

RESEARCH

Open Access



# A novel improved extreme learning machine algorithm in solving ordinary differential equations by Legendre neural network methods

Yunlei Yang<sup>1</sup>, Muzhou Hou<sup>1\*</sup>  and Jianshu Luo<sup>2</sup>

\*Correspondence:

[houluzhou@sina.com](mailto:houluzhou@sina.com)

<sup>1</sup>School of Mathematics and Statistics, Central South University, Changsha, China

Full list of author information is available at the end of the article

## Abstract

This paper develops a Legendre neural network method (LNN) for solving linear and nonlinear ordinary differential equations (ODEs), system of ordinary differential equations (SODEs), as well as classic Emden–Fowler equations. The Legendre polynomial is chosen as a basis function of hidden neurons. A single hidden layer Legendre neural network is used to eliminate the hidden layer by expanding the input pattern using Legendre polynomials. The improved extreme learning machine (IELM) algorithm is used for network weights training when solving algebraic equation systems, and several algorithm steps are summed up. Convergence was analyzed theoretically to support the proposed method. In order to demonstrate the performance of the method, various testing problems are solved by the proposed approach. A comparative study with other approaches such as conventional methods and latest research work reported in the literature are described in detail to validate the superiority of the method. Experimental results show that the proposed Legendre neural network with IELM algorithm requires fewer neurons to outperform the numerical algorithm in the latest literature in terms of accuracy and execution time.

**Keywords:** Legendre polynomial; Legendre neural network; Improved extreme learning machine; ODEs; Classic Emden–Fowler equation

## 1 Introduction

Many problems encountered in science and engineering, for example, physics, chemistry, biology, mechanics, astronomy, population, resources, economics, and so on, are related to a mathematical model in the form of differential equations. Generally, the analytical expressions of mathematical solutions for practical problems do not exist or are difficult to find. Therefore, it is necessary to study the numerical method of solving differential equations. This means calculating approximate value  $y_i$  of exact solution  $y(x_i)$  for differential equations at discrete points  $x_i, i = 0, 1, \dots$  in the solution domain.

For a long time, many numerical methods were proposed for solving ODEs [1], including single- and multi-step methods; single-step methods include Euler first order method (EM) [2], second order Runge–Kutta (R–K) method inspired by Taylor’s expansion, Suen third order R–K method (Suen-R-K3), the classic fourth-order R–K method (R-K4) [3–

5], etc. In addition, in order to obtain high accuracy, based on the numerical integration method, a lot of linear multi-step methods were proposed, such as Adams implicit formulas [6], methods based on Taylor expansion [7], prediction–correction algorithms [8], shooting methods [9], difference methods [10], etc. The numerical methods for solving boundary value problem (BVP) have many mature research results, with high calculation accuracy, but there is a problem that with increasing sample size, the execution time increases rapidly.

With the development of artificial intelligence and computer technology, more and more researchers have developed a keen interest in neural network methods. Neural networks have been used in many fields such as pattern recognition [11], graphics processing [12], risk assessment [13], control systems [14], forecasting [15–18], and classification [19], showing wide application prospects. Based on the advantages of neural network methods, the use of neural network function approximation capabilities [20–22] has led to the development of a number of adopted neural network model for solving differential equations. The neural network methods for solving differential equations mainly include the following categories: multilayer perceptron neural network [23–28], radial basis function neural network [29–31], multi-scale radial basis function neural network [32–35], cellular neural network [36, 37], finite element neural network [38–46] and wavelet neural network [28]. The main research focuses on two parts: the construction of the approximate solution and the weights training algorithm.

Approximate solutions of differential equations are often constructed by selecting different activation functions: Meade and Fernandez [47, 48] used a hard limit function as an activation function to construct the neural network model; Lagaris and Likas [23] proposed that multi-layer perceptrons can be used to construct approximate solutions; a hybrid technique for constructing the neural network was studied by Ioannis and Tsoulos [49]; Mall and Chakraverty [50] used a Legendre polynomial as an activation function to construct an approximate solution; Xu Liying, Wen Hui and Zeng Zhezhaoh [51] proposed using a triangular basis function as an activation function to construct approximate solutions for solving ODEs. Regarding the research on network weights training optimization algorithm, we mention Reidmiller and Braun [52] who proposed RPROP algorithm based on local adaptation; Lagaris and Likas [23] proposed using DE-evolutionary algorithm to train the weights in the neural network model of partial differential equations; Malek and Shekari Beidokhti [53] presented an optimization algorithm for hybrid neural network model; Rudd and Ferrari [54] analyzed the constrained integral method (CINT) combining the classical Galerkin method with the constrained BP process; Lucie and Peter [55] proposed genetic algorithms for solving a neural network model.

This paper presents a novel Legendre neural network method with improved extreme learning machine algorithm for solving several types of linear or nonlinear differential equations. Candidate solutions are expressed by using Legendre network. With the boundary conditions taken into account, the problem of solving differential equations is transformed into that of nonlinear algebraic equation systems. We call this method for training network weights the improved extreme learning machine algorithm. Convergence analysis, numerical experiments and a comparative study show the superiority of the present method to other classical methods or methods in the recent literature. We believe that the proposed method may be the first to use Legendre neural network model with IELM algorithm in solving differential equations.

The aim and motivation of the present method is to propose a new Legendre neural network with IELM algorithm to solve differential equations such as linear or nonlinear ordinary differential equations, system of ordinary differential equations, and singular initial value Emden–Fowler equations. IELM algorithm is used here for training the network weights. The advantages of the proposed approach are as follows:

- It is a single hidden layer neural network—by randomly choosing of the input layer weights, we only need to train the weights of the output layer.
- It is easy to implement and runs quickly.
- The improved extreme learning machine algorithm is an unsupervised learning algorithm, and we use no optimization technique.
- Calculation accuracy is higher than for other numerical methods presented in the recent literature.

The organization of this paper is as follows: we give a description of the problem to be solved in the next section. Section 3 talks about constructing Legendre neural network for approximating and solving ODEs. IELM algorithm for training network weights is proposed and several algorithm steps are summed up in Sect. 4. In Sect. 5, convergence analysis of the proposed Legendre network is verified. We provide many numerical results to verify the effectiveness of the algorithm and its superiority in performance in Sect. 6. Finally, in Sect. 7 we present some conclusions and directions for future research.

## 2 Description of the problem

We first introduce the general form of the following ordinary differential equations.

### 2.1 Second-order ordinary differential equations

We usually describe two-point BVP of second-order ODEs in the following form:

$$\begin{cases} y'' = f(x, y, y'), \\ y(a) = \alpha_1, \quad y(b) = \alpha_2, \end{cases} \quad a \leq x \leq b. \tag{1}$$

### 2.2 First-order system of ordinary differential equations

Let us use the following formula to represent the first-order SODE:

$$\begin{cases} y'_i = f_i(x, y_1, y_2, \dots, y_n), \\ y_i(a) = \alpha_i, \end{cases} \quad a \leq x \leq b \quad (i = 0, 1, \dots, n). \tag{2}$$

We know that first-order ODEs is a particular case of a system of ordinary differential equations (2).

### 2.3 Higher-order ODEs and higher-order SODE problem

Higher-order ODEs have the general form as below:

$$\begin{cases} y^{(n)} = f(x, y, y', y'', \dots, y^{(n-1)}), \\ y(a) = \alpha_0, \quad y'(a) = \alpha_1, \dots, y^{(n-1)}(a) = \alpha_{n-1}, \end{cases} \quad a \leq x \leq b. \tag{3}$$

If we make the transformation  $y_1 = y, y_2 = y', \dots, y_n = y^{(n-1)}$ , the higher-order ODEs change to the following SODE:

$$\begin{cases} y_1' = y_2, \\ y_2' = y_3, \\ \vdots \\ y_n' = f(x, y, y', \dots, y^{(n-1)}) = f(x, y_1, y_2, \dots, y_n), \end{cases} \tag{4}$$

where the initial conditions are  $y_1(a) = \alpha_0, y_2(a) = \alpha_1, \dots, y_n(a) = \alpha_{n-1}$ .

If for a higher-order SODE composed of two higher-order ODEs

$$\begin{cases} x^{(m)} = f(x, x', x'', \dots, x^{(m-1)}, y, y', \dots, y^{(n-1)}), \\ y^{(n)} = g(x, x', x'', \dots, x^{(m-1)}, y, y', y'', \dots, y^{(n-1)}), \\ x(a) = \alpha_0, \quad x'(a) = \alpha_1, \dots, x^{(m-1)}(a) = \alpha_{m-1}, \\ y(a) = \alpha_m, \quad y'(a) = \alpha_{m+1}, \dots, y^{(n-1)}(a) = \alpha_{m+n-1}, \end{cases} \quad a \leq x \leq b \tag{5}$$

we select state variables  $y_1 = x, y_2 = x', \dots, y_m = x^{(m-1)}, y_{m+1} = y, y_{m+2} = y', \dots, y_{m+n} = y^{(n-1)}$ , then the above system of higher-order ODEs can be expressed as:

$$\begin{cases} y_1' = y_2, \\ y_2' = y_3, \\ \vdots \\ y_m' = f(t, y_1, y_2, \dots, y_m, y_{m+1}, \dots, y_{m+n}), \\ y_{m+1}' = y_{m+2}, \\ \vdots \\ y_{m+n}' = g(t, y_1, y_2, \dots, y_m, y_{m+1}, \dots, y_{m+n}), \end{cases} \quad a \leq x \leq b \tag{6}$$

with initial conditions  $y_i(a) = \alpha_{i-1}, i = 1, 2, \dots, m + n$ .

Considering the same notation as that of Jose [56], we can describe the above linear or nonlinear ordinary differential equations in the following general form:

$$\mathcal{L}\mathbf{y}(x) = \mathbf{f}(x) \quad \text{in } I \tag{7}$$

with the initial or boundary conditions

$$\mathcal{B}\mathbf{y}(x) = \boldsymbol{\alpha} \quad \text{on } \partial I, \tag{8}$$

where  $\mathcal{L}$  and  $\mathcal{B}$  are differential operators on the interval  $I$ ;  $\mathbf{y}(x)$  denotes the vector to be found,  $\mathbf{f}(x)$  is a linear or nonlinear source term, which depends on  $x, \mathbf{y}(x)$  and its derivatives;  $\boldsymbol{\alpha}$  denotes the value of  $\mathbf{y}(x)$  and its derivatives at the end points of interval  $I$ .

By established the ODEs problem, differential equations (7) and (8) can be transformed into a constrained optimization problem in the following form:

$$\text{minimize} \tag{9}$$

$$\arg \min \| \mathcal{L}y(x) - \mathbf{f}(x) \|$$

$$\text{subject to} \tag{10}$$

$$\| \mathcal{B}y(x) - \boldsymbol{\alpha} \| = 0.$$

### 3 Legendre basis function neural network for approximating and solving ODEs

#### 3.1 Legendre basis function neural networks and approximation

In this subsection, employing the recursive properties of Legendre polynomials, we will discuss construction of approximate solutions based on Legendre basis function neural network.

**Theorem 1** *Suppose that the vector  $\mathbf{P}(x)$  is defined as  $\mathbf{P}(x) = [P_0(x), P_1(x), \dots, P_{N+1}(x)]$ , in which  $P_n(x), n = 0, 1, \dots, N + 1$  is the  $n$ th order Legendre polynomial in the interval  $[0, 1]$ , and let  $\mathbf{P}'(x)$  be defined as  $\mathbf{P}'(x) = [P'_0(x), P'_1(x), \dots, P'_{N+1}(x)]$ , where  $P'_n(x), n = 0, 1, \dots, N + 1$  is the derivative of the  $n$ th order Legendre polynomial  $P_n(x)$ . Then  $\mathbf{P}'(x) = \mathbf{P}(x)\mathbf{M}$ , where  $\mathbf{M}$  is the Legendre operational matrix given by*

$$\mathbf{M} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & \dots & 0 & 1 \\ 0 & 0 & 3 & 0 & 3 & 0 & \dots & 3 & 0 \\ 0 & 0 & 0 & 5 & 0 & 5 & \dots & 0 & 5 \\ 0 & 0 & 0 & 0 & 7 & 0 & \dots & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & \dots & 0 & 9 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 2n - 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{(N+2) \times (N+2)}$$

*Proof* The derivatives of Legendre polynomials satisfy the recurrence relation

$$P'_{n+1}(x) - P'_{n-1}(x) = (2n + 1)P_n(x), \tag{11}$$

and from this property, we can easily draw the conclusion. □

**Theorem 2** *For any continuous function  $y : [a, b] \rightarrow R$ , there is a natural number  $N$ , constants  $a_n, b_n, \beta_n (n = 0, 1, \dots, N)$ , and Legendre polynomials  $P_0(x), P_1(x), \dots, P_n(x)$ , such that the Legendre neural network with  $N + 1$  neurons is given by*

$$y_{\text{LNN}}(x) = \sum_{n=0}^N \beta_n P_n(a_n x + b_n), \tag{12}$$

$y_{\text{LNN}}$  is an approximation of  $y$ , and

$$\| y(x) - y_{\text{LNN}}(x) \| = \left\| y(x) - \sum_{n=0}^N \beta_n P_n(a_n x + b_n) \right\| < \varepsilon. \tag{13}$$

### 3.2 Legendre basis function neural networks for solving ODEs

Legendre basis neural networks consist of three layers: an input layer, a hidden Legendre basis function layer and an output layer. The output of Legendre basis neural network for general differential problem described in (7) is as follows:

$$y_{LNN}(x) = \sum_{n=0}^N P_n(a_n x + b_n) \beta, \tag{14}$$

where  $a_n$  is a weight connecting input to the  $n$ th hidden node,  $b_n$  is the bias of the  $n$ th hidden node,  $\beta$  is the hidden layer-to-output layer weight vector.

Substituting the approximate solution (14) into (7) and boundary conditions (8), we can obtain an equation system of weights  $\beta$ , and the new equation system is

$$\begin{aligned} \mathcal{L}y_{LNN}(x) &= f(x) \quad \text{in } I, \\ \mathcal{B}y_{LNN}(x) &= \alpha \quad \text{on } \partial I. \end{aligned} \tag{15}$$

Using a discretization of interval  $I = \{x_i : x_i \in I, i = 0, 1, \dots, M\}$ , define  $f_i = f(x_i)$ . Then the weights  $a_n, b_n, \beta$  can be solved for from the following system of equations:

$$\begin{bmatrix} \mathcal{L}(\sum_{n=0}^N P_n(a_n x_i + b_n)) \\ \dots \\ \mathcal{B}(\sum_{n=0}^N P_n(a_n x_{\text{boundary}} + b_n)) \end{bmatrix} [\beta] = \begin{bmatrix} f_i \\ \dots \\ \alpha \end{bmatrix}. \tag{16}$$

Let us take the following SODE as an example:

$$\begin{cases} y_1' + g_1(x)y_1 + g_2(x)y_2 = f_1(x), \\ y_2' + h_1(x)y_1 + h_2(x)y_2 = f_2(x), \quad a \leq x \leq b. \\ y_1(a) = \alpha_1, \quad y_2(a) = \alpha_2, \end{cases} \tag{17}$$

Assume that the weight connecting input to the  $n$ th hidden node is 1, and the bias of the  $n$ th hidden node is 0. Then the approximate solutions  $y_{1LNN}, y_{2LNN}$  of the SODE are given as below:

$$y_{1LNN}(x) = \sum_{n=0}^N \beta_{1n} P_n(x) = P(x) \beta_1, \tag{18}$$

$$y_{2LNN}(x) = \sum_{n=0}^N \beta_{2n} P_n(x) = P(x) \beta_2, \tag{19}$$

where  $\beta_1 = [\beta_{10}, \beta_{11}, \dots, \beta_{1N}]^T$ ,  $\beta_2 = [\beta_{20}, \beta_{21}, \dots, \beta_{2N}]^T$ . By Theorem 1, we can rewrite problem (17) as:

$$\begin{cases} (P(x)M + g_1(x)P(x))\beta_1 + g_2(x)P(x)\beta_2 = f_1(x), \\ h_1(x)P(x)\beta_1 + (P(x)M + h_2(x)P(x))\beta_2 = f_2(x), \\ P(a)\beta_1 = \alpha_1, \\ P(a)\beta_2 = \alpha_2. \end{cases} \tag{20}$$

Noting that  $x_i = a + \frac{b-a}{M}i, i = 0, 1, \dots, M$ , and defining

$$\mathbf{H} = \begin{bmatrix} w_{11}(x_0), w_{12}(x_0) \\ \vdots \\ w_{11}(x_M), w_{12}(x_M) \\ w_{21}(x_0), w_{22}(x_0) \\ \vdots \\ w_{21}(x_M), w_{22}(x_M) \\ w_{31}, w_{32} \\ w_{41}, w_{42} \end{bmatrix}_{(2M+4) \times (2N+2)}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} f_1(x_0) \\ \vdots \\ f_1(x_M) \\ f_2(x_0) \\ \vdots \\ f_2(x_M) \\ \alpha_1 \\ \alpha_2 \end{bmatrix}_{(2M+4) \times 1},$$

where

$$\begin{aligned} w_{11}(x) &= P(x)M + g_1(x)P(x), & w_{12} &= g_2(x)P(x), & w_{21} &= h_1(x)P(x), \\ w_{22}(x) &= P(x)M + h_2(x)P(x), & w_{31} &= P(a), \\ w_{32} = w_{41} &= (\mathbf{0})_{(N+1) \times 1}, & w_{42} &= w_{31}, \end{aligned}$$

we can rewrite equation (20) in the form:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T}. \tag{21}$$

By solving the new system equation (21), the unknown weights of the Legendre neural network are obtained.

Generally, by using the Legendre basis function neural network, the approximate solution of ODEs can be constructed. Then substituting the true solution of the problem by the approximate solution and its derivatives, we can obtain a system equation for finding network weights; the process is shown in Fig. 1.

### 4 IELM algorithm for training the Legendre neural networks

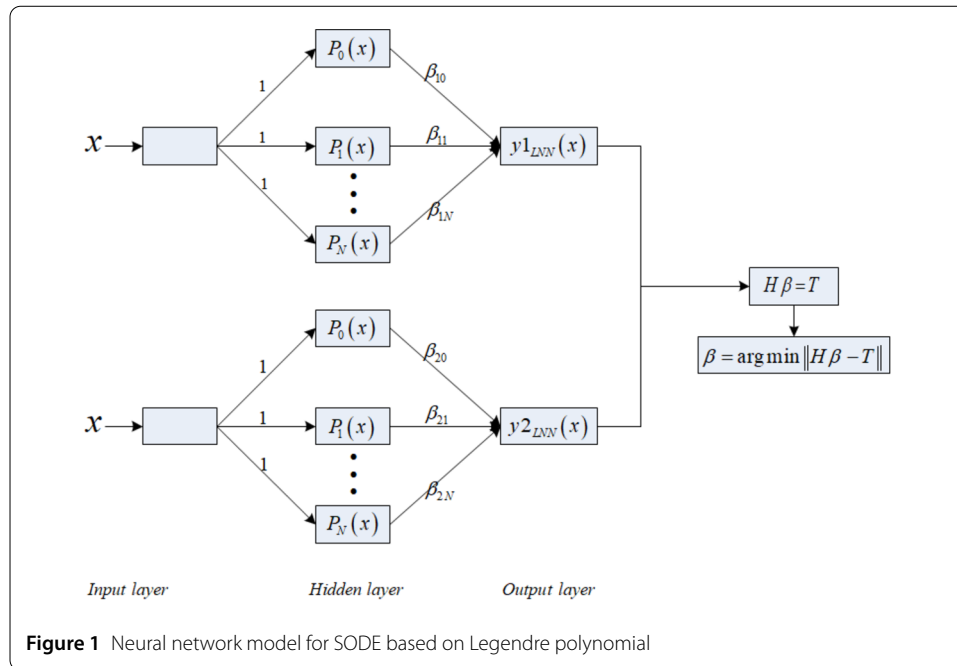
There are many numerical algorithms for solving system equation (21). In this paper, following the ELM algorithm proposed by Huang Guangbin [57], we use IELM algorithm to train the Legendre network.

**Theorem 3** *The system equation  $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$  is solvable in the following several cases:*

- (I) *If matrix  $\mathbf{H}$  is a square invertible matrix, then  $\boldsymbol{\beta} = \mathbf{H}^{-1}\mathbf{T}$ .*
- (II) *If matrix  $\mathbf{H}$  is rectangular, then  $\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T}$ , and  $\boldsymbol{\beta}$  is the minimal least-squares solution of  $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$ , that is,  $\boldsymbol{\beta} = \arg \min \|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\|$ .*
- (III) *If  $\mathbf{H}$  is a singular matrix, then  $\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T}$ , and  $\mathbf{H}^\dagger = \mathbf{H}^T(\lambda\mathbf{I} + \mathbf{H}\mathbf{H}^T)^{-1}$ ,  $\lambda$  is regularization coefficient, which can be set according to a specific instance.*

*Proof* For the proof of the theorem we refer to the related facts about the generalized inverse matrix in matrix theory [58] and the paper by Guang-Bin Huang [57].

According to Theorem 2, and as in the article of Guang-Bin Huang [57], when using extreme learning machine (ELM) algorithm to solve neural network model, that is, when



solving  $\mathbf{H}\beta = \mathbf{T}$ , the number of hidden neuron nodes must be less than or equal to the sample size, that is,  $N \leq M$ .

But by matrix analysis theory, if matrix  $\mathbf{H}$  is rectangular, there exists a  $\beta$ , such that it is the minimal least-squares solution of  $\mathbf{H}\beta = \mathbf{T}$ , that is,  $\beta = \arg \min \|\mathbf{H}\beta - \mathbf{T}\|$ . Here,  $\mathbf{H}$  is a rectangular matrix, and the number of hidden neuron nodes does not have to be less than or equal to the sample size; we call this improved algorithm for solving  $\mathbf{H}\beta = \mathbf{T}$  the improved extreme learning machine (IELM).

The steps for solving ODEs using Legendre network and IELM algorithm are as follows:

Step 1. Discretize the domain as  $a = x_0 < x_1 < \dots < x_M = b$ ,  $x_i = a + \frac{b-a}{M}i$ ,  $i = 0, 1, \dots, M$ , and construct an approximate solution by using Legendre polynomial as an activation function, that is,  $y_{LNN}(x) = \sum_{n=0}^N \beta_n P_n(x)$ ;

Step 2. At discrete points, substitute the approximate solution  $y_{LNN}(x)$  and its derivatives into the differential equation and its boundary conditions, and obtain the system equation  $\mathbf{H}\beta = \mathbf{T}$ ;

Step 3. Solve the system equation  $\mathbf{H}\beta = \mathbf{T}$  by IELM algorithm introduced in Theorem 3, and obtain the network weights  $\beta = \mathbf{H}^+ \mathbf{T}$ ,  $\beta = \arg \min \|\mathbf{H}\beta - \mathbf{T}\|$ ;

Step 4. Form the approximate solution as  $y_{LNN}(x) = \sum_{n=0}^N \beta_n P_n(x) = \mathbf{P}(x)\beta$ . □

### 5 Convergence analysis

In this section, we will verify the feasibility and convergence of the LNN method in solving differential equations by proving another theorem.

**Theorem 4** *Given a standard single layer feedforward neural network with  $n + 1$  hidden nodes and Legendre basis function  $P_i(x) : \mathbb{R} \rightarrow \mathbb{R}$ ,  $i = 0, 1, \dots, n$ , suppose that the approximate solution of one-dimensional differential equations is given by (14). If we have any  $m + 1$  distinct samples  $(\mathbf{x}, \mathbf{f})$ , for any  $a_n, b_n$  randomly chosen from any intervals of  $\mathbb{R}$ , respectively, according to any continuous probability distribution, then the hidden layer output matrix  $\mathbf{H}$  of Legendre network is invertible and  $\|\mathbf{H}\beta - \mathbf{T}\| = 0$ .*



*Proof* According to Legendre network, for any  $m + 1$  arbitrary and distinct samples  $(\mathbf{x}, \mathbf{f})$ , with  $\mathbf{x} = [x_0, \dots, x_m]^T$ ,  $\mathbf{f} = [f_0, \dots, f_m]^T$ , let us consider the  $(i + 1)$ th column of the Legendre hidden layer output matrix  $\mathbf{c}(b_i)$ ,  $\mathbf{c}(b_i) \in R^{m+1}$ , and suppose that  $b_i \in I$ , where  $I$  is an open interval of  $R$ , and

$$\mathbf{c}(b_i) = [P_i(a_i x_0 + b_i), P_i(a_i x_1 + b_i), \dots, P_i(a_i x_m + b_i)]^T, \tag{22}$$

then following the proof of Huang Guangbin [57], we can easily prove by contradiction that the vector  $\mathbf{c}$  does not belong to any subspace whose dimension less than  $m + 1$ .

We assume that  $a_i$  is generated randomly based on a continuous probability distribution, for any  $k \neq k'$ , and we have  $a_i x_k \neq a_i x_{k'}$ . Suppose that  $\mathbf{c}$  belongs to an  $m$ -dimensional subspace and vector  $\alpha$  is perpendicular to this subspace. Then we have

$$\begin{aligned} (\alpha, \mathbf{c}(b_i) - \mathbf{c}(a)) &= \alpha_0 \cdot P_i(d_0 + b_i) + \alpha_1 \cdot P_i(d_1 + b_i) + \dots \\ &+ \alpha_m \cdot P_i(d_m + b_i) - c = 0, \end{aligned} \tag{23}$$

where  $d_k = a_i x_k, k = 0, 1, \dots, m$  and  $c = \alpha \cdot \mathbf{c}(a)$ , so we may as well assume  $\alpha_m \neq 0$ , and (23) can be rewritten as

$$P_i(d_m + b_i) = - \sum_{k=0}^{m-1} \gamma_k P_i(d_k + b_i) + c/\alpha_m, \tag{24}$$

where  $\gamma_k = \alpha_k/\alpha_m, k = 0, \dots, m - 1$ , by the infinite differentiability of the function on the left-hand side of (24). Calculating the derivatives of  $b_i$  on both sides, we obtain

$$P_i^{(l)}(d_m + b_i) = - \sum_{k=0}^{m-1} \gamma_k P_i^{(l)}(d_k + b_i), \quad l = 1, \dots, m, m + 1, \dots, \tag{25}$$

where the number of coefficients  $\gamma_k$  is less than the number of equations  $l$ , which produces a contradiction, and so  $\mathbf{c}$  does not belong to any subspace whose dimension less than  $m + 1$ .

This means that for any  $a_n, b_n$  randomly chosen from any intervals of  $R$ , such as  $a_n = 1, b_n = 0$ , according to any continuous probability distribution, the column vectors of  $\mathbf{H}$  can be made of full rank with probability one, which validates the above theorem.

Moreover, there exists an  $n \leq m$ , so that matrix  $\mathbf{H}$  is rectangular, and given any small positive value  $\varepsilon > 0$  and Legendre activation function  $P_i(x) : R \rightarrow R$ , for  $m$  arbitrary distinct samples  $(\mathbf{x}, \mathbf{f})$ , and for any  $a_n, b_n$  randomly chosen from any intervals of  $R$ , according to any continuous probability distribution, we have  $\|\mathbf{H}_{m \times n} \boldsymbol{\beta}_{n \times 1} - \mathbf{f}_{m \times 1}\| < \varepsilon$ . □

### 6 Numerical results and comparative study

Numerical experiments were conducted to verify the effectiveness and superiority of the Legendre network with IELM algorithm. The new scheme was tested on linear differential equations (such as first order, second order ODEs, and SODE) and nonlinear differential equations. A comparative study with other approaches is also described in this section, including traditional methods and latest research works. We will discuss a differential equation appearing in practice (such as Emden–Fowler type equation) to validate the proposed

Legendre neural network with IELM algorithm and will show that this method is very encouraging at the end of this section.

All numerical results are obtained using MATLAB R2015a, on a computer with INTEL Core I7-6500U CPU, 4 GB of memory, 512 GB SSD and WIN10 operating system.

We use mean absolute deviation (MAD) to measure the error of numerical solution:

$$\text{MAD} = \frac{1}{m}(\Delta_1 + \Delta_2 + \dots + \Delta_m), \tag{26}$$

where  $\Delta_1, \Delta_2, \dots, \Delta_m$  is the absolute error at each discrete point.

### 6.1 Experimental results

*Example 1* First, we consider the initial value problem of ODEs expressed as

$$y' + \frac{\cos(x)}{\sin(x)}y = \frac{1}{\sin(x)},$$

$$y(1) = \frac{3}{\sin(1)}.$$

This problem has been solved in [1, 2], and the exact solution is  $y(x) = \frac{(x+2)}{\sin(x)}$ .

*Example 2* The second problem is given by

$$y' + \frac{1}{5}y = e^{-\frac{x}{5}} \cos(x),$$

$$y(0) = 0$$

and considered on the interval [0, 2]. The exact solution is  $y(x) = e^{-\frac{x}{5}} \sin(x)$ .

*Example 3* For the third problem, we consider

$$y' + \left(x + \frac{1 + 3x^2}{1 + x + x^3}\right)y = x^3 + 2x + \frac{x^2 + 3x^4}{1 + x + x^3},$$

$$y(0) = 1,$$

which has the exact solution  $y(x) = \frac{e^{-\frac{x^2}{2}}}{1+x+x^3} + x^2$  on the interval [0, 1].

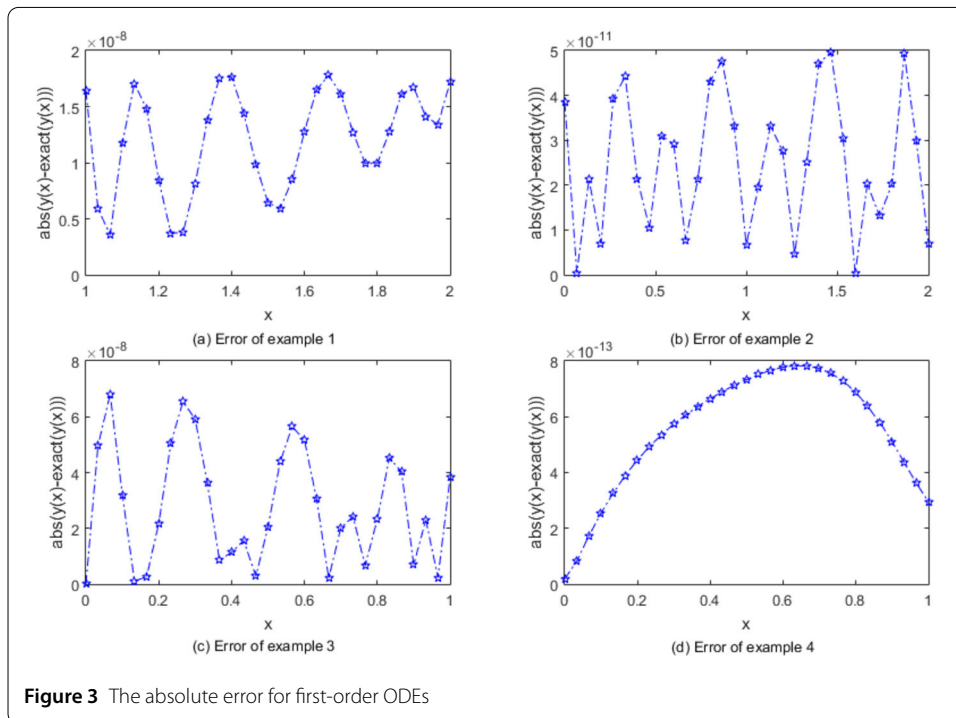
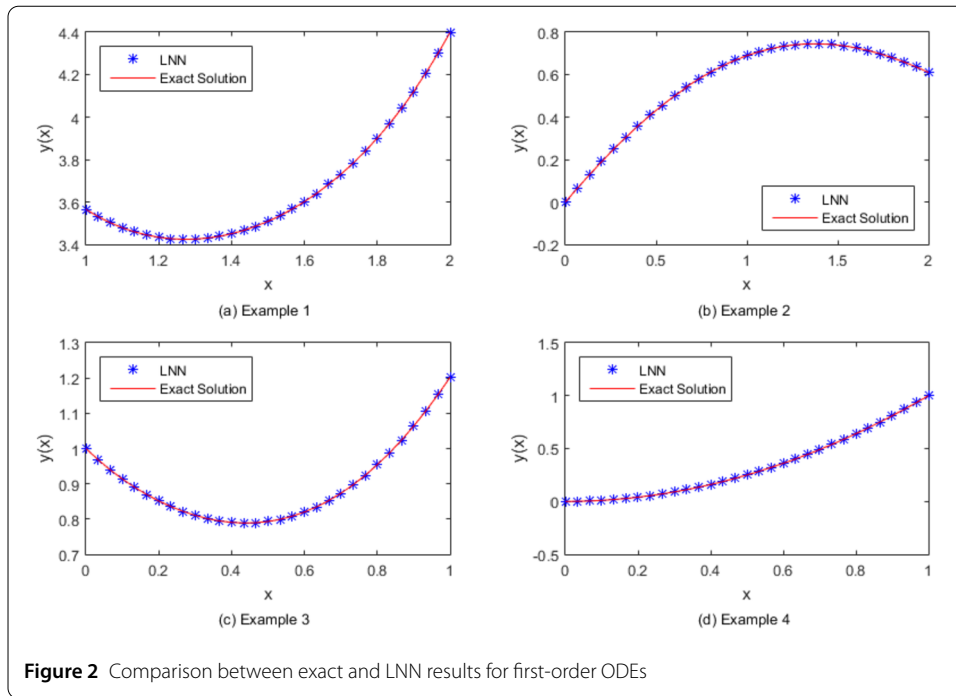
*Example 4* The last initial value problem of ODEs is as below:

$$y' - \sin(x)y = 2x - x^2 \sin(x),$$

$$y(0) = 0.$$

It has been solved in [0, 1], with the exact solution being  $y(x) = x^2$ .

Figures 2 and 3 show the numerical solutions and absolute errors in Examples 1–4. The numerical results are obtained by considering  $n = 10, m = 30$ .

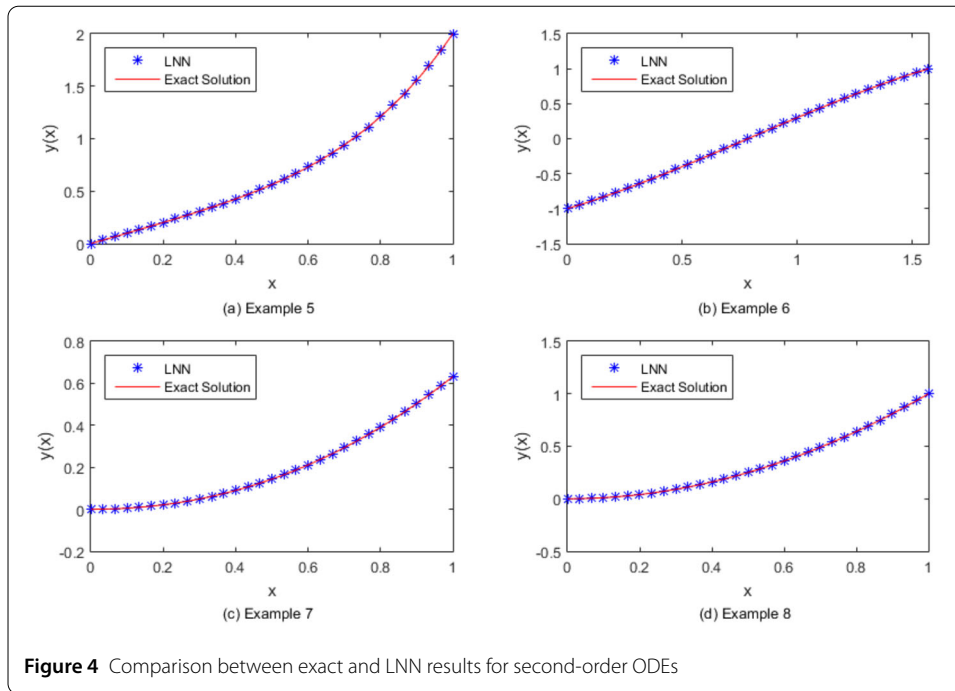


*Example 5* Here we consider the second-order ODEs given as follows:

$$y'' + xy' - 4y = 12x^2 - 3x,$$

$$y(0) = 0, \quad y(1) = 2$$

with the interval being  $[0, 1]$ , and the exact solution  $y(x) = x^4 + x$ .



**Figure 4** Comparison between exact and LNN results for second-order ODEs

*Example 6* The next problem is given by

$$y'' - y' = -2 \sin(x),$$

$$y(0) = -1, \quad y\left(\frac{\pi}{2}\right) = 1$$

and considered on the interval  $[0, \frac{\pi}{2}]$ , with the exact solution being  $y(x) = \sin(x) - \cos(x)$ .

*Example 7* One more problem to be solved is

$$y'' + 2y' + y = x^2 + 3x + 1,$$

$$y(0) = 0, \quad y(1) = -e^{-1} + 1,$$

which is considered on the interval  $[0, 1]$ , and the exact solution is  $y(x) = -e^{(-x)} + x^2 - x + 1$ .

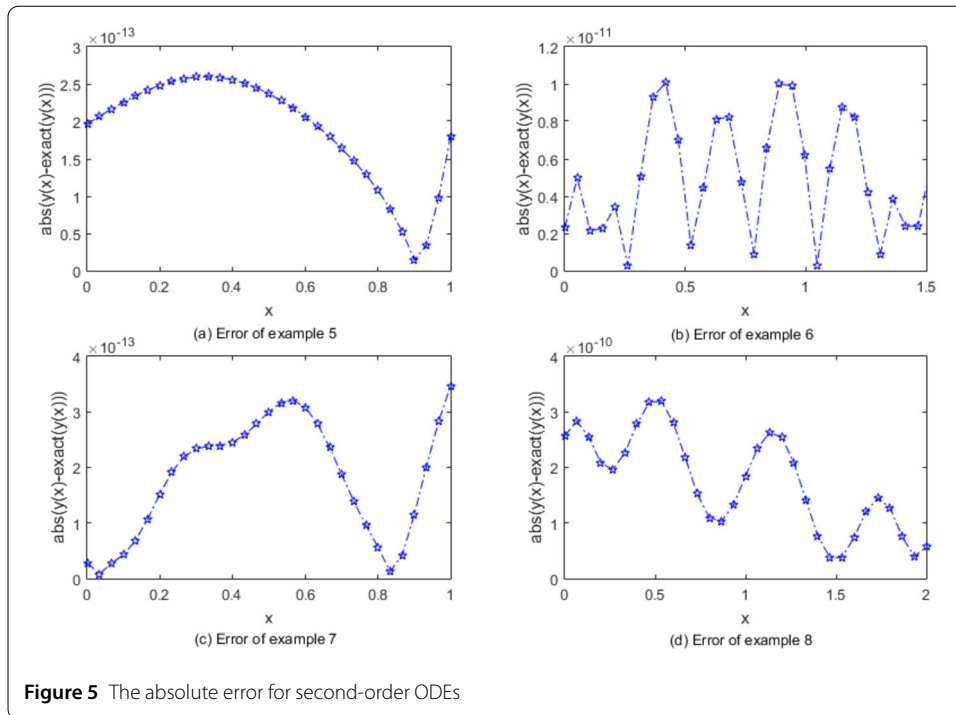
*Example 8* As the last problem we consider the following problem:

$$y'' + \frac{1}{5}y' + y = -\frac{1}{5}e^{-\frac{x}{5}} \cos(x),$$

$$y(0) = 0, \quad y(2) = e^{-\frac{2}{5}} \sin(2),$$

defined on the interval  $[0, 2]$  and having the exact solution  $y(x) = e^{-\frac{x}{5}} \sin(x)$ .

Figures 4 and 5 show the numerical solutions and absolute errors in Examples 5–8. The numerical results are obtained by considering  $n = 10, m = 30$ .



**Figure 5** The absolute error for second-order ODEs

*Example 9* Next we consider the initial value problem of SODEs expressed as

$$\begin{aligned} y_1' + 2y_1 + y_2 &= \sin(x), \\ y_2' - 4y_1 - 2y_2 &= \cos(x), \\ y_1(0) &= 0, \quad y_2(0) = -3, \end{aligned}$$

which has been solved in  $[0, 2]$ , and the exact solution is

$$\begin{cases} y_1(x) = 2 \sin(x) + x, \\ y_2(x) = -3 \sin(x) - 2 \cos(x) - 2x - 1. \end{cases}$$

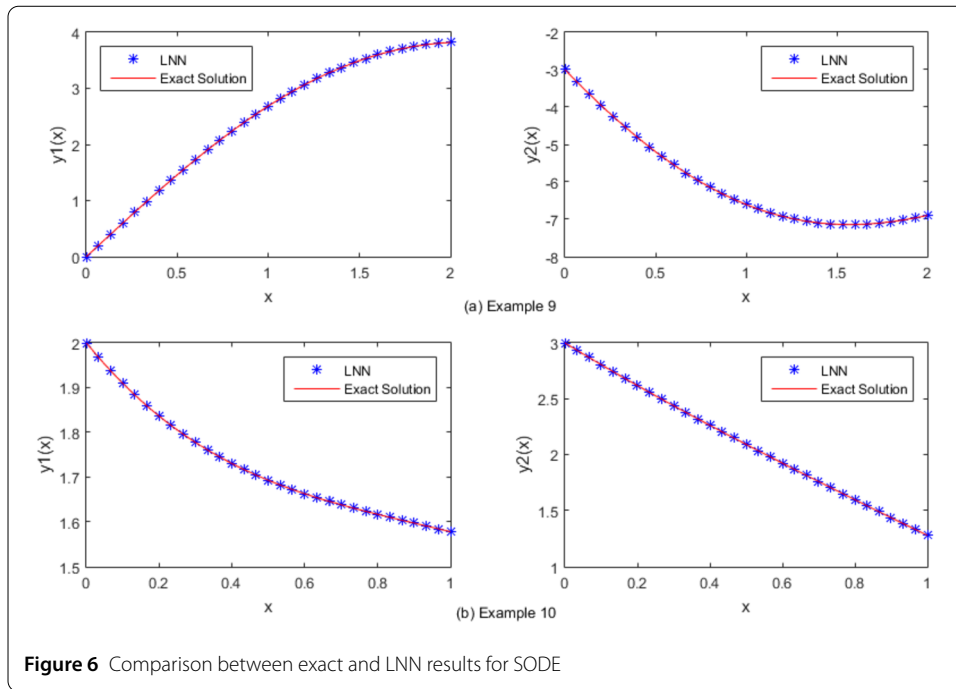
*Example 10* One more problem is given by

$$\begin{aligned} y_1' + 2y_1 - y_2 &= 2 \sin(x), \\ y_2' - y_1 + 2y_2 &= 2(\cos(x) - \sin(x)), \\ y_1(0) &= 2, \quad y_2(0) = 3, \end{aligned}$$

and it is defined on the interval  $[0, 2]$ , with the exact solution being

$$\begin{cases} y_1(x) = 2e^{-x} + \sin(x), \\ y_2(x) = 2e^{-x} + \cos(x). \end{cases}$$

Figures 6 and 7 show the numerical solutions and absolute errors in Examples 9–10. The numerical results are obtained by considering  $n = 10, m = 30$ .



**Figure 6** Comparison between exact and LNN results for SODE

*Example 11* Consider the nonlinear boundary value problem from [59]

$$y'' = \frac{1}{2x^2}(y^3 - 2y^2),$$

$$y(1) = 1, \quad y(2) = 4/3$$

with the exact solution  $y(x) = 2x/(x + 1)$ .

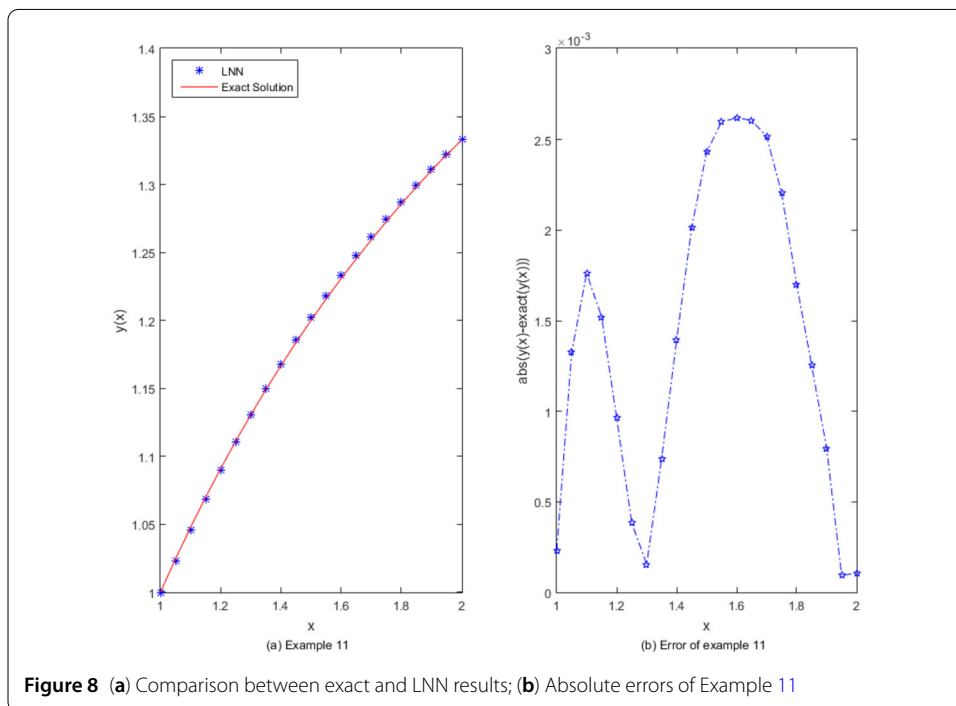
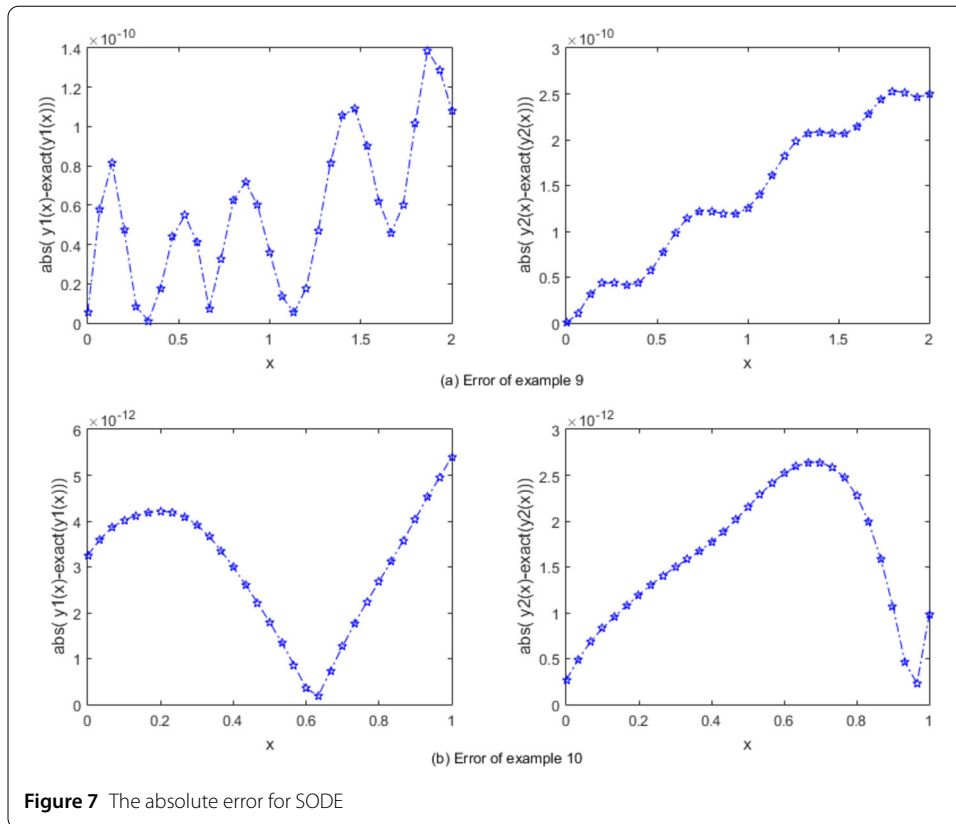
Figure 8 shows the numerical solutions and absolute errors of Example 11. These numerical results are obtained with  $n = 22, m = 20$ .

We also calculated all the test examples with different parameters. Tables 1 and 2 show the mean absolute deviation and execution time of each example. The execution time in Table 2 is the average time of 100 repetitions (in seconds). By analyzing the data of Tables 1 and 2, the best parameter value for each testing problem becomes evident, and we observe that the execution time was only slightly affected by changing network parameters.

### 6.2 Comparative study

A comparative study with other approaches such as traditional methods and latest research work is described in this subsection to verify the superiority of the proposed method. We first compared our approach with some common traditional methods.

For test Examples 1–4, different methods such as Euler method (EM), Suen third-order Runge–Kutta method (Suen-R-K3), classical fourth-order Runge–Kutta method (R-K4), cosine basis function neural network based on gradient descent algorithm (CNN(GD)) and improved extreme learning machine algorithm (CNN(IELM)) were used. Tables 3 and 4 show the numerical results by all the methods mentioned above; they show that LNN method has maximum accuracy and minimum execution time. Table 5 gives the parameters of each algorithm.



Shooting method (SM), difference method (DM), and CNN(IELM) method were used to calculate solutions in Examples 5–8. All calculation were taken over 100 sample points, and the number of neurons was 10 for neural network. The mean absolute deviation and

**Table 1** Mean absolute deviation of test examples with different parameters

Example	$m = 100$			$n = 10$		
	$n = 5$	$n = 8$	$n = 10$	$m = 50$	$m = 200$	$m = 500$
Example 1	2.179822e-04	1.124613e-07	6.070510e-08	1.594745e-08	3.258493e-08	1.893462e-08
Example 2	2.898579e-05	8.788694e-09	1.654316e-10	2.375011e-11	8.614644e-11	5.764976e-11
Example 3	2.651596e-04	1.352667e-06	2.191930e-08	2.476702e-08	2.041171e-08	1.950112e-08
Example 4	1.365455e-15	8.406017e-14	2.180178e-12	9.360722e-13	3.13950e-12	5.474343e-12
Example 5	4.048192e-15	1.721054e-14	1.911390e-13	1.914236e-13	2.815815e-13	3.639675e-13
Example 6	1.785691e-06	3.153652e-09	6.852182e-12	4.679648e-12	1.464149e-11	1.074478e-11
Example 7	1.535907e-05	2.385380e-11	1.783986e-13	8.363132e-14	9.388591e-14	3.149325e-13
Example 8	9.719036e-04	1.792439e-08	8.837987e-11	1.108446e-10	9.767964e-11	1.867956e-10
Example 9	$y_1$ 1.011548e-04	1.198037e-08	2.726976e-11	4.126236e-11	3.043601e-11	2.383795e-11
	$y_2$ 1.696858e-04	2.137740e-08	4.481164e-11	1.15183e-10	4.740707e-11	1.311124e-11
Example 10	$y_1$ 4.814629e-07	7.071364e-12	2.296702e-12	2.432451e-12	2.747277e-12	1.456190e-12
	$y_2$ 2.391016e-07	3.228814e-11	1.792269e-12	8.192227e-13	6.997846e-12	5.358554e-12
Example 11	5.861896e-02	2.837823e-03	1.038392e-02	2.361003e-02	1.044330e-02	1.047949e-02

**Table 2** Execution time of test examples with different parameters

Example	$m = 100$			$n = 10$		
	$n = 5$	$n = 8$	$n = 10$	$m = 50$	$m = 200$	$m = 500$
Example 1	0.0068	0.0073	0.0073	0.0073	0.0073	0.0077
Example 2	0.0057	0.0059	0.0065	0.0063	0.0063	0.0064
Example 3	0.0056	0.0059	0.0061	0.0060	0.0062	0.0068
Example 4	0.0037	0.0040	0.0040	0.0041	0.0042	0.0048
Example 5	0.0060	0.0063	0.0066	0.0063	0.0064	0.0071
Example 6	0.0056	0.0063	0.0062	0.0062	0.0061	0.0067
Example 7	0.0058	0.0059	0.0062	0.0060	0.0063	0.0067
Example 8	0.0057	0.0060	0.0061	0.0060	0.0062	0.0075
Example 9	0.0120	0.0121	0.0121	0.0122	0.0124	0.0132
Example 10	0.0118	0.0112	0.0114	0.0113	0.0117	0.0123
Example 11	0.0025	0.0030	0.0037	0.0033	0.0049	0.0082

**Table 3** Mean absolute deviation of different methods for first-order ODEs

ODEs	EM	Suen-R-K3	R-K4	CNN(GD)	CNN(IELM)	LNN
Example 1	0.009438	7.199659e-08	3.516257e-11	3.038010e-04	0.018023	6.070510e-08
Example 2	0.006143	2.556570e-08	2.508454e-11	5.306315e-04	0.010317	1.654316e-10
Example 3	0.003830	6.992166e-08	4.140991e-10	2.502357e-04	0.007621	2.191930e-08
Example 4	0.005874	4.650619e-09	2.364301e-11	2.463780e-04	0.010018	2.180179e-12

**Table 4** Execution time of different methods for first-order ODEs

ODEs	EM	Suen-R-K3	R-K4	CNN(GD)	CNN(IELM)	LNN
Example 1	1.4122	4.3523	5.7430	57.2749	0.0311	0.0056
Example 2	1.4694	4.1127	5.8199	3.6168	0.0315	0.0106
Example 3	1.3024	3.9871	5.2435	75.1474	0.0268	0.0063
Example 4	1.2431	3.9994	5.3399	46.1268	0.0272	0.0043

execution time are shown in Tables 6 and 7. It is tempting to conclude that LNN method has maximum accuracy, its execution time is less than that of the shooting and CNN (IELM) methods, and the difference is not significant with the difference method.

We calculated 100 sample points for two SODE problems (Examples 9 and 10) using EM, R-K4, CNN(IELM) methods; the number of neurons in the neural network method is 10. Tables 8 and 9 show the experimental results. It is easy to conclude that LNN method has maximum accuracy, its execution time is less than that of CNN(IELM) method, and the differences are not significant with Euler and R-K4 methods.



**Table 5** Parameters of different methods

Methods	Neurons (n)	Sample size (m)	Iterations	Error sum ( $\epsilon$ )	Moment ( $\lambda$ )
EM	–	100	100	–	–
Suen-R-K3	–	100	100	–	–
R-K4	–	100	100	–	–
CNN(GD)	10	100	–	0.01	0.5
CNN(IELM)	10	100	–	–	–
LNN	10	100	–	–	–

**Table 6** Mean absolute deviation of different methods for second-order ODEs

ODEs	SM	DM	CNN (IELM)	LNN
Example 5	3.630769e–10	1.838151e–05	0.004091	1.911390e–13
Example 6	5.590491e–11	1.046468e–05	0.004280	6.852182e–12
Example 7	1.794887e–11	1.282581e–06	0.003461	1.783986e–13
Example 8	1.110794e–09	1.439213e–05	0.001528	8.837987e–11

**Table 7** Execution time of different methods for second-order ODEs

ODEs	SM	DM	CNN(IELM)	LNN
Example 5	0.0375	0.0060	0.0299	0.0064
Example 6	0.0365	0.0052	0.0330	0.0056
Example 7	0.0372	0.0051	0.0339	0.0057
Example 8	0.0371	0.0051	0.0294	0.0059

**Table 8** Mean absolute deviation of different methods for SODE

ODEs		EM	R-K4	CNN(IELM)	LNN
Example 9	$y_1$	0.018838	2.247437e–10	0.005677	2.726976e–11
	$y_2$	0.046376	5.494596e–10	0.021612	4.481164e–11
Example 10	$y_1$	0.001791	1.411037e–10	0.003651	2.296702e–12
	$y_2$	0.001202	1.041297e–10	0.006159	1.792269e–12

**Table 9** Execution time of different methods for SODE

ODEs	EM	R-K4	CNN(IELM)	LNN
Example 9	0.0155	0.0144	0.1182	0.0107
Example 10	0.0133	0.0140	0.1188	0.0104

The execution time for each algorithm in Table 4 is the average time of 30 repetitions, while in Tables 7 and 9, we averaged over 100 times (results are in seconds).

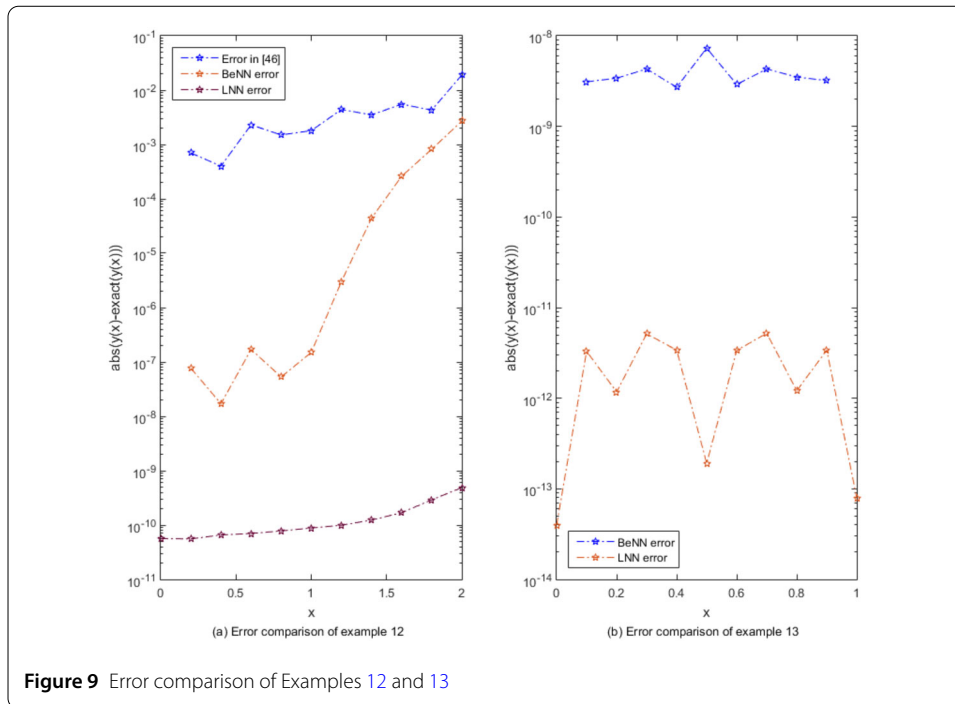
By comparing with traditional methods, we testified the superiority of the new method both in terms of calculation accuracy and execution time. In order to further prove the superiority of the proposed method, a comparison with the latest reported methods is done. The following three ODEs chosen for testing the proposed method are boundary problems.

*Example 12* The first problem chosen from [51] is

$$y' = y - x^2 + 1,$$

$$y(0) = 0.5$$

with  $x \in [0, 2]$  and the exact solution being  $y(x) = (x + 1)^2 - 0.5e^x$ .



In our numerical experiment, the sampling parameter is  $m = 10$ , the result is shown in Fig. 9(a). By comparison, the maximum error in [51] is  $1.9e-2$ , the maximum error of BeNN method in [60] is  $2.7e-3$ , and the maximum error of the proposed LNN method is  $4.9e-10$ . It is easy to see from Fig. 9(a) that LNN method can obtain higher solution accuracy than the other two, which fully validates the superiority of LNN method with IELM algorithm.

*Example 13* The second differential equation is given by [61]

$$y'' + y = 2,$$

$$y(0) = 1, y(1) = 0$$

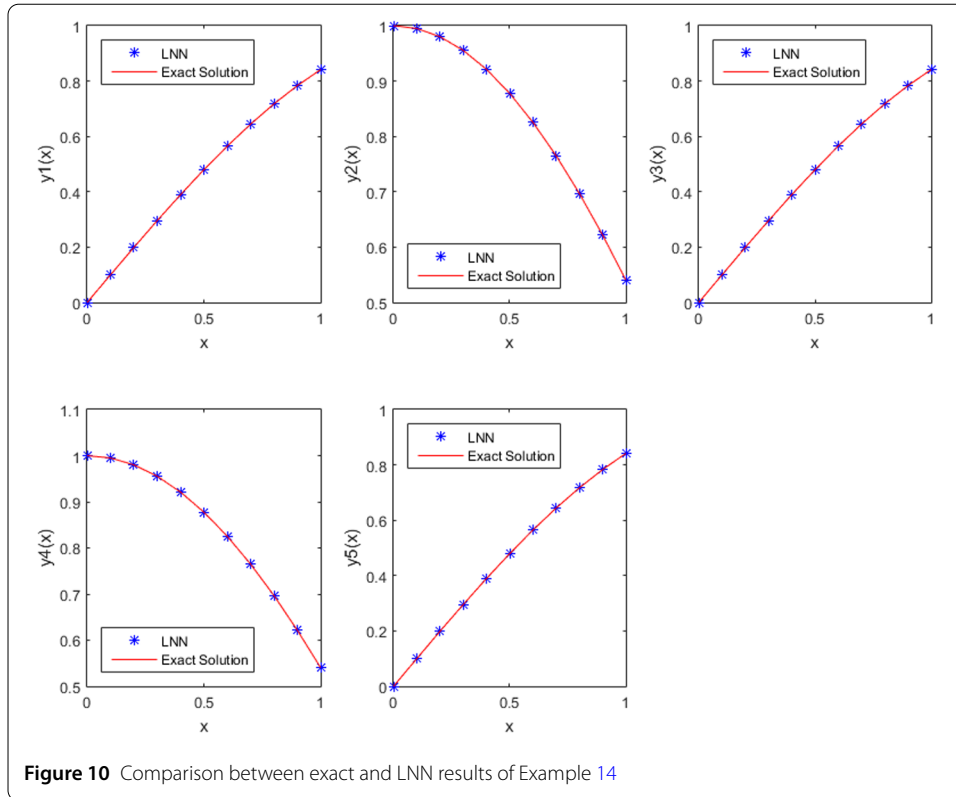
with the exact solution being  $y(x) = \frac{\cos(1)-2}{\sin(1)} \sin(x) - \cos(x) + 2, x \in [0, 1]$ .

The sampling parameter in this test experiment is  $m = 10$ , the result is shown in Fig. 9(b). By comparison, the maximum error of BeNN method in [60] is  $7.3e-9$ , and the maximum error of the proposed LNN method is  $5.2e-12$ , so it is easy to see from Fig. 9(b) that LNN method can obtain higher accuracy solution than BeNN method in [60]. Considering the method given by [61], the maximum error is  $3.5e-2$  with  $m = 50$ , while, using LNN method, we are able to obtain a higher accuracy with maximum error  $2.4e-13$  by  $n = 10$  neurons, which also fully validates the superiority of LNN method with IELM algorithm.

*Example 14* We also test the SODE given by [49]

$$y'_1 = \cos(x), \quad y_1(0) = 0,$$

$$y'_2 = -y_1, \quad y_2(0) = 1,$$



**Figure 10** Comparison between exact and LNN results of Example 14

$$\begin{aligned}
 y_3' &= y_2, & y_3(0) &= 0, \\
 y_4' &= -y_3, & y_4(0) &= 1, \\
 y_5' &= y_4, & y_5(0) &= 0
 \end{aligned}$$

and its exact solution  $y_1(x) = \sin(x), y_2(x) = \cos(x), y_3(x) = \sin(x), y_4(x) = \cos(x), y_5(x) = \sin(x), x \in [0, 1]$ .

As shown in Figs. 10 and 11, the sampling parameter is  $m = 10$  in this study, and only with  $n = 9$  neurons. By comparison with the method proposed in [49] and BeNN method in [60], the maximum errors are  $2.3e-5$  and  $6.8e-8$ , the obtained maximum error of LNN method with IELM algorithm is  $4.0e-12$ , which fully validates the superiority of the new proposed method.

### 6.3 Classic Emden–Fowler equation

Many problems in science and engineering can be modelled by Emden–Fowler equation. A lot of attention has been focused on the numerical solution of Emden–Fowler type equation. In this subsection, we will apply the proposed alternative approach of Legendre neural network (LNN) with IELM algorithm to solve classic Emden–Fowler equation.

*Example 15* Let us consider the classic Emden–Fowler equation given by [62]

$$\begin{aligned}
 y'' + \frac{2}{x}y' + 2y &= 0, \\
 y(0) &= 1, & y'(0) &= 0,
 \end{aligned}$$

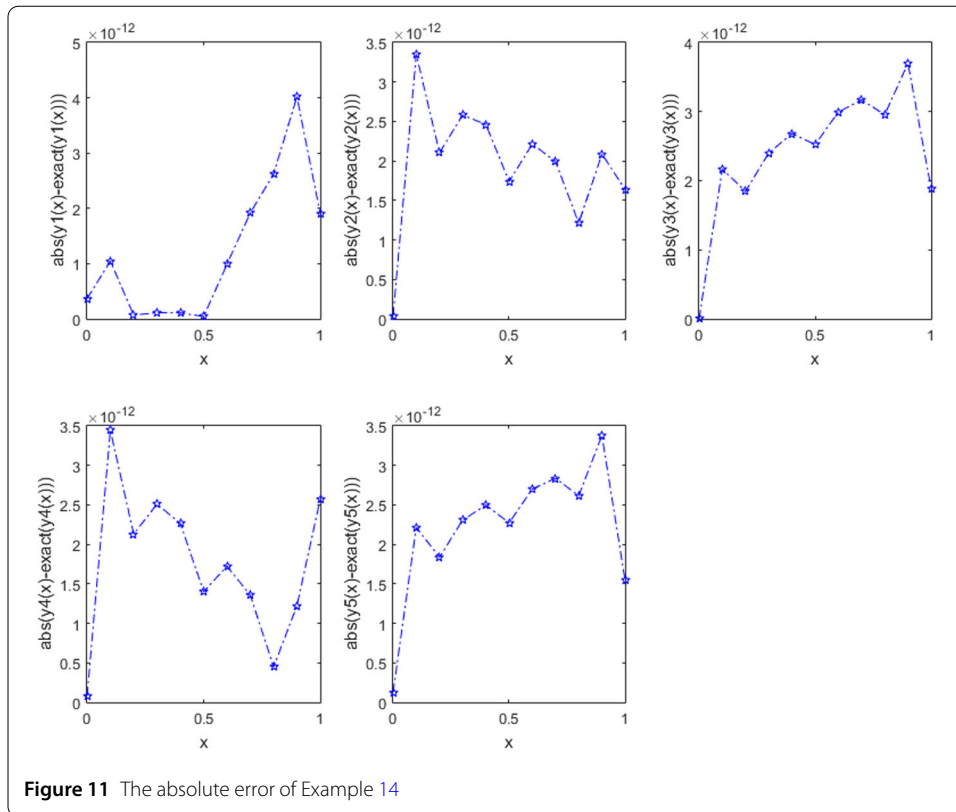


Figure 11 The absolute error of Example 14

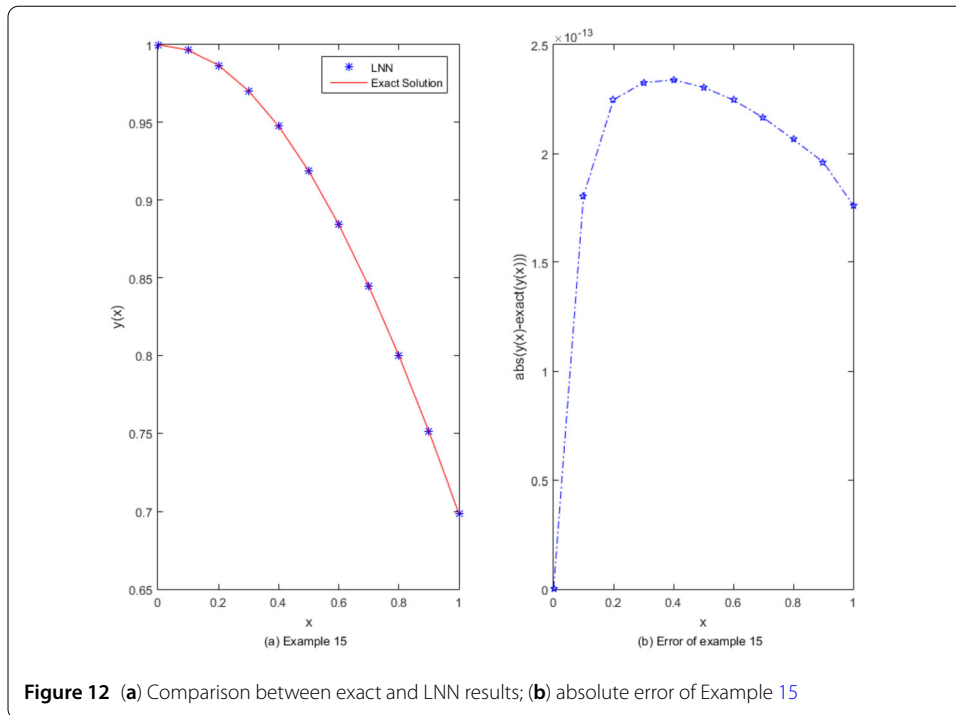
where the exact solution is  $y = \frac{\sin \sqrt{2x}}{\sqrt{2x}}, x \in [0, 1]$ .

Here we choose 10 equidistant points in the domain  $[0, 1]$  to train the proposed Legendre network and with 12 neurons. Figure 12 shows a comparison between the exact solution and LNN results, and the absolute error at each point. Table 10 provides a more intuitive comparison of the exact and numerical solution.

### 7 Conclusions

In this paper, we have presented a novel Legendre neural network to solve several linear or nonlinear ODEs. A Legendre polynomial was chosen as a basis function of the hidden neurons. We used Legendre polynomials to eliminate the hidden layer of the network by expanding the input pattern. An improved extreme learning machine (IELM) algorithm was used for network weights training when solving the algebraic equation systems. Convergence analysis has proved the feasibility of this method. The accuracy of the proposed method has been examined by solving a lot of testing examples, and the results obtained by the proposed method have been compared with the exact solution. We found the presented method to be better. A comparative study has fully validated the superiority of the new proposed method over other numerical algorithms published in the latest literature. An application of the approach to solve the classic Emden–Fowler equation also shows the feasibility and applicability of our method. From the presented investigation we can see that the LNN neural network with IELM algorithm is straightforward, easily implementable and has higher accuracy when solving ODEs.

In addition, the neural network for solving ODEs and PDEs has been discussed a lot and still has some potential to work on. The recent research articles such as [63–66] have stud-



**Figure 12** (a) Comparison between exact and LNN results; (b) absolute error of Example 15

**Table 10** Exact and LNN neural network results

Input points $x_k$	Exact results	LNN results	Absolute error
0	1	1	0
0.1	0.9966699984131	0.9966699984129	1.8e-13
0.2	0.9867198985254	0.9867198985252	2.2e-13
0.3	0.9702688457452	0.9702688457450	2.3e-13
0.4	0.9475135272247	0.9475135272245	2.3e-13
0.5	0.9187253698655	0.9187253698653	2.3e-13
0.6	0.8842466786034	0.8842466786032	2.2e-13
0.7	0.8444857748514	0.8444857748511	2.2e-13
0.8	0.7999112103978	0.7999112103976	2.1e-13
0.9	0.7510451462491	0.7510451462489	2.0e-13
1	0.6984559986366	0.6984559986364	1.8e-13

ied using neural network method to solve several fractional differential equations (FDEs). We have never dealt with the numerical solution of FDEs using neural network method. This will become an important research direction for us in the future. As mentioned in many articles, a variety of phenomena in astrophysics and mathematical physics can be described by Emden–Fowler equations, so this differential equation will also become a research direction for us in the future. If we consider only one type of orthogonal polynomial, there are some published papers as [67, 68], hybrid methods may also be a new research direction.

**Acknowledgements**

The authors sincerely thank the reviewers for their careful reading and valuable comments, which improved the quality of this paper.

**Funding**

This work was supported by the National Natural Science Foundation of China under Grants 61375063, 61271355, 11301549 and 11271378.

### Competing interests

The authors declare that they have no competing interests.

### Authors' contributions

All authors contributed to the draft of the manuscript, all authors read and approved the final manuscript.

### Author details

<sup>1</sup>School of Mathematics and Statistics, Central South University, Changsha, China. <sup>2</sup>College of Arts and Science, National University of Defense Technology, Changsha, China.

### Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 13 September 2018 Accepted: 9 December 2018 Published online: 19 December 2018

### References

1. Han, X.: Numerical Analysis. Higher Education Press, Beijing (2011) (in Chinese)
2. Zhang, Q.: Single-step method for solving the initial value problem of ordinary differential equations. *Bull. Sci. Tech.* **28**(2), 7–9 (2012)
3. Wang, B.: Numerical solution of differential equations and Matlab implementation. *J. Tongling Vocat. Tech. Coll.* **10**(3), 95–97 (2011)
4. Wang, P., Yuan, H., Liu, P., et al.: Finite difference method and implicit Runge–Kutta method for solving Burgers equation. *J. Changchun Univ. Sci. Technol.* **36**(1), 158–160 (2013)
5. Deng, C., Dai, Z., Jiang, S.: Higher-order explicit index Runge–Kutta method. *J. Beijing Univ. Chem. Technol.* **40**(5), 123–127 (2013)
6. Li, M., Chen, H., Zhang, Z.: Fifth-order Adams scheme for solving first order ODEs. *J. Shaoguan Univ.* **34**(12), 14–17 (2013)
7. Huang, Z., Hu, Z., Wang, C.: A study on construction for linear multi-step methods based on Taylor expansion. *Adv. Appl. Math.* **4**(4), 343–356 (2015)
8. Liu, D.: Contrast test of predictor-corrector method for fourth-order Adams–Bashforth combination formula. *J. Xichang Coll.* **26**(3), 43–46 (2012)
9. Le, A.: Shooting method in the application of ordinary difference equations boundary value problem. *Sci. Mosaic* **5**, 247–249 (2011)
10. Li, S.: Finite difference method for ordinary differential equations and its simple application. Anhui University (2010)
11. Abdulla, M.B., Costa, A.L., Sousa, R.L.: Probabilistic identification of subsurface gypsum geohazards using artificial neural networks. *Neural Comput. Appl.* **29**(12), 1377–1391 (2018)
12. Jiao, Y., Pan, X., Zhao, Z., et al.: Learning sparse partial differential equations for vector-valued images. *Neural Comput. Appl.* **29**(11), 1205–1216 (2018)
13. Wang, Y., Liu, M., Bao, Z., et al.: Stacked sparse autoencoder with PCA and SVM for data-based line trip fault diagnosis in power systems. *Neural Comput. Appl.* (2018). <https://doi.org/10.1007/s00521-018-3490-5>
14. Qiao, J., Zhang, W.: Dynamic multi-objective optimization control for wastewater treatment process. *Neural Comput. Appl.* **29**(11), 1261–1271 (2018)
15. Armaghani, D.J., Hasanipanah, M., Mahdiyar, A., et al.: Airblast prediction through a hybrid genetic algorithm—ANN model. *Neural Comput. Appl.* **29**(9), 619–629 (2018)
16. Hernández-Travieso, J.G., Ravelo-García, A.G., Alonso-Hernández, J.B., et al.: Neural networks fusion for temperature forecasting. *Neural Comput. Appl.* (2018). <https://doi.org/10.1007/s00521-018-3450-0>
17. Hou, M., Liu, T., Yang, Y.: A new hybrid constructive neural network method for impacting and its application on tungsten price prediction. *Appl. Intell.* **47**(1), 28–43 (2017)
18. Xi, L., Hou, M., Lee, M., et al.: A new constructive neural network method for noise processing and its application on stock market prediction. *Appl. Soft Comput.* **15**, 57–66 (2014)
19. Dwivedi, A.K.: Artificial neural network model for effective cancer classification using microarray gene expression data. *Neural Comput. Appl.* **29**(12), 1545–1554 (2018)
20. Hou, M., Han, X.: Constructive approximation to multivariate function by decay RBF neural network. *IEEE Trans. Neural Netw.* **21**(9), 1517–1523 (2010)
21. Hou, M., Han, X.H.: The multidimensional function approximation based on constructive wavelet RBF neural network. *Appl. Soft Comput.* **11**(2), 2173–2177 (2011)
22. Hou, M., Han, X.: Multivariate numerical approximation using constructive L-2(R) RBF neural network. *Neural Comput. Appl.* **21**(1), 25–34 (2012)
23. Lagaris, I.E., Likas, A.C.: Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **9**(5), 987–1000 (1998)
24. He, S., Reif, K., Unbehauen, R.: Multilayer networks for solving a class of partial differential equations. *Neural Netw.* **13**(3), 385–396 (2000)
25. Mai-Duy, N., Tran-Cong, T.: Numerical solution of differential equations using multiquadric radial basis function networks. *Neural Netw.* **14**(2), 185–199 (2001)
26. Chua, L.O., Yang, L.: Cellular neural networks: theory. *IEEE Trans. Circuits Syst.* **35**, 1257–1272 (1988)
27. Ramuhalli, P., Udpa, L., Udpa, S.S.: Finite element neural networks for solving differential equations. *IEEE Trans. Neural Netw.* **16**(6), 1381–1392 (2005)
28. Li, X., Ouyang, J., Li, Q., Ren, J.: Integration wavelet neural network for steady convection dominated diffusion problem. In: 3rd International Conference on Information and Computing, vol. 2, pp. 109–112 (2010)
29. Li, J., Luo, S., Qi, Y., Huang, Y.: Numerical solution of differential equations by radial basis function neural networks. In: Proc. Int. Jt. Conf. Neural Netw., vol. 1, pp. 773–777 (2002)

30. Moody, J.E., Darken, C.: Fast learning in networks of locally tuned processing units. *Neural Comput.* **1**(2), 281–294 (1989)
31. Esposito, A., Marinaro, M., Oricchio, D., Scarpetta, S.: Approximation of continuous and discontinuous mappings by a growing neural RBF-based algorithm. *Neural Netw.* **13**(6), 651–665 (2000)
32. Park, J., Sandberg, I.W.: Approximation and radial basis function networks. *Neural Comput.* **5**, 305–316 (1993)
33. Haykin, S.: *Neural Networks: A Comprehensive Foundation*. Pearson Education, Singapore (2002)
34. Franke, R.: Scattered data interpolation: tests of some methods. *Math. Comput.* **38**(157), 181–200 (1982)
35. Mai-Duy, N., Tran-Cong, T.: Approximation of function and its derivatives using radial basis function networks. *Neural Netw.* **27**(3), 197–220 (2003)
36. Manganaro, G., Arena, P., Fortuna, L.: Cellular neural networks: chaos. In: *Complexity and VLSI Processing*, pp. 44–45. Springer, Berlin (1999)
37. Chedhou, J.C., Kyamakya, K.: Solving stiff ordinary and partial differential equations using analog computing based on cellular neural networks. *ISAST Trans. Comput. Intell. Syst.* **4**(2), 213–221 (2009)
38. Takeuchi, J., Kosugi, Y.: Neural network representation of the finite element method. *Neural Netw.* **7**(2), 389–395 (1994)
39. Beltzer, A.I., Sato, T.: Neural classification of finite elements. *Comput. Struct.* **81**(24–25), 2331–2335 (2003)
40. Topping, B.H.V., Khan, A.I., Bahreininejad, A.: Parallel training of neural networks for finite element mesh decomposition. *Comput. Struct.* **63**(4), 693–707 (1997)
41. Manevitz, L., Bitar, A., Givoli, D.: Neural network time series forecasting of finite-element mesh adaptation. *Neurocomputing* **63**, 447–463 (2005)
42. Jilani, H., Bahreininejad, A., Ahmadi, M.T.: Adaptive finite element mesh triangulation using self-organizing neural networks. *Adv. Eng. Softw.* **40**(11), 1097–1103 (2009)
43. Arndt, O., Barth, T., Freisleben, B., Grauer, M.: Approximating a finite element model by neural network prediction for facility optimization in groundwater engineering. *Eur. J. Oper. Res.* **166**(3), 769–781 (2005)
44. Koroglu, S., Sergeant, P., Umrkan, N.: Comparison of analytical, finite element and neural network methods to study magnetic shielding. *Simul. Model. Pract. Theory* **18**(2), 206–216 (2010)
45. Deng, J., Yue, Z.Q., Tham, L.G., Zhu, H.H.: Pillar design by combining finite element methods, neural networks and reliability: a case study of the Feng Huangshan copper mine, China. *Int. J. Rock Mech. Min. Sci.* **40**(4), 585–599 (2003)
46. Ziemianski, L.: Hybrid neural network finite element modeling of wave propagation in infinite domains. *Comput. Struct.* **81**(8–11), 1099–1109 (2003)
47. Meade, A.J. Jr., Fernandez, A.A.: The numerical solution of linear ordinary differential equations by feedforward neural networks. *Math. Comput. Model.* **19**, 1–25 (1994)
48. Meade, A.J. Jr., Fernandez, A.A.: Solution of nonlinear ordinary differential equations by feedforward neural networks. *Math. Comput. Model.* **20**(9), 19–44 (1994)
49. Tsoulos, I.G., Gavrili, D., Glavas, E.: Solving differential equations with constructed neural networks. *Neurocomputing* **72**(10–12), 2385–2391 (2009)
50. Mall, S., Chakraverty, S.: Application of Legendre neural network for solving ordinary differential equations. *Appl. Comput.* **43**, 347–356 (2016)
51. Xu, L.Y., Hui, W., Zeng, Z.Z.: The algorithm of neural networks on the initial value problems in ordinary differential equations. In: *Industrial Electronics and Applications. 2007. Iciea 2007. IEEE Conference on*, pp. 813–816 IEEE New York (2007)
52. Reidmiller, M., Braun, H.: A direct adaptive method for faster back propagation learning: the RPROP algorithm. In: *Proceedings of the IEEE International Conference on Neural Networks*, pp. 586–591 (1993)
53. Malek, A., Beidokhti, R.S.: Numerical solution for high order differential equations using a hybrid neural network-optimization method. *Appl. Math. Comput.* **183**(1), 260–271 (2006)
54. Rudd, K., Ferrari, S.: A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks. *Neurocomputing* **155**, 277–285 (2015)
55. Aarts, L.P., Veer, P.V.: Neural network method for partial differential equations. *Neural Process. Lett.* **14**(3), 261–271 (2001)
56. Chaquet, J.M., Carmona, E.: Solving differential equations with Fourier series and evolution strategies. *Appl. Soft Comput.* **12**, 3051–3062 (2012)
57. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. *Neurocomputing* **70**(1), 489–501 (2006)
58. Dai, H., *Matrix Theory*. Science Press, Beijing (2001)
59. Mall, S., Chakraverty, S.: Application of Legendre neural network for solving ordinary differential equations. *Appl. Soft Comput.* **43**, 347–356 (2016)
60. Sun, H., Hou, M., Yang, Y., et al.: Solving partial differential equation based on Bernstein neural network and extreme learning machine algorithm. *Neural Process. Lett.* (2018). <https://doi.org/10.1007/s11063-018-9911-8>
61. Yazdi, H.S., Pakdaman, M., Modaghegh, H.: Unsupervised kernel least mean square algorithm for solving ordinary differential equations. *Neurocomputing* **74**, 2062–2071 (2011)
62. Chowdhury, M.S.H., Hashim, I.: Solutions of Emden–Fowler equations by homotopy-perturbation method. *Nonlinear Anal., Real World Appl.* **10**, 104–115 (2009)
63. Zuniga-Aguilar, C.J., Coronel-Escamilla, A., Gomez-Aguilar, J.F., et al.: New numerical approximation for solving fractional delay differential equations of variable order using artificial neural networks. *Eur. Phys. J. Plus* **133**(2), 75 (2018)
64. Zuniga-Aguilar, C.J., Romero-Ugalde, H.M., Gomez-Aguilar, J.F., et al.: Solving fractional differential equations of variable-order involving operators with Mittag-Leffler kernel using artificial neural networks. *Chaos Solitons Fractals* **103**, 382–403 (2017)
65. Rostami, F., Jafarian, A.: A new artificial neural network structure for solving high-order linear fractional differential equations. *Int. J. Comput. Math.* **95**(3), 528–539 (2018)
66. Pakdaman, M., Ahmadian, A., Effati, S., et al.: Solving differential equations of fractional order using an optimization technique based on training artificial neural network. *Appl. Math. Comput.* **293**, 81–95 (2017)

67. Mall, S., Chakraverty, S.: Chebyshev neural network based model for solving Lane–Emden type equations. *Appl. Math. Comput.* **247**, 100–114 (2014)
68. Chaharborj, S.S., Chaharborj, S.S., Mahmoudi, Y.: Study of fractional order integro-differential equations by using Chebyshev neural network. *J. Math. Stat.* **13**(1), 1–13 (2017)

**Submit your manuscript to a SpringerOpen<sup>®</sup> journal and benefit from:**

- ▶ Convenient online submission
- ▶ Rigorous peer review
- ▶ Open access: articles freely available online
- ▶ High visibility within the field
- ▶ Retaining the copyright to your article

---

Submit your next manuscript at ▶ [springeropen.com](https://www.springeropen.com)

---