AIMS Mathematics, 11(6): 16511–16533.
DOI: 10.3934/math.2026677
Received: 30 March 2026
Revised: 04 May 2026
Accepted: 19 May 2026
Published: 09 June 2026

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

Research article

Internal variable reformulation of Volterra integro-differential equations with exponential kernels for physics-informed neural networks

Yongseok Jang*

Department of Mathematics, Chonnam National University, Gwangju, 61186, South Korea

* **Correspondence:** Email: yongseok.jang@chonnam.ac.kr.

Abstract: We developed a physics-informed neural network (PINN) framework for solving integro-differential equations (IDEs), with particular emphasis on Volterra-type problems with exponentially decaying kernels. While PINNs provide a flexible approach for incorporating physical laws without mesh-based discretization, the treatment of convolution integral terms remains computationally demanding. To address this issue, we introduced internal variables for exponential kernels, transforming the original IDE into an equivalent system of differential equations, eliminating the need for explicit quadrature, and significantly reducing computational cost and memory requirements. The proposed method incorporates both differential and integral operators within the PINN framework. Numerical results demonstrate that the method maintains accuracy while significantly improving computational efficiency. It also extends naturally to inverse problems, where viscoelastic parameters are accurately identified from sparse observations.

Keywords: physics-informed neural network; integro-differential equation; exponential kernels; internal variables; viscoelasticity
Mathematics Subject Classification: 45D05, 65N75, 68T20, 74D05

1. Introduction

Physics-informed neural networks (PINNs) provide an effective framework for solving differential equations by embedding physical laws directly into the training process [1, 2]. Compared with traditional numerical methods such as finite difference and finite element methods, PINNs offer a mesh-free approach based on automatic differentiation, which enables flexible approximation of solutions in complex and high-dimensional settings [3].

This flexibility makes PINNs particularly attractive for solving integro-differential equations (IDEs), which arise in models involving memory effects and nonlocal interactions, especially in continuum mechanics and viscoelasticity. A major challenge in solving such IDEs is the treatment

of convolution integral terms, which typically require quadrature-based discretization and storage of past states, leading to significant computational cost and memory demands. Related memory-dependent problems have also been studied using semi-analytical and optimization-based approaches, including differential transform methods for integro-differential equations with delayed arguments [4] and metaheuristic algorithms for inverse problems involving fractional diffusion models [5]. Recently, Gao et al. [6] addressed related memory effects by approximating power-law type kernels with finite sums of exponentially decaying kernels and introducing auxiliary ordinary differential equation (ODE) systems. The present work considers a more general dynamic setting in which the temporal operator includes both first- and second-order derivatives, corresponding to parabolic and hyperbolic Volterra IDEs. The exponential kernels arise directly from the generalized Maxwell viscoelastic model rather than as a kernel approximation, and the resulting internal variable system is therefore an exact equivalent of the governing IDE.

Building on this exact reformulation, we develop a quadrature-free PINN framework. The convolution operator is replaced by internal variables, and the dynamic nature of the model further allows the second-order time derivative to be recast as a first-order system through the introduction of velocity variables. The resulting coupled PDE-ODE system avoids the storage and quadrature of past states, eliminates higher-order temporal differentiation in automatic differentiation, and provides a more favorable optimization landscape for PINN training. The same residual framework extends naturally to inverse problems, enabling parameter identification from sparse observations.

The contributions of this work are fourfold. First, we propose a quadrature-free PINN residual for Volterra IDEs with exponential relaxation kernels, in which the convolution operator is exactly replaced by internal variables in two formulations. Second, exploiting the dynamic nature of the model, we recast the hyperbolic formulation as a parabolic formulation through velocity augmentation and demonstrate that this reformulation substantially improves the accuracy and training stability of the PINN. Third, we examine two complementary internal variable formulations, the displacement form and the velocity form, and discuss the situations in which each is preferable. Fourth, the same residual framework is used without modification for inverse identification of viscoelastic parameters from sparse observations, including the case of noisy data.

Although internal variable reformulations have long been used in finite element methods for dynamic viscoelasticity [7,8], their systematic use inside the PINN residual has not, to our knowledge, been studied before. The empirical comparison between the displacement and velocity forms, and between the second-order and first-order residual variants, is specific to the optimization-based training of PINNs and has no direct counterpart in the finite element setting. We do not claim a universal advantage of PINNs over classical solvers; rather, the proposed reformulation primarily addresses the cost of quadrature in PINN-based training and enables a unified forward–inverse framework in a fully mesh-free setting.

The paper is organized as follows. In Section 2, we introduce IDEs and provide the necessary background in linear viscoelasticity, along with internal variable formulations. Section 3 presents the framework of PINNs and proposes our models for solving IDEs. We demonstrate the effectiveness of the proposed method through various numerical examples in Section 4. Finally, Section 5 concludes the paper by summarizing the key findings and discussing potential avenues for future research.

Notation. In this paper, we adopt the following notation. Scalar quantities are denoted by lowercase letters (e.g., u, v), while vector-valued functions and variables are represented in bold (e.g., $\boldsymbol{u}$, $\boldsymbol{x}$). The

spatial dimension is denoted by d, so that $\boldsymbol{x} \in \mathbb{R}^d$. We use the symbol $*$ to denote the convolution operation, which for two functions f and g is defined as

$$(f * g)(t) = \int_0^t f(\tau)g(t-\tau)\,d\tau = \int_0^t f(t-\tau)g(\tau)\,d\tau.$$

Time derivatives are represented using overdots, so that $\dot{u} = \partial_t u = \partial u/\partial t$ and $\ddot{u} = \partial_{tt} u = \partial^2 u/\partial t^2$. The gradient of a function u with respect to the spatial variable $\boldsymbol{x}$ is denoted by ∇u, and the Laplacian operator is given by $\Delta u = \nabla \cdot \nabla u$.

2. Integro-differential equations

The IDEs arise in mathematical models involving memory effects, where the current state depends on past values through convolution terms. In this work, we consider time-dependent IDEs of Volterra type, which preserve causality and are commonly used in applications, such as viscoelasticity.

We consider the following general form:

$$\mathcal{D}u + \mathcal{L}u + \tilde{\mathcal{L}}(K * u) = f, \qquad \text{for } \boldsymbol{x} \in \Omega,\ t \in (0, T], \tag{2.1}$$

where $\mathcal{D}$, $\mathcal{L}$, and $\tilde{\mathcal{L}}$ denote differential operators in time or space, respectively, and $K(t)$ is a kernel function representing memory effects.

For example, we define the time operator as

$$\mathcal{D}u = a\ddot{u} + b\dot{u} + cu, \tag{2.2}$$

and the spatial operators as

$$\mathcal{L}u = \sum_{i=1}^{d} \alpha_i \frac{\partial^2 u}{\partial x_i^2} + \sum_{i=1}^{d} \beta_i \frac{\partial u}{\partial x_i}, \quad \tilde{\mathcal{L}}u = \sum_{i=1}^{d} \gamma_i \frac{\partial^2 u}{\partial x_i^2} + \tilde{\gamma} u, \tag{2.3}$$

where the coefficients are given functions or constants.

Depending on the choice of coefficients, the equation can represent different types of IDEs. In particular, when $a = 1$, the equation is classified as a hyperbolic integro-differential equation (HIDE), whereas when $a = 0$ and $b = 1$, it corresponds to a parabolic IDE (PIDE). In this work, we focus on Volterra-type IDEs involving both temporal and spatial differential operators.

2.1. Volterra integro-differential equations

We consider Dirichlet boundary and initial conditions:

$$u(\boldsymbol{x}, t) = g(\boldsymbol{x}, t) \quad \text{for } \boldsymbol{x} \in \partial\Omega,\ t \in (0, T], \tag{2.4}$$

$$u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Omega, \tag{2.5}$$

and, in the hyperbolic case, an additional initial condition

$$\dot{u}(\boldsymbol{x}, 0) = w_0(\boldsymbol{x}).$$

The main computational difficulty in solving Volterra IDEs arises from the convolution term, which requires evaluation of the entire solution history. Classical numerical approaches rely on quadrature rules and time-stepping schemes, leading to increased computational cost and memory requirements.

In this work, we restrict attention to kernels that are bounded and integrable in time.

Quadrature rules. Typically, numerical integrations are computed by quadrature rules. For instance, Gauss-Legendre quadrature [9] of degree n approximates the integral:

$$K * u(t) \approx \sum_{i=1}^{n} w_i K(t - s_i(t))u(s_i(t)), \tag{2.6}$$

where w_i are quadrature weights and $s_i(t)$ are quadrature points. The positivity of the quadrature rule will lead to a priori analysis of numerical schemes [10–12]. While quadrature rules provide exact integration for polynomials up to a certain degree, they generally introduce quadrature errors when applied to arbitrary functions.

2.2. Linear viscoelasticity

Linear viscoelasticity describes the time-dependent mechanical behavior of materials that exhibit both elastic and viscous responses under deformation. Unlike purely elastic materials, which respond instantaneously to applied stress, viscoelastic materials show a gradual strain evolution influenced by their internal relaxation mechanisms. In particular, we focus on the generalized Maxwell model [13–15], which consists of a series of Maxwell elements, each comprising a spring and a dashpot in series. The stress-strain relation is formulated as a convolution integral with a relaxation modulus $\varphi(t)$ represented as a sum of exponentials:

$$\varphi(t) = \varphi_0 + \sum_{q=1}^{N_\varphi} \varphi_q e^{-t/\tau_q}, \tag{2.7}$$

where $\varphi_0, \ldots, \varphi_{N_\varphi} \geq 0$, $\tau_1, \ldots, \tau_{N_\varphi} > 0$, and $\varphi(0) = 1$. This model effectively captures a wide range of viscoelastic behaviors by tuning the number of Maxwell elements N_φ.

While the primary model problem of linear viscoelasticity is vector-valued, we restrict our focus to its scalar analog, which arises naturally in the context of antiplane shear deformation in viscoelastic materials. Specifically, in three-dimensional (3D) antiplane shear problems, the governing equations simplify, reducing the vector-valued system to a scalar wave equation in two dimensions (2D) (see, e.g., [16, 17]).

The material deformation follows the momentum equation:

$$\rho \ddot{\boldsymbol{u}} - \nabla \cdot \underline{\boldsymbol{\sigma}} = \boldsymbol{f} \qquad \text{on } \Omega \times (0, T], \tag{2.8}$$

where $\rho > 0$ is the density and the stress tensor is given by the constitutive relation:

$$\underline{\boldsymbol{\sigma}}(t) = \underline{\boldsymbol{D}}\varphi(t)\underline{\boldsymbol{\varepsilon}}(0) + \int_0^t \underline{\boldsymbol{D}}\varphi(t-s)\dot{\underline{\boldsymbol{\varepsilon}}}(s)ds = \underline{\boldsymbol{D}}\varphi(0)\underline{\boldsymbol{\varepsilon}}(t) - \int_0^t \underline{\boldsymbol{D}}\dot{\varphi}(t-s)\underline{\boldsymbol{\varepsilon}}(s)ds, \tag{2.9}$$

with $\underline{\boldsymbol{D}}$ defined as

$$D_{ijkl} = 2\mu\delta_{ik}\delta_{jl} + \tilde{\mu}\delta_{ij}\delta_{kl},$$

where μ and $\tilde{\mu}$ are the Lamé parameters. The strain tensor $\boldsymbol{\varepsilon}$ is expressed as

$$\varepsilon_{ij}(\boldsymbol{u}) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \qquad \text{for } i, j = 1, \ldots, d.$$

For antiplane shear deformation, the displacement vector is given by $\boldsymbol{u} = (0, 0, u(x, y, t))$. According to this displacement vector, the strain tensor is given as

$$\boldsymbol{\varepsilon} = \begin{bmatrix} 0 & 0 & \frac{1}{2}\frac{\partial u}{\partial x} \\ 0 & 0 & \frac{1}{2}\frac{\partial u}{\partial y} \\ \frac{1}{2}\frac{\partial u}{\partial x} & \frac{1}{2}\frac{\partial u}{\partial y} & 0 \end{bmatrix},$$

and so the stress tensor (2.9) can be written as

$$\boldsymbol{\sigma} = \begin{bmatrix} 0 & 0 & \mu(\varphi(0)\frac{\partial u}{\partial x} - \dot{\varphi} * \frac{\partial u}{\partial x}(t)) \\ 0 & 0 & \mu(\varphi(0)\frac{\partial u}{\partial y} - \dot{\varphi} * \frac{\partial u}{\partial y}(t)) \\ \mu(\varphi(0)\frac{\partial u}{\partial x} - \dot{\varphi} * \frac{\partial u}{\partial x}(t)) & \mu(\varphi(0)\frac{\partial u}{\partial y} - \dot{\varphi} * \frac{\partial u}{\partial y}(t)) & 0 \end{bmatrix}.$$

By substitution of this simplification into (2.8), we reduce the governing equations to a scalar integro-differential model that characterizes the shear wave propagation in viscoelastic materials. The resulting shear wave equation takes the form

$$\rho\ddot{u} - \nabla \cdot \mu\nabla\left(u - \sum_{q=1}^{N_\varphi} K_q * u\right) = f, \tag{2.10}$$

where the memory kernel is given by $K_q(t) = (\varphi_q/\tau_q)e^{-t/\tau_q}$ for each q. The presence of the convolution term introduces computational challenges, requiring efficient numerical techniques such as quadrature-based discretization and time-stepping schemes to ensure stability and accuracy in simulations. Noting that

$$K_q * u(t) = \int_0^t \frac{\varphi_q}{\tau_q} e^{-(t-s)/\tau_q} u(s)ds = \varphi_q u(t) - \varphi_q e^{-t/\tau_q} u_0 - \int_0^t \varphi_q e^{-(t-s)/\tau_q} \dot{u}(s)ds,$$

by integration by parts in time for convolution, we can rewrite (2.10) as

$$\begin{aligned} \rho\ddot{u} - \nabla \cdot \mu\nabla\left(\varphi_0 u + \sum_{q=1}^{N_\varphi} \int_0^t \varphi_q e^{-(t-s)/\tau_q} \dot{u}(s)ds\right) = \; & \rho\ddot{u} - \nabla \cdot \mu\nabla\left(\varphi_0 u + \sum_{q=1}^{N_\varphi} \tilde{K}_q * \dot{u}\right) \\ = \; & f + \nabla \cdot \mu\nabla \sum_{q=1}^{N_\varphi} \varphi_q e^{-t/\tau_q} u_0, \end{aligned} \tag{2.11}$$

where $v = \dot{u}$ and $\tilde{K}_q(t) = \varphi_q e^{-t/\tau_q}$ for each q. If $\varphi_0 = 0$ and $u_0 = 0$, (2.11) reduces to

$$\rho\dot{v} - \nabla \cdot \mu\nabla \sum_{q=1}^{N_\varphi} \tilde{K}_q * v = f, \tag{2.12}$$

where $v = \dot{u}$.

Internal variables. As proposed in [7, 8, 18], the convolution terms can be replaced by auxiliary internal variables. This avoids quadrature-based computations and introduces additional ordinary differential equations.

By defining $\psi_q = K_q * u$, solving (2.10) is equivalent to the system

$$\begin{cases} \rho\ddot{u} - \nabla\cdot\mu\nabla u + \nabla\cdot\mu\nabla\sum\limits_{q=1}^{N_\varphi}\psi_q = f, \\ \tau_q\dot{\psi}_q + \psi_q = \varphi_q u \qquad \text{for each } q. \end{cases} \tag{2.13}$$

Similarly, (2.11) can be rewritten as

$$\begin{cases} \rho\ddot{u} - \nabla\cdot\mu\varphi_0\nabla u - \nabla\cdot\mu\nabla\sum\limits_{q=1}^{N_\varphi}\zeta_q = f + \nabla\cdot\mu\nabla\sum\limits_{q=1}^{N_\varphi}\varphi_q e^{-t/\tau_q}u_0, \\ \tau_q\dot{\zeta}_q + \zeta_q = \tau_q\varphi_q\dot{u} \qquad \text{for each } q, \end{cases} \tag{2.14}$$

where $\zeta_q = \tilde{K}_q * v$ with $v = \dot{u}$. We refer to (2.13) and (2.14) as the displacement form and velocity form, respectively.

Remark 2.1. *From the convolution definitions of the internal variables, both types of internal variables satisfy the zero initial conditions*

$$\psi_q(\boldsymbol{x}, 0) = 0, \qquad \zeta_q(\boldsymbol{x}, 0) = 0, \qquad \forall q = 1, \ldots, N_\varphi, \qquad \textit{and} \qquad \boldsymbol{x} \in \Omega,$$

since the corresponding memory integrals vanish at $t = 0$.

This reformulation replaces convolution integrals with a finite set of auxiliary ODEs, eliminating the need for storing the full solution history. As a result, both memory usage and computational cost are significantly reduced compared to quadrature-based approaches. This advantage becomes particularly pronounced in long-time simulations, where accumulated storage and quadrature errors can be substantial. The equivalence of the model formulations and their numerical behavior have been previously validated using finite element methods [7, 8, 19], providing a reliable reference for the PINN-based approach considered in this work.

Remark 2.2. *Indeed, employing internal variables is feasible for any exponential kernel. Due to the eigenfunction property, exponential kernels naturally lead to associated ODE systems, making them well-suited for this approach. However, this property does not hold for polynomial-type kernels, which lack a corresponding ODE representation.*

3. Physics-informed neural networks

PINNs provide a mesh-free framework for solving differential equations by embedding governing equations into the training process [1, 2]. In this approach, the solution is approximated by a neural network $\hat{u}(\boldsymbol{x}, t; \boldsymbol{\theta})$, and its derivatives with respect to the input variables are computed using automatic differentiation (AD) [3].

Given the integro-differential equation (2.1), we define the residual operator

$$\mathcal{F}(u) = \mathcal{D}u + \mathcal{L}u + \tilde{\mathcal{L}}(K * u) - f = 0, \qquad \text{for } \boldsymbol{x} \in \Omega,\ t \in (0, T], \tag{3.1}$$

together with boundary and initial conditions

$$\mathcal{B}(u) = 0, \qquad \text{on } \boldsymbol{x} \in \partial\Omega \text{ and/or } t = 0. \tag{3.2}$$

The neural network $\hat{u}$ is trained by minimizing the residual $\mathcal{F}(\hat{u})$ and the boundary operator $\mathcal{B}(\hat{u})$ over a set of collocation points. This formulation allows the IDE to be solved without explicit discretization of the differential operators. However, the presence of the convolution term $K * u$ introduces additional computational challenges, which will be addressed using internal variable reformulations.

3.1. Solutions to IDEs

In the PINN framework, the solution $u(\boldsymbol{x}, t)$ is approximated by a feed-forward neural network $\hat{u}(\boldsymbol{x}, t; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$. The network is constructed through successive affine transformations and nonlinear activation functions.

The derivatives of $\hat{u}$ with respect to space and time are computed using automatic differentiation, allowing direct evaluation of the differential operators $\mathcal{D}$, $\mathcal{L}$, and $\tilde{\mathcal{L}}$. However, the convolution term $K * u$ requires additional numerical treatment, typically involving quadrature-based approximation.

To train the network, we define a set of collocation points

$$\mathcal{X} = \{(\boldsymbol{x}_i, t_i)\}_{i=1}^{N_\mathcal{X}} \subset \bar{\Omega} \times [0, T],$$

which is divided into interior points $\mathcal{X}_{\text{PDE}}$ and boundary points $\mathcal{X}_{\text{BC}}$.

The loss function is defined as

$$\mathrm{L}_{\text{total}}(\mathcal{X}; \boldsymbol{\theta}) = \varpi_{\text{PDE}} \mathrm{L}_{\text{PDE}}(\mathcal{X}_{\text{PDE}}; \boldsymbol{\theta}) + \varpi_{\text{BC}} \mathrm{L}_{\text{BC}}(\mathcal{X}_{\text{BC}}; \boldsymbol{\theta}), \tag{3.3}$$

where

$$\mathrm{L}_{\text{PDE}} = \frac{1}{|\mathcal{X}_{\text{PDE}}|} \sum \|\mathcal{F}(\hat{u})\|^2, \qquad \mathrm{L}_{\text{BC}} = \frac{1}{|\mathcal{X}_{\text{BC}}|} \sum \|\mathcal{B}(\hat{u})\|^2.$$

The network parameters $\boldsymbol{\theta}$ are obtained by minimizing the loss function using gradient-based optimization methods, such as Adam and L-BFGS. Also, we employ smooth activation functions, such as `tanh` and `swish`.

The main computational bottleneck in this framework arises from the evaluation of the convolution term $K * u$, which requires storing and processing the solution history. This issue is addressed using an internal variable reformulation as follows.

PINNs for linear viscoelasticity. To solve IDEs within the PINN framework, the convolution term $K * \hat{u}$ is typically evaluated via quadrature-based approximations such as (2.6). However, for linear viscoelastic problems governed by (2.10) and (2.11), the convolution terms can be replaced by auxiliary internal variables, such as $\hat{\psi}_q$ and $\hat{\zeta}_q$.

Specifically, we introduce additional ODE constraints and define the PINN residual vector $\mathcal{F}$ corresponding to the coupled PDE-ODE system. For the displacement formulation (2.13), we define

$$\mathcal{F}(\hat{u}, \hat{\psi}_1, \ldots, \hat{\psi}_{N_\varphi}; \boldsymbol{x}_i, t_i; \boldsymbol{\theta}) = \begin{bmatrix} \mathcal{D}\hat{u} + \mathcal{L}\hat{u} + \sum_{q=1}^{N_\varphi} \tilde{\mathcal{L}}\hat{\psi}_q - f \\ \tau_1 \partial_t \hat{\psi}_1 + \hat{\psi}_1 - \varphi_1 \hat{u} \\ \vdots \\ \tau_{N_\varphi} \partial_t \hat{\psi}_{N_\varphi} + \hat{\psi}_{N_\varphi} - \varphi_{N_\varphi} \hat{u} \end{bmatrix}, \tag{3.4}$$

where the internal variables satisfy zero initial conditions, which may be incorporated into the boundary operator $\mathcal{B}$. The internal variable equations may become stiff when some relaxation times are very small or when the relaxation times are widely separated. Although the equation is written in the scaled form

$$\tau_q \partial_t \psi_q + \psi_q - \varphi_q u = 0,$$

the equivalent evolution form contains coefficients of order $1/\tau_q$, so small τ_q values generate fast relaxation modes on the short time scale $O(\tau_q)$. The scaled residual form used in the PINN loss avoids explicit $1/\tau_q$ coefficients and helps reduce stiffness-related loss imbalance.

Similarly, for the velocity formulation (2.14), we define

$$\mathcal{F}(\hat{u}, \hat{\zeta}_1, \ldots, \hat{\zeta}_{N_\varphi}; \boldsymbol{x}_i, t_i; \boldsymbol{\theta}) = \begin{bmatrix} \mathcal{D}\hat{u} + \mathcal{L}\hat{u} - \sum_{q=1}^{N_\varphi} \tilde{\mathcal{L}}\hat{\zeta}_q - f - \tilde{\mathcal{L}} \sum_{q=1}^{N_\varphi} \varphi_q e^{-t/\tau_q} u_0 \\ \tau_1 \partial_t \hat{\zeta}_1 + \hat{\zeta}_1 - \tau_1 \varphi_1 \partial_t \hat{u} \\ \vdots \\ \tau_{N_\varphi} \partial_t \hat{\zeta}_{N_\varphi} + \hat{\zeta}_{N_\varphi} - \tau_{N_\varphi} \varphi_{N_\varphi} \partial_t \hat{u} \end{bmatrix}, \tag{3.5}$$

which corresponds to the internal variable formulation of the velocity-based model.

The definition of the residual $\mathcal{F}$ depends on the differential operators $\mathcal{D}$, $\mathcal{L}$, and $\tilde{\mathcal{L}}$, leading to corresponding variations in the PDE–ODE residuals and loss functions. For each formulation, either displacement-based or velocity-based internal variables are incorporated into a multi-output neural network. The resulting model approximates both the primary solution and auxiliary internal variables while enforcing the coupled system via automatic differentiation. The overall training procedure is summarized in Algorithm 1.

Algorithm 1 Training procedure of the proposed PINN with internal variables.

1: Choose formulation: Displacement or velocity form; define $\hat{z}_q$ as either $\hat{\psi}_q$ or $\hat{\zeta}_q$ for $q = 1, \ldots, N_\varphi$
2: Initialize network parameters θ
3: Define multi-output neural network

$$(\hat{u}, \hat{v}, \hat{z}_1, \ldots, \hat{z}_{N_\phi})(x, t; \theta)$$

4: Generate collocation points $\mathcal{X}_{\text{PDE}}$ and $\mathcal{X}_{\text{BC}}$
5: **while** not converged **do**
6: Evaluate network outputs
7: Define $\mathcal{F}$ and $\mathcal{B}$ using automatic differentiation
8: Compute PDE-ODE losses L_{PDE}
9: Compute boundary and initial losses L_{BC}
10: Form total loss L_{total}
11: Update the parameters using the Adam optimizer
12: **end while**
13: Refine the parameters using the L-BFGS optimizer
14: **return** trained model

Error analysis of PINNs. Let u be the exact solution of (2.1) and $\hat{u}$ its PINN approximation. The total error can be decomposed as

$$\|u - \hat{u}\| \leq \underbrace{\|u - u_\theta^*\|}_{\text{approximation error}} + \underbrace{\|u_\theta^* - \hat{u}_{\theta^*}\|}_{\text{optimization error}} + \underbrace{\|\hat{u}_{\theta^*} - \hat{u}\|}_{\text{generalization error}},$$

where u_θ^* denotes the best approximation in the chosen hypothesis class and $\hat{u}_{\theta^*}$ is the minimizer of the empirical loss.

From the approximation viewpoint, classical results show that neural networks with non-polynomial activation functions can approximate sufficiently smooth functions and their derivatives with arbitrary accuracy, provided that the network is sufficiently large [20]. For PINNs, recent studies have developed theoretical analyses of generalization and optimization errors for several classes of PDEs, including elliptic, parabolic, and hyperbolic problems [22–24]. In particular, under suitable assumptions on the differential operators and sampling strategy, generalization error bounds of order $O(|\mathcal{X}_{\text{PDE}}|^{-1})$ have been obtained in several settings [21, 22, 24].

For PINNs applied to IDEs, an additional source of error arises from the treatment of convolution terms. In quadrature-based formulations, the convolution operator is approximated numerically, which introduces an additional consistency error in the residual.

For exponentially decaying kernels, the internal variable formulation replaces the convolution operator by a finite-dimensional system of ODEs. Consequently, no additional consistency error associated with numerical quadrature is introduced at the continuous level.

Therefore, the dominant error contributions reduce to the approximation, optimization, and generalization errors of the neural network, together with the residual error arising from the enforcement of the coupled PDE-ODE system.

A complete rigorous error analysis for PINNs applied to IDEs remains open. In particular, the interaction between neural network approximation, sampling, optimization, and the auxiliary ODE constraints has not yet been fully understood. For this reason, the present work focuses on the formulation and numerical validation of the proposed method rather than on deriving a full *a priori* error estimate.

3.2. Solutions to inverse problems

Inverse problems aim to identify unknown model parameters from partial observations of the system response. In the PINN framework, this is achieved by treating the model parameters as additional trainable variables and incorporating observational data into the loss function.

In this study, we consider the identification of parameters in linear viscoelastic models, including μ, $\{\varphi_q\}$, and $\{\tau_q\}$, collectively denoted by $\boldsymbol{\lambda}$. The neural network is defined as $\hat{u}(\boldsymbol{x}, t; \boldsymbol{\theta}, \boldsymbol{\lambda})$, where both the network parameters $\boldsymbol{\theta}$ and the model parameters $\boldsymbol{\lambda}$ are treated as trainable variables.

Let $\mathcal{X}_{\text{O}}$ denote a set of observation points with corresponding measurements $\{\bar{u}_i\}$. The total loss function is defined as

$$\text{L}_{\text{total}}(\mathcal{X}; \boldsymbol{\theta}, \boldsymbol{\lambda}) = \varpi_{\text{PDE}}\text{L}_{\text{PDE}}(\mathcal{X}_{\text{PDE}}; \boldsymbol{\theta}, \boldsymbol{\lambda}) + \varpi_{\text{BC}}\text{L}_{\text{BC}}(\mathcal{X}_{\text{BC}}; \boldsymbol{\theta}, \boldsymbol{\lambda}) + \varpi_{\text{O}}\text{L}_{\text{O}}(\mathcal{X}_{\text{O}}; \boldsymbol{\theta}, \boldsymbol{\lambda}), \tag{3.6}$$

where

$$\text{L}_{\text{O}}(\mathcal{X}_{\text{O}}; \boldsymbol{\theta}, \boldsymbol{\lambda}) = \frac{1}{|\mathcal{X}_{\text{O}}|} \sum_{(\boldsymbol{x}_i, t_i) \in \mathcal{X}_{\text{O}}} \|\hat{u}(\boldsymbol{x}_i, t_i) - \bar{u}_i\|^2.$$

The optimization is carried out jointly over the network parameters $\boldsymbol{\theta}$ and the model parameters $\boldsymbol{\lambda}$. This formulation enables simultaneous solution of the forward problem and parameter identification within a unified framework. We refer to [1, 25] for general methodologies and theoretical results on inverse problems using PINNs, including generalization error estimates.

4. Numerical experiments

In this section, we present numerical experiments to assess the accuracy and efficiency of the proposed PINN framework for both forward and inverse IDE problems. All implementations are carried out using DeepXDE [26] with a TensorFlow backend [27].

General parameter settings. For all simulations, we employ `swish` or `tanh` activation functions and use a weighted loss with coefficients 1 for PDE residuals and 0.1 for boundary conditions. Network parameters are initialized with the Glorot uniform initializer; the model is first trained with the Adam optimizer and then refined by L-BFGS. Training points are generated using the Hammersley sequence to ensure good coverage of the computational domain. All computations are performed in FP32 precision, leading to round-off errors on the order of $O(10^{-7})$.

To evaluate performance, we compute relative L_2 errors between predicted and reference solutions, and for inverse problems, we additionally measure errors in the identified parameters. The numerical experiments are designed to demonstrate the effectiveness and scalability of the proposed approach across different settings:

- **Section 4.1–Simple forward IDE problem:** Comparison between quadrature-based and internal variable formulations
- **Section 4.2–HIDE and PIDE problems:** Validation of accuracy for viscoelastic models with known parameters
- **Section 4.3–2D linear viscoelasticity:** Extension to higher-dimensional problems
- **Section 4.4–Inverse problem:** Parameter identification from sparse observations

These experiments demonstrate that the proposed framework provides accurate and computationally efficient solutions for IDEs and is effective for parameter identification in viscoelastic systems.

The loss weights, network architecture, and scaling parameters are empirical choices that affect training stability, approximation capacity, and computational cost. Their appropriate values depend on the nondimensionalization of the problem, the scale of the governing equations, and the available training budget. Detailed configurations of the network architectures and training settings used in the numerical experiments are summarized in Tables 1 and 2.

Table 1. Network architectures and training parameters for all numerical experiments.

Test case	Formulation	Depth	Width	Output	Activation	Adam learning rate
Section 4.1	Quadrature	4	20	1	swish	10^{-3}
	Internal variable	5	40	2	swish	10^{-3}
Section 4.2	Displacement	5	120	3	tanh	10^{-4}
	Velocity	5	120	4	tanh	10^{-4}
Section 4.3	PDMS	6	120	4	tanh	10^{-4}
	Energy dissipation	7	150	4	tanh	5×10^{-5}
Section 4.4	1D inverse	4	60	4	swish	10^{-4}
	2D inverse	6	80	3	tanh	multi-stage

Table 2. Summary of training settings for all numerical experiments.

Test case	Adam iterations	Collocation points	BC ratio
Section 4.1	3–4×10^4	10^2–4×10^3	5%
Section 4.2	8×10^4	3000	15–20%
Section 4.3 (PDMS)	8×10^4	3000	20%
Section 4.3 (Energy dissipation)	10^5	5000	20%
Section 4.4 1D	multi-stage ($2 \times 10^3 \sim 1 \times 10^5$)	1000	15%
Section 4.4 2D	multi-stage ($1 \times 10^4 \sim 2 \times 10^5$)	2000	15%

4.1. Space-time IDE problem

Consider the following IDE:

$$\begin{aligned} \dot{u}(x,t) - \Delta u(x,t) + \int_0^t e^{s-t} u(x,s) ds = f(x,t), \qquad (x,t) \in (0,1) \times (0,5], \\ u(x,t) = 0, \qquad x = 0, 1, \\ u(x,0) = u_0, \qquad x \in [0,1]. \end{aligned}$$

We take the exact solution

$$u(x,t) = e^{-t} \sin(\pi x),$$

from which f and u_0 are obtained analytically.

Within the PINN framework, the IDE is treated in two ways. The first approach uses a quadrature-based formulation,

$$\mathcal{F}(\hat{u}; \mathcal{X}) = \partial_t \hat{u} - \Delta \hat{u} + \mathcal{K}\hat{u} - f, \tag{4.1}$$

where $\mathcal{K}$ denotes the matrix representation of the convolution operator. The second approach introduces an auxiliary variable ψ to replace the convolution term:

$$\mathcal{F}(\hat{u}, \hat{\psi}; \mathcal{X}) = \begin{bmatrix} \partial_t \hat{u} - \Delta \hat{u} + \hat{\psi} - f \\ \partial_t \hat{\psi} + \hat{\psi} - \hat{u} \end{bmatrix}. \tag{4.2}$$

These two formulations lead to different network architectures. The quadrature formulation uses a smaller network with a single output, while the internal variable formulation requires an additional output for ψ. In our implementation, we use $N_{in} = 2$, $N_{depth} = 4$, $N_{width} = 20$, $N_{out} = 1$ for (4.1), and $N_{in} = 2$, $N_{depth} = 5$, $N_{width} = 40$, and $N_{out} = 2$ for (4.2).

Both models are trained until the total loss reaches $O(10^{-7})$. We compare their performance by computing the L_2 relative error over 1024 evaluation points while varying the number of training samples $|\mathcal{X}|$.

As shown in Figure 1, both approaches achieve similar convergence behavior, with an average convergence rate of approximately 1.5. However, the computational cost differs significantly. The quadrature-based formulation incurs substantially higher training time due to the evaluation of the convolution operator. For example, with 4000 training points, it requires approximately 8000 seconds, whereas the internal variable formulation achieves comparable accuracy in about 10% of that time.

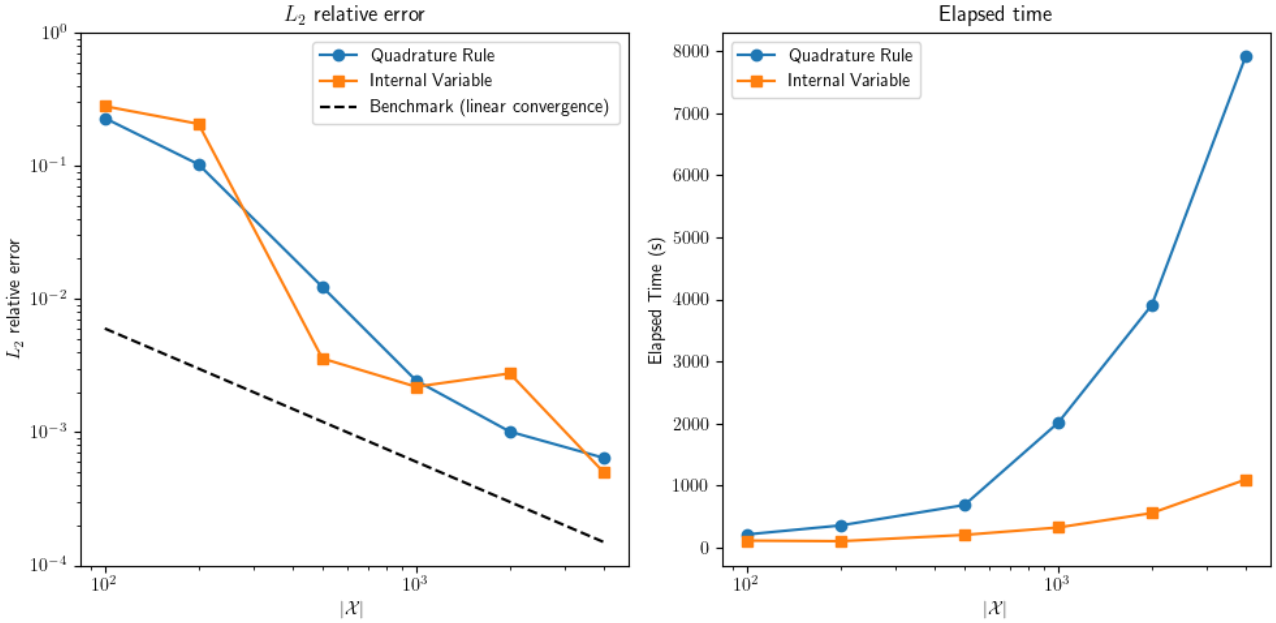


Figure 1. Comparison of PINN models using the quadrature rule and internal variable formulation.

Furthermore, the quadrature-based formulation yields significant computational and memory costs due to the need to store history-dependent terms. As the number of training points increases, this can become prohibitive in practice. For example, in our TensorFlow-based implementation, tensors exceeding 2 GB cannot be processed, which prevents training beyond a certain problem size. In contrast, the internal variable formulation replaces the convolution operator with a finite number of auxiliary variables, resulting in substantially reduced memory usage and computational cost.

4.2. Linear viscoelasticity problem with two types of internal variables

Next, we consider a one-dimensional linear viscoelastic problem on $[0, 1] \times [0, 5]$. For simplicity, we set $\mu = 1$, $\rho = 1$, and use two internal variables with parameters $\varphi_0 = 0$, $\varphi_1 = 0.4$, $\varphi_2 = 0.6$,

$\tau_1 = 0.5$, and $\tau_2 = 2$. The governing equation is given by

$$\ddot{u}(x,t) - \Delta u(x,t) + \Delta\left(K_1 * u(x,t) + K_2 * u(x,t)\right) = f(x,t),$$

where $K_q = (\varphi_q/\tau_q)e^{-t/\tau_q}$ for each q. The exact solution is chosen as

$$u(x,t) = (1 - e^{-t})\sin(\pi x),$$

from which the source term and all auxiliary quantities are determined analytically.

To construct PINN models, we consider two formulations. The first directly approximates the second-order equation using displacement variables:

$$\mathcal{F}(\hat{u}, \hat{\psi}_1, \hat{\psi}_2; \mathcal{X}) = \begin{bmatrix} \partial_{tt}\hat{u} - \Delta\hat{u} + \Delta\hat{\psi}_1 + \Delta\hat{\psi}_2 - f \\ \tau_1\partial_t\hat{\psi}_1 + \hat{\psi}_1 - \varphi_1\hat{u} \\ \tau_2\partial_t\hat{\psi}_2 + \hat{\psi}_2 - \varphi_2\hat{u} \end{bmatrix}, \tag{4.3}$$

while the second introduces the velocity $\hat{v}$ to obtain a first-order system:

$$\mathcal{F}(\hat{u}, \hat{v}, \hat{\psi}_1, \hat{\psi}_2; \mathcal{X}) = \begin{bmatrix} \partial_t\hat{v} - \Delta\hat{u} + \Delta\hat{\psi}_1 + \Delta\hat{\psi}_2 - f \\ \hat{v} - \partial_t\hat{u} \\ \tau_1\partial_t\hat{\psi}_1 + \hat{\psi}_1 - \varphi_1\hat{u} \\ \tau_2\partial_t\hat{\psi}_2 + \hat{\psi}_2 - \varphi_2\hat{u} \end{bmatrix}. \tag{4.4}$$

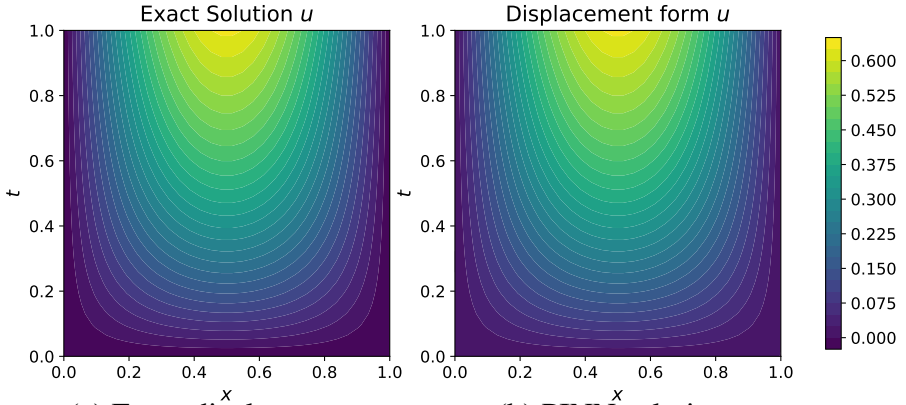
The first-order formulation facilitates the imposition of initial conditions and simplifies implementation within the PINN framework.

Alternatively, using velocity-type internal variables ζ_1 and ζ_2, the system can be written as

$$\mathcal{F}(\hat{v}, \hat{\zeta}_1, \hat{\zeta}_2; \mathcal{X}) = \begin{bmatrix} \partial_t\hat{v} - \Delta\hat{\zeta}_1 - \Delta\hat{\zeta}_2 - f \\ \tau_1\partial_t\hat{\zeta}_1 + \hat{\zeta}_1 - \tau_1\varphi_1\hat{v} \\ \tau_2\partial_t\hat{\zeta}_2 + \hat{\zeta}_2 - \tau_2\varphi_2\hat{v} \end{bmatrix}. \tag{4.5}$$

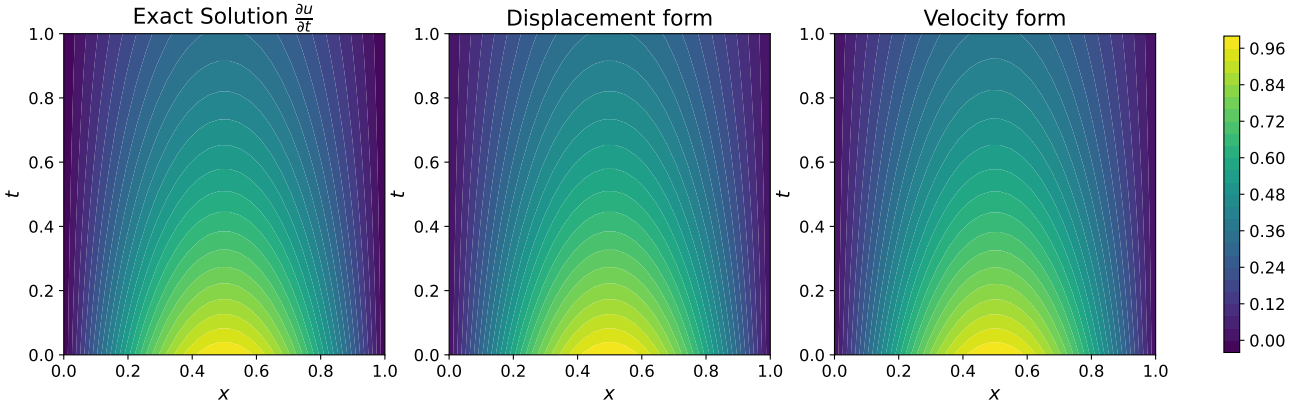
For all formulations, we use the same network structure with $N_{in} = 2$, $N_{depth} = 5$, and $N_{width} = 120$, and set $|\mathcal{X}| = 3000$. The number of outputs is adjusted according to the formulation.

Figures 2 and 3 show that all models accurately reproduce the reference solution. However, their performance differs significantly in terms of accuracy and efficiency.



(a) Exact displacement (b) PINN solution

Figure 2. Comparison of the exact and PINN-predicted displacement fields u using formulation (4.4).



(a) Exact velocity (b) Displacement form (c) Velocity form

Figure 3. Comparison of the exact and PINN-predicted velocity fields $\dot{u}$ using formulations (4.4) and (4.5).

Table 3 shows that the PIDE formulation (4.4) significantly outperforms the HIDE formulation (4.3) in both accuracy and efficiency. While the HIDE formulation yields errors of order $O(10^{-1})$, the PIDE formulation achieves errors of order $O(10^{-3})$ with comparable or lower training cost. This improvement is attributed to the more direct enforcement of initial conditions and the avoidance of additional differentiation through automatic differentiation.

Table 3. PINN model performance for different PDE types and internal variable formulations.

PDE loss type	Internal variable type	Impose v_0	L_2 relative error		Training time(s)
			u	$\dot{u}$	
HIDE (4.3)	Displacement	no	8.57×10^{-2}	1.23×10^{-1}	3481
		yes	1.12×10^{-1}	1.72×10^{-1}	3744
PIDE (4.4)	Displacement	no	4.33×10^{-2}	4.84×10^{-2}	3086
		yes	2.89×10^{-3}	4.53×10^{-3}	3672
PIDE (4.5)	Velocity	yes		2.67×10^{-3}	2244

When only the velocity field is of interest, the velocity-type formulation (4.5) provides a reduced model with fewer outputs. It achieves comparable accuracy for $\dot{u}$ while requiring lower computational cost. These results demonstrate that the PIDE formulation offers a more efficient and reliable approach for PINN-based simulation of viscoelastic systems.

To further investigate the empirical convergence behavior of the proposed formulation, we conduct an additional refinement study with respect to the number of collocation points. In this experiment, the network architecture and optimizer settings are fixed, while the number of interior collocation points $|\mathcal{X}|$ is varied. We compare the HIDE formulation (4.3) with the PIDE formulation (4.4), both of which employ the same internal variable representation.

Figure 4 shows the relative L_2 error as a function of $|\mathcal{X}|$. The PIDE formulation consistently achieves a smaller error than the HIDE formulation over all tested collocation sizes. Moreover, as $|\mathcal{X}|$ increases, the error of the PIDE formulation decreases from approximately 10^{-1} to the order of 10^{-3}, whereas

the HIDE formulation remains at a noticeably larger error level. These results indicate that the PIDE residual provides a more favorable training objective for PINNs, likely because it avoids the direct use of higher-order temporal derivatives and decomposes the governing equation into a coupled system of lower-order residuals.

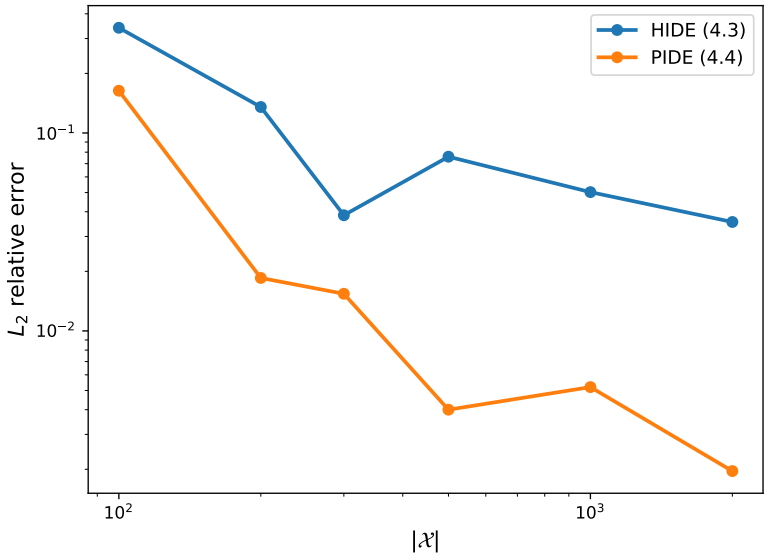


Figure 4. Empirical collocation convergence of u for the HIDE formulation (4.3) and the PIDE formulation (4.4), with the velocity initial condition imposed.

Overall, the results in Table 3 and Figure 4 indicate that the PIDE formulation consistently achieves higher accuracy than the HIDE formulation. The improvement can be attributed primarily to the residual formulation rather than to a different treatment of the hereditary memory term. The first-order coupled PIDE system appears to provide a better-conditioned training objective by avoiding direct higher-order temporal differentiation and by separating the governing equation into lower-order residual components. Therefore, in the remaining experiments, we adopt the PIDE formulation.

4.3. 2D linear viscoelasticity

To validate the model with realistic material parameters, we consider a 2D shear viscoelastic wave equation using material parameters corresponding to polydimethylsiloxane (PDMS) [28]. The parameters are given by

$$\mu = 0.455\,\text{MPa}, \quad \rho = 965\,\text{kg m}^{-3}, \quad \varphi_0 = 0.89,\ \varphi_1 = 0.08,\ \varphi_2 = 0.03,\ \tau_1 = 0.165,\ \tau_2 = 5.$$

The neural network uses $N_{in} = 3$, $N_{depth} = 6$, and $N_{width} = 120$, and is trained with 3000 collocation points, with 15% allocated to boundary conditions.

Validation with analytical solution. We first validate the proposed PINN framework using the exact solution

$$u(x, y, t) = e^{-t} \sin(\pi x) \sin(\pi y), \qquad (x, y, t) \in [0, 1]^2 \times [0, 1].$$

The model is constructed using the PIDE formulation with displacement-type internal variables and both initial conditions imposed.

To ensure stable training, we introduce a scaling factor γ in the residual:

$$\mathcal{F}(\hat{u}, \hat{v}, \hat{\psi}_1, \hat{\psi}_2; \mathcal{X}) = \begin{bmatrix} (\rho\partial_t\hat{v} - \mu\Delta\hat{u} + \mu\Delta\hat{\psi}_1 + \mu\Delta\hat{\psi}_2 - f)/\gamma \\ \hat{v} - \partial_t\hat{u} \\ \tau_1\partial_t\hat{\psi}_1 + \hat{\psi}_1 - \varphi_1\hat{u} \\ \tau_2\partial_t\hat{\psi}_2 + \hat{\psi}_2 - \varphi_2\hat{u} \end{bmatrix}. \tag{4.6}$$

With $\gamma = \mu$, the model achieves relative errors of 1.62×10^{-3} for displacement and 4.98×10^{-3} for velocity. Figures 5 and 6 confirm that the PINN accurately captures both spatial and temporal dynamics.

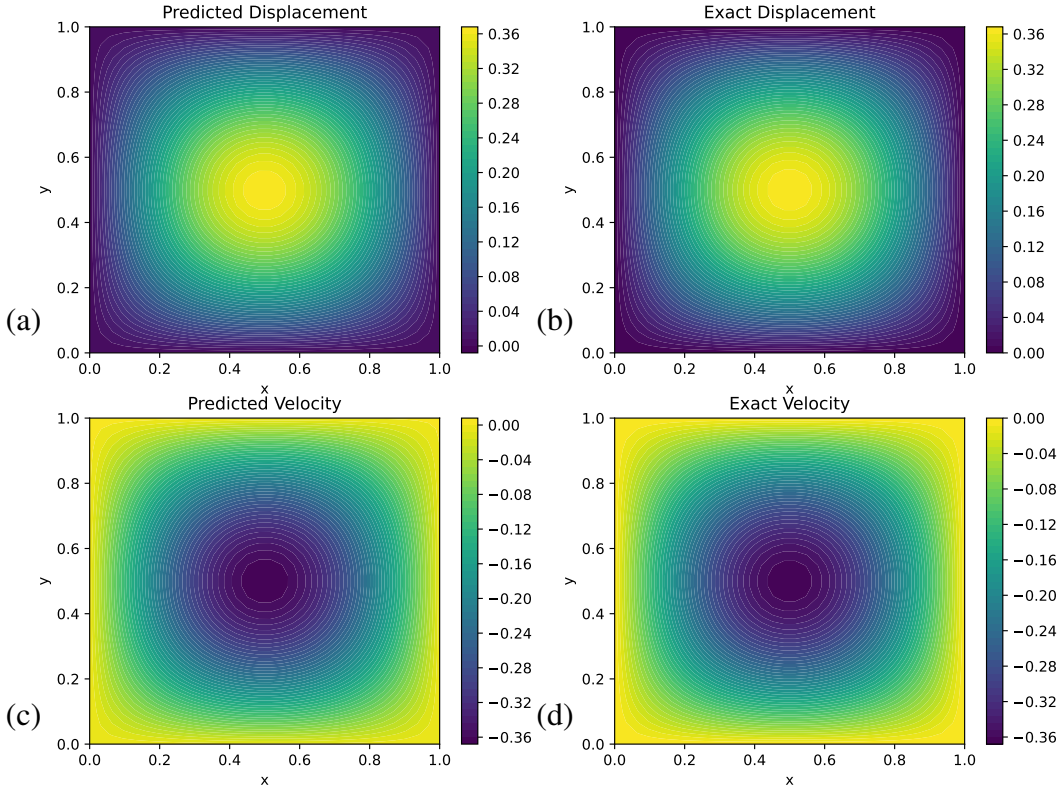
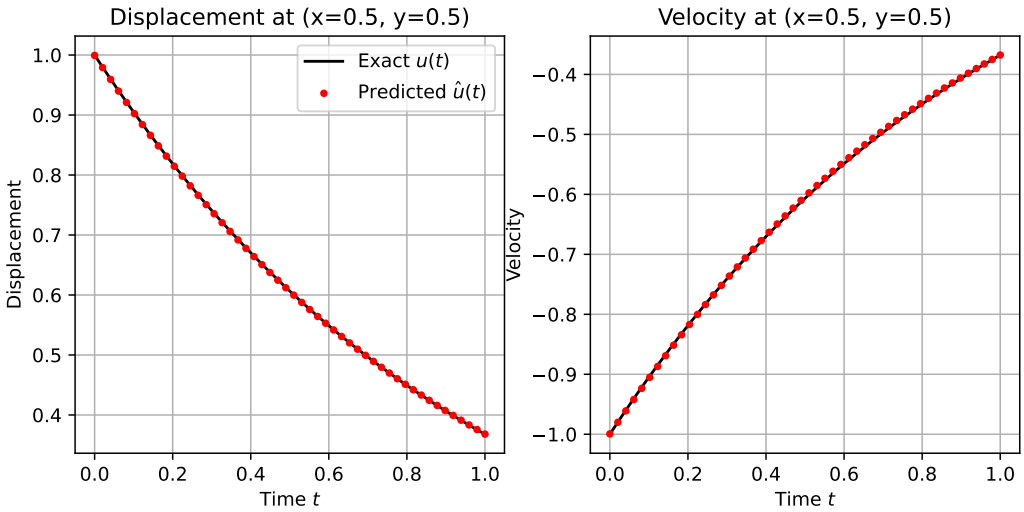


Figure 5. Comparison of the predicted and exact solution fields at $t = T$: (a) predicted displacement, (b) exact displacement, (c) predicted velocity, and (d) exact velocity.



(a) Displacement (b) Velocity

Figure 6. Comparison of the exact and predicted values at the center of the domain.

Energy dissipation and constraint enforcement. We next consider a free-decay problem ($f = 0$) without an analytical solution. To assess the physical consistency of the model, we track the total mechanical energy

$$E(t) = \frac{1}{2}\int_{\Omega} (\rho \dot{u}^2 + \mu |\nabla u|^2)\, d\Omega.$$

For the initial condition $u_0 = \sin(\pi x)\sin(\pi y)/\gamma$ and $v_0 = 0$, the corresponding initial energy is

$$E(0) = \frac{\mu\pi^2}{4\gamma^2}.$$

In elastic systems, $E(t)$ is conserved, whereas in viscoelastic systems it decays over time due to internal damping.

We compare two strategies for imposing initial conditions:

- **Soft constraint:** enforced via loss terms;
- **Hard constraint:** enforced exactly through the network ansatz.

The hard constraint is implemented as

$$\hat{u}(\boldsymbol{x}, t) = u_0(\boldsymbol{x}) + tN_u(\boldsymbol{x}, t), \qquad \hat{v}(\boldsymbol{x}, t) = v_0(\boldsymbol{x}) + tN_v(\boldsymbol{x}, t),$$

which ensures exact satisfaction of initial conditions.

To evaluate the total energy $E(t)$, we compute ∇u using AD and apply the midpoint rule on a uniform 256×256 spatial grid. Table 4 compares soft and hard constraint approaches for initial conditions. The soft constraint leads to non-negligible errors in the initial displacement and significantly underestimates the initial energy. In contrast, the hard constraint imposes the initial condition exactly and recovers the correct energy level with over 99% accuracy.

Table 4. Comparison of soft and hard constraint approaches for imposing initial conditions. Reference total energy is $E(0) = 2.4674$.

Method	L_2 relative error in u_0	Computed $E(0)$	Energy accuracy (%)
Soft constraint	2.31×10^{-2}	1.2050	48.85
Hard constraint	1.31×10^{-7}	2.4868	99.21

Figure 7 shows that both approaches capture energy dissipation, consistent with viscoelastic behavior. However, the hard constraint yields a more accurate initial energy and a more physically consistent evolution over time. These results highlight the importance of accurately enforcing initial conditions in PINN-based simulations.

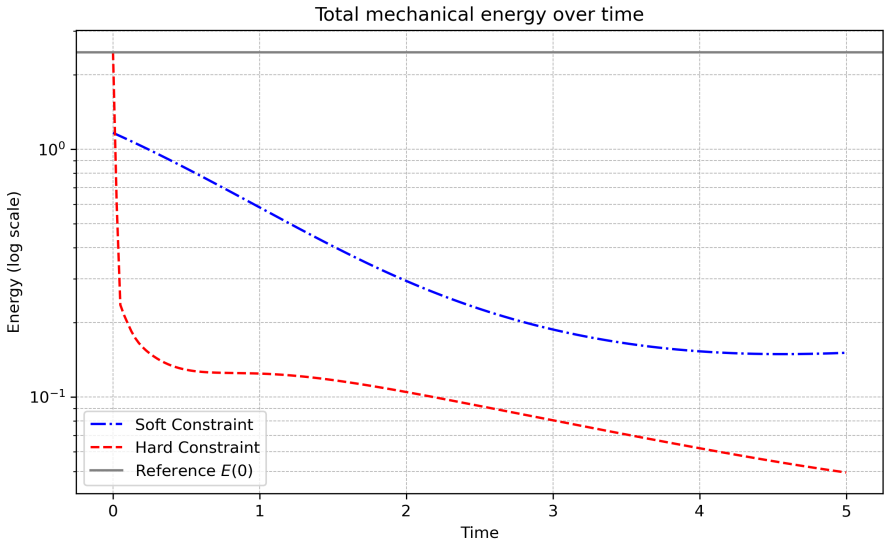


Figure 7. Total mechanical energy over time comparing soft and hard constraint approaches with the reference energy $E(0) = 2.4674$.

4.4. Inverse problem for linear viscoelasticity

We test the capability of PINNs to identify unknown material parameters in linear viscoelasticity from observed data. Specifically, we estimate the shear modulus μ, relaxation coefficient φ, and relaxation time τ from synthetic displacement data generated using known parameters $\mu = 0.5$, $\varphi = 0.42$, and $\tau = 0.88$.

The displacement fields are defined as

$$\begin{aligned} \text{1D}: &\quad u(x,t) = \sin(5\pi x)e^{-t}, \\ \text{2D}: &\quad u(x,y,t) = \sin(2\pi x)\sin(2\pi y)e^{-t}, \end{aligned}$$

with source terms constructed accordingly. The observation set $\mathcal{X}_O$ is built from eight uniform spatial samples (per axis) and 50 time steps: $\mathcal{X}_O = (x_i, t_j)$ in 1D and (x_i, y_j, t_k) in 2D.

For inverse parameter identification, we minimize the total loss defined in (3.6), which includes the PDE residual and observation mismatch. As boundary conditions are imposed via hard constraints, the boundary loss term is omitted. Training begins with a warm-up phase using the Adam optimizer,

where only the observation loss L_O is activated by setting $\varpi_{PDE} = 0$ and $\varpi_O = 0.01$ to stabilize the network around the observation data. In subsequent stages, both components are enabled with weights $\varpi_{PDE} = 1$ and $\varpi_O = 0.01$, and training continues with progressively decreasing learning rates. Finally, the model is fine-tuned using the L-BFGS optimizer. The unknown material parameters are treated as external trainable variables and updated throughout all optimization phases. To enforce admissible parameter ranges, we use the transformations

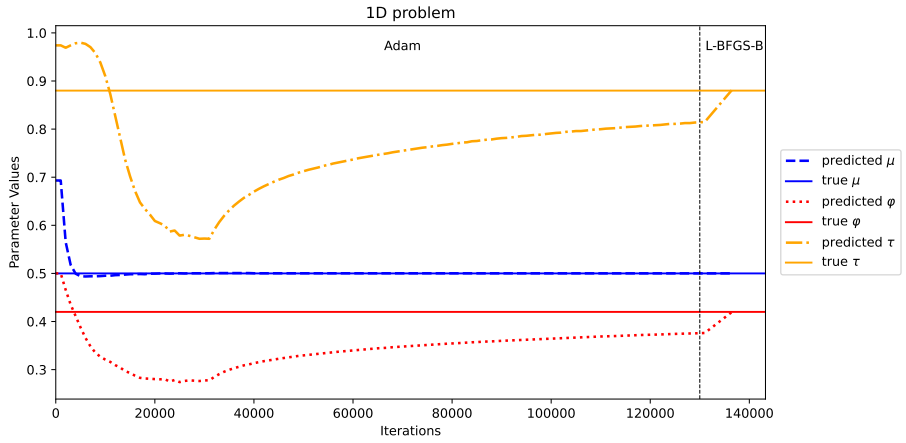
$$\hat{\mu} = \texttt{softplus}(\xi_1), \quad \hat{\tau} = \texttt{softplus}(\xi_2), \quad \hat{\varphi} = \texttt{sigmoid}(\xi_3),$$

where ξ_1, ξ_2, and ξ_3 are the raw trainable variables initialized at $\xi_1 = \xi_2 = 0$ and $\xi_3 = 0.5$.

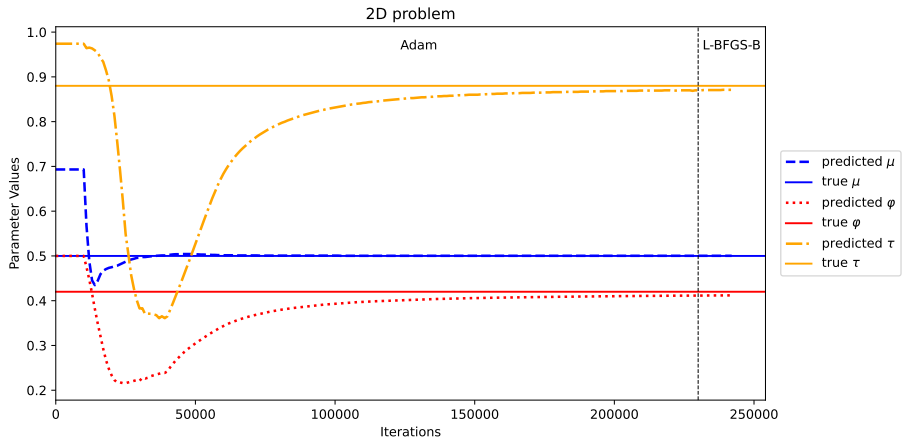
The PINN residual is constructed using the PIDE formulation with displacement-type internal variables:

$$\mathcal{F}(\hat{u}, \hat{v}, \hat{\psi}; \hat{\mu}, \hat{\tau}, \hat{\varphi}; \mathcal{X}) = \begin{bmatrix} (\rho \partial_t \hat{v} - \hat{\mu}\Delta\hat{u} + \hat{\mu}\Delta\hat{\psi} - f)/\gamma \\ \hat{v} - \partial_t \hat{u} \\ \hat{\tau}\partial_t\hat{\psi} + \hat{\psi} - \hat{\varphi}\hat{u} \end{bmatrix}. \tag{4.7}$$

Figure 8 shows that all parameters converge to their true values in both 1D and 2D settings. Table 5 summarizes the final estimates, demonstrating high accuracy.



(a) 1D inverse problem



(b) 2D inverse problem

Figure 8. Convergence of the predicted material parameters to their true values.

Table 5. Identified viscoelastic parameters compared to true values.

Parameter	True	1D	2D
μ	0.5	0.5001	0.5003
τ	0.88	0.8807	0.8710
φ	0.42	0.4194	0.4119

We observe that μ converges faster than τ and φ, reflecting its more direct influence on the PDE residual. In contrast, the relaxation parameters are associated with internal variables and history-dependent effects, which require more iterations to resolve. While L-BFGS accelerates convergence in 1D, optimization in 2D is more challenging due to increased model complexity. Overall, the results demonstrate that the proposed PINN framework reliably identifies viscoelastic parameters across different dimensions.

Since practical measurements inevitably contain noise, we further examine the sensitivity of the identified parameters to noisy observation data. Specifically, Gaussian noise is added to the observations, and the resulting relative errors are summarized in Table 6. The table shows that the elastic parameter μ is accurately identified in both 1D and 2D, even under noisy observations. In the 1D case, the relaxation parameters τ and φ also remain accurate for the tested noise levels. In the 2D case, these relaxation parameters show larger variation, which is consistent with the increased sampling and optimization complexity of higher-dimensional inverse problems. Further improvements may be possible through longer training, improved loss balancing, additional observations, or regularization.

Table 6. Relative errors of the identified viscoelastic parameters for the inverse problems under different Gaussian noise levels.

Parameter	1D			2D		
	0% noise	1% noise	5% noise	0% noise	1% noise	5% noise
μ	1.89×10^{-4}	1.29×10^{-4}	1.07×10^{-5}	6.00×10^{-4}	7.49×10^{-5}	2.62×10^{-5}
τ	7.95×10^{-4}	6.70×10^{-4}	9.41×10^{-3}	1.02×10^{-2}	9.81×10^{-2}	8.75×10^{-2}
φ	1.43×10^{-3}	1.57×10^{-2}	1.34×10^{-2}	1.93×10^{-2}	9.98×10^{-2}	9.35×10^{-2}

5. Conclusions and discussion

We proposed a physics-informed neural network framework for solving forward and inverse problems involving integro-differential equations in linear viscoelasticity. On exponential-type relaxation kernels, the hereditary integral formulation is reformulated into a system of PDEs with internal variables, enabling efficient and scalable implementation within the PINN framework.

The numerical results demonstrate that directly evaluating convolution terms via quadrature leads to substantial computational and memory costs, particularly for high-resolution or high-dimensional problems. In contrast, the internal variable formulation replaces the convolution operator with a finite system of ODEs, significantly improving efficiency and scalability while maintaining accuracy in the reported experiments. Building on this reformulation, expressing the governing equations as a first-order system through state augmentation further enhances accuracy and training stability. The resulting PIDE formulation consistently outperforms the corresponding second-order HIDE formulation in

terms of accuracy, stability, and enforcement of initial conditions. This improvement may be attributed to the avoidance of repeated higher-order differentiation in automatic differentiation, which leads to a more favorable optimization landscape. Furthermore, the use of hard constraints enhances physical consistency, particularly in preserving the expected energy behavior of viscoelastic systems. In the inverse setting, the proposed method accurately recovers viscoelastic material parameters from sparse space-time observations. The elastic parameter converges more rapidly than relaxation parameters, reflecting their different roles in the governing dynamics. These results demonstrate that the proposed approach provides a reliable and effective tool for parameter identification in viscoelastic systems.

Possible directions for future work include extensions to 3D viscoelastic models in elastodynamics, other constitutive laws, and applications to experimental data with measurement noise. The handling of non-exponential memory kernels, such as power-law and Mittag-Leffler kernels arising in fractional-order viscoelastic models, is also of interest. Such kernels may be approximated by finite sums of exponentials, yielding an approximate internal variable system within the same PINN residual framework. Biomechanical applications, such as the identification of viscoelastic wall properties in arterial stenosis models, provide a motivating class of inverse problems for future investigation.

Use of Generative-AI tools declaration

The author declares he has not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was partially supported by Chonnam National University and by Global-Learning & Academic research institution for Master's · PhD students, and Postdocs(LAMP) Program of the National Research Foundation of Korea(NRF) grant funded by the Ministry of Education(No. RS-2024-00442775). This research was also supported by the Regional Innovation System & Education (RISE) Glocal University 30 program through the Gwangju RISE Center, funded by the Ministry of Education (MoE) and Gwangju Metropolitan City, Republic of Korea (2026-RISE(Glocal University 30)-05-011).

Conflict of interest

The author declares that there is no conflict of interest.

References

1. M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *J. Comput. Phys.*, **378** (2019), 686–707. https://doi.org/10.1016/j.jcp.2018.10.045
2. G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, *Nat. Rev. Phys.*, **3** (2021), 422–440. https://doi.org/10.1038/s42254-021-00314-5

3. A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic differentiation in machine learning: a survey, *J. Mach. Learn. Res.*, **18** (2018), 1–43.
4. E. Hetmaniok, M. Pleszczyński, Y. Khan, Solving the integral differential equations with delayed argument by using the differential transform method, *Sensors*, **22** (2022), 4124. https://doi.org/10.3390/s22114124
5. R. Brociek, M. Goik, J. Miarka, M. Pleszczynski, C. Napoli, Solution of inverse problem for diffusion equation with fractional derivatives using metaheuristic optimization algorithm, *Informatica*, **35** (2024), 453–481. https://doi.org/10.15388/24-INFOR563
6. R. Gao, L. Ju, D. Li, Kernel-based PINNs for time fractional differential equations and Volterra integral-differential equations, *Phys. D*, **490** (2026), 135174. https://doi.org/10.1016/j.physd.2026.135174
7. Y. Jang, S. Shaw, Finite element approximation and analysis of a viscoelastic scalar wave equation with internal variable formulations, *J. Comput. Appl. Math.*, **412** (2022), 114340. https://doi.org/10.1016/j.cam.2022.114340
8. Y. Jang, S. Shaw, A priori analysis of a symmetric interior penalty discontinuous Galerkin finite element method for a dynamic linear viscoelasticity model, *Comput. Methods Appl. Math.*, **23** (2023), 647–669. https://doi.org/10.1515/cmam-2022-0201
9. G. H. Golub, J. H. Welsch, Calculation of Gauss quadrature rules, *Math. Comput.*, **23** (1969), 221–230. https://doi.org/10.2307/2004418
10. W. McLean, V. Thomée, Numerical solution of an evolution equation with a positive-type memory term, *ANZIAM J.*, **35** (1993), 23–70. https://doi.org/10.1017/S0334270000007268
11. Y. Jang, S. Shaw, A priori error analysis for a finite element approximation of dynamic viscoelasticity problems involving a fractional order integro-differential constitutive law, *Adv. Comput. Math.*, **47** (2021), 46. https://doi.org/10.1007/s10444-021-09857-8
12. Y. Jang, S. Shaw, Discontinuous Galerkin finite element method for dynamic viscoelasticity models of power-law type, *Numer. Methods Partial Differ. Equ.*, **40** (2024), e23107. https://doi.org/10.1002/num.23107
13. S. Shaw, J. R. Whiteman, Some partial differential Volterra equation problems arising in viscoelasticity, In: *Proceedings of Equadiff 9, Conference on differential equations and their applications*, 1998, 183–200.
14. J. M. Golden, G. A. C. Graham, *Boundary value problems in linear viscoelasticity*, Heidelberg: Springer, 2013. https://doi.org/10.1007/978-3-662-06156-5
15. A. D. Drozdov, *Viscoelastic structures: Mechanics of growth and aging*, Academic Press, 1998.
16. G. H. Paulino, Z. H. Jin, Viscoelastic functionally graded materials subjected to antiplane shear fracture, *J. Appl. Mech.*, **68** (2001), 284–293. https://doi.org/10.1115/1.1354205
17. T. V. Hoarau-Mantel, A. Matei, Analysis of a viscoelastic antiplane contact problem with slip-dependent friction, *Int. J. Appl. Math. Comput. Sci.*, **12** (2002), 51–58.
18. A. R. Johnson, Modeling viscoelastic materials using internal variables, *Shock. Vib. Dig.*, **31** (1999), 91–100. https://doi.org/10.1177/058310249903100201

19. Y. Jang, *Spatially continuous and discontinuous Galerkin finite element approximations for dynamic viscoelastic problems*, PhD. Thesis, Brunel University London, 2020.

20. A. Pinkus, Approximation theory of the MLP model in neural networks, *Acta Numer.*, **8** (1999), 143–195. https://doi.org/10.1017/S0962492900002919

21. Y. Shin, J. Darbon, G. E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs, *Commun. Comput. Phys.*, **28** (2020), 2042–2074. https://doi.org/10.4208/cicp.OA-2020-0193

22. Y. Shin, Z. Zhang, G. E. Karniadakis, Error estimates of residual minimization using neural networks for linear PDEs, *J. Mach. Learn. Model. Comput.*, **4** (2023), 73–101. https://doi.org/10.1615/JMachLearnModelComput.2023050411

23. S. Mishra, R. Molinaro, Estimates on the generalization error of physics-informed neural networks for approximating PDEs, *IMA J. Numer. Anal.*, **43** (2023), 1–43. https://doi.org/10.1093/imanum/drab093

24. Y. Qian, Y. Zhang, Y. Huang, S. Dong, Physics-informed neural networks for approximating dynamic (hyperbolic) PDEs of second order in time: Error analysis and algorithms, *J. Comput. Phys.*, **495** (2023), 112527. https://doi.org/10.1016/j.jcp.2023.112527

25. S. Mishra, R. Molinaro, Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs, *IMA J. Numer. Anal.*, **42** (2022), 981–1022. https://doi.org/10.1093/imanum/drab032

26. L. Lu, X. Meng, Z. Mao, G. E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, *SIAM Rev.*, **63** (2021), 208–228. https://doi.org/10.1137/19M1274067

27. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, et al., TensorFlow: Large-scale machine learning on heterogeneous systems, *arXiv:1603.04467*, 2016. https://doi.org/10.48550/arXiv.1603.04467

28. I. K. Lin, K. S. Ou, Y. M. Liao, Y. Liu, K. S. Chen, X. Zhang, Viscoelastic characterization and modeling of polymer transducers for biological applications, *J. Microelectromech. Syst.*, **18** (2009), 1087–1099. https://doi.org/10.1109/JMEMS.2009.2029166

AIMS Press