Int. J. Nonlinear Anal. Appl. In Press, 1–28
ISSN: 2008-6822 (electronic)
http://dx.doi.org/10.22075/ijnaa.2026.35050.5235

RNN-MCTS-PINN: A Novel Integration of Recurrent Neural Networks, Monte Carlo Tree Search, and Physics-Informed Neural Networks for Symbolic Solutions to the Lane-Emden Type Equation

Hassan Dana Mazraeh[a,*], Kourosh Parand[b,c]

[a]School of Mathematics and Computer Science, Damghan University, P.O.Box 36715-364, Damghan, Iran.

[b]Department of Computer and Data Sciences, Faculty of Mathematical Sciences, Shahid Beheshti University, Tehran, Iran.

[c]Department of Cognitive Modeling, Institute for Cognitive and Brain Sciences, Shahid Beheshti University, Tehran, Iran.

Abstract

In this paper, we introduce an innovative approach that integrates three robust architectures and methods: recurrent neural networks, the Monte Carlo tree search algorithm, and the physics-informed neural networks approach. This method is designed to symbolically solve the Lane-Emden type equation, a nonlinear ordinary differential equation. The proposed scheme consists of three phases. In Phase 1, a recurrent neural network is employed to find a numerical solution for the equation. Phase 2 involves using a carefully crafted Monte Carlo tree search algorithm to perform symbolic regression on the results from the previous phase. During Phase 2, a symbolic solution to the equation is found. This involves designing a Monte Carlo tree search algorithm that utilizes a context-free grammar to construct a mathematical expression with certain coefficients through a tree representation of successive substitution in the grammar. In Phase 3, the coefficients of the symbolic expression identified in Phase 2 are fine-tuned using the physics-informed neural networks approach. To refine the coefficients found by the Monte Carlo tree search, Phase 3 treats the computational graph of the mathematical expression generated in PyTorch as a network. The physics-informed neural networks approach is then applied to optimize the parameters (coefficients) of the symbolic expression. To evaluate the effectiveness of the proposed scheme, we apply it to the Lane-Emden type equation, a challenging and suitable benchmark. The results demonstrate that our method is effective for symbolically solving nonlinear ordinary differential equations.

Keywords: Recurrent neural networks, Monte Carlo tree search, Physics-informed neural networks, The Lane-Emden type equation, Symbolic solution.
2020 MSC: 34A45; 68T07

1 Introduction

Numerous phenomena encountered in science and engineering are described by nonlinear ordinary differential equations (NODEs). These equations describe systems in terms of their derivatives. NODEs are widely used in various scientific and engineering fields, including astrophysics [38, 15], disease modeling [17], and financial markets [50].

*Corresponding author
Email addresses: dana@du.ac.ir (Hassan Dana Mazraeh), k_parand@sbu.ac.ir (Kourosh Parand)

Received: August 2024 *Accepted:* November 2026

Symbolic solutions to differential equations are crucial in scientific computing, offering analytical insights that are valuable across different applications. These solutions reveal dependencies on equation parameters, enhancing our understanding of the underlying mathematical structures. They also validate numerical solutions obtained through methods like finite element analysis, finite difference methods, finite volume methods, and spectral methods. Symbolically solving differential equations is vital for comprehending system dynamics and developing efficient control strategies. Furthermore, symbolic solutions provide insights into system dynamics, stability, and critical points.

Given the importance of symbolic solutions, researchers worldwide have developed various symbolic methods for solving differential equations. The advent of machine learning has further boosted the use of these methods for tackling complex differential equations. This paper introduces a novel hybrid method that combines recurrent neural networks, Monte Carlo tree search based on context-free grammars, and physics-informed neural networks to symbolically solve NODEs. This research is the first attempt to combine a Recurrent Neural Network (RNN) architecture, a Monte Carlo Tree Search (MCTS) algorithm, and a Physics-Informed Neural Networks (PINNs) approach for symbolically solving nonlinear ordinary differential equations. The proposed hybrid scheme consists of three powerful architectures and methods. The first architecture is a deep RNN, which is highly effective in handling sequences of inputs. Since our study requires the input and output of the network to be fed in sequence order, the RNN architecture is a suitable choice for our work. In the first phase of our paper, a deep RNN is employed to find accurate numeric solutions for the NODE under consideration. The second one is a MCTS algorithm, an excellent method for finding a sequence of actions in Phase 2. This sequence of actions is then transformed into a mathematical expression using a context-free grammar. The expression is crafted to best fit the numeric values obtained in Phase 1 using the deep RNN model. Finally, the third member is a PINN approach, which excels in determining the correct parameters of a network for solving differential equations. In Phase 3, a PINN optimizes the coefficients of the expression found in Phase 3 so that it becomes a more accurate solution to the equation at hand. This paper integrates these three methods to create a powerful scheme for symbolically solving nonlinear and challenging equations. The Lane-Emden type equation is used to demonstrate the effectiveness of the proposed method. This equation is significant for two reasons. Firstly, it has practical applications in astrophysics, describing the behavior of a self-gravitating, spherically symmetric, polytropic fluid. It represents a dimensionless form of Poissons equation for the gravitational potential of a Newtonian fluid. Secondly, the Lane-Emden type equation is challenging to solve due to its nonlinear nature and singularities, and it often lacks exact solutions. Consequently, this equation serves as an excellent benchmark for evaluating the proposed method. Solving the Lane-Emden type equation shows the method's capability to handle a wide range of real-world applications.

The primary form of the Lane-Emden equation is given by [12, 32]:

$$y''(x) + \frac{2}{x}y'(x) + f(y(x)) = 0, \quad x \geq 0, \tag{1.1}$$

with initial conditions:

$$y(0) = \alpha, \quad y'(0) = 0.$$

Here, α is a real constant, and $f(y(x))$ is a specified function. In Eq. (1.1), when $f(y(x)) = y^m(x)$ and $\alpha = 1$, the equation simplifies to the standard Lane-Emden equation as follows:

$$y''(x) + \frac{2}{x}y'(x) + y^m(x) = 0, \quad x \geq 0, \tag{1.2}$$

with initial conditions:

$$y(0) = 1, \quad y'(0) = 0.$$

This paper is organized as follows: In Section 2, we discuss the related works. In Section 3, we present our hybrid scheme for solving NODEs. In Section 4, we provide the experimental results of our proposed method. In Section 5, we discuss the results obtained in this study. Finally, in the conclusion section, we conclude the paper.

2 Related Works

Given the extensive research conducted on differential equations in recent years, this section will focus solely on recent articles related to the topic of this paper. It is divided into three subsections. First, Subsection 2.1 examines key articles on solving differential equations using deep recurrent neural networks. Next, Subsection 2.2 reviews research papers on the use of Monte Carlo tree search algorithms for solving differential equations. Finally, Subsection 2.3 discusses various methods proposed for solving the Lane-Emden type equation.

2.1 A brief literature review on recurrent neural networks for solving differential equations

In [19], the authors address the challenge of solving a system of ordinary differential equations with an unknown functional form for a sink term. They presume that partial time series measurements of the state variables are available and utilize a recurrent neural network to learn the reaction rate from this data. This is accomplished by integrating discretized ordinary differential equations into the recurrent neural network training framework. In [29], the authors investigate the fundamental trade-offs between training stability, temporal dynamics, and the complexity of recurrent neural network architectures by applying numerical methods from ordinary differential equations. They introduce a broad category of RNNs termed ODERNN, linking the composition rules of RNNs to the integration methods of ODEs at discrete time steps. Their analysis demonstrates that the degree of functional nonlinearity n in RNNs and the extent of their temporal memory t can be associated with the stages of Runge-Kutta recursion and the order of time derivatives in ODEs. They show that well-known RNN architectures correspond to different orders of n and t within the ODERNN framework. In [7], the authors suggest employing recurrent neural networks to learn and represent nonlinear integral operators found in nonlinear integro-differential equations. The training process involves feeding the model with discretized inputs at time step $t = \delta i$ and obtaining an estimated solution of the collision integral for $t = \delta i$ as the output. This RNN model is trained using a highly accurate numerical solution to the integro-differential equations over a time interval $t \in [0, T]$. More recently, Long et al. [24] proposed a hybrid scheme that combines gated recurrent units with an implicit numerical time-stepping method for the identification of unknown parameters of time-dependent partial differential equations from sparse interior data. In their approach, the network first produces an approximation of the solution over the whole domain; the physical constraint is then imposed through the residual of the time iteration scheme, and this residual is combined with a data-misfit term in the loss function. The authors report successful experiments on the Burger, Allen–Cahn, and nonlinear Schrödinger equations, and motivate the design by the observation that standard physics-informed neural networks deteriorate when interior data are sparse. This line of work is close in spirit to Phase 1 of the present paper, where a recurrent network is trained through a finite-difference discretization of the governing equation without any measurement data.

2.2 A brief literature review on the Monte Carlo tree search algorithm for solving differential equations

The Monte Carlo Tree Search and related Monte Carlo methods offer innovative approaches to solving differential equations, as evidenced by several key research papers. In [16], the researchers applied the Monte Carlo method to solve both linear and nonlinear ordinary differential equations, transforming the system into integral equations and using branched Markov chains for solutions, particularly useful for large systems and low smoothness initial functions. In [36] the authors presented a Monte Carlo method for solving ordinary differential equations by random enumeration of Butcher trees in the Runge-Kutta method, offering direct evaluation without multiple time steps. In [43], a technique is introduced that utilizes the MCTS algorithm to derive analytical expressions for modeling nonlinear dynamics using limited data. The researchers developed an innovative symbolic physics learner machine to uncover the mathematical framework of nonlinear dynamics. The core idea involves interpreting mathematical operations and system state variables through computational rules and symbols, constructing symbolic reasoning of mathematical formulas via expression trees, and using a MCTS agent to identify optimal expression trees based on measurement data. In [15], a computational approach leveraging Monte Carlo algorithms is proposed to solve the Lane-Emden type equation. The authors examined four specific equations related to astrophysics: those involving positive and negative indices of polytropic gas spheres, the isothermal gas sphere, and the white dwarf equation. Monte Carlo tree search has remained an active tool for symbolic expression discovery during 2025 and 2026. Yu et al. [52] introduced SR4MDL, in which a pretrained transformer (MDLformer) estimates the description length of the formula underlying the data and this estimate is used as the search objective of a Monte Carlo tree search, replacing the usual prediction error. Huang et al. [22] investigated several modifications that improve the efficiency of Monte Carlo tree search for symbolic regression. Saveliev et al. [40] presented influence-guided symbolic regression, in which a large language model proposes candidate basis functions, the marginal contribution of every term to generalization accuracy is used for pruning, and the whole cycle is embedded in a Monte Carlo tree search that balances the exploration of new functional forms against the exploitation of influential terms. These works target general symbolic regression from data; none of them couples the search with a context-free grammar over production rules, a recurrent-network numerical solution, and a physics-informed refinement of the coefficients, as is done in the present paper.

2.3 A brief literature review on methods utilized to solve the Lane-Emden-type equation

Finite difference and finite element methods have been used to solve the Lane-Emden type equation by several researchers [20, 10, 23, 44, 6]. Other studies have explored meshless methods for the same equation [30, 34, 33]. The

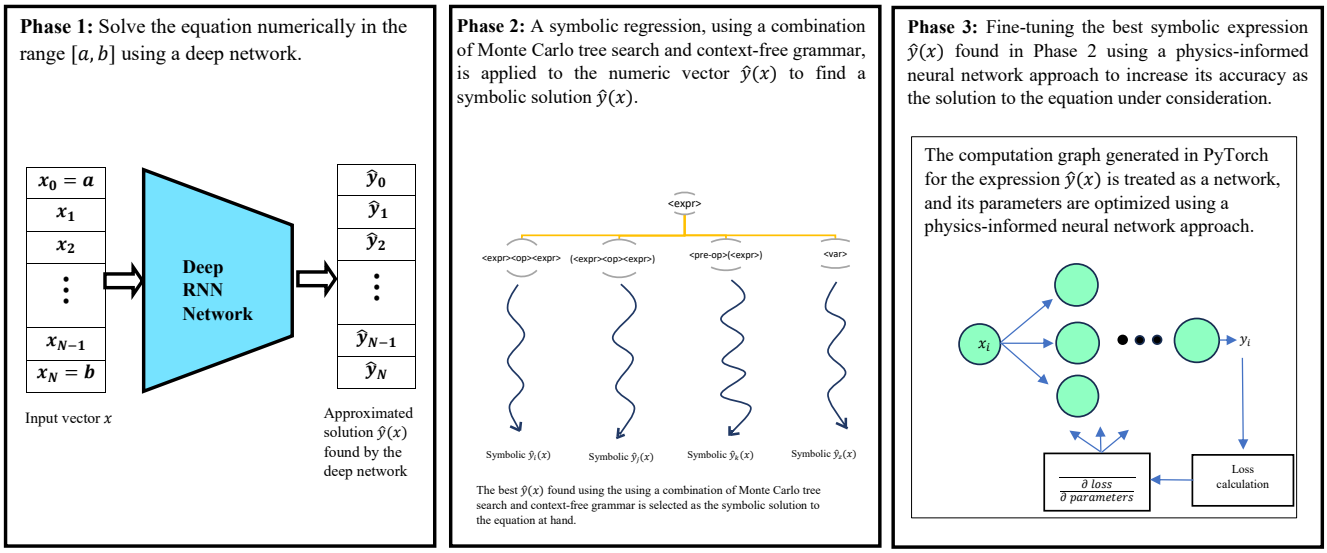


Figure 1: A three-phase procedure for finding a symbolic solution to the equation under study.

variational iteration method has also been employed in attempts to solve the Lane-Emden type equation [12, 49, 42, 41]. Spectral methods have been applied by multiple researchers [35, 53, 9, 14]. Additionally, various analytical and semi-analytical approaches have been presented for solving this equation [1, 48, 54, 5, 13, 4, 18]. Artificial neural networks have been utilized in solving the Lane-Emden type equation as well [27, 3, 47, 39, 31, 8, 25, 11, 51].

2.4 The innovation of this research

The focus of this research paper and use of the RNN, MCTS, and PINN approaches differ from the research papers mentioned in Subsections 2.1, 2.2, and 2.3. Our research introduces an innovative perspective on each algorithm. Notably, we have developed an MCTS that uses a context-free grammar to construct branches in a tree. Additionally, we utilize the PINN approach to fine-tune the coefficients of symbolic expressions. Finally, we present an innovative combination of these three algorithms in a scheme that yields a symbolic solution for nonlinear ordinary differential equations.

3 Combination of Recurrent Neural Networks, the Monte Carlo Tree search, and Physics-Informed Neural Network Approach

In this section, we present a scheme that combines three powerful methods: recurrent neural networks, the Monte Carlo tree search, and the physics-informed neural networks approach. These methods have demonstrated great capability in their respective domains. Figure 1 illustrates the main framework of this research. This scheme consists of three phases.

1. Phase 1: This phase utilizes a deep recurrent neural network for the numerical solution of the equation at hand. This phase is detailed in Subsection 3.1.
2. Phase 2: In this phase, a Monte Carlo Tree Search algorithm, based on a context-free grammar, finds an appropriate symbolic expression for the numerical values obtained in the previous phase. This phase is explained in Subsection 3.2.
3. Phase 3: In this phase, the physics-informed neural networks approach is used to fine-tune the coefficients of the symbolic expression found in the previous phase. Phase 3 is detailed in Subsection 3.3.

3.1 Phase 1: Recurrent neural network for numeric solutions to the Lane-Emden type equation

In this study, in Phase 1, as shown in Figure 1, an RNN is utilized to find a numerical solution for the equation under consideration. In fact, in phase 1, we use a finite difference method for training a network to solve the equation

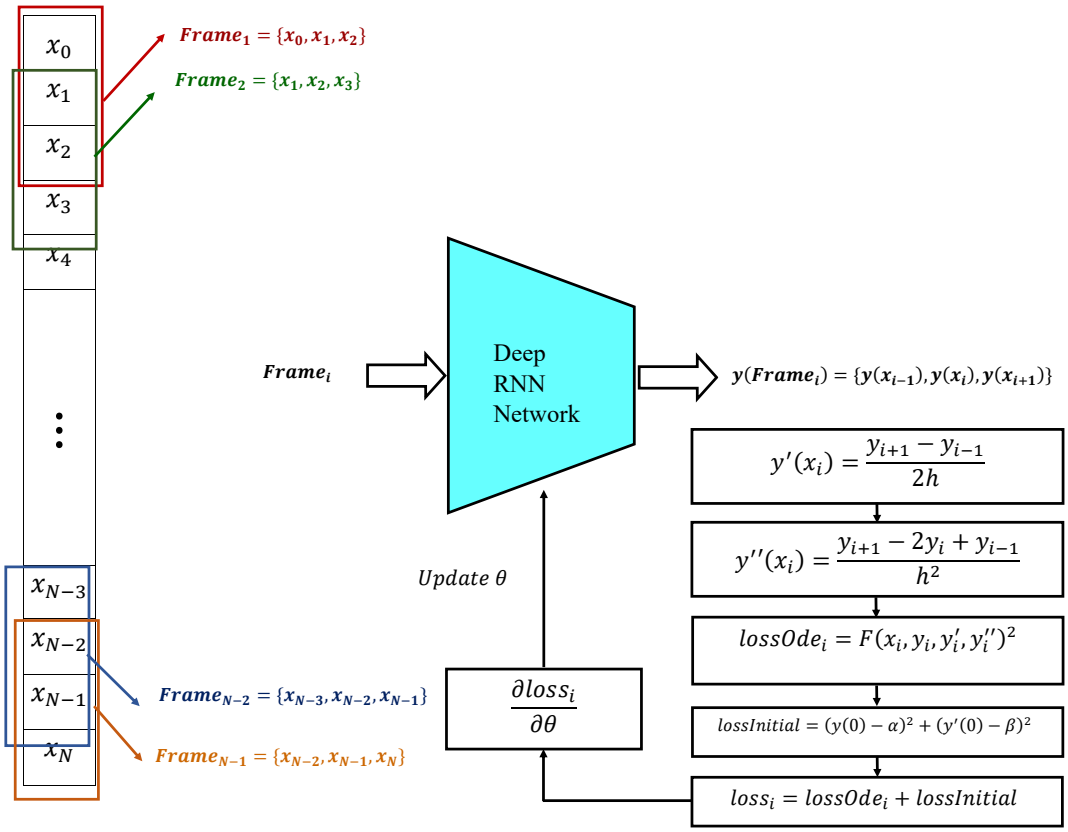


Figure 2: The procedure of the training of a network using the finite difference scheme.

at hand. Suppose we are dealing with second-order ordinary differential equations. The main form of second-order ordinary differential equations is as follows:

$$F(x, y, y', y'') = 0, \tag{3.1}$$

with initial conditions:

$$y(0) = \alpha, \quad y'(0) = \beta.$$

Here, $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}$, y is the unknown function of the independent variable x, y' is the first derivative of y with respect to x, y'' is the second derivative of y with respect to x, F is a function of the independent variable x, y, y', and y''.

To train a network for solving equation (3.1) in range $[a, b]$ using the finite difference method, we use the following formula for the first and second derivatives:

$$y'(x) \approx \frac{y(x+h) - y(x-h)}{2h}, \tag{3.2}$$

$$y''(x) \approx \frac{y(x+h) - 2y(x) + y(x-h)}{h^2}, \tag{3.3}$$

where h is a small step size, $y(x+h)$ is the function value at $x+h$, $f(x-h)$ is the function value at $x-h$.

We descitisize the range $[a, b]$ with the step size h as follows:

$$a = x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N = b, \quad x_i = h \times i, \quad N = \frac{b-a}{h}.$$

Therefore, to train a network for solving second-order differential equations we need to feed the network with three points x_{i-1}, x_i, and x_{i+1} for $i = 1, \ldots, N-1$ and calculate the first and second derivative of $y(x)$ based on the output of the network for each $i = 1, \ldots, N-1$. Then calculate the loss value based on the main form of the equation under consideration and its initial conditions. Next, we need to update the parameters (θ) of the network using the value of $\frac{\partial loss}{\partial \theta}$. Figure 2 illustrates the procedure of the training of a network using the finite difference method.

The rationale behind using recurrent neural networks for implementing the scheme shown in Figure 2 is as follows:

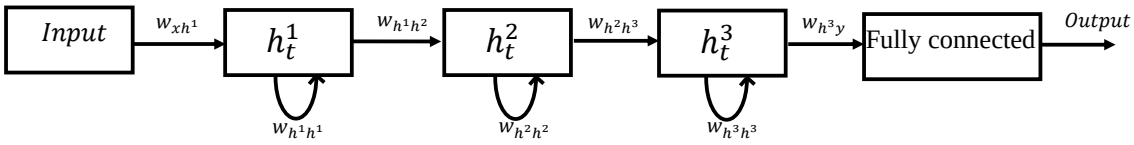


Figure 3: The deep network architecture used in this paper for finding numerical values of $y(x_i), i = 0, \ldots, N$.

1. In the scheme shown in Figure 2, we encounter sequential window frames. Therefore, we need a deep network architecture that can handle sequential data frames. This requirement is met by using recurrent neural networks, which are capable of processing sequential data types.
2. Differential equations can be defined over various ranges, necessitating a deep network architecture capable of handling different input ranges with a consistent input layer structure. The challenge lies in creating a neural network with a fixed number of parameters that can process a variable number of inputs. This is achieved through recurrent neural networks. In an RNN, each layer corresponds directly to specific positions in the sequence, known as time stamps [2]. Instead of having a variable number of nodes in the input layer, we use a fixed number of nodes in both the input layer and the recurrent hidden layers. Due to the recurrent nature of the hidden layers, this architecture can handle various input lengths, which is crucial for solving differential equations, as they can be considered over various intervals $[a, b]$.

Figure 3 illustrates the deep network architecture used in this paper to determine the numerical values of $y(x_i)$, for $i = 0, \ldots, N$. This deep network is depicted within the blue block in Figure 2. After training the network shown in Figure 3, a Monte Carlo tree search procedure is applied to the numerical values $y(x_i)$ to find a symbolic expression. This procedure is detailed in Subsection 3.2. The activation functions used in the hidden state of the recurrent neural network shown in Figure 3 are all $\tanh(\cdot)$. The fully connected layer calculates the logit value as $y(x_i)$. To keep the figure simple, the bias terms have not been shown in the block diagram. However, all hidden units and the fully connected layer do have bias vectors as well.

3.2 Phase 2: Monte Carlo tree search for symbolic solutions to the Lane-Emden type equation

In this subsection, we present an innovative combination of Monte Carlo tree search and context-free grammars (CFGs) for finding symbolic expressions as solutions to the Lane-Emden type equation. As shown in Figure 1, Phase 2 involves performing symbolic regression on the results obtained by a deep recurrent neural network in Phase 1. To demonstrate how we combine MCTS and CFGs for symbolic regression, we first briefly describe the procedure for constructing a mathematical expression from an integer vector $v = [a_0, a_1, \ldots, a_n]$, where $a_i \in \{0, 1, 2, 3\}$ and $i = 0, \ldots, n$, based on a context-free grammar. Then, we present the MCTS procedure, which uses a CFG to find a symbolic solution to the Lane-Emden type equation.

3.2.1 Constructing a mathematical expression from an integer vector based on a context-free grammar

A context-free grammar is a formal system used to define the syntactic structure of a language or mathematical expressions. It consists of a set of production rules, denoted as R, that describe how the expressions can be generated. Each rule specifies how a symbol can be replaced by a combination of other symbols. The grammar is "context-free" because these rules apply regardless of the surrounding symbols, focusing only on the symbol being replaced [26].

A context-free grammar G has four components, represented as $G = (V, T, R, S_0)$. The main components of a context-free grammar are as follows:

1. A set of non-terminal (variable) symbols, denoted by V: These symbols can be replaced with sequences of variables and terminal symbols based on the production rules.
2. A set of terminal symbols, denoted by T: These are the final characters of a mathematical expression, which appear in the final output. They cannot be replaced further.
3. A set of production rules, denoted by R: These define how variable symbols can be transformed into other variables or terminal symbols. Each rule has the form $A \to \alpha$, where A is a variable symbol, and α is a string of variable and/or terminal symbols.
4. A start symbol, denoted by S_0: This is a special variable symbol from which the production begins. It represents the entire structure of the language.

For example, the context-free grammar G_0 could be considered as follows:

$$G_0 = (V, T, S_0, R)\,. \tag{3.4}$$

In this grammar the set of variables V has four variables as follows:

$$V = \{< expr >, < op >, < pre - op >, < var >\}.$$

The terminal set T is as follows:

$$T = \{sin, cos, exp, log, +, -, \times, /, x, w, (,)\}.$$

The starting variable S_0 is as follows:

$$S_0 =< expr > .$$

Finally, the set of production rules R is as follows:

$$\begin{aligned}
< expr > \rightarrow & < expr >< op >< expr > && | \quad RN = 0\\
& (< expr >< op >< expr >) && | \quad RN = 1\\
& < pre - op > (< expr >) && | \quad RN = 2\\
& < var > && \quad RN = 3\\
< var > \rightarrow & x && | \quad RN = 0\\
& w && \quad RN = 1\\
< pre - op > \rightarrow & sin && | \quad RN = 0\\
& cos && | \quad RN = 1\\
& exp && | \quad RN = 2\\
& log && \quad RN = 3\\
< op > \rightarrow & + && | \quad RN = 0\\
& - && | \quad RN = 1\\
& / && | \quad RN = 2\\
& \times && \quad RN = 3
\end{aligned}$$

In this context, the vertical bar symbol | is used to indicate different options. It shows that a variable symbol can be replaced in several ways, with each option separated by the vertical bar. For instance, the variable < expr > can be replaced by any of the following four options: < expr >< op >< expr > as specified by rule number ($RN = 0$), (< expr >< op >< expr >) as specified by $RN = 1$, < pre-op > (< expr >) as specified by $RN = 2$, and < var > as specified by $RN = 3$.

To convert an integer vector into a mathematical expression, the leftmost derivation rule is employed [26]. In context-free grammars, a leftmost derivation is a sequence of production rule applications that begins with the start symbol S_0 and produces a specific string of terminal symbols, which forms a mathematical expression. During a leftmost derivation, the leftmost variable symbol in the current string is replaced by one of its production rules at each step. This replacement process continues until all non-terminal (variable) symbols are substituted by terminal symbols, resulting in a mathematical expression. The transformation process using the leftmost derivation for an integer vector $v = [a_0, a_1, \ldots, a_n]$ is outlined as follows:

1. Start with $i = 0$ and initialize at $S_0 =<$ expr $>$.
2. Repeat the following steps until either the sequence concludes or an expression made entirely of terminal symbols is obtained:
   (a) Identify the rule number (RN) for the current variable as follows:
   $$\text{RN} = a_i,$$
   (b) Replace the current variable symbol with the corresponding production rule determined by RN.
   (c) Increase the counter i by one:
   $$i = i + 1.$$

For clarity, we provide an example of converting an integer vector into a mathematical expression. Table 1 illustrates the transformation of the vector $v = [0, 3, 1, 0, 0, 0, 3, 1, 3, 3, 0, 3, 3, 0]$ into a mathematical expression using grammar G_0. After reindexing the parameters w in the expression obtained in Table 1, we will have the following expression:

$$y(x) = w_0 + w_1 x^2. \tag{3.5}$$

Expression	Rule Number (RN)	Production rule
$<\mathbf{expr}>$	$RN = a_0 = 0$	$<expr>\rightarrow<expr><op><expr>$
$<\mathbf{expr}><op><expr>$	$RN = a_1 = 3$	$<expr>\rightarrow<var>$
$<\mathbf{var}><op><expr>$	$RN = a_2 = 1$	$<var>\rightarrow w$
$w<\mathbf{op}><expr>$	$RN = a_3 = 0$	$<op>\rightarrow +$
$w+<\mathbf{exp}>$	$RN = a_4 = 0$	$<expr>\rightarrow<expr><op><expr>$
$w+<\mathbf{exp}><op><expr>$	$RN = a_5 = 0$	$<expr>\rightarrow<expr><op><expr>$
$w+<\mathbf{exp}><op><exp><op><expr>$	$RN = a_6 = 3$	$<expr>\rightarrow<var>$
$w+<\mathbf{var}><op><exp><op><expr>$	$RN = a_7 = 1$	$<var>\rightarrow w$
$w+w<\mathbf{op}><exp><op><expr>$	$RN = a_8 = 3$	$<op>\rightarrow \times$
$w+w\times<\mathbf{expr}><op><expr>$	$RN = a_9 = 3$	$<expr>\rightarrow<var>$
$w+w\times<\mathbf{var}><op><expr>$	$RN = a_{10} = 0$	$<var>\rightarrow x$
$w+w\times x<\mathbf{op}><expr>$	$RN = a_{11} = 3$	$<op>\rightarrow \times$
$w+w\times x\times<\mathbf{expr}>$	$RN = a_{12} = 3$	$<expr>\rightarrow<var>$
$w+w\times x\times<\mathbf{var}>$	$RN = a_{13} = 0$	$<var>\rightarrow x$
$w+w\times x\times x$	-	-

Table 1: Transforming the vector $v = [0, 3, 1, 0, 0, 0, 3, 1, 3, 3, 0, 3, 3, 0]$ into a mathematical expression using crammer G_0.

3.2.2 Monte Carlo tree search based on context-free grammars for symbolic solution to the Lane-Emden type

Monte Carlo tree search is a powerful algorithm used for making decisions in artificial intelligence. It combines the principles of tree search and random sampling to evaluate and optimize potential moves. The process involves building a search tree incrementally and using random simulations to estimate the value of each node. MCTS consists of four main stages: selection, expansion, simulation, and backpropagation [28]. Each stage plays a crucial role in navigating the search space, improving the accuracy of the decision-making process, and ultimately guiding the algorithm towards optimal choices. The main stages of the MCTS algorithm are as follows [45]:

- Selection: This stage involves traversing the current tree from the root to a leaf node, choosing child nodes based on a selection policy, typically one that balances exploration and exploitation.
- Expansion: Once a leaf node is reached, the algorithm expands the tree by adding one or more child nodes, representing possible future states of the decision process.
- Rollout (Simulation): During this stage, a simulation is run from the newly added node to a terminal state, using a default policy to make decisions at each step. The outcome of this simulation provides an estimate of the value of the node.
- Backpropagation: The final stage updates the nodes along the path from the expanded node back to the root, propagating the simulation results upwards to refine the value estimates and improve future selections.

In this paper, we present an MCTS algorithm based on grammar G_0 for finding a symbolic expression that fits the numerical solution from Phase 1. To achieve this, the MCTS algorithm is designed to identify the optimal sequence of production rules (sequence of RN) that yields a symbolic expression closely matching the numerical solution produced by the deep RNN, as illustrated in Figure 3. In our study, we treat the optimal sequence of substantiation rules as a path in a tree, beginning from the root state $Root =< expr >$. Each node of the tree is considered as follows:

$$
\begin{aligned}
&Node = \\
&\quad \{ \\
&\quad state && \text{\\A list of integers from the set \{0, 1, 2, 3\} representing the actions } (RN) \text{ taken} \\
&\quad && \text{\\to reach this node thus far.} \\
&\quad Parent && \text{\\Represents the parent of this node.} \\
&\quad Children && \text{\\Represents the child nodes of this node.} \\
&\quad Value && \text{\\Represents the value of this node.} \\
&\quad n && \text{\\Represents the number of times this node has been visited.} \\
&\quad is_leaf && \text{\\A boolean flag indicating whether this node is a leaf node.} \\
&\quad n_actions && \text{\\This represents the number of actions available for a given state.} \\
&\quad && \text{\\The number is calculated based on the expression derived from the state list by} \\
&\quad && \text{\\applying the transformation procedure explained in Subsection 3.2.1.} \\
&\quad && \text{\\It is 4 if the first variable of the string on the left side is} \\
&\quad && \text{\\} < expr >, < op >, \text{ or } < pre-p >; \text{ otherwise, it is 2.} \\
&\quad Termination && \text{\\A boolean flag indicating whether this node is a terminal node.} \\
&\quad && \text{\\A node is considered terminal if the expression derived from its state list} \\
&\quad && \text{\\results in a string without variable symbols.} \\
&\quad \}
\end{aligned}
\tag{3.6}
$$

In this study, during the rollout and expansion stages, a branch terminates if a terminal node is reached or a predefined maximum depth is achieved. A node is considered terminal if the expression derived from its state list results in a string without variable symbols. In the rollout stage, if a branch terminates due to a terminal node, the value of that branch is evaluated, followed by the backpropagation stage. Figure 4 illustrates how a branch in the tree leads to a mathematical expression. In this study, the tree begins with the root node S_0 and its children nodes S_1, S_2, S_3, and S_4. For illustration purposes, we present the node values of three actions in the branch, shown in green, in detail. In

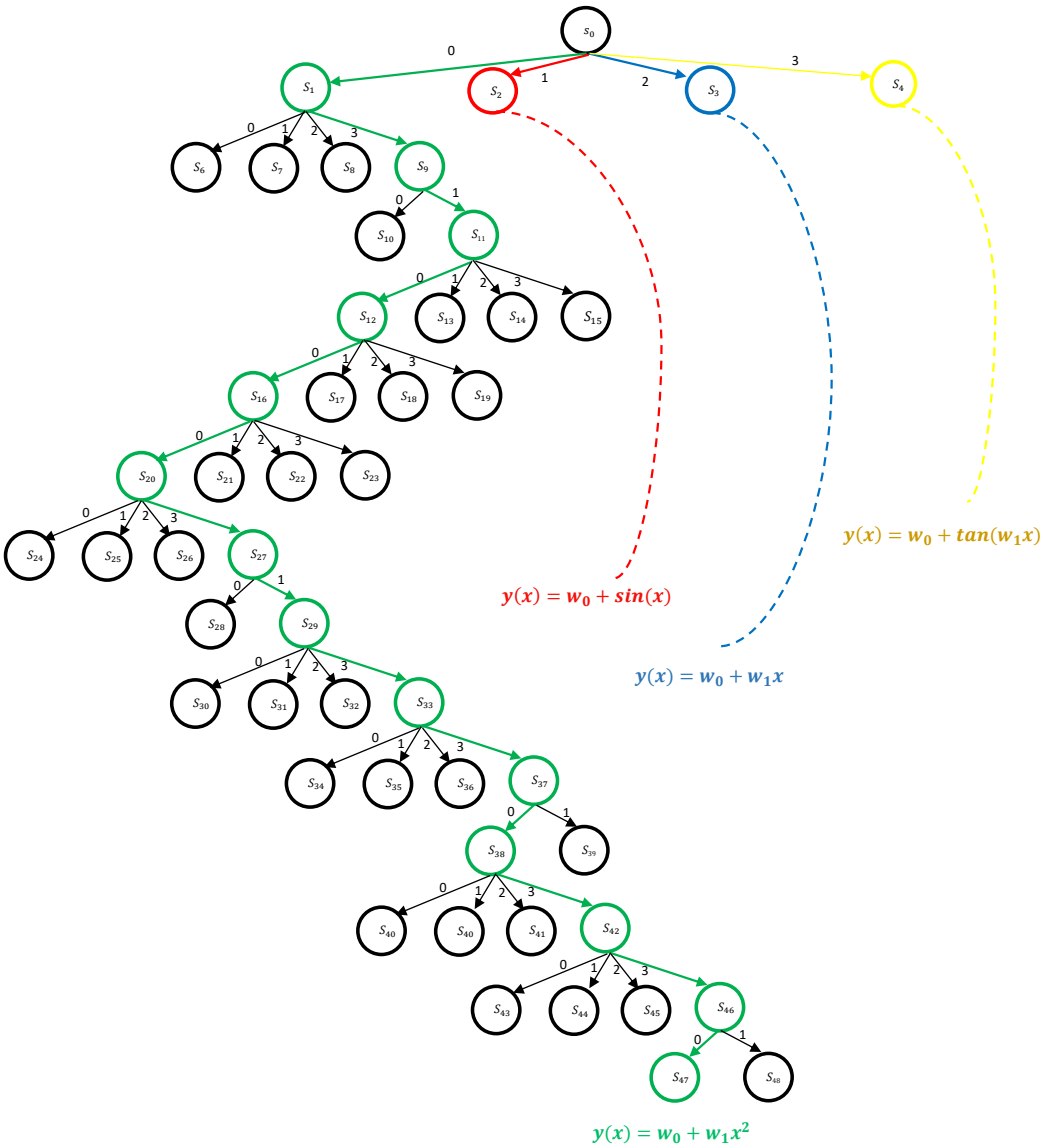


Figure 4: Derivation of a mathematical expression in the rollout stage in a tree structure.

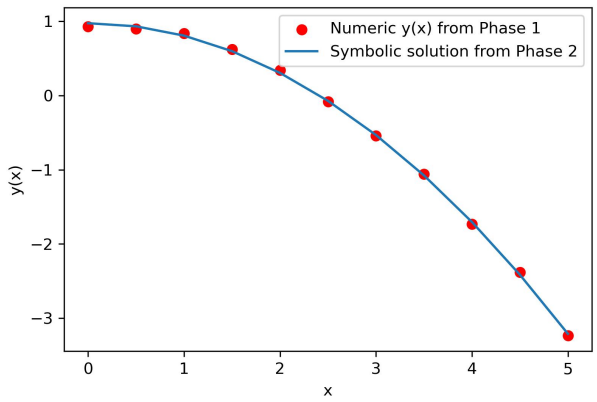


Figure 5: The symbolic expression obtained in a rollout (the green branch in Figure 4) and the numeric values shown in Table 6.

the first iteration, the node S_0 is as follows:

$$
\begin{aligned}
&S_0 = \\
&\quad \{ \\
&\quad state = Null \\
&\quad Parent = Null \\
&\quad Children = \{S_1, S_2, S_3, S_4\} \\
&\quad Value = 0 \\
&\quad n = 1 \\
&\quad is_leaf = False \\
&\quad n_actions = 4 \\
&\quad Termination = False \\
&\quad \}
\end{aligned}
$$

In node S_0, there are four possible choices because the expression derived at this node is $< expr >$. In the first iteration of the algorithm, all these possible nodes are added to node S_0. Then the branching starts. Suppose action 0 is chosen at this node. Consequently, the next node, S_1, will be as follows:

$$
\begin{aligned}
&S_1 = \\
&\quad \{ \\
&\quad state = [0] \\
&\quad Parent = S_0 \\
&\quad Possible \;\; children = \{S_6, S_7, S_8, S_9\} \\
&\quad Value = 0 \\
&\quad n = 1 \\
&\quad is_leaf = False \\
&\quad n_actions = 4 \\
&\quad Termination = False \\
&\quad \}
\end{aligned}
$$

In node S_1, there are four possible choices because the expression derived at this node is $< expr >< op >< expr >$, and the left variable of this expression is $< expr >$. Suppose action 3 is chosen at node S_1. Consequently, the next node, S_9, will be as follows:

$$
\begin{aligned}
&S_9 = \\
&\quad \{ \\
&\quad state = [0, 3] \\
&\quad Parent = S_2 \\
&\quad Possible \;\; children = \{S_{10}, S_{11}\} \\
&\quad Value = 0 \\
&\quad n = 1 \\
&\quad is_leaf = False \\
&\quad n_actions = 2 \\
&\quad Termination = False \\
&\quad \}
\end{aligned}
$$

In node S_9, there are two possible choices because the expression derived at this node is $< var >< op >< expr >$, and the left variable of this expression is $< var >$. Suppose action 1 is chosen at this node. The next node will be S_{11}. The procedure continues until the terminal node S_{47} is reached. This is a terminal node because the mathematical expression associated with the state of this node is $w_0 + w_1 x^2$. In this expression, there is no variable symbol.

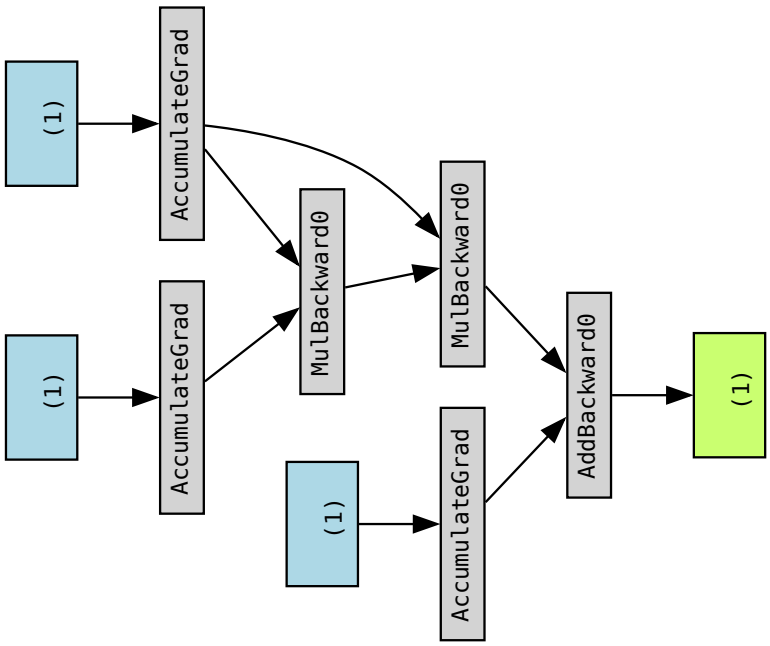


Figure 6: The computational graph generated in PyTorch for Expression (3.5)

Suppose this branch is generated in a rollout stage starting from state S_1. After this rollout stage, the value of this branch is calculated. To calculate the value of a branch resulting in a mathematical expression, we treat the computation graph of the expression as a network and apply a regression task to optimize its parameters. Specifically, we train the network so that the loss value (mean squared error between numerical values obtained in Phase 1 and the values obtained from the mathematical expression) is minimized. For example, Figure 6 illustrates the computational graph of Expression (3.5) using the following Python code:

```
import torch
from torchviz import make_dot
x=torch.tensor([1.],requires_grad=True)
w0=torch.tensor([1.],requires_grad=True)
w1=torch.tensor([1.],requires_grad=True)
y=w0+w1*x*x
make_dot(y).view()
```

For illustration purposes, we have simplified the network depicted in Figure 6. The resulting simplified network, generated in PyTorch, is shown in Figure 7.

The value of a branch is equal to $\frac{1}{loss}$ after training the network (computation graph) obtained from a mathematical expression.

Let us consider a nonlinear ordinary equation as follows:

$$y''(x) + \frac{2}{x}y'(x) + 1 = 0, \tag{3.7}$$

with initial conditions:

$$\begin{aligned} y'(0) &= 0, \\ y(0) &= 0. \end{aligned}$$

Eq. (3.7) can be rewritten as follows:

$$xy''(x) + 2y'(x) + x = 0, \tag{3.8}$$

x_i	$f(x_i)$
0.0	0.92871940135955810547
0.5	0.89662259817123413086
1.0	0.83621585369110107422
1.5	0.62077730894088745117
2.0	0.34092375636100769043
2.5	-0.08404587209224700928
3.0	-0.54181569814682006836
3.5	-1.05920553207397460938
4.0	-1.73174142837524414062
4.5	-2.37870573997497558594
5.0	-3.23180031776428222656

Table 2: The numerical values for $x \in [0, 5]$ with a step size $h = 0.5$, obtained in Phase 1 for Eq. (3.8).

with initial conditions:

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

Suppose the numerical values for $x \in [0, 5]$ with a step size $h = 0.5$, obtained in Phase 1 for Eq. (3.8), are as shown in Table 2. The following Python code applies a regression task to the values shown in Table 2 and the expression $y(x) = w_0 + w_1 x^2$. It optimizes the parameters of the network shown in Figure 7 for calculating the loss and, consequently, the value of the branch.

```
import torch
import numpy as np
import matplotlib.pyplot as plt
import torch.nn as nn
torch.set_printoptions(precision=20, threshold=None, edgeitems=None, linewidth=None, profile=None,
                                                    sci_mode=None)
x = torch.linspace(0.0, 5, 11, requires_grad=True)
f=torch.tensor([ 0.92871940135955810547,0.89662259817123413086,0.83621585369110107422,0.
                                                    62077730894088745117, 0.34092375636100769043, -0.
                                                    08404587209224700928,-0.54181569814682006836, -1.
                                                    05920553207397460938,-1.73174142837524414062,-2.
                                                    37870573997497558594,-3.23180031776428222656],
                                                    requires_grad=True,dtype=torch.float)
w=torch.tensor([0.01*torch.rand(1),0.01*torch.rand(1)],requires_grad=True,dtype=torch.float);
loss_fn = nn.MSELoss()
optimizer = torch.optim.Adam([w], lr=0.01)
for i in range(1000):
    model=w[0]+w[1]*x**2
    loss=loss_fn(model,f)
    optimizer.zero_grad(set_to_none=True)
    loss.backward(retain_graph=True)
    optimizer.step()
# Visualization
f_approximatd=w[0]+w[1]*x**2
f_approximatd_numpy=[i.detach().numpy() for i in f_approximatd]
f_numpy=[i.detach().numpy() for i in f]
xx=[i.detach().numpy() for i in x]
plt.scatter(xx,f_numpy,c='red',)
plt.plot(xx,f_approximatd_numpy)
plt.xlabel('x')
plt.ylabel('y(x)')
plt.legend(['Numeric y(x) from Phase 1', 'Symbolic solution from Phase 2'])
plt.savefig('Fig_numeric_and_exact.jpg',dpi=300)
plt.show()
# Printing the parameters and the value
print('parameters=',w)
print('Value=',1/loss.item())
```

In a run of this code on our computer, the parameters w_0 and w_1 were optimized to be 0.97203069925308227539 and -0.16723527014255523682, respectively. Furthermore, the value of the rollout (the green branch in Figure 4) was 1203.8234497484318. It is important to note that these values might vary slightly across different runs. Figure 5

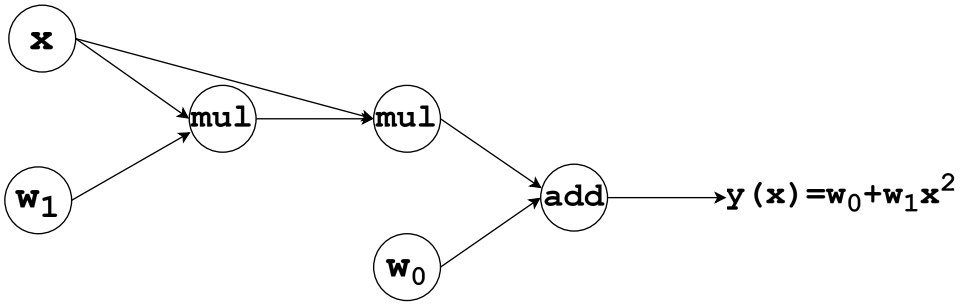


Figure 7: The simplified computational graph illustrated in Figure 6

presents the symbolic expression obtained in this rollout and the numeric values shown in Table 2. After calculating the value of this branch, the propagation stage is completed and the values of nodes S_0 and S_1 are updated.

Figure 4 illustrates another rollout examples. In this figure, it was assumed that in three other rollouts starting from nodes S_2, S_3, and S_4, the expressions $w_0 + \sin(x)$, $w_0 + w_1 x$, and $w_0 + \tan(w_1 x)$ were obtained, respectively. The values of the rollouts obtained by executing the previous Python code were 1.4913418926384554, 6.76866712692903, and 8.027632941604292, respectively. Note that in the previous code, the variable "model" should be updated to the expressions "w[0] + torch.sin(x)", "w[0] + w[1]*x", and "w[0] + torch.tan(w[1]*x)" respectively, for the rollouts shown in red, blue, and yellow branches in Figure 4.

Figure 8 provides a visual representation of the flowchart for the MCTS algorithm employed in this paper.

In this study, during the expansion stage, all possible actions from the node are added as children to that node. In the rollout stage, a branch is selected randomly until a terminal node is reached. Then, the value of the branch is evaluated as mentioned previously. In the backpropagation stage, the values of the current node and the nodes along the path from the current node to the root node S_0 are updated based on the value of the rollout. Additionally, the visit counts of these nodes are incremented by 1. The selection stage employs the Upper Confidence Bound (UCB) formula to evaluate and choose the best child among the node's children. The UCB formula helps in selecting the next node to explore by balancing exploration and exploitation rates. The UCB formula used in this study is as follows:

$$a^* = \max_{a \in A(S)} \frac{Q(S,a)}{n(a)} + \eta \sqrt{\frac{\log N}{n(a)}}. \tag{3.9}$$

Here, S is the node from which the selection is made, $A(S)$ is the set of available actions from the state S, $Q(S,a)$ represents the value of the state S' if action a is taken in state S and the state S' is reached, $n(a)$ is the number of times S' has been visited if action a is taken in state S and the state S' is reached, N represents the number of iterations, and η represents the exploration and exploitation hyperparameter tuning.

The flowchart shown in Figure 8 has a nested loop. Before the nested loop starts, an expansion stage is executed, making the tree consist of a root node and its four child nodes. At the beginning of this flowchart, S_0 is the root node. Then, the nested loop starts. The outer loop counter is depicted by D. The outer loop repeats until the counter D reaches the maximum depth represented by N_{Depth}. After completing all iterations of the inner loop, the best child among the children of S_0 is selected as the next S_0, and the inner loop starts again from the new S_0. Essentially, the inner loop is responsible for finding the best node to be selected among the children of S_0.

The inner loop counter is denoted by I. The inner loop repeats until the counter I reaches the maximum iteration number represented by $N_{Iterations}$. In this flowchart, the current node is depicted by C. Additionally, C_n represents the number of times the current node C has been visited. At each iteration of the inner loop, a rollout stage is executed. As shown in the flowchart, if the current node C is not a leaf node, the selection stage continues until C becomes a leaf node. If the current node C is a leaf node and has not been visited before, a rollout stage is followed by the backpropagation stage. If the current node C is a leaf node and has been visited before, an expansion stage is conducted. Then, the leftmost node among the children of the current node becomes the new current node. After that, the rollout stage is followed by the backpropagation stage.

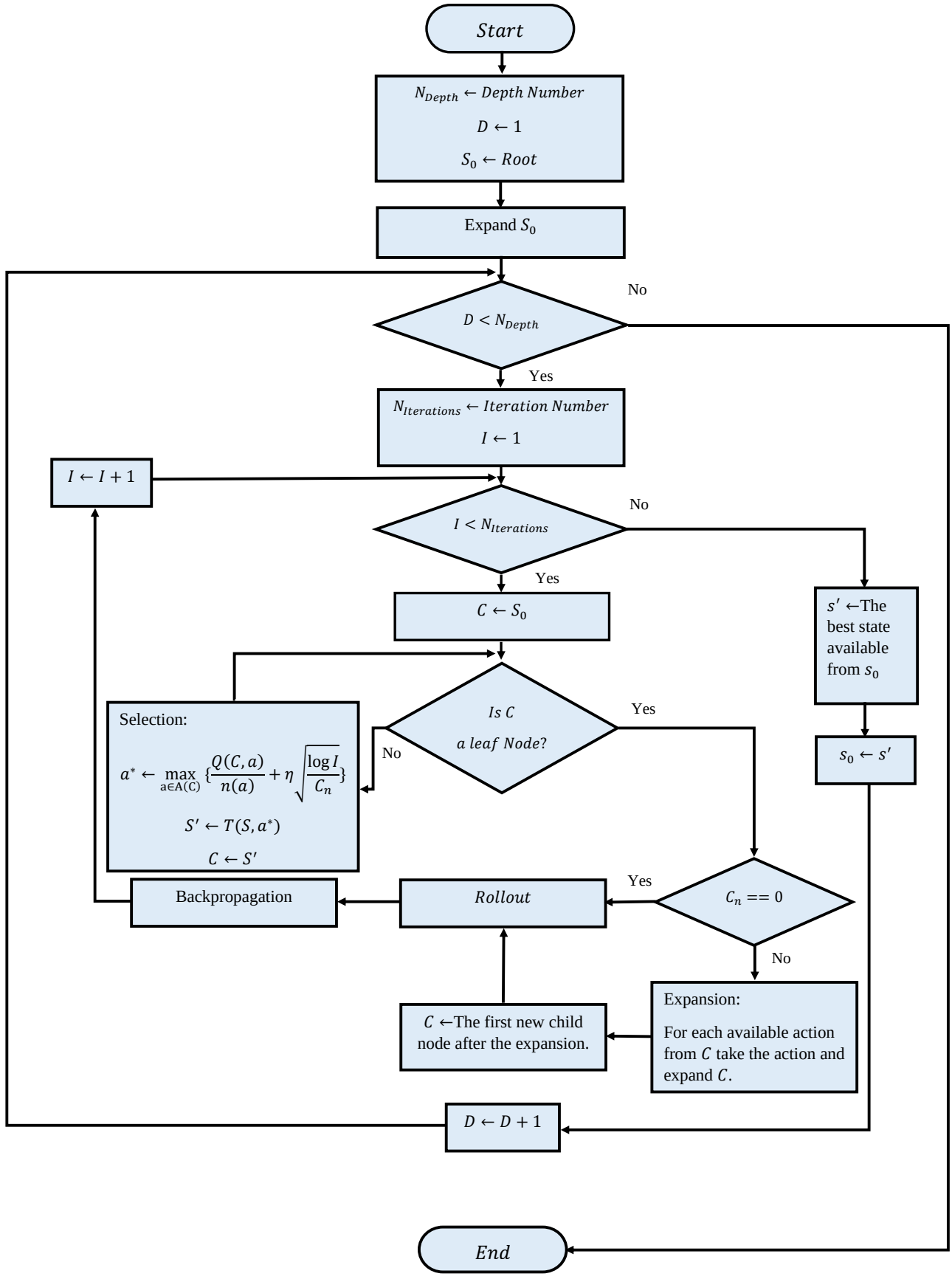


Figure 8: Flowchart of the Monte Carlo tree search algorithm employed in Phase 2 to derive symbolic expressions from numerical values obtained in Phase 1.

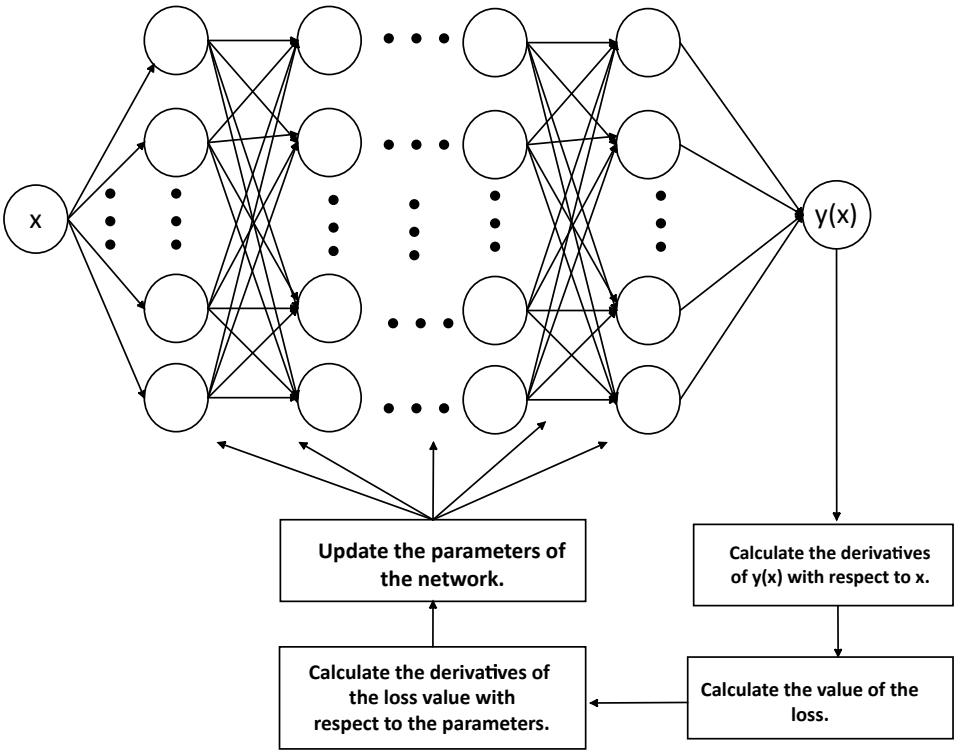


Figure 9: The general form of fully connected PINNs for solving ODES.

3.3 Phae 3: Utilizing physics-informed neural networks to fine-tune coefficients in mathematical expressions

A physics-informed neural network is a specialized neural network designed to solve differential equations by embedding physical laws directly into its training process. Unlike conventional neural networks that rely purely on data for training, PINNs integrate the governing equations of the physical system [37]. As a result, even with limited data, PINNs can produce accurate and generalizable solutions, constrained by the inherent physical laws of the problem. A key application of PINNs is in solving ordinary differential equations. In this role, PINNs are particularly effective because they can manage complex boundary conditions and non-linearities that pose challenges for traditional numerical methods. For instance, ODEs are commonly encountered in engineering and physics for modeling dynamic systems, such as the movement of particles under various forces or the temperature changes in materials. PINNs solve these ODEs by training a neural network to approximate the solution, with a loss function that minimizes the difference between the network's output and the ODE residuals. Figure 9 illustrates the general form of fully connected PINNs for solving ODEs. The general form of an ordinary differential equation of order n is as follows [46]:

$$F(x, y(x), y'(x), y''(x), \dots, y^{(n)}(x)) = 0, \tag{3.10}$$

with initial conditions:

$$y(x_0) = y_0, \quad y'(x_0) = y'_0, \quad \dots \quad , y^{(n-1)}(x_0) = y_0^{(n-1)}.$$

To solve the ordinary differential equations depicted in Eq. (3.10) using a fully connected PINN, as illustrated in Figure 9, it is essential to define the loss function appropriately. The loss function for this purpose is formulated as follows:

$$loss_{ode} = \sum_{i=1}^{m} F(x_i, \hat{y}(x_i), \hat{y}'(x_i), \hat{y}''(x_i), \dots, \hat{y}^{(n)}(x_i))^2, \tag{3.11}$$

$$loss_{initial} = (y_0 - \hat{y}_0)^2 + (y'_0 - \hat{y}'_0)^2 + \dots + (y_0^{(n-1)} - \hat{y}_0^{(n-1)})^2,$$

$$loss = loss_{ode} + loss_{initial}.$$

In this paper, to fine-tune the coefficients of a mathematical expression using the PINN approach, we treat the computational graph of the expression generated in PyTorch as a network. We then optimize the parameters (the

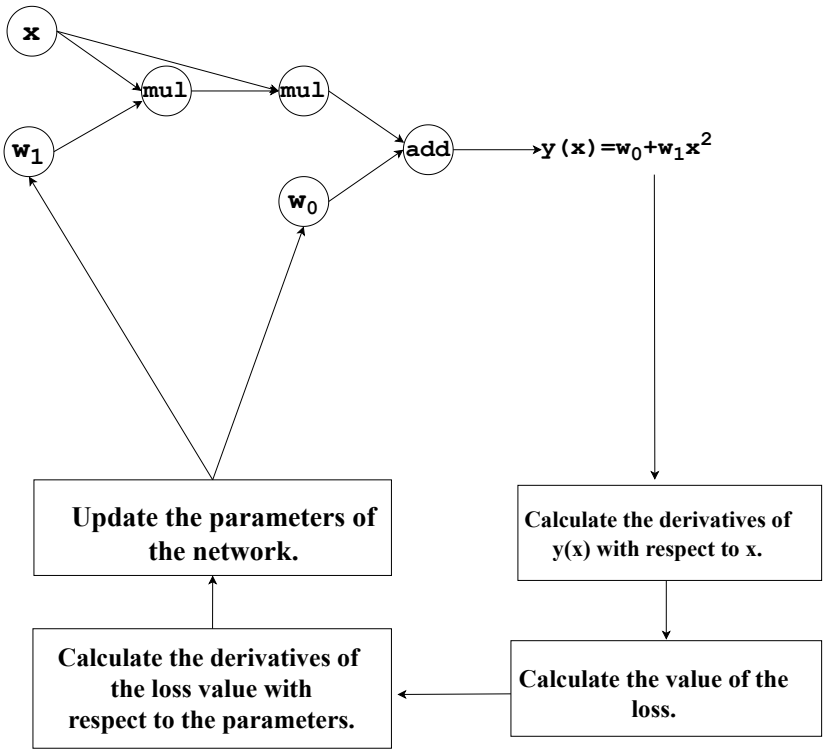


Figure 10: optimize the coefficients of the network shown in Figure 7 using the PINN approach so that it becomes a solution to Eq. (3.8).

coefficients of the mathematical expression) of this network (computational graph) to minimize the loss function. The lower the values of the loss function, the more accurate the symbolic solution for the differential equation under consideration. Suppose, the nonlinear ordinary differential equation under consideration is Eq. (3.8). Suppose in Phase 2 the MCTS has found the following sequence of integers to solve Eq. (3.8) as shown in the green branch in Figure 4:

$$v = [0, 3, 1, 0, 0, 0, 3, 1, 3, 3, 0, 3, 3, 0]. \tag{3.12}$$

The mathematical expression derived from vector v shown in (3.12) is $y(x) = w_0 + w_1 x^2$. Furthermore, we illustrated that the computational graph of this expression is as shown in Figure 7. Now, we aim to optimize the coefficients of this expression using the PINN approach so that it becomes an accurate solution to Eq. (3.8). To achieve this, we will use the architecture shown in Figure 10. The loss function associated with Eq. (3.8) and the architecture depicted in Figure 10 is as follows:

$$\begin{aligned} loss_{ode} &= \sum_{i=1}^{m} \left(x_i y''(x_i) + 2y'(x_i) + x_i\right)^2, \\ loss_{initial} &= \left(y(0) - 1\right)^2 + \left(y'(0) - 0\right)^2, \\ loss &= \gamma_1 loss_{ode} + \gamma_2 loss_{initial}. \end{aligned} \tag{3.13}$$

Here, γ_1 and γ_2 are hyperparameters that should be tuned to optimal values. The following code applies the PINN approach to fine-tune the coefficients of the expression $y(x) = w_0 + w_1 x^2$ so that it serves as a symbolic solution to Eq. (3.8). The initial values for w_0 and w_1 were set to 0.97203069925308227539 and -0.16723527014255523682 respectively (the values obtained in the previous subsection), although these initial values are not critical. The optimization algorithms (specifically Adam optimization from the PyTorch package) can effectively determine optimal values provided that hyperparameters such as the learning rate, γ_1, and γ_2 are appropriately chosen.

```
import torch
gamma1=1.0
gamma2=1.0
```

```
torch.set_printoptions(precision=20, threshold=None, edgeitems=None, linewidth=None, profile=None,
                                                    sci_mode=None)
def dy_dx(y, x):
    return torch.autograd.grad(
        y, x, grad_outputs=torch.ones_like(y), create_graph=True,allow_unused=True
    )[0]
x = torch.linspace(0.0, 5, 501, requires_grad=True)
w=torch.tensor([0.97203069925308227539,-0.16723527014255523682],requires_grad=True,dtype=torch.float
                                                    );
optimizer = torch.optim.Adam([w], lr=0.0001)
for i in range(10000):
    y=w[0]+w[1]*x*x
    y_p = dy_dx(y, x)
    y_pp = dy_dx(y_p, x)
    residual = x*y_pp+2.0*y_p+x
    initial1 =(y[0]-1)**2
    initial2= (y_p[0])**2
    loss = (residual**2).mean() + gamma1*initial1+gamma2*initial2
    optimizer.zero_grad(set_to_none=True)
    loss.backward(retain_graph=False)
    optimizer.step()
    del y_p
    del y_pp
    optimizer.zero_grad(set_to_none=True)
print('parameters=',w)
```

In a run of this code, the parameters w_0 and w_1 were determined to be 1.0000000000000000000 and -0.16666665673255920410 respectively. It is important to note that these values may vary slightly across different runs. Therefore, the symbolic expression would be as follows:

$$y(x) = 1.0000000000000000000 - 0.16666665673255920410x^2.$$

Where the exact solution of Eq. (3.8) is as follows:

$$y(x) = 1 - 0.16666666666666666667x^2.$$

In fact, if in Phase 1 the RNN finds an accurate numeric solution, and in Phase 2 the MCTS identifies a suitable expression for this numeric solution, then in Phase 3, the PINN approach can optimize the coefficients accurately.

4 Experimental Results

In this section, we show the efficacy of our novel hybrid approach in solving complex nonlinear ordinary differential equations. Specifically, we consider the Lane-Emden equation, which has real-world applications in astrophysics and is a challenging nonlinear ordinary differential equation. The standard Lane-Emden equation is formulated as follows:

$$y''(x) + \frac{2}{x}y'(x) + y^m(x) = 0, \quad x \geq 0, \tag{4.1}$$

with initial conditions:

$$y(0) = 1, \quad y'(0) = 0. \tag{4.2}$$

Here, m is a constant real number. In this paper, we consider $m \in \{0, 1, 2, 3, 4, 5\}$. For $m = 0, 1, 5$, the exact solutions are $y(x) = 1 - \frac{x^2}{6}$, $y(x) = \frac{\sin(x)}{x}$, and $y(x) = \left(1 + \frac{x^2}{3}\right)^{-\frac{1}{2}}$, respectively [32].

Eq. (4.1) can be rewritten as follows:

$$xy''(x) + 2y'(x) + xy^m(x) = 0, \tag{4.3}$$

with initial conditions:

$$y(0) = 1, \quad y'(0) = 0. \tag{4.4}$$

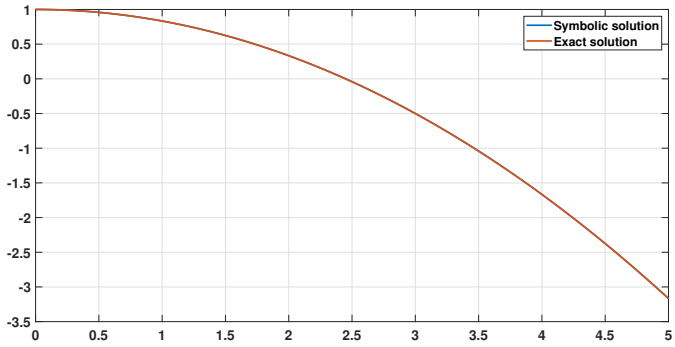


Figure 11: The exact and symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 0$.

Since there is no exact solution for certain cases of Eq. (4.3), the equation is challenging to solve. The primary reason for choosing this equation is that solving it demonstrates the capability of our proposed method in addressing nonlinear differential equations. In this section, we will present different cases (with m taking values in $\{0, 1, 2, 3, 4, 5\}$) of Eq. (4.3) in several subsections.

4.1 Results of the proposed method for the standard Lane-Emden equation with $m = 0$

The solution obtained from our proposed method for the standard Lane-Emden equation with $m = 0$ is given by:

$$y(x) = ((w[0] * (x * x)) + w[1] * w[2]). \tag{4.5}$$

Table 3 lists the values of the coefficients in Eq. (4.5).

Table 3: Values of the coefficients in Eq. (4.5).

Coefficients	Values
w[0]	-0.16666665673255920410
w[1]	-0.99982750415802001953
w[2]	-1.00017178058624267578

After simplifying Eq. (4.5), we obtain the following symbolic solution:

$$y(x) = 0.99999925511282583556749159470201 - 0.16666665673255920410156 25x^2,$$

where the exact solution is:

$$y(x) = 1.0 - 0.16666666666666666666666666666667x^2.$$

The mean absolute error (MAE) between the exact solution and the symbolic solution derived by our method for $x \in [0, 5]$ with a step size $h = 0.01$ is MAE = 0.000000662020. Figure 11 illustrates both the exact solution and the symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 0$.

4.2 Results of the proposed method for the standard Lane-Emden equation with $m = 1$

The solution obtained from our proposed method for the standard Lane-Emden equation with $m = 1$ is given by:

$$y(x) = cos(x * sin(sin(exp(((w[0] * x) * x - w[1] - w[2]))))). \tag{4.6}$$

Table 4 shows the values of the coefficients in Eq. (4.6).

After simplifying Eq. (4.6), we obtain the following symbolic solution:

$$y(x) = cos(xsin(sin(e^{-0.027110999450087547302246093 75x^2 - 0.32848583161830902099609375}))),$$

Table 4: Values of the coefficients in Eq. (4.6).

Coefficients	Values
w[0]	-0.02711099945008754730
w[1]	0.16487455368041992188
w[2]	0.16361127793788909912

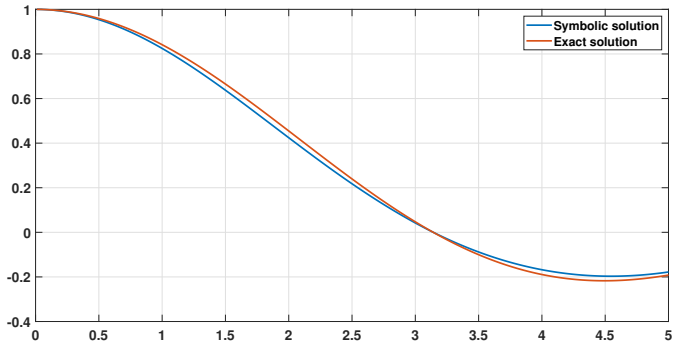


Figure 12: The exact and symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 1$.

where the exact solution is:

$$y(x) = \frac{sin(x)}{x}.$$

The mean absolute error between the exact solution and the symbolic solution derived by our method for $x \in [0.01, 5]$ with a step size $h = 0.01$ is MAE $= 0.016741208893$. Figure 12 illustrates both the exact solution and the symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 1$.

4.3 Results of the proposed method for the standard Lane-Emden equation with $m = 2$

The solution obtained from our proposed method for the standard Lane-Emden equation with $m = 2$ is given by:

$$y(x) = cos(log(x - w[0]/cos(w[1] * w[2] * w[3]. \tag{4.7}$$

Table 5 lists the values of the coefficients in Eq. (4.7).

Table 5: Values of the coefficients in Eq. (4.7).

Coefficients	Values
w[0]	-0.77601307630538940430
w[1]	0.92973065376281738281
w[2]	0.93009436130523681641
w[3]	0.62315535545349121094

After simplifying Eq. (4.7), we obtain the following symbolic solution:

$$y(x) = cos(0.9604143993272 * log(x + 0.7760130763054)),$$

There is no exact solution available for the standard Lane-Emden equation with $m = 2$. However, the researcher in [21] has provided the exact value of the first root for this case as $x = 4.3528746$. The first root of the symbolic solution obtained using the proposed method is $x = 4.3562164316897$. The absolute error (AE) between the exact and approximated first root is $AE = 0.003341832$.

Figure 13 illustrates the symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 2$.

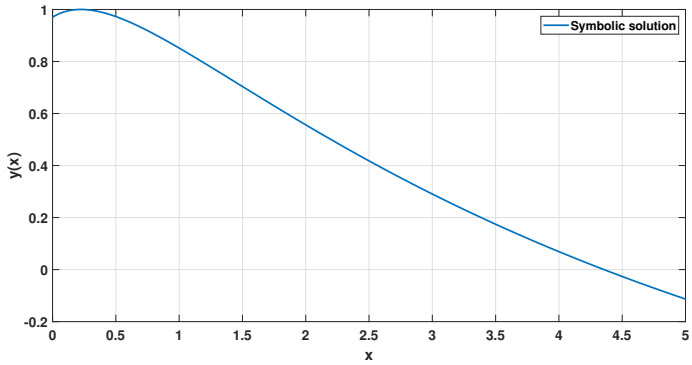


Figure 13: The symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 2$.

4.4 Results of the proposed method for the standard Lane-Emden equation with $m = 3$

The solution obtained from our proposed method for the standard Lane-Emden equation with $m = 3$ is given by:

$$y(x) = cos(log(w[0]/w[1] + ((w[2] + w[3] * x))). \tag{4.8}$$

Table 6 presents the values of the coefficients in Eq. (4.8).

Table 6: Values of the coefficients in Eq. (4.8).

Coefficients	Values
w[0]	0.00697763031348586082
w[1]	0.00644001364707946777
w[2]	0.26959139108657836914
w[3]	0.27137738466262817383

After simplifying Eq. (4.8), we obtain the following symbolic solution:

$$y(x) = cos(log(0.540968775749x + 1.083480671916)),$$

There is no true solution for the standard Lane-Emden equation with $m = 3$. However, the researcher in [21] has provided exact values for specific interior points, as displayed in Table 7. We calculated the mean absolute error between the values obtained from the symbolic solution using our proposed method and the exact values listed in Table 7. The mean absolute error is $MAE = 0.029899779486$.

Figure 14 presents the symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 3$.

Table 7: A comparison between the exact values provided by [21] and those obtained using our proposed method for the standard Lane-Emden equation with $m = 3$.

x	Exact value	Our method	Absolute error
0.0	1.0000000	0.996787409369	0.003212590631
0.1	0.9983358	0.991703757033	0.006632042967
0.5	0.9598391	0.954434479819	0.005404620180
1.0	0.8550576	0.884596167345	0.029538567345
5.0	0.1108198	0.236607269177	0.125787469177
6.0	0.0437380	0.105197004001	0.061459004001
6.8	0.0041678	0.010114053805	0.005946253805
6.9	0.0041678	0.001181687779	0.001217687779

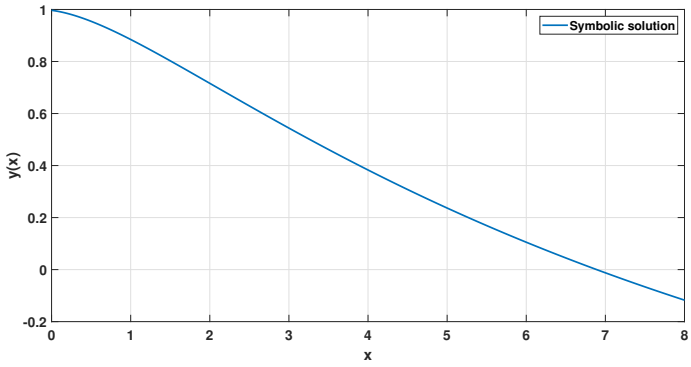


Figure 14: The symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 3$.

4.5 Results of the proposed method for the standard Lane-Emden equation with $m = 4$

The solution obtained from our proposed method for the standard Lane-Emden equation with $m = 4$ is as follows:

$$y(x) = sin(exp((w[0] + (w[1]/w[2]/(x + w[3] + w[4] * x)))); . \tag{4.9}$$

Table 8 shows the values of the coefficients in Eq. (4.9).

Table 8: Values of the coefficients in Eq. (4.9).

Coefficients	Values
w[0]	0.45131763815879821777
w[1]	0.00737074948847293854
w[2]	0.00980063341557979584
w[3]	0.63002085685729980469
w[4]	0.63464415073394775391

After simplifying Eq. (4.9), we will have the following symbolic solution:

$$y(x) = sin(e^{\frac{0.0000000019(5419570198146423.0x+2570499852860048.0)}{8388608.0x+10608779.0}}) \tag{4.10}$$

There is no exact solution for the standard Lane-Emden equation with $m = 4$. However, the researcher in [21] has provided exact values for specific interior points, as shown in Table 9. We computed the mean absolute error between the values derived from the symbolic solution using our proposed method and the exact values listed in Table 9. The mean absolute error is $MAE = 0.016438187787$. Figure 15 illustrates the symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 4$.

Table 9: A comparison between the exact values provided by [21] and those obtained using our proposed method for the standard Lane-Emden equation with $m = 4$.

x	Exact value	Our method	Absolute error
0.0	1.0000000	0.999999913342	0.000000086658
0.1	0.9983367	0.996081419312	0.002255280688
0.2	0.9933862	0.985681793093	0.007704406907
0.5	0.9603109	0.931404103395	0.028906796605
1.0	0.8608138	0.814970232593	0.045843567407
5.0	0.2359227	0.275844514080	0.039921814080
10.0	0.0596727	0.079861040789	0.020188340789
14.0	0.0083305	0.011451413470	0.003120913471
14.9	0.0005764	0.000573916524	0.000002483476

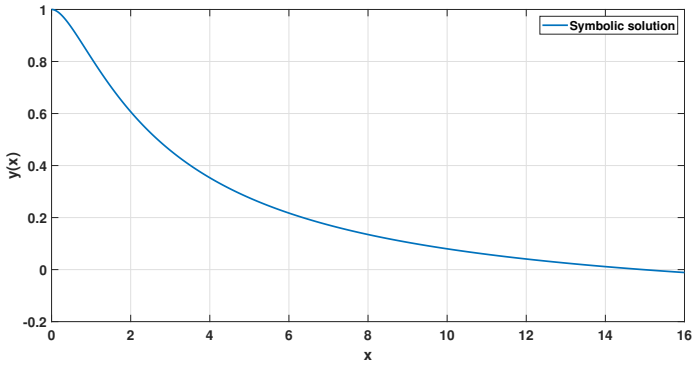


Figure 15: The symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 4$.

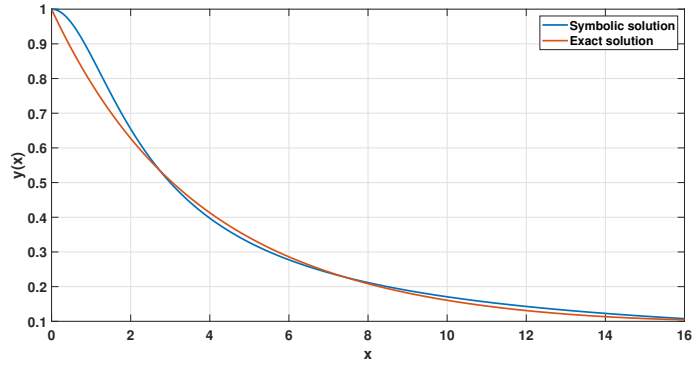


Figure 16: The exact and symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 5$.

4.6 Results of the proposed method for the standard Lane-Emden equation with $m = 5$

The solution obtained from our proposed method for the standard Lane-Emden equation with $m = 5$ is given by:

$$y(x) = exp((sin(((w[0] * exp((x * w[1])) - cos(w[2])) * x). \tag{4.11}$$

Table 10 lists the values of the coefficients in Eq. (4.11).

Table 10: Values of the coefficients in Eq. (4.11).

Coefficients	Values
w[0]	0.44821190834045410156
w[1]	0.01321445405483245850
w[2]	0.80176007747650146484

After simplifying Eq. (4.11), we obtain the following symbolic solution:

$$y(x) = e^{x \sin(0.448211908340 e^{0.013214454055x} - 0.695443028547)}$$

where the exact solution is as follows:

$$y(x) = (1 + \frac{x^2}{3})^{-\frac{1}{2}}.$$

The mean absolute error between the exact solution and the symbolic solution derived by our method for $x \in [0, 15]$ with step size $h = 0.01$ is MAE $= 0.015946963757$. Figure 16 illustrates the exact solution and the symbolic solution obtained using our proposed method for the standard Lane-Emden equation with $m = 5$.

Table 11: Comparison between the first exact root of different cases presented in [21] and the first root of the symbolic solution, along with the execution time (in minutes) for the standard Lane-Emden equation. These execution times were achieved on a system with an Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90GHz, 12.0 GB of installed RAM, and the PyTorch 2.0.1 CPU version.

m	Exact root [21]	Approximated root	Absolute error	Time (minutes)
0	2.4494897	2.4494889	0.00000080	103
1	3.1415926	3.1423459	0.00075330	133
2	4.3528746	4.3562164	0.00334183	99
3	6.8968486	6.8894858	0.00736280	113
4	14.9715463	14.9503489	0.02119740	320
5	-	-	-	590

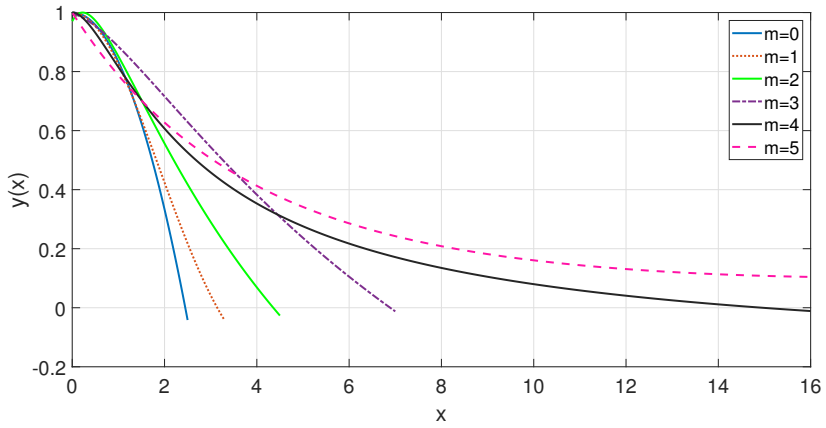


Figure 17: The symbolic solutions derived from our proposed method for the standard Lane-Emden equation with $m = 0, 1, 2, 3, 4, 5$.

4.6.1 A summary of the results achieved for the standard Lane-Emden equation using the proposed method

In this subsection, we present an overview of the results achieved using our proposed approach for the standard Lane-Emden equation. Table 11 compares the first roots of the standard Lane-Emden equation, contrasting our method with the exact numerical data provided by Horedt in [21]. Furthermore, this table shows the execution time for different cases. These execution times were achieved on a system with an Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90GHz, 12.0 GB of installed RAM, and the PyTorch 2.0.1 CPU version. Figure 17 displays the symbolic solutions obtained from our proposed method for the standard Lane-Emden equation with $m = 0, 1, 2, 3, 4, 5$. Note that for the case $m = 5$, there is no root.

5 Discussion

In our proposed scheme, several hyperparameters significantly impact the performance and accuracy of the solution. This section discusses the most crucial parameters in each phase.

As illustrated in Figure 3, Phase 1 employs a network consisting of three recurrent layers and a fully connected layer. We experimented with various layer configurations and weight spaces. Based on our trials, the optimal architecture for the standard Lane-Emden type equation solved in this paper is depicted in Figure 3.

In Phase 2, in the UCB formula shown in Eq. (3.9), the parameter η controls the balance between exploration and exploitation. A higher η encourages the algorithm to explore new actions and nods, while a lower η favors exploiting the current knowledge to select actions with the highest estimated value. Our experiments indicate that for $m = 0, 1, 2, 3, 4, 5$, the best results are achieved when η is set to 1, 1, 1, 1, 0.5, and 2.0, respectively. Extremely low η values lead to suboptimal convergence in the tree, whereas values above 10 cause significant variation, preventing accurate solutions. Effective solutions are obtained when η ranges between $[0.5, 2]$.

During the rollout stage of this study, we performed k random simulations from the corresponding node and averaged the values to determine the rollout stage value. The value of k is adaptive, decreasing as the number of inner

loop iterations increases. This adaptation is rational because, as the tree grows, less exploration is needed. To avoid infinite branching in the tree, a maximum number of branches is set for the rollout stage. If this maximum is reached without obtaining a valid mathematical expression, the branch value is set to a negative value.

Another important parameter is the iteration number of the inner loop, which determines the level of subtrees starting from S_0 and the order of the rollout stage execution. The iteration number influences the exploration rate, exploitation rate, and computational cost. More iterations allow the algorithm to explore different paths and improve its decision-making. However, increasing iterations also raises computational costs. Thus, a trade-off between iterations and practical system constraints is necessary. Our study found that the best balance between computational cost and accuracy was achieved with iteration numbers 20, 20, 20, 20, 20, and 10 for $m = 0, 1, 2, 3, 4, 5$ respectively.

The iteration number of the outer loop, denoted by D in Figure 8, also affects the algorithm. This parameter controls how many times the main stages of the MCTS are repeated. In each outer loop iteration, the best child of node S_0 is chosen as the new S_0, and the MCTS algorithm restarts with this new S_0. Our experiments show that the best results occur when the outer loop iteration number is set to 30.

In Phase 3, we fine-tuned the coefficients of the symbolic expression using the Adam optimizer in PyTorch with learning rates of 0.00001, 0.000001, 0.00000015, 0.000001, 0.001, and 0.0004 for $m = 0, 1, 2, 3, 4, 5$ respectively. Since this phase focuses on fine-tuning the coefficients, the learning rates are generally low.

From a computational perspective, Phases 1 and 2 are the most time-consuming. Phase 3, however, typically completes within minutes. Therefore, the execution times reported in Table 11 are mainly influenced by Phases 1 and 2. As evident from the table, execution time generally increases with the range $[a, b]$ considered for solving the corresponding case. We believe that computational costs could be reduced by running the algorithm on a GPU-based version of PyTorch or a system with higher hardware configuration.

6 Conclusion

This paper introduces an innovative approach that integrates recurrent neural networks, the Monte Carlo tree search algorithm, and physics-informed neural networks to find symbolic solutions for nonlinear ordinary differential equations. The Lane-Emden type equation, known for their complexity and significance in astrophysics, were used to evaluate the method's effectiveness. The results indicate that the proposed approach successfully provides symbolic solutions for these nonlinear equations. The advantages of this method include:

- By combining three types of algorithms RNNs, MCTS, and PINNs this approach uses the strengths of each, resulting in a more powerful solution than any single algorithm could achieve alone.
- The framework is flexible, allowing for the substitution of different algorithms in each phase, which can be tailored to specific problems.
- The method produces symbolic expressions, which are highly valuable for applications requiring such outputs.
- The Lane-Emden equation is particularly difficult to solve due to its nonlinearity and singularities, and it often lacks an exact solution. Given its importance in astrophysics, the ability of the proposed method to solve such equations demonstrates its potential for addressing a wide range of real-world problems.
- The method utilizes a context-free grammar with basis functions that can be easily adapted to suit the domain of a particular equation. For instance, orthogonal set functions might be more effective for certain problems.

Future research could extend this approach to a wider range of equations, such as integral equations, partial differential equations, and fractional order equations. Additionally, exploring other algorithms in Phase 2, like meta-heuristic algorithms and policy-based reinforcement learning algorithms, could further enhance the method. Established basis functions, such as Jacobi and Gegenbauer polynomials, could also be considered as options for the underlying grammar.

References

[1] G. Adomian, R. Rach, and N. T. Shawagfeh, *On the analytic solution of the lane-emden equation*, Foundations of Physics Letters **8** (1995), 161–181.

[2] C. C. Aggarwal, *Neural networks and deep learning: A textbook*, 2 ed., Springer, 2023.

[3] I. Ahmad, M. A. Z. Raja, M. Bilal, and F. Ashraf, *Neural network methods to solve the laneemden type equations arising in thermodynamic studies of the spherical gas cloud model*, Neural Computing & Applications **28** (2017), no. Suppl 1, 929–944.

[4] H. Aminikhah, *Solutions of the singular ivps of lane-emden type equations by combining laplace transformation and perturbation technique*, Nonlinear Engineering **7** (2018), no. 4, 273–278.

[5] O. Abu Arqub, A. El-Ajou, A. Sami Bataineh, and I. Hashim, *A representation of the exact solution of generalized lane-emden equations using a new analytical method*, Abstract and Applied Analysis **2013** (2013), Article ID 378593, 10 pages.

[6] S. Aydinlik, A. Kiris, and P. Roul, *An effective approach based on smooth composite chebyshev finite difference method and its applications to bratu-type and higher order laneemden problems*, Mathematics and Computers in Simulation **202** (2022), 193–205.

[7] H. Bassi, Y. Zhu, S. Liang, J. Yin, C. C. Reeves, V. Vlek, and C. Yang, *Learning nonlinear integral operators via recurrent neural networks and its application in solving integro-differential equations*, Machine Learning with Applications **15** (2024), 100524.

[8] H. Baty, *Modelling laneemden type equations using physics-informed neural networks*, Astronomy and Computing **44** (2023), 100734.

[9] V. Calvert, S. Mashayekhi, and M. Razzaghi, *Solution of laneemden type equations using rational bernoulli functions*, Mathematical Methods in the Applied Sciences **39** (2016), no. 5, 1268–1284.

[10] M. Chapwanya, R. Dozva, and G. Muchatibaya, *A nonstandard finite difference technique for singular lane-emden type equations*, Engineering Computations **36** (2019), 1566–1578.

[11] H. Dana Mazraeh and K. Parand, *Gepinn: An innovative hybrid method for a symbolic solution to the laneemden type equation based on grammatical evolution and physics-informed neural networks*, Astronomy and Computing **48** (2024), 100846.

[12] M. Dehghan and F. Shakeri, *Approximate solution of a differential equation arising in astrophysics using the variational iteration method*, New Astronomy **13** (2008), no. 1, 53–59.

[13] S. Deniz and N. Bildik, *A new analytical technique for solving Lane - Emden type equations arising in astrophysics*, Bulletin of the Belgian Mathematical Society - Simon Stevin **24** (2017), no. 2, 305 – 320.

[14] A. Karimi Dizicheh, S. Salahshour, A. Ahmadian, and D. Baleanu, *A novel algorithm based on the legendre wavelets spectral technique for solving the laneemden equations*, Applied Numerical Mathematics **153** (2020), 443–456.

[15] S.H. El-Essawy, M.I. Nouh, A.A. Soliman, H.I. Abdel Rahman, and G.A. Abd-Elmougod, *Monte carlo simulation of lane-emden type equations arising in astrophysics*, Astronomy and Computing **42** (2023), 100665.

[16] S. M. Ermakov and T. M. Tovstik, *Monte carlo method for solving ode systems*, Vestnik St. Petersburg University, Mathematics **52** (2019), 272–280.

[17] P. Hiram Guzzi, F. Petrizzelli, and T. Mazza, *Disease spreading modeling and analysis: a survey*, Briefings in Bioinformatics **23** (2022), no. 4, bbac230.

[18] S. Gmgm, *Taylor wavelet solution of linear and nonlinear lane-emden equations*, Applied Numerical Mathematics **158** (2020), 44–53.

[19] T. Hagge, P. Stinis, E. Yeung, and A. M. Tartakovsky, *Solving differential equations with unknown constitutive relations as recurrent neural networks*, 2017.

[20] G. Hojjati and K. Parand, *An efficient computational algorithm for solving the nonlinear lane-emden type equations*, International Journal of Mathematics and Computer Sciences **7** (2011), no. 4, 182–187.

[21] G. P. Horedt, *applications in astrophysics and related fields*, Kluwer Academic Publishers, Dordrecht, 2004.

[22] Z. Huang, D. Z. Huang, T. Xiao, D. Ma, Z. Ming, H. Shi, and Y. Wen, *Improving monte carlo tree search for symbolic regression*, The Thirty-ninth Annual Conference on Neural Information Processing Systems (2025).

[23] M. Izadi, *A discontinuous finite element approximation to singular lane-emden type equations*, Applied Mathematics and Computation **401** (2021), 126115.

[24] A. Khaliq J. Long and Kh. M. Furati, *Parameter identification for pdes using sparse interior data and a recurrent neural network*, Scientific Reports **15** (2025).

[25] R. Kumar, S. Aeri, and D. Baleanu, *Artificial neural networks for the wavelet analysis of lane-emden equations: exploration of astrophysical enigma*, International Journal of Modelling and Simulation (2024), 1–12.

[26] P. Linz, *An introduction to formal languages and automata*, 6th ed., Jones & Bartlett Learning, 2019.

[27] S. Mall and S. Chakraverty, *Chebyshev neural network based model for solving laneemden type equations*, Applied Mathematics and Computation **247** (2014), 100–114.

[28] J. Madziuk, K. Godlewski, B. Sawicki, and J. Madziuk, *Monte carlo tree search: a review of recent modifications and applications*, Artificial Intelligence Review **56** (2023), 2497–2562.

[29] M. Y. Niu, L. Horesh, and I. Chuang, *Recurrent neural networks in the eye of differential equations*, 2019.

[30] K. Parand, S. Abbasbandy, S. Kazem, and A. R. Rezaei, *An improved numerical method for a class of astrophysics problems based on radial basis functions*, Physica Scripta **83** (2011), no. 1, 015011.

[31] K. Parand, A. A. Aghaei, S. Kiani, T. Ilkhas Zadeh, and Z. Khosravi, *A neural network approach for solving nonlinear differential equations of laneemden type*, Engineering with Computers (2023).

[32] K. Parand, M. Dehghan, A.R. Rezaei, and S.M. Ghaderi, *An approximation algorithm for the solution of the nonlinear laneemden type equations arising in astrophysics using hermite functions collocation method*, Computer Physics Communications **181** (2010), no. 6, 1096–1108.

[33] K. Parand and S. Hashemi, *Rbf-dq method for solving non-linear differential equations of lane-emden type*, Ain Shams Engineering Journal **9** (2018), no. 4, 615–629.

[34] K. Parand and M. Hemami, *Numerical study of astrophysics equations by meshless collocation method based on compactly supported radial basis function*, International Journal of Applied and Computational Mathematics **3** (2015), 1053–1075.

[35] K. Parand, M. Shahini, and M. Dehghan, *Rational legendre pseudospectral approach for solving nonlinear differential equations of laneemden type*, Journal of Computational Physics **228** (2009), no. 23, 8830–8840.

[36] G. Penent and N. Privault, *Numerical evaluation of ode solutions by monte carlo enumeration of butcher series*, BIT Numerical Mathematics **62** (2022), 1921–1944.

[37] M. Raissi, P. Perdikaris, and G.E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational Physics **378** (2019), 686–707.

[38] R. V. Ramnath, *On a class of nonlinear differential equations of astrophysics*, Journal of Mathematical Analysis and Applications **35** (1971), no. 1, 27–47.

[39] Z. Sabir, H. Abdul Wahab, M. Umar, M. G. Sakar, and M. A. Zahoor Raja, *Novel design of morlet wavelet neural network for solving second order laneemden equation*, Mathematics and Computers in Simulation **172** (2020), 1–14.

[40] E. Saveliev, S. Holt, N. Seedat, D. L. Bentley, J. Weatherall, and M. van der Schaar, *Influence-guided symbolic regression: Scientific discovery via LLM-driven equation search with granular feedback*, Forty-third International Conference on Machine Learning (2026).

[41] M. Shirazian, *A new acceleration of variational iteration method for initial value problems*, Mathematics and Computers in Simulation **214** (2023), 246–259.

[42] R. Singh, N. Das, and J. Kumar, *The optimal modified variational iteration method for the lane-emden equations with neumann and robin boundary conditions*, Eur. Phys. J. Plus **132** (2017), 251.

[43] F. Sun, Y. Liu, J. Wang, and H. Sun, *Symbolic physics learner: Discovering governing equations via monte carlo tree search*, arXiv:2205.13134 (2022).

[44] J. Sunday, A. Shokri, R. O. Akinola, K. Victor Joshua, and K. Nonlaopon, *A convergence-preserving non-standard finite difference scheme for the solutions of singular lane-emden equations*, Results in Physics **42** (2022), 106031.

[45] R. S. Sutton and A. G. Barto, *Reinforcement learning an introduction*, 2nd ed., MIT Press, 2020.

[46] R. J. Swift and S. A. Wirkus, *A course in ordinary differential equations*, Taylor & Francis, 2006.

[47] A. Verma and M. Kumar, *Numerical solution of laneemden type equations using multilayer perceptron neural network method*, International Journal of Applied and Computational Mathematics **5** (2019), 141.

[48] A. M. Wazwaz, *Adomian decomposition method for a reliable treatment of the emdenfowler equation*, Applied Mathematics and Computation **161** (2005), no. 2, 543–560.

[49] A. M. Wazwaz, *Solving two emdenfowler type equations of third order by the variational iteration method*, Applied Mathematics and Information Sciences **9** (2015), 2429–2436.

[50] X. Yang, Y. Liu, and G. Park, *Parameter estimation of uncertain differential equation with application to financial market*, Chaos, Solitons & Fractals **139** (2020), 110026.

[51] A. Yildirim, *A dna-inspired physics-informed machine learning framework for singular laneemden type equations*, Astronomy and Computing **58** (2027), 101183.

[52] Z. Yu, J. Ding, Y. Li, and D. Jin, *Symbolic regression via mdlformer-guided search: from minimizing prediction error to minimizing description length*, 13th International Conference on Learning Representations (ICLR) (2025), 65306–65333.

[53] . Yzba and M. Sezer, *An improved bessel collocation method with a residual error function to solve a class of laneemden differential equations*, Mathematical and Computer Modelling **57** (2013), no. 5, 1298–1311.

[54] A. Yldrm and T. zi, *Solutions of singular ivps of laneemden type by homotopy perturbation method*, Physics Letters A **369** (2007), no. 1, 70–76.