AIMS Mathematics, 10(9): 21853–21872.
DOI: 10.3934/math.2025972
Received: 20 July 2025
Revised: 09 September 2025
Accepted: 12 September 2025
Published: 19 September 2025

https://www.aimspress.com/journal/Math

Research article

Physics-informed neural networks framework for solving the highly nonlinear Bratu equation arising in combustion theory

Saurabh Tomar[1,2] **and Higinio Ramos**[3,*]

[1] C3I Center, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur, UP 208016, India

[2] Department of Applied Sciences, Rajiv Gandhi National Aviation University, Amethi, UP 229302, India

[3] Scientific Computing Group, Universidad de Salamanca, Plaza de la Merced, Salamanca 37008, Spain

* **Correspondence:** Email: higra@usal.es.

Abstract: This research introduced a physics informed neural network (PINN) framework designed to effectively solve the highly nonlinear Bratu equation, which arises in various physical contexts such as chemical reaction theory and combustion processes. PINNs provide a mesh-free solution by embedding physical laws directly into the loss function of the neural network and utilizing automatic differentiation for accurate derivative calculations. However, standard PINNs often face challenges in strictly enforcing boundary conditions (BCs), resulting in numerical inaccuracies and slow convergence. To overcome this, we proposed an innovative method that precisely enforces BCs through a transformation, thereby eliminating residual errors and significantly improving the reliability and performance of the PINN framework. Numerical experiments validated the effectiveness of the proposed approach, showing improved accuracy, faster convergence, and more stable training dynamics. Detailed analyses were conducted to investigate the influence of key hyperparameters, such as activation functions, network architecture, and learning rates, on the model's performance.

Keywords: Bratu problem; physics-informed neural networks; machine learning; neural network
Mathematics Subject Classification: 34B16, 65L10, 65L60

1. Introduction

Recent advancements in machine learning (ML) and deep learning have led to the emergence of neural network (NN)-based frameworks to address challenges in numerous fields such as computer vision, natural language processing, and game theory [1–3]. NNs are computational models inspired

by biological systems, extensively studied and applied in various scientific and engineering areas since their inception in the early 1950s. The foundational concepts of contemporary artificial NNs were introduced in 1944 by Warren McCulloch and Walter Pitts, through the influential McCulloch-Pitts model. This model, featuring interconnected input, hidden, and output layers, effectively modeled basic logic gates, paving the way for the evolution of more sophisticated NN architectures [4]. Traditional NNs generally employ supervised learning methods to create a functional relationship between input data and their corresponding output values. This is achieved by minimizing an error function that quantifies the difference between predicted and actual outputs, making NNs particularly powerful for nonlinear approximations.

Lagaris et al. [5] were the first to introduce the idea of employing NNs to solve differential equations (DEs). Nonetheless, because of the limited computational power available at that time, this method did not gain significant practical traction immediately after its inception. In 2019, Raissi et al. revisited the idea with PINNs, which addressed key limitations of earlier models by embedding physical laws directly into the training process of neural networks, significantly improving both accuracy and data efficiency [6]. Furthermore, advancements in automatic differentiation (AD) [7] and the broad accessibility of open-source Python libraries, such as TensorFlow and PyTorch, have played a significant role in the growing popularity of PINNs [8].

PINNs enhance the conventional NN framework by incorporating extra collocation points where the predicted outcomes must align with the governing physical equations. This technique utilizes the ability of AD to accurately compute the differential operator at these collocation points, which is a key advantage of the PINNs method. The goal of PINNs is to integrate the governing laws, represented as DEs, into NN architectures. The PINN framework employs NNs to simulate the physical properties of a system, including its initial and BCs [6]. Progress in PINN frameworks holds promise for developing more transparent and reliable scientific ML models, particularly for addressing scientific problems in which model accuracy and flexibility are crucial [9]. PINNs are trained by embedding the governing physical laws, expressed as mathematical representations, into the NN architecture. In recent years, numerous studies have investigated the use of PINNs across a range of technical fields, such as mechanics, fluid dynamics, power systems, finance, and geosciences [6, 10–12]. Lu et al. created a Python library, DeepXDE, that supports the implementation of the PINNs approach for various types of partial DEs [13].

Nonlinear boundary value problems (BVPs) are essential for modeling a wide range of physical, chemical, and engineering phenomena. Among them, the Bratu equation is particularly noteworthy because of its importance in areas such as chemical reaction theory, combustion, electrospinning, astrophysics, and nanotechnology [14–18]. Originally introduced by G. Bratu [19] in 1914 during investigations into thermal explosion theory, this equation has been the subject of extensive study both for its mathematical characteristics and its practical applications.

In this study, we consider the following well-known Bratu problem:

$$z''(x) = -ke^{z(x)}, \quad 0 \le x \le 1, \tag{1.1}$$

$$z(0) = 0, \quad z(1) = 0. \tag{1.2}$$

For $k > 0$, the exact solution of (1.1) and (1.2) was obtained in [20–23] as

$$z(x) = -2\ln\Big[\frac{\cosh(0.5(x-0.5)\theta)}{\cosh(\theta/4)}\Big],$$

where θ satisfies the transcendental equation

$$\theta = \sqrt{2k}\cosh(\theta/4). \tag{1.3}$$

When $k > k_c$, (1.1) and (1.2) have no solution, whereas for $k < k_c$, they have two solutions and a unique solution for $k = k_c$, where k_c is known as the critical value given by

$$1 = \frac{1}{4}\sqrt{2k_c}\sinh(\theta/4)$$

and evaluated in [15, 24] as $k_c = 3.513830719$.

Problem (1.1) is also recognized as a quasi-linear parabolic problem [12], and can be formulated as

$$\begin{cases} z(x) = \Delta z(x) + k(1-\delta z(x))^m e^{z(x)/(1-\delta z(x))}, & x \in \Omega, \\ z(x) = 0, & x \in \Gamma, \end{cases}$$

where Ω denotes a bounded domain, Γ denotes a smooth boundary, and δ represents the activation energy. In the context of combustion theory, (1.1) illustrates a model for solid fuel ignition. As demonstrated by Gidas et al. [25], all the solutions to this system exhibit radial symmetry for general domains $\Omega = B_1(0)$, which is the unit ball in R^1. Consequently, the nonlinear BVP equivalent to (1.1) can be formulated [12, 25] as

$$\begin{cases} z''(x) + \frac{M-1}{x}z'(x) + ke^{z(x)} = 0, \quad 0 \le x \le 1, \\ z'(0) = z(1) = 0. \end{cases}$$

In the case of $M = 1$, we get

$$z''(x) + ke^{z(x)} = 0,$$

and the above equation is known as the nonlinear Bratu problem (1.1).

The Bratu equation, a highly nonlinear BVP, has attracted considerable attention from researchers owing to its wide range of applications in various physical domains. This problem is recognized as a challenging benchmark problem because of the difficulty in finding a solution that is close to the critical value. The Bratu problem has been addressed using numerous analytical and numerical methods owing to its scientific importance. Solutions to problems (1.1) and (1.2) have been developed using techniques such as Taylor wavelets [26], Adomian decomposition [27], Lie-group shooting [28], the finite difference scheme [29], the Chebyshev polynomial expansion approach [30], Sinc-Galerkin methods [31], splines [32, 33], a method combining Laplace transforms with a decomposition technique [34], differential transformation [35], the continuous block Nyström method [36], variational iteration [37], Chebyshev wavelets [38], perturbation iteration [39], Legendre wavelets [40], a decomposition method [41], Jacobi-Gauss collocation [42], and bio-inspired algorithms [43]. A comprehensive overview of the Bratu problem can be found in [15, 26, 44] and the references included therein.

Recently, PINNs have garnered attention as robust alternatives to conventional methods for solving DEs [6, 12, 13, 45, 46]. PINNs, which utilize ML and NN techniques [3], address several limitations of classical approaches, including their reliance on computationally expensive mesh generation, difficulties with irregular domains, incomplete BCs, and challenges in high-dimensional problems.

The PINN approach transforms the task of approximating solutions into an optimization problem with the aim of minimizing a loss function. This is accomplished by utilizing NNs to represent the solution and define the loss function as the aggregate of residuals from the initial/BCs and the governing DE evaluated at the chosen collocation points. This method trains NNs to accurately model the physical behavior of a problem by incorporating initial/BCs and governing laws into the network structure.

Traditional PINNs incorporate BCs through penalty terms in the loss function and rely on network optimization to reduce residual errors at the boundaries, which might not ensure perfect enforcement. Typically, the standard PINN framework exhibits higher training and test losses for BCs when they are not fully satisfied, resulting in a network that performs well within the domain but fails to accurately represent the specified boundary values. By using a suitable output transformation, the BCs are naturally met at each training step, thus removing residual errors at the boundaries.

When BCs are imposed solely through penalty terms in the loss function, a challenge arises between minimizing the physical residual and satisfying the boundary constraint. This competition can result in a slower convergence owing to an imbalanced loss function and potential numerical instability. By contrast, enforcing BCs via transformation allows the NN to concentrate entirely on learning the unknown part of the solution. This approach leads to faster convergence and more stable training dynamics, as the network is not burdened with the additional task of satisfying the BCs through penalty terms. Standard PINNs frequently encounter issues with boundary discrepancies, where the solutions they learn tend to diverge near the boundaries owing to inadequate training. By enforcing the BCs, the model is guided to learn a solution that precisely meets the specified constraints, thereby enhancing generalization, accuracy, and robustness. Furthermore, although determining a solution when k is close to the critical value is challenging, the proposed method provides accurate approximate analytical solutions.

The remainder of this article is structured as follows. Section 2 introduces the proposed method in detail. Section 3 presents numerical experiments and a detailed investigation of the influence of key hyperparameters. Finally, Section 4 concludes the study and discusses future work.

2. Methodology

This section begins with an introduction to NNs, and subsequently introduces PINNs to address the Bratu problem.

2.1. An overview of neural networks (NNs)

NNs have emerged as powerful artificial intelligence techniques for function approximation because of the universal approximation theorem [47,48], demonstrating the ability to capture intricate nonlinear relationships between input and output variables. Inspired by the architecture and operation of the human brain, NNs consist of interconnected units, known as neurons, that communicate with each other. Each neuron applies an activation function to the weighted sum of its inputs. This configuration enables NNs to represent complex input-output relationships through the collaboration of numerous neurons, nonlinear transformations via activation functions, and optimization algorithms that adjust the network parameters. The fundamental elements of an NN include the input and output layers, hidden layers, connection weights, and neuron biases. These networks can handle either single or multiple input vectors and produce corresponding single or multiple output vectors. An NN is composed of

several layers of interconnected neurons, with each layer using an activation function to process the inputs and generate the outputs for the next layer. The output of a particular layer is determined by combining the weighted outputs from the preceding layer with the biases of neurons in the current layer.

In formal terms, an NN is defined as a function that converts an input vector $\mathbf{x} \in \mathbb{R}^{d_{in}}$ into an output vector $\mathbf{z} \in \mathbb{R}^{d_{out}}$ using the equation

$$\mathbf{z} = \mathcal{N}(\mathbf{x}, \boldsymbol{\theta}),$$

where $\mathcal{N} : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ represents the NN function with model parameters denoted as θ. According to the universal approximation theorem, a feed-forward NN with only one hidden layer and a finite number of neurons can approximate any continuous function on a compact input domain as long as the activation function is non-constant, bounded, and generally increases monotonically [48]. This capability makes NNs a powerful tool for solving DEs because they can be trained to approximate unknown functions that serve as solutions. The NN aims to approximate the target function $z(x)$, which represents the solution to an ordinary differential equation (ODE). The NN output z_θ is the estimated solution evaluated at different x values. In a traditional NN framework, this can be expressed as a fully connected feedforward neural network:

$$z_\theta(x) = (\mathcal{Z}_N \odot \mathcal{Z}_{N-1} \ldots \mathcal{Z}_0)(x), \tag{2.1}$$

where $\odot$ signifies the composition operator and $\theta = \{W_l, b_l\}_{l=1,N}$ includes the trainable parameters, such as weight matrices W_l and bias vectors b_l of the network. The NN is composed of $(N+1)$ layers, which include $(N-1)$ hidden, input, and output layers. The input layer $\mathcal{Z}_0$ represents independent variables x. The output layer $\mathcal{Z}_N$, which contains a single neuron, represents the predicted solution z_θ and serves as the NN approximation of the target function. Figure 1 provides a schematic representation of a typical NN architecture.

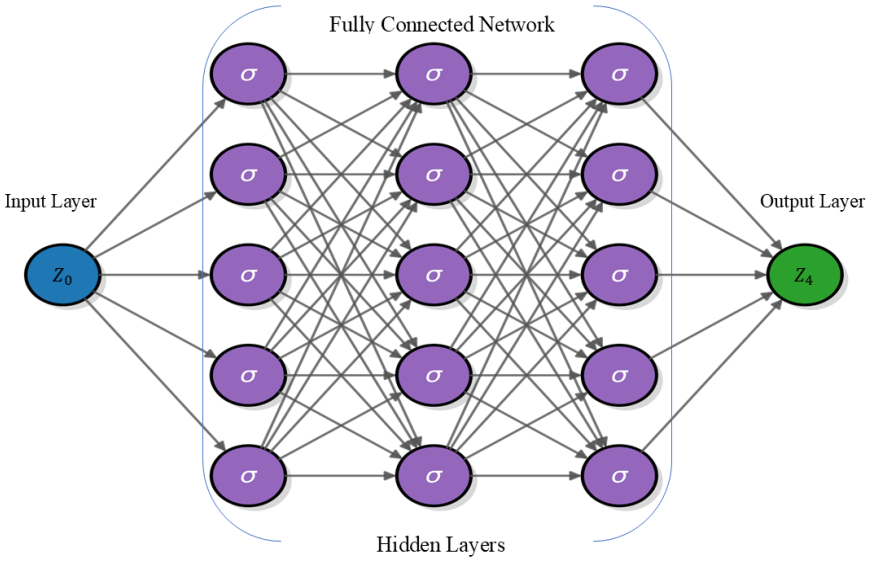


Figure 1. Architecture of a standard NN, comprising three hidden layers each with five neurons, as well as an input layer and an output layer.

The solution is depicted as an NN, which consists of nonlinear functions applied throughout the hidden layers of the architecture, given by

$$\mathcal{Z}_l(x) = \sigma\left(W_l \mathcal{Z}_{l-1}(x) + b_l\right), \quad 1 \leq l \leq (N-1), \tag{2.2}$$

where $W_l \in \mathbb{R}^{d_{l-1} \times d_l}$, $b_l \in \mathbb{R}^{d_l}$ (d_l denotes the dimension of the input vector for layer l), and $\sigma(.)$ signifies a nonlinear activation function applied individually to each input element. Neurons in an NN employ activation functions to handle the input and output data. These functions are crucial to the network's ability to learn and represent intricate nonlinear relationships. The primary function of activation functions is to introduce non-linear transformations, allowing NNs to understand the inherent nonlinearity and represent complex nonlinear function mappings. Commonly used activation functions include the hyperbolic tangent (tanh), swish, exponential linear unit (ELU), sigmoid, rectified linear unit (ReLU), and leaky ReLU. Among these, the tanh function is the most frequently used [49]. A list of the activation functions is provided in Table 1 and plotted in Figure 2.

Table 1. Activation functions ($\sigma(x)$).

Tanh	Swish	ELU	ReLU	Sigmoid
$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	$\frac{x}{1+e^{-x}}$	$\begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases}$	$\begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$	$\frac{1}{1+e^{-x}}$

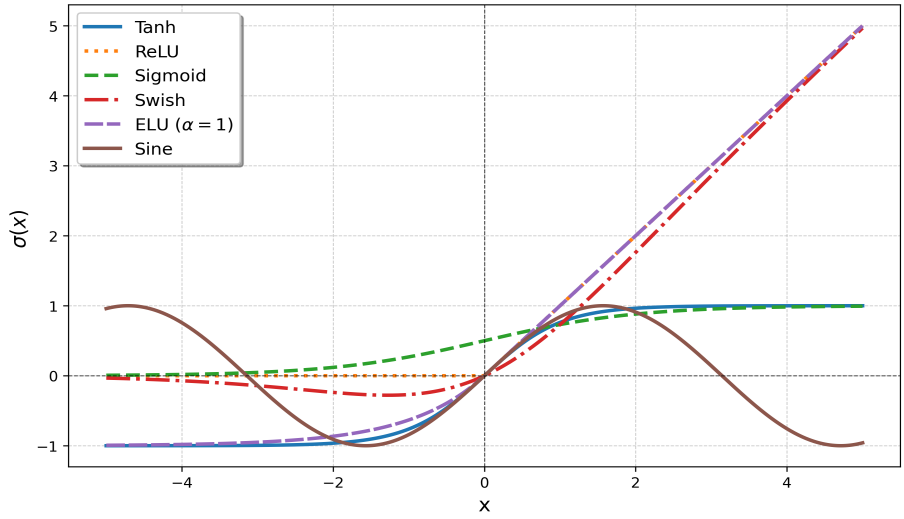


Figure 2. Plot of different activation functions.

The parameters of the NN are then fine-tuned by minimizing a loss function. This function was formulated as the discrepancy between the governing DE, BCs, and initial conditions assessed at specific points within the specified domain of interest.

2.2. PINNs for solving the Bratu problem

The core idea behind the PINN approach is to seamlessly incorporate physical laws, represented by the governing DEs and their related boundary/initial conditions, into the NN framework. Unlike

conventional data-driven NNs, which depend solely on input-output training data, PINNs integrate the governing physical equations directly into their structural design. This approach allows the network to learn from both the available data and the intrinsic physical constraints of the problem, providing a more thorough understanding of the system being analyzed. The general approach of PINNs for solving ordinary DEs can be illustrated by considering a general ODE

$$\mathcal{L}[z(x)] = f(x) \quad \forall \quad x \in \Omega, \tag{2.3}$$

with

$$\mathcal{B}[z(x)] = h(x) \quad \forall \quad x \in \partial\Omega, \tag{2.4}$$

where $\mathcal{L}$ denotes the differential operator and $\mathcal{B}$ represents the BCs. Let us denote the approximate solution of the DE derived from (2.2) as $z_\theta(x) = \mathcal{Z}_N$. The PINN loss $\mathfrak{L}_{total}$ is defined as the sum of the residuals of the problem (i.e., related to physics) L_p and its BCs L_{bc}. The goal of the optimization problem is to find $z_\theta(x)$ that approximates $z(x)$ by minimizing $\mathfrak{L}_{total}$, which acts as the loss function and is given by

$$\mathfrak{L}_{total}(\theta) = L_p + L_{bc}, \tag{2.5}$$

where

$$L_p = \frac{1}{N_p} \sum_{i=1}^{N_p} |\mathcal{L}[z_\theta(x_i)] - f(x_i)|^2 ,$$

and

$$L_{bc} = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} |\mathcal{B}[z_\theta(x_i)] - h(x_i)|^2 .$$

Here, N_p and N_{bc} represent the number of collocation points in the domain Ω and boundary $\partial\Omega$, respectively. The essential principle of PINNs involves minimizing the loss function (2.5) with equal weights to train the NNs. A notable feature of PINNs is their use of AD to compute derivatives of the anticipated solution with respect to x, which are crucial components of the loss function. This technique eliminates the truncation and discretization errors that are typically associated with traditional numerical methods.

Now (1.1) can be expressed in an equivalent form to implement the PINN framework as

$$\mathcal{F}[z](x) = z''(x) + ke^{z(x)} = 0. \tag{2.6}$$

To establish the PINN framework for deriving the approximate solution $z_\theta(x)$ of (1.1), we can construct the loss function component, which represents the governing equation, as follows:

$$L_p(z_\theta) = \frac{1}{N_p} \sum_{i=1}^{N_p} |\mathcal{F}[z](x_i)|^2 , \tag{2.7}$$

with N_p grid points on $[0, 1]$. In addition, we established another component of the loss function (related to BCs) as follows:

$$L_{bc}(z_\theta) = |z_\theta(0)|^2 + |z_\theta(1)|^2 . \tag{2.8}$$

Conventional PINNs employ a combined loss function $\mathfrak{L}_{total}(z_\theta) = L_p(z_\theta) + L_{bc}(z_\theta)$ which is minimized during the training phase. However, the primary shortcoming of this method is that it

may result in a predicted solution that may not ensure that BCs are exactly satisfied. To overcome this issue, we propose a modified framework that introduces an additional layer in which the NN output is transformed to inherently satisfy the BCs

$$\widehat{z_\theta} = g(x)z_\theta(x), \quad \text{where} \quad g(x) = \frac{x(1-x)k}{7}.$$

Note that $g(x)$ vanishes at the prescribed boundary points. In the case of the Bratu equation with Dirichlet BCs $z(0) = z(1) = 0$, this function satisfies

$$g(0) = 0, \quad g(1) = 0.$$

Therefore,

$$\tilde{z}_\theta(0) = g(0)\, z_\theta(0) = 0, \quad \tilde{z}_\theta(1) = g(1)\, z_\theta(1) = 0,$$

for any $z_\theta(x)$ produced by the neural network, independent of the training process. The exact boundary satisfaction holds for any such scaling factor $k/7$, and the $(k/7)$ factor was adopted purely for practical training convenience.

The training process involves minimizing the total objective function $L_{total}(\widehat{z_\theta})$. In our proposed PINN framework, the following loss function $\mathcal{L}$ is defined as follows:

$$\mathfrak{L}_{total}(\widehat{z_\theta}) = L_p(\widehat{z_\theta}). \tag{2.9}$$

The enhanced PINN approach incorporates BCs through an output transformation rather than penalty terms in the loss function. This method ensures that the BCs are naturally met at each training step, eliminating residual errors and improving the convergence and stability. By removing the need to balance physics residuals with boundary constraints, the NN can focus on learning unknown aspects of the solution. This approach reduces computational overhead, enhances optimization efficiency, and improves generalization and accuracy. The result is a more precise, stable, and computationally efficient PINN implementation compared with standard methods that rely on penalty terms. Figure 3 illustrates a schematic representation of our PINN architecture for solving the Bratu problem.

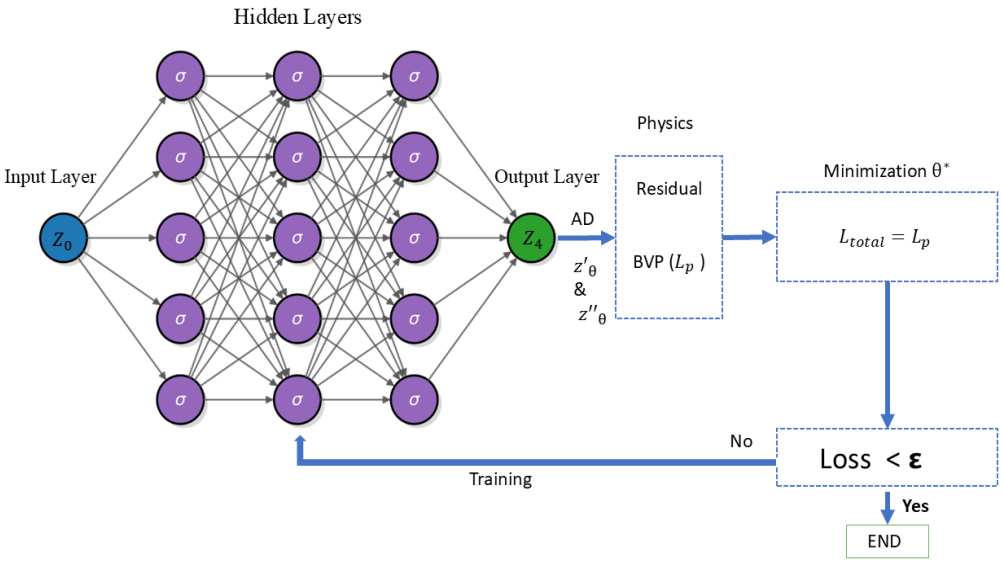


Figure 3. Architecture of PINNs.

The core of PINNs minimizes the loss function to train the NNs. Notably, PINNs utilize AD to obtain the exact derivatives of the expected solution with respect to x, which are required in the loss function. Finally, the training process involves using a gradient descent algorithm to minimize the loss function until convergence is achieved for a required accuracy level (or defined maximum iteration number) as

$$\theta_{i+1} = \theta_i - \eta \nabla_\theta \Omega_{total}(\theta_i),$$

for the ith iteration leading to

$$\theta^* = \text{argmin}_\theta \, \Omega_{total}(\theta).$$

Parameter η, referred to as the learning rate, is essential to the training process and a carefully chosen fixed learning rate is sufficient for stable training. The NN's adjustable parameters include the bias vector and weight matrix, which are generally initialized randomly using either a normal or uniform distribution, or through methods such as Xavier/Glorot uniform or normal initialization. This initialization, introduced in [50], is a technique designed to optimize the training of NNs. Backpropagation [51] offers an efficient method for computing gradients and facilitates gradient-based optimization. During the training of the NN, the gradients obtained through backpropagation are used to iteratively refine network parameters. Various optimizers that are crucial to the training process have been utilized by researchers. Among the popular methods are stochastic gradient descent, limited-memory Broyden–Fletcher–Goldfarb–Shanno scheme (BFGS), and Adam. To perform gradient descent optimization, it is essential to calculate the derivatives of NN outputs with respect to inputs as well as the derivative of the loss function concerning the trainable parameters. The conventional AD method is vital for computing derivatives (∇_θ) with respect to parameters, such as weights and biases. AD uses the chain rule to efficiently determine these derivatives through a two-step process: a forward pass to evaluate the variable values, followed by a backward pass to calculate the derivatives. Numerous computational frameworks have offered AD libraries to streamline this process. The integration of AD has significantly advanced PINNs, evolving them from promising ideas to powerful tools for tackling complex problems. The goal is to fine-tune the network's trainable parameter θ, such that the NN output $z_\theta(x)$ aligns closely with the actual or desired solution $z(x)$. This integration has transformed PINNs into robust tools for solving complex differential equations without requiring mesh-based discretization, thus allowing for flexible treatment of complicated geometries and BCs.

We selected the Adam optimizer [52] and employed the Glorot uniform-initialization technique. Selecting an appropriate optimizer is crucial for PINNs to avoid local minima and to enable higher accuracy and convergence. The network training process uses optimization and backpropagation to adjust the trainable parameters to minimize the loss function and approximate the target solution. Training is implemented using our PINN framework with TensorFlow, incorporating the Adam optimization solver [52]. Adam integrates the adaptive gradient and root-mean-squared propagation, harnessing the advantages of both. The method used to initialize unknown parameters and the learning rate of the optimizer are crucial aspects. The learning rate determines the magnitude of the steps in gradient descent training. A high learning rate may cause overshoots, while an excessively low rate can cause the optimization to get stuck in a local minimum, resulting in suboptimal performance. Fine-tuning an NN is crucial for achieving accurate results. Our study explored the impact of key factors and hyperparameters.

3. Numerical results

In this section, we present numerical simulations to validate the proposed PINNs approach. The implementation was carried out using TensorFlow [53] as the Python framework on a system with an Intel Core i5 processor with 8 GB of RAM. To evaluate the performance of the PINN framework, we conducted extensive numerical experiments on the Bratu problem for various values of the parameter k. The following error metrics were used to evaluate the performance using test points:

$$E_{\text{abs}}(x_i) = |z_{\text{pred}}(x_i) - z_{\text{exact}}(x_i)|.$$

The relative error is computed as:

$$E_{relative} = \frac{E_{\text{abs}}(x_i)}{|z_{\text{exact}}(x_i)| + \epsilon},$$

with a stabilization constant $\epsilon = 10^{-10}$ to avoid division by zero at the boundaries. The mean squared error (MSE) is defined as:

$$E_{\text{MSE}} = \frac{1}{N_p} \sum_{i=1}^{N_p} \left(z_{\text{pred}}(x_i) - z_{\text{exact}}(x_i)\right)^2.$$

To solve problem (1.1) using the proposed PINN, we trained the NN architecture to minimize the total loss function (2.9). The Adam optimization algorithm with a learning rate of 0.001 was employed with epochs/steps for the optimizer set to 10^4 using the tanh activation function. The NN architecture consisted of three hidden layers, each comprising 50 neurons. A total of collocation/training points $N_p = 200$ were used, and the number of data/testing points was set to 200. The training and test losses for the residual of the BVP and BCs ($z(0)$ and $z(1)$) are presented in Tables 2–5 for the proposed and standard PINNs.The results show that the proposed PINNs framework enforces BCs exactly, unlike standard PINNs, which demonstrate noticeable discrepancies. A comparison of the predicted and exact solutions is provided in Figure 4 for different values of k. From Figure 4, it is evident that the predicted solution is in good agreement with the exact solution. In addition, we plot the solution surface for different values of k in Figure 5. Figure 6 illustrates the training loss convergence. Further, the MSE and absolute and relative errors are shown in Figure 7 and Figure 8. The computation cost for $k = 1$ is 34 seconds for 10^4 epochs as shown in Table 2.

The proposed PINN framework exhibited a superior performance in solving the Bratu equation. By enforcing BCs directly through an output transformation, residual boundary errors were eliminated. The approach yielded high accuracy, low MSE, and robust convergence, even near the Bratu bifurcation point. The predicted solutions closely aligned with analytical ones, as confirmed by both visual comparisons and error metrics.

Table 2. Comparison of train/test losses of standard PINNs with proposed PINNs for $k = 1$.

Component	Loss	Proposed PINNs	Standard PINNs
Residual (BVP)	Train	$2.48e-08$	$1.40e-06$
	Test	$2.57e-08$	$1.20e-06$
Left boundary $z(0)$	Train	0.00	$1.07e-11$
	Test	0.00	$1.07e-11$
Right boundary $z(1)$	Train	0.00	$7.85e-11$
	Test	0.00	$7.85e-11$

Table 3. Comparison of train/test losses of standard PINNs with proposed PINNs for $k = 2$.

Component	Loss	Proposed PINNs	Standard PINNs
Residual (BVP)	Train	$7.87e-08$	$5.74e-06$
	Test	$7.79e-08$	$4.65e-06$
Left boundary $z(0)$	Train	0.00	$2.95e-11$
	Test	0.00	$2.95e-11$
Right boundary $z(1)$	Train	0.00	$1.17e-10$
	Test	0.00	$1.17e-10$

Table 4. Comparison of train/test losses of standard PINNs with proposed PINNs $k = 3$.

Component	Loss	Proposed PINNs	Standard PINNs
Residual (BVP)	Train	$2.93e-08$	$1.05e-04$
	Test	$3.08e-08$	$9.00e-05$
Left boundary $z(0)$	Train	0.00	$1.23e-07$
	Test	0.00	$1.23e-07$
Right boundary $z(1)$	Train	0.00	$1.10e-06$
	Test	0.00	$1.10e-06$

Table 5. Comparison of train/test losses of standard PINNs with proposed PINNs $k = 3.51$.

Component	Loss	Proposed PINNs	Standard PINNs
Residual (BVP)	Train	$2.58e-07$	$6.69e-04$
	Test	$2.47e-07$	$6.31e-04$
Left boundary $z(0)$	Train	0.00	$1.29e-04$
	Test	0.00	$1.29e-04$
Right boundary $z(1)$	Train	0.00	$1.76e-04$
	Test	0.00	$1.76e-04$

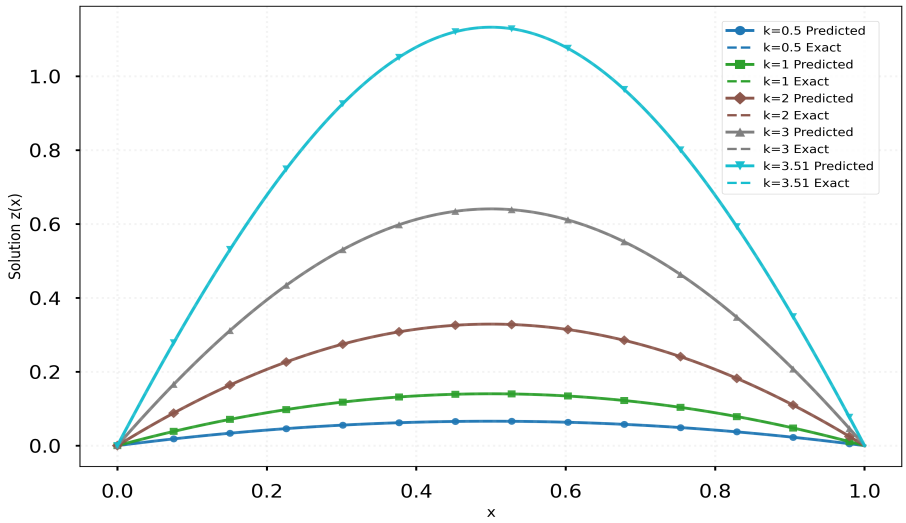


Figure 4. Predicted and exact solutions for different values of k.

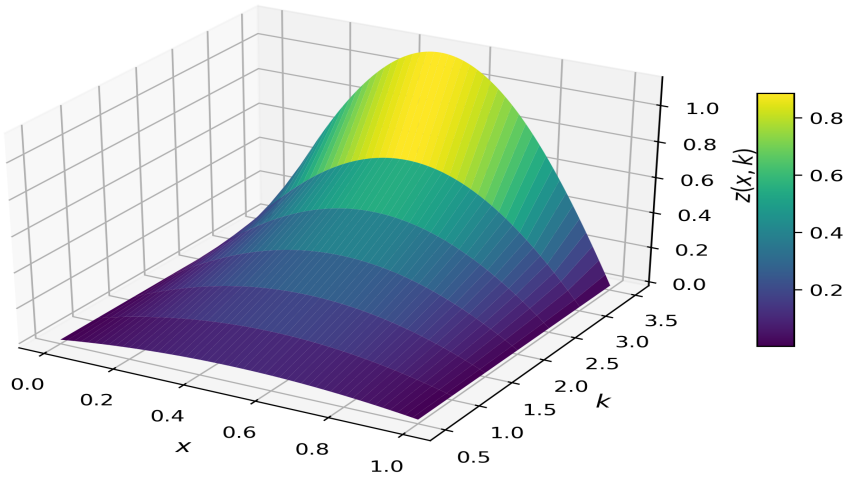


Figure 5. Solution surface for different values of k.

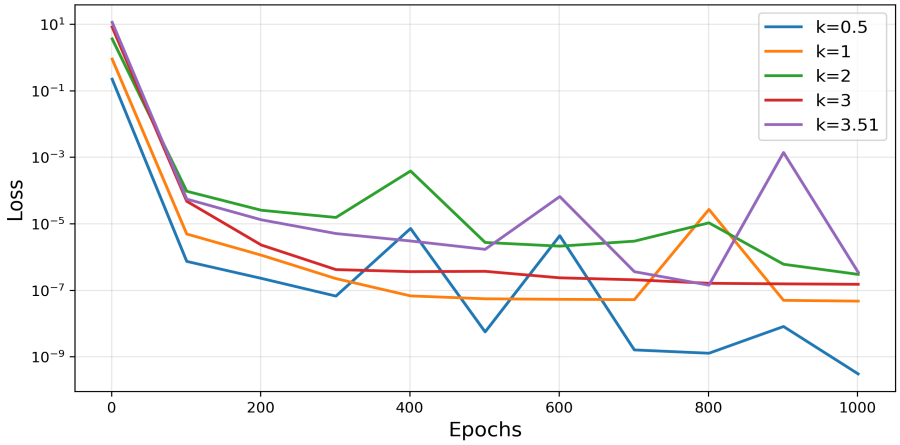


Figure 6. Training convergence for different values of k.

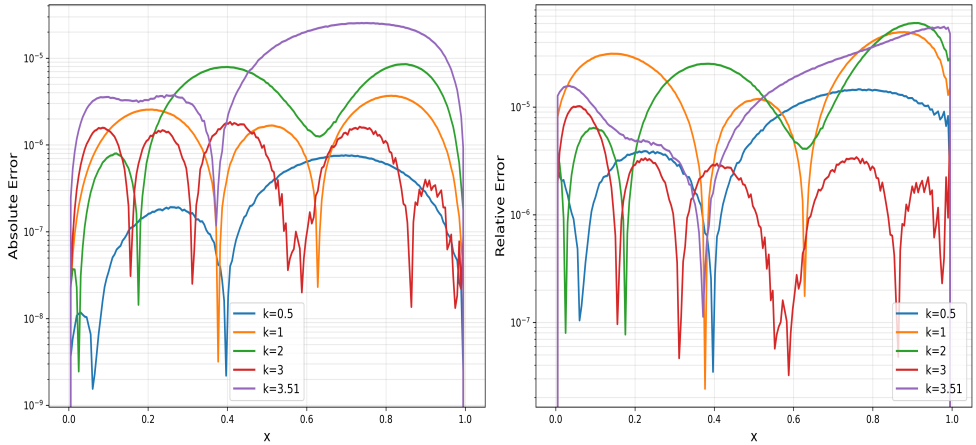


Figure 7. Errors for different values of k.

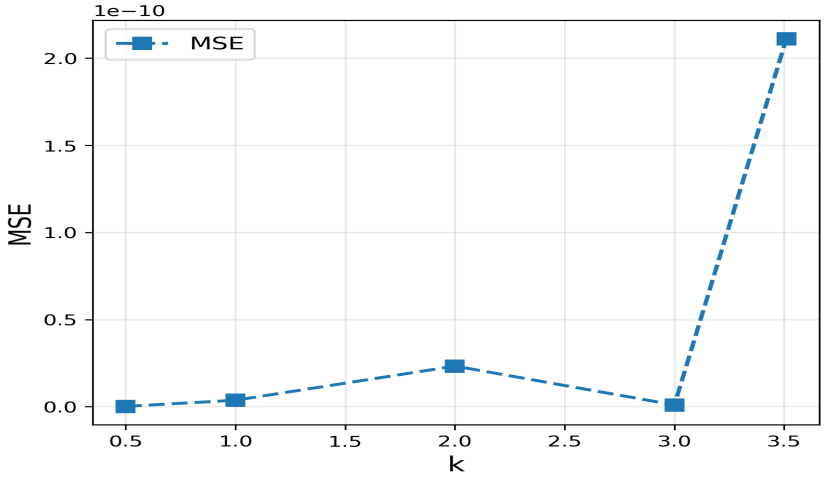
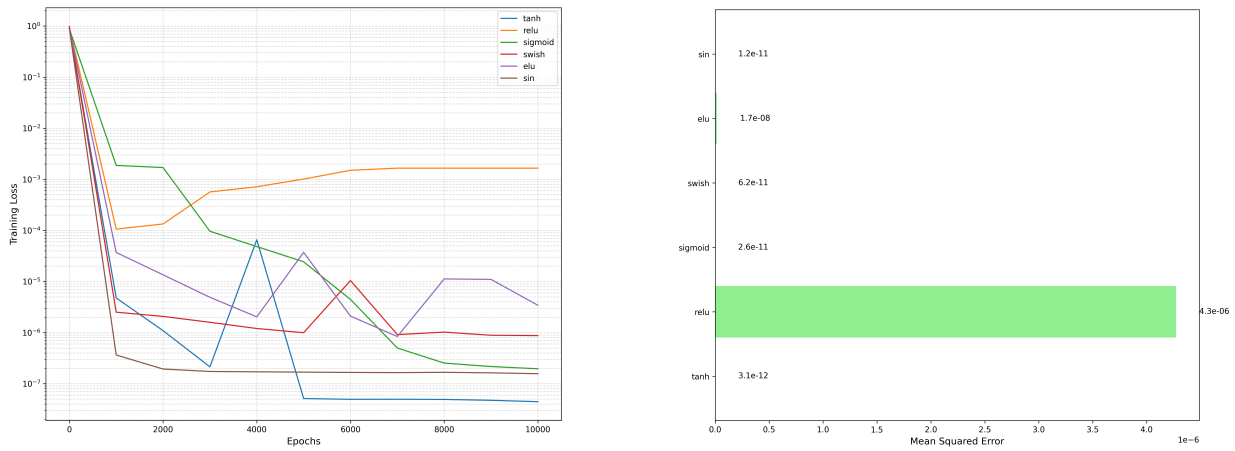


Figure 8. Mean squared error (MSE) for different values of k.

3.1. *Influence of different parameters*

This study explores how different activation functions affect the performance of NNs in solving the Bratu problem. The network is structured with the same three hidden layers, each containing 50 neurons, and is trained with a learning rate of 0.001 using 200 collocation points. We assessed various activation functions, including tanh, ReLU, sigmoid, swish, ELU, and sin. Figure 9 offers a detailed comparison of training loss convergence and solution accuracy (MSE) for these activation functions. The findings clearly indicate that the tanh activation function outperformed the others, achieving both faster convergence in the training loss and better accuracy. This enhanced performance is likely due to the smooth gradient characteristics and symmetric output range of tanh, which are particularly advantageous for solving DEs that require balanced positive and negative outputs. Other activation functions also perform well, whereas ReLU showed faster initial convergence but ended with a higher final error, probably because of the dying ReLU issue in deeper networks. These results imply that smooth, bounded activation functions, such as tanh, are particularly effective for physics-informed NNs tackling DEs.



(a) Training loss convergence

(b) Solution accuracy (MSE)

Figure 9. Performance comparison of different activation functions.

The impact of learning rates (LRs) on network performance was assessed using a tanh-activated model with three hidden layers, each containing 50 neurons, trained on 200 collocation points. Figure 10 illustrates the varied optimization behaviors across LRs ranging from 10^{-5} to 10^{-1}. A higher LR (LR = 0.1) caused unstable training with large loss fluctuations, whereas very low rates (LR = 10^{-5}) led to extremely slow convergence. The ideal balance is found at LR = 0.001, which achieves rapid exponential loss reduction. This performance relationship, where deviations from the optimal LR increase the error, underscores the importance of selecting the right LR in PINNs. The smooth gradients of tanh worked well with moderate LRs to ensure a stable backpropagation. These findings offer quantitative guidance for solving DEs, with LR $\approx$ 0.001 optimizing both the efficiency and accuracy of the architecture.

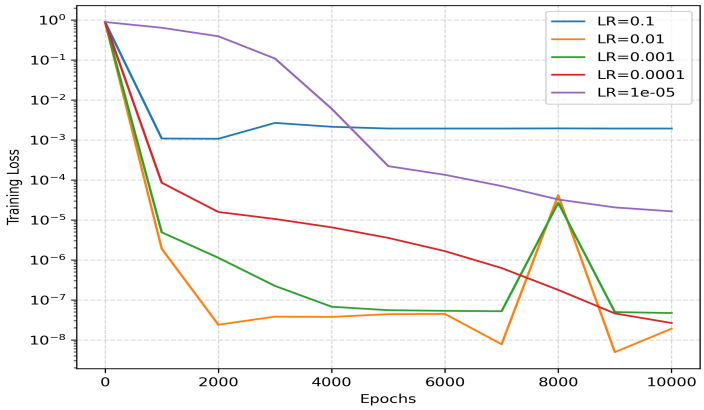


Figure 10. Comparison of training loss with different learning rates (LRs).

The impact of the NN architecture on solution accuracy was thoroughly examined using the tanh activation function, maintaining a constant learning rate of 0.001 and utilizing 200 training points. As shown in Figure 11, the evaluation of different architectures highlights unique performance traits. Networks with 1-2 layers tend to have an MSE ranging from $e-11$ to $e-12$, with narrow setups (20 neurons per layer) having difficulty handling nonlinearities. The best results are observed with intermediate architectures, where a 3-layer, 50-neuron setup achieved the lowest MSE of $3.2e-12$, marking an improvement over similar 2-layer networks. More complex architectures (4-5 layers) show diminishing returns, with MSE values leveling off around $5.1e-12$ despite having more parameters, indicating possible overparameterization. Interestingly, expanding shallow networks (e.g., 1-layer with 100 neurons) is less effective than adding a moderate depth (3-layer with 50 neurons), underscoring the significance of hierarchical feature learning in solving PDEs. Additionally, training dynamics revealed that deeper networks require more epochs to converge, although they eventually reach more stable minima. These findings quantitatively illustrate that for tanh-activated networks addressing the problem, a 3-layer architecture with 50 neurons per layer achieves the best balance between model expressiveness and training efficiency.

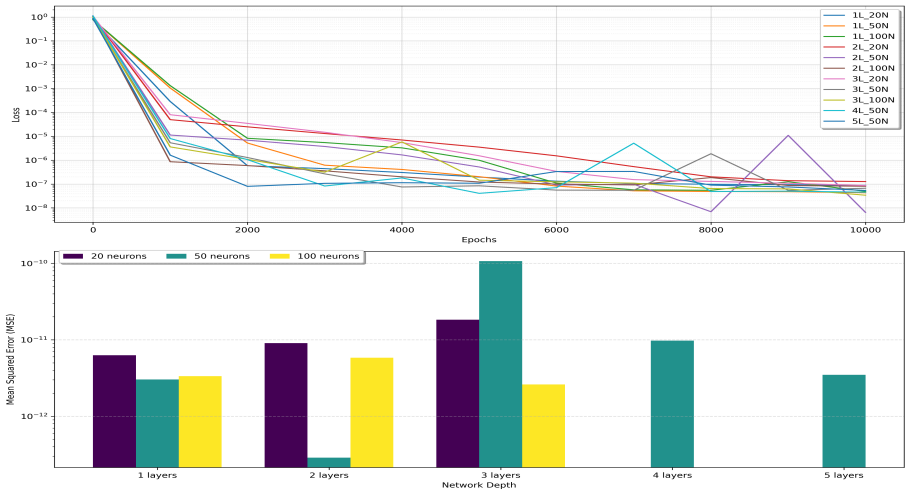


Figure 11. Comparison of architecture study: (Top) Training loss convergence with layer (L) and neuron (N); (Bottom) MSE versus network depth.

4. Conclusions

In this study, we introduced a PINN framework to tackle the highly nonlinear Bratu equation, overcoming the shortcomings of traditional PINNs in enforcing BCs and obtaining precise solutions. Our method incorporates BCs directly via transformation, ensuring strict adherence to the constraints of the problem. This approach significantly enhances both the accuracy and convergence speed of the model.

Numerical experiments confirmed the effectiveness of the proposed method, demonstrating its ability to manage the nonlinearity and critical parameter values of the Bratu equation with good accuracy. Comprehensive analyses of hyperparameters, including activation functions, network depth, and learning rates, offered valuable insights into optimizing the PINN performance. The tanh activation function proved to be the most effective, while a moderate learning rate and balanced network architecture were essential for achieving optimal results. Our findings highlight the potential of PINNs as powerful tools for solving complex DEs, offering advantages over traditional mesh-based methods in terms of flexibility and computational efficiency. Future research could extend this framework to higher-dimensional Bratu problems. In addition, further exploration of adaptive activation functions and advanced optimization techniques can enhance the model's performance and scalability. Developing rigorous error estimates of PINNs applied to nonlinear problems like the Bratu equation is also an important direction for future research.

Author contributions

Saurabh Tomar: Conceptualization, investigation, writing–original draft, writing–review and editing, methodology, validation; Higinio Ramos: Writing–review and editing, writing–original draft, validation, investigation. All authors have read and approved the final version of the manuscript for publication.

Use of Generative-AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

The first author gratefully acknowledges the support of the Indian Institute of Technology Kanpur. The authors are grateful for the valuable feedback from the ditor and the reviewers, which has helped to improve the manuscript significantly.

Conflict of interest

Higinio Ramos is an editorial board member for AIMS Mathematics and was not involved in the editorial review and/or the decision to publish this article. All authors declare no conflicts of interest in this paper.

References

1. J. A. Nichols, H. W. H. Chan, M. A. B. Baker, Machine learning: Applications of artificial intelligence to imaging and diagnosis, *Biophys. Rev.*, **11** (2019), 111–118. https://doi.org/10.1007/s12551-018-0449-9
2. H. Wang, T. Fu, Y. Du, W. Gao, K. Huang, Z. Liu, et al., Scientific discovery in the age of artificial intelligence, *Nature*, **620** (2023), 47–60. https://doi.org/10.1038/s41586-023-06221-2
3. T. Hey, K. Butler, S. Jackson, J. Thiyagalingam, Machine learning and big scientific data, *Philos. Trans. A Math. Phys. Eng. Sci.*, **378** (2020), 20190054. https://doi.org/10.1098/rsta.2019.0054
4. W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophys.*, **5** (1943), 115–133. https://doi.org/10.1007/BF02478259
5. I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, *IEEE Trans. Neural Netw.*, **9** (1998), 987–1000. https://doi.org/10.1109/72.712178
6. M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *J. Comput. Phys.*, **378** (2018), 686–707. https://doi.org/10.1016/j.jcp.2018.10.045
7. A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic differentiation in machine learning: A survey, *J. Mach. Learn. Res.*, **18** (2018), 5595–5637.
8. S. R. Vadyala, S. N. Betgeri, N. P. Betgeri, Physics-informed neural network method for solving one-dimensional advection equation using PyTorch, *Array*, **13** (2022), 100110. https://doi.org/10.1016/j.array.2021.100110
9. D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, S. Hoyer, Machine learning–accelerated computational fluid dynamics, *Proc. Natl. Acad. Sci. U.S.A.*, **118** (2021), e2101784118. https://doi.org/10.1073/pnas.2101784118
10. J. Abbasi, P. Andersen, Simulation and prediction of countercurrent spontaneous imbibition at early and late time using Physics-informed neural networks, *Energy Fuels*, **37** (2023), 13721–13733. https://doi.org/10.1021/acs.energyfuels.3c02271
11. G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, *Nat. Rev. Phys.*, **3** (2021), 422–440. https://doi.org/10.1038/s42254-021-00314-5
12. V. R. Hosseini, A. A. Mehrizi, A. Gungor, H. H. Afrouzi, Application of a physics-informed neural network to solve the steady-state Bratu equation arising from solid biofuel combustion theory, *Fuel*, **332** (2023), 125908. https://doi.org/10.1016/j.fuel.2022.125908
13. L. Lu, X. Meng, Z. Mao, G. E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, *SIAM Rev.*, **63** (2021), 208–228. https://doi.org/10.1137/19M1274067
14. Y. Q. Wan, Q. Guo, N. Pan, Thermo-electro-hydrodynamic model for electrospinning process, *Int. J. Nonlinear Sci. Numer. Simul.*, **5** (2004), 5–8.
15. J. T. Jacobsen, K. Schmitt, The Liouville–Bratu–Gelfand problem for radial operators, *J. Differ. Equ.*, **184** (2002), 283–298. https://doi.org/10.1006/jdeq.2001.4151

16. I. M. Gelfand, Some problems in the theory of quasilinear equations, *Trans. Amer. Math. Soc.*, **29** (1963), 295–381.

17. J. V. Baxley, S. B. Robinson, Nonlinear boundary value problems for shallow membrane caps, II, *J. Commun. Appl. Anal.*, **88** (1998), 203–224. https://doi.org/10.1016/S0377-0427(97)00216-1

18. R. W. Dickey, Rotationally symmetric solutions for shallow membrane caps, *Quart. Appl. Math.*, **47** (1989), 571–581. https://doi.org/10.1090/qam/1012280

19. G. Bratu, Sur les équations intégrales non linéaires, *Bull. Soc. Math. France*, **42** (1914), 113–142.

20. R. Saleh, S. M. Mabrouk, M. Kassem, Truncation method with point transformation for exact solution of Liouville Bratu Gelfand equation, *Comput. Math. Appl.*, **76** (2018), 1219–1227. https://doi.org/10.1016/j.camwa.2018.06.016

21. M. Baccouch, H. Temimi, A new derivation of the closed-form solution of Bratu's problem, *Int. J. Appl. Comput. Math.*, **9** (2023), 100. https://doi.org/10.1007/s40819-023-01570-y

22. U. M. Ascher, R. M. M. Mattheij, R. D. Russell, *Numerical solution of boundary value problems for ordinary differential equations*, 1995. https://doi.org/10.1137/1.9781611971231

23. A. Mohsen, A simple solution of the Bratu problem, *Comput. Math. Appl.*, **67** (2014), 26–33. https://doi.org/10.1016/j.camwa.2013.10.003

24. R. Buckmire, Application of a Mickens finite-difference scheme to the cylindrical Bratu–Gelfand problem, *Numer. Methods Partial Differ. Equ.*, **20** (2004), 327–337. https://doi.org/10.1002/num.10093

25. B. Gidas, W. Ni, L. Nirenberg, Symmetry and related properties via the maximum principle, *Commun. Math. Phys.*, **68** (1979), 209–243. https://doi.org/10.1007/BF01221125

26. E. Keshavarz, Y. Ordokhani, M. Razzaghi, The Taylor wavelets method for solving the initial and boundary value problems of Bratu-type equations, *Appl. Numer. Math.*, **128** (2018), 205–216. https://doi.org/10.1016/j.apnum.2018.02.001

27. A. M. Wazwaz, Adomian decomposition method for a reliable treatment of the Bratu-type equations, *Appl. Math. Comput.*, **166** (2005), 652–663. https://doi.org/10.1016/j.amc.2004.06.059

28. S. Abbasbandy, M. S. Hashemi, L. S. Liu, The Lie-group shooting method for solving the Bratu equation, *Commun. Nonlinear Sci. Numer. Simul.*, **16** (2011), 4238–4249. https://doi.org/10.1016/j.cnsns.2011.03.033

29. H. Temimi, M. Ben-Romdhane, An iterative finite difference method for solving Bratu's problem, *J. Comput. Appl. Math.*, **292** (2016), 76–82. https://doi.org/10.1016/j.cam.2015.06.023

30. J. P. Boyd, Chebyshev polynomial expansions for simultaneous approximation of two branches of a function with application to the one-dimensional Bratu equation, *Appl. Math. Comput.*, **143** (2003), 189–200. https://doi.org/10.1016/S0096-3003(02)00345-4

31. J. Rashidinia, K. Maleknejad, N. Taheri, Sinc-Galerkin method for numerical solution of the Bratu's problems, *Numer. Algor.*, **62** (2013), 1–11. https://doi.org/10.1007/s11075-012-9560-3

32. H. Çağlar, N. Çağlar, M. Özer, A. Valarístos, A. Anagnostopoulos, B-spline method for solving Bratu's problem, *Int. J. Comput. Math.*, **87** (2010), 1885–1891. http://dx.doi.org/10.1080/00207160802545882

33. R. Jalilian, Non-polynomial spline method for solving Bratu's problem, *Comput. Phys. Commun.*, **181** (2010), 1868–1872. https://doi.org/10.1016/j.cpc.2010.08.004
34. S. A. Khuri, A new approach to Bratu's problem, *Appl. Math. Comput.*, **147** (2004), 131–136. https://doi.org/10.1016/S0096-3003(02)00656-2
35. I. H. A. Hassan, V. S. Ertürk, Applying differential transformation method to the one-dimensional planar Bratu problem, *Int. J. Contemp. Math. Sci.*, **2** (2007), 1493–1504.
36. S. N. Jator, V. Manathunga, Block Nyström type integrator for Bratu's equation, *J. Comput. Appl. Math.*, **327** (2018), 341–349. https://doi.org/10.1016/j.cam.2017.06.025
37. A. S. V. R. Kanth, K. Aruna, He's variational iteration method for treating nonlinear singular boundary value problems, *Comput. Math. Appl.*, **60** (2010), 821–829. https://doi.org/10.1016/j.camwa.2010.05.029
38. C. Yang, J. Hou, Chebyshev wavelets method for solving Bratu's problem, *Bound. Value Probl.*, **2013** (2013), 142.
39. Y. Aksoy, M. Pakdemirli, New perturbation–iteration solutions for Bratu-type equations, *Comput. Math. Appl.*, **59** (2010), 2802–2808. https://doi.org/10.1016/j.camwa.2010.01.050
40. S. G. Venkatesh, S. K. Ayyaswamy, S. R. Balachandar, The Legendre wavelet method for solving initial value problems of Bratu-type, *Comput. Math. Appl.*, **63** (2012), 1287–1295. https://doi.org/10.1016/j.camwa.2011.12.069
41. E. Deeba, S. A. Khuri, S. Xie, An algorithm for solving boundary value problems, *J. Comput. Phys.*, **159** (2000), 125–138. https://doi.org/10.1006/jcph.2000.6452
42. E. H. Doha, A. H. Bhrawy, D. Baleanu, R. M. Hafez, Efficient Jacobi-Gauss collocation method for solving initial value problems of Bratu type, *Comput. Math. Math. Phys.*, **53** (2013), 1292–1302. https://doi.org/10.1134/S0965542513090121
43. M. A. Z. Raja, R. Samar, E. S. Alaidarous, E. Shivanian, Bio-inspired computing platform for reliable solution of Bratu-type equations arising in the modeling of electrically conducting solids, *Appl. Math. Model.*, **40** (2016), 5964–5977. https://doi.org/10.1016/j.apm.2016.01.034
44. J. S. McGough, Numerical continuation and the Gelfand problem, *Appl. Math. Comput.*, **89** (1998), 225–239. https://doi.org/10.1016/S0096-3003(97)81660-8
45. X. Jiang, Z. Wang, W. Bao, Y. Xu, Generalization of PINNs for elliptic interface problems, *Appl. Math. Lett.*, **157** (2024), 109175. https://doi.org/10.1016/j.aml.2024.109175
46. S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, F. Piccialli, Scientific machine learning through physics–informed neural networks: Where we are and what's next, *J. Sci. Comput.*, **92** (2022), 88. https://doi.org/10.1007/s10915-022-01939-z
47. K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, *Neural Netw.*, **2** (1989), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8
48. K. Funahashi, On the approximate realization of continuous mappings by neural networks, *Neural Netw.*, **2** (1989), 183–192. https://doi.org/10.1016/0893-6080(89)90003-8
49. A. D. Jagtap, K. Kawaguchi, G. E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, *J. Comput. Phys.*, **404** (2020), 109136. https://doi.org/10.1016/j.jcp.2019.109136

50. X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, In: *Proceedings of the 13th international conference on artificial intelligence and statistics (AISTATS) 2010*, 2010, 249–256.

51. C. C. Aggarwal, *Neural networks and deep learning*, Springer, 2018. https://doi.org/10.1007/978-3-031-29642-0

52. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, *arXiv:1412.6980*, 2014. https://doi.org/10.48550/arXiv.1412.6980

53. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al., TensorFlow: A system for large-scale machine learning, In: *Proceedings of the 12th USENIX symposium on operating systems design and implementation (OSDI '16)*, 2016, 265–283.

AIMS Press