IJOCTA
An International Journal of Optimization and Control: Theories & Applications
2010

RESEARCH ARTICLE

Numerical solution of coupled system of Emden-Fowler equations using artificial neural network technique

Ashish Kumar[a], Manoj Kumar[b*], Pranay Goswami[a]

[a] School of Liberal Studies, Dr. B. R. Ambedkar University Delhi, Delhi, India
[b] Department of Applied Sciences and Humanities, Indira Gandhi Delhi Technical University for Women, Delhi, India
ashish2.20@stu.aud.ac.in, manojkumar7281@gmail.com, pranay@aud.ac.in

ARTICLE INFO	ABSTRACT
	In this paper, a deep artificial neural network technique is proposed to solve the coupled system of Emden-Fowler equations. A vectorized form of algorithm is developed. Implementation and simulation of this technique is performed using Python code. This technique is implemented in various numerical examples, and simulations are conducted. We have shown graphically how accurately this method works. We have shown the comparison of numerical solution and exact solution using error tables. We have also conducted a comparative analysis of our solution with alternative methods, including the Bernstein collocation method and the Homotopy analysis method. The comparative results are presented in error tables. The efficiency and accuracy of this method are demonstrated by these graphs and tables.

1. Introduction

Differential equations are used to model natural phenomena in many fields of science, including biology, ecology, physics, engineering, chemistry, etc. Due to the vast applications of these differential equations in many branches of science([1], [2]), there is a great need to solve these kinds of equations. The Emden-Fowler equation is also an important differential equation in astrophysics and physics. It is also very difficult to solve this equation numerically due to the singularity behavior of the equation at the point $\zeta = 0$. This equation arises in the study of stellar structure [3], which involves the evolution of a star under the laws of physics. Stars' gravity balances the stars' core radiations. This balance is called hydrostatic equilibrium. The Emden-Fowler equation arises when this equilibrium state is modeled while studying the stellar structure. In 1870, Jonathan Lane [4] introduced the equation, and Jacob Emden [5] generalized it further. Lane -

Emden equation is given as

$$\frac{d^2\omega}{d\zeta^2} + \frac{2}{\zeta}\frac{d\omega}{d\zeta} + \omega^n = 0, \tag{1}$$

where n is the polytropic index. Many astrophysicists were interested in the behavior of the solution of these equations under some initial conditions. Fowler studied the these equations during 1914-1931. The more general form of (1) is

$$\frac{d^2\omega}{d\zeta^2} + \frac{\vartheta}{\zeta}\frac{d\omega}{d\zeta} + \varphi(\omega) = \psi(\zeta), \tag{2}$$

where $\varphi(\omega)$ is a linear or non-linear function. Emden-Fowler equation is also used in chemical reactors, fluid mechanics, and gas dynamics. There are many variants of Emden-Fowler equations including coupled system of Emden-Fowler equations that reads as follows-

$$\begin{aligned} \frac{d^2\omega_1}{d\zeta^2} + \frac{\vartheta_1}{\zeta}\frac{d\omega_1}{d\zeta} = \varphi_1(\zeta, \omega_1, \omega_2), \\ \frac{d^2\omega_2}{d\zeta^2} + \frac{\vartheta_2}{\zeta}\frac{d\omega_2}{d\zeta} = \varphi_2(\zeta, \omega_1, \omega_2), \end{aligned} \tag{3}$$

*Corresponding Author

where $\vartheta_1 > 0$, $\vartheta_2 > 0$ are real constants. φ_1 and φ_2 are nonlinear functions of ζ, ω_1 and ω_2.

The groundwork of artificial neural network(ANN) was started in 1943 when McCulloch and Pitts modeled a neuron as a switch that receives input from other neurons and, depending on total weighted input, is either activated or remains inactive. Thus, artificial neural networks are inspired by sensory processing of the brain. ANN can be created by simulating a network of model neurons in a computer. The structure comprises input layer, output layer, and hidden layers. Every layer contains neurons. By using an algorithm we can make the network learn to solve mathematical problems. A model neuron receives the input from other units, weighs each unit and add them up. If the total input is above a threshold we get the output.

When a structure has multiple hidden layers, deep neural networks (DNNs) are used. Deep neural networks have been proposed as a way to produce more predictive models. Combining a large number of layers endows the model with high prediction power.

2. Related work

Machine learning techniques such as deep neural networks have a wide range of applications such as speech recognition, image classification [6], computer vision tasks [7], machine translation, drug design, climate science, cybersecurity ([8], [9]). DNN is also very useful in automated driving. Automative researchers are able to detect stop signs, traffic lights, pedestrians that helps lower accidents. The deep layers in DNN capture more variances.

The main issues of deep neural networks are robustness, stability and adversarial perturbation which are discussed in ([10], [11], [12], [13]). The source of instability arises from the high depth of neural networks, where a "shattered gradient" effect was observed [14]. More advances in the applications of deep neural networks are presented in ([15], [16]).

Solving differential equations using optimization strategies has been very popular for quite some time in the form of least squares method ([17], [18]) and Galerkin methods [19]. These techniques are also used in neural networks. Nowadays researchers are using ANN to solve differential equations. ANN solutions are in closed-analytic form and are differentiable. Solving a differential equation was first described by Lagaris et al. [20]. Lee sen ten [21] solved ODEs with modified back propagation(mBP) with multilayer perception neural network (MPNN) as ANN technique. Craig Michoski et. al. [22] Solved PDEs using deep neural networks (DNNs) They discussed and analyzed the sod shocktube solution to the compressible Euler equations. Jiequen Han [23] also used deep learning to solve high-dimensional PDEs such as Black-Scholes equation, Allen-cahn equations, Bellman equation. Sabir et al. [24] solved fourth order Emden-Fowler equation using heuristic computing technique. Raja et al. [25] used Neuro-swarms intelligent computing technique using kernel to solve Emden-Fowler equations. Sabir et al. [26] also discussed the applications of neural networks in COVID-19 model using the effects of lockdown. Fernandez [27] used a feedforward neural algorithm to solve linear ordinary differential equations using a limit activation function to construct a feedforward neural network. Fotiadis (1998) also solved ODEs and PDEs using ANN and compared the solution with a numerical method like the finite element method. Aarts and Ven Der (2001) [28] solved PDEs with boundary and initial conditions using neural networks. They used single hidden layers with multiple inputs and a single output. They tested their method on physics and geological problems. Sadogi Yazdi (2011) [29] implemented an ANN technique with kernel mean square algorithm to solve Ordinary differential equations. Asady and Nazarlue (2014) [30] solved the Fredholm integral equation using ANN with great precision. Nystorm (2018) [31] solved PDEs in complex geometries using deep feedforward artificial neural networks. Viana (2020) [32] introduced the recurrent neural networks to integrate ordinary differential equations. For more related works we can check Rackauckas [33], Wang Huan [34].

3. Neural network architecture and working

We are using vectorized algorithm (given in [35]) to solve the system of Emden-Fowler equations using deep neural networks (DNN).
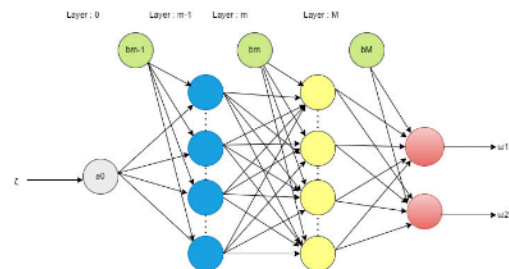

Figure 1. Deep ANN schematic diagram.

As indicated in Figure 1, the network has input layer containing one neuron as an independent variable for system of equations and n neurons in output layer. We form a matrix $T = [\zeta^1, ..., \zeta^r] \in R^{1 \times r}$ using r sample points from domain and take $N \in R^{n \times r}$ as output matrix. For example, $N_\lambda(\zeta^\tau, Q_\lambda)$ is the λth output corresponding to τth point, where Q_λ is related to the weights and bias parameters. For each $\zeta \in [a, b]$ a trial solution, which is an extension of trial solution given in [35], is considered as

$$\widehat{\omega}_j(\zeta, Q_j) =$$
$$a_j + (\zeta - a)N_j(\zeta, Q_j) + \frac{(\zeta - a)^2}{2!}N_j(\zeta, Q_j) + ..., \quad (4)$$

where $j = 1, 2, ..., n$. This trial solution satisfies the initial conditions. The total cost function is

$$H = \sum_{\tau=1}^{r} \sum_{j=1}^{n} \left(\frac{d^2\widehat{\omega}_j}{d\zeta^2} + \frac{\vartheta_j}{\zeta} \frac{d\widehat{\omega}_j}{d\zeta} - \varphi_j \right)^2, \quad (5)$$

converges to 0, $\varphi_j = \varphi_j(\zeta^\tau, \widehat{\omega}_j, Q_j)$.

Forward propagation

We represent $m - 1$ and m layer node by s and j respectively. The value that goes into jth node,

$$u_j^m = \sum_{s=1}^{d} y_{js}^m a_s^{m-1} + b_s^m, \quad (6)$$

In the mth layer, the variable d represents the number of neurons. The equation (6) can be expressed in matrix form as follows:

$$u^m = Y^m a^{m-1} + b^m, \quad (7)$$

where Y^m contains all the weights and b^m is the bias. An appropriate activation function passes The values to the next hidden layer. For consistency, we select the identical activation function for all nodes within a layer. The values for the subsequent layer are as follows:

$$a^m = \pi^m(u^m) = \pi^m(Y^m a^{m-1} + b^m), \quad (8)$$

where π^m is activation function. For the r grid points, where $\tau = 1, 2, ..., r$, we proceed with the following steps. In the initial hidden layer,

$$u^{1(\tau)} = Y^1 z^\tau + b^1,$$
$$a^{1(\tau)} = \pi^1(u^{1(\tau)}),$$

at the second hidden layer,

$$u^{2(\tau)} = Y^2 a^{1(\tau)} + b^2,$$
$$a^{2(\tau)} = \pi^2(u^{2(\tau)}),$$

similarly the output layer,

$$u^{M(\tau)} = Y^M a^{M-1(\tau)} + b^M,$$
$$a^{M(\tau)} = \pi^M(u^{M(\tau)}).$$

So we have the matrix form as

$$U^1 = Y^1 Z + b^1,$$
$$A^1 = \pi^1(Y^1),$$

for second layer

$$U^2 = Y^2 A^1 + b^2,$$
$$A^2 = \pi^2(Y^2),$$

and so on. The output layer

$$U^M = Y^M A^{M-1} + b^M,$$
$$A^M = \pi^M(Y^M),$$

The algorithm

Step 1 : Take r distinct points and form a vector $T = [\zeta^1, ..., \zeta^r] \in R^{1 \times r}$.

Step 2 : We define the structure of neural networks such as number of layers M, input layer having one unit, $M - 2$ hidden layers with g^m units, and an output layer with n units, where n corresponds to the number of unknowns in the system.

Step 3 : Initialize the parameters such as weights and bias, Q_j, $j = 1, 2, ..n$, where n denotes number of neurons. *Step 4 :* Use forward propagation

- Assign $A^0 = Z$ in input layer.
- In hidden layers
$$U^m = Y^m A^{m-1} + b^m,$$
$$A^m = \pi^m(Y^m),$$
where $m = 1, ...M-1$, π^m is the activation function for mth hidden layer.
- For output layer,
$$U^M = Y^M A^{M-1} + b^M,$$
$$A^M = \pi^M(Y^M),$$
- We use trial solution of order four in (4). We need to initialize sets of parameters to find an unknown function.

Step 5 : Use (5) to calculate the cost, gradients, and learning parameters. Apply the automatic differentiation [36].

Step 6 : Utilize gradient descent or any other optimization method to update the parameters. We update the parameters according to following rule

$$Q_j^{r+1} = Q_j^r - \varepsilon \nabla H(Q_j^r),$$

where ε is learning rate and r denotes iteration.

There are many advanced methods we can choose instead of the gradient descent method such as

the moment method, which is a modification of the gradient method. The updating role is

$$Q_j^{r+1} = Q_j^r + L_j^{r+1},$$
$$L_j^{r+1} = \gamma L_j^r - \varepsilon \nabla \mathrm{H}(Q^r),$$

where γ is a coefficient between 0 and 1, we call it momentum. L_j is a velocity parameter starting from 0 corresponding to each unknown.

Another method is Nesterov accelerated gradient obtained from the moment method with update rule

$$Q_j^{r+1} = Q_j^r + L_j^{r+1},$$
$$L_j^{r+1} = \gamma L_j^r - \varepsilon \nabla \mathrm{H}(Q^r + \gamma L_j^r).$$

The adaptive gradient method is

$$L_j^r = L_j^{r-1} + (\nabla \mathrm{H}(Q^r))^2,$$
$$Q_j^{r+1} = Q_j^r - \frac{\varepsilon}{\sqrt{L_j^r + c}} \nabla \mathrm{H}(Q^r),$$

where c is a minimal number to avoid division by 0. Clearly, the learning rate is decaying because of it.

The update rule for root mean square propagation is

$$L_j^r = \rho L_j^{r-1} + (1 - \rho)(\nabla \mathrm{H}(Q^r))^2,$$
$$Q_j^{r+1} = Q_j^r - \frac{\varepsilon}{\sqrt{L_j^r + c}} \nabla \mathrm{H}(Q^r),$$

where $\rho \in (0, 1)$ is forgetting factor. Another method is the Adam adaptive method which is a combination of the last two methods with updating rule

$$F_j^r = \rho_1 F_j^{r-1} + (1 - \rho_1)\nabla \mathrm{H}(Q^r),$$
$$L_j^r = \rho_2 L_j^{r-1} + (1 - \rho_2)(\nabla \mathrm{H}(Q^r))^2,$$
$$\widehat{F_j^r} = \frac{F_r}{1 - \rho_1{}^r}, \widehat{L_j^r} = \frac{L_r}{1 - \rho_2{}^r},$$
$$Q_j^{r+1} = Q_j^r - \frac{\varepsilon}{\sqrt{\widehat{L_j^r} + c}} \widehat{F_j^r},$$

where ρ_1 and ρ_2, both belonging to the interval $[0, 1)$, represent the rates of decay for moment estimation. Initially, we set the parameters F_r and L_r to 0.

4. Numerical illustrations

This section presents the implementation of an algorithm designed to solve known systems of second-order nonlinear differential equations. The initial step involves experimenting to determine the appropriate number of layers and neurons within the layer. The algorithm's accuracy is then validated by comparing the analytic and numerical solutions obtained using conventional methods. To achieve this objective, we simulate the problems related to systems of second-order nonlinear differential equations as follows:

Example 1. *Let us consider a problem of Emden-Fowler equations documented in [37].*

$$\frac{d^2\omega_1}{d\zeta^2} + \frac{3}{\zeta}\frac{d\omega_1}{d\zeta} = -(3 + \omega_2^2)\omega_1^5, \\ \frac{d^2\omega_2}{d\zeta^2} + \frac{4}{\zeta}\frac{d\omega_2}{d\zeta} = (4\omega_1^{-2} + 1)\omega_2^{-3}, \quad (9)$$

with the initial conditions (ICs)

$$\omega_1(0) = \omega_2(0) = 1 \ and \ \omega_1'(0) = \omega_2'(0) = 0.$$

The exact solutions to the problem are $\omega_1(\zeta) = \frac{1}{\sqrt{1+\zeta^2}}$ and $\omega_2(\zeta) = \sqrt{1+\zeta^2}$.

4.1. Investigating the network via experimentation for solving (9)

We performed an experiment to determine how many neurons are in a layer. We examined how the number of hidden layers, the number of neurons, and the number of iterations affected the cost function. We have taken the trial solution up to $n = 4$. During the simulation, we varied the number of hidden neurons, specifically using sizes $h = 4, 13, 27, 60, 90$, and 180. We then compared the convergence of the cost function across these different sizes by plotting it against the number of iterations. In addition, we recorded the corresponding cost values and calculation times for each neuron size at the end of the iterations. It is worth noting that all other parameters remained constant throughout the experiment.

The findings illustrated in Figures 2(a)-(c) demonstrate that achieving the necessary level of accuracy is feasible with only one neuron in the hidden layer. However, many iterations are required when working with a smaller number of neurons, which can pose computational challenges. Increasing the number of neurons can improve the model's performance, but it is not always necessary. For example, when comparing a hidden layer with $h = 60$ neurons to one with $h = 180$ neurons, both achieve similar accuracy, but the former requires less computational time. In our next experiment, we investigated the impact of the number of hidden layers on the model's performance. Previous research by Saeed et al. (2023) [38] demonstrated that to solve nonlinear singular fractional differential equations, increasing the number of hidden layers results in improved performance in terms of error. Similarly, Panghal and Kumar (2021) [39] observed improved accuracy when simulating a delay and
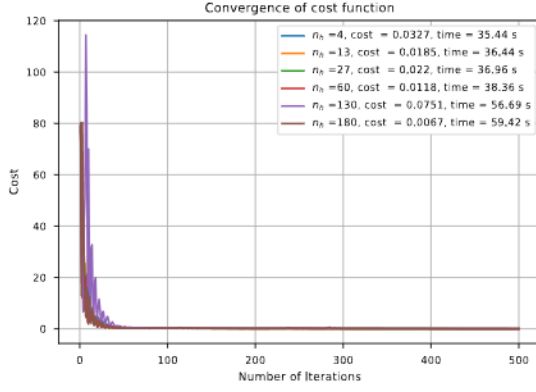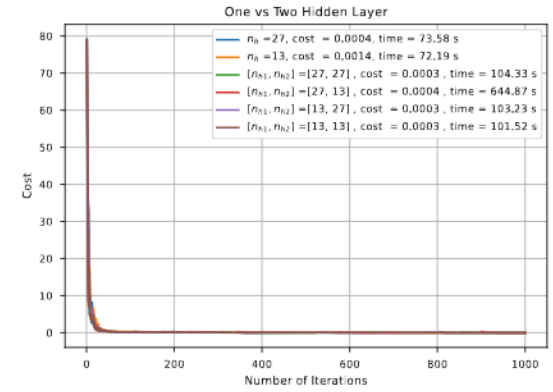
(a) Sub-figure 1
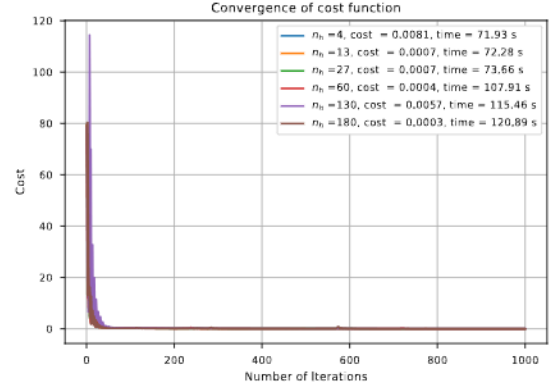
(b) Sub-figure 2
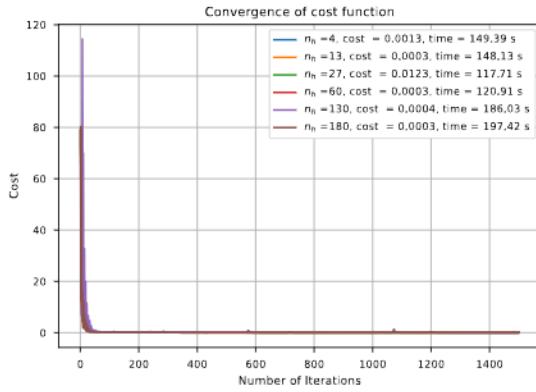
(c) Sub-figure 2

(d) Sub-figure 2

Figure 2. Comparison of loss functions for ANN models with one vs two hidden layers.

first-order differential equation system with multiple hidden layers. However, Dufera (2021) [35] denied more than one hidden layer does not lead to better performance in his experiments when solving first-order a system of differential equations. We conducted numerous experiments to evaluate the performance of neural networks with one hidden layer against those with two hidden layers. The experiments involved varying the number of neurons in the hidden layers. All other parameters and activation functions, such as Tanh, remained the same in all experiments. The results, depicted in Figure 2(d), show that for a system of 2nd order differential equations (9), adding more hidden layers with an appropriate number of neurons leads to improved performance.

4.2. Numerical solutions for solving (9)

We opted for two hidden layers comprising 13 and 27 neurons (tuning in the plus-minus range was not expected to have a significant impact). We set $m = 11$ for this experiment and sampled uniform grid points within the given interval. ANN

and analytic solutions are presented in Figure 3. Table 1 contains the numerical values, and Table 2 shows the error resulting from our approach.

4.3. Advantages of using ANN over a large number of data points for solving (9)

We operate uniform grid points of size (m=6, 11, 21, 55, 100) over the domain [0, 2] and [0, 3] to calculate solutions of the ODEs system using the ANN method. As shown by the mean absolute error (MAE) in Table 3, one major advantage of ANN techniques is that they maintain solution accuracy compared to smaller or larger grid points.

Similarly, experiments have been organized for the problems (10)-(13). Using this architecture, we compared the numerical and exact solution in Tables (4)-(11) for each example. We have also compared numerical and exact solution graphically with their error plots in Figure 4 and Figure 5.
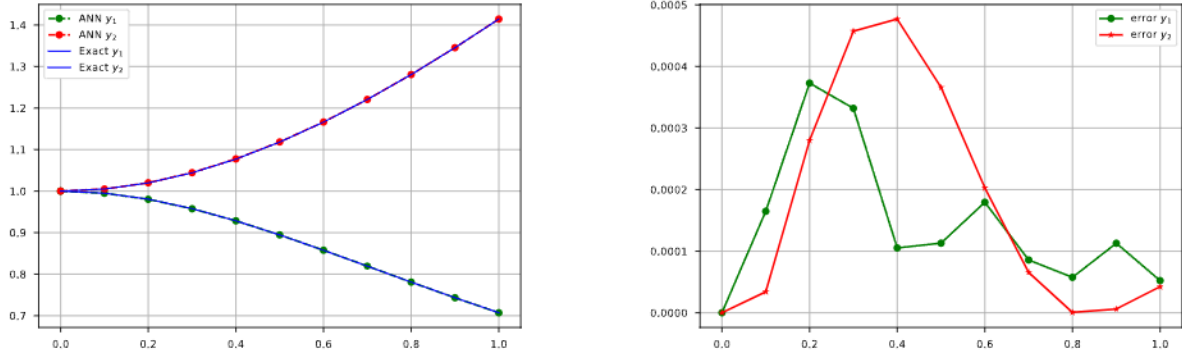
Figure 3. Comparison of ANN and exact solutions for system (9) with error plot.

Table 1. ANN and analytical solutions for system (9).

Grid point	ANN ω_1	Analytic ω_1	ANN ω_2	Analytic ω_2	BCM ω_1 [37]	BCM ω_2 [37]
0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
0.1	0.994872	0.995037	1.005022	1.004988	0.996045	1.005800
0.2	0.980208	0.980581	1.020084	1.019804	0.981491	1.020560
0.3	0.957494	0.957826	1.044488	1.044031	0.958620	1.044698
0.4	0.928371	0.928477	1.077510	1.077033	0.929146	1.077590
0.5	0.894540	0.894427	1.118400	1.118034	0.894959	1.118473
0.6	0.857672	0.857493	1.166393	1.166190	0.857883	1.166515
0.7	0.819318	0.819232	1.220721	1.220656	0.819498	1.220874
0.8	0.780811	0.780869	1.280624	1.280625	0.781042	1.280750
0.9	0.743181	0.743294	1.345368	1.345362	0.743387	1.345412
1.0	0.707055	0.707107	1.414256	1.414214	0.707106	1.414213

Table 2. Computing the absolute difference between ANN and exact solutions for system (9).

Grid point	error ω_1	error ω_2
0.0	0.000000	0.000000
0.1	1.65e-04	3.407319e-05
0.2	3.73e-04	2.797772e-04
0.3	3.32e-04	4.574576e-04
0.4	1.05e-04	4.769526e-04
0.5	1.13e-04	3.659818e-04
0.6	1.79e-04	2.029116e-04
0.7	8.6e-05	6.545180e-05
0.8	5.8e-05	5.426605e-07
0.9	1.13e-004	6.053325e-06
1.0	5.2e-05	4.223276e-05

Example 2. *Let us consider a problem of Emden-Fowler equations documented in [40].*

$$\frac{d^2\omega_1}{d\zeta^2} + \frac{2}{\zeta}\frac{d\omega_1}{d\zeta} = -6(e^{\frac{\omega_2}{3}} + 4)e^{\frac{2\omega_1}{3}},$$
$$\frac{d^2\omega_2}{d\zeta^2} + \frac{2}{\zeta}\frac{d\omega_2}{d\zeta} = 6(e^{\frac{-\omega_1}{3}} + 4)e^{\frac{-2\omega_2}{3}}, \quad (10)$$

with ICs

$$\omega_1(0) = -3\log(2), \omega_2(0) = 3\log(2), \omega_1'(0) = \omega_2'(0) = 0.$$

The exact solutions to the problem are $\omega_1(\zeta) = -3\log(2 + \zeta^2)$ and $\omega_2(\zeta) = 3\log(2 + \zeta^2)$.

Example 3. *Let us consider a problem of Emden-Fowler equations documented in [40].*

$$\frac{d^2\omega_1}{d\zeta^2} + \frac{5}{\zeta}\frac{d\omega_1}{d\zeta} = -8e^{\omega_1} - 16e^{\frac{-2\omega_2}{2}},$$
$$\frac{d^2\omega_2}{d\zeta^2} + \frac{3}{\zeta}\frac{d\omega_2}{d\zeta} = 8e^{-\omega_2} + 8e^{\frac{\omega_1}{2}}, \quad (11)$$
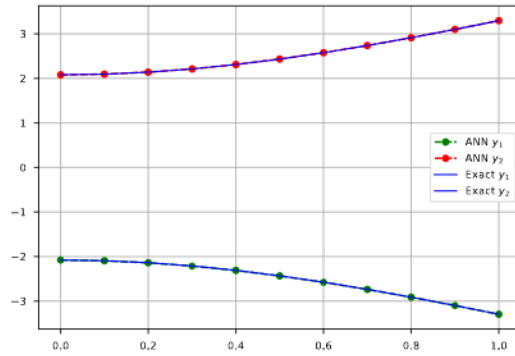
with ICs

$$\omega_1(0) = \omega_2(0) = 0, \omega_1'(0) = \omega_2'(0) = 0.$$

The exact solutions to the problem are $\omega_1(\zeta) = -2\log(1 + \zeta^2)$ and $\omega_2(\zeta) = 2\log(1 + \zeta^2)$.

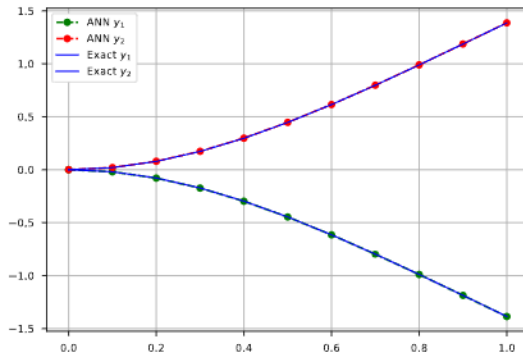**Table 3.** MAE for different grid points over intervals.

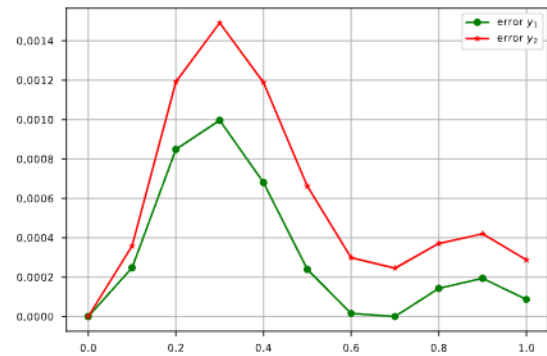| End points	$t \in [0,2]$		$t \in [0,2]$	
Grid point	mae ω_1	mae ω_2	mae ω_1	mae ω_2
6	0.050588	0.051056	0.426025	0.631669
11	0.044180	0.059593	0.405198	0.763566
21	0.045351	0.032434	0.455094	0.323775
55	0.068090	0.065440	0.803243	0.848116
100	0.051830	0.031469	0.511720	0.320044

(a) Comparison of ANN and exact of system (10)
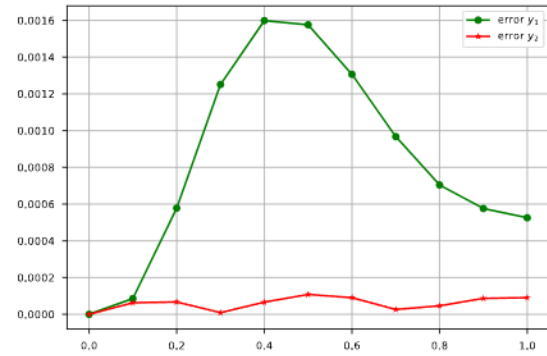
(b) Error plot of system (10)

(c) Comparison of ANN and exact of system (11)

(d) Error plot of system (11)

Figure 4. Comparison of ANN and Exact Solutions of Examples with Error Plot.

Example 4. *Let us consider a problem of Emden-Fowler equations documented in [41].*

$$\frac{d^2\omega_1}{d\zeta^2} + \frac{8}{\zeta}\frac{d\omega_1}{d\zeta} = 4\omega_1 \log(\omega_2) - 18\omega_1,$$
$$\frac{d^2\omega_2}{d\zeta^2} + \frac{4}{\zeta}\frac{d\omega_2}{d\zeta} = 10\omega_2 - 4\omega_2 \log(\omega_1), \quad (12)$$

with ICs

$$\omega_1(0) = \omega_2(0) = 1, \omega_1'(0) = \omega_2'(0) = 0.$$

The exact solutions to the problem are $\omega_1(\zeta) = e^{-\zeta^2}$ *and* $\omega_2(\zeta) = e^{\zeta^2}$.

Example 5. *Let us consider a problem of Emden-Fowler equations documented in [42].*

$$\frac{d^2\omega_1}{d\zeta^2} + \frac{1}{\zeta}\frac{d\omega_1}{d\zeta} = (4\zeta^2 + 5)\omega_1 - \omega_1^2\omega_2,$$
$$\frac{d^2\omega_2}{d\zeta^2} + \frac{2}{\zeta}\frac{d\omega_2}{d\zeta} = (4\zeta^2 - 5)\omega_2 - \omega_1\omega_2^2, \quad (13)$$

with ICs

$$\omega_1(0) = \omega_2(0) = 1, \omega_1'(0) = \omega_2'(0) = 0.$$

The exact solutions to the problem are $\omega_1(\zeta) = e^{\zeta^2}$ *and* $\omega_2(\zeta) = e^{-\zeta^2}$.

Table 4. ANN and analytical solutions for system (10).

Grid point	ANN ω_1	Analytic ω_1	ANN ω_2	Analytic ω_2	HAM ω_1 [37]	ADM ω_1 [37]	HAM ω_2 [37]	ADM ω_2 [37]
0.0	-2.079442	-2.079442	2.079442	2.079442				
0.1	-2.094319	-2.094404	2.094467	2.094404	-2.103387	-2.138529	2.103387	2.138529
0.2	-2.139427	-2.138849	2.138916	2.138849	-2.147076	-2.180143	2.147076	2.180143
0.3	-2.212743	-2.211492	2.211501	2.211492	-2.218576	-2.248419	2.218576	2.248419
0.4	-2.311923	-2.310325	2.310259	2.310325	-2.316029	-2.341807	2.316029	2.341807
0.5	-2.434367	-2.432791	2.432683	2.432791	-2.437034	-2.458259	2.437034	2.458259
0.6	-2.577291	-2.575985	2.575894	2.575985	-2.578826	-2.595353	2.578826	2.595353
0.7	-2.737815	-2.736848	2.736822	2.736848	-2.738459	-2.750419	2.738459	2.750419
0.8	-2.913041	-2.912337	2.912383	2.912337	-2.912983	-2.920675	2.912983	2.920675
0.9	-3.100129	-3.099553	3.099640	3.099553	-3.099603	-3.103358	3.099603	3.103358
1.0	-3.296362	-3.295837	3.295927	3.295837	-3.295836	-3.295836	3.295836	3.295836

Table 5. Computing the absolute difference between ANN and exact solutions for system (10).

Grid point	error ω_1	error ω_2
0.0	0.0000	0.0000
0.1	8.6e-05	6.3e-05
0.2	5.78e-04	6.7e-05
0.3	1.251e-03	9e-06
0.4	1.599e-03	6.6e-05
0.5	1.576e-03	1.08e-04
0.6	1.306e-03	9.1e-05
0.7	9.67e-04	2.6e-05
0.8	7.04e-04	4.6e-05
0.9	5.76e-04	8.7e-05
1.0	5.25e-04	9.1e-05

Table 6. Computing the absolute difference between ANN and exact solutions for system (11).

Grid point	error y_1	error y_2
0.0	0.00000	0.00000
0.1	2.475294e-04	3.58e-04
0.2	8.486477e-04	1.192e-03
0.3	9.964365e-04	1.491e-03
0.4	6.801929e-04	1.189e-03
0.5	2.392059e-04	6.62e-04
0.6	1.564534e-05	2.99e-04
0.7	1.703470e-07	2.45e-04
0.8	1.429000e-04	3.71e-04
0.9	1.941502e-04	4.20e-04
1.0	8.640381e-05	2.87e-04

**Table 7.** Computing the absolute difference between ANN and exact solutions for system (12).

Grid point	error ω_1	error ω_2
0.0	0.000000	0.000000
0.1	1.3e-05	1.72e-04
0.2	2.51e-04	9.3e-05
0.3	4.09e-04	1.0e-04
0.4	4.04e-04	8.4e-05
0.5	2.76e-04	8.1e-05
0.6	1.09e-04	9.9e-05
0.7	1.10e-05	6.2e-05
0.8	3.50e-05	9.0e-05
0.9	3.0e-05	6.8e-05
1.0	1.07e-04	5.0e-05

## 5. Conclusions and future directions

In this paper, we used a vectorized algorithm that employs the ANN method for solving systems of ordinary differential equations (ODEs). We have compared the numerical and exact solution. Results show the stability between target and predicted results, this validates the model. Through various experiments with Python code and accompanying graphical simulations, we gained insight into the nature of the model architecture.
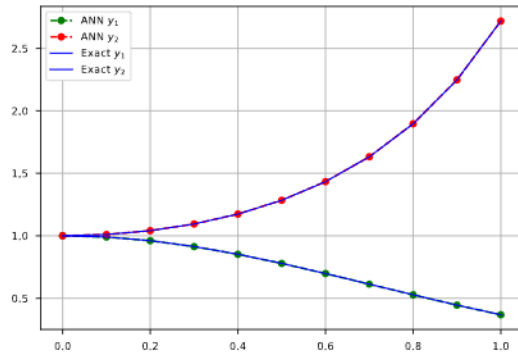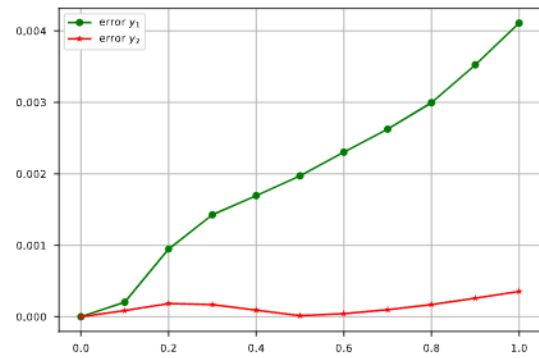
(a) Comparison of ANN and exact of System (12)
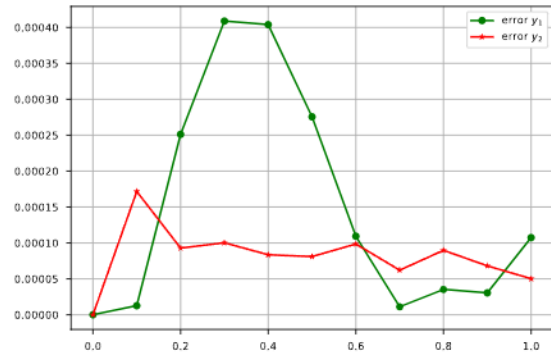
(b) Error plot of System (12)
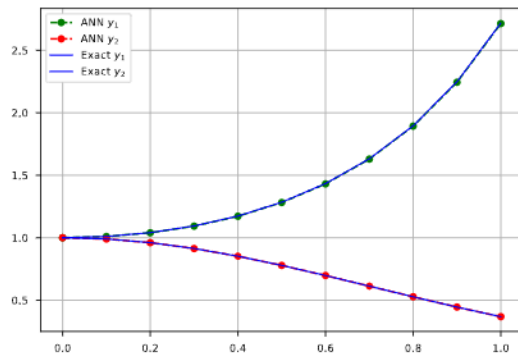
(c) Comparison of ANN and exact of system (13)

(d) Error plot of system (13)

Figure 5. Comparison of ANN and exact solutions of examples with error plot.

Table 8. ANN and analytical solutions for system (11).

Grid point	ANN ω_1	Analytic ω_1	ANN ω_2	Analytic ω_2	HAM ω_1 [40]	HAM ω_2 [40]
0.0	0.000000	-0.000000	0.000000	0.000000		
0.1	-0.020148	-0.019901	0.020259	0.019901	-0.037713	0.054825
0.2	-0.079290	-0.078441	0.079633	0.078441	-0.093487	0.109631
0.3	-0.173352	-0.172355	0.173847	0.172355	-0.183502	0.198153
0.4	-0.297520	-0.296840	0.298029	0.296840	-0.303739	0.316521
0.5	-0.446526	-0.446287	0.446949	0.446287	-0.449283	0.459979
0.6	-0.614954	-0.614969	0.615268	0.614969	-0.614869	0.623398
0.7	-0.797552	-0.797552	0.797797	0.797552	-0.795395	0.801766
0.8	-0.989535	-0.989392	0.989763	0.989392	-0.986339	0.990600
0.9	-1.186848	-1.186654	1.187074	1.186654	-1.184086	1.186262
1.0	-1.386381	-1.386294	1.386582	1.386294	-1.386294	1.386294

Specifically, we found that for specific problems, even a single neuron in the hidden layer can achieve the necessary accuracy. In contrast, more significant numbers of neurons provide greater precision at the cost of increased parameter learning iterations. However, we caution against arbitrary increases in neuron size and recommend selecting an optimal size based on the underlying problem.

This study suggests the possibility of developing neural software that automatically adjusts the number of hidden layers and neurons based on the problem. Future research should focus on conducting additional analytical investigations to enhance the theoretical underpinnings of DNNs

Table 9. ANN and analytical solutions for system (12).

Grid point	ANN ω_1	Analytic ω_1	ANN ω_2	Analytic ω_2
0.0	1.000000	1.000000	1.000000	1.000000
0.1	0.990037	0.990050	1.009879	1.010050
0.2	0.960538	0.960789	1.040718	1.040811
0.3	0.913522	0.913931	1.094274	1.094174
0.4	0.851740	0.852144	1.173594	1.173511
0.5	0.778525	0.778801	1.283944	1.284025
0.6	0.697567	0.697676	1.433231	1.433329
0.7	0.612637	0.612626	1.632378	1.632316
0.8	0.527328	0.527292	1.896571	1.896481
0.9	0.444828	0.444858	2.247840	2.247908
1.0	0.367772	0.367879	2.718332	2.718282

Table 10. ANN and analytical solutions for system (13).

Grid point	ANN ω_1	Analytic ω_1	ANN ω_2	Analytic ω_2
0.0	1.000000	1.000000	1.000000	1.000000
0.1	1.009847	1.010050	0.989964	0.990050
0.2	1.039864	1.040811	0.960605	0.960789
0.3	1.092748	1.094174	0.913761	0.913931
0.4	1.171816	1.173511	0.852052	0.852144
0.5	1.282053	1.284025	0.778786	0.778801
0.6	1.431027	1.433329	0.697719	0.697676
0.7	1.629692	1.632316	0.612724	0.612626
0.8	1.893485	1.896481	0.527462	0.527292
0.9	2.244383	2.247908	0.445117	0.444858
1.0	2.714172	2.718282	0.368234	0.367879

Table 11. Computing the absolute difference between ANN and exact solutions for system (13).

Grid point	error ω_1	error ω_2
0.0	0.000000	0.000000
0.1	2.03e-04	8.6e-05
0.2	9.47e-04	1.84e-04
0.3	1.427e-03	1.70e-04
0.4	1.695e-03	9.2e-05
0.5	1.972e-03	1.5e-05
0.6	2.302e-03	4.3e-05
0.7	2.624e-03	9.8e-05
0.8	2.995e-03	1.70e-04
0.9	3.525e-03	2.59e-04
1.0	4.110e-03	3.54e-04

for solving systems of ODEs, encompassing areas such as delay differential equations and fractional differential equations. Such investigations would involve assessing the consistency, convergence, and suitability of DNNs in the context of solving systems of ODEs.

References

[1] Braun, M., Golubitsky, M. (1983). *Differential Equations and Their Applications* (Vol. 2). New York: Springer-Verlag.

[2] Simmons, G. F. (2016). *Differential Equations with Applications and Historical Notes*. CRC Press.

[3] Chandrasekhar, S., Chandrasekhar, S. (1957). *An Introduction to the Study of Stellar Structure* (Vol. 2). Courier Corporation.

[4] Lane, H. J. (1870). On the theoretical temperature of the sun, under the hypothesis of a gaseous mass maintaining its volume by its internal heat, and depending on the laws of gases as known to terrestrial experiment. *American Journal of Science*, 2(148), 57-74.

[5] Fowler, R. H. (1914). Some results on the form near infinity of real continuous solutions of a certain type of second order differential equation. *Proceedings of the London Mathematical Society*, 2(1), 341-371.

[6] Huang, J. T., Li, J., Gong, Y. (2015, April). An analysis of convolutional neural networks for speech recognition. *In 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 4989-4993). IEEE.

[7] Seifert, C., Aamir, A., Balagopalan, A., Jain, D., Sharma, A., Grottel, S., Gumhold, S. (2017). Visualizations of deep neural networks in computer vision: A survey. *Transparent Data Mining for Big and Small Data*, 123-144.

[8] Dixit, P., Silakari, S. (2021). Deep learning algorithms for cybersecurity applications: A technological and status review. *Computer Science Review*, 39, 100317.

[9] Vigneswaran, R. K., Vinayakumar, R., Soman, K. P., Poornachandran, P. (2018, July). Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security. *In 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)* (pp. 1-6). IEEE.

[10] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv*:1312.6199.

[11] Malladi, S., Sharapov, I. (2018). FastNorm: improving numerical stability of deep network training with efficient normalization.

[12] Zheng, Z., Hong, P. (2018). Robust detection of adversarial attacks by modeling the intrinsic properties of deep neural networks. *Advances in Neural Information Processing Systems*, 31.

[13] Haber, E., Ruthotto, L. (2017). Stable architectures for deep neural networks. *Inverse Problems*, 34(1), 014004.

[14] Balduzzi, D., Frean, M., Leary, L., Lewis, J. P., Ma, K. W. D., McWilliams, B. (2017, July). The shattered gradients problem: If resnets are the answer, then what is the question?. *In International Conference on Machine Learning* (pp. 342-350). PMLR.

[15] Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234, 11-26.

[16] Samek, W., Montavon, G., Lapuschkin, S., Anders, C. J., Müller, K. R. (2021). Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109(3), 247-278.

[17] Björck, A. (1990) Least squares methods. *Handbook of Numerical Analysis*, Edited by P.G. Ciarlet and J.L. Lions, Volume 1, 465–652.

[18] Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2), 164-168.

[19] Fletcher, C. A., Fletcher, C. A. J. (1984). *Computational Galerkin Methods*. Springer Berlin Heidelberg, pp. 72-85

[20] Lagaris, I. E., Likas, A., Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5), 987-1000.

[21] Tan, L. S., Zainuddin, Z., Ong, P. (2018, June). Solving ordinary differential equations using neural networks. *In AIP Conference Proceedings* (Vol. 1974, No. 1). AIP Publishing.

[22] Michoski, C., Milosavljević, M., Oliver, T., Hatch, D. R. (2020). Solving differential equations using deep neural networks. *Neurocomputing*, 399, 193-212.

[23] Han, J., Jentzen, A., E, W. (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34), 8505-8510.

[24] Sabir, Z., Saoud, S., Raja, M. A. Z., Wahab, H. A., Arbi, A. (2020). Heuristic computing technique for numerical solutions of nonlinear fourth order Emden–Fowler equation. *Mathematics and Computers in Simulation*, 178, 534-548.

[25] Sabir, Z., Raja, M. A. Z., Arbi, A., Altamirano, G. C., Cao, J. (2021). Neuro-swarms intelligent computing using Gudermannian kernel for solving a class of second order Lane-Emden singular nonlinear model. *AIMS Mathematics*, 6(3), 2468-2485.

[26] Sabir, Z., Raja, M. A. Z., Alhazmi, S. E., Gupta, M., Arbi, A., Baba, I. A. (2022). Applications of artificial neural network to solve the nonlinear COVID-19 mathematical model based on the dynamics of SIQ. *Journal of Taibah University for Science*, 16(1), 874-884.

[27] Meade Jr, A. J., Fernandez, A. A. (1994). Solution of nonlinear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 20(9), 19-44.

[28] Aarts, L. P., Van Der Veer, P. (2001). Neural network method for solving partial differential equations. *Neural Processing Letters*, 14, 261-271.

[29] Yazdi, H. S., Pakdaman, M., Modaghegh, H. (2011). Unsupervised kernel least mean square

algorithm for solving ordinary differential equations. *Neurocomputing, 74*(12-13), 2062-2071.

[30] Asady, B., Hakimzadegan, F., Nazarlue, R. (2014). Utilizing artificial neural network approach for solving two-dimensional integral equations. *Mathematical Sciences, 8*, 1-9.

[31] Berg, J., Nyström, K. (2018). A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing, 317*, 28-41.

[32] Nascimento, R. G., Viana, F. A. (2020). Cumulative damage modeling with recurrent neural networks. *AIAA Journal, 58*(12), 5459-5471.

[33] Innes, M., Edelman, A., Fischer, K., Rackauckas, C., Saba, E., Shah, V. B., Tebbutt, W. (2019). A differentiable programming system to bridge machine learning and scientific computing. *arXiv preprint arXiv:1907.07587.*

[34] Wang, H., Qin, C., Bai, Y., Zhang, Y., Fu, Y. (2021). Recent advances on neural network pruning at initialization. *arXiv preprint arXiv:2103.06460.*

[35] Dufera, T. T. (2021). Deep neural network for system of ordinary differential equations: Vectorized algorithm and simulation. *Machine Learning with Applications, 5*, 100058.

[36] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research, 18*, 1-43.

[37] Shahni, J., Singh, R. (2021). Numerical solution of system of Emden-Fowler type equations by Bernstein collocation method. *Journal of Mathematical Chemistry, 59*(4), 1117-1138.

[38] Althubiti, S., Kumar, M., Goswami, P., Kumar, K. (2023). Artificial neural network for solving the nonlinear singular fractional differential equations. *Applied Mathematics in Science and Engineering, 31*(1), 2187389.

[39] Panghal, S., Kumar, M. (2022). Neural network method: delay and system of delay differential equations. *Engineering with Computers, 38*(Suppl 3), 2423-2432.

[40] Singh, R., Singh, G., Singh, M. (2021). Numerical algorithm for solution of the system of Emden–Fowler type equations. *International Journal of Applied and Computational Mathematics, 7*(4), 136.

[41] Singh, R. (2018). Analytical approach for computation of exact and analytic approximate solutions to the system of Lane-Emden-Fowler type equations arising in astrophysics. *The European Physical Journal Plus, 133*(8), 320.

[42] Wazwaz, A. M. (2011). The variational iteration method for solving systems of equations of Emden–Fowler type. *International Journal of Computer Mathematics, 88*(16), 3406-3415.

Ashish kumar *received graduation degree in mathematics (Honours) from Ramjas College, University of Delhi, and Msc degree from Hindu College, University of Delhi. Currently, he is a junior research fellow at School of Liberal Studies, Dr. B.R. Ambedkar University Delhi, Delhi. His research interests include numerical analysis and applied mathematics.*
🆔 https://orcid.org/0009-0003-8968-8074

Manoj Kumar *is a visiting faculty member at the Department of Applied Sciences and Humanities, Indira Gandhi Delhi Technical University for Women, Delhi, India. He received a PhD degree in Mathematics from Dr. B. R. Ambedkar University Delhi, India. His research interests include traffic congestion modeling, machine learning, and deep learning.*
🆔 https://orcid.org/0000-0001-8218-8278

Pranay Goswami *is currently an Assistant Professor at the School of Liberal Studies, Dr. B.R. Ambedkar University Delhi. He received PhD degree in Mathematics from University of Rajasthan, India. His research interests include differential equations, fractional differential equations, numerical methods, and mathematical modeling.*
🆔 https://orcid.org/0000-0003-1205-1975