solution and the exact solution of the PINN is presented in Figure 7.

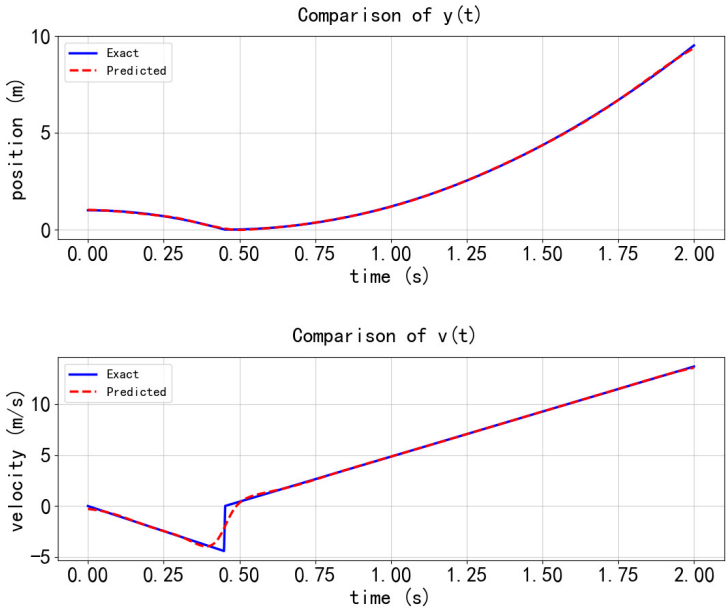


**Figure 7.** Comparison between exact solutions and predicted solutions of the PINN for Example 2.

As shown in Figure 7, the PINN shows a notable difference between the predicted velocity and the exact velocity at the collision point, and the absolute error accuracy of the predicted solution is at the level of $10^0 - 10^{-1}$. To enhance the model's capacity to capture the characteristics of $v(t)$, we strengthen the application of the LGL collocation method. We continue to employ the time-domain decomposition strategy. In the discontinuous region covering the interval $[t_c - 0.1, t_c + 0.1]$, 200 uniformly distributed collocation points are generated. The remaining 300 collocation points are evenly distributed over the intervals $[t_0, t_c - 0.1]$ and $[t_c + 0.1, t_{end}]$, thus guaranteeing global time-domain coverage. The learning rate is set to $1e - 3$, the number of iterations is 5000, and the scheduler is chosen as ReduceLR On Plateau with a decay coefficient of 0.5. Through dense sampling in key regions, the neural network acquires more training information near the discontinuity points, allowing the training points to better capture the local characteristics of $v(t)$. We still quantify its predictive accuracy under different neural network architectures.

**Table 5.** Comparison of $L_2$ errors of predicted solutions of LGL-PINN with different numbers of hidden layers and different numbers of neurons per layer for Example 2.

| Layers \ Neurons | 100 | 150 | 200 | 250 |
|---|---|---|---|---|
| 2 | 2.39E-02 | 1.55E-02 | 1.39E-02 | 1.28E-02 |
| 4 | 6.52E-03 | 3.84E-02 | 4.41E-03 | 2.36E-02 |
| 6 | 3.81E-02 | 1.13E-02 | 7.26E-03 | 7.24E-03 |
| 8 | 1.28E-02 | 1.33E-02 | 7.54E-03 | 3.00E-02 |

Table 5 presents the resultant relative $L_2$ errors corresponding to varying numbers of hidden layers and distinct neuron counts per layer. Through comprehensive experimentation, the optimal performance was attained when the network was configured with four hidden layers and two hundred neurons per hidden layer.

**Table 6.** Comparison of training times of predicted solutions of LGL-PINN with different numbers of hidden layers and different numbers of neurons per layer for Example 2.

| Layers \ Neurons | 100 | 150 | 200 | 250 |
|---|---|---|---|---|
| 2 | 5 s | 6 s | 7 s | 8 s |
| 4 | 9 s | 11 s | 14 s | 17 s |
| 6 | 12 s | 17 s | 21 s | 27 s |
| 8 | 16 s | 22 s | 28 s | 37 s |

From the Table 6, it can be seen that as the complexity of the network architecture increases, the corresponding training time shows an increasing trend. Furthermore, there is a significant difference in the calculation time between the two cases, and the main reason for this is the varying complexity of the events they correspond to.

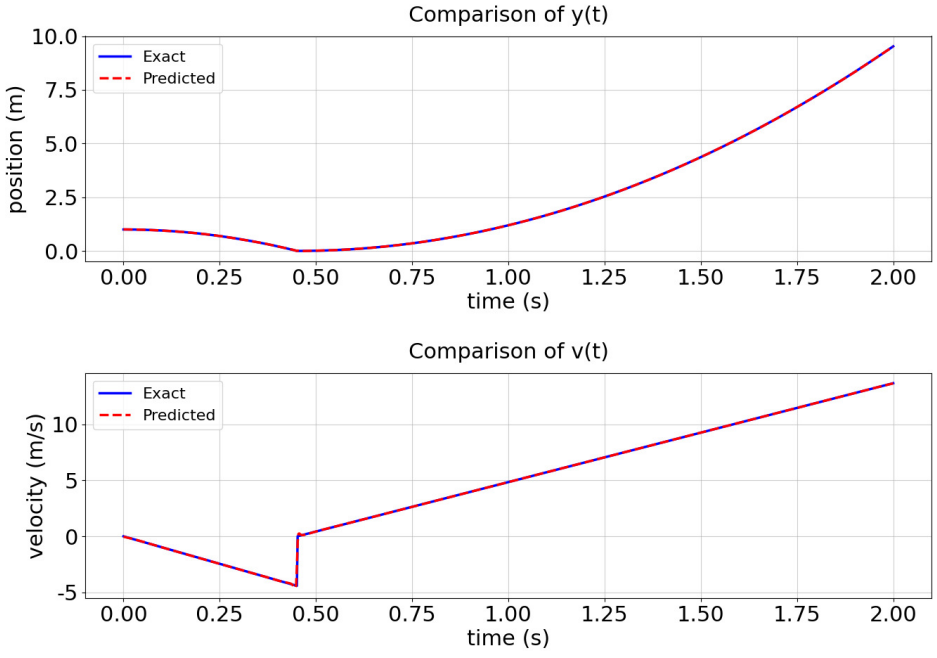


**Figure 8.** Comparison between exact solutions and predicted solutions of the LGL-PINN for Example 2.

It can be seen from Figure 8 that the images of displacement $y(t)$ versus velocity $v(t)$ for the predicted solutions and exact solutions of the LGL-PINN are in excellent agreement.

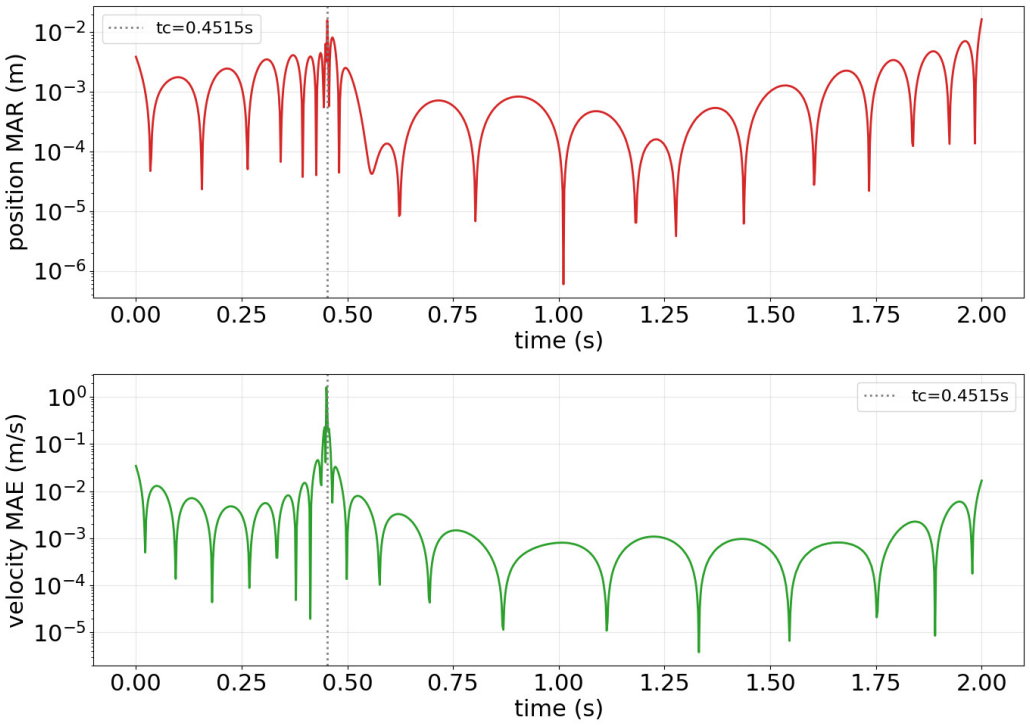


**Figure 9.** Error distribution plots of the LGL-PINN for Example 2.

As shown in Figure 9, the absolute error of the predicted solution attains an accuracy level ranging from $10^{-2}$ to $10^{-3}$. Moreover, as the number of iterations increases, the solution converges to the true value. Simultaneously, the variation of the absolute error curve stabilizes within one order of magnitude.

**Table 7.** $L_2$ Relative error comparison between exact solutions and predicted solutions of the two methods for Example 2.

| Method | $y(t)$ | $v(t)$ |
|---|---|---|
| PINN | 1.58E-02 | 2.52E-01 |
| LGL-PINN | 2.21E-03 | 6.63E-03 |

It can be observed from Table 7 that the prediction precision of the LGL-PINN is at least one order of magnitude greater than that of the PINN. This further validates the superiority of LGL-PINN in resolving DAE encompassing discontinuous events.

We also discussed the impact of the number of LGL nodes on the error and training time.

**Table 8.** Influence of different LGL node numbers on $L_2$ error and training time for Example 2.

| Total number of nodes | $L_2$ error | Training time |
|---|---|---|
| 200 | 6.9E-03 | 11 s |
| 400 | 6.2E-03 | 15 s |
| 600 | 5.0E-03 | 16 s |
| 800 | 5.5E-03 | 19 s |
| 1000 | 2.2E-02 | 22 s |

For the bouncing ball system, we conducted experiments using a network structure with 4 hidden layers and 200 neurons per layer. It can be observed from Table 8 that as the number of LGL nodes increases, the training time of the model will rise accordingly; however, the prediction error does not continue to decrease with the increase in the number of nodes. Instead, there exists a relatively optimal node configuration.

## 5. Conclusions

A physics-informed neural network founded on the LGL-PINN is introduced in this article, which demonstrates high-precision resolution abilities for discontinuous events and adaptability to complex constraints. Simultaneously, a time-domain decomposition scheme is employed to refine the neural network in the time domain, thus enabling high-precision continuous-time prediction. Numerical examples illustrate that the LGL-PINN enhances the solution accuracy by one to two orders of magnitude in comparison with the PINN, achieving high-precision and high-stability continuous-time solutions for DAE. Consequently, the LGL-PINN method offers a viable high-precision solution approach for solving challenging DAE systems featuring discontinuous events.

It is worth noting that the research presented in this paper unavoidably exhibits certain limitations and deficiencies. Although the LGL-PINN method has attained remarkable outcomes in terms of computational precision and stability, several issues still await further exploration. First, the LGL-PINN method, when dealing with complex differential-algebraic equations, needs to increase the number of LGL nodes to ensure accuracy, which leads to an increase in training parameters, an increase in computational cost, and an extension of convergence time. This makes its applicability limited in real-time demanding engineering scenarios; Second, this method is currently mainly verified for medium complexity linear and weakly nonlinear DAEs. The accuracy, convergence stability, and robustness of this method in complex physical systems with strong stiffness, high-order coupling, or discontinuous solutions still need to be fully verified.

At the same time, note the limitations of the current research, and further discussion on future directions will include expanding the application scope, optimizing the model structure and computational strategies, and integrating traditional numerical methods, in order to continuously improve the solution efficiency, applicable scenarios, and robustness of the method.

## Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

## Acknowledgments

This work was supported in part by the Foundation of Shandong Jianzhu University under Grants H21010Z, H22135Z, and XJG2023034; in part by the Ministry of Education's Industry University Cooperation Collaborative Education Project under Grant 231107099132923; and in part by the 2023 Jinan City School Integration Development Strategy Project: Research and Demonstration of a Federated Learning Open Platform for E-Commerce Recommendation under Grant JNSX2023064; in part by 2025 new university 20 projects in Jinan city: Research and application of high precision ceramic additive manufacturing based on DLP technology under Grant 202534031; in part by 2025 Shandong province graduate high quality professional degree teaching case library project under Grant SDYAL2025080; and in part by 20 new university projects in Jinan City: Research on Traffic Engineering Monitoring Based on Satellite Remote Sensing under Grant 202333019.

## Conflict of interest

The authors declare there is no conflict of interest.

## References

1. E. Haug, An index 0 differential-algebraic equation formulation for multibody dynamics: Holonomic constraints, *Mech. Based Des. Struct. Mach.*, **45** (2017), 479–506. https://doi.org/10.1080/15397734.2016.1246370
2. R. März, C. Tischendorf, Recent results in solving index-2 differential-algebraic equations in circuit simulation, *SIAM J. Sci. Stat. Comput.*, **18** (1997), 139–159. https://doi.org/10.1137/S1064827595287250
3. S. Campbell, A. Ilchmann, V. Mehrmann, T. Reis, *Applications of Differential-Algebraic Equations: Examples and Benchmarks*, Springer, Cham, 2018. https://doi.org/10.1007/978-3-030-03718-5
4. C. Gear, Simultaneous numerical solution of differential-algebraic equations, *IEEE Trans. Circuits Theory*, **18** (1971), 89–95.
5. U. M. Ascher, L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-algebraic Equations*, Society for Industrial and Applied Mathematics, 1998.
6. U. M. Ascher, L. R. Petzold, Projected implicit runge–kutta methods for differential-algebraic equations, *SIAM J. Numer. Anal.*, **28** (1991), 1097–1120. https://doi.org/10.1137/0728059
7. J. R. Cash, Modified extended backward differentiation formulae for the numerical solution of stiff initial value problems in odes and daes, *J. Comput. Appl. Math.*, **125** (2000), 117–130. https://doi.org/10.1016/S0377-0427(00)00463-5
8. M. Saravi, E. Babolian, R. England, M. Bromilow, System of linear ordinary differential and differential-algebraic equations and pseudo-spectral method, *Comput. Math. Appl.*, **59** (2010), 1524–1531. https://doi.org/10.1016/j.camwa.2009.12.022
9. M. M. Hosseini, Adomian decomposition method for solution of differential-algebraic equations, *J. Comput. Appl. Math.*, **197** (2006), 495–501. https://doi.org/10.1016/j.amc.2006.03.027

10. C. K. Newman, *Exponential Integrators for the Incompressible Navier-stokes Equations*, Virginia Polytechnic Institute and State University, 2003.
11. J. Ding, Z. Pan, Generalized-$\alpha$ projection method for differential-algebraic equations of multibody system dynamics (in Chinese), *Eng. Mech.*, **30** (2013), 380–384.
12. J. Lu, J. Tang, X. Qin, Y. Feng, Improved group-preserving algorithm and its application in chaotic systems (in Chinese), *Acta Phys. Sin.*, **65** (2016), 110501. https://doi.org/10.7498/aps.65.110501
13. C. S. Liu, W. Chen, L. W. Liu, Solving mechanical systems with nonholonomic constraints by a lie-group differential algebraic equations method, *J. Eng. Mech.*, **143** (2017), 04017097. https://doi.org/10.1061/(ASCE)EM.1943-7889.0001298
14. J. Tang, J. Lu, Modified extended lie-group method for hessenberg differential algebraic equations with index-3, *Mathematics*, **11** (2023), 2360. https://doi.org/10.3390/MATH11102360
15. 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. https://doi.org/10.1016/j.jcp.2018.10.045
16. E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnston, D. Mortari, Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations, *Neurocomputing*, **457** (2021), 334–356. https://doi.org/10.1016/J.NEUCOM.2021.06.015
17. M. Yang, J. T. Foster, Multi-output physics-informed neural networks for forward and inverse pde problems with uncertainties, *Comput. Methods Appl. Mech. Eng.*, **402** (2022), 115041. https://doi.org/10.1016/j.cma.2022.115041
18. I. Ali, Advanced machine learning technique for solving elliptic partial differential equations using legendre spectral neural networks, *Electron. Res. Arch.*, **33** (2025), 826–848. https://doi.org/10.3934/era.2025037
19. C. Leake, D. Mortari, Deep theory of functional connections: A new method for estimating the solutions of partial differential equations, *Mach. Learn. Knowl. Extr.*, **2** (2020), 37–55. https://doi.org/10.3390/make2010004
20. H. Liu, H. Liu, J. Xu, L. Li, J. Song, Jacobi neural network method for solving linear differential-algebraic equations with variable coefficients, *Neural Process. Lett.*, **53** (2021), 3357–3374. https://doi.org/10.1007/S11063-021-10543-5
21. C. Moya, G. Lin, Dae-pinn: A physics-informed neural network model for simulating differential algebraic equations with application to power networks, *Neural Comput. Appl.*, **35** (2023), 3789–3804. https://doi.org/10.1007/S00521-022-07886-Y
22. K. Liang, *Research on Solving Methods for Several Types of Partial Differential-algebraic Equations Based on PINNs*, Guangzhou University, 2025. https://doi.org/10.27040/d.cnki.ggzdu.2025.000811
23. C. Yin, H. Hu, A parameter identification model for time-delay chaotic systems based on temporal attention mechanism (in Chinese), *J. Comput. Appl.*, **43** (2023), 842–847. https://doi.org/10.11772/j.issn.1001-9081.2022010122

24. Z. Yang, J. Lan, Y. Wu, Neural network methods for solving several classes of differential-algebraic equations (in Chinese), *Appl. Math. Mech.*, **40** (2019), 115–126. https://doi.org/10.21656/1000-0887.390122
25. J. Chen, *Research on Solving Forward and Inverse Problems of Differential-algebraic Equations Based on Physics-informed Neural Networks*, Guangzhou University, 2025. https://doi.org/10.27040/d.cnki.ggzdu.2025.000786
26. M. H. Hanke, R. März, On the computation of accurate initial conditions for linear higher-index differential-algebraic equations and its application in initial value solvers, *Numerical Algorithms*, (2025), 1–61. https://doi.org/10.1007/S11075-025-02116-7
27. M. V. Bulatov, O. S. Budnikova, Extrapolation multistep methods for numerical solution of linear second-order differential algebraic equations, *Numer. Anal. Appl.*, **18** (2025), 101–118. https://doi.org/10.1134/S1995423925020016
28. C. L. Yang, L. Yang, S. Y. Ren, P. Huo, R. Liang, F. Jiang, et al., Research on transient simulation of partial discharge based on differential-algebraic equation approach, *J. Phys. Conf. Ser.*, **3110** (2025), 012008. https://doi.org/10.1088/1742-6596/3110/1/012008
29. N. T. The, Stochastic differential algebraic equations of index 2, *Lobachevskii J. Math.*, **45** (2025), 6569–6580. https://doi.org/10.1134/S1995080224606726
30. S. R. Saratha, A. Yildirim, A novel approach to fractional differential equations via JSN transform coupled fractional residual power series method, *Phys. Scr.*, **99** (2024), 125285. https://doi.org/10.1088/1402-4896/AD92BB
31. R. Brociek, M. Pleszczyński, Differential transform method (DTM) and physics-informed neural networks (PINNs) in solving integral–algebraic equation systems, *Symmetry*, **16** (2024), 1619. https://doi.org/10.3390/SYM16121619
32. Z. Wu, B. Xie, Z. Y. Yu, Probabilistic interpretation for a system of quasilinear parabolic partial differential-algebraic equations: The classical solution, *Chin. Ann. Math. Ser. B*, **46** (2025), 1–36. https://doi.org/10.1007/S11401-025-0051-Y
33. Y. Guo, Z. Fu, J. Min, S. Lin, X. Liu, Y. F. Rashed, et al., Long-term simulation of physical and mechanical behaviors using curriculum-transfer-learning based physics-informed neural networks, *Neural Networks*, **191** (2025), 107825. https://doi.org/10.1016/j.neunet.2025.107825
34. Z. Fu, W. Xu, S. Liu, Physics-informed kernel function neural networks for solving partial differential equations, *Neural Networks*, **172** (2024), 106098. https://doi.org/10.1016/j.neunet.2024.106098
35. X. Huang, F. Wang, B. Zhang, H. Liu, Enriched physics-informed neural networks for dynamic Poisson-Nernst-Planck systems, *Math. Comput. Simul.*, **237** (2025), 231–246. https://doi.org/10.1016/j.matcom.2025.04.037
36. X. Li, F. Wang, R. Wang, S. Zhao, D. Yang, Fundamental solution neural networks for solving inverse Cauchy problems for the Laplace and biharmonic equations, *Comput. Math. Appl.*, **201** (2026), 1–17. https://doi.org/10.1016/j.camwa.2025.09.032

37. F. Wang, X. Li, H. Liu, L. Qiu, X. Yue, An adaptive method of fundamental solutions using physics-informed neural networks, *Eng. Anal. Boundary Elem.*, **178** (2025), 106295. https://doi.org/10.1016/j.enganabound.2025.106295

38. S. Lai, J. Tang, K. Liang, J. Chen, Solving differential-algebraic equations with pinn based on the lobatto method and legendre polynomials (in Chinese), *J. Comput. Appl.*, **45** (2025), 911–919. https://doi.org/10.11772/j.issn.1001-9081.2024030313


AIMS Press