Astronomy and Computing 44 (2023) 100734

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

Modelling Lane–Emden type equations using Physics-Informed Neural Networks

Hubert Baty

Observatoire Astronomique, CNRS UMR 7550, Université de Strasbourg, 67000, Strasbourg, France

ARTICLE INFO

Article history:
Received 2 May 2023
Accepted 4 July 2023
Available online 13 July 2023

Dataset link: https://github.com/hubertbaty/PINNS-LE

Keywords:
Deep learning
Neural networks
Lane–Emden equations
Polytropic and isothermal gas spheres
White dwarf equation

ABSTRACT

The solutions of Lane–Emden (LE) type equations arising in astrophysics for the polytropic gas spheres, the isothermal gas sphere, and the white dwarf stars are revisited. We explore the potentiality of recent methods of deep learning using neural networks constrained by the physics and called Physics-Informed Neural Networks (PINNs). The basics of PINNs is introduced for solving each equation individually. The method consists in constraining the equation residual at some collocation dataset in addition to the boundary data via a minimization procedure. When the training process is complete, a learned differentiable function is obtained that can generate solution at any value of the variable. The novelty of this study is the additional possibility of learning solutions for several equations collectively with the same network, e.g. for the polytropic equations family for all the indices. We demonstrate the performances of PINNs in comparison with classical numerical methods. Advantages and drawbacks are highlighted. Interestingly, PINNs are meshless methods that can quasi-instantaneously generate the solution and its derivative once trained. However, the training procedure and accuracy of the method remain two future points of improvement.

1. Introduction

Machine learning and more precisely deep learning techniques based on Neural Networks (NNs) are actually widely used to solve problems in a variety of domains including computer vision, language processing, game theory, etc. (see Le Cun et al., 2015 and references therein). The idea of applying NNs to solve differential equations was first proposed by Lagaris et al. (1998), but it was not immediately put into practice in the following years due to the lack of computational resources. Leveraging prior knowledge of the physics in the learning process of NNs with the aim to solve partial differential equations (PDEs) was introduced only recently (Raissi et al., 2017, 2019). Benefitting also from technical progress on automatic differentiation and the facilitated use of Python open source software libraries like Tensorflow or Pytorch, promising new methods so-called Physics-Informed Neural Networks (PINNs) were born.

In classical NNs, a supervised approach is generally considered that consists in finding a mapping function between given input objects and their associated output values. This is done by using knowledge about a dataset containing several input/output pairs. This dataset is used to parameterize the NN such that it minimizes the error between solutions predicted by the NN and exact known solutions in the dataset of the so-called training data. The convergence is achieved by minimizing a loss function which expression is a measure of the error (e.g. the mean squared error). In other words, this is an efficient non-linear approximation procedure. However, in cases of PDEs the training dataset is generally drastically reduced to the sole knowledge of the initial/boundary conditions.

PINNs approach consists in enhancing classical NNs by defining some other set of data, called collocation points, at which the estimated solution must additionally ensure the equation. In the original method usually called vanilla-PINN, a second loss function corresponding to the physics is thus defined and added to the previous one in the learning process. The latter optimizes the NN so that the residual of the equation is minimal. In this way, unlike the classical NNs, PINNs does not require a large amount of training data. Moreover, an advantage of this approach is given by the possibility to evaluate exactly the differential operators at the collocation points by using automatic differentiation. Note that many PINNs-variants can be actually found in the literature. In this work, we mainly focus on vanilla-PINNs as introduced originally by Raissi et al. (2017, 2019).

The aim of this work consists in assessing the advantages and drawbacks of PINNs for modelling Lane–Emden (LE) type equations. LE equation is one of the most important classical differential equation of mathematical physics which originally appeared in astrophysics to model the static star structures. Besides its applications in astrophysics, this type of equation is also well known to be useful to simulate more general physics

E-mail address: hubert.baty@unistra.fr.

https://doi.org/10.1016/j.ascom.2023.100734

phenomena. The main difficulty in solving LE equation is due to its singularity at the origin (i.e. at the point $x = 0$, see below). The development of solving techniques has therefore attracted considerable attention from researchers in the last decades. Some methods are analytical ones, numerical ones, or combinations of both. A rather complete summary of the various approaches can be found in He et al. (2020). Classical numerical methods including the traditional Runge–Kutta methods have to circumvent the singularity problem, by using some transformation, Taylor-like expansion, or avoiding the origin. Many other recent approaches have been developed, as for example ones based on Lattice Boltzmann method (Zhang et al., 2003), genetic algorithm or Monte Carlo simulations (El-Essawi et al., 2023 and references therein). Other recent approaches based on artificial neural networks have been brought forward. This is the case of neural networks methods using Chebyshev polynomials (Mall and Chakraverty, 2014), stochastic approach in combination with genetic algorithm (Sabir et al., 2019), and Morlet wavelet neural networks (Sabir et al., 2020). However, the above methods require specific carefully designed basis functions to approximate the solution. In our deep learning PINNs method, neural networks are directly used to solve LE type equations without the need of basis functions.

The paper is organized as follows. We first briefly summarize the different LE differential equations concerned in Section 2. Section 3 presents the basics of PINNs including our proper PINN algorithm for solving differential equations. The results of solving individual and multiple Lane–Emden equations are reported in Sections 4 and 5 respectively. Finally, conclusions are drawn in Section 6.

2. Lane–Emden type equations

In this section, we summarize the different LE type equations considered in this study. Three equations families represented by three different second order differential equations appear below. They are subject to two boundary conditions, namely on the solution and on its first order derivative at the origin. Consequently, the concern of this work is the integration of initial value problems (IVPs) of ordinary differential equations (ODEs). The numerically challenging aspect of the LE equations is due to the singularity at $x = 0$ (see below).

2.1. Polytropic cases

In the theory of stellar structure, the polytropic model of a gas sphere is represented by the following fundamental equation to be solved:

$$\frac{1}{x^2}\frac{d}{dx}\left(x^2\frac{dy(x)}{dx}\right) + y^n = 0, \tag{1}$$

where n is a positive polytropic index. The variable x is a dimensionless radius, and the solution $y(x)$ is a normalized quantity related to the mass density ρ via $\rho = \rho_c y^n$ for the central density ρ_c. The index n comes from the polytropic equation of state $P = K\rho^{(1+1/n)}$, with P the thermal pressure and K a constant of proportionality. More details about obtaining this LE equation starting from a self-gravitating spherically symmetric fluid in hydrostatic equilibrium can be found elsewhere (see for example El-Essawi et al., 2023 and reference therein).

Note that obtaining the solution of Eq. (1) holds under the two above mentioned boundary conditions imposed at $x = 0$, that are $y(0) = 1$ and $\frac{dy(0)}{dx} = 0$. The physically relevant values for n invade the whole range between 0 and $+\infty$ and are not necessarily given by an integer value. Finally, an exact solution exists only for three n values as

$$\begin{aligned} n = 0 &\longrightarrow y(x) = 1 - \tfrac{x^2}{6}, \\ n = 1 &\longrightarrow y(x) = \tfrac{\sin(x)}{x}, \\ n = 5 &\longrightarrow y(x) = \tfrac{1}{\sqrt{1+x^2/3}}. \end{aligned} \tag{2}$$

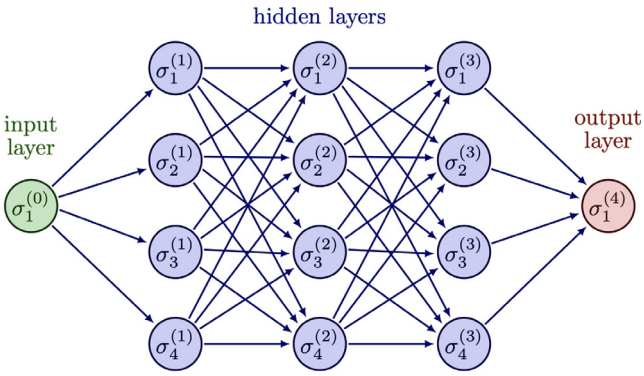

Fig. 1. Schematic representation of the structure for a classical Neural Network example having 3 hidden layers with 4 neurons per layer, one input layer (single neuron), and one output layer (single neuron). In the context of a single ODE, the input neuron noted $\sigma_1^{(0)}$ can be the x variable, and the output one $\sigma_1^{(0)}$ can therefore be the approximated solution $y_\theta(x)$.

2.2. Isothermal case

The isothermal model can be deduced from the previous equation as a special case setting $n \longrightarrow +\infty$. The finally obtained equation to be solved is thus (El-Essawi et al., 2023),

$$\frac{1}{x^2}\frac{d}{dx}\left(x^2\frac{d\tilde{y}(x)}{dx}\right) - e^{-\tilde{y}(x)} = 0, \tag{3}$$

where now $\tilde{y}$ is defined via $\rho = \rho_c e^{-\tilde{y}}$. Another equivalent equation is generally considered in the literature. Indeed, using the transform $\tilde{y} \longrightarrow -\tilde{y} = y$, we have

$$\frac{1}{x^2}\frac{d}{dx}\left(x^2\frac{dy(x)}{dx}\right) + e^{y(x)} = 0, \tag{4}$$

which is subject to the boundary conditions at $x = 0$, $y(0) = 0$ and $\frac{dy(0)}{dx} = 0$.

2.3. White dwarf cases

The white dwarf equation was initially derived by Chandrasekhar (1958) as

$$\frac{1}{x^2}\frac{d}{dx}\left(x^2\frac{dy(x)}{dx}\right) + (y^2 - C)^{3/2} = 0, \tag{5}$$

where C take constant values in the range $0.01 - 0.8$. Note that if $C = 0$, the latter equation reduces to the polytropic LE equation of index $n = 3$. This problem is subject to the two boundary conditions at $x = 0$, $y(0) = 1$ and $\frac{dy(0)}{dx} = 0$.

3. The basics of PINNs

In this section, we first introduce the basic concepts underlying the approximation of a desired function with NNs. The particularity of the physics constraint that is embedded into the NN and our proper PINN algorithm are detailed after.

3.1. The basics of NNs for non linear approximation

We consider a desired function $y(x)$ that could be (see below) the solution on an ODE equation with y_θ being the approximated solution at different x values, and where θ is a set of model parameters. Using a classical neural network, we can write

$$y_\theta(x) = (\mathcal{N}_L \circ \mathcal{N}_{L-1} \ldots \mathcal{N}_0)(x), \tag{6}$$

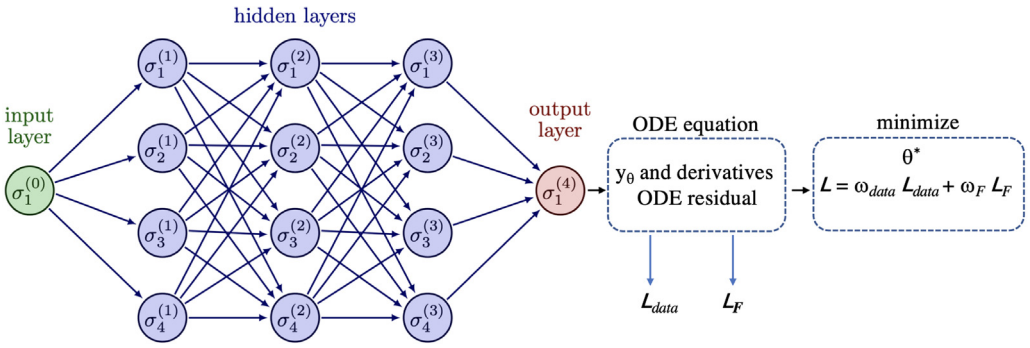

Fig. 2. Schematic representation of network structure for a vanilla-PINN modelling an ODE. The previous NN architecture (see previous figure) is used to evaluate the residual of the ODE equation (via y_θ and associated first and second order derivatives). Two partial loss functions are used to form a total loss function with associated weights (see text) that is finally minimized.

where the operator $\circ$ denotes the composition and $\theta = \{\boldsymbol{W}_l, \boldsymbol{b}_l\}_{l=1,L}$ represents the trainable parameters (weight matrices and bias vectors) of the network. The latter is composed of $L+1$ layers including $L-1$ hidden layers of neurons, one input layer, and one output layer. The input layer is composed of one or two neurons in this work, and it represents variables including the normalized radius x (first neuron) and also eventually a variable parameter (second neuron). For the sake of illustration, we consider only the single input neuron in this subsection. The output layer with a single neuron represents the predicted solution $y_\theta(x)$. In other words, the solution is written as a sequence of non linear functions as for the hidden layers ($1 \leq l \leq L-1$), we have

$$\mathcal{N}_l(x) = \sigma(\boldsymbol{W}_l \mathcal{N}_{l-1}(x) + \boldsymbol{b}_l), \tag{7}$$

where we denote the weight matrix and bias vector in the lth layer by $\boldsymbol{W}_l \in \mathbb{R}^{d_{l-1} \times d_l}$ and $\boldsymbol{b}_l \in \mathbb{R}^{d_l}$ (d_l being the dimension of the input vector for the lth layer). $\sigma(.)$ is a non linear activation function, which is applied element-wisely. Such activation function allows the network to map nonlinear relationship that is fundamental for automatic differentiation and therefore the calculation of the derivatives (see below). In this work, I choose the most commonly used hyperbolic tangent *tanh* function.

The optimization problem aiming to find a non linear approximation $y_\theta(x) \simeq y(x)$ is based on the minimization of a function $\mathcal{L}_{data}$, called loss function, that can be expressed using a mean squared error formulation as

$$\mathcal{L}_{data}(\theta) = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} \left| y_\theta(x_i) - y_i^{data} \right|^2 . \tag{8}$$

The latter expression assumes that a set of N_{data} data is available for $y(x)$ taken at different x_i, i.e. input/output pairs (x_i, y_i^{data}) are known that are generally called training data in the literature. For the purely non linear approximation problem, the loss function $\mathcal{L}$ is simply $\mathcal{L}_{data}$. Finally, a gradient descent algorithm is used until convergence towards the minimum is obtained for a predefined accuracy (or a given maximum iteration number) as

$$\theta_{i+1} = \theta_i - \eta \nabla_\theta \mathcal{L}(\theta_i), \tag{9}$$

for the ith iteration also called epoch in the literature, leading to $\theta^* = \text{argmin}_\theta \mathcal{L}(\theta)$, where η is known as the learning rate parameter. This is the so-called training procedure. A schematic of a standard NN is shown in Fig. 1. In this work, we choose the well known *Adam* optimizer. The standard automatic differentiation technique is necessary to compute derivatives (i.e. ∇_θ) with respect to the NN parameters (e.g. weights and biases).

The goal is to calibrate the trainable parameters θ (weight matrices and bias vectors) of the network such that $y_\theta(x)$ approximates the target solution $y(x)$. However, in cases of differential equations the training dataset is generally drastically reduced to the sole knowledge of the initial/boundary conditions.

3.2. The basics of PINNs for solving an ODE

In order to tackle the limitation due to the reduced training dataset, PINNs approach considers a second dataset called collocation data aiming at evaluating the residual of the equation written as

$$\mathcal{F}\left[x, y(x), y_x(x), y_{xx}(x)\right] = 0, \qquad x \in [0, D], \tag{10}$$

for second order ODE as considered in this work. We use the notation $y_x = \frac{dy}{dx}$, and $y_{xx} = \frac{dy_x}{dx}$, D defining the right boundary of the integration interval which may vary from case to case in this work.

Consequently, a second loss function associated to the physics (i.e. the equation) can be defined as

$$\mathcal{L}_{\mathcal{F}}(\theta) = \frac{1}{N_c} \sum_{j=1}^{N_c} \left| \mathcal{F}[x_j, y_\theta(x_j), y_{x,\theta}(x_j), y_{xx,\theta}(x_j)] \right|^2 , \tag{11}$$

that must be evaluated at a set of N_c data points located at x_j (generally called collocation points, $j \in [1, N_c]$). Note that, the collocation data are not necessarily coinciding with the training data. As an important property characterizing PINNs, the derivatives of the expected solution with respect to the variable x (i.e. the NN input) needed in the previous loss function are obtained via the automatic differentiation, avoiding truncation/discretization errors inevitable in traditional numerical methods.

In the vanilla-PINN framework, a new total loss function $\mathcal{L}$ is defined that take into account the two previous partial loss functions as

$$\mathcal{L}(\theta) = \omega_{data} \mathcal{L}_{data}(\theta) + \omega_{\mathcal{F}} \mathcal{L}_{\mathcal{F}}(\theta), \tag{12}$$

where weights (also called hyper-parameters) $(\omega_{data}, \omega_{\mathcal{F}})$ are introduced in order to ameliorate the eventual unbalance between the two partial losses during the training process (see Fig. 2). These weights and the learning rate can be user-specified or automatically tuned. In the present work, for simplicity we fix the ω_{data} value to be constant and equal to unity, and the other weight parameters are determined with values varying from case to case. More technical details about the PINNs methods can be found elsewhere (see Baty and Baty, 2023 and references therein).

3.3. Our PINN algorithm

Taking into account the initial/boundary conditions can be done in different ways in the PINNs variants found in the literature. In the vanilla-PINNs, the condition $y(x=0) = y_0$ is generally imposed through the training dataset via $\mathcal{L}_{data}(\theta)$ as a soft constraint, thus $N_{data} = 1$ (in the single input case), contrary to other PINNs variants where this condition is incorporated into

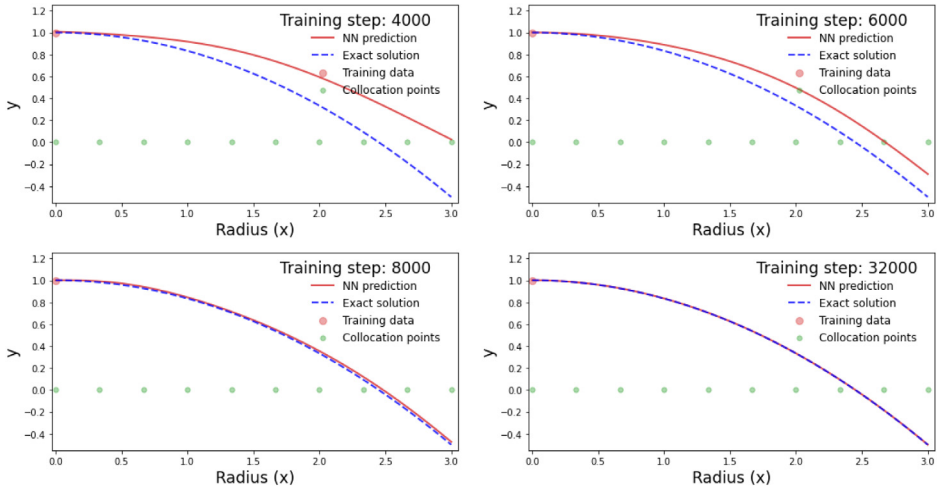

Fig. 3. Snapshots taken at different epochs (training steps) of the predicted PINN solution versus the exact one for the $n = 0$ polytropic LE equation. The radius x values (for collocation points) at which the physical loss function is evaluated are indicated with the small green circle on x axis.

$\mathcal{L}_{\mathcal{F}}(\theta)$ thus as a hard constraint. In this work we follow the first option. Moreover, taking into account the first derivative at $x = 0$, that is $y_x = 0$ in this study, must be also done. For our PINN algorithm we have found efficient to use the first collocation point at $x_1 = 0$ in order to constrain the condition on $y_{x,\theta}(x_1)$ to be close to zero (as required for LE boundary conditions), thus constituting a second soft constraint.

Consequently, we define a specific loss data $\mathcal{L}_{data}$, called loss function, that can be expressed using a mean squared error formulation as

$$\mathcal{L}_{data}(\theta) = \left| y_\theta(x_1) - y_1^{data} \right|^2 + \omega_d \left| y_{x,\theta}(x_1) \right|^2, \tag{13}$$

with $y_1^{data} = y_0$ (with a value of 0 or 1 in this work) for our LE equations. We have also introduced a new weight parameter (ω_d) for the derivative. In other words, a hybrid data loss function is used. Other options exist like solving an equivalent system of two first-order differential equations instead of a single second order one, as for a standard analytical approach (He et al., 2020). Nevertheless, as shown by our results, our method is efficient. One must note that the singularity at the origin ($x = 0$) is consequently handled by using the two soft constraints together with the residual equation form (see just below).

As explained above (see Eq. (12)), a total loss function is finally assembled and used into the gradient descent algorithm (see Eq. (9)). A summary of the different parameters involved in our algorithm and their associated explored values are reported in Table 1. The other parameters are fixed to $N_{data} = 1$ (except for Section 5 where a second input variable is considered), and $\omega_{data} = 1$. For the sake of simplicity, the distribution of the collocation points is chosen to be uniform over the integration interval. This choice is not fundamental for this study as the solutions are relatively smooth.

Table 1
Algorithm parameters and typical values investigated in this work.

Name	
Number of hidden layers	$1 \leq L - 1 \leq 5$
Number of neurons per layer	$5 \leq N_n \leq 50$
Learning rate	$1 \times 10^{-4} \leq \eta \leq 1 \times 10^{-3}$
Weight for derivative	$1 \times 10^{-2} \leq \omega_d \leq 1 \times 10^{-1}$
Weight for physics	$1 \times 10^{-3} \leq \omega_{\mathcal{F}} \leq 1 \times 10^{-1}$
Number of collocation points	$5 \leq N_c \leq 50$

4. Solving individual Lane–Emden type equations with PINNs

4.1. Polytropic cases

The previous polytropic LE equation can be re-written in the following equivalent form

$$x\frac{d^2y(x)}{dx^2} + 2\frac{dy(x)}{dx} + xy^n = 0, \tag{14}$$

that is the residual expression effectively used and minimized in our PINN algorithm. In this way, the singularity at the origin which generally poses numerical difficulties in traditional discretization schemes disappears. The convergence towards the solution is illustrated in Figs. 3–4 for the $n = 0$ case for which an analytical solution exists (see Eq. (2)). We have considered the integration over the radius interval [0, 3] that encompasses the physically relevant values for this index value (as positive y values are required). Indeed, the different snapshots taken at different epochs show the progression of the training process that is stopped after 32000 steps. Note that a very modest number of collocation points (here $N_c = 10$) is enough to obtain an average absolute error (square root of the MSE deduced from Fig. 4) significantly smaller than 10^{-3}. The MSE is evaluated using the standard expression, MSE $= \frac{1}{N_{eval}} \sum_{i=1}^{N_{eval}} \left| y_\theta(x_i) - y_i^{eval} \right|^2$, where the evaluation $y_\theta(x_i)$ is done on $N_{eval} = 300$ points uniformly distributed within the whole space interval, and y_i^{eval} is the expected exact solution at $x = x_i$. This dataset introduced to test the accuracy of the method must not be confused with the collocation dataset at which the loss function is evaluated and used to make the progress of the training. We have also checked that the maximum absolute error is relatively very small, that is approximately 5×10^{-4}. The maximum absolute error is defined as the maximum absolute value of the difference between the expected exact solution and the predicted solution evaluated on the N_{eval} points distributed within the whole space interval. These results have been obtained using the following combination of parameters, $\eta = 1 \times 10^{-4}$, $\omega_{\mathcal{F}} = 2 \times 10^{-2}$, and $\omega_d = 1 \times 10^{-2}$. The architecture of the NN is composed of 2 hidden layers with 20 neurons for each layer.

A similar behaviour is obtained for the other polytropic index values. This is illustrated in Figs. 5 and 6 for the cases $n = 1$

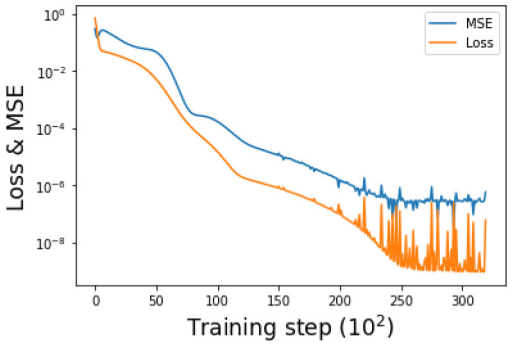

Fig. 4. Histories of the total loss function $L(\theta)$ and MSE during the training process corresponding to the $n = 0$ polytropic case (previous figure). The MSE is evaluated using 300 points uniformly distributed within the whole x interval.

and $n = 5$ respectively for which analytical solutions also exist (see Eq. (2)), where the last epoch for the solution and Loss/MSE histories is plotted. The same combination of parameters and NN architecture is also taken, except that the number of collocation points is now $N_c = 20$ in order to take into account of the larger integration intervals that are $[0, 7]$ and $[0, 10]$ for $n = 1$ and $n = 5$ cases respectively. We have checked that the average and maximum absolute error remains again slightly smaller than 10^{-3} over the integration intervals. The other n values for which analytical solution does not exist can also be obtained in the same way. A general view of the solutions obtained for different n values is available in the following section, where multiple LE equations are integrated using a single NN.

A natural question that arises about these results concerns the effect of varying the different parameters, i.e. more precisely the architecture of the network and the number of collocation points N_c. The effect of varying the learning rate and the weights is not negligible but are less determinant. Indeed, for example taking a too high learning rate is known to lead to strong oscillations in the loss due to jumps over minima, while a too low rate will take very long convergence.

Convergence studies have been performed with the number of collocation points, number of neurons (for a fixed number of layers), and number of layers (for a fixed number of neurons per layer), with the aim to examine the influence on the precision. Note that we can evaluate the maximum absolute error on the solution y itself but also on the derivative y_x, as the latter can be directly deduced via automatic differentiation from the predicted solution. The exact solution first order derivative is $y_x(x) = \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2}$ for the $n = 1$ case. The results that are reported below are obtained for the $n = 1$ case, but they are similar for other n values (more precisely they are slightly better for the $n = 0$ case). First, one can see in Table 2 that a modest number of collocation points (i.e. 12) already allows an acceptable solution. Increasing N_c ameliorates significantly the accuracy with a maximum absolute error being 3×10^{-5} when N_c is of order 30. Increasing further N_c has no additional amelioration. This is not completely surprising as the maximum accuracy expected using such minimization procedure (i.e. with a gradient descent algorithm) is known to be limited by residual oscillations around the minimum. A precision situated between the machine precision and its square root value is thus expected (Press et al., 2007). Note that as in most of machine learning studies, we use the standard version of the Pytorch library that works in single precision. The effect of the architecture of the network on the accuracy is less decisive, except that a minimum number of layers and neurons is preferable, as one can see in Tables 3–4 (with N_n and L varying between 5 and 50, and 2 and 6 respectively).

Table 2
Convergence with the number of collocation points N_c, for a NN architecture having 2 hidden layers and 20 neurons per layer. Polytropic $n = 1$ LE equation.

N_c	Error on y	Error on y_x
12	1.5×10^{-2}	2.7×10^{-2}
14	5×10^{-3}	1×10^{-2}
16	2.5×10^{-3}	5×10^{-3}
20	4×10^{-4}	4×10^{-4}
24	8×10^{-5}	2×10^{-4}
32	3×10^{-5}	1.5×10^{-4}
40	3×10^{-4}	2×10^{-4}
48	1.5×10^{-4}	1×10^{-4}

Table 3
Convergence with the number of neurons N_n, for a NN architecture having 2 hidden layers and $N_c = 32$ collocation points. Polytropic $n = 1$ LE equation.

N_n	Error on y	Error on y_x
5	2.5×10^{-3}	5×10^{-3}
10	1×10^{-3}	3×10^{-3}
15	4×10^{-4}	1.2×10^{-3}
20	3×10^{-5}	1.5×10^{-4}
25	2×10^{-4}	6×10^{-4}
30	2×10^{-4}	5×10^{-4}
35	3×10^{-3}	9×10^{-3}

Table 4
Convergence with the number of hidden layers N_h, for a NN architecture having 20 neurons per layer and $N_c = 32$ collocation points. Polytropic $n = 1$ LE equation.

N_h	Error on y	Error on y_x
1	6×10^{-4}	2×10^{-3}
2	3×10^{-5}	1.5×10^{-4}
3	6×10^{-5}	2×10^{-4}
5	1.3×10^{-4}	4×10^{-4}

For completeness, we can have a look on the error distribution of the absolute error over the integration domain for the solution and its first order derivative, for the combination parameters $N_c = 32$, $N_h = 2$, and $N_n = 20$. The results are plotted in Fig. 6. The errors are not necessarily increasing with the radius, and the error on first derivative is only slightly higher compared to the solution one, especially close to $x = 0$.

Finally, we have made a close comparison of our algorithm with two other integrations techniques, namely Monte Carlo and Chebyshev Neural Network methods. The results are detailed in the appendices, and show that PINNs method has very good accuracy properties especially when considering the rather modest number of collocation points.

4.2. Isothermal case

As for the isotropic equation, the previous isothermal equation considered (Eq. (3)) can be written as

$$x\frac{d^2y(x)}{dx^2} + 2\frac{dy(x)}{dx} - xe^{-y(x)} = 0. \tag{15}$$

There is no exact solution, but an approximate one can be found and expressed as

$$y(x) \simeq \frac{1}{6}x^2 - \frac{1}{5 \times 4!}x^4 + \frac{8}{21 \times 6!}x^6 - \frac{122}{81 \times 8!}x^8, \tag{16}$$

by using series expansion (Iacono and De Felice, 2014).

Our PINN algorithm is also able to find an accurate solution $y_\theta(x)$ as one can see in the results plotted in Fig. 7. This has been obtained using only $N_c = 6$ collocation points with the

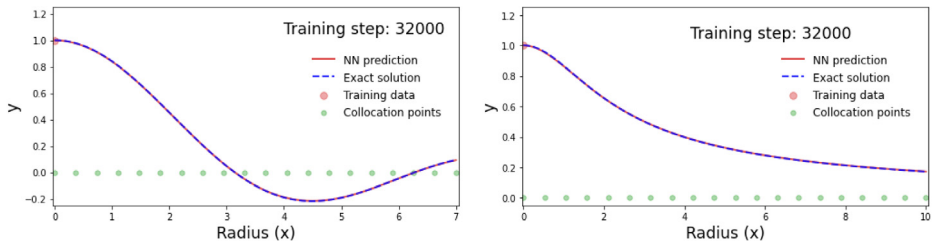

Fig. 5. Predicted PINN $y_\theta(x)$ solutions versus the exact ones obtained at the end of the training for the $n = 1$ and $n = 5$ polytropic LE equations in left and right panel respectively.

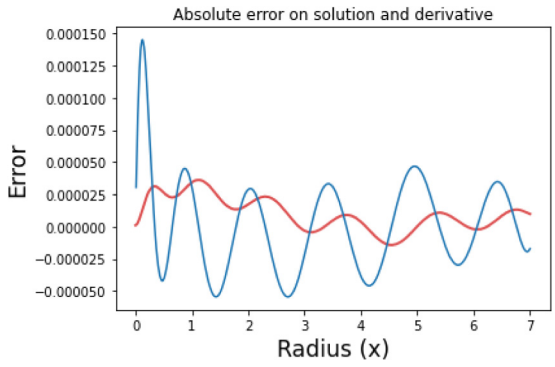

Fig. 6. Absolute error on the solution $y_\theta(x)$ (red) and first order derivative (blue) for the polytropic $n = 1$ LE case. See text for the exact combination of chosen parameters.

combination of parameters, $\eta = 1 \times 10^{-4}$, $\omega_{\mathcal{F}} = 2 \times 10^{-2}$, and $\omega_d = 1 \times 10^{-2}$. The NN is chosen to have 2 hidden layers with 20 neurons for each layer.

4.3. White dwarf cases

As for the previous LE equations, the previously considered white dwarf equation can be written as

$$x\frac{d^2y(x)}{dx^2} + 2\frac{dy(x)}{dx} + x(y^2 - C)^{3/2} = 0. \tag{17}$$

There is no exact solution in this case. During our PINN training process, negative values of $y^2 - C$ can be generated leading to non defined non-integer power exponent. Thus the previous equation is advantageously replaced by,

$$x\frac{d^2y(x)}{dx^2} + 2\frac{dy(x)}{dx} + x\left|y^2 - C\right|^{3/2} = 0, \tag{18}$$

for the residual form to be minimized in our procedure. The results obtained for different values of the C parameter are presented in the following section, as instead of solving individual problem for each C value separately, it is also possible to train a NN to solve a collection of solutions for multiple equations and then predict any solution $y_\theta(x)$ corresponding to different C values.

5. Solving multiple Lane–Emden type equations with PINNs

In the previous section we have focused on solving LE equations individually. For each equation, a dedicated NN was therefore trained and the corresponding trainable parameters definitively calibrated at the end of the PINN training process. In this section, we show that a collection of several equations can also be solved using a similar PINN algorithm. The idea is to consider one parameter defining a particular equation belonging to one family (e.g. the n index for polytropic equations) as an input variable for the NN. Indeed, the PINN algorithm would be able to learn a collection of solutions for several equations corresponding to the different n values situated in an interval $n \in [n_{inf}, n_{sup}]$.

5.1. Polytropic family

We build a new NN architecture having two neurons in the input layer, the first one for the x radius variable and the second neuron representing the polytropic n index value. The output layer/neuron can be now written as $y_\theta(x, n)$.

We first used a first PINN algorithm solving LE equations for x varying in the range $x \in [0, 7]$ and $n \in [0, 5]$. A corresponding set of uniformly distributed collocation data is taken with 32 points for x and 6 points for n (i.e. $n = 0, 1, 2, 3, 4$, and 5). Thus, $N_c = 32 \times 6$. The training dataset representing the boundary conditions for the solution at $x = 0$ is now $N_{data} = 1 \times 6$. The results at the end of the training process are plotted in Fig. 8. The chosen combination of parameters is, $\eta = 1 \times 10^{-4}$, $\omega_{\mathcal{F}} = 8 \times 10^{-2}$, and $\omega_d = 1 \times 10^{-2}$. The NN is chosen to have 4 hidden layers (two times larger than for individual PINN modelling) with 20 neurons for each layer. First, Fig. 8 shows that the $n = 0$, 1 and 5 values that are learned by the network are perfectly superposed with the exact solutions over the whole radius range. Second, a purely predicted solution can be obtained for intermediate not learned n values (as for $n = 2.5$ plotted in the figure). The latter can be quasi-instantaneously generated once the training process is finished. However, note that predicting solution for a specific n value between 0 and 1 would require more collocation points than only 6 because the solution drastically changes between $n = 0$ and $n = 1$.

In a second PINN algorithm, only 4 n values (i.e. $n = 1, 2, 3$, and 4) instead of 6 are now considered in the learning process, all the different parameters being the same compared to the ones used just above. The results plotted in Fig. 9, show that the learned $n = 1$ solution is nicely predicted. As an interesting result, even for a not learned n index (value situated outside the range of trained n indices) the purely predicted $n = 5$ solution compares rather well to the exact solution. Such PINNs methods are not expected to be good extrapolators.

5.2. White dwarf family

We follow the same idea as the one developed in the previous subsection by considering a second input variable that is the coefficient C. Indeed, in this way another single PINN algorithm can be designed in order to learn multiple solutions $y_\theta(x, C)$ of the white dwarf family that is represented by Eq. (18).

We use a PINN algorithm for x varying in the range $x \in [0, 5]$ and C varying in the range $C \in [0, 0.8]$. A corresponding set of collocation data with uniformly distributed $N_c^1 = 28$ points is taken for x and $N_c^2 = 5$ points for C (i.e. $C = 0, 0.2, 0.4, 0.6$, and 0.8). The results at the end of the training process are plotted in Fig. 10. The chosen combination of parameters is, $\eta = 1 \times 10^{-4}$,

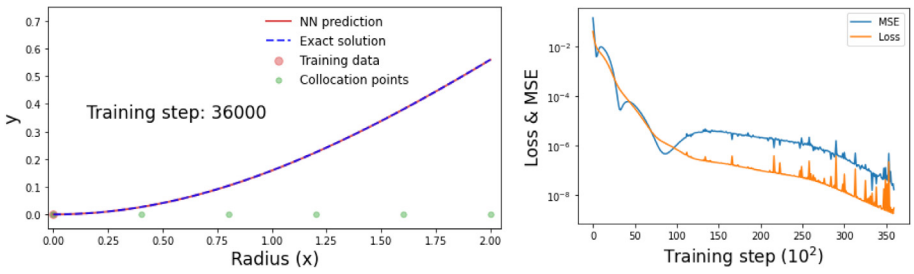

Fig. 7. Predicted PINN solution $y_\theta(x)$ versus the expected one (given by Eq. (16)) obtained at the end of the training for the isothermal LE equation in left panel, and corresponding histories of the loss function and MSE as function of epochs during training in right panel.

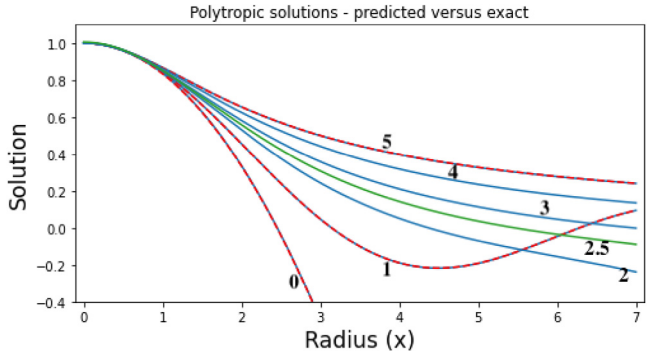

Fig. 8. Predicted polytropic PINN solutions for different n indices as functions of radius. The 6 different learned solutions (see text) are plotted in blue, and compared to exact solutions (in dashed red line) for $n = 0, 1$ and 5. A purely predicted $n = 2.5$ solution in plotted in green.

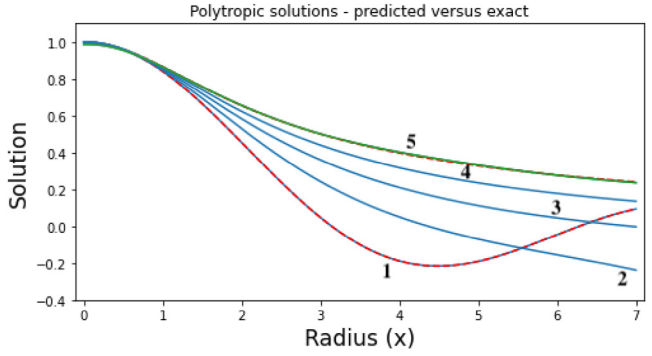

Fig. 9. Same as previous figure for a second PINN algorithm trained for 4 n indices (between 1 and 4). A purely predicted $n = 5$ solution in plotted in green. and compared to the exact solution (dashed red).

$\omega_{\mathcal{F}} = 8 \times 10^{-2}$, and $\omega_d = 1 \times 10^{-2}$. The NN is chosen to have 4 hidden layers (two times larger than for individual PINN modelling) with 20 neurons for each layer.

We have checked that the predicted solutions obtained agree very well with published tabulated values obtained with different methods (El-Essawi et al., 2023). This is the case for solutions corresponding to learned C values but also to non learned ones that are quasi-instantaneously generated once the training process in finished. The measured maximum absolute error is typically close to 10^{-4}. Smaller errors of order 10^{-5} can be also reached but would require a more fine tuning of the different parameters of the network.

6. Conclusions

We show that PINNs are promising numerical tools for solving Lane–Emden type and other similar differential equations.

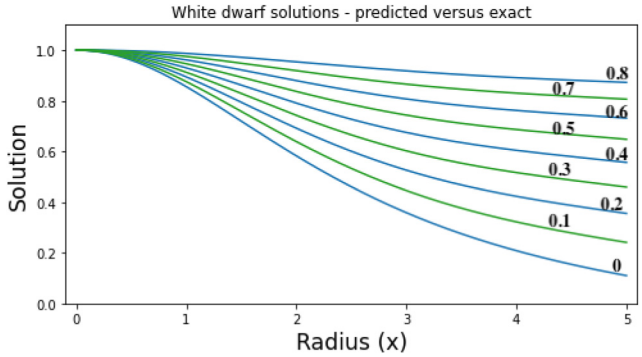

Fig. 10. Predicted white dwarf solutions for different C values as functions of radius. The 5 different learned solutions (see text) are plotted with blue colour, and 4 purely predicted solutions for $C = 0.1, 0.3, 0.5$, and 0.7 are plotted using green colour.

Compared to traditional numerical methods, they present some advantages listed below.

1. They are meshless methods in the sense that the solution is not computed on a given grid. Indeed, a dataset of collocation points is introduced with the aim to minimize the equation residual, but the solution can be directly evaluated (via a learned function) at any variable points (i.e. x value). The required number of collocation points is very reduced compared to classical integration schemes for a similar accuracy.
2. Once trained, the solution and derivatives can be quasi-instantaneously generated in the trained spatial domain. The solution obtained with our method is valid over the entire domain without the need for interpolation (unlike RK tabular solutions).
3. Trial functions carefully designed according to the initial conditions are not needed as in other approaches based on neural networks (see discussion in He et al., 2020), and the activation function is user-defined.
4. Multiple solutions can be learned with the same NN by considering a parameter defining the individual equation as a variable input (like the polytropic index n or the constant C for the polytropic LE and white dwarf equations respectively). In this way, the solution of a given individual equation is also quasi-instantaneously generated if the parameter lies in the range of trained values.
5. The formulation based on the equation residual (that is a second order derivative form) does not require the use of some equivalent system of two first order differential equations. The solution derivative (with respect to the x variable) is also quasi-instantaneously obtained with an accuracy similar to the solution.

However, our results also highlight some drawbacks listed below.

1. The training process depends on a combination of many parameters like, the learning rate, the weights in the loss function, and the architecture of the network, which determines the efficiency of the minimization. Consequently, a fine tuning in order to find optimal parameter values is quite complex, and can therefore be computationally expensive.
2. Even if the accuracy obtained in this work is excellent, PINNs seem to be potentially less accurate than classical methods where for example refining a grid (e.g. Runge–Kutta schemes) allows a precision close to the machine one. This limitation is partly inherent to minimization techniques.

Anyway, PINNs are promising tools that are called upon to develop in future years. For example, ameliorations using self-adaptive techniques are expected in order to improve the previously cited drawbacks (Karniadakis et al., 2021; Cuomo et al., 2022). They also offer a different and complementary approach to traditional methods. They can be also used in different ways in association with differential equations, as with the aim to discover some unknown parameters (i.e. to discover the physics). In this work, we focus on second order LE type equations. In future work, it would be also of interest to generalize our PINNs method in order to investigate solutions of higher order differential equations having multiple singularities (Sabir et al., 2022).

CRediT authorship contribution statement

Hubert Baty: Conceptualization, Methodology, Software, Writing – original draft, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The Pytorch-Python codes used in this work will be made available after acceptance of the paper on the GitHub repository at https://github.com/hubertbaty/PINNS-LE.

Acknowledgements

The author thanks Emmanuel Franck, Victor Michel-Dansac, and Vincent Vigon (IRMA, Strasbourg), for associating him to the supervision of the Master2 internship of Vincent Italiano in February–July 2022, which also gave him want to learn the PINNs technique. The author also thanks the anonymous referee for helpful comments that improved the quality of the manuscript.

Appendix A. Comparison to Monte-Carlo method

We compare our results using PINNs with results obtained using a Monte-Carlo (MC) method that are published in El-Essawi et al. (2023). Indeed we consider integrations over the x intervals [0, 2.4], [0, 3], and [0, 6], for the three polytropic cases with index $n = 0$, 1 and 3 respectively. In our PINN computations we choose a combination of parameters that give results close to optimal ones. The MC reported results use up to 10^6 samples. We use the data shown in Tables A.1, A.2, and A.3 of El-Essawi et al. (2023) to deduce the absolute error at different x values. The comparisons are reported in Tables 5–7.

One can see that our results have a better accuracy by one order of magnitude.

Table 5
Comparison for n = 0 at different radius values.

x	PINN error	MC error
0	1.4×10^{-6}	0.0×10^{0}
0.3	1.6×10^{-5}	1×10^{-4}
0.6	8.0×10^{-6}	2×10^{-4}
0.9	1.8×10^{-5}	3×10^{-4}
1.2	1.1×10^{-5}	4×10^{-4}
1.5	1.4×10^{-5}	5×10^{-4}
1.8	1.7×10^{-5}	9×10^{-4}
2.1	1.1×10^{-5}	7×10^{-4}
2.4	1.3×10^{-5}	8×10^{-4}

Table 6
Comparison for n = 1 at different radius values.

x	PINN error	MC error
0	3.1×10^{-6}	0.0×10^{0}
0.4	3.3×10^{-5}	1×10^{-4}
0.8	2.7×10^{-5}	3×10^{-4}
1.2	1.9×10^{-5}	4×10^{-4}
1.6	2.5×10^{-5}	4×10^{-4}
2.0	1.1×10^{-5}	4×10^{-4}
2.4	1.3×10^{-5}	3×10^{-4}
2.8	3.3×10^{-5}	3×10^{-4}

Table 7
Comparison for n = 5 at different radius values.

x	PINN error	MC error
0	2.1×10^{-5}	0.0×10^{0}
1	4.7×10^{-6}	2×10^{-4}
2	2.1×10^{-5}	1×10^{-4}
3	2.2×10^{-5}	3×10^{-4}
4	2.1×10^{-5}	3×10^{-4}
5	2.4×10^{-5}	7×10^{-4}
6	1.5×10^{-5}	1×10^{-4}

Table 8
Comparison for different n values of the maximum absolute error over integration interval [0, 1].

n	PINN error	MC error	CNN error
0	3.2×10^{-5}	3.3×10^{-4}	5.4×10^{-3}
1	1.1×10^{-5}	3.0×10^{-4}	6.4×10^{-3}
5	2.3×10^{-5}	2.0×10^{-4}	7.6×10^{-3}

Appendix B. Comparison to Monte-Carlo and Chebyshev Neural Network methods

We compare our results using PINNs with results obtained using a Monte-Carlo (MC) method that are published in El-Essawi et al. (2023), and also with results obtained using a Chebyshev Neural Network (CNN) method (Mall and Chakraverty, 2014). We consider integration over the x interval [0, 1] for the three polytropic indices ($n = 0$, 1, and 5). In order to make the comparison the close as possible to CNN results, we have chosen the same number of collocation points $N_c = 10$ for $n = 0, 5$ cases and $N_c = 20$ for the $n = 1$. We deduce the maximum absolute error taken over the whole interval from Tables 1–3 in El-Essawi et al. (2023) for MC and from Tables 1–3 in Mall and Chakraverty (2014) for CNN.

The results reported in Table 8 clearly show that our results are again more accurate by one and even two orders of magnitude when compared to MC and CNN methods respectively.

References

Baty, H., Baty, L., 2023. Solving differential equations using physics informed deep learning: a hand-on tutorial with benchmark tests. doi:10.48550/arXiv.2302.12260, Preprint.

Chandrasekhar, S., 1958. An Introduction to the Study of Stellar Structure. Dover Publications, Inc., Mineola.

Cuomo, S., Di Cola, V.S., Giampaolo, F., Rozza, G., Raissi, M., Piccialli, F., 2022. Scientific machine learning through physics-informed neural networks: Where we are and what's next. J. Sci. Comput. 92 (88).

El-Essawi, S.H., Nouh, M.I., Soliman, A.A., Abdel Rahman, H.I., Abd-Elmougod, G.A., 2023. Monte Carlo simulation of Lane-Emden equations arising in astrophysics. Astron. Comput. 42, 100665.

He, J., Long, P., Wang, X., He, K., 2020. A deep-learning-based method for solving nonlinear singular Lane-Emden type equation. IEEE Access 8, 203674-203684.

Iacono, R., De Felice, M., 2014. Celestial mechanics and dynamical astronomy. 118, pp. 291–298.

Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., Yang, L., 2021. Physics-informed machine learning. Nature Rev. 3, 422–440.

Lagaris, E., Likas, A., DI Fotiadis, L., 1998. Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Netw. 9 (5), 987–1000.

Le Cun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444.

Mall, S.M., Chakraverty, S., 2014. Chebyshev neural network based model for solving Lane–Emden type equations. Appl. Math. Comput. 247, 100–114.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 2007. Numerical Recipes, third ed..

Raissi, M., Perdikaris, P., Karniadakis, G.E., 2017. Physics informed deep learning (Part I): Data-driven solutions of nonlinear partial differential equations. doi:10.48550/arXiv.1711.10561, Preprint.

Raissi, M., Perdikaris, P., Karniadakis, G.E., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. 378, 686–707.

Sabir, Z., Ali, M.R., Fathurrochman, I., Raja, A.Z., Sadat, R., Baleanu, D., 2022. Dynamics of multi-point singular fifth-order Lane–Emden system with neuro-evolution heuristics. Evol. Syst. 13, 795–806.

Sabir, Z., Wahab, H.A., Umar, M., Erdogan, F., 2019. Stochastic numerical approach for solving second order nonlinear singular functional differential equation. Appl. Math. Comput. 363, 124605.

Sabir, Z., Wahab, H.A., Umar, M., Sakar, M.G., Raja, A.Z., 2020. Novel design of Morlet wavelet neural network for solving second order Lane–Emden equation. Math. Comput. Simulation 172, 1–14.

Zhang, W.S., Yang, Y.H., Xu, J.Y., 2003. Theory and application of Lattice Boltzmann method. Mod. Mach. 4, 4–6.