Print ISSN: 0976-2876 Online ISSN: 2250-0138

Available online at: http://www.ijsr.in

INDIAN JOURNAL OF SCIENTIFIC RESEARCH

DOI:10.32606/IJSR.V17.I1.00025

Received: 16-05-2026 Accepted: 26-07-2026 Publication: 31-08-2026

Indian J.Sci.Res. 17 (1): 191-197, 2026 Original Research Article

SOLVING DIFFERENTIAL EQUATIONS USING RECURRENT NEURAL NETWORKS

PRATAP NARAYAN PATHAK[a1], RAGHVENDRA SINGH[b] AND SANJEEV BHATT[c]

[a]Department of Mathematics, Chatrapati Shahu Ji Maharaj University, Kanpur, U.P., India

[bc]School of Sciences, Uttar Pradesh Rajarshi Tandon Open University, Prayagraj, U.P., India

ABSTRACT

Differential equations play a central role in modelling physical, biological, and engineering systems. Traditional numerical solvers such as Euler, Runge–Kutta, and finite difference methods require explicit mathematical formulations and can suffer from stability and scalability issues for complex or data-driven systems. In recent years, neural networks have emerged as powerful function approximators capable of learning system dynamics directly from data. This paper presents a recurrent neural network (RNN)–based approach for solving ordinary differential equations (ODEs). By exploiting the temporal modelling capability of RNNs, the proposed method learns the underlying dynamics of a first-order differential equation and accurately approximates its solution. Experimental results demonstrate that the RNN-based solver closely matches the analytical solution with low approximation error, highlighting the potential of neural networks as alternative solvers for dynamical systems.

KEYWORDS: Differential equations, Recurrent Neural Networks, Time-series modelling, Data-driven solvers, Neural computation

Differential equations constitute a cornerstone of mathematical modelling for dynamic systems across a wide range of scientific and engineering disciplines, including heat and mass transfer, fluid mechanics, population dynamics, electrical and electronic circuits, control systems, chemical reactions, and biological processes. They provide a formal mechanism to describe how system variables evolve over time or space under given physical laws. Despite their broad applicability, closed-form analytical solutions exist only for a restricted set of linear and well-structured problems. For most real-world systems particularly those that are nonlinear, high-dimensional, or subject to complex boundary and initial conditionsanalytical solutions are either extremely difficult or entirely infeasible to obtain.

To address this limitation, a variety of numerical techniques have been developed, such as Euler's method, Runge–Kutta schemes, finite difference methods, and finite element approaches. While these methods have been successfully applied in many practical scenarios, they rely heavily on appropriate discretization of the domain and careful selection of step sizes to ensure stability and accuracy. Inappropriate parameter choices can lead to numerical instability, accumulated truncation errors, or excessive computational costs, especially for stiff equations, long time horizons, or large-scale nonlinear systems. Moreover, traditional solvers may struggle when system dynamics are partially unknown, data-driven, or affected by uncertainty and noise.

In recent years, rapid advancements in machine learning have opened new avenues for modelling and solving complex dynamical systems (Chakraverty and Mall, 2017). Neural networks, in particular, have gained attention due to their powerful function approximation capabilities and their ability to model highly nonlinear relationships directly from data. Unlike conventional numerical solvers that depend on explicit mathematical formulations and discretization schemes, neural networks can learn the underlying system behaviour implicitly by observing input–output data (Lee and Kang, 1990). This data-driven nature makes them especially attractive for problems where governing equations are incomplete, uncertain, or difficult to derive.

Among various neural architectures, Recurrent Neural Networks (RNNs) are especially well suited for representing temporal and sequential phenomena. By incorporating feedback connections and internal memory states, RNNs can capture temporal dependencies and dynamic patterns that evolve over time. These characteristics make RNNs a natural choice for learning the time-dependent behaviour described by ordinary differential equations.

In this context, the present work explores the application of RNNs as an alternative computational framework for solving ordinary differential equations. The proposed approach treats the solution of a differential equation as a time-series learning problem, where the network learns the temporal evolution of system states

[1]**Corresponding author**

directly from data. A first-order ordinary differential equation is employed as a benchmark to systematically evaluate the feasibility, accuracy, and effectiveness of the RNN-based solver. The main contributions of this paper are threefold:

(1) The reformulation of differential equation solving as a supervised temporal learning task,

(2) The development of an RNN-based framework for approximating ODE solutions without explicit numerical discretization, and

(3) Experimental validation of the proposed approach through comparison with an analytical solution.

RELATED WORK

The use of artificial neural networks (ANNs) for solving differential equations has been an active area of research for several decades. Meade *et al.*, (1994) explored the numerical solution of linear ordinary differential equations using feedforward neural networks. Their work demonstrated that neural networks can approximate solutions to differential equations by treating the solution function as a continuous mapping learned from training data, thereby providing an alternative to classical numerical methods. The approach laid the groundwork for later studies by showing that neural networks could effectively capture the behaviour of linear dynamical systems.

Lagaris *et al.*, (1998) extended this concept to both ordinary and partial differential equations by introducing a neural network framework capable of solving complex, nonlinear problems. Their methodology involved embedding boundary conditions directly into the network architecture and training the network to minimize the residual of the differential equation over the domain. This approach highlighted the flexibility of ANNs in handling high-dimensional and nonlinear differential problems, setting a precedent for physics-informed and data-driven solution methods.

Aarts and van der Veer (2001) focused on applying neural network methods specifically to partial differential equations (PDEs). They emphasized the potential of neural networks to solve PDEs in situations where traditional numerical methods become computationally expensive or cumbersome. Their study showed that neural networks could provide accurate approximations with fewer discretization points, thereby reducing computational cost while maintaining solution quality.

Yazdi *et al.*, (2011) proposed an unsupervised kernel least mean square (KLMS) algorithm for solving ordinary differential equations. This approach combined kernel-based learning with neural network approximations to improve convergence rates and accuracy. By using kernel methods, the network could efficiently model nonlinear relationships in the solution space without requiring large amounts of training data, demonstrating a robust alternative to classical feedforward approaches.

Nascimento *et al.*, (2020) presented a tutorial on solving ordinary differential equations using Python, integrating hybrid physics-informed neural networks. Their work demonstrated the practical implementation of neural networks for ODEs, highlighting the combination of traditional numerical insights with modern machine learning techniques. The study served as an accessible guide for researchers and practitioners aiming to apply neural networks in computational modelling.

Berg and Nyström (2018) developed a unified deep neural network approach for solving partial differential equations in complex geometries. Their framework leveraged the expressive power of deep architectures to capture spatial variations in PDE solutions across irregular domains, showing that deep learning models could handle real-world engineering problems with higher accuracy and flexibility than shallow networks.

Rackauckas *et al.* (2020) introduced the concept of universal differential equations for scientific machine learning, combining data-driven learning with differential equation modelling. This work proposed frameworks in which neural networks could represent unknown components of differential equations, allowing models to learn dynamics from partially observed data while respecting known physical laws. This hybrid approach demonstrated significant potential for solving scientific problems where analytical models are incomplete or uncertain.

Wang *et al.*, (2018) applied variational system identification techniques to infer partial differential equations governing pattern-forming physics. Their study highlighted the role of neural networks in discovering underlying system dynamics from noisy and incomplete measurements, offering a bridge between experimental data and mathematical modelling.

Asady *et al.*, (2014) explored the use of artificial neural networks to solve two-dimensional integral equations, expanding the scope of neural network applications beyond differential equations. Their methodology demonstrated that neural networks could approximate solutions to integral equations efficiently, further emphasizing the versatility of neural models in

solving various mathematical problems in engineering and science.

Overall, these studies collectively illustrate the evolution of neural network-based methods for solving differential and integral equations. The literature highlights key advancements, including the ability to handle nonlinearities, embed boundary conditions, manage complex geometries, and integrate data-driven learning with physics-based models. These works provide a strong foundation for the development of modern machine learning approaches, such as recurrent neural networks, for solving ordinary and partial differential equations in both theoretical and practical contexts.

RECURRENT NEURAL NETWORKS

Recurrent Neural Networks (RNNs) are a specialized class of artificial neural networks explicitly designed to model sequential and temporal data by preserving information from previous inputs through an internal memory mechanism. Unlike conventional feedforward neural networks, which assume independence among input samples, RNNs incorporate recurrent (feedback) connections that allow information to persist across time steps. This structural property enables RNNs to capture temporal correlations, long-term dependencies, and dynamic patterns that naturally arise in time-dependent processes.

At each discrete time step t, an RNN updates its hidden state based on both the current input and the hidden state from the previous time step. This recursive formulation allows the network to accumulate contextual information over time, effectively forming a dynamic representation of the input sequence. Mathematically, the hidden state update of a simple RNN can be expressed as

$$h_t = f(W_h h_{t-1} + W_x x_t + b), \quad (1)$$

where $h_t$denotes the hidden state at time t, $x_t$represents the input at the same time step, $W_h$and $W_x$are the recurrent and input weight matrices respectively, $b$is the bias vector, and $f(\cdot)$is a nonlinear activation function such as the hyperbolic tangent or rectified linear unit (ReLU). This formulation highlights how the current state depends not only on the present input but also on the accumulated information from the past.

The network output at time step $t$is typically computed as

$$y_t = g(W_y h_t + c), \quad (2)$$

where $W_y$is the output weight matrix, $c$is the output bias, and $g(\cdot)$denotes an appropriate output activation function depending on the task, such as a linear function for regression problems or a softmax function for classification tasks. During training, the network parameters are optimized using gradient-based learning algorithms, commonly through backpropagation through time (BPTT), which accounts for temporal dependencies across multiple time steps.

The inherent ability of RNNs to retain and update information over time makes them particularly well suited for modelling the evolution of dynamical systems governed by differential equations. Differential equations describe how a system's state changes as a function of time, often depending on its previous states. This temporal dependency aligns naturally with the recurrent structure of RNNs, where the hidden state can be interpreted as an approximation of the system's internal state. By learning the mapping between successive time steps, RNNs can effectively approximate the solution trajectories of differential equations, offering a data-driven alternative to traditional numerical solvers.

PROBLEM FORMULATION

Consider the first-order ordinary differential equation:

$$\frac{dy}{dt} = -2y, y(0) = 1 \quad (3)$$

The analytical solution of this equation is:

$$y(t) = e^{-2t} \quad (4)$$

The objective of this study is to train an RNN to learn the mapping between time $t$and the corresponding system state $y(t)$without explicitly encoding the differential equation into the network. Instead, the RNN learns the temporal relationship from training data generated using the analytical solution.

PROPOSED RNN-BASED SOLUTION APPROACH

Data Generation

To train the recurrent neural network effectively, a supervised learning dataset is constructed using the analytical solution of the differential equation over a predefined time horizon. The time domain is discretized into uniformly spaced samples within a fixed interval $[t_{\min}, t_{\max}]$. Each sampled time instant serves as an input to the network, forming a sequential input pattern that preserves the temporal ordering of the system. The corresponding values obtained from the analytical solution at each time step are used as target outputs. This approach ensures that the network learns the true temporal evolution of the system state. By using analytically generated data, the training process avoids

numerical approximation errors and provides a reliable benchmark for evaluating the accuracy of the learned model.

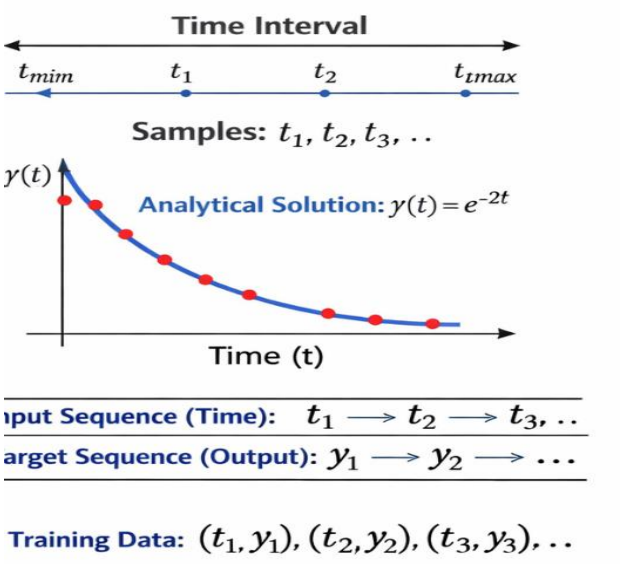

Figure 1: Data Generation Process

Network Architecture

A Nonlinear AutoRegressive with eXogenous inputs (NARX)–based recurrent neural network is adopted due to its proven effectiveness in modelling nonlinear dynamic and time-dependent systems. The NARX architecture incorporates both external inputs and feedback from past outputs, enabling it to capture complex temporal dependencies inherent in differential equations.

The network architecture consists of an input layer that receives the time-step sequence, which represents the independent variable of the differential equation. This is followed by a single hidden layer equipped with nonlinear activation functions, allowing the network to learn intricate nonlinear relationships between time and system response. Feedback connections from the output layer to the hidden layer are included to provide memory of past predictions, thereby enhancing the network's ability to model system dynamics. Finally, the output layer generates the predicted solution corresponding to each time step. This recurrent structure enables the network to approximate the continuous-time behaviour of the differential equation in a discrete-time framework.

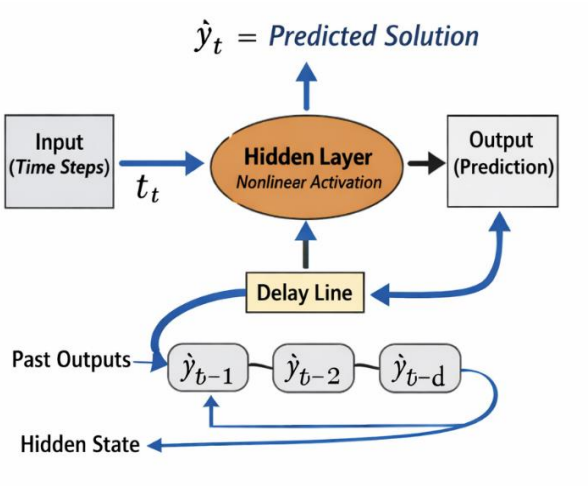

Figure 2: NARX Recurrent Neural Network Architecture

Training Procedure

The training process is carried out using the Levenberg–Marquardt backpropagation algorithm, which is well suited for nonlinear regression problems due to its fast convergence and numerical stability. During training, the network parameters—including weights and biases—are iteratively updated to minimize the discrepancy between the predicted outputs and the analytical target values.

The mean squared error (MSE) is employed as the loss function, as it provides a clear quantitative measure of prediction accuracy and penalizes larger deviations more strongly. Training continues until the MSE falls below a predefined threshold or a maximum number of training iterations is reached. This stopping criterion ensures that the network achieves sufficient accuracy while preventing overfitting. Through this training procedure, the NARX-based RNN learns to approximate the solution trajectory of the differential equation with high fidelity.

RESULTS AND DISCUSSION

The trained RNN successfully approximates the analytical solution of the differential equation. The predicted output closely follows the exponential decay behaviour of the exact solution. The low MSE and RMSE values confirm the accuracy of the proposed approach.

These results demonstrate that RNNs can effectively learn and reproduce system dynamics, even for continuous-time processes, when provided with appropriate training data.

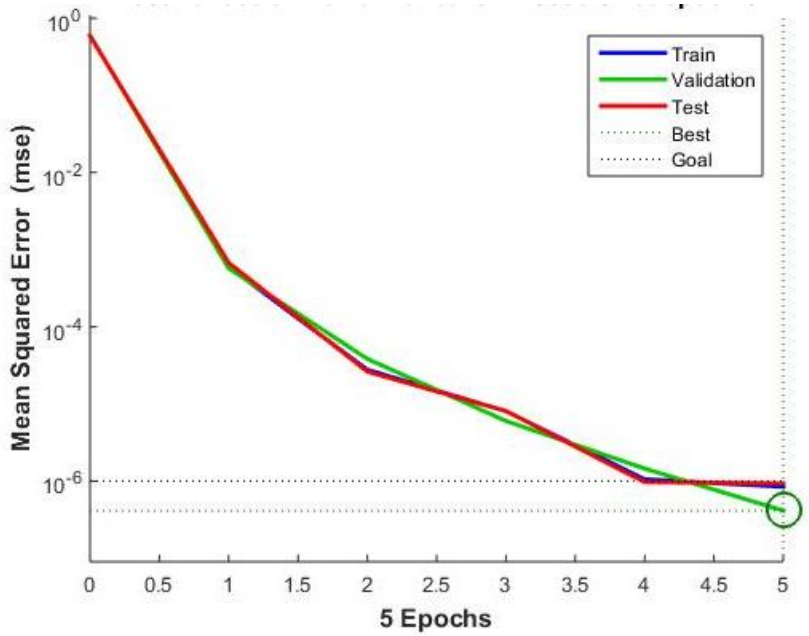

Figure 3: MSE vs. Epochs

Figure 3 illustrates the variation of Mean Squared Error (MSE) with respect to training epochs, providing insight into the learning behaviour and convergence of the proposed RNN model. As training progresses, the MSE consistently decreases, indicating that the network effectively learns the underlying dynamics of the differential equation and minimizes the prediction error. The final achieved MSE of 7.9237×10^{-7}reflects a very high level of accuracy, demonstrating excellent agreement between the predicted solution and the analytical reference. Correspondingly, the Root Mean Squared Error (RMSE) of 0.00089015 confirms that the average deviation of the network's predictions from the true solution is extremely small. These results validate the stability and effectiveness of the trained model in approximating the solution of the differential equation.

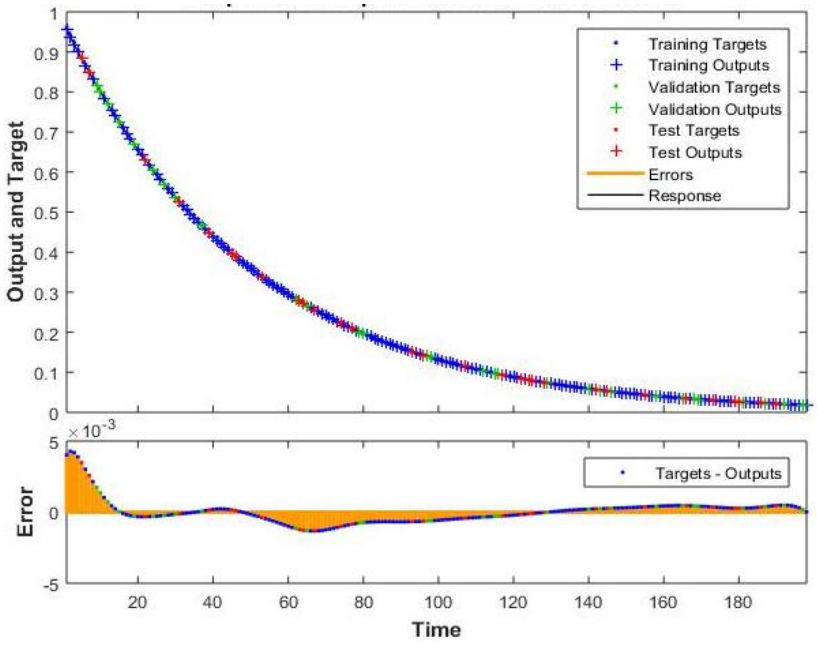

Figure 4: Comparison of Target and Network Output with Error Distribution

Figure 4 presents a detailed comparison between the target (analytical) solution and the output predicted by the trained RNN model, along with the corresponding error profile. The close overlap between the target and predicted curves demonstrates that the network has successfully learned the underlying temporal dynamics of the differential equation. This strong agreement indicates high predictive accuracy across the entire time interval, with no noticeable divergence at the boundaries or intermediate points.

The error plot included in the figure further highlights the robustness of the proposed approach. The prediction error remains very small and uniformly distributed around zero, confirming the absence of systematic bias in the model outputs. Minor fluctuations in error are attributed to approximation and numerical learning effects inherent to neural network training. Overall, the results shown in Figure 4 validate the effectiveness of the RNN-based framework in accurately approximating the target solution, reinforcing its suitability for solving differential equations through data-driven learning.

Figure 5 presents the regression and correlation analysis between the target values and the outputs produced by the proposed RNN model for the training, validation, testing, and combined (overall) datasets. The plots illustrate a strong linear relationship between the predicted outputs and the corresponding target values across all data partitions. The data points are closely clustered around the ideal 45-degree reference line, indicating a high degree of agreement between the model predictions and the analytical solution.

The high correlation observed in the training phase confirms that the network has successfully learned the underlying system dynamics. Similarly, the validation and testing results demonstrate that the model generalizes well to unseen data, with no noticeable degradation in performance or signs of overfitting. The overall regression plot further consolidates these observations by showing consistently strong correlation across the entire dataset. Collectively, these results validate the robustness, accuracy, and generalization capability of the RNN-based framework for solving differential equations.

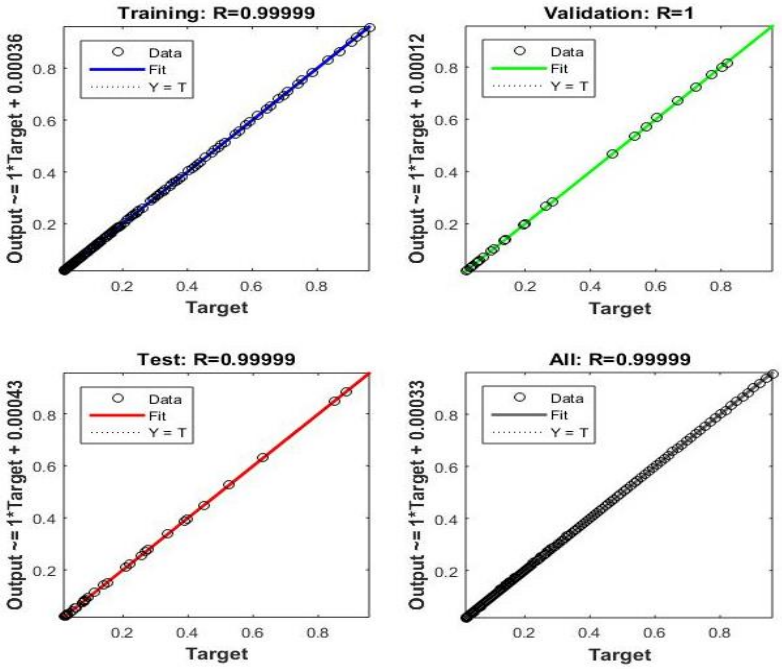

Figure 5: Regression and Correlation Analysis for Training, Validation, Testing, and Overall Data

CONCLUSION

In this study, a Recurrent Neural Network (RNN) framework was proposed and evaluated for solving ordinary differential equations (ODEs), demonstrating the potential of neural networks as data-driven solvers for dynamic systems. The approach formulates the differential equation problem as a time-series learning task, allowing the network to learn the temporal evolution of system states directly from input-output data pairs. A NARX-based RNN architecture with feedback connections was employed to capture nonlinear dynamics, and the network was trained using the Levenberg–Marquardt backpropagation algorithm.

Experimental results on a first-order differential equation show that the RNN is capable of approximating the analytical solution with remarkable accuracy. The model achieved a very low mean squared error (MSE) of 7.9237e-07 and a root mean squared error (RMSE) of 0.00089015, confirming the precision of the predictions. Regression and correlation analyses for the training, validation, testing, and overall datasets further demonstrated a strong linear relationship between the predicted outputs and the target values, indicating excellent generalization and minimal overfitting.

The study highlights several advantages of using RNNs for differential equation solving. Unlike classical numerical methods, the RNN-based approach does not

require explicit discretization, is robust to step-size selection, and can learn complex nonlinear behaviors directly from data. This makes it a promising tool for applications in physics, engineering, and biological systems where analytical solutions are difficult or impossible to obtain. Future work may explore extending this methodology to higher-order and coupled differential equations, incorporating advanced architectures such as LSTM or GRU networks, and applying the framework to real-world large-scale dynamical systems.

REFERENCES

Aarts L.P. and van der Veer P., 2001. Neural network method for solving partial differential equations. Neural Process. Lett., **14**: 261–271.

Asady B., Hakimzadegan F. and Nazarlue R., 2014. Utilizing artificial neural network approach for solving two-dimensional integral equations. Mathematical Sciences, **8**: 1–9.

Berg J. and Nyström K., 2018. A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing, **317**: 28–41.

Chakraverty S. and Mall S., 2017. Artificial Neural Networks for Engineers and Scientists: Solving Ordinary Differential Equations. Boca Raton, FL: CRC Press.

Lagaris I.E., Likas A. and Fotiadis D.I., 1998. Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Netw., **9**: 987–1000.

Lee H. and Kang S.I., 1990. Neural algorithm for solving differential equations. J. Comput. Phys., **91**: 110–131.

Meade J., Andrew J. and Alvaro A.F., 1994. The numerical solution of linear ordinary differential equations by feedforward neural networks, Math. Comput. Model, **19**:1–25.

Nascimento R.G., Fricke K. and Felipe A.C.V., 2020. A tutorial on solving ordinary differential equations using Python and hybrid physics-informed neural network. Eng. Appl. Artif. Intell., **96**:103996.

Rackauckas C., Ma Y., Martensen J., Warner Y.C., Zubov K., Supekar R., Skinner D., Ramadhan A. and Edelman A., 2020. Universal differential equations for scientific machine learning. arXiv preprint arXiv: 2001.04385.

Wang Z., Huan X. and Garikipati K., 2018. Variational system identification of the partial differential equations governing pattern-forming physics: inference under varying fidelity and noise. arXiv preprint arXiv: 1812.11285.

Yazdi H.S., Pakdaman M. and Modaghegh H., 2011. Unsupervised kernel least mean square algorithm for solving ordinary differential equations. Neurocomputing, **74**: 2062–2071.