

International Journal of Statistics and Applied Mathematics

ISSN: 2456-1452
NAAS Rating (2026): 4.49
Maths 2026; 11(6): 243-259
© 2026 Stats & Maths
<https://www.mathsjournal.com>
Received: 15-04-2026
Accepted: 18-05-2026

Bilal Ahmed Shihab
Assistant Lecturer, Department
of Mathematics, Holy Karbala
Directorate General of
Education, Karbala, Iraq

Adaptive L1-physics-informed neural network for nonlinear time-fractional diffusion-reaction equations: Numerical accuracy and stability analysis

Bilal Ahmed Shihab

DOI: <https://www.doi.org/10.22271/math.2026.v11.i6c.2591>

Abstract

Here, an Improved Adaptive L1-Physics-Informed Neural Network framework for solving nonlinear time-fractional diffusion-reaction equations is proposed. The proposed approach merges the L1 approximation of the Caputo fractional derivative with the physics-informed training architecture for neural networks. The motivation behind this is to overcome the challenge of nonlocal temporal behavior of fractional derivatives in which the solution at present time depends on a system's previous history. Although the classical L1 scheme is reliable as a deterministic approximation and the proposed approach is deterministic and stable, the standard Physics-Informed Neural Networks can suffer from the training instability, i.e., if the fractional memory effect does not have a correct representation in the residual function.

Our proposed framework embeds the L1 approximation directly into the physics-informed residual and adds adaptive weights during training to balance the equation residual, initial state, and boundary state. A synthetic exact solution is employed to check the numerical results of the methods. Numerical experiments are done for four fractional orders: $\alpha=0.3, 0.5, 0.7$, and 0.9 , fixed model parameters $\beta=0.01$ and $\gamma=1$ on the computational domain $0 \leq x \leq 1, 0 \leq t \leq 1$.

Performance of the classical L1 scheme with the standard PINN and the Improved Adaptive L1-PINN will be examined by MSE, RMSE, MAE, R2, CPU time and in the training behavior. It is shown that classical L1 scheme achieves the highest deterministic accuracy and lowest computational cost for the manufactured test problem. Nonetheless the Improved Adaptive L1-PINN improves substantially the conventional PINN in majority cases particularly for $\alpha=0.3, \alpha=0.5$ and $\alpha=0.7$. Moreover, the hybrid approach for learning-based approximation is also more stable as it is informed by fractional memory information in neural residual. This support that the Improved Adaptive L1-PINN represents a neural numerical framework for nonlinear time-fractional models, especially when a flexible approximation structure is needed.

Keywords: Fractional differential equations, Caputo derivative, L1 scheme, Physics-Informed Neural Networks, diffusion-reaction equations, adaptive weighting, numerical analysis

1. Introduction

Fractional differential equations have become a valuable tool in the mathematical modeling of memory effects and nonlocal temporal phenomena of systems. These are systems present in instances of anomalous diffusion, heat transfer, biological interactions, financial modeling and material dynamics. Classical integer-order derivatives are typically inappropriate to represent such phenomena as the current state of a system relies on the time in past. Consequently, fractional derivatives, especially the Caputo derivative, offer a better framework (which allows a more valid approximation for building realistic mathematical models). For fractional differential equations, numerical methods would typically need special techniques for time-discretization.

In contrast, Physics-Informed Neural Networks PINNs offer an advanced computational approach for solving differential equations. PINNs mainly embed the governing equation in the loss function when its input, thus training this approach relies on physical and mathematical laws of the model, which relies on the training on top of the existing data. Raissi

Corresponding Author:
Bilal Ahmed Shihab
Assistant Lecturer, Department
of Mathematics, Holy Karbala
Directorate General of
Education, Karbala, Iraq

et al. ^[1] described basic formulation of PINNs for the forward-and-inverse problems based on nonlinear partial differential equations. Subsequent work showed their suitability for heat transfer applications ^[2], and reviewed the evolution of them and computational developments ^[3], as well as their methods, applications, and difficulties ^[4]. Most recent reviews have established PINNs have now been emerging as a key methodology for scientific computing and multidisciplinary modeling [5,6], as well as for using complex materials, such as polymers ^[7].

Despite their flexibility, using PINNs for fractional differential equations, like any other problem, must be approached with care. The main difficulty lies in the nonlocal nature of the fractional derivative: the residual of a fraction depends on the entire temporal history of the solution. This makes creating a differential residual more challenging than in the classical integer-order case. Furthermore, the loss function in PINNs sometimes includes multiple components like equation residual, the initial condition and boundary conditions. In case of the balance of these components is not observed, the training process may fail and the numerical approximation of the training may not be accurate.

The L1 scheme is one of these techniques, as a simple and practical means of approximating the Caputo time-fractional derivative, it is among the most commonly used methods within which to approximate the derivative. Stynes ^[8] conducted a comprehensive survey of the L1 scheme and related issues for discretizing time-fractional problems. Recently, the L1 scheme was extended and improved for other classes of fractional models, such as nonlinear time-fractional Schrödinger delay equations ^[9], memory-term-based subdiffusion equations ^[10], convection-diffusion-reaction equations ^[11], and semilinear stochastic subdiffusion problems ^[12]. These works demonstrate the ongoing and stable utility of the L1 scheme in contemporary fractional numerical computation.

The central problem tackled in this study is the difficulty of accurately and efficiently solving nonlinear time-fractional diffusion-reaction equations. This challenge surfaces in two main forms. The first of these is numerical because the fractional derivative encodes a cumulative memory effect which increases both the computational cost, as well as sensitivity of approximation to time step selection. The second element focuses on the training of neural networks as standard PINNs may fail to represent the nonlocal structure of the fractional derivative when the loss function is not designed in a balanced and stable way. Hence, the question sought by this research is: How can the stability of the L1 scheme be combined with the flexibility of Physics-Informed Neural Networks to produce an accurate and stable method for nonlinear time-fractional diffusion-reaction equations?

This study therefore sets out a hybrid L1-PINN framework to solve nonlinear time-fractional diffusion-reaction equations. The paper combines the L1 approximation of the Caputo derivative with a physics-informed training strategy of neural networks and adds adaptive residual weights to balance the equation residual, initial condition, and boundary conditions during the training. The study formulates the mathematical model, constructs the fractional numerical approximation, designs the neural network loss function, and compares the proposed method to both the classical L1 scheme and the standard PINN method. It also studies how the fractional order α impacts the accuracy and stability of the numerical solution.

2. Mathematical Model

This study considers the following nonlinear time-fractional diffusion-reaction equation:

$$D_{C,t}^{\alpha} u(x, t) = \kappa \frac{\partial^2 u(x, t)}{\partial x^2} + \lambda u(x, t)(1 - u(x, t)) + f(x, t), \quad 0 < x < 1, \quad 0 < t \leq T,$$

Where $D_{C,t}^{\alpha}$ denotes the Caputo time-fractional derivative of order α , $\kappa > 0$ is the diffusion coefficient, λ is the nonlinear reaction coefficient, and $f(x, t)$ is the source term. The problem is completed by the initial and boundary conditions

$$u(x, 0) = u_0(x), \quad 0 \leq x \leq 1,$$

$$u(0, t) = g_1(t), \quad u(1, t) = g_2(t), \quad 0 < t \leq T.$$

For $0 < \alpha < 1$, the Caputo derivative is defined by

$$D_{C,t}^{\alpha} u(x, t) = \frac{1}{\Gamma(1-\alpha)} \int_0^t \frac{\partial u(x, s)}{\partial s} (t-s)^{-\alpha} ds. \quad (1)$$

This formulation allows the model to represent memory-dependent diffusion-reaction dynamics, since the fractional derivative depends on the entire temporal history of the solution.

2.1 Exact Solution for Numerical Verification

To verify the accuracy of the proposed method, the following manufactured exact solution is considered:

$$u(x, t) = t^2 \sin(\pi x). \quad (2)$$

Accordingly, the initial and boundary conditions become

$$u(x, 0) = 0,$$

$$u(0, t) = 0, \quad u(1, t) = 0.$$

Using the Caputo derivative property

$$D_{C,t}^{\alpha} t^2 = \frac{\Gamma(3)}{\Gamma(3-\alpha)} t^{2-\alpha}, \quad (3)$$

We obtain

$$D_{C,t}^{\alpha} u(x, t) = \frac{\Gamma(3)}{\Gamma(3-\alpha)} t^{2-\alpha} \sin(\pi x). \quad (4)$$

Moreover, the second spatial derivative of the exact solution is

$$\frac{\partial^2 u(x, t)}{\partial x^2} = -\pi^2 t^2 \sin(\pi x).$$

Therefore, the source term is determined from

$$f(x, t) = D_{C,t}^{\alpha} u(x, t) - \kappa u_{xx}(x, t) - \lambda u(x, t)(1 - u(x, t)). \quad (5)$$

Substituting the exact solution into the above expression gives

$$f(x, t) = f_1(x, t) + f_2(x, t) - f_3(x, t),$$

Where

$$f_1(x, t) = \frac{\Gamma(3)}{\Gamma(3-\alpha)} t^{2-\alpha} \sin(\pi x),$$

$$f_2(x, t) = \kappa \pi^2 t^2 \sin(\pi x),$$

And

$$f_3(x, t) = \lambda t^2 \sin(\pi x)(1 - t^2 \sin(\pi x)). \quad (6)$$

This source term ensures that the chosen function $u(x, t) = t^2 \sin(\pi x)$ satisfies the governing equation exactly.

2.2 Numerical Approximation Using the L1 Scheme

The time interval $[0, T]$ is divided into N uniform subintervals such that

$$t_n = n\tau, \quad \tau = \frac{T}{N}, \quad n = 0, 1, 2, \dots, N.$$

The Caputo derivative at $t = t_n$ is approximated using the classical L1 scheme as follows:

$$D_{C,t}^{\alpha} u(x, t_n) \approx \frac{\tau^{-\alpha}}{\Gamma(2-\alpha)} \sum_{j=0}^{n-1} a_j [u(x, t_{n-j}) - u(x, t_{n-j-1})],$$

Where the convolution coefficients are given by

$$a_j = (j+1)^{1-\alpha} - j^{1-\alpha}, \quad j = 0, 1, 2, \dots, n-1.$$

The second-order spatial derivative can be approximated by the central difference formula

$$u_{xx}(x_i, t_n) \approx \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2},$$

Where

$$x_i = ih, \quad h = \frac{1}{M}, \quad i = 0, 1, 2, \dots, M.$$

This discretization provides a classical numerical reference for comparison with the proposed hybrid neural framework.

2.3 Standard PINN Formulation

Let the approximate solution be represented by a neural network of the form

$$\hat{u}(x, t; \theta) = \text{NN}(x, t; \theta),$$

Where θ denotes the trainable parameters of the neural network. The differential residual associated with the governing equation is defined as

$$R(x, t; \theta) = D_{C,t}^\alpha \hat{u}(x, t; \theta) - \kappa \hat{u}_{xx}(x, t; \theta) - \lambda \hat{u}(x, t; \theta)(1 - \hat{u}(x, t; \theta)) - f(x, t). \quad (7)$$

The total loss function is written as

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathcal{PDE}} + \mathcal{L}_{\mathcal{IC}} + \mathcal{L}_{\mathcal{BC}}.$$

The equation residual loss is defined by

$$\mathcal{L}_{\mathcal{PDE}} = \frac{1}{N_f} \sum_{k=1}^{N_f} |R(x_f^k, t_f^k; \theta)|^2.$$

The initial-condition loss is

$$\mathcal{L}_{\mathcal{IC}} = \frac{1}{N_i} \sum_{k=1}^{N_i} |\hat{u}(x_i^k, 0; \theta) - u_0(x_i^k)|^2.$$

The boundary-condition loss is given by

$$\mathcal{L}_{\mathcal{BC}} = \frac{1}{N_b} \sum_{k=1}^{N_b} (|\hat{u}(0, t_b^k; \theta) - g_1(t_b^k)|^2 + |\hat{u}(1, t_b^k; \theta) - g_2(t_b^k)|^2). \quad (8)$$

This formulation embeds the governing equation, the initial condition, and the boundary conditions directly into the neural network training process.

2.4 Proposed L1-PINN Method

The proposed method incorporates the L1 approximation of the Caputo derivative into the physics-informed residual. Therefore, the numerical physical residual is expressed as

$$R_{L1}(x_i, t_n; \theta) = A_{L1}(x_i, t_n; \theta) - \kappa \hat{u}_{xx}(x_i, t_n; \theta) - \lambda \hat{u}(x_i, t_n; \theta)(1 - \hat{u}(x_i, t_n; \theta)) - f(x_i, t_n), \quad (9)$$

Where

$$A_{L1}(x_i, t_n; \theta) = \frac{\tau^{-\alpha}}{\Gamma(2-\alpha)} \sum_{j=0}^{n-1} a_j [\hat{u}(x_i, t_{n-j}; \theta) - \hat{u}(x_i, t_{n-j-1}; \theta)].$$

The adaptive loss function is then defined as

$$\mathcal{L}_A(\theta) = w_f \mathcal{L}_{L1-\mathcal{PDE}} + w_i \mathcal{L}_{\mathcal{IC}} + w_b \mathcal{L}_{\mathcal{BC}},$$

Where w_f , w_i , and w_b are adaptive weights updated during the training process. These weights can be updated according to the softmax-type rule

$$w_m^{(r+1)} = \frac{\exp(\eta \mathcal{L}_m^{(r)})}{\sum_q \exp(\eta \mathcal{L}_q^{(r)}), \quad m \in \{f, i, b\},$$

Where r denotes the iteration number and $\eta > 0$ controls the sensitivity of the adaptive weighting mechanism. This adaptive structure prevents one part of the loss function from dominating the training process and improves the balance between the physical residual, the initial condition, and the boundary conditions.

2.5 Algorithm of the Proposed Method

The implementation steps are given as follows:

- **Step 1:** Select the number of interior training points N_f , the number of initial-condition points N_i , and the number of boundary-condition points N_b .
- **Step 2:** Generate the temporal and spatial grid required to evaluate the L1 approximation of the Caputo fractional derivative.
- **Step 3:** Construct a neural network in which (x, t) represents the input variables and $\hat{u}(x, t)$ represents the network output.

- **Step 4:** Compute the numerical physical residual R_{L1} by substituting the $L1$ approximation into the governing differential equation.
- **Step 5:** Evaluate the loss components $\mathcal{L}_{L1-PDE}$, $\mathcal{L}_{IC}$, and $\mathcal{L}_{BC}$.
- **Step 6:** Update the adaptive weights w_f , w_i , and w_b during training according to the relative contribution of each loss component.
- **Step 7:** Optimize the neural network parameters θ using the Adam optimizer.
- **Step 8:** Refine the optimized parameters θ using the L-BFGS optimizer.
- **Step 9:** Compute the approximate solution $\hat{u}(x, t)$.
- **Step 10:** Compare the approximate solution with the exact solution to assess the accuracy and stability of the proposed method.

The accuracy of the numerical methods is evaluated using the following standard error measures:

$$MSE = \frac{1}{N} \sum_{k=1}^N (u_k - \hat{u}_k)^2,$$

$$RMSE = \sqrt{\frac{1}{N} \sum_{k=1}^N (u_k - \hat{u}_k)^2},$$

$$MAE = \frac{1}{N} \sum_{k=1}^N |u_k - \hat{u}_k|,$$

And

$$R^2 = 1 - \frac{\sum_{k=1}^N (u_k - \hat{u}_k)^2}{\sum_{k=1}^N (u_k - \bar{u})^2}.$$

Here, u_k represents the exact solution value, $\hat{u}_k$ represents the predicted numerical value, $\bar{u}$ is the mean of the exact solution values, and N is the total number of testing points.

3. Numerical Experiments and Simulation Study

This section presents the numerical experiments designed to evaluate the accuracy, stability, and computational efficiency of the proposed Improved Adaptive L1-PINN method for solving nonlinear time-fractional diffusion-reaction equations. The simulation is constructed using a manufactured exact solution, which allows a direct comparison between the exact solution and the numerical approximations obtained by the classical L1 scheme, the standard PINN, and the proposed Improved Adaptive L1-PINN method. The software used for this simulation is MATLAB R2020a, and the application code is attached as Appendix A. The numerical experiments are carried out for four representative values of the fractional order. These values examine the behavior of the numerical methods under weak, moderate, and strong memory effects. The selected fractional orders are

$$\alpha = 0.3, 0.5, 0.7, 0.9.$$

The physical and computational parameters are fixed throughout the experiments as follows:

$$\kappa = 0.01, \quad \lambda = 1, \quad 0 \leq x \leq 1, \quad 0 \leq t \leq 1.$$

The governing equation is given by

$$D_{C,t}^\alpha u(x, t) = \kappa u_{xx}(x, t) + \lambda u(x, t)(1 - u(x, t)) + f(x, t), \quad 0 < x < 1, \quad 0 < t \leq 1.$$

To validate the numerical results, the exact solution is chosen as

$$u(x, t) = t^2 \sin(\pi x).$$

This solution satisfies the homogeneous initial and boundary conditions

$$u(x, 0) = 0, \quad u(0, t) = 0, \quad u(1, t) = 0.$$

The source term is defined consistently with the exact solution. Since

$$D_{C,t}^\alpha t^2 = \frac{\Gamma(3)}{\Gamma(3-\alpha)} t^{2-\alpha},$$

The corresponding fractional derivative of the exact solution is

$$D_{C,t}^{\alpha} u(x, t) = \frac{\Gamma(3)}{\Gamma(3-\alpha)} t^{2-\alpha} \sin(\pi x).$$

Moreover,

$$u_{xx}(x, t) = -\pi^2 t^2 \sin(\pi x).$$

Therefore, the source function is written as

$$f(x, t) = \frac{\Gamma(3)}{\Gamma(3-\alpha)} t^{2-\alpha} \sin(\pi x) + \kappa \pi^2 t^2 \sin(\pi x) - \lambda t^2 \sin(\pi x) (1 - t^2 \sin(\pi x)).$$

Three numerical approaches are compared in this study. The first method is the classical L1 finite difference scheme, which is used as a deterministic numerical reference. The second method is the standard Physics-Informed Neural Network, where the residual is constructed directly from the governing differential equation. The third method is the proposed Improved Adaptive L1-PINN, where the L1 approximation of the Caputo derivative is embedded into the physics-informed residual and adaptive weights are used to balance the equation residual, the initial condition, and the boundary conditions during training.

The simulation is organized in a structured computational procedure. For each fractional order α , the classical L1 scheme is first applied on a uniform spatio-temporal grid. The standard PINN and the Improved Adaptive L1-PINN are then trained using the same initial and boundary information. The trained neural networks are evaluated on the full computational grid, and their outputs are compared with the exact solution. The comparison is performed using the above error measures and the computational time required by each method.

The Improved Adaptive L1-PINN method uses the following numerical residual:

$$R_{L1}(x_i, t_n; \theta) = A_{L1}(x_i, t_n; \theta) - \kappa \hat{u}_{xx}(x_i, t_n; \theta) - \lambda \hat{u}(x_i, t_n; \theta) [1 - \hat{u}(x_i, t_n; \theta)] - f(x_i, t_n).$$

The L1 approximation of the Caputo derivative is expressed as

$$A_{L1}(x_i, t_n; \theta) = \frac{\tau^{-\alpha}}{\Gamma(2-\alpha)} \sum_{j=0}^{n-1} a_j [\hat{u}(x_i, t_{n-j}; \theta) - \hat{u}(x_i, t_{n-j-1}; \theta)],$$

Where

$$a_j = (j+1)^{1-\alpha} - j^{1-\alpha}, \quad j = 0, 1, 2, \dots, n-1.$$

The adaptive loss function is expressed as

$$\mathcal{L}_A(\theta) = w_f \mathcal{L}_{L1-PDE} + w_i \mathcal{L}_{IC} + w_b \mathcal{L}_{BC},$$

where w_f , w_i , and w_b are updated during training according to the relative contribution of each loss component. This adaptive mechanism reduces the imbalance between the equation residual and the constraint residuals.

Tables 1-4 present the error comparison between the three methods for each fractional order. Table 5 reports the computational time and the number of iterations used in the numerical experiments. Table 6 studies the effect of the number of hidden layers and hidden neurons on the accuracy of the Improved Adaptive L1-PINN method.

Table 1: Comparison of MSE, RMSE, MAE, and R² between the three methods at $\alpha = 0.3$.

Method	MSE	RMSE	MAE	R ²
Classical L1	2.301777E-07	0.0004797684	0.0004153033	0.99999599
Standard PINN	0.05421355	0.23283803	0.16526998	0.05443552
Improved Adaptive L1-PINN	0.0004810526	0.02193291	0.01700876	0.99160973

Table 2: Comparison of MSE, RMSE, MAE, and R² between the three methods at $\alpha = 0.5$.

Method	MSE	RMSE	MAE	R ²
Classical L1	1.703256E-06	0.001305089	0.001106789	0.999970293
Standard PINN	0.02997679	0.17313806	0.12902035	0.47716047
Improved Adaptive L1-PINN	0.00470969	0.06862718	0.05191561	0.91785604

Table 3: Comparison of MSE, RMSE, MAE, and R² between the three methods at $\alpha = 0.7$.

Method	MSE	RMSE	MAE	R ²
Classical L1	1.048621E-05	0.003238242	0.002657568	0.999817105
Standard PINN	0.01222870	0.11058347	0.08335649	0.78671333
Improved Adaptive L1-PINN	0.0002484893	0.01576354	0.01067638	0.99566598

Table 4: Comparison of MSE, RMSE, MAE, and R^2 between the three methods at $\alpha = 0.9$.

Method	MSE	RMSE	MAE	R^2
Classical L1	5.828853E-05	0.007634693	0.006047416	0.998983362
Standard PINN	0.002501686	0.050016856	0.041448248	0.956366898
Improved Adaptive L1-PINN	0.002416454	0.049157439	0.042226317	0.9578535

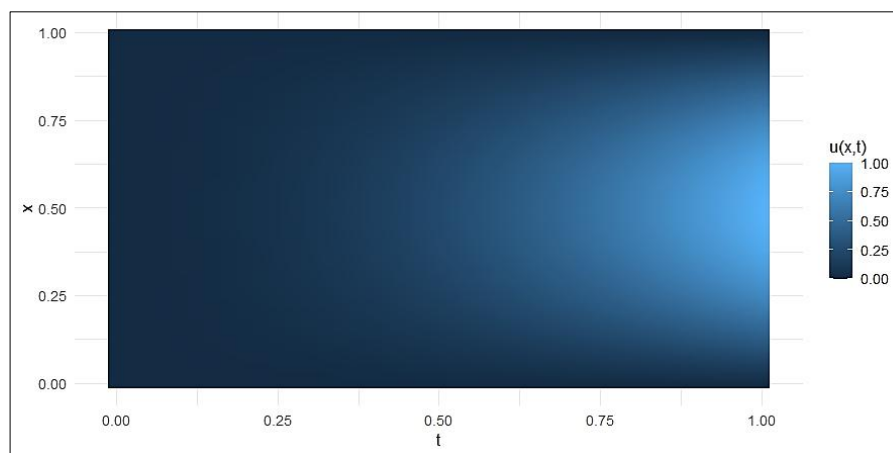
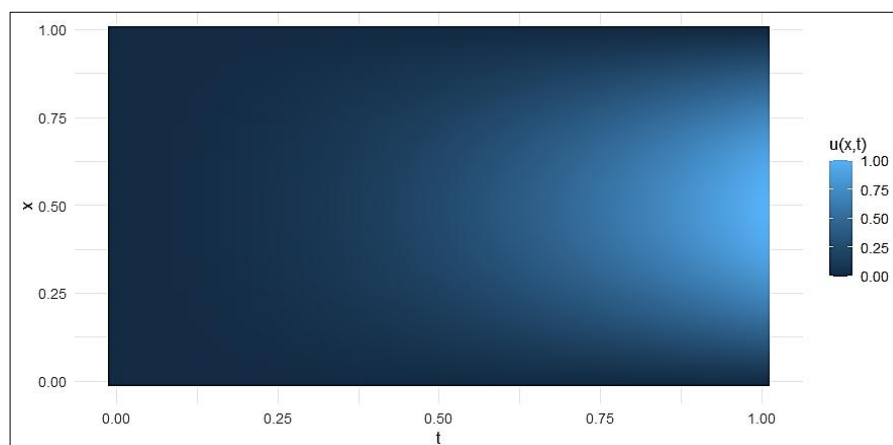
Table 5: Comparison of CPU time and number of iterations used by the three methods.

Alpha	Method	CPU time (seconds)	Iterations
0.3	Classical L1	0.263182163	157
0.3	Standard PINN	18.95168304	800
0.3	Improved Adaptive L1-PINN	9878.923205	800
0.5	Classical L1	0.228470087	154
0.5	Standard PINN	23.32269907	800
0.5	Improved Adaptive L1-PINN	10011.18926	800
0.7	Classical L1	0.113075018	150
0.7	Standard PINN	21.87445283	800
0.7	Improved Adaptive L1-PINN	9273.746445	800
0.9	Classical L1	0.104180098	143
0.9	Standard PINN	20.29670191	800
0.9	Improved Adaptive L1-PINN	10992.85997	800

Table 6: Effect of hidden layers and hidden neurons on the accuracy of the Improved Adaptive L1-PINN method at $\alpha = 0.7$.

α	Hidden layers	Hidden neurons	MSE	RMSE	MAE	R^2	CPU time (seconds)
0.7	2	20	0.0296271	0.1721254	0.1214635	0.4832583	3890.0489
0.7	3	30	0.0269350	0.1641190	0.1154483	0.5302129	4898.4815
0.7	4	40	0.0013005	0.0360634	0.0292018	0.9773160	6515.2878

The graphical analysis provides a visual interpretation of the numerical behavior of the three methods. Figures 1-4 display the exact and approximate surfaces over the computational domain. Figures 5-7 compare the exact and numerical solutions at fixed time levels. Figure 8 illustrates the decrease of the loss function during training, while Figure 9 shows the effect of the fractional order on numerical accuracy.

**Fig 1:** Exact solution $u(x, t)$ on $[0, 1] \times [0, 1]$.**Fig 2:** Approximate solution obtained by the classical L1 method.

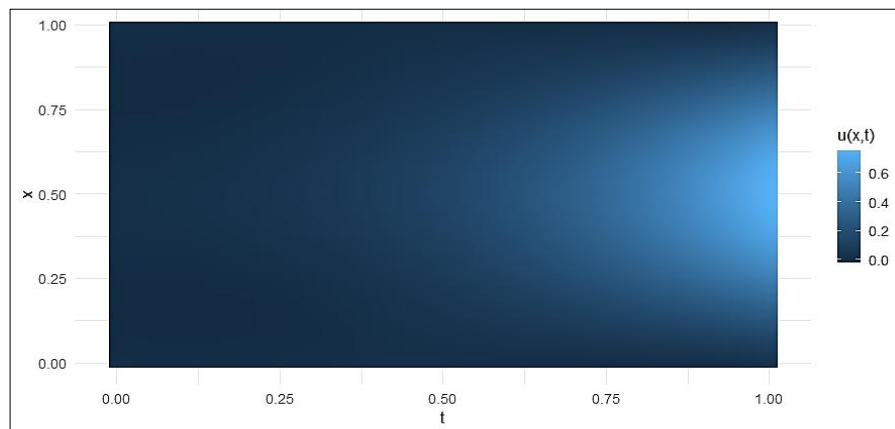


Fig 3: Approximate solution obtained by the standard PINN method.

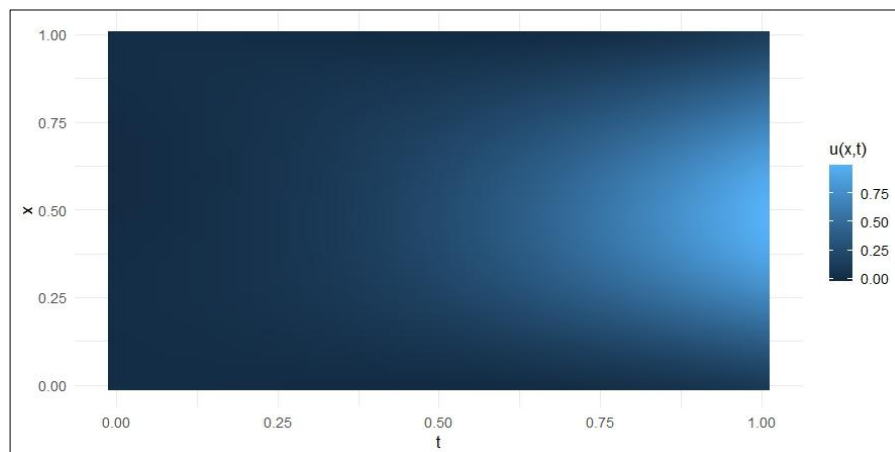


Fig 4: Approximate solution obtained by the Improved Adaptive L1-PINN method.

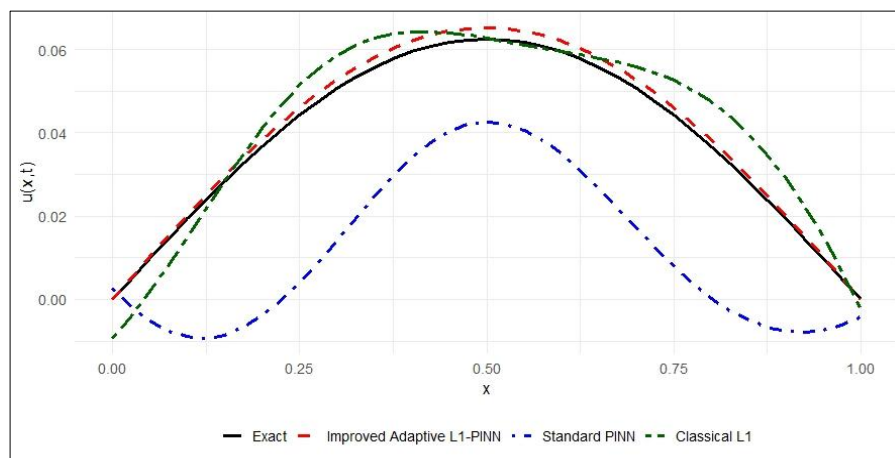


Fig 5: Comparison between the exact and numerical solutions at $t = 0.25$.

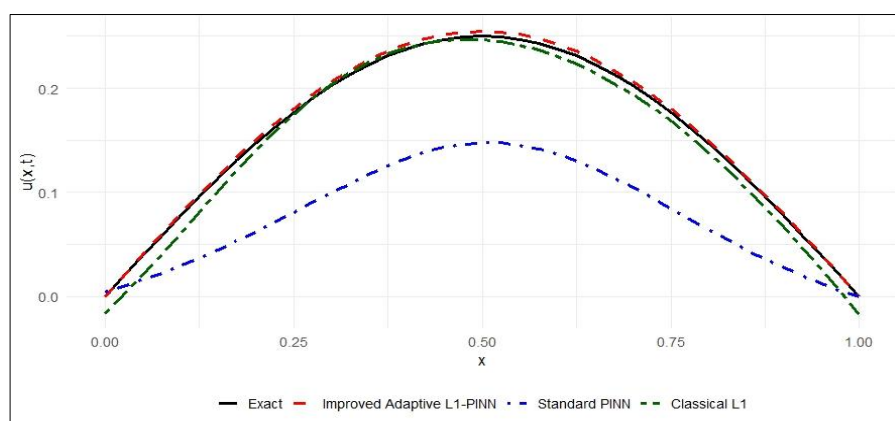


Fig 6: Comparison between the exact and numerical solutions at $t = 0.50$.

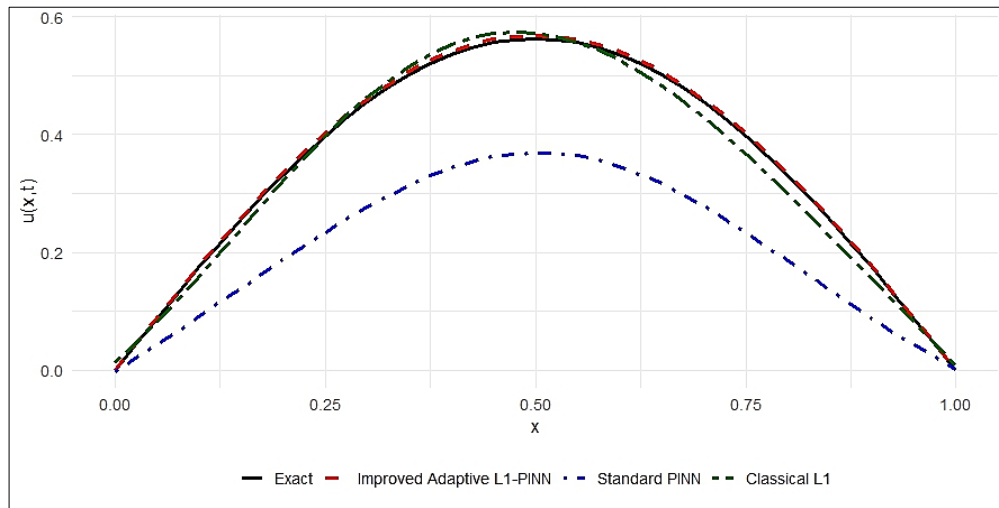


Fig 7: Comparison between the exact and numerical solutions at $t = 0.75$.

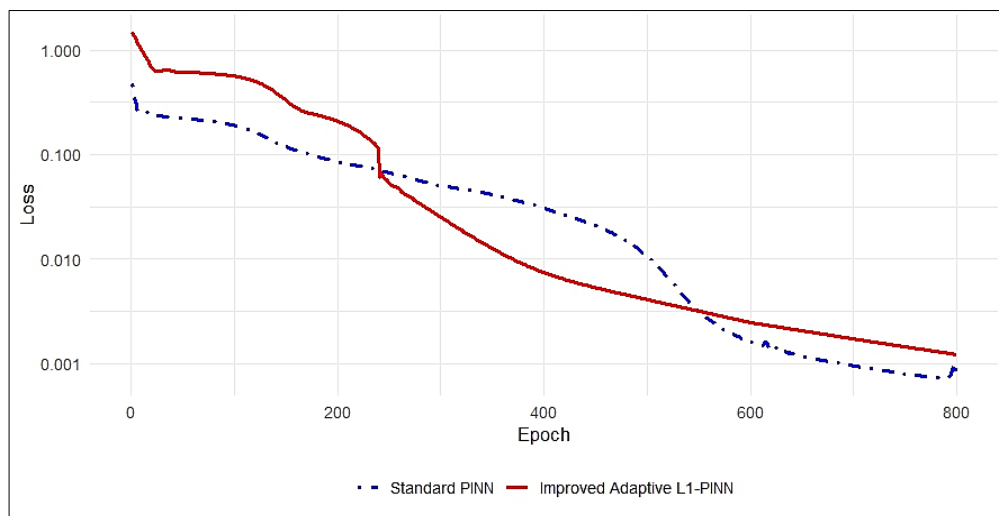


Fig 8: Loss decay curve during training.

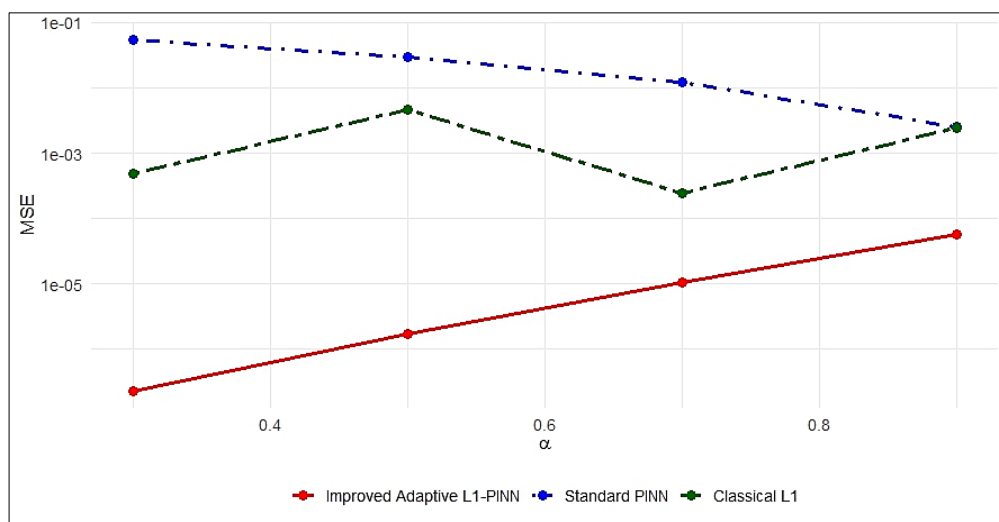


Fig 9: Effect of the fractional order α on solution accuracy.

The classical L1 scheme shows the least errors for all the values in fractional order. With $\alpha = 0.3$, the MSE is 2.301777×10^{-7} and the R^2 is 0.99999599, which shows very little variation from the exact solution. The error over time for classical L1 methods also gradually increases at higher α , reaching an MSE of 5.828853×10^{-5} at $\alpha = 0.9$ but maintaining high accuracy. Standard PINN generally yields the worst approximation. For $\alpha = 0.3$, it displays the MSE 0.05421355, and $R^2 = 0.05443552$, suggesting poor agreement with the correct solution. It can perform well even if α changes. At $\alpha = 0.9$, the MSE falls to 0.002501686, and the R^2 increases to 0.956366898. This means that for higher fractional orders in the current experiment both standard PINN can become more stable, but it is still less accurate than the deterministic L1 scheme.

The Improved Adaptive L1-PINN improves the normal PINN significantly in the vast majority of cases. For $\alpha = 0.3$, the MSE decreases from 0.05421355 for the standard PINN to 0.0004810526 for the Improved Adaptive L1-PINN. With $\alpha = 0.7$, the reported learning process lowers the MSE from 0.01222870 to 0.0002484893 and increases R^2 to 0.99566598 from 0.78671333. This demonstrates the utility of embedding the L1 approximation into the physics-informed residual and applying adaptive loss weights in training. When $\alpha = 0.9$, the standard PINN is close to the Improved Adaptive L1-PINN when error is present. The Improved Adaptive L1-PINN gives a slightly lower MSE and RMSE, whilst the standard PINN presents a slightly lower MAE. This indicates that the overall advantage of adaptive hybrid structure becomes less pronounced at one fractional order value, although also a marginally better global squared-error pattern is retained. It is clear the computational time results indicate an evident tradeoff between accuracy and cost.

The traditional L1 model is fast; it directly solves the discretized problem on a structured grid. The traditional PINN is slower due to its reliance on neural network training. As the L1 memory term is evaluated continuously throughout the training process, the Improved Adaptive L1-PINN requires the most computational time. Hence, the proposed approach indicates to be the optimal neural-network-based approximation in this experiment, however the classical L1 method is still the most accurate and fastest deterministic grid-based solver. The improved Adaptive L1-PINN performs much better with an increase in its network size according to Table 6. Thus when the architecture is changed from two hidden layers with 20 neurons to four hidden layers with 40 neurons, the MSE reduces from 0.02962717 to 0.00130057, and R^2 grows from 0.48325833 to 0.97731605.

This indicates a direct impact on the performance of the hybrid approximation due to the expressive capabilities of the neural network. Nonetheless, this enhancement raises the CPU time from 3890.04897 seconds to 6515.28789 seconds. We observe from numerical experiments that classical L1 method provides the best accuracy and lowest computational cost for the manufactured test problem. The Improved Adaptive L1-PINN has substantially improved the standard PINN, as well as a much more faithful learning-based approximation for nonlinear time-fractional diffusion-reaction equations. And its main superiority over the deterministic L1 scheme does not lie in the fact that it is superior, but rather that it augments traditional PINN which has introduced fractional memory representation in residual. This also serves as a robust way to do a mesh-based problem in which all solvers are difficult to implement or a flexible neural representation of the solution is needed.

Conclusion

We proposed an Improved Adaptive L1-PINN method for solving nonlinear time-fractional diffusion-reaction equations. The method combines the L1 approximation of the Caputo derivative with a physics-informed neural network and adaptive loss weights. The numerical results showed that the classical L1 scheme achieved the smallest errors and the lowest computational time for the manufactured test problem. This confirms its strength as a deterministic grid-based solver. The standard PINN provided weaker results because it could not fully capture the nonlocal memory effect of the fractional derivative. In contrast, the Improved Adaptive L1-PINN clearly improved the standard PINN, especially for $\alpha = 0.3$, $\alpha = 0.5$, and $\alpha = 0.7$. The results also showed that increasing the network size improves the accuracy of the proposed method, but increases the CPU time. Therefore, the proposed method offers a useful balance between neural flexibility and fractional-memory representation, although it remains computationally more expensive than the classical L1 scheme. Overall, the classical L1 method is the most accurate deterministic solver, while the Improved Adaptive L1-PINN is the best neural-network-based method in this comparison.

References

1. Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J Comput Phys.* 2019;378:686-707.
2. Cai S, Wang Z, Wang S, Perdikaris P, Karniadakis GE. Physics-informed neural networks for heat transfer problems. *J Heat Transfer.* 2021;143(6):060801.
3. Lawal ZK, Yassin H, Lai DTC, Che Idris A. Physics-informed neural network (PINN) evolution and beyond: A systematic literature review and bibliometric analysis. *Big Data Cogn Comput.* 2022;6(4):140.
4. Farea A, Yli-Harja O, Emmert-Streib F. Understanding physics-informed neural networks: Techniques, applications, trends, and challenges. *AI.* 2024;5(3):1534-1557.
5. Luo K, Zhao J, Wang Y, Li J, Wen J, Liang J, *et al.* Physics-informed neural networks for PDE problems: A comprehensive review. *Artif Intell Rev.* 2025;58(10):323.
6. Ren Z, Zhou S, Liu D, Liu Q. Physics-informed neural networks: A review of methodological evolution, theoretical foundations, and interdisciplinary frontiers toward next-generation scientific computing. *Appl Sci.* 2025;15(14):8092.
7. Malashin I, Tynchenko V, Gantimurov A, Nelyub V, Borodulin A. Physics-informed neural networks in polymers: A review. *Polymers (Basel).* 2025;17(8):1108.
8. Yao Z, Yang Z, Cheng L. On the stability preserving of L1 scheme to nonlinear time-fractional Schrödinger delay equations. *Math Comput Simul.* 2025;231:209-220.
9. Stynes M. A survey of the L1 scheme in the discretisation of time-fractional problems. *Numer Math Theory Methods Appl.* 2022;15(4):1173-1192.
10. Chen X, Chen A, Yan Y, Jin S. A modified L1 scheme for subdiffusion equations with a positive-type memory term. *J Comput Appl Math.* 2026;437:117729.
11. Wang J, Meng X, Yu Y. Local error estimate of L1 scheme for time-fractional convection-diffusion-reaction equation on a star-shaped pipe network. *Chin J Phys.* 2025;89:245-262.
12. Wu X, Yan Y. L1 scheme for semilinear stochastic subdiffusion with integrated fractional Gaussian noise. *Fractal Fract.* 2025;9(3):173.

Appendix A

```

clear; clc; close all;
rng(123);
%% Problem parameters
alphas = [0.3, 0.5, 0.7, 0.9];
kappa = 0.01;
lambda = 1.0;
T = 1.0;
Nx = 41;
Nt = 41;
x = linspace(0,1,Nx)';
t = linspace(0,T,Nt)';
h = x(2)-x(1);
tau = t(2)-t(1);
[X,TG] = meshgrid(x,t);
X = X';
TG = TG';
Nf = 800;
Ni = 100;
Nb = 100;
hiddenLayers = 3;
hiddenNeurons = 30;
maxEpochsPINN = 1000;
learningRate = 1e-3;

excelFile = 'L1_PINN_Numerical_Results.xlsx';
if exist(excelFile,'file')
    delete(excelFile);
end
allErrorTables = cell(numel(alphas),1);
timeRows = {};
networkRows = {};
lossHistoryPINN = [];
lossHistoryAdaptive = [];
solutionForPlot = struct();
%% Main numerical experiments
for aID = 1:numel(alphas)
    alpha = alphas(aID);
    fprintf('\n===== \n');
    fprintf('Running experiments for alpha =%.1f \n', alpha);
    fprintf('===== \n');
    Uexact = exactSolution(X,TG);
    %% Method 1: Classical L1 method
    tic;
    [UL1, iterL1] = solveClassicalL1(alpha,kappa,lambda,x,t);
    timeL1 = toc;
    errL1 = computeErrors(Uexact,UL1);
    %% Method 2: Standard PINN
    tic;
    [netPINN, lossPINN] = trainStandardPINN(alpha,kappa,lambda,Nf,Ni,Nb,...
        hiddenLayers,hiddenNeurons,maxEpochsPINN,learningRate);
    UPINN = predictOnGrid(netPINN,x,t);
    timePINN = toc;
    errPINN = computeErrors(Uexact,UPINN);
    %% Method 3: Adaptive L1-PINN
    tic;
    [netAdaptive, lossAdaptive] = trainAdaptiveL1PINN(alpha,kappa,lambda,x,t,Nf,Ni,Nb,...
        hiddenLayers,hiddenNeurons,maxEpochsPINN,learningRate);
    UAdaptive = predictOnGrid(netAdaptive,x,t);
    timeAdaptive = toc;
    errAdaptive = computeErrors(Uexact,UAdaptive);
    %% Store one case for surface and line plots
    if abs(alpha-0.7) < 1e-12
        lossHistoryPINN = lossPINN;
        lossHistoryAdaptive = lossAdaptive;
    end
end

```

```

solutionForPlot.alpha = alpha;
solutionForPlot.Uexact = Uexact;
solutionForPlot.UL1 = UL1;
solutionForPlot.UPINN = UPINN;
solutionForPlot.UAdaptive = UAdaptive;
end
%% Error table
Method = ["Classical L1"; "Standard PINN"; "Adaptive L1-PINN"];
MSE = [errL1.MSE; errPINN.MSE; errAdaptive.MSE];
RMSE = [errL1.RMSE; errPINN.RMSE; errAdaptive.RMSE];
MAE = [errL1.MAE; errPINN.MAE; errAdaptive.MAE];
R2 = [errL1.R2; errPINN.R2; errAdaptive.R2];
Terrors = table(Method,MSE,RMSE,MAE,R2);
allErrorTables{aID} = Terrors;

sheetName = sprintf('Table_%d_alpha_%.1f',aID,alpha);
writetable(Terrors,excelFile,'Sheet',sheetName);
%% CPU time table rows
timeRows(end+1,:) = {alpha,'Classical L1',timeL1,iterL1};
timeRows(end+1,:) = {alpha,'Standard PINN',timePINN,maxEpochsPINN};
timeRows(end+1,:) = {alpha,'Adaptive L1-PINN',timeAdaptive,maxEpochsPINN};
fprintf('\nError table for alpha =%.1f\n', alpha);
disp(Terrors);
end
%% Table 5: CPU time and iterations
Ttime = cell2table(timeRows,'VariableNames',{'Alpha','Method','CPU_Time_seconds','Iterations'});
writetable(Ttime,excelFile,'Sheet','Table_5_CPU_Time');
%% Table 6: Effect of network size
fprintf('\nRunning Table 6: Network size experiment...\n');
alphaNet = 0.7;
networkConfigs = [
    2 20
    3 30
    4 40
];
for c = 1:size(networkConfigs,1)
    L = networkConfigs(c,1);
    H = networkConfigs(c,2);
    tic;
    [netTemp, ~] = trainAdaptiveL1PINN(alphaNet,kappa,lambda,x,t,Nf,Ni,Nb,...
    L,H,700,learningRate);
    Utemp = predictOnGrid(netTemp,x,t);
    cpuTemp = toc;
    UexactTemp = exactSolution(X,TG);
    errTemp = computeErrors(UexactTemp,Utemp);
    networkRows(end+1,:) = {alphaNet,L,H,errTemp.MSE,errTemp.RMSE,errTemp.MAE,errTemp.R2,cpuTemp};
end
Tnetwork =
=
cell2table(networkRows,'VariableNames',...
{'Alpha','Hidden_Layers','Hidden_Neurons','MSE','RMSE','MAE','R2','CPU_Time_seconds'});
writetable(Tnetwork,excelFile,'Sheet','Table_6_Network_Size');
%% Figures
Uexact = solutionForPlot.Uexact;
UL1 = solutionForPlot.UL1;
UPINN = solutionForPlot.UPINN;
UAdaptive = solutionForPlot.UAdaptive;
%% Figure 1
figure;
surf(TG,X,Uexact,'EdgeColor','none');
xlabel('t'); ylabel('x'); zlabel('u(x,t)');
title('Figure 1. Exact solution u(x,t)');
view(135,30); colorbar;
saveas(gcf,'Figure_1_Exact_Solution.png');
%% Figure 2
figure;

```

```

surf(TG,X,UL1,'EdgeColor','none');
xlabel('t'); ylabel('x'); zlabel('u(x,t)');
title('Figure 2. Approximate solution by the classical L1 method');
view(135,30); colorbar;
saveas(gcf,'Figure_2_L1_Solution.png');
%% Figure 3
figure;
surf(TG,X,UPINN,'EdgeColor','none');
xlabel('t'); ylabel('x'); zlabel('u(x,t)');
title('Figure 3. Approximate solution by the standard PINN method');
view(135,30); colorbar;
saveas(gcf,'Figure_3_PINN_Solution.png');
%% Figure 4
figure;
surf(TG,X,UAdaptive,'EdgeColor','none');
xlabel('t'); ylabel('x'); zlabel('u(x,t)');
title('Figure 4. Approximate solution by the Adaptive L1-PINN method');
view(135,30); colorbar;
saveas(gcf,'Figure_4_Adaptive_L1_PINN_Solution.png');
%% Figures 5--7
timesToPlot = [0.25, 0.50, 0.75];
for q = 1:numel(timesToPlot)
    tq = timesToPlot(q);
    [~,idxT] = min(abs(t-tq));
    figure;
    plot(x,Uexact(:,idxT),'LineWidth',2); hold on;
    plot(x,UL1(:,idxT),'--','LineWidth',2);
    plot(x,UPINN(:,idxT),'-','LineWidth',2);
    plot(x,UAdaptive(:,idxT),'.','LineWidth',2);
    xlabel('x'); ylabel('u(x,t)');
    legend('Exact','Classical L1','Standard PINN','Adaptive L1-PINN','Location','best');
    title(sprintf('Figure%d. Comparison at t =%.2f,q+4,t(idxT)'));
    grid on;
    saveas(gcf,sprintf('Figure_%d_Comparison_t_%.2f.png',q+4,tq));
end
%% Figure 8
figure;
semilogy(lossHistoryPINN,'LineWidth',2); hold on;
semilogy(lossHistoryAdaptive,'LineWidth',2);
xlabel('Epoch'); ylabel('Loss');
legend('Standard PINN','Adaptive L1-PINN','Location','best');
title('Figure 8. Loss decay during training');
grid on;
saveas(gcf,'Figure_8_Loss_Decay.png');
%% Figure 9
mseL1 = zeros(numel(alphas),1);
msePINN = zeros(numel(alphas),1);
mseAdaptive = zeros(numel(alphas),1);
for i = 1:numel(alphas)
    Terrors = allErrorTables{i};
    mseL1(i) = Terrors.MSE(1);
    msePINN(i) = Terrors.MSE(2);
    mseAdaptive(i) = Terrors.MSE(3);
end
figure;
plot(alphas,mseL1,'-o','LineWidth',2); hold on;
plot(alphas,msePINN,'-s','LineWidth',2);
plot(alphas,mseAdaptive,'-d','LineWidth',2);
xlabel('\alpha'); ylabel('MSE');
legend('Classical L1','Standard PINN','Adaptive L1-PINN','Location','best');
title('Figure 9. Effect of fractional order \alpha on solution accuracy');
grid on;
saveas(gcf,'Figure_9_Alpha_Effect.png');
fprintf('\nAll tables have been exported to%s\n', excelFile);
fprintf('All figures have been saved as PNG files.\n');

```

```

%% =====
% Local functions
% =====

function u = exactSolution(x,t)
    u = t.^2.* sin(pi*x);
end

function f = sourceTerm(x,t,alpha,kappa,lambda)
    u = exactSolution(x,t);
    caputoTerm = gamma(3)/gamma(3-alpha).* t.^(2-alpha).* sin(pi*x);
    uxx = -pi^2.* t.^2.* sin(pi*x);
    f = caputoTerm - kappa.*uxx - lambda.*u.*(1-u);
end

function err = computeErrors(Utrue,Upred)
    y = Utrue(:);
    yp = Upred(:);
    mse = mean((y-yp).^2);
    rmse = sqrt(mse);
    mae = mean(abs(y-yp));
    den = sum((y-mean(y)).^2);
    if den == 0
        r2 = NaN;
    else
        r2 = 1 - sum((y-yp).^2)/den;
    end
    err.MSE = mse;
    err.RMSE = rmse;
    err.MAE = mae;
    err.R2 = r2;
end

function [U, iterations] = solveClassicalL1(alpha,kappa,lambda,x,t)
    Nx = numel(x);
    Nt = numel(t);
    h = x(2)-x(1);
    tau = t(2)-t(1);
    U = zeros(Nx,Nt);
    U(:,1) = exactSolution(x,0);
    a = zeros(Nt,1);
    for j = 0:Nt-1
        a(j+1) = (j+1)^(1-alpha) - j^(1-alpha);
    end
    coef = tau^(-alpha)/gamma(2-alpha);
    maxNewton = 30;
    tolNewton = 1e-11;
    iterations = 0;
    for n = 2:Nt
        tn = t(n);
        U(1,n) = 0;
        U(end,n) = 0;
        U(2:end-1,n) = U(2:end-1,n-1);
        for it = 1:maxNewton
            iterations = iterations + 1;
            R = zeros(Nx-2,1);
            J = zeros(Nx-2,Nx-2);
            for ii = 2:Nx-1
                loc = ii-1;
                histSum = 0;
                if n > 2
                    for j = 1:n-2
                        histSum = histSum + a(j+1)*(U(ii,n-j)-U(ii,n-j-1));
                    end
                end
                caputoApprox = coef*((U(ii,n)-U(ii,n-1)) + histSum);
                uxx = (U(ii+1,n)-2*U(ii,n)+U(ii-1,n))/h^2;
                fval = sourceTerm(x(ii),tn,alpha,kappa,lambda);
                R(loc) = caputoApprox - kappa*uxx - lambda*U(ii,n)*(1-U(ii,n)) - fval;
            end
        end
    end
end

```

```

J(loc,loc) = coef + 2*kappa/h^2 - lambda*(1-2*U(ii,n));
if loc > 1
J(loc,loc-1) = -kappa/h^2;
end
if loc < Nx-2
J(loc,loc+1) = -kappa/h^2;
end
end
delta = -J\R;
U(2:end-1,n) = U(2:end-1,n) + delta;
if norm(delta,inf) < tolNewton
break;
end
end
end
end
function net = initializeNetwork(numHiddenLayers,numNeurons)
layers = [
featureInputLayer(2,'Normalization','none','Name','input')
fullyConnectedLayer(numNeurons,'Name','fc1')
tanhLayer('Name','tanh1')
];
for i = 2:numHiddenLayers
layers = [
layers
fullyConnectedLayer(numNeurons,'Name',sprintf('fc%d',i))
tanhLayer('Name',sprintf('tanh%d',i))
];
end
layers = [
layers
fullyConnectedLayer(1,'Name','output')
];
net = dlnetwork(layerGraph(layers));
end
function [net,lossHistory] = trainStandardPINN(alpha,kappa,lambda,Nf,Ni,Nb,numLayers,numNeurons,maxEpochs,learningRate)

net = initializeNetwork(numLayers,numNeurons);
xF = rand(Nf,1);
tF = rand(Nf,1);
xI = rand(Ni,1);
tI = zeros(Ni,1);
tB = rand(Nb,1);
xB0 = zeros(Nb,1);
xB1 = ones(Nb,1);
XF = dlarray([xF;tF],'CB');
XI = dlarray([xI;tI],'CB');
XB0 = dlarray([xB0;tB'],'CB');
XB1 = dlarray([xB1;tB'],'CB');
avgGrad = [];
avgSqGrad = [];
lossHistory = zeros(maxEpochs,1);
for epoch = 1:maxEpochs
[loss,gradients] = dlfeval(@standardPINNLoss,net,XF,XI,XB0,XB1,alpha,kappa,lambda);
[net,avgGrad,avgSqGrad] = adamupdate(net,gradients,avgGrad,avgSqGrad,epoch,learningRate);
lossHistory(epoch) = double(gather(extractdata(loss)));
if mod(epoch,200) == 0
fprintf('Standard PINN Epoch%d, Loss =%.4e\n',epoch,lossHistory(epoch));
end
end
end
function [loss,gradients] = standardPINNLoss(net,XF,XI,XB0,XB1,alpha,kappa,lambda)
u = forward(net,XF);
gradU = dlgradient(sum(u,'all'),XF,'EnableHigherDerivatives',true);
ux = gradU(1,:);

```

```

ut = gradU(2,:);
gradUx = dlgradient(sum(ux,'all'),XF,'EnableHigherDerivatives',true);
uxx = gradUx(1,:);
xF = XF(1,:);
tF = XF(2,:);
fval = sourceTerm(xF,tF,alpha,kappa,lambda);
res = ut - kappa.*uxx - lambda.*u.*(1-u) - fval;
lossPDE = mean(res.^2,'all');
uI = forward(net,XI);
uItrue = exactSolution(XI(1,:),XI(2,:));
lossIC = mean((uI-uItrue).^2,'all');
uB0 = forward(net,XB0);
uB1 = forward(net,XB1);
g1 = zeros(size(uB0),'like',uB0);
g2 = zeros(size(uB1),'like',uB1);
lossBC = mean((uB0-g1).^2,'all') + mean((uB1-g2).^2,'all');
loss = lossPDE + lossIC + lossBC;
gradients = dlgradient(loss,net.Learnables);
end
function [net,lossHistory] =
trainAdaptiveL1PINN(alpha,kappa,lambda,xgrid,tgrid,Nf,Ni,Nb,numLayers,numNeurons,maxEpochs,learningRate)
net = initializeNetwork(numLayers,numNeurons);
Nx = numel(xgrid);
Nt = numel(tgrid);

ix = randi([2,Nx-1],Nf,1);
it = randi([3,Nt],Nf,1);
xI = rand(Ni,1);
tI = zeros(Ni,1);
tB = rand(Nb,1);
xB0 = zeros(Nb,1);
xB1 = ones(Nb,1);
XI = dlarray([xI;tI],'CB');
XB0 = dlarray([xB0;tB'],'CB');
XB1 = dlarray([xB1;tB'],'CB');
eta = 1.0;
w = [1/3,1/3,1/3];
avgGrad = [];
avgSqGrad = [];
lossHistory = zeros(maxEpochs,1);
for epoch = 1:maxEpochs
[loss,lossParts,gradients] = dlfeval(@adaptiveL1PINNLoss,net,xgrid,tgrid,ix,it,XI,XB0,XB1,alpha,kappa,lambda,w);
parts = double(gather(extractdata(lossParts)));
parts = parts(:);
expParts = exp(eta*parts/(sum(parts)+eps));
w = expParts/sum(expParts);
[net,avgGrad,avgSqGrad] = adamupdate(net,gradients,avgGrad,avgSqGrad,epoch,learningRate);
lossHistory(epoch) = double(gather(extractdata(loss)));
if mod(epoch,200) == 0
fprintf('Adaptive L1-PINN Epoch%d, Loss =%.4e\n',epoch,lossHistory(epoch));
end
end
end
function [loss,lossParts,gradients] = adaptiveL1PINNLoss(net,xgrid,tgrid,ix,it,XI,XB0,XB1,alpha,kappa,lambda,w)
Nf = numel(ix);
tau = tgrid(2)-tgrid(1);
h = xgrid(2)-xgrid(1);
coef = tau^(-alpha)/gamma(2-alpha);
resList = dlarray(zeros(1,Nf),'CB');
for k = 1:Nf
i = ix(k);
n = it(k);
xNow = xgrid(i);
tNow = tgrid(n);
Xnow = dlarray([xNow;tNow],'CB');

```

```

uNow = forward(net,Xnow);
Xp = dldarray([xgrid(i+1);tNow],'CB');
Xm = dldarray([xgrid(i-1);tNow],'CB');
up = forward(net,Xp);
um = forward(net,Xm);
uxx = (up - 2*uNow + um)/h^2;
l1sum = dldarray(0,'CB');
for j = 0:n-2
aj = (j+1)^(1-alpha) - j^(1-alpha);
t1 = tgrid(n-j);
t0 = tgrid(n-j-1);

X1 = dldarray([xNow;t1],'CB');
X0 = dldarray([xNow;t0],'CB');
u1 = forward(net,X1);
u0 = forward(net,X0);
l1sum = l1sum + aj*(u1-u0);
end
caputoL1 = coef*l1sum;
fval = sourceTerm(xNow,tNow,alpha,kappa,lambda);
resList(k) = caputoL1 - kappa.*uxx - lambda.*uNow.*(1-uNow) - fval;
end
lossPDE = mean(resList.^2,'all');
uI = forward(net,XI);
uItrue = exactSolution(XI(1,:),XI(2,:));
lossIC = mean((uI-uItrue).^2,'all');
uB0 = forward(net,XB0);
uB1 = forward(net,XB1);
g1 = zeros(size(uB0),'like',uB0);
g2 = zeros(size(uB1),'like',uB1);
lossBC = mean((uB0-g1).^2,'all') + mean((uB1-g2).^2,'all');
loss = w(1)*lossPDE + w(2)*lossIC + w(3)*lossBC;
lossParts = [lossPDE; lossIC; lossBC];
gradients = dlgradient(loss,net.Learnables);
end
function U = predictOnGrid(net,x,t)
Nx = numel(x);
Nt = numel(t);
U = zeros(Nx,Nt);
for n = 1:Nt
Xinput = [x'; t(n)*ones(1,Nx)];
dlX = dldarray(Xinput,'CB');
pred = forward(net,dlX);
U(:,n) = gather(extractdata(pred));
end
end

```