

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/367329953>

# An efficient numerical method to solve ordinary differential equations using Fibonacci neural networks

Article in Computational and Applied Mathematics · January 2023

DOI: 10.1007/s40314-023-02197-x

CITATION

1

READS

269

2 authors:



Kushal Dhar Dwivedi

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING

9 PUBLICATIONS 85 CITATIONS

SEE PROFILE



J.F. Gómez-Aguilar


CONACyT-Tecnológico Nacional de México/CENIDET

472 PUBLICATIONS 14,708 CITATIONS

SEE PROFILE



# An efficient numerical method to solve ordinary differential equations using Fibonacci neural networks

Kushal Dhar Dwivedi<sup>1,2</sup> · J. F. Gómez-Aguilar<sup>3,4</sup> 

Received: 4 December 2021 / Revised: 4 January 2023 / Accepted: 8 January 2023

© The Author(s) under exclusive licence to Sociedade Brasileira de Matemática Aplicada e Computacional 2023

## Abstract

The authors present a method to solve differential equations with any kind of initial and boundary conditions using the Fibonacci neural network (FNN). Fibonacci polynomial has been used as an activation function in the middle layer to construct the FNN. The trial solution of the differential equation is considered as the output of the feed-forward neural network, which consists of adjustable parameters (weights). The weights are adjusted with Newtons' like method for equality constraints. The authors have also shown the stability and convergence of the weights with iteration through the graphs. The application of the current method is range from single ordinary differential equations (ODEs) to system of ODE's. The authors have implemented the method to solve a variety of differential equations and established the exactness of the current method by comparison of the solution obtained by previously solved methods.

**Keywords** Differential equation · Neural network · Convergence · Newton method · Fibonacci polynomial

**Mathematics Subject Classification** 92B20 · 33E30 · 49M15 · 40Axx

---

Communicated by Jose Alberto Cuminato.

---

✉ J. F. Gómez-Aguilar  
jose.ga@cenidet.tecnm.mx

Kushal Dhar Dwivedi  
kddwivedi1993@gmail.com

<sup>1</sup> Department of Mathematical Sciences IIT (BHU), Varanasi 221005, India

<sup>2</sup> Department of Mathematics, S. N. Govt. P. G. College, Khandwa 450001, India

<sup>3</sup> CONACyT-Tecnológico Nacional de México/CENIDET, Interior Internado Palmira S/N, Col. Palmira, C.P. 62490 Cuernavaca, Morelos, Mexico

<sup>4</sup> Universidad Tecnológica de México, UNITEC MÉXICO-Campus En Línea, Mexico, Mexico

# 1 Introduction

It is well-known fact that the differential equations is used as a backbone to describe any physical phenomena from a very long time. Many engineering, mathematical, physical and economical problems are modeled by differential equations. The main challenge is to find the solution of a nonlinear differential equation, which describes a particular phenomenon. In many cases, it is not possible to find the exact solutions of the various differential equations due to the complexity associated with it. So, the numerical methods are introduced to solve such differential equations. Many numerical methods were developed by researcher to solve the differential equation numerically, viz. Wambecq (1978), finite difference method, finite element method Reddy (1993), etc. In the current era of time, the main challenge is not only to develop a numerical method that solve the differential equations but that particular method should be more accurate than previously existing methods. In recent years many researchers have developed various efficient and accurate methods than previously existing methods. Viz, Lyu and Vong (2020) developed an efficient method to solve q-fractional differential equations. Hatamzadeh-Varmazyar and Masouri (2018) have developed a very accurate method to solve differential equation of any order numerically. Khudair et al. (2016) have used an efficient scheme based on the differential transformation method to solve the second-order random differential equations very efficiently.

In the recent years, an artificial neural network (ANN) became more popular due to its ability of doing unbelievable tasks, viz., classifying and finding patterns in real life data for various companies to boost their capitals, speech recognition, image processing, developing machines that can take decisions more accurately than humans, and much more. From the last one or decades ANN showed his true potential performance. So the main question arises that does ANN can perform that much efficiently to solve differential equations and can be better than previously existing numerical methods? To find the answer of this question Lee and Kang (1990) first introduced the Hopfield neural network to solve the ordinary differential equations. Nowadays, many researchers have developed various types of ANN to solve differential equations, that perform better than traditional methods, viz., Yazdi et al. (2011) used trained ANN with unsupervised learning to solve the ordinary differential equations. He et al. (2000) trained multi-layer ANN to solve class of partial differential equations(PDEs). Tsoulos et al. (2009) developed a hybrid method to solve ordinary differential equations(ODEs), system of ODEs and PDEs efficiently. McFall and Mahan (2009) introduced a very accurate method to solve PDEs with arbitrary boundary conditions with the help of ANN. Chakraverty and Mall (2014) trained ANN with the help of regression-based weight generation algorithm to solve initial and boundary value problems. During the lecture survey (Pakdaman et al. 2017; Raja et al. 2010; Mall and Chakraverty 2014; Zúñiga-Aguilar et al. 2017; Bélair et al. 1996; Kumar and Yadav 2011), authors have found many types of ANN to solve differential equation. In Pakdaman et al. (2017), the authors studied the solution of fractional differential equations using artificial neural networks, an optimization approach was exploited to adjust the weights of ANNs such that the approximated solution satisfies the FDE. In Raja et al. (2010), the authors proposed a stochastic technique for the solution of nonlinear fractional Riccati differential equation. Particle swarm optimization (PSO) was used to obtain rapid global search method. The scheme proposed was very efficient for solving integer-order or fractional-order Riccati differential equations. The authors in Mall and Chakraverty (2014), solved ordinary differential equations of Lane–Emden type considering Chebyshev Neural Network. In this manuscript, homogeneous and non-homogeneous Lane–Emden equations were considered to show effectiveness of Chebyshev Neural Network model. The solution of

Caputo fractional differential equations with variable-order was studied in Zúñiga-Aguilar et al. (2017), the solutions were optimized using the Levenberg–Marquardt algorithm and the solutions of the neural networks were compared with the analytical and the numerical simulations obtained through the Adams–Bashforth–Moulton method.

In the present article, the authors propose a Fibonacci Neural Network(FNN) by adding the various degree of the Fibonacci polynomial as activation function in the hidden layer. The proposed FNN contains one input, one hidden, and one output layer. A perceptrons in the hidden layer is constructed by different degree of Fibonacci polynomial with unit weights. The output layer is the summation of all perceptrons after multiplying different weights. These weights are assigned randomly in the initial stage of the training of FNN. Then after these weights will be updated by appropriate back-propagation algorithm. However, the training algorithm of the FNN is discussed in the forthcoming sections but there is nothing bad to discuss something briefly about it here. The motivation behind the construction of FNN is that, if we assign the output of the FNN as the solution of the differential equation that to be solved, then the whole idea of construction is that we are somehow approximating the desired solution with different degree of the Fibonacci polynomial. Now the only thing left is to minimize the residual of the differential equation with certain equality constraints, that will be obtained by initial/boundary conditions of the problem. In this article, the authors will apply the discussed method on certain differential equation and show that this proposed method is more accurate than previously existing methods through comparison.

## 2 Fibonacci neural network (FNN) model

In this section, we will introduce the structure of the FNN and learning methods to solve the ordinary differential equation.

### 2.1 Architecture of Fibonacci neural network (FNN) model

The architecture of FNN can be seen in the above Fig. 1. The architecture contains one perceptron in the input-output layer. The hidden layer is created with  $n$  perceptron and activated and inputs with the help of different degree of Fibonacci polynomial. The number of perceptron in the hidden layer will depend on the considered problem.  $F_i(x)$  denotes the Fibonacci polynomial of degree  $i$  in the hidden layer, The FNN will operate in the following way

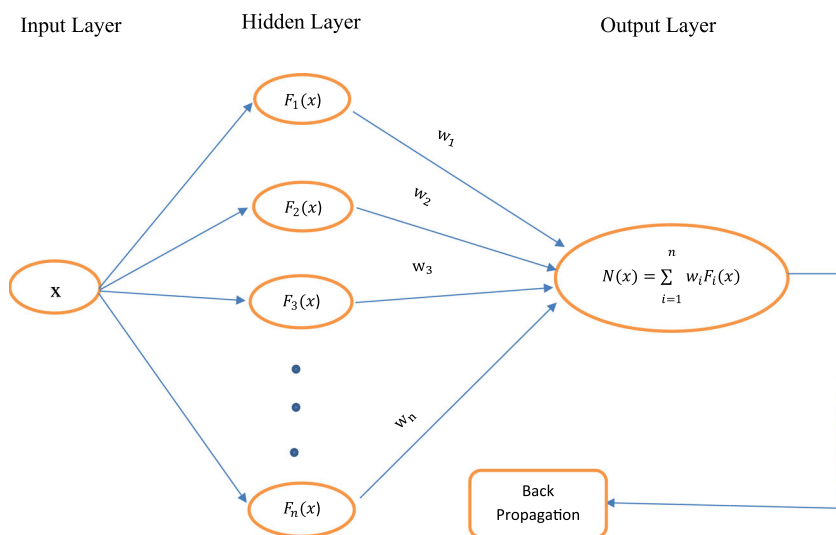
- From the input layer, input  $x$  will be given.
- In the hidden layer, different degree of Fibonacci polynomial has been used to activate the input from the input layers.
- The output layer is a linear combination of inputs with different weights( $w$ ) of the previous layer.

Since we have used Fibonacci polynomial as the activation function in the hidden layer, so it is good to discuss in brief it here.

Fibonacci polynomial of any order can be constructed with the help of following recurrence relation:

$$F_{m+2}(x) = xF_{m+1}(x) + F_m(x), \quad m \geq 0,$$

with initial conditions as



**Fig. 1** Architecture of FNN for solving ODE's

$$F_0(x) = 0, \quad F_1(x) = 1.$$

The series form of Fibonacci polynomial of degree  $m$  can be obtained with the help of above relations

$$F_m(x) = \sum_{r=0}^{\lfloor \frac{m-1}{2} \rfloor} \binom{m-r-1}{r} x^{m-2r-1}, \quad (1)$$

where the notation  $\lfloor \cdot \rfloor$  denotes the floor function.

Now, rewriting Eq. (1), we get

$$F_i(x) = \sum_{j=0}^i \frac{(\frac{i+j-1}{2})!}{j!(\frac{i-j-1}{2})!} x^j, \quad (j+i) = \text{odd}, \quad i \geq 0. \quad (2)$$

In the next section, a method has been discussed to solve differential equation with FNN.

## 2.2 Method to apply the FNN to solve differential equations

In this section, the authors illustrate the developed approach to solve the ordinary differential equations with the help of FNN and we generalize our approach to solving the various types of differential equations in the next step. Let us consider the following general form of ordinary differential equation:

$$y^{(n)}(x) = f(x, y(x), y^{(1)}(x), y^{(2)}(x), \dots, y^{(n-1)}(x)), \quad (3)$$

with the initial conditions

$$y(a) = a_0, \quad y^{(1)}(a) = a_1, \quad y^{(2)}(a) = a_2, \dots, \quad y^{(n-1)}(a) = a_{n-1}. \quad (4)$$

Let  $y_i(x) = N(x, w)$  be the trial solution of above considered problem (3) with initial conditions (4). Thus above problem is transformed in the following equality constrained

minimization problem.

$$\min_w \sum_{x_k} \left( N^{(n)}(x_k, w) - f(x_k, N^{(1)}(x_k, w), N^{(2)}(x_k, w), \dots, N^{(n-1)}(x_k, w)) \right)^2, \quad (5)$$

with constraints

$$N(a, w) = a_0, N^{(1)}(a, w) = a_1, N^{(2)}(a, w) = a_2, \dots, N^{(n-1)}(a, w) = a_{n-1}, \quad (6)$$

where  $N(x, w)$  is single output feed forward FNN with one hidden layer having  $l$  perceptron discussed in subsection 2.1, then

$$N(x, w) = \sum_{i=1}^l w_i F_i(x), \quad (7)$$

where  $x$  is the input data and  $F_i(x)$  is Fibonacci polynomial of degree  $i$  and  $w = [w_1, w_2, \dots, w_l]$  is weight which needs to update that minimize the cost function (5) with the equality constraints (6).

### 2.3 Learning algorithm of Fibonacci neural network (FNN)

In this section the authors discuss the method to minimize the cost function with the prescribed equality constraints. The authors have used the Newton like method for equality constrained minimization see (pp. 234) Bertsekas (2014). Let us train the FNN on  $x = [x_1, x_2, \dots, x_h]$ , and cost function for the problem (3) is

$$E(w) = \frac{1}{2h} \sum_{j=1}^h \left( N^{(n)}(x_j, w) - f(x_j, N^{(1)}(x_j, w), N^{(2)}(x_j, w), \dots, N^{(n-1)}(x_j, w)) \right)^2, \quad (8)$$

with equality constraints (6). Then the Lagrangian function will be

$$L(w, \lambda) = E(w) + \sum_{m=0}^{n-1} \lambda_m \{N^{(m)}(a, w) - a_m\}, \quad (9)$$

where  $w = [w_1, w_2, \dots, w_l]$  are weights and  $\lambda = [\lambda_0, \lambda_1, \dots, \lambda_m]$  are Lagrange multiplier. The Newton-like method to minimize the  $E(w)$  with constraints (7) as follows:

$$\begin{aligned} w^{k+1} &= w^k + \Delta w^k, \\ \lambda^{k+1} &= \lambda^k + \Delta \lambda^k, \end{aligned} \quad (10)$$

where  $k$  denotes the  $k$ th iteration of  $w$  and  $\lambda$ . Now  $(\Delta w^k, \Delta \lambda^k) \in R^{l+n-1}$  which will be obtained solving the following set of equations:

$$\begin{bmatrix} H_k & N_k \\ N_k^T & 0 \end{bmatrix} \begin{bmatrix} \Delta w^k \\ \Delta \lambda^k \end{bmatrix} = - \begin{bmatrix} \nabla_w L(w^k, \lambda^k) \\ h^T(w^k) \end{bmatrix}, \quad (11)$$

where  $h^T(w) = [N(a, w) - a_0, N^{(1)}(a, w) - a_1, N^{(2)}(a, w) - a_2, \dots, N^{(n-1)}(a, w) - a_{n-1}]$ ,  $H_k = \nabla_w^2 L(w^k, \lambda^k)$  and  $N_k = \nabla_w h^T(w^k)$ .

We will update the value of weights  $w$  with this method and put back to the FNN to find the value of  $N(x, w)$  till then minimization of (8) with the constraints (6). This process is called back-propagation. In the next section, the authors will discuss the method more briefly with the help of live example.

The authors have not restricted themselves for single differential equation but also demonstrate how to solve a system of differential equations with the same methods. Let us consider

the following general system of differential equations:

$$\left\{ \begin{array}{l} y_1^n(x) = f_1(x, y_1(x), y_1^{(1)}(x), y_1^{(2)}(x), \dots, y_1^{(n-1)}(x), y_2^{(1)}(x), \\ \quad \dots y_2^{(n-1)}(x), \dots, y_m^{(1)}(x)), \\ y_2^n(x) = f_2(x, y_1(x), y_1^{(1)}(x), y_1^{(2)}(x), \dots, y_1^{(n-1)}(x), y_2^{(1)}(x), \\ \quad \dots y_2^{(n-1)}(x), \dots, y_m^{(1)}(x)), \\ \vdots \\ y_i^n(x) = f_i(x, y_1(x), y_1^{(1)}(x), y_1^{(2)}(x), \dots, y_1^{(n-1)}(x), y_2^{(1)}(x), \\ \quad \dots y_2^{(n-1)}(x), \dots, y_m^{(1)}(x)), \end{array} \right.$$

with the conditions

$$y_i(a) = a_{0,i}, \quad y_i^1(a) = a_{1,i}, \dots, y_i^{n-1}(a) = a_{n-1,i}. \quad (12)$$

The cost function for this system of ODEs is defined as

$$E(w) = \frac{1}{2h} \sum_{j=1}^h \sum_{l=1}^i \left( N_l^{(n)}(x_j, w) - f(x, N_l^{(1)}(x_j, w), N_l^{(2)}(x_j, w), \dots, N_l^{(n-1)}(x_j, w)) \right)^2, \quad (13)$$

where  $y_i(x) = N_i(x, w)$ . The solution of the system ODEs will be obtained by minimizing the cost function with the constrained (12) with the help of above-discussed method.

### 3 Numerical result and discussion

In this section, the authors have applied the method on some examples to test the accuracy which is discussed in Sect. 2.1 and, also compares the numerical solution with previously solved methods.

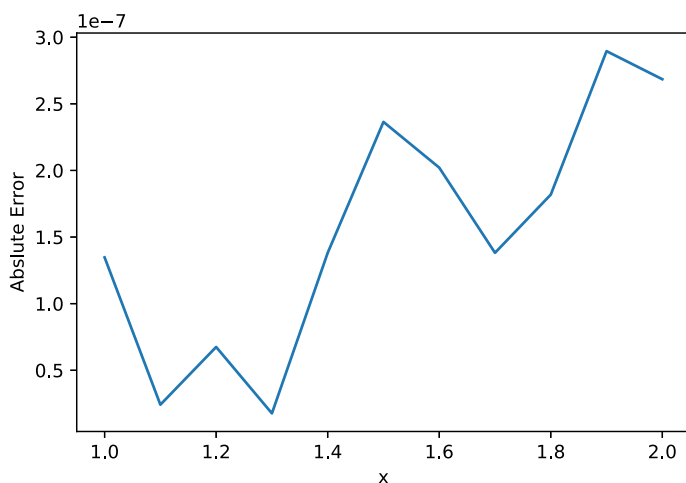
**Example 1** Consider the following boundary value problem

$$\frac{d^2 y}{dx^2} = \frac{1}{2x^2} (y^3 - 2y^2), \quad (14)$$

with boundary conditions  $y(1) = 1$ ,  $y(2) = 4/3$ , which have the exact solution  $y(x) = \frac{2x}{x+1}$ . Consider FNN with 6 perceptrons in the middle layer to solve this problem. Then we have

$$N(x, w) = \sum_{i=1}^6 w_i F_i(x),$$

$$N''(x, w) = \sum_{i=1}^6 w_i F_i''(x).$$



**Fig. 2** Absolute error of Example 1

Now consider the trial solution  $y_t(x) = N(x, w)$ . Then cost function and from boundary conditions, we have

$$E(w) = \frac{1}{h} \sum_{i=1}^h \left\{ N''(x_i, w) - \frac{1}{2x_i^2} (N^3(x_i, w) - 2N^2(x_i, w)) \right\},$$

$$P_1 = w_1 + w_2 + 2w_3 + 3w_4 + 5w_5 + 8w_6 - 1,$$

$$P_2 = w_1 + 2w_2 + 5w_3 + 12w_4 + 29w_5 + 70w_6 - \frac{4}{3}.$$

Now define  $h^T(w) = [P_1, P_2]$ . Lagrangian function will be

$$L(w, \lambda) = E(w) + \lambda_1 P_1 + \lambda_2 P_2.$$

Now the derivatives of Lagrangian function with different weights

$$\frac{dL}{dw_l} = \frac{1}{2h} \sum_{i=1}^h \left\{ F_l''(x_i) - \frac{1}{2x_i^2} (3N(x_i, w) - 2N(x_i, w)) F_l(x_i) \right\} + \lambda_1 \frac{P_1}{dw_l} + \lambda_2 \frac{P_2}{dw_l},$$

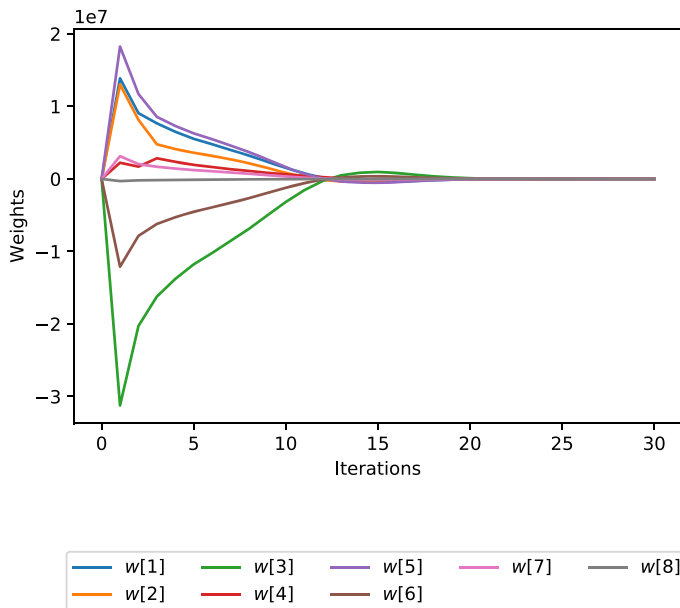
$$\frac{d^2 L(x, w)}{dw_l, w_m} = \frac{1}{2h} \sum_{i=1}^h \left\{ F_l''(x_i) - \frac{1}{2x_i^2} (3N(x_i, w) - 2N(x_i, w)) F_l(x_i) \right\}$$

$$\times \left\{ F_m''(x_i) - \frac{1}{2x_i^2} (3N(x_i, w) - 2N(x_i, w)) F_m(x_i) \right\}.$$

Now

$$N = \begin{bmatrix} \frac{dP_1}{dw_1} & \frac{dP_2}{dw_1} \\ \frac{dP_1}{dw_2} & \frac{dP_2}{dw_2} \\ \vdots & \vdots \\ \frac{dP_1}{dw_6} & \frac{dP_2}{dw_6} \end{bmatrix}, H = \begin{bmatrix} \frac{d^2 L}{dw_1 dw_1} & \frac{d^2 L}{dw_1 dw_2} & \cdots & \frac{d^2 L}{dw_1 dw_6} \\ \frac{d^2 L}{dw_2 dw_1} & \frac{d^2 L}{dw_2 dw_2} & \cdots & \frac{d^2 L}{dw_2 dw_6} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{d^2 L}{dw_6 dw_1} & \frac{d^2 L}{dw_6 dw_2} & \cdots & \frac{d^2 L}{dw_6 dw_6} \end{bmatrix}.$$





**Fig. 3** Convergence of weights over iterations

Now we have all necessary values to use Eq. (11) for each iteration  $k$ , to update the weights  $w$  and Lagrangian multipliers  $\lambda$  from Eq. (10). Figure 2 shows the absolute difference between the exact solution and the numerical solution obtained by the above-discussed method. Figure 3 is drawn to show how weights are converging quickly to a fixed value in less iteration. This shows the stability of the method. Since at the first iteration these weights will be assigned randomly so this will also affect the quick convergence of weights. Mall and Chakraverty (2016) have also solved this problem with a feed-forward Legendre Neural network and the minimum error obtain by this method is around  $e-2$ , but the maximum error obtain by our proposed method is around  $e-7$  which is far better.

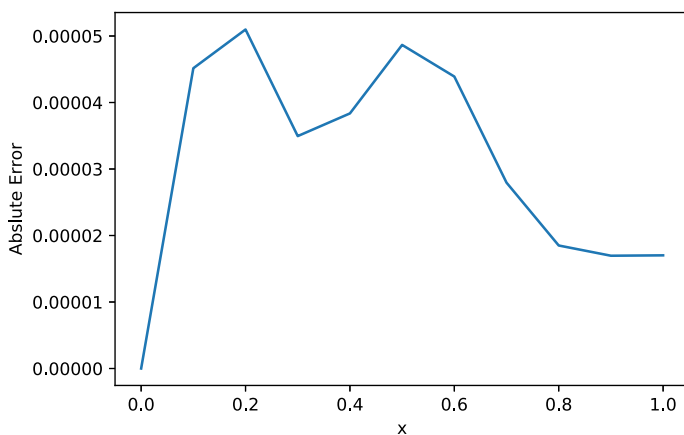
**Example 2** Let us consider the following problem:

$$\frac{d^2y}{dx^2} + \frac{2}{x} \frac{dy}{dx} + 4(2e^y + e^{\frac{y}{2}}) = 0, \quad (15)$$

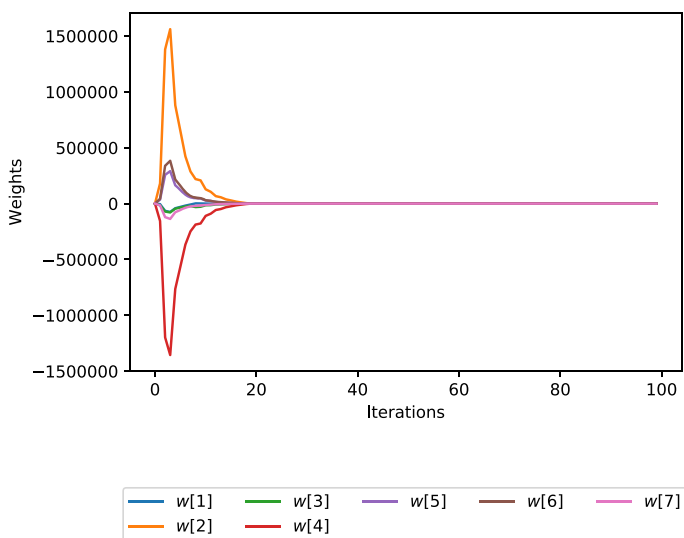
with initial conditions

$$y(0) = 0, \quad \frac{dy}{dx}|_{x=0} = 0 \quad (16)$$

have the exact solution  $y(x) = -2\log(1 + x^2)$ . The authors have trained the FNN over the input  $x = [0, 1]$  and draw the absolute error between analytical and numerical solution in Fig. 4. Figure 5 shows the convergence of weights over the iterations, and it is clearly visible from this figure that, weights are getting settled around some exact value only in very few iterations. Mall and Chakraverty (2016) have also solved this problem and showed that their method gives better results than previously solved methods. The minimum error obtain by their methods is around  $e-3$  but from Fig. 4, its is clearly visible that maximum error from our proposed method is  $e-5$ .



**Fig. 4** Absolute error of Example 2



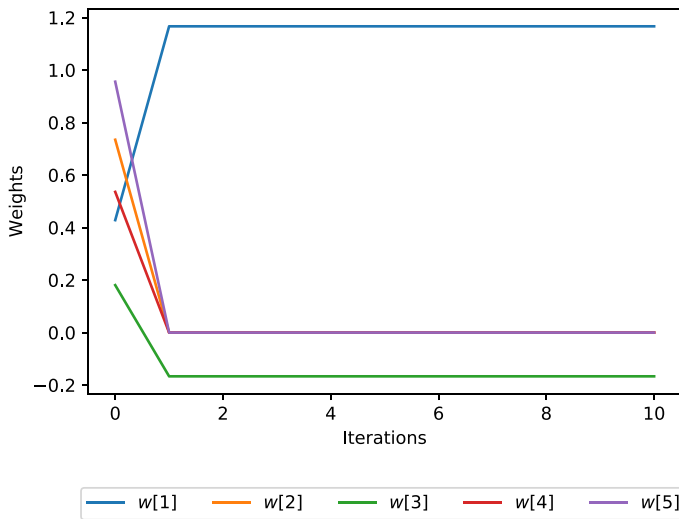
**Fig. 5** Convergence of weights over iterations

**Example 3** Let us consider the following Lane–Emden equation

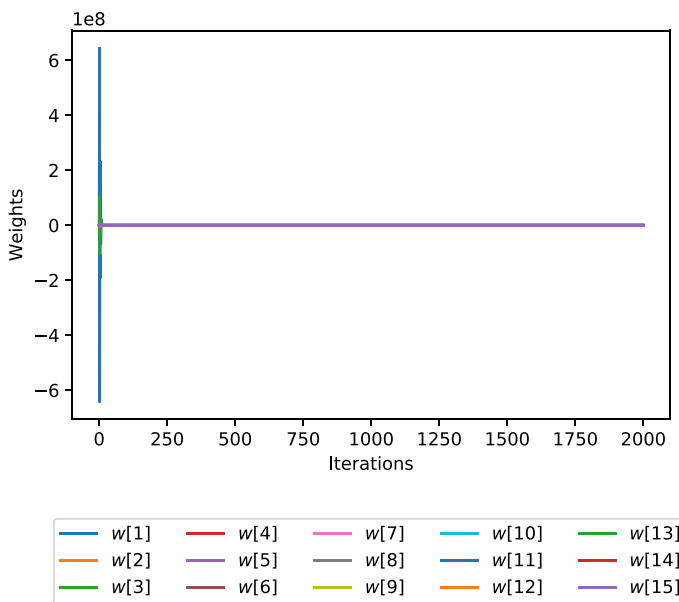
$$y'' + \frac{2}{x}y'(x) + y^m(x), \quad (17)$$

where  $m \in [0, 5]$ , with the following initial conditions  $y(0) = 0$ ,  $y'(0) = 0$ . This differential equation have different solution for each value of  $m$ . Wazwaz (2001) provides an analytical solution of the above problem for every value of  $m$ , which is as follows:

$$y(x) \approx 1 - \frac{x^2}{6} + \frac{mx^4}{120} - \frac{m(8m-5)x^6}{3 \times 7!} + \frac{m(70-183m+122m^2)x^8}{8 \times 8!} + \frac{m(3150-1080m+12142m^2-5032m^3)x^{10}}{45 \times 11!}.$$



**Fig. 6** Convergence of weights over iterations for  $m = 0$



**Fig. 7** Convergence of weights over iterations for  $m = 1$

We have trained the FNN over the input  $x \in [0, 2.5]$  for  $m = 0$  and  $m = 1$  with 5 and 15 perceptrons, respectively, in hidden layer. Figures 6 and 7 are drawn to show the convergence of weights of the FNN. Higher the complexity of FNN will take more iteration to train the model. In Table 1, the obtained results from the FNN are compared with some previously solved methods.

**Table 1** Comparison of absolute Error of example 3

$m$	$x$	Error with presented method	PS-SQP (Ahmad et al. 2016)	PS (Ahmad et al. 2016)	ChNN (Mall and Chakraverty 2014)
0	0.0	0.0	$5.6\text{e}-10$	$4.9\text{e}-7$	0.0
	0.1	$1.1\text{e}-16$	$1.0\text{e}-7$	$3.4\text{e}-5$	$1.0\text{e}-3$
	0.2	$1.1\text{e}-16$	$3.6\text{e}-7$	$8.8\text{e}-5$	$3.2\text{e}-3$
	0.3	$1.1\text{e}-16$	$4.9\text{e}-7$	$1.0\text{e}-4$	$2.8\text{e}-3$
	0.4	0.0	$4.6\text{e}-7$	$8.8\text{e}-5$	$3.3\text{e}-3$
	0.5	0.0	$3.7\text{e}-7$	$7.0\text{e}-5$	$1.9\text{e}-3$
	0.6	$1.1\text{e}-16$	$3.1\text{e}-7$	$6.6\text{e}-5$	$5.4\text{e}-3$
	0.7	0.0	$3.7\text{e}-7$	$7.6\text{e}-5$	$4.4\text{e}-3$
	0.8	0.0	$3.5\text{e}-7$	$8.7\text{e}-5$	$4.1\text{e}-3$
	0.9	0.0	$3.7\text{e}-7$	$8.8\text{e}-5$	$1.7\text{e}-3$
	1.0	0.0	$3.4\text{e}-7$	$8.0\text{e}-5$	$1.1\text{e}-3$
	1.5	0.0	—	—	—
	2.0	$1.1\text{e}-16$	—	—	—
	2.4	$1.1\text{e}-16$	—	—	—
1	0.0	$8.8\text{e}-16$	$7.0\text{e}-9$	$1.8\text{e}-7$	0
	0.1	$1.1\text{e}-16$	$1.4\text{e}-6$	$6.2\text{e}-6$	$3.5\text{e}-3$
	0.2	$3.3\text{e}-16$	$2.0\text{e}-5$	$6.5\text{e}-6$	$2.8\text{e}-3$
	0.3	$5.5\text{e}-16$	$1.2\text{e}-4$	$8.5\text{e}-6$	$1.2\text{e}-3$
	0.4	$2.2\text{e}-16$	$4.1\text{e}-4$	$7.2\text{e}-6$	$1.0\text{e}-4$
	0.5	$5.5\text{e}-16$	$1.0\text{e}-3$	$8.2\text{e}-6$	$9.0\text{e}-4$
	0.6	$3.3\text{e}-16$	$2.0\text{e}-3$	$7.4\text{e}-6$	$6.0\text{e}-4$
	0.7	$4.4\text{e}-16$	$3.8\text{e}-3$	$7.1\text{e}-6$	$7.0\text{e}-4$
	0.8	$4.4\text{e}-16$	$6.4\text{e}-3$	$7.9\text{e}-6$	$4.2\text{e}-3$
	0.9	$5.5\text{e}-16$	$1.0\text{e}-2$	$7.5\text{e}-6$	$4.0\text{e}-4$
	1.0	$4.4\text{e}-16$	$1.5\text{e}-2$	$7.9\text{e}-6$	$1.6\text{e}-3$
	1.5	$3.3\text{e}-16$	—	—	—
	2.0	$1.1\text{e}-16$	—	—	—
	2.4	$2.7\text{e}-16$	—	—	—

**Example 4** Consider the following system of ODEs

$$\begin{aligned} y_1'' + \frac{3}{x} y_1' &= 4(y_1 + y_2), \\ y_2'' + \frac{2}{x} y_2' &= -3(y_1 + y_2), \end{aligned} \quad (18)$$

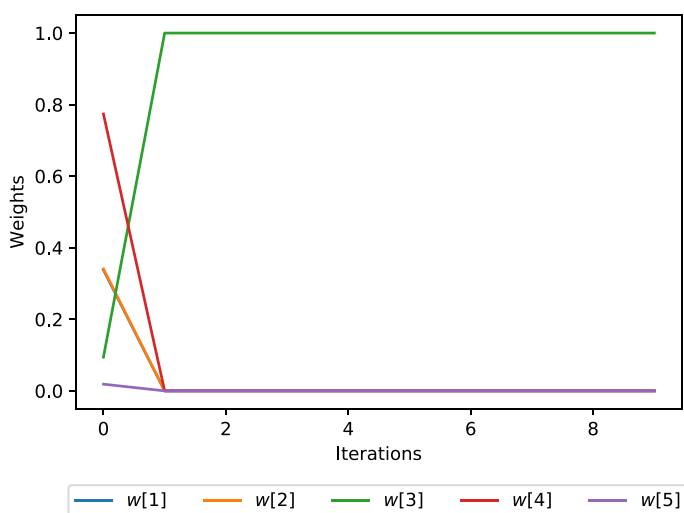
with the initial conditions

$$\begin{aligned} y_1(0) &= 1, \quad y_1'(0) = 0, \\ y_2(0) &= 1, \quad y_2'(0) = 0, \end{aligned} \quad (19)$$

have the exact solutions  $y_1(x) = 1 + x^2$ ,  $y_2(x) = 1 - x^2$ . Table 2 is drawn to show the numerical results obtained by training the FNN with the 5 perceptrons in the middle.

**Table 2** Error table for example 1 for  $n = 6$ 

$x$	Analytical $y_1$	Absolute error	Analytical $y_2$	Absolute error
0.0	1.00	0.00	1.00	0.00
0.1	1.01	0.00	0.99	0.00
0.2	1.04	0.00	0.96	0.00
0.3	1.09	0.00	0.91	0.00
0.4	1.16	0.00	0.84	3.33e-16
0.5	1.25	0.00	0.75	2.22e-16
0.6	1.36	0.00	0.64	2.22e-16
0.7	1.49	0.00	0.51	1.11e-16
0.8	1.64	0.00	0.36	0.00
0.9	1.81	0.00	0.19	0.00
1.0	2.00	0.00	0.00	0.00

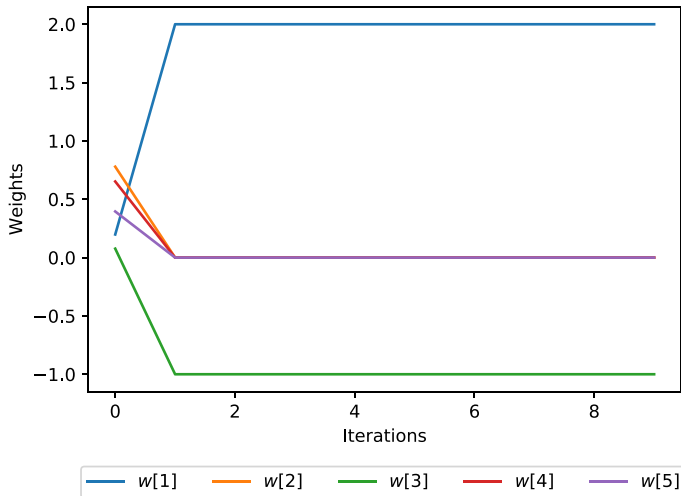
**Fig. 8** Convergence of weights over iterations

Figures 8 and 9 show that weights of both the FNN that required to solve the this problem, are converging to a fixed value in very less iterations.

**Example 5** The following system of differential equation in the domain  $x \in [0, 2]$ .

$$\begin{aligned} y_1' &= \cos(x) + y_1^2 + y_2 - (1 + x^2 + \sin^2(x)), \\ y_2' &= 2x - (1 + x^2) \sin(x) + y_1 y_2, \end{aligned} \quad (20)$$

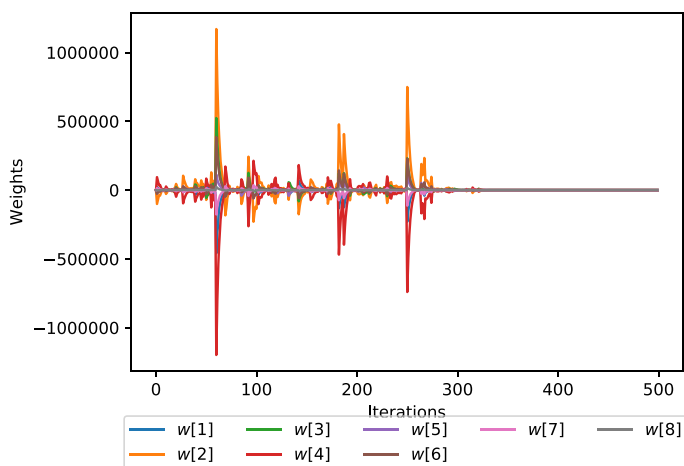
with initial conditions  $y_1(0) = 0$  and  $y_2(0) = 1$  have the exact solutions  $y_1(x) = \sin(x)$  and  $y_2(x) = 1 + x^2$ . To solve this problem the authors have taken FNN with the 8 perceptrons in the middle layer and trained over the input  $x \in [0, 2]$ . The numerical results obtained after training the model is depicted through Table 3. Figures 10 and 11 show about the convergence of weights of both the FNN. From these two figures one can easily see that due



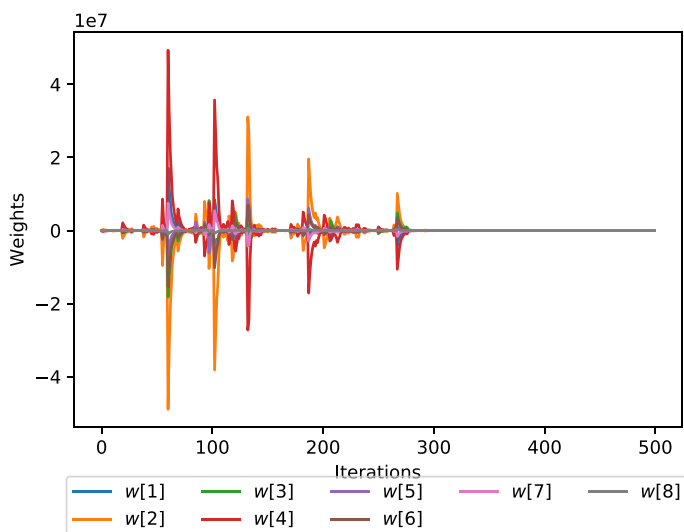
**Fig. 9** Convergence of weights over iterations

**Table 3** absolute Error table for example 5

$x$	Analytical $y_1$	Absolute error	Analytical $y_2$	Absolute error
0.0	0.00000	0.00000000	1.0000	0.0000000
0.1	0.09981	1.9707e-07	1.0100	2.91937e-08
0.2	0.1987	4.93344e-07	1.0400	7.38304e-08
0.3	0.2955	5.3868e-07	1.0900	1.38538e-07
0.4	0.3894	4.06324e-07	1.1600	2.15379e-07
0.5	0.4794	2.85317e-07	1.2500	2.98383e-07
0.6	0.5646	3.19047e-07	1.3600	3.90658e-07
0.7	0.6442	5.49113e-07	1.4900	5.06247e-07
0.8	0.7174	9.27513e-07	1.6400	6.68912e-07
0.9	0.7833	1.36498e-06	1.8100	9.10003e-07
1.0	0.8415	1.78889e-06	2.0000	1.26759e-06
1.1	0.8912	2.19059e-06	2.2100	1.78903e-06
1.2	0.9320	2.6492e-06	2.4400	2.53915e-06
1.3	0.9636	3.32663e-06	2.6900	3.61616e-06
1.4	0.9854	4.43706e-06	2.9600	5.1776e-06
1.5	0.9975	6.20296e-06	3.2500	7.47831e-06
1.6	0.9996	8.81901e-06	3.5600	1.09227e-05
1.7	0.9917	1.24549e-05	3.8900	1.61337e-05
1.8	0.9738	1.73378e-05	4.2400	2.40399e-05
1.9	0.9463	2.39652e-05	4.6100	3.59838e-05
2.0	0.9093	3.35092e-05	5.0000	5.38531e-05



**Fig. 10** Convergence of weights over iterations



**Fig. 11** Convergence of weights over iterations

to complexity of the problem weights are take little more iteration to train the Susmita Mall and Chakraverty (2016) have also solved this problem and they are end-up with minimum absolute error around  $e-3$ . So from Table 3 it is clear that FNN is performing very much better than previously solved method.

## 4 Conclusion and future scope

In this paper, we have proposed a three-layer neural network with Fibonacci polynomial as activation function in the middle layer to solve the ODEs and system of ODEs. The weights

of FNN are updated with Newton's like method, which is quite good in minimizing the cost functions with their equality constraints. The convergence of weights of each example is shown with graphs. The authors have also shown that present method is better than some previously solved methods. The future scope this paper is that one can extend this work by adding more hidden layer in FNN that will surely minimize the error or can extant this work by solving partial differential equations. In future papers we will explore the idea of involving fractional operators to solve fractional differential equations.

**Acknowledgements** José Francisco Gómez Aguilar acknowledges the support provided by CONACyT: Cátedras CONACyT para jóvenes investigadores 2014 and SNI-CONACyT.

**Author Contributions** KDD: conceptualization, methodology, writing—original draft, and supervision. JFG-A: conceptualization, methodology, writing—original draft preparation, and supervision. All authors read and approved the final manuscript.

**Data availability statement** This manuscript has no associated data.

## Declarations

**Conflict of interest** The authors declare no conflict of interest.

## References

- Ahmad I, Raja MAZ, Bilal M, Ashraf F (2016) Bio-inspired computational heuristics to study Lane–Emden systems arising in astrophysics model. *SpringerPlus* 5(1):1866
- Bélair J, Campbell SA, van den Driessche P (1996) Frustration, stability, and delay-induced oscillations in a neural network model. *SIAM J Appl Math* 56(1):245–255
- Bertsekas DP (2014) *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, Cambridge
- Chakraverty S, Mall S (2014) Regression-based weight generation algorithm in neural network for solution of initial and boundary value problems. *Neural Comput Appl* 25(3–4):585–594
- Hatamzadeh-Varmazyar S, Masouri Z (2018) An efficient numerical algorithm for solving linear differential equations of arbitrary order and coefficients. *Int J Ind Math* 10(2):131–142
- He S, Reif K, Unbehauen R (2000) Multilayer neural networks for solving a class of partial differential equations. *Neural Netw* 13(3):385–396
- Khudair AR, Haddad S, Khalaf SL et al (2016) Mean square solutions of second-order random differential equations by using the differential transformation method. *Open J Appl Sci* 6(04):287
- Kumar M, Yadav N (2011) Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: a survey. *Comput Math Appl* 62(10):3796–3811
- Lee H, Kang IS (1990) Neural algorithm for solving differential equations. *J Comput Phys* 91(1):110–131
- Lyu P, Vong S (2020) An efficient numerical method for q-fractional differential equations. *Appl Math Lett* 103:106156
- Mall S, Chakraverty S (2014) Chebyshev neural network based model for solving Lane–Emden type equations. *Appl Math Comput* 247:100–114
- Mall S, Chakraverty S (2016) Application of legendre neural network for solving ordinary differential equations. *Appl Soft Comput* 43:347–356
- McFall KS, Mahan JR (2009) Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Trans Neural Netw* 20(8):1221–1233
- Pakdaman M, Ahmadian A, Effati S, Salahshour S, Baleanu D (2017) Solving differential equations of fractional order using an optimization technique based on training artificial neural network. *Appl Math Comput* 293:81–95
- Raja MAZ, Khan JA, Qureshi IM (2010) A new stochastic approach for solution of Riccati differential equation of fractional order. *Ann Math Artif Intell* 60(3–4):229–250
- Reddy JN (1993) *An introduction to the finite element method*, vol 27. McGraw-Hill Education, New York
- Tsoulos IG, Gavrilis D, Glavas E (2009) Solving differential equations with constructed neural networks. *Neurocomputing* 72(10–12):2385–2391



- Wambecq A (1978) Rational Runge–Kutta methods for solving systems of ordinary differential equations. *Computing* 20(4):333–342
- Wazwaz A-M (2001) A new algorithm for solving differential equations of Lane–Emden type. *Appl Math Comput* 118(2–3):287–310
- Yazdi HS, Pakdaman M, Modagheh H (2011) Unsupervised kernel least mean square algorithm for solving ordinary differential equations. *Neurocomputing* 74(12–13):2062–2071
- Zúñiga-Aguilar C, Romero-Ugalde H, Gómez-Aguilar J, Escobar-Jiménez R, Valtierra-Rodríguez M (2017) Solving fractional differential equations of variable-order involving operators with Mittag–Leffler kernel using artificial neural networks. *Chaos Solitons Fractals* 103:382–403

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.