Research Article ISSN Online: 2783-4859 Computational Mathematics and Computer Modeling with Applications

Received 3 March 2025
Accepted 3 April 2025

DOI: 10.48308/CMCMA.4.1.1
AMS Subject Classification: 68T07; 34A34

A new rational Legendre neural network for solving the Blasius equation

Mohammad Mohammadi[a], Yeganeh Ghaderi[a], Hassan Dana Mazraeh[a] and Kourosh Parand[a,b]

In this paper, we present a novel artificial neural network framework for solving the Blasius equation, a nonlinear ordinary differential equation defined on a semi-infinite domain. Our experiments revealed that traditional activation functions, such as Tanh and ReLU, did not produce satisfactory results. To address this, we employed custom activation functions based on Rational Legendre polynomials, which demonstrated superior performance in approximating the solution. The results highlight the effectiveness and potential of this approach, offering a valuable contribution to the scientific community for addressing similar nonlinear differential equations.

Keywords: The Blasius equation; Rational Legendre polynomials; Neural network; Boundary-layer flow; Semi-infinite domains; Nonlinear differential equations.

1. Introduction

The Blasius equation is a fundamental equation in fluid mechanics that describes the laminar, two-dimensional flow of an incompressible viscous fluid over a flat surface. While much of the boundary layer may consist of turbulent regions with eddies and fluctuations in flow parameters, the Blasius equation specifically arises from boundary layer theory, developed in the early 20th century to analyze fluid behavior near solid boundaries.

The FalknerSkan equation, a nonlinear differential equation in fluid dynamics introduced by Falkner and Skan, represents a broader class of equations derived from the NavierStokes equations. These equations are used to analyze boundary-layer flows under specific conditions, typically for two-dimensional, steady flows over surfaces with defined shapes[32].

The Blasius equation is a special case of the FalknerSkan equations and remains one of the key equations in fluid mechanics for describing boundary-layer flows near solid surfaces.

1.1. Form of the Blasius equation

By applying scaling arguments to the Navier–Stokes equations, many terms can be shown to be negligible under specific conditions, leading to the simplified boundary layer equations. In fluid dynamics, the boundary layer forming near a flat plate with fluid moving over it can be described using boundary layer conditions and corresponding initial conditions [19, 22, 39].

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{1}$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \frac{1}{\rho}\frac{\partial p}{\partial x} + v\frac{\partial^2 u}{\partial y^2}. \tag{2}$$

The flat plate problem is subject to the following conditions:

[a] *Department of Computer and Data Sciences, Faculty of Mathematical Sciences, Shahid Beheshti University, Tehran, Iran.*
[b] *Department of Cognitive Modeling, Institute for Cognitive and Brain Sciences, Shahid Beheshti University, Tehran, Iran.*
* *Correspondence to: K. Parand. Email: k_parand@sbu.ac.ir*

$$\begin{aligned} u(0,y) &= u_\infty, \\ u(x,\infty) &= u_\infty, \\ u(x,0) &= v(x,0) = 0. \end{aligned} \tag{3}$$

Introducing the dimensionless similarity variable η and performing further manipulations [32] yields the following form:

$$f'''(\eta) + \frac{1}{2} f(\eta) f''(\eta) + \beta \left[1 - f'^2(\eta)\right] = 0, \quad \eta \in (0, \infty) \tag{4}$$

The conditions are as follows:

$$f(0) = f'(0) = 0, \quad f'(\infty) = 1. \tag{5}$$

For a simplified form, the Blasius equation can also be expressed as:

$$f'''(\eta) + \frac{1}{2} f(\eta) f''(\eta) = 0, \quad \eta \in (0, \infty). \tag{6}$$

With conditions identical to those stated in Eq. (5).

In Eq.(4), the nonlinear term reflects the effects of friction in the boundary layer and is applicable under certain turbulent conditions. However, in this paper, we focus on the standard form of the Blasius equation (Eq.(6)), which is valid for steady flows in smooth boundary layers.

2. Related Works

Various approaches have been proposed to solve problems on infinite domains[3, 6, 43]. Among these, spectral methods are particularly well-regarded [1, 20, 36, 44]. Boyd [9] investigated some of these spectral methods on semi-infinite domains, Guo [15, 16] employed Jacobi and Gegenbauer collocation, and Shen [23] studied Hermite spectral methods.

On unbounded intervals, numerous numerical methods have been used [18, 22, 24, 40, 46]. In recent years, artificial neural networks have gained traction for solving differential equations owing to their convergence properties and fast runtime. By the universal approximation theorem, neural networks can approximate any continuous function to an arbitrary level of precision [32]. Since the late 1990s, machine learning techniques have been employed to solve differential equations [13, 21, 28], and hybrid methods combining machine learning with optimization have been applied to higher-order ODEs [29, 35, 47]. Researchers have also adopted neural networks for specific differential equation problems [10, 28, 34, 41]. In studies [25, 26], researchers integrated context-free grammars with deep reinforcement learning networks and genetic programming to analytically solve the Blasius equation and other nonlinear equations. Additionally, in [27, 11, 8, 12], these types of equations were applied to various artificial intelligence algorithms, including neural networks and meta-heuristic algorithms.

The authors in [30, 31, 42, 7] have pioneered the use of neural networks and deep neural networks to solve differential equations on unbounded domains. Gao [14] utilized convolutional neural networks for parameterized steady-state PDEs on irregular domains.

Least-squares support vector regression (LS-SVR) is another powerful machine learning technique that has been successfully used to tackle nonlinear differential equations. This method has been extensively applied to address engineering problems and other complex challenges in [5, 33, 37, 38]. In this paper, we employ fractional Legendre polynomials as activation functions in neural networks to approximate the solution of the Blasius equation.

3. Backgrounds

Legendre polynomials constitute a classical family of orthogonal polynomials that arise as solutions to Legendres differential equation. Their orthogonality and recurrence properties make them indispensable in various fields, including potential theory, approximation theory, and numerical analysis [2, 45].

The recursive formula of the Legendre polynomials is as follows:

$$P_{n+1}(x) = \frac{2n+1}{n+1} x P_n(x) - \frac{n}{n+1} P_{n-1}(x), \quad n > 1, \tag{7}$$

The initial conditions are as follows:

$$P_0(x) = 1 \quad , \quad P_1(x) = x. \tag{8}$$

Table 1 lists the expressions for the first nine Legendre polynomials, which form the foundation of many spectral and numerical methods.

Table 1. Legendre Polynomials.

n	$P_n(x)$
0	1
1	x
2	$\frac{1}{2}(3x^2-1)$
3	$\frac{1}{2}(5x^3-3x)$
4	$\frac{1}{8}(35x^4-30x^2+3)$
5	$\frac{1}{8}(63x^5-70x^3+15x)$
6	$\frac{1}{16}(231x^6-315x^4+105x^2-5)$
7	$\frac{1}{16}(429x^7-693x^5+315x^3-35x)$
8	$\frac{1}{16}(6435x^8-12012x^6+6930x^4-1260x^2+35)$

For many applications, especially those involving differential equations on unbounded domains, it is advantageous to map the infinite domain to a finite interval. A common mapping is to substitute x with $\frac{x-L}{x+L}$, where $L \in \mathbb{R}$. This transformation leads to the construction of Rational Legendre polynomials, which inherit many of the favorable properties of the classical Legendre polynomials while being better suited for approximating functions on semi-infinite domains [17]. Table 2 presents the first nine Rational Legendre polynomials derived from this transformation.

Table 2. Rational Legendre Polynomials.

n	$R_n(x) = P_n\left(\frac{x-L}{x+L}\right)$
0	1
1	$\frac{x-L}{x+L}$
2	$\frac{1}{2}\left(3\left(\frac{x-L}{x+L}\right)^2-1\right)$
3	$\frac{1}{2}\left(5\left(\frac{x-L}{x+L}\right)^3-3\left(\frac{x-L}{x+L}\right)\right)$
4	$\frac{1}{8}\left(35\left(\frac{x-L}{x+L}\right)^4-30\left(\frac{x-L}{x+L}\right)^2+3\right)$
5	$\frac{1}{8}\left(63\left(\frac{x-L}{x+L}\right)^5-70\left(\frac{x-L}{x+L}\right)^3+15\left(\frac{x-L}{x+L}\right)\right)$
6	$\frac{1}{16}\left(231\left(\frac{x-L}{x+L}\right)^6-315\left(\frac{x-L}{x+L}\right)^4+105\left(\frac{x-L}{x+L}\right)^2-5\right)$
7	$\frac{1}{16}\left(429\left(\frac{x-L}{x+L}\right)^7-693\left(\frac{x-L}{x+L}\right)^5+315\left(\frac{x-L}{x+L}\right)^3-35\left(\frac{x-L}{x+L}\right)\right)$
8	$\frac{1}{16}\left(6435\left(\frac{x-L}{x+L}\right)^8-12012\left(\frac{x-L}{x+L}\right)^6+6930\left(\frac{x-L}{x+L}\right)^4-1260\left(\frac{x-L}{x+L}\right)^2+35\right)$

4. Methodology

In this section, we describe the core architecture of our proposed neural network approach and outline the training procedure used to solve the target boundary-value problem. The implementation is based on the *PyTorch* framework, and we experimented with multiple optimization strategies (e.g., Adam, SGD, etc.) to evaluate both convergence speed and solution accuracy. Empirical tests revealed that *Adam* provides a balanced trade-off between rapid convergence and model performance, making it our primary choice.Finally, the codes modular design makes it easy to add custom loss functions and specific boundary conditions.

4.1. Rational Legendre Neural Network

The architecture of our network consists of 5 layers, described as follows:

- One input layer with 1 neuron,
- Two hidden layers (h_1 and h_2) with 9 neurons each, using the Tanh activation function,
- One hidden layer (h_3) with 9 neurons, employing a custom Rational Legendre activation function,
- One output layer with 1 neuron.

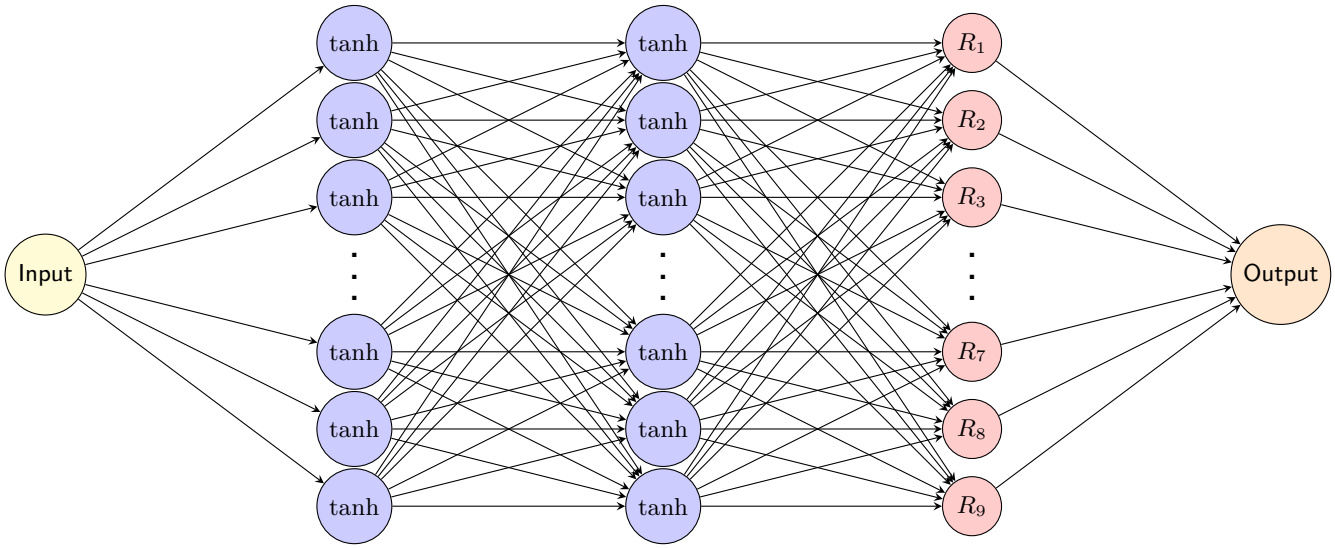

Figure 1. The structure of feed-forward neural network.

We chose the Tanh activation function for all neurons in the first two hidden layers for the following reasons:

1. **Nonlinearity:** Nonlinear differential equations inherently require nonlinear modeling. The Tanh function, being a nonlinear activation function, enables the neural network to effectively learn complex and nonlinear patterns.
2. **Domain Constraint:** The Tanh function outputs values constrained to the range [-1 , 1] , which helps stabilize the network during training.
3. **Differentiability:** The Tanh function has continuous and easily computable derivatives, a crucial property for optimization algorithms like backpropagation.

Rational Legendre polynomials are employed as activation functions in the third hidden layer to leverage their ability to approximate functions over semi-infinite domains. Each polynomial is assigned to a single neuron. Specifically, the first neuron uses the first Rational Legendre polynomial, the second neuron uses the second Rational Legendre polynomial, and so on, up to the ninth neuron, which uses the ninth Rational Legendre polynomial.

After defining an appropriate loss function and employing the Adam optimizer, we trained the network for a specified number of epochs (e.g., 20,000). This framework facilitates a rapid and accurate approximation of the Blasius solution.

4.2. Parameter Tuning and Implementation Details

In practice, we followed these steps to obtain the solution:

1. **Data Preparation:** We generated (or used) a small set of reference data points for the Blasius solution, which were then loaded in batches during training. These data points primarily guided the network to match known solution values at selected locations.
2. **Architecture Setup:** As outlined earlier, the network consists of four layers:
   - One input layer (1 neuron),
   - Two hidden layers (h_1 and h_2), each with 9 neurons and Tanh activation,
   - One hidden layer (h_3) with 9 neurons, where each neuron employs a different Rational Legendre polynomial as its activation function,
   - One output layer (1 neuron), incorporating an additional constant shift (denoted as $+1.0$ in Figure 1). Without this constant shift, the model tends to converge to zero to minimize the residual and initial losses. To prevent this undesired behavior and ensure a more meaningful solution, we introduce a $+1$ constant shift to the output of the network.
3. **Differential Equation Residual:** In addition to fitting the known data points, we minimized the residual of the Blasius differential equation:
$$f'''(\eta) + \frac{1}{2} f(\eta) f''(\eta) = 0,$$
subject to the conditions $f(0) = 0$, $f'(0) = 0$, and $f'(\infty) \approx 0.99$. This ensures that the network's output aligns with both the available data and the governing physical model.
4. **Parameter Tuning:**
   - *Number of Layers and Neurons:* We experimented with various architectures and found that three hidden layers with 9 neurons each provided an optimal balance between accuracy and computational efficiency.

- *The Role of L in Rational Legendre Polynomials:* The parameter L in the transformation $\left(\frac{x-L}{x+L}\right)$ dictates how the semi-infinite domain is mapped to a finite interval and influences the polynomial's behavior. After testing different values ($L = 1$ to 4), we found that $L = 4$ resulted in the most accurate approximation of the Blasius solution. Larger values of L generally improved the solution's accuracy by extending the range where the polynomials effectively capture the function behavior, enhancing alignment with conditions and training data.

4.3. Loss Function and Training

In order to solve the Blasius equation, we consider two main components in our total loss:

1. **Residual Loss** (Loss_{res}): Ensures that the network output $\hat{f}(\eta)$ satisfies the differential equation
$$f'''(\eta) + \frac{1}{2} f(\eta) f''(\eta) = 0.$$
In practice, we sample points $\{\eta_i\}$ and enforce the ODE residual as follows:
$$\text{Loss}_{\text{res}} = \sum_{i=1}^{n} \left(\hat{f}'''(\eta_i) + \frac{1}{2} \hat{f}(\eta_i) \hat{f}''(\eta_i) \right)^2$$
.
2. **Conditions Losses** (Loss_{Cs}): We impose the conditions $f(0) = 0$, $f'(0) = 0$, and $f'(\infty) = 1$. These appear as mean-squared constraints, such as:
$$\text{Loss}_{\text{Cs}} = \big(f(0)\big)^2 + \big(f'(0)\big)^2 + \big(f'(\infty) - 1\big)^2.$$

In this study, to handle the infinity condition in $\big(f'(\infty) - 1\big)^2$, we have used a very large number α in the PyTorch package. Therefore, instead of $\big(f'(\infty) - 1\big)^2$, we have used $\big(f'(\alpha) - 1\big)^2$ in the construction of the loss function. Hence, the total loss $\text{Loss}_{\text{total}}$ can be written as:
$$\text{Loss}_{\text{total}} = \lambda_1 \text{Loss}_{\text{res}} + \lambda_2 \text{Loss}_{\text{Cs}}$$
where λ_1 and λ_2 are weighting coefficients that balance the ODE residual and conditions loss. In this stydy, parameter λ_1 and λ_2 are introduced to balance these different loss terms, which may have varying magnitudes or importance depending on the problem. Without proper weighting, one term could dominate the optimization process, leading to poor convergence or solutions that violate certain physical constraints.

5. Results

In this section, numerical results are presented for solving the Blasius equation using the method described in Section 4.1, which has been implemented with the PyTorch library in Python. We compare the values of $f(\eta)$, $f'(\eta)$, and $f''(\eta)$ for various η with those obtained by other methods.

Table 3. Comparison of $f''(0)$ between the present method and various methods

Number of layers	Present method	Pakniyat [32]	Parand [39]	Howarth [19]	Rafael [40]	LCDNN [4]	LDNN [4]
$[1, 9, 9, 9, 1]$	0.332057102	0.332057335	0.33206	0.33206	0.33206	0.3320590	0.3321312

Table 4. Comparison of $f(\eta)$ values between the present method and various methods

η	Present	Pakniyat [32]	Parand [39]	Howarth [19]	Rafael [40]
1.0	0.17191576	0.16541105	0.1655731	0.16557	0.16557
2.0	0.65599083	0.64967773	0.6500351	0.65002	0.65003
3.0	1.39814352	1.39647618	1.3968254	1.39681	1.39682
4.0	2.29890894	2.30574561	2.3057619	2.30575	2.30576
5.0	3.26717567	3.28384346	3.2832913	3.28327	3.28330
6.0	4.25438499	4.28098773	4.2796473	4.27962	4.27965

Table 5. Comparison of $f'(\eta)$ values between the present method and various methods

η	Present	Pakniyat [32]	Parand [39]	Howarth [19]	Rafael [40]
1.0	0.33214044	0.329566712	0.3297956	0.32979	0.32978
2.0	0.62707352	0.629571785	0.6297737	0.62977	0.62977
3.0	0.83942985	0.846109525	0.8460586	0.84605	0.84605
4.0	0.94632434	0.956103098	0.9555253	0.95552	0.95552
5.0	0.98240298	0.992407319	0.9915583	0.99155	0.99155
6.0	0.98969644	0.999894820	0.9989882	0.99898	0.99898

Table 6. Comparison of $f''(\eta)$ values between the present method and various methods

η	Present	Pakniyat [32]	Parand [39]	Howarth [19]	Rafael [40]
0.0	0.33176034	0.332057102	0.332057335	0.33206	0.33206
1.0	0.31695914	0.322884293	0.3230174	0.32301	0.32301
2.0	0.26285910	0.266958108	0.2667514	0.26675	0.26675
3.0	0.15745177	0.161256399	0.1613615	0.16136	0.16136
4.0	0.06335017	0.063990964	0.0642426	0.06424	0.06423
5.0	0.01621076	0.016547122	0.0159142	0.01591	0.01591
6.0	0.00190105	0.003129914	0.0024067	0.00240	0.00240

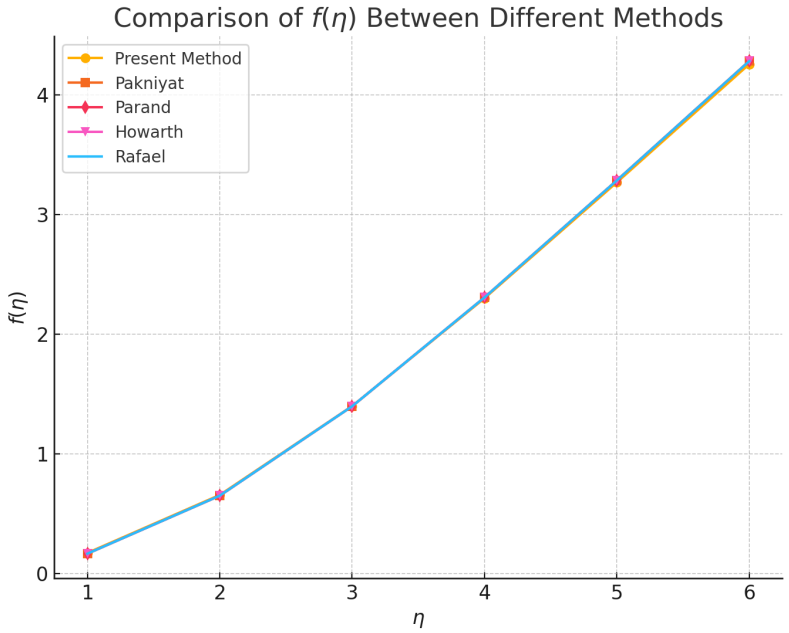

Figure 2. A comparison of the values of $f(\eta)$ obtained using the present method with those reported by other researchers, including Pakniyat [32], Parand [39], Howarth [19], and Rafael [40]..

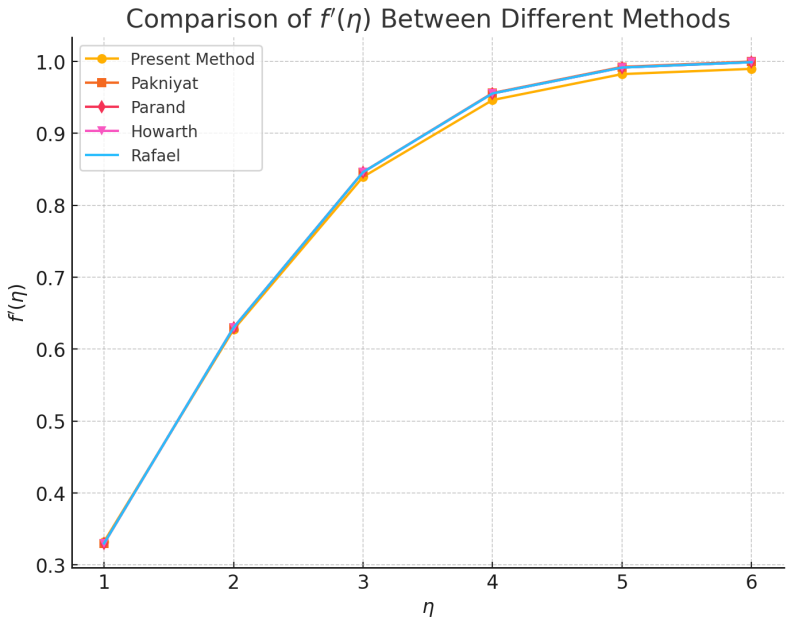

Figure 3. A comparison of the values of $f'(\eta)$ obtained using the present method with those reported by other researchers, including Pakniyat [32], Parand [39], Howarth [19], and Rafael [40]..

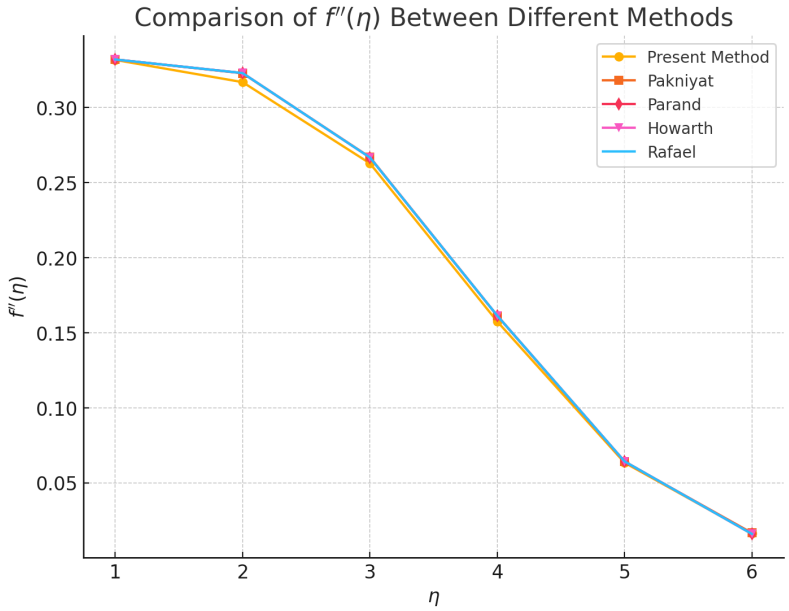

Figure 4. A comparison of the values of $f''(\eta)$ obtained using the present method with those reported by other researchers, including Pakniyat [32], Parand [39], Howarth [19], and Rafael [40]..

6. Conclusions

In this paper, we proposed a novel approach for solving the Blasius equation, utilizing Rational Legendre polynomials as activation functions in a neural network framework. The primary contribution of this work lies in the ability to handle semi-infinite domains effectively while achieving high accuracy and rapid convergence. By integrating Rational Legendre polynomials into the network architecture, we demonstrated a significant improvement over traditional methods.

The numerical results presented in this paper confirm the effectiveness of the proposed method. Our framework not only satisfies the conditions of the Blasius equation but also yields results that are highly comparable to those obtained by other advanced methods. Additionally, incorporating Tanh activation functions in the initial layers ensures the stability and nonlinearity necessary for solving complex differential equations.

PINNs have shown considerable potential for tackling both forward and inverse problems related to differential equations, yet they face notable limitations. A key difficulty lies in the extensive tuning of hyperparameters, especially in intricate, high-dimensional scenarios, where suboptimal settings can make training computationally costly. Additionally, their effectiveness in inverse problems heavily depends on access to high-quality training data, and performance may suffer when confronted with noisy or partial datasets. Looking ahead, research should prioritize enhancing training efficiency by adopting advanced optimization methods, such as dynamic learning rate adjustments or meta-learning techniques. Creating hybrid approaches that combine PINNs with conventional numerical solvers or multi-fidelity frameworks could improve their precision and adaptability. Moreover, investigating more resilient network designs could address these limitations.

References

1. M. Abbazadeh, M. Bayat, M. Dehghan, and M. I. Azis. Investigation of generalized couette hydromagnetic flow of two-step exothermic chemical reaction in a channel via the direct meshless local PetrovGalerkin method. *Engineering Analysis with Boundary Elements*, 125:178–189, 2021.
2. M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. Dover Publications, 1964.
3. A. A. Aghaei and K. Parand. Hyperparameter optimization of orthogonal functions in the numerical solution of differential equations. *Mathematical Methods in the Applied Sciences*, 48(1):605–625, 2025.
4. A. A. Aghaei, K. Parand, A. Nikkhah, and S. Jaberi. Solving Falkner-Skan type equations via Legendre and Chebyshev neural blocks. *arXiv*, page arXiv:2308.03337, 2023.
5. P. Ahadian and K. Parand. Support vector regression for the temperature-stimulated drug release. *Chaos, Solitons & Fractals*, 165:112871, 2022.
6. M. A. Arefin, M. A. Nishu, M. N. Dhali, and M. H. Uddin. Analysis of reliable solutions to the boundary value problems by using shooting method. *Mathematical Problems in Engineering*, 2022:1–9, 2022.
7. F. Baharifard and K. Parand. Numerical solution of differential equations of Lane-Emden type by Gegenbauer and rational Gegenbauer collocation methods. *Computational Mathematics and Computer Modeling with Applications (CMCMA)*, 1(1):69–85, 2022.
8. M. Bolhassani, H. D. Mazraeh, and K. Parand. A new method based on least-squares support vector regression for solving optimal control problems. *Kybernetika*, 60:513–534, 2024.

9. J. P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Publications, Mineola, New York, 2000.
10. F. Calabrò, G. Fabiani, and C. Sieos. Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients. *Computer Methods in Applied Mechanics and Engineering*, 387:114188, 2021.
11. H. Dana Mazraeh, K. Parand, H. Farahani, and S. Kheradpisheh. An improved imperialist competitive algorithm for solving an inverse form of the Huxley equation. *Iranian Journal of Numerical Analysis and Optimization*, 14(Issue 3):681–707, 2024.
12. A. N. Firoozsalari, H. D. Mazraeh, A. A. Aghaei, and K. Parand. deepFDEnet: A Novel Neural Network Architecture for Solving Fractional Differential Equations, 2023.
13. M. Frank, D. Drikakis, and V. Charissis. Machine-learning methods for computational science and engineering. *Computation*, 8(1):1–35, 2020.
14. H. Gao, L. Sun, and J. X. Wang. Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. *Journal of Computational Physics*, 428:110079, 2021.
15. B. Guo. Gegenbauer approximation and its applications to differential equations with rough asymptotic behaviors at infinity. *Applied Numerical Mathematics*, 38:403–425, 2001.
16. B. Y. Guo. Jacobi approximations in certain Hilbert spaces and their applications to singular differential equations. *Journal of Mathematical Analysis and Applications*, 243:373–408, 2000.
17. B. Y. Guo and Z. Wang. A spectral method for differential equations on unbounded domains using rational Legendre functions. *Journal of Computational Physics*, 173(2):387–404, 2002.
18. Z. Hajimohammadi, F. Baharifard, and K. Parand. A new numerical learning approach to solve general Falkner Skan model. *Engineering with Computers*, 38:121–137, 2022.
19. L. Howarth. On the calculation of steady flow in the boundary layer near the surface of a cylinder in a stream. *Aero Res. Counc. Lond. R & M*, 164:16–32, 1935.
20. A. Jalilian, A. A. bdi, and G. Hojjat. Variable stepsize SDIMSIMs for ordinary differential equations. *Applied Numerical Mathematics*, 168:115–126, 2021.
21. A. Kratsios. The universal approximation property: characterization, construction, representation, and existence. *Annals of Mathematics and Artificial Intelligence*, 89:435–469, 2021.
22. S. J. Liao. An explicit, totally analytic approximate solution for Blasius viscous flow problems. *International Journal of Non-Linear Mechanics*, 34(4):759–778, 1999.
23. Z. Mao and J. Shen. Hermite spectral methods for fractional PDEs in unbounded domains. *SIAM Journal on Scientific Computing*, 39(5):A1928–A1950, 2017.
24. H. D. Mazraeh, M. Kalantari, S. H. Tabasi, A. A. Aghaei, Z. Kalantari, and F. Fahim. Solving Fredholm integral equations of the second kind using an improved cuckoo optimization algorithm. *Global Analysis and Discrete Mathematics*, 7(1):33–52, 2023.
25. H. D. 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*, 48:100846, 2024.
26. H. D. Mazraeh and K. Parand. An innovative combination of deep Q-networks and context-free grammars for symbolic solutions to differential equations. *Engineering Applications of Artificial Intelligence*, 142:109733, 2025.
27. H. D. Mazraeh, K. Parand, M. Hosseinzadeh, J. Lansky, and V. Nulíek. An improved water strider algorithm for solving the inverse Burgers Huxley equation. *Scientific Reports*, 2024.
28. C. Michoski, M. Milosavljevi, T. Oliver, and D. R. Hatch. Solving differential equations using deep neural networks. *Neurocomputing*, 399:193–212, 2020.
29. M. B. Milovanovic, D. S. Anti, S. S. Nikoli, S. L. Peric, M. T. Milojkovi, and M. D. Spasic. Neural network based on orthogonal polynomials applied in magnetic levitation system control. *Elektronika ir elektrotechnika*, 23(3):24–29, 2017.
30. M. M. Moayeri, J. A. Rad, and K. Parand. Dynamical behavior of reaction-diffusion neural networks and their synchronization arising in modeling epileptic seizure: A numerical simulation study. *Computers & Mathematics with Applications*, 80(8):1887–1927, 2020.
31. M. Omidi, B. Arab, A. H. Rasanan, J. A. Rad, and K. Parand. Learning nonlinear dynamics with behavior ordinary/partial/system of the differential equations: looking through the lens of orthogonal neural networks. *Engineering with Computers*, pages 1–20, 2021.
32. A. Pakniyat and K. Parand. Hermite neural network for solving the Blasius equation. *Computational Mathematics and Computer Modeling with Applications (CMCMA)*, 1:86–94, 2022.
33. A. Pakniyat, K. Parand, and M. Jani. Least squares support vector regression for differential equations on unbounded domains. *Chaos, Solitons & Fractals*, 151:111232, 2021.
34. N. Panda, M. G. Fernndez-Godino, H. C. Godinez, and C. Dawson. A data-driven non-linear assimilation framework with neural networks. *Computational Geosciences*, 25(1):233–242, 2021.
35. K. Parand, A. A. Aghaei, M. Jani, and A. Ghodsi. A new approach to the numerical solution of Fredholm integral equations using least squares-support vector regression. *Mathematics and Computers in Simulation*, 180:114–128, 2021.
36. K. Parand, A. A. Aghaei, M. Jani, and A. Ghodsi. Parallel LS-SVM for the numerical simulation of fractional Volterras population model. *Alexandria Engineering Journal*, 60(6):5637–5647, 2021.
37. K. Parand, G. S. Ghaemi Javid, and M. Jani. A machine learning approach for solving inverse Stefan problem. *International Journal of Nonlinear Analysis and Applications*, 13(2):2233–2246, 2022.
38. K. Parand, M. Razzaghi, R. Sahleh, and M. Jani. Least squares support vector regression for solving Volterra integral equations. *Engineering with Computers*, 38:789796, 2022.
39. K. Parand and A. Taghavi. Rational scaled generalized laguerre function collocation method for solving the Blasius equation. *Journal of Computational and Applied Mathematics*, 233(4):980–989, 2009.
40. C. Rafael. Numerical solutions of the classical blasius flat-plate problem. *Applied Mathematics and Computation*, 170:706–710, 2005.
41. M. Raissi and G. E. Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.
42. A. H. Rasanan, D. Rahmati, S. Gorgin, and K. Parand. A single layer fractional orthogonal neural network for solving various types of LaneEmden equation. *New Astronomy*, 75:101307, 2020.
43. F. Salehi, S. H. Shahraki, M. K. Fallah, and M. Hemamia. Numerical investigation of differential biological models via Gaussian RBF collocation method with genetic strategy. *Computational Mathematics and Computer Modeling with Applications*, 1(2):46–64, 2022.

44. J. Shen, T. Tang, and L.-L. Wang. *Spectral Methods: Algorithms, Analysis and Applications*. Springer, New York, 2011.
45. G. Szegö. *Orthogonal Polynomials*. American Mathematical Society, 4th edition, 1975.
46. M. H. Uddin, M. A. Arefin, M. A. Akbar, and M. Inc. New explicit solutions to the fractional-order Burgers equation. *Mathematical Problems in Engineering*, 2021:1–11, 2021.
47. N. Vukovic, M. Petrovic, and Z. Miljkovic. A comprehensive experimental evaluation of orthogonal polynomial expanded random vector functional link neural networks for regression. *Applied Soft Computing*, 70:1083–1096, 2018.

