Electronic Research Archive

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

ERA, 33(2): 826–848.
DOI: 10.3934/era.2025037
Received: 08 December 2024
Revised: 19 January 2025
Accepted: 23 January 2025
Published: 13 February 2025

Research article

Advanced machine learning technique for solving elliptic partial differential equations using Legendre spectral neural networks

Ishtiaq Ali[*]

Department of Mathematics and Statistics, College of Science, King Faisal University, P. O. Box 400, Al-Ahsa 31982, Saudi Arabia

* **Correspondence:** Email: iamirzada@kfu.edu.sa.

Abstract: In this work, a novel approach based on a single-layer machine learning Legendre spectral neural network (LSNN) method is used to solve an elliptic partial differential equation. A Legendre polynomial based approach is utilized to generate neurons that fulfill the boundary conditions. The loss function is computed by using the error back-propagation principles and a feed-forward neural network model combined with automatic differentiation. The main advantage of using this methodology is that it does not need to solve a system of nonlinear and nonsparse equations compared with other traditional numerical schemes, which makes this algorithm more convenient for solving higher-dimensional equations. Further, the hidden layer is eliminated with the help of a Legendre polynomial to enlarge the input pattern. The neural network's training accuracy and efficiency were significantly enhanced by the innovative sampling technique and neuron architecture. Moreover, the Legendre spectral approach can handle equations on more complex domains because of numerous networks. Several test problems were used to validate the proposed scheme, and a comparison was made with other neural network schemes consisting of the physics-informed neural network (PINN) scheme. We found that our proposed scheme has a very good agreement with PINN, which further enhances the reliability and efficiency of our proposed method. The absolute and relative error in both L_2 and L_∞ between exact and numerical solutions are provided, which shows that our numerical method converges exponentially.

Keywords: feed-forward neural network; single-layer Legendre neural network; automatic differentiation; elliptic partial differential equations; complex domains

1. Introduction

Among the most frequent outcomes of mathematical modeling for describing physical, biological, or chemical phenomena are partial differential equations (PDEs), which are used in all branches of engineering and science. In addition, a dramatic increase has been seen in recent years in the use of these PDEs in fields like chemistry, computer sciences (especially in relation to image processing and graphics), and economics (finance) [1]. These PDEs are subject to some appropriate initial and boundary conditions. Solving these PDEs and analyzing their solutions is one of the key factors. In most cases, the closed form solution of these PDEs is not possible or is very hard to obtain. Therefore, one must look for some approximate method to find their solution. Most of these methods are based on predictor–corrector, Runge–Kutta, finite difference, and finite element [2–4]. In order to use these numerical approaches, the domain must be discretized into a number of finite domains or locations where the functions are locally approximated. However, such methods still have many limitations, despite their strength and potential to help us approximate solutions to issues where analytical approaches are unable to. Finite difference techniques involve the creation of a grid on which the solution is to be found and can be difficult to specify over complicated geometries. Similar to the finite difference approach, the finite element method is simple to define over complicated geometries but can struggle with nonlinear issues and may become slow or unfeasible when dealing with high-dimensional problems or when an extraordinarily small error is needed. There is still a need for innovative numerical techniques to solve PDEs because of these constraints [5].

Elliptic PDEs are a class of PDEs that explain the global and steady-state phenomena that arise in many areas of computational sciences, such as fluid dynamics, heat transfer, electromagnetism, geophysics, biology, and other application areas. They are used by coastal engineers to approximate the motion of the sea, and are used to model electric potential. Many elliptical problems can be solved easily, but those with complicated meshes or complex geometries are more difficult to solve. Only in very rare situations can closed analytical expressions for their solutions be found, most of which are of not much theoretical or practical significance. Apart from this, many elliptical PDEs are defined in higher-dimensional domains, which cause computational complexities for traditional numerical schemes. Therefore, it is natural that mathematicians and scientists look for techniques for estimating solutions. In fact, the development of computational mathematics, which focuses primarily on the development and mathematical analysis of numerical methods for the approximate solution of elliptic PDEs, has been accelerated by the introduction of digital computers.

Neural network techniques are emerging as effective tools for resolving PDEs, especially elliptical PDEs when alternative approaches are impractical or may not work. If one can figure out a way to sample from the underlying domain, neural networks can be used to solve high-dimensional problems without suffering too much from the curse of dimensionality, as well as nonlinear partial differential equations, and even domains with complex geometries. Neural network-based approaches were employed whenever the need for highly precise solutions to PDEs was in demand. In terms of accuracy, the neural networks that were able to acquire any nonlinear function performed more effectively than the conventional techniques. More than two and a half decades ago, Lagaris et al. [6] applied neural networks to solve ordinary and PDEs, which marked the beginning of the use of neural networks in the solution of differential equations. Since then, there have been a number of advancements in the use of neural networks in the numerical solution of differential equations. For example, Mall and Chakraverty in [7], and Dufera in [8] introduced a deep neural network-based approach to solve ordinary differential equations (ODEs). The neural network was trained using the traditional gradient descent approach. Rivera et al. in [9], proposed a method to solve the PDEs using neural networks after determining the loss function using the quadrature approach. The quadrature weight was added to the loss function using this application of the quadrature rule. Tan et al. [10]

solved two-dimensional elliptic PDEs using a machine learning technique based on the unsupervised form of wavelet neural networks. Recently, Sabir et al. [11] used artificial neural network procedures to solve the susceptible-exposed-infectious-recovered (SEIR) mathematical model for Zika virus transmission, together with the hybridization effectiveness of local search schemes and global swarming. An accurate, fast, and reliable neural network approach based on physics that solves differential equations was used in [12]. For more details about the use of neural network technique to solve integer-order differential equations, we refer the reader to [13–18]. Apart from solving the integer-order differential equations, the neural network technique has also been effectively applied to solve fractional-order differential equations [19, 20].

The development of techniques for solving PDEs has advanced significantly in scientific computing over the past years. Because of their natural hierarchical structure, links to approximation theory, and advantageous convergence qualities, spectral methods are an important component of scientific computing's tools. For the purpose of guaranteeing that the underlying PDE is satisfied in a suitable sense, spectral methods often expand the solution of a PDE as a linear combination of basis functions and estimate the coefficients of the linear combination. Despite their potential potency, spectral approaches' are highly dependent on the basis function selection, which is not always clear for practical uses. The geometry of the domain where the answer is to be approximated can be a source of difficulties. For instance, while commonly used basis functions, such as orthogonal polynomials, are appropriate only for regular domains, fluid dynamics applications usually use complicated domains. The existence of highly localized characteristics in the solution, such as excessively steep gradients, might also cause issues. The approximation of the order function characterizing the evolving sharp phase boundary is one example of an application in phase field modeling. Unless the specifics of the application are taken into account while building the basis functions, the resolution of such localized features can reduce the effectiveness of a spectrum approach because the basis functions employed in spectral methods are global in nature [21–25]. For more details about spectral methods, we refer the reader to [26–31].

Among the significant sequences of orthogonal polynomials that have been extensively investigated and are utilized in computational fluid dynamics, interpolation and approximation theory, numerical integration, and the solution of differential equations, particularly elliptic differential equations, are the Legendre polynomials. These are an essential part of numerical integration and the approximate solution of differential and integral equations, along with being an effective tool for approximating hard-to-calculate functions. Excellent error characteristics are demonstrated by the Legendre spectral methods when approximating a smooth function. Orthogonal polynomial expansion appears in many real-world scenarios and applications and is crucial to many branches of physics and mathematics [32]. Typical neural network structures have several disadvantages, despite the fact that they have generated outstanding outcomes when it comes to solving differential equations. For example physics-informed neural networks (PINNs) have a weaker theoretical underpinning and a lack of useful error analysis tools due to their recent inception as a numerical method. Since the provided loss function in PINNs is frequently non-convex, it might be difficult to comprehend the convergence criteria in the optimization process, where solutions may become trapped in the local minima. It is necessary to overcome problems such as weak generality and low solution precision, and, in some cases, training can be difficult, particularly when the equation's solution has high-frequency features [33]. Other issues with conventional neural networks, including the local minimum, poor convergence speed, and initial weight values. As a result, numerous scientists are investigating ways to overcome these obstacles by adjusting the network architecture, and training methodologies, as well as other strategies to overcome these challenges.

Traditional spectral methods, such as Fourier and Chebyshev spectral methods, rely on solving systems of linear or nonlinear equations derived from the discretization of the governing PDE. This process involves matrix assembly and factorization, which can become computationally expensive for high-dimensional problems or irregular domains. Additionally, the global nature of traditional spectral basis functions may require a dense representation of the solution domain, further increasing computational effort. In contrast, the Legendre spectral neural network (LSNN) bypasses the need for matrix assembly by directly minimizing a loss function through an optimization process. The elimination of hidden layers and the use of Legendre–Gauss–Lobatto points significantly reduce the computational overhead. Moreover, the LSNN employs automatic differentiation for evaluating derivatives, which simplifies the implementation and avoids the overhead associated with numerical differentiation. Inspired by [34], a novel single-layer neural network is developed in this study that is based on the Legendre spectral method, which is a natural choice due to its exponential rate of convergence combined with its inherited property to satisfy the boundary conditions without making any adjustment, and a robust strategy for sampling in the form of Legendre–Gauss–Lobatto points enhances the training speed and accuracy. Neural network methods based on Legendre polynomials have been extensively used for solving differential equations in [35–41]. Most recently, the PINN technique for solving the PDEs, mainly the elliptic PDEs has been successfully used in [42–48]. Among these methods, the LSNN method holds significant potential for real-world applications across various fields. In fluid dynamics, it can efficiently model steady-state flows and heat transfer in complex geometries. In material science, LSNNs can simulate stress-strain analysis and thermal conduction with high precision.

The remainder of the paper is structured as follows. In Section 2, we demonstrate the proposed scheme for both regular and irregular geometries in details. Section 3 provides some error analysis, followed by a number of numerical examples in Section 4. Section 5 presents the concluding remarks.

2. Description of the Legendre spectral neural network method

Consider the general differential equation for which we seek the solution $v(\boldsymbol{x})$, defined over a domain $\mathcal{D} \subset \mathbb{R}^d$, of the form:

$$\mathcal{G}(\boldsymbol{x}, v(\boldsymbol{x}), \nabla v(\boldsymbol{x}), \nabla^2 v(\boldsymbol{x}), \ldots, \nabla^m v(\boldsymbol{x})) = 0, \quad \boldsymbol{x} \in \mathcal{D}, \tag{2.1}$$

where $\mathcal{G}$ is the operator defining the structure of the differential equation, $v(\boldsymbol{x})$ is the unknown solution, and $\nabla, \nabla^2, \ldots, \nabla^m$ represent the gradient, Hessian, and higher-order derivatives of $v(\boldsymbol{x})$. Equation (2.1) is subject to boundary conditions (BCs) given by (2.2)

$$\mathcal{H}(v(\boldsymbol{x}), \boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \partial\mathcal{D}, \tag{2.2}$$

where $\mathcal{H}$ represents the boundary operator, imposing conditions such as Dirichlet, Neumann, or Robin.

To solve the model equation (2.1), subject to the BCs given by Eq (2.2) using the LSNN method, we construct the approximate function space for the solution based on Legendre polynomials of the form:

$$\Upsilon_k(x) = L_k(x) + \gamma_k L_{k+1}(x) + \delta_k L_{k+2}(x), \tag{2.3}$$

where $L_k(x)$ denotes the k-th Legendre polynomial and γ_k, δ_k are coefficients determined by the boundary conditions. Legendre–Gauss–Lobatto (LGL) points are special sampling points that include the endpoints of the domain, -1 and 1, ensuring that the boundary conditions are automatically satisfied. These points are the roots of the equation:

$$(1 - x^2)L_k'(x) = 0. \tag{2.4}$$

The corresponding weights for the LGL points are given by:

$$w_i = \frac{2}{k(k+1)[L_k(x_i)]^2}, \tag{2.5}$$

where x_i are the LGL points, k is the polynomial degree, and w_i is the weight associated with x_i.

2.1. *LSNNs for rectangular geometry*

Consider a one-dimensional boundary value problem with homogeneous boundary conditions:

$$p_1 v(-1) + q_1 v'(-1) = 0, \quad p_2 v(1) + q_2 v'(1) = 0. \tag{2.6}$$

If the boundary conditions are non-homogeneous, the problem is homogenized by introducing a function $W(x)$ such that:

$$p_1 W(-1) + q_1 W'(-1) = c_1, \quad p_2 W(1) + q_2 W'(1) = c_2. \tag{2.7}$$

The solution $v(x)$ is then written as:

$$v(x) = W(x) - \hat{v}(x), \tag{2.8}$$

where $\hat{v}(x)$ satisfies the homogeneous boundary conditions. In this case, the coefficients γ_k, δ_k given in Eq (2.1) are given by:

$$\gamma_k = -\frac{(p_2 + q_2(k+2)^2)(-p_1 + q_1 k^2) - (p_1 - q_1(k+2)^2)(-p_2 - q_2 k^2)}{\Delta_k}, \tag{2.9}$$

$$\delta_k = \frac{(p_2 + q_2(k+1)^2)(-p_1 + q_1 k^2) + (p_1 - q_1(k+1)^2)(-p_2 - q_2 k^2)}{\Delta_k}, \tag{2.10}$$

where the determinant Δ_k is given by:

$$\Delta_k = 2p_2 p_1 - 2q_2 q_1 (k+1)^2 (k+2)^2 + (p_1 q_2 - p_2 q_1)((k+1)^2 + (k+2)^2). \tag{2.11}$$

For homogeneous Dirichlet boundary conditions, the basis functions simplify to:

$$\Upsilon_k(x) = L_k(x) - L_{k+2}(x). \tag{2.12}$$

The approximation space for one-dimensional problems is represented as:

$$\mathcal{V}_N = \text{span}\{\Upsilon_i(x) : i = 0, 1, \ldots, N-2\}. \tag{2.13}$$

For higher-dimensional domains, such as $\mathcal{D} = [a, b] \times [c, d]$, the basis functions are formed by tensor products:

$$\mathcal{V}_N = \text{span}\{\Upsilon_i(x)\Upsilon_j(y) : i, j = 0, 1, \ldots, N-2\}. \tag{2.14}$$

The detailed structure of an LSNN is shown in Figure 1. In the context of LSNNs, the residual points for training are chosen as LGL points, which ensures that the spectral properties of Legendre polynomials are fully utilized and that the solution space achieves high accuracy.

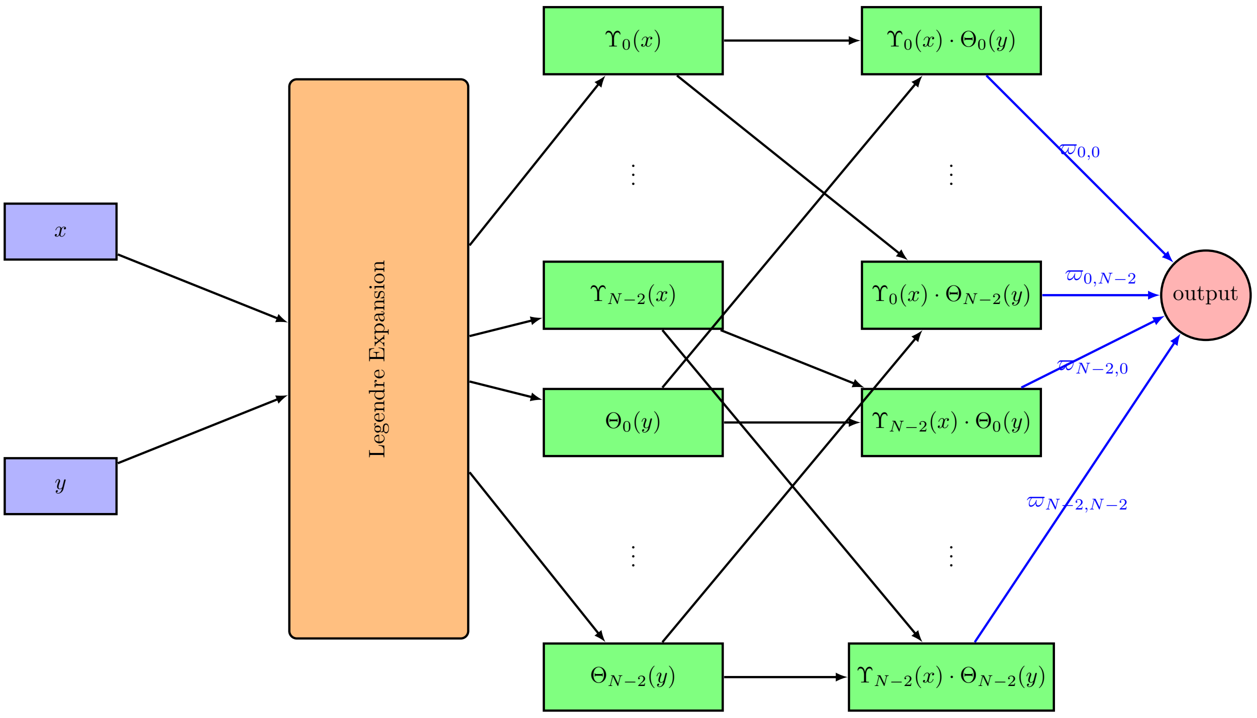

Figure 1. Schematic structure of a single-layer Legendre neural network.

The residual loss function for the LSNN is formulated as:

$$\mathcal{L}(\Phi; \mathcal{T}) = \frac{1}{|\mathcal{T}|} \sum_{\boldsymbol{x}\in\mathcal{T}} \left\| \mathcal{G}(\boldsymbol{x}, \hat{v}(\boldsymbol{x}; \Phi), \nabla\hat{v}(\boldsymbol{x}; \Phi), \nabla^2\hat{v}(\boldsymbol{x}; \Phi), \ldots) \right\|_2^2, \tag{2.15}$$

where Φ represents the trainable parameters, and $\mathcal{T}$ is the set of residual points sampled at the LGL points. By minimizing this loss, the trainable parameters Φ are updated to approximate the solution $v(\boldsymbol{x})$. Automatic differentiation (AD) is used to compute derivatives of the neural network output $\hat{v}(x; \Phi)$, allowing the evaluation of all differential terms in the governing equations $\mathcal{G}$. For a given output $\hat{v}(x; \Phi)$, AD is used to compute [49].

$$\frac{\partial^p \hat{v}(x; \Phi)}{\partial x^p}, \quad p = 1, 2, \ldots, n, \tag{2.16}$$

which eliminates the need for manual differentiation or finite difference approximations.

In order to minimize the loss and ensure convergence to the desired solution, an advanced optimizer based on the Adam optimizer is used, which combines momentum and adaptive learning rates for efficient training and updates the parameters by [50].

$$\Phi^{(k+1)} = \Phi^{(k)} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t, \tag{2.17}$$

where $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected moment estimates of the gradients. This procedure is summarized in Algorithm 1.

Algorithm 1 Pseudocode for solving a rectangular geometry problem with an LSNN

```
1: Input governing equation $G(x, v, \nabla v, \nabla^2 v)$, boundary conditions and rectangular domain
2: if boundary conditions are non-homogeneous then
3:     Define transformation $W(x)$ to satisfy the boundary conditions
4:     Represent solution as $v(x) = W(x) - \hat{v}(x)$, where $\hat{v}(x)$ satisfies the homogeneous conditions
5: end if
6: Compute the Legendre polynomial coefficients $\gamma_k$ and $\delta_k$ based on boundary conditions
7: Construct the basis functions $\Upsilon_k(x)$ using Legendre polynomials and the coefficients $\gamma_k, \delta_k$
8: Sample the residual points $T$ using LGL points
9: Compute the derivatives $\nabla\hat{v}(x)$ and $\nabla^2\hat{v}(x)$ using AD
10: Define the loss function:

        $L(\Phi; T)$ = SUM over all points in $T$ of squared residuals of $G(x, \hat{v}, \nabla\hat{v}, \nabla^2\hat{v})$

11: Initialize the neural network parameters $\Phi$
12: while the loss function has not converged do
13:     Update the parameters $\Phi$ using the Adam optimizer
14: end while
15: Output The trained parameters $\Phi$ and the approximate solution $\hat{v}(x; \Phi)$
```

2.2. *LSNNs for complex geometry*

For complex geometries, the LSNN employs coordinate transformations to map irregular domains into rectangular ones. Consider a domain described in polar coordinates:

$$f_{\min}(\theta) \le r \le f_{\max}(\theta), \quad 0 \le \theta < 2\pi. \tag{2.18}$$

The mapping is defined as:

$$z = 2\frac{r - f_{\min}(\theta)}{f_{\max}(\theta) - f_{\min}(\theta)} - 1, \quad \theta = \theta. \tag{2.19}$$

This transforms the original domain to:-

$$-1 \le z \le 1, \quad 0 \le \theta < 2\pi. \tag{2.20}$$

LGL points are used in the radial direction z, while uniform or Fourier sampling is applied in the angular direction θ. This approach ensures spectral-level resolution in the radial direction while preserving periodicity in the angular direction. Agian, AD is used for computing mixed partial derivatives with respect to the transformed variables. For the neural network output $\hat{v}(z, \theta; \Phi)$, AD computes:

$$\frac{\partial^p \hat{v}(z, \theta; \Phi)}{\partial z^p}, \quad \frac{\partial^q \hat{v}(z, \theta; \Phi)}{\partial \theta^q}, \quad p, q = 1, 2, \dots, \tag{2.21}$$

which ensures efficient handling of nonlinear terms and mixed derivatives arising from transformations. Algorithm 2, summarizes this procedure.

Algorithm 2 Pseudocode for solving complex geometry problem with LSNN

```
1: **Input** governing equation $G(x, v, \nabla v, \nabla^2 v)$ and the complex domain
2: Transform the complex domain to a rectangular domain:
3:     **for** polar coordinates:
4:         Map $r$ to $z = 2 \cdot \frac{r - r_{\min}}{r_{\max} - r_{\min}} - 1$
5:         Map $\theta$ to $\theta$ without change
6:         Update governing equation for transformed variables
7: Sample the residual points:
8:     Use LGL points for radial direction $z$
9:     Use Fourier sampling for the angular direction $\theta$
10: Construct basis functions:
11:     Define the Legendre polynomial basis for the radial direction $z$
12:     Define the Fourier basis forthe angular direction $\theta$
13: Compute the mixed derivatives AD
14: Define the loss function:

    $L(\Phi; T)$ = SUM over all points in $T$ of the squared residuals of the transformed $G(z, \theta, \hat{v}, \nabla\hat{v}, \nabla^2\hat{v})$

15: Initialize neural network parameters $\Phi$
16: **while** loss function has not converged **do**
17:     Update parameters $\Phi$ using the Adam optimizer
18: **end while**
19: **Output** The trained parameters $\Phi$ and the approximate solution $v(z, \theta)$
```

3. Error analysis

Consider the model in Eq (2.1), subject to the BCs given in Eq (2.2). The total error E in can be expressed as:

$$E := ||\hat{v} - v|| \leq ||\hat{v} - v_N|| + ||v_N - v||, \tag{3.1}$$

where $\hat{v}$ denotes the neural network solution, v_N corresponds to an approximation using the Legendre polynomial basis, and v is the exact solution.

The term $||\hat{v} - v_N||$ accounts for the error introduced by the neural network optimization process, while $||v_N - v||$ is the approximation error inherent in the Legendre polynomial representation.

For a one-dimensional boundary value problem of the form:

$$-p(x)v''(x) + q(x)v'(x) + r(x)v(x) = f(x), \quad x \in (-1, 1), \tag{3.2}$$

with the boundary conditions:

$$v(-1) = v(1) = 0, \tag{3.3}$$

the solution $v_N(x)$ satisfies the variational formulation:

$$\begin{aligned}\int_{-1}^{1} \Big[\quad p(x)v_N''(x) + q(x)v_N'(x) + r(x)v_N(x)\Big] w(x)\, dx \\ = \int_{-1}^{1} f(x)w(x)\, dx, \quad \forall w(x) \in \mathcal{V}_N.\end{aligned} \tag{3.4}$$

Let $v \in H^m(I)$ and $f \in H^k(I)$, where $H^m(I)$ is the Sobolev space. The error bounds for the Legendre spectral approximation are:

$$\|v - v_N\|_1 \lesssim N^{1-m}\|\partial_x^m v\| + N^{-k}\|\partial_x^k f\|, \quad m, k \geq 1. \tag{3.5}$$

$$\|v - v_N\|_2 \lesssim N^{-m}\|\partial_x^m v\| + N^{-k}\|\partial_x^k f\|, \quad m, k \geq 1. \tag{3.6}$$

In Eq (3.5), $\|\cdot\|_1$ refers to the H^1-norm, which measures the error in both the function v and its first derivative and is defined by:

$$\|v\|_{H^1} = \left(\|v\|_{L^2}^2 + \|\partial_x v\|_{L^2}^2\right)^{1/2}.$$

The term $N^{1-m}\|\partial_x^m v\|$ indicates that the approximation error decreases with increasing smoothness m of the solution v, while $N^{-k}\|\partial_x^k f\|$ accounts for the smoothness k of the source term f, which demonstrates that both the solution v and the source term f significantly influence the overall error in the H^1-norm.

In Eq (3.6), $\|\cdot\|_2$ refers to the L^2-norm, defined as:

$$\|v\|_{L^2} = \left(\int_I |v(x)|^2\, dx\right)^{1/2}.$$

In this equation, $N^{-m}\|\partial_x^m v\|$ represents the contribution of the smoothness m of v to the error, which decays faster than the H^1-norm, as derivatives are not considered. The term $N^{-k}\|\partial_x^k f\|$ reflects the influence of the regularity k of the source term f on the approximate error. The constants in both Eqs (3.5) and (3.6) depend on the Sobolev norms of v and f. The H^1-norm includes derivative contributions, which may lead to a slower convergence rate compared with the L^2-norm, which only accounts for the function values.

4. Numerical examples

To confirm the exponential rate of convergence, in this section a number of numerical examples taken from [34]. In our computations, we use a PC with a 12th Gen Intel(R) Core(TM) i7-1255U processor, and 16 GB RAM. The accuracy of the LSNN method is evaluated using the following three standard error metrics:

$$\begin{aligned}
||E||_{\text{absolute-}L^\infty} &= \max_{i=1,\dots,K} \left|\hat{v}(\boldsymbol{x}_i) - v^*(\boldsymbol{x}_i)\right|, \\
||E||_{\text{absolute-}L^2} &= \sqrt{\frac{\sum_{i=1}^{K} \left|\hat{v}(\boldsymbol{x}_i)-v^*(\boldsymbol{x}_i)\right|^2}{K}}, \\
||E||_{\text{relative-}L^2} &= \frac{\sqrt{\sum_{i=1}^{K} \left|\hat{v}(\boldsymbol{x}_i)-v^*(\boldsymbol{x}_i)\right|^2}}{\sqrt{\sum_{i=1}^{K} \left|v^*(\boldsymbol{x}_i)\right|^2}}.
\end{aligned} \tag{4.1}$$

Example 4.1. We consider the following second-order differential equation:

$$v''(x) + xv'(x) - v(x) = (24 + 5x)e^{5x} + (2 + 2x^2)\cos(x^2) - (4x^2 + 1)\sin(x^2), \quad x \in [-1, 1], \tag{4.2}$$

subject to the Robin boundary conditions:

$$v(-1) - v'(-1) = -4e^{-5} + \sin(1) + 2\cos(1), \quad v(1) + v'(1) = 6e^5 + \sin(1) + 2\cos(1). \tag{4.3}$$

The exact solution of this problem is given as:

$$v(x) = e^{5x} + \sin(x^2). \tag{4.4}$$

To solve this problem using the LSNN approach, we use different values of $N = 12, 14, 16, 18, 20, 22$ to construct the networks. Each network is trained for $300 \times N$ epochs. The initial learning rate is set to 0.1 and dynamically adjusted using a scheduler that reduces it by a factor of 0.7 if the loss function does not decrease for 600 consecutive iterations. The computational domain is uniformly sampled with 501 test points within the interval $[-1, 1]$. The results demonstrate that as N increases, the solution error decreases rapidly. This highlights the efficiency and accuracy of the LSNN method compared with traditional approaches. The comparison between the exact and LSNN solution is shown in Figure 2. The errors obtained for various N values confirm the exponential convergence characteristic of the LSNN, showcasing its ability to achieve high precision even with relatively small networks, as shown in Figure 3.

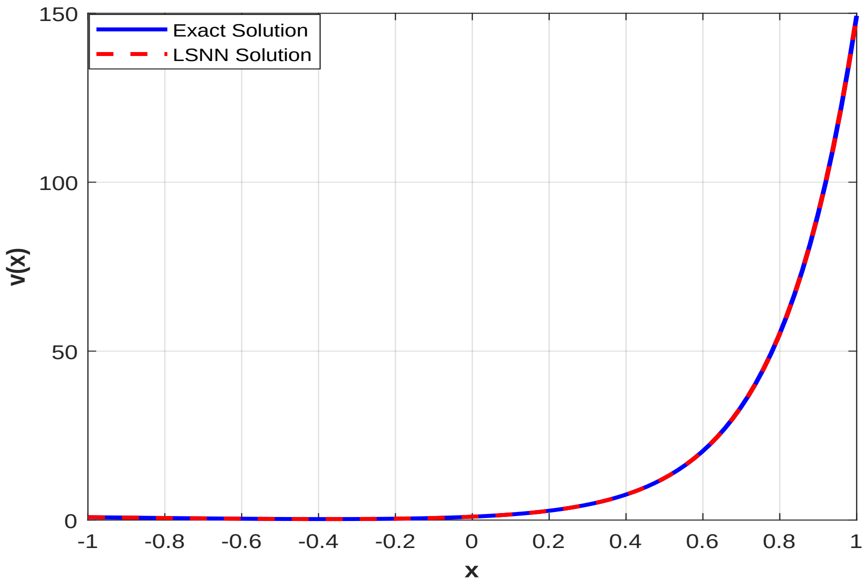

Figure 2. Example 4.1: Exact vs. LSNN solution for $N = 22$ collocation points.

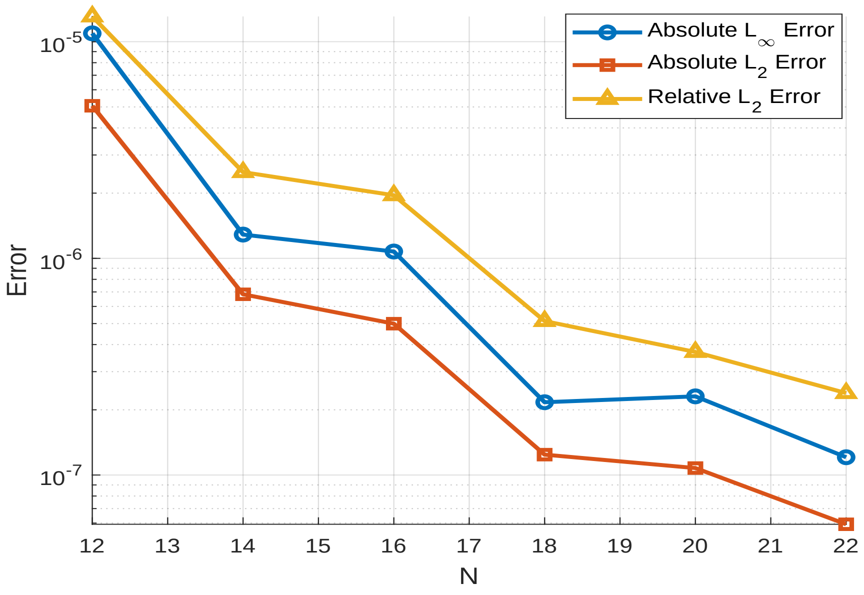

Figure 3. Example 4.1: Error behavior at different collocation points.

Example 4.2. Consider the two-dimensional Poisson equation defined as:

$$-\Delta v(x, y) = g(x, y), \quad (x, y) \in \Omega = [-1, 1]^2, \tag{4.5}$$

where the source term $g(x, y)$ is determined by the exact solution:

$$v(x, y) = e^{-x} \sin(\pi y). \tag{4.6}$$

The boundary conditions are Dirichlet and given as:

$$\phi_0(y) = v(-1, y), \quad \phi_1(y) = v(1, y), \tag{4.7}$$

$$\psi_0(x) = v(x, -1), \quad \psi_1(x) = v(x, 1), \tag{4.8}$$

where these conditions satisfy the following continuity relations:

$$\phi_0(-1) = v(-1, -1) = \psi_0(-1), \quad \phi_0(1) = v(-1, 1) = \psi_1(-1), \tag{4.9}$$

$$\phi_1(-1) = v(1, -1) = \psi_0(1), \quad \phi_1(1) = v(1, 1) = \psi_1(1). \tag{4.10}$$

The trial solution is constructed as:

$$\chi(x, y) = A(x, y) + \tilde{v}(x, y; \theta), \tag{4.11}$$

where $\tilde{v}(x, y; \theta)$ represents the output from the LSNN and $A(x, y)$ is a function that satisfies the nonhomogeneous boundary conditions:

$$\begin{aligned} A(x, y) = &\left(\frac{1-x}{2}\phi_0(y) + \frac{1+x}{2}\phi_1(y)\right) + \left(\frac{1-y}{2}\left[\psi_0(x) - \left(\frac{1-x}{2}\psi_0(-1) + \frac{1+x}{2}\psi_0(1)\right)\right]\right) \\ &+ \left(\frac{1+y}{2}\left[\psi_1(x) - \left(\frac{1-x}{2}\psi_1(-1) + \frac{1+x}{2}\psi_1(1)\right)\right]\right). \end{aligned} \tag{4.12}$$

The Adam optimizer is used for training with an initial learning rate of 0.1 and a maximum of 2000 iterations. A grid of 300 equally spaced points in both x and y directions is used to evaluate the solution. Figure 4 provides a comparison of the LSNN and the exact solution. The numerical experiments reveal that the LSNN approach achieves high accuracy, with the errors decreasing significantly as the network complexity N increases, as shown in Table 1.

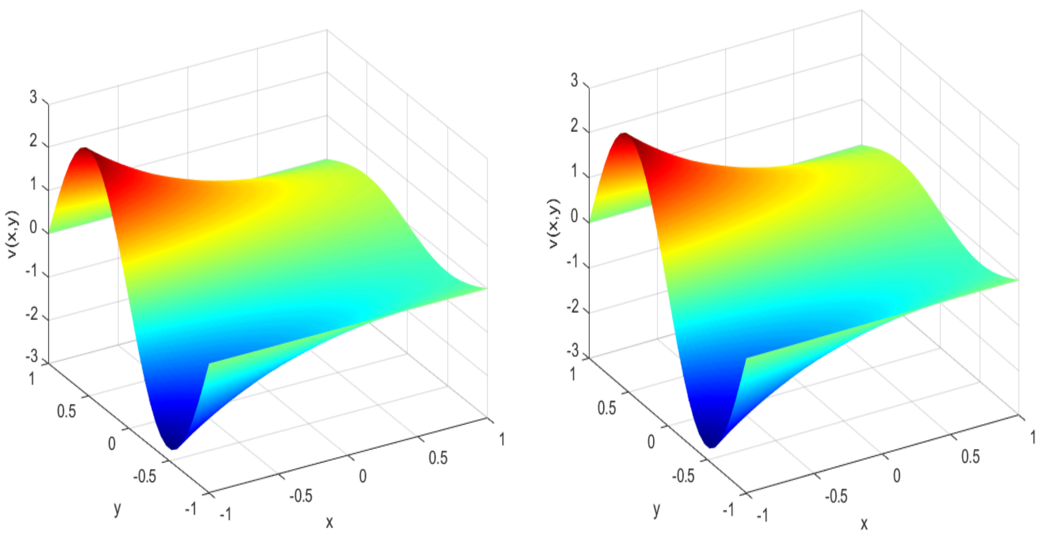

Figure 4. Example 4.2: Approximate (left) vs. exact solution (right) at $N = 30$ collocation points.

Table 1. Example 4.2: Error behavior of approximate vs. exact solutions.

N	Absolute L_∞ error	Absolute L_2 error	Relative L_2 error
6	2.2331×10^{-5}	9.2731×10^{-6}	1.0175×10^{-5}
8	5.2232×10^{-6}	2.1809×10^{-6}	2.3697×10^{-6}
10	2.9164×10^{-6}	1.0119×10^{-6}	1.0939×10^{-6}
12	1.5641×10^{-6}	4.2737×10^{-7}	4.6047×10^{-7}
15	4.6069×10^{-7}	1.9532×10^{-7}	2.0976×10^{-7}
20	2.0294×10^{-7}	6.4067×10^{-8}	6.8584×10^{-8}

Example 4.3. Consider the following two-dimensional (2D) Helmholtz equation on an annular domain:

$$\Delta v(x,y) + v(x,y) = g(x,y), \quad (x,y) \in \Omega = \{(r,\theta) \mid 0.5 \le r \le 1,\ 0 \le \theta < 2\pi\}, \tag{4.13}$$

where the source term $g(x,y)$ and the Dirichlet boundary conditions are determined by the exact solution:

$$v(x,y) = e^x \sin(\pi y). \tag{4.14}$$

The annular domain is transformed into a rectangular parameter space using the coordinate mapping:

$$x = \frac{s+3}{4}\cos((t+1)\pi), \quad y = \frac{s+3}{4}\sin((t+1)\pi), \tag{4.15}$$

where $s, t \in [-1,1]$. In the transformed domain, the Helmholtz equation becomes:

$$\frac{1}{s+3}\frac{\partial}{\partial s}\left((s+3)\frac{\partial v}{\partial s}\right) + \frac{1}{(s+3)^2}\frac{\partial^2 v}{\partial t^2} + v = g\left(\frac{s+3}{4}\cos((t+1)\pi), \frac{s+3}{4}\sin((t+1)\pi)\right). \tag{4.16}$$

In the s-direction, Legendre basis functions are used, with sampling points chosen as LGL nodes. For the t-direction, due to periodicity, trigonometric basis functions are employed, with uniform sampling over the interval $[-1,1]$. The loss function is defined as:

$$\mathcal{L}(\theta;\mathcal{T}) = \frac{1}{|\mathcal{T}|}\sum_{x\in\mathcal{T}}\left|\frac{1}{s+3}\frac{\partial}{\partial s}\left((s+3)\frac{\partial v}{\partial s}\right) + \frac{1}{(s+3)^2}\frac{\partial^2 v}{\partial t^2} + v - g\right|^2, \tag{4.17}$$

where $\mathcal{T}$ is the training set of sampled points. The initial learning rate is set to 0.01 and dynamically adjusted by reducing it by a factor of 0.6 if the loss does not decrease after 500 iterations. Using an LSNN network with $N = 20$, the training converges after 2000 iterations. The results are illustrated in Table 2, which shows the error behaviors between the exact solution and numerical solution at different collocation points. Figure 5 illustrates the LSNN and the exact solution.

To compare the efficiency and accuracy of the LSNN with traditional methods, we also applied the PINNs approach. The PINNs' setup included a network with four hidden layers, each with 128 neurons, and the same learning rate of 0.001. After 2000 iterations, the convergence of PINNs' solution with that of the LSNN is shown in Figure 6. This experiment highlights the ability of LSNNs to solve Helmholtz equations efficiently on non-standard domains, achieving high accuracy with significantly reduced computational effort.

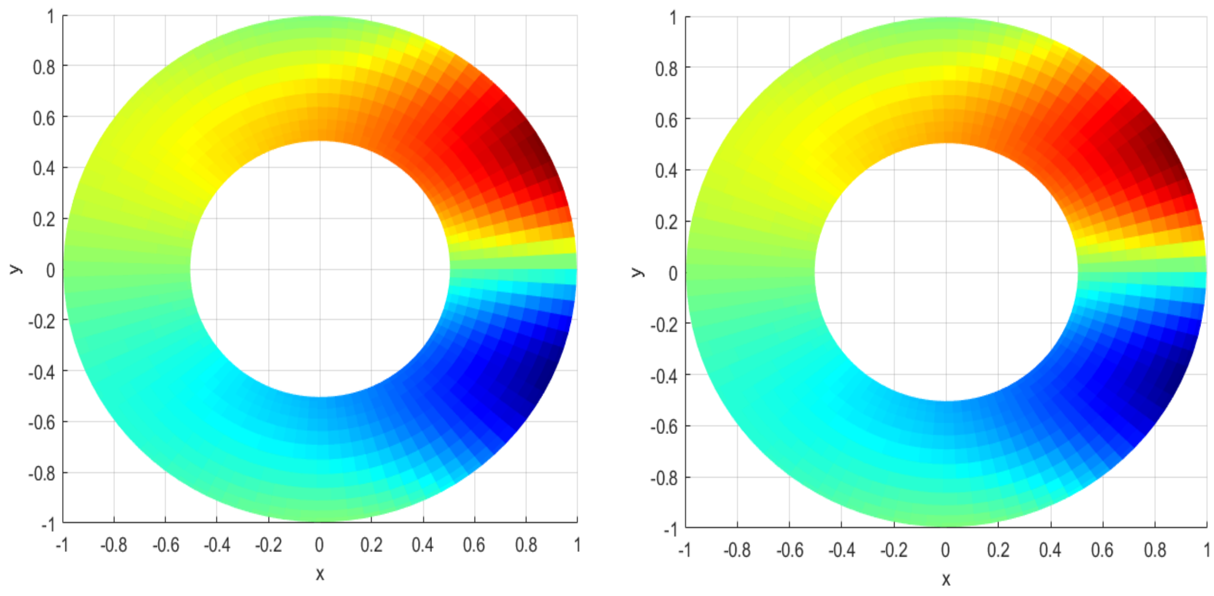

Figure 5. Example 4.3: Approximate (left) vs. exact solution (right) at $N = 30$ collocation points.

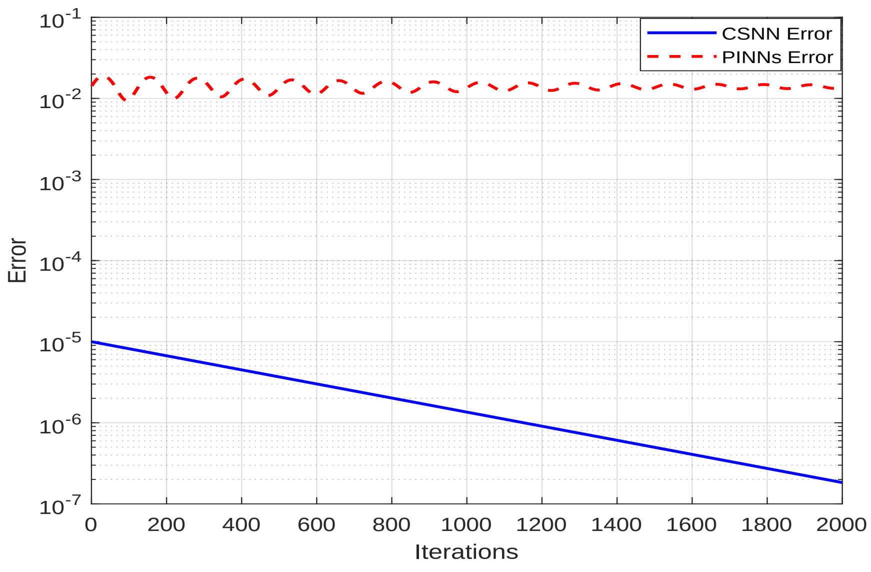

Figure 6. Example 4.3: Error convergence comparison of LSNN vs PINNs.

Table 2. Example 4.3: Error behavior of approximate vs. exact solutions.

N	Absolute L_∞ error	Absolute L_2 error	Relative L_2 error
6	8.33×10^{-4}	2.40×10^{-4}	7.12×10^{-6}
8	6.94×10^{-4}	2.08×10^{-4}	5.34×10^{-6}
10	7.80×10^{-4}	1.96×10^{-4}	4.51×10^{-6}
12	6.93×10^{-4}	1.79×10^{-4}	3.76×10^{-6}
14	6.09×10^{-4}	1.62×10^{-4}	3.15×10^{-6}
16	6.00×10^{-4}	1.49×10^{-4}	2.72×10^{-6}
18	4.85×10^{-4}	1.43×10^{-4}	2.46×10^{-6}
20	5.98×10^{-4}	1.35×10^{-4}	2.19×10^{-6}

Example 4.4. In this example, we address the Poisson equation on a circular domain:

$$-\Delta v(x,y) = g(x,y), \quad (x,y) \in \Omega = \{(r,\theta) \mid 0 \le r \le 1,\ 0 \le \theta < 2\pi\}, \tag{4.18}$$

where the source term $g(x,y)$ and the Dirichlet boundary conditions are determined by the exact solution:

$$v(x,y) = e^x \cos(\pi y). \tag{4.19}$$

To simplify the problem, the circular domain is mapped to a rectangular parameter space. Using polar coordinates, the transformation is defined as:

$$x = r\cos(\theta), \quad y = r\sin(\theta), \tag{4.20}$$

where $r \in [0,1]$ and $\theta \in [0,2\pi)$. In this transformed domain, the Poisson equation takes the following form:

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial v}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 v}{\partial \theta^2} = g(r\cos(\theta), r\sin(\theta)). \tag{4.21}$$

The circular domain introduces an additional challenge: the value at the center of the circle ($r = 0$) is undetermined. To address this, an auxiliary single-parameter neural network $\hat{v}_0$ is introduced to approximate the value at the center. This parameter is optimized alongside the main LSNN. The loss function is defined as:

$$\mathcal{L}(\theta;\mathcal{T}) = \frac{1}{|\mathcal{T}|}\sum_{x\in\mathcal{T}}\left|\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial v}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 v}{\partial \theta^2} - g\right|^2, \tag{4.22}$$

where $\mathcal{T}$ is the set of training points. Using the same parameter values as in Example 4.3, Figure 7 provides a comparison of the LSNN and true solution, while Figure 8 shows that error behavior for different collocation points.

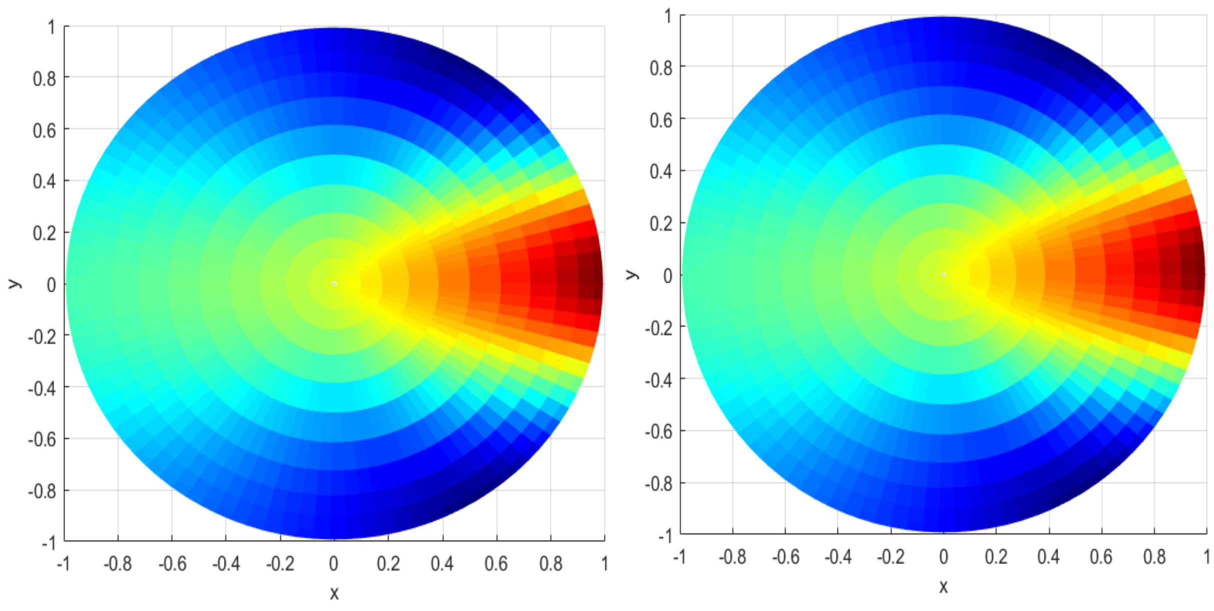

Figure 7. Example 4.4: Approximate (left) vs. exact solution (right) at $N = 30$ collocation points.

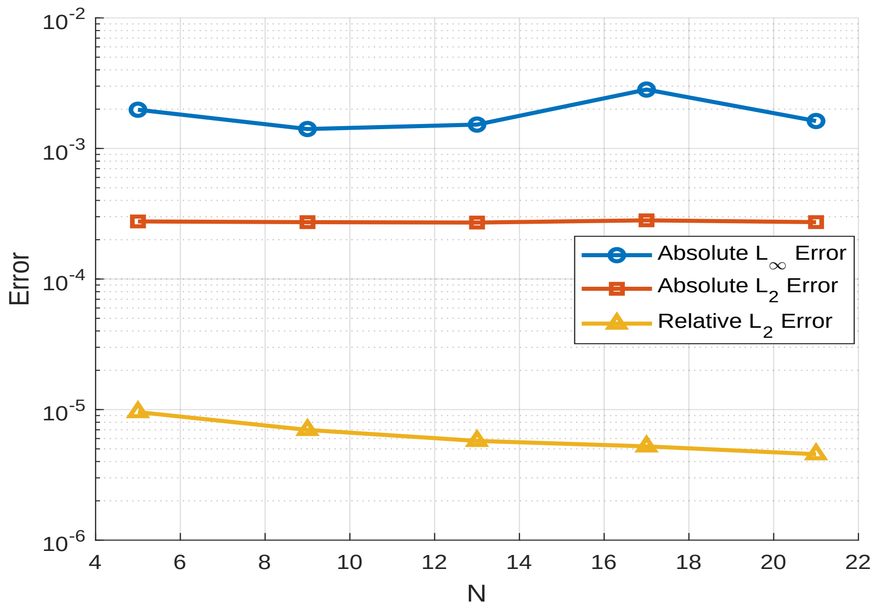

Figure 8. Example 4.4: Error behavior at different collocation points.

The success of the LSNN method on this domain depends significantly on the learning rates for both the main and auxiliary networks. Improper learning rate settings can result in slow convergence or failure to reach the global minimum. To mitigate this issue, periodic corrections to the auxiliary network parameter $\hat{v}_0$ are applied, ensuring better alignment with the true solution.

Example 4.5. Consider a three-dimensional (3D) elliptic PDE with variable coefficients:

$$\nabla \cdot (\sigma(x,y,z)\nabla v) - \kappa(x,y,z)v = g(x,y,z), \quad (x,y,z) \in \Omega, \tag{4.23}$$

where the coefficients are given as:

$$\sigma(x,y,z) = \cos(\pi(x+y+z)) + 2, \quad \kappa(x,y,z) = \sin(\pi(x+y+z)) + 2. \tag{4.24}$$

The source term $g(x,y,z)$ and Dirichlet boundary conditions are determined by the exact solution:

$$v(x,y,z) = e^{\frac{2}{3}x+\frac{2}{3}y+\frac{1}{3}z}\sin(\pi(x+y+z)). \tag{4.25}$$

The computational domain is a rolled-up cylindrical shell defined as:

$$\Omega = \{(x,y,z) \mid r \in [0.5,1],\ \theta \in [0,2\pi),\ z \in [-1,1]\}, \tag{4.26}$$

where r, θ are the cylindrical coordinates. The domain is mapped to a rectangular region using the coordinate transformation:

$$x = r\cos(\theta), \quad y = r\sin(\theta), \tag{4.27}$$

where $r \in [0.5,1]$, $\theta \in [0,2\pi)$, and $z \in [-1,1]$. In the transformed domain, the elliptic operator becomes:

$$\nabla \cdot (\sigma\nabla v) = \frac{1}{r}\frac{\partial}{\partial r}\left(r\sigma\frac{\partial v}{\partial r}\right) + \frac{1}{r^2}\frac{\partial}{\partial \theta}\left(\sigma\frac{\partial v}{\partial \theta}\right) + \frac{\partial}{\partial z}\left(\sigma\frac{\partial v}{\partial z}\right). \tag{4.28}$$

In the r- and z-directions, Legendre basis functions are used with LGT points for sampling. In the θ-direction, due to its periodicity, trigonometric basis functions are employed with uniform sampling. The loss function is defined as:

$$\mathcal{L}(\theta; \mathcal{T}) = \frac{1}{|\mathcal{T}|}\sum_{x\in\mathcal{T}} |\nabla \cdot (\sigma\nabla v) - \kappa v - g|^2, \tag{4.29}$$

where $\mathcal{T}$ represents the training points. Again keeping the same parameter values as in the previous example, the true and computed solution is plotted in Figure 9. The error behaviors for different norms and at different collocation points are shown in Figure 10, which confirms the exponential convergence of our proposed scheme.

The LSNN method demonstrates excellent performance in solving this 3D problem, even for irregular geometries like cylindrical shells. The method achieves high accuracy and rapid convergence with modest computational resources, highlighting its robustness for variable-coefficient PDEs in complex domains.

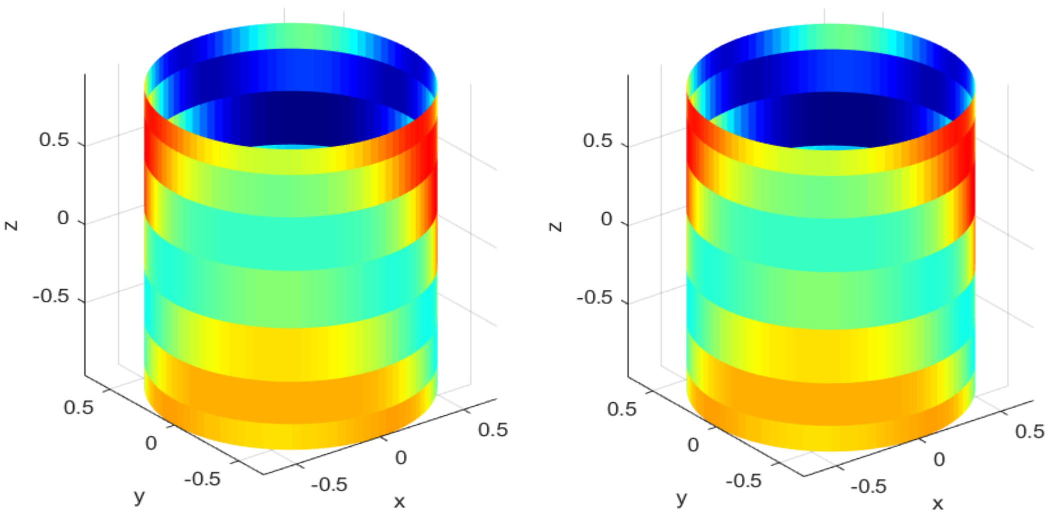

Figure 9. Example 4.5: Approximate (left) vs. exact solution (right) at $N = 30$ collocation points.

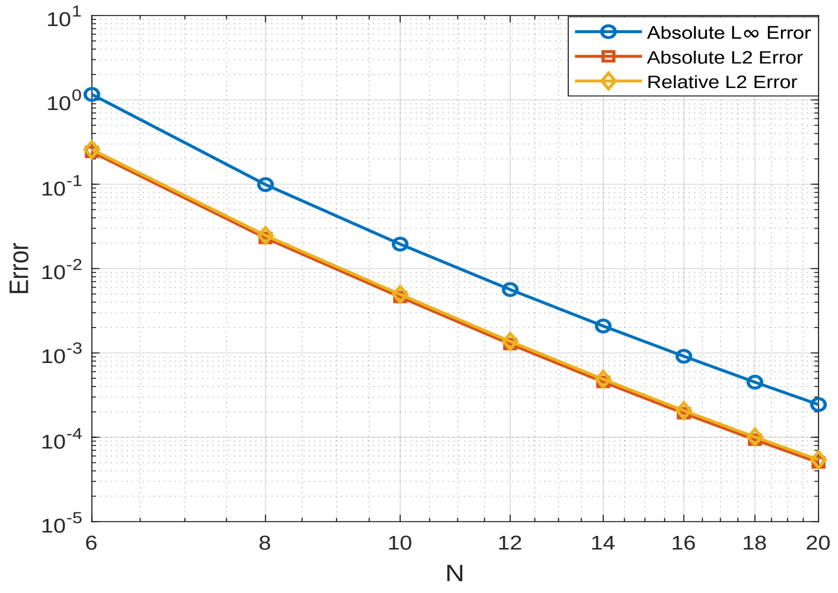

Figure 10. Example 4.5: Error behavior at different collocation points.

5. Conclusions

In this work, we demonstrate the ability of LSNNs to handle the problems of regular as well as irregular domains. We show, through a number of numerical examples that our proposed scheme is more stable and computationally efficient compared with other neural network techniques, e.g., PINNs. In particular, our scheme overcomes issues related to training instability and error oscillation and achieves an exponential rate of convergence. Our numerical examples, especially those related to complex geometries of annular and cylindrical shells, further enhance its potential to deal with real-world problems across various scientific disciplines. Moreover, the network structure is made simple by eliminating the presence of hidden layers using a Legendre-based neuron layout that improves training efficacy without compromising the solution's accuracy. The present study highlights the effectiveness of the LSNN method; however, its convergence may be impacted when using a large number of nodes due to numerical instabilities and optimization challenges. Future work could focus on adaptive strategies and improved optimization techniques to address these limitations and extend it to time-dependent and nonlinear equations.

Use of AI tools declaration

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

Acknowledgments

This work was supported by the Deanship of Scientific Research, Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Saudi Arabia [Grant No. KFU250330].

Conflict of interest

The author declares there are no conflicts of interest.

References

1. Y. Pinchover, J. Rubinstein, *An Introduction to Partial Differential Equations*, Cambridge University Press, 2005. https://doi.org/10.1017/CBO9780511801228
2. J. Douglas, B. F. Jones, On predictor-corrector methods for nonlinear parabolic differential equations, *J. Soc. Ind. Appl. Math.*, **11** (1963), 195–204. https://doi.org/10.1137/0111015
3. A. Wambecq, Rational Runge-Kutta methods for solving systems of ordinary differential equations, *Computing*, **20** (1978), 333–342. https://doi.org/10.1007/BF02252381
4. J. N. Reddy, *An Introduction to the Finite Element Method*, McGraw-Hill, 1993.
5. R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-state and Time-dependent Problems*, SIAM, 2007. https://doi.org/10.1137/1.9780898717839
6. I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, *IEEE Trans. Neural Networks*, **9** (1998), 987–1000. https://doi.org/10.1109/72.712178
7. S. Mall, S. Chakraverty, Application of Legendre neural network for solving ordinary differential equations, *Appl. Soft Comput.*, **43** (2016), 347–356. https://doi.org/10.1016/j.asoc.2015.10.069
8. T. T. Dufera, Deep neural network for system of ordinary differential equations: Vectorized algorithm and simulation, *Mach. Learn. Appl.*, **5** (2021), 100058. https://doi.org/10.1016/j.mlwa.2021.100058

9. J. A. Rivera, J. M. Taylor, A. J. Omella, D. Pardo, On quadrature rules for solving partial differential equations using neural networks, *Comput. Methods Appl. Mech. Eng.*, **393** (2022), 114710. https://doi.org/10.1016/j.cma.2022.114710

10. L. S. Tan, Z. Zainuddin, P. Ong, Wavelet neural networks based solutions for elliptic partial differential equations with improved butterfly optimization algorithm training, *Appl. Soft Comput.*, **95** (2020), 106518. https://doi.org/10.1016/j.asoc.2020.106518

11. Z. Sabir, S. A. Bhat, M. A. Z. Raja, S. E. Alhazmi, A swarming neural network computing approach to solve the Zika virus model, *Eng. Appl. Artif. Intell.*, **126** (2023), 106924. https://doi.org/10.1016/j.engappai.2023.106924

12. 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

13. E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnston, D. Mortari, Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations, *Neurocomputing*, **457** (2021), 334–356. https://doi.org/10.1016/j.neucom.2021.06.015

14. S. Dong, Z. Li, Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations, *Comput. Methods Appl. Mech. Eng.*, **387** (2021), 114129. https://doi.org/10.1016/j.cma.2021.114129

15. S. Dong, J. Yang, On computing the hyperparameter of extreme learning machines: Algorithm and application to computational PDEs, and comparison with classical and high-order finite elements, *J. Comput. Phys.*, **463** (2022), 111290. https://doi.org/10.1016/j.jcp.2022.111290

16. G. B. Huang, Q. Y. Zhu, C. K. Siew, Extreme learning machine: Theory and applications, *Neurocomputing*, **70** (2006), 489–501. https://doi.org/10.1016/j.neucom.2005.12.126

17. V. Dwivedi, B. Srinivasan, Physics-informed extreme learning machine (PIELM) – A rapid method for the numerical solution of partial differential equations, *Neurocomputing*, **391** (2020), 96–118. https://doi.org/10.1016/j.neucom.2019.12.099

18. F. Calabrò, G. Fabiani, C. Siettos, Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients, *Comput. Methods Appl. Mech. Eng.*, **387** (2021), 114188. https://doi.org/10.1016/j.cma.2021.114188

19. S. M. Sivalingam, P. Kumar, V. Govindaraj, A Chebyshev neural network-based numerical scheme to solve distributed-order fractional differential equations, *Comput. Math. Appl.*, **164** (2024), 150–165. https://doi.org/10.1016/j.camwa.2024.04.005

20. A. Jafarian, M. Mokhtarpour, D. Baleanu, Artificial neural network approach for a class of fractional ordinary differential equations, *Neural Comput. Appl.*, **28** (2017), 765–773. https://doi.org/10.1007/s00521-015-2104-8

21. I. Ali, S. U. Khan, A dynamic competition analysis of stochastic fractional differential equation arising in finance via the pseudospectral method, *Mathematics*, **11** (2023), 1328. https://doi.org/10.3390/math11061328

22. S. U. Khan, M. Ali, I. Ali, A spectral collocation method for stochastic Volterra integro-differential equations and its error analysis, *Adv. Differ. Equ.*, **2019** (2019), 161. https://doi.org/10.1186/s13662-019-2096-2

23. I. Ali, S. U. Khan, Dynamics and simulations of stochastic COVID-19 epidemic model using Legendre spectral collocation method, *AIMS Math.*, **8** (2023), 4220–4236. https://doi.org/10.3934/math.2023210

24. S. U. Khan, I. Ali, Application of Legendre spectral-collocation method to delay differential and stochastic delay differential equations, *AIP Adv.*, **8** (2018), 035301. https://doi.org/10.1063/1.5016680

25. C. Canuto, M. Y. Hussaini, A. Quarteroni, T. A. Zang, *Spectral Methods: Fundamentals in Single Domains*, Springer, 2006. https://doi.org/10.1007/978-3-540-30726-6

26. G. Mastroianni, D. Occorsio, Optimal systems of nodes for Lagrange interpolation on bounded intervals: A survey, *J. Comput. Appl. Math.*, **134** (2001), 325–341. https://doi.org/10.1016/S0377-0427(00)00557-4

27. D. Gottlieb, S. A. Orszag, *Numerical Analysis of Spectral Methods: Theory and Applications*, SIAM, 1977. https://doi.org/10.1137/1.9781611970425

28. J. P. Boyd, *Chebyshev and Fourier Spectral Methods*, Dover Publications, 2001.

29. J. S. Hesthaven, S. Gottlieb, D. Gottlieb, *Spectral Methods for Time-dependent Problems*, Cambridge University Press, 2007. https://doi.org/10.1017/CBO9780511618352

30. C. Canuto, M. Y. Hussaini, A. Quarteroni, T. A. Zang, *Spectral Methods in Fluid Dynamics*, Springer, 2012. https://doi.org/10.1007/978-3-642-84108-8

31. J. Shen, T. Tang, L. L. Wang, *Spectral Methods: Algorithms, Analysis and Aapplications*, Springer, 2011. https://doi.org/10.1007/978-3-540-71041-7

32. N. Liu, *Theory and Applications and Legendre Polynomials and Wavelets*, Ph.D thesis, The University of Toledo, 2008.

33. S. Wang, X. Yu, P. Perdikaris, When and why PINNs fail to train: A neural tangent kernel perspective, *J. Comput. Phys.*, **449** (2022), 110768. https://doi.org/10.1016/j.jcp.2021.110768

34. P. Yin, S. Ling, W. Ying, Chebyshev spectral neural networks for solving partial differential equations, preprint, arXiv:2407.03347.

35. Y. Yang, M. Hou, H. Sun, T. Zhang, F. Weng, J. Luo, Neural network algorithm based on Legendre improved extreme learning machine for solving elliptic partial differential equations, *Soft Comput.*, **24** (2020), 1083–1096. https://doi.org/10.1007/s00500-019-03944-1

36. M. Xia, X. Li, Q. Shen, T. Chou, Learning unbounded-domain spatiotemporal differential equations using adaptive spectral methods, *J. Appl. Math. Comput.*, **70**, (2024), 4395–4421. https://doi.org/10.1007/s12190-024-02131-2

37. Y. Ye, Y. Li, H. Fan, X. Liu, H. Zhang, SLeNN-ELM: A shifted Legendre neural network method for fractional delay differential equations based on extreme learning machine, *Netw. Heterog. Media*, **18** (2023), 494–512. https://doi.org/10.3934/nhm.2023020

38. Y. Yang, M. Hou, J. Luo, A novel improved extreme learning machine algorithm in solving ordinary differential equations by Legendre neural network methods, *Adv. Differ. Equ.*, **2018** (2018), 469. https://doi.org/10.1186/s13662-018-1927-x

39. Y. Wang, S. Dong, An extreme learning machine-based method for computational PDEs in higher dimensions, *Comput. Methods Appl. Mech. Eng.*, **418** (2024), 116578. https://doi.org/10.1016/j.cma.2023.116578

40. D. Yuan, W. Liu, Y. Ge, G. Cui, L. Shi, F. Cao, Artificial neural networks for solving elliptic differential equations with boundary layer, *Math. Methods Appl. Sci.*, **45** (2022), 6583–6598. https://doi.org/10.1002/mma.8192

41. H. Liu, B. Xing, Z. Wang, L. Li, Legendre neural network method for several classes of singularly perturbed differential equations based on mapping and piecewise optimization technology, *Neural Process. Lett.*, **51** (2020), 2891–2913. https://doi.org/10.1007/s11063-020-10232-9

42. X. Li, J. Wu, X. Tai, J. Xu, Y. Wang, Solving a class of multi-scale elliptic PDEs by Fourier-based mixed physics-informed neural networks, *J. Comput. Phys.*, **508** (2024), 113012. https://doi.org/10.1016/j.jcp.2024.113012

43. S. Zhang, J. Deng, X. Li, Z. Zhao, J. Wu, W. Li, et al., Solving the one-dimensional vertical suspended sediment mixing equation with arbitrary eddy diffusivity profiles using temporal normalized physics-informed neural networks, *Phys. Fluids*, **36** (2024), 017132. https://doi.org/10.1063/5.0179223

44. X. Li, J. Deng, J. Wu, S. Zhang, W. Li, Y. Wang, Physics-informed neural networks with soft and hard boundary constraints for solving advection-diffusion equations using Fourier expansions, *Comput. Math. Appl.*, **159** (2024), 60–75. https://doi.org/10.1016/j.camwa.2024.01.021

45. X. Li, J. Wu, L. Zhang, X. Tai, Solving a class of high-order elliptic PDEs using deep neural networks based on its coupled scheme, *Mathematics*, **10** (2022), 4186. https://doi.org/10.3390/math10224186

46. X. Li, J. Wu, Y. Huang, Z. Ding, X. Tai, L. Liu, et al., Augmented physics informed extreme learning machine to solve the biharmonic equations via Fourier expansions, preprint, arXiv:2310.13947.

47. Z. Fu, W. Xu, S. Liu, Physics-informed kernel function neural networks for solving partial differential equations, *Neural Networks*, **172** (2024), 106098. https://doi.org/10.1016/j.neunet.2024.106098

48. J. Bai, G. Liu, A. Gupta, L. Alzubaidi, X. Feng, Y. Gu, Physics-informed radial basis network (PIRBN): A local approximating neural network for solving nonlinear partial differential equations, *Comput. Methods Appl. Mech. Eng.*, **415** (2023), 116290. https://doi.org/10.1016/j.cma.2023.116290

49. A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, et al., Automatic differentiation in PyTorch, in *NIPS 2017 Workshop on Autodiff*, (2017), 1–4.

50. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, preprint, arXiv:1412.6980.