Solution of Biharmonic Equation in Complicated Geometries With Physics Informed Extreme Learning Machine

Vikas Dwivedi[1]
Heat Transfer and Thermal Power Laboratory,
Department of Mechanical Engineering,
Indian Institute of Technology Madras,
Chennai 600036, Tamil Nadu, India
e-mail: me15d080@smail.iitm.ac.in

Balaji Srinivasan
Associate Professor
Heat Transfer and Thermal Power Laboratory,
Department of Mechanical Engineering,
Indian Institute of Technology Madras,
Chennai 600036, Tamil Nadu, India
e-mail: sbalaji@iitm.ac.in

Recently, physics informed neural networks (PINNs) have produced excellent results in solving a series of linear and nonlinear partial differential equations (PDEs) without using any prior data. However, due to slow training speed, PINNs are not directly competitive with existing numerical methods. To overcome this issue, the authors developed Physics Informed Extreme Learning Machine (PIELM), a rapid version of PINN, and tested it on a range of linear PDEs of first and second order. In this paper, we evaluate the effectiveness of PIELM on higher-order PDEs with practical engineering applications. Specifically, we demonstrate the efficacy of PIELM to the biharmonic equation. Biharmonic equations have numerous applications in solid and fluid mechanics, but they are hard to solve due to the presence of fourth-order derivative terms, especially in complicated geometries. Our numerical experiments show that PIELM is much faster than the original PINN on both regular and irregular domains. On irregular domains, it also offers an excellent alternative to traditional methods due to its meshless nature. [DOI: 10.1115/1.4046892]

Keywords: complicated geometries, biharmonic equation, physics informed neural networks, Physics Informed Extreme Learning Machine

1 Introduction

In this paper, we consider a two-dimensional (2D) biharmonic equation

$$\nabla^4 u(x, y) = \frac{\partial^4 u}{\partial x^4} + 2\frac{\partial^4 u}{\partial x^2 \partial y^2} + \frac{\partial^4 u}{\partial y^4} = R(x, y), \quad (x, y) \in \Omega \qquad (1)$$

and Dirichlet boundary conditions (BCs)

$$u = G_1(x, y), \quad \frac{\partial u}{\partial n} = G_2(x, y), \quad (x, y) \in \partial\Omega \qquad (2)$$

Here, Ω is a bounded open set in $\mathfrak{R}^2$ with a smooth boundary $\partial\Omega$, $\partial u/\partial n$ is the normal derivative of u on $\partial\Omega$, and $\mathbf{n}$ is the unit vector pointing outward (see Fig. 1).

Many problems in solid and fluid mechanics may be reduced to solving the biharmonic equation. In fluid mechanics, the solution $u(x, y)$ of Eq. (1) can be used to model the slow 2D (two-dimensional) Newtonian viscous fluid flows [1,2]. The study of slow viscous fluid flow (especially in complex geometries) is important for both scientific and industrial applications. For example, the biharmonic equation is used to model the geophysical flow in the Earth's mantle [3], roll coating [4], and polymer processing [5]. The biharmonic equation is also used as a benchmark problem for testing new computational fluid dynamics (CFD) algorithms [6]. In linear elasticity, $u(x, y)$ can represent airy stress function in 2D elastostatic problems [7]. In the theory of thin plates, Eq. (1) can be used to represent a "clamped plate," where R is the external load and the solution $u(x, y)$ is the vertical displacement.

Conventional numerical methods like finite element method (FEM) and finite difference method (FDM) can be efficiently used to solve the biharmonic equation for regular domains. However, if the computational domain has an irregular shape, then either it is impossible to create the mesh or the computational cost of mesh generation is prohibitively high [8]. Under these circumstances, physics informed neural networks (PINNs) [9] offer a much more flexible framework compared to conventional numerical methods. In a typical PINN, a deep neural network approximates the solution of partial differential equation (PDE). No specific training dataset is required. The collocation and boundary points, which are randomly distributed in the computational domain proxy as the training dataset and penalties, are imposed for not satisfying the governing PDE and the BCs at these points. The mean square of these penalties, i.e., PDE and BCs residuals, acts as a physics-based cost function, and standard optimization techniques (like stochastic gradient descent) are used to minimize it. The idea of treating the problem of solving PDEs as an optimization problem with a physics-based cost function is the centerpiece of all PINN (and PINN related) algorithms. Being meshless, these methods can easily handle the complex geometries. For example, Berg and Nyström [8] have recently developed a PINN-based deep unified ANN to solve the 2D Laplace equation on very complicated computational domains.

The readers may refer to the seminal work of Lagaris et al. [10] in which they used neural networks to solve initial and boundary value problems. Some of the related works in this direction include the

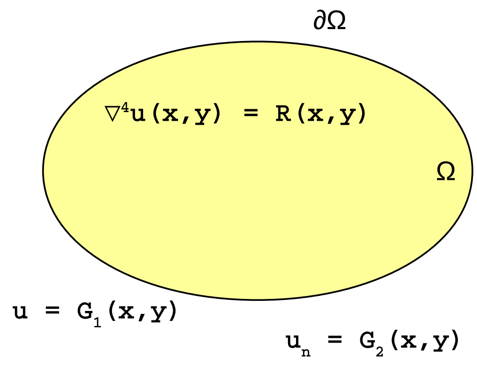


Fig. 1 A diagram of the problem: a biharmonic equation on an irregular domain

[1]Corresponding author.

Manuscript received November 5, 2019; final manuscript received April 4, 2020; published online May 26, 2020. Assoc. Editor: Conrad S. Tucker.

discrete Galerkin method (DGM) [11] to solve high-dimensional PDEs with applications in finance, application of PINNs for hemodynamic applications [12]. Recently, Nabian and Meidani [13] have used physics-driven regularization for enhanced engineering design.

Despite various advantages of using the deep neural networks, there are many unresolved issues associated with PINNs, some of which are as follows:

(1) **Speed**: PINNs use gradient descent optimization, which is prohibitively slow for practical problems.
(2) **Optimization errors**: PINNs are prone to vanishing gradient problems (difficulty in training of network when the gradient of the loss function approaches zero) for deep networks [8]. There is also a danger that a solution may get stuck in a local minimum.
(3) **Hand-tuning**: The learning process of PINNs is hand-tuned. For example, there is no way to determine how much data or which architecture is good enough.

To overcome these problems, we developed a fast version of PINN called Physics Informed Extreme Learning Machine (PIELM) [14]. The quickness of PIELM is the result of the combination of PINN with a rapid machine learning technique called Extreme Learning Machine (ELM). References [15,16] describe the theoretical details and universal approximation properties of ELM, respectively. Previously, ELMs have been used in approximating functions [17] and solving ordinary differential equations [18] and stationary PDEs [19] using Legendre and Bernstein polynomial basis functions, respectively.

Unlike PINNs that have multiple hidden layers, ELMs (and thus PIELM) have a simple neural network architecture consisting of a single hidden layer. Conceptually, ELM is inspired by the fact that the human brain is an intelligent system that can handle diverse tasks without human intervention. It means that there are some parts of the brain where neuron configurations do not depend on the external environment. Import of this neurological argument to the PINN framework allows us to keep some parameters of the neural network to be tuning free. In the case of PIELM, the weights of the hidden layer are randomly initialized and are kept tuning free. As a result, the PIELM approximation remains linear in the output layer weights. Due to it, the output layer parameters can be determined in a direct (i.e., not iterative) fashion using the least-squares method. This skipping of the optimization using the backpropagation algorithm is the main reason behind the *extreme* learning of PIELM.

Similar to its parent algorithms, PIELM is both data efficient and fast compared to the iterative gradient descent-based methods. It is free from problems typically faced by deep neural networks like vanishing gradients and the danger of getting stuck in a local minimum. The main contributions of this paper are as follows:

(1) Formulation of a new machine learning-based algorithm to solve biharmonic equations without using any prior data.
(2) Direct comparison with original PINN to show that PIELM is at least a thousand times faster.
(3) Direct comparison with conventional FDM-based solver to show that PIELM is slower than FDM for regular domains, but it is an excellent alternative to traditional mesh methods for complicated geometries.

Although we have shown the results for the 2D biharmonic equation, it is straightforward to extend the method to three dimensions. For example, the study of blood flow in complicated circulatory systems or heat transfer characteristics of complicated shaped objects of engineering importance like gyroid [20].

The organization of this paper is as follows. Section 2 presents a summary of traditional mesh-based methods and PINN. Section 3 presents the PIELM in detail. In Sec. 4, we validate our PIELM using standard methods like the method of manufactured solutions and test it on standard benchmark CFD problems like the lid-driven cavity flow. We also compare the performance of PIELM with original PINN and a fast FDM-based solver. In Sec. 5, we summarize the computational aspects of PIELM, PINN, and conventional mesh-based methods. Finally, we conclude this study in Sec. 6.

2 Brief Review of Mesh-Based Numerical Methods and PINN

2.1 Mesh-Based Methods. Three standard mesh-based approaches to solve PDEs are FEM, FDM, and finite volume methods (FVMs). The full details of these methods can be found in Refs. [21–23]. Given a governing PDE, we reformulate it depending on the method of our choice, e.g., integral conservation form (in case of FVM) or weighted integral form (in case of FEM). Although different in their mathematical formulations, the solution procedure of these mesh-based methods is similar to each other. The procedure for solving PDEs using mesh-based methods can be summarized as follows:

(1) Discretization of computational domain into multiple subdomains.
(2) Creation of grid (FDM) or mesh (FEM or FVM). This step is tough for complicated computational domains (see Sec. 4.3).
(3) Numerical approximation of derivative terms. This step introduces discretization errors in the solution.
(4) Iterative solution of discretized equations.

2.2 PINN. The principle of PINN has been explained in Sec. 1. For the mathematical formulation, readers may refer to the original paper by Raissi et al. [9]. The procedure for solving PDEs using PINN can be summarized as follows:

(1) Identify the PDE to be solved along with the initial and boundary conditions.
(2) Decide the architecture of PINN, for example, number of layers, activation function, and so on.
(3) Approximate the correct solution with PINN. The approximate solution is expressed in terms of tunable parameters.
(4) Find expressions for the PDE, BCs, and initial condition (IC) in terms of PINN and its derivatives. The derivatives are typically found using automatic differentiation [24].
(5) Define a loss function that penalizes for error in PDE, BCs, and IC. For example, mean square of error residuals of PDE, BCs, and IC.
(6) Minimize the loss with gradient-based algorithms and store the value of parameters obtained at this stage. The learning of PINN implies the tuning of the parameters so as to minimize the loss function. It will be shown in Sec. 4.1 that iterative learning of PINN is very slow compared to PIELM.

3 Proposed PIELM

Figure 2 shows the architecture of the PIELM. The number of neurons in the hidden layers is N^*. If we define $\boldsymbol{\chi} = [x, y, 1]^T$, $\mathbf{m} = [m_1, m_2, \ldots, m_{N^*}]^T$, $\mathbf{n} = [n_1, n_2, \ldots, n_{N^*}]^T$, $\mathbf{b} = [b_1, b_2, \ldots, b_{N^*}]^T$, and $\mathbf{c} = [c_1, c_2, \ldots, c_{N^*}]^T$, then the output of the kth hidden neuron is

$$h_k = \varphi(z_k)$$

where $z_k = [m_k, n_k, b_k]\boldsymbol{\chi}$ and $\varphi = \tanh$ is the nonlinear activation function. As per original ELM approach, $\mathbf{m}$, $\mathbf{n}$, and $\mathbf{b}$ are randomly assigned nontrainable parameters, and we need to calculate only $\mathbf{c}$. The PIELM output is given by

$$f(\boldsymbol{\chi}) = \mathbf{hc} \tag{3}$$

The expressions for $\partial^p f/\partial x^p$ and $\partial^p f/\partial y^p$ are as follows:

$$\frac{\partial^p f_k}{\partial x^p} = m_k^p \frac{\partial^p \varphi}{\partial z^p} \tag{4}$$

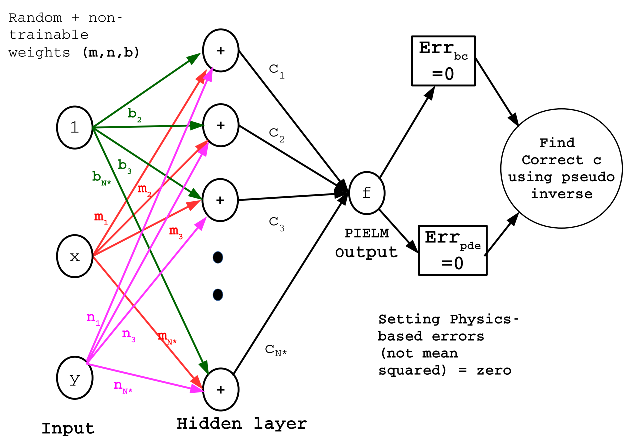


Fig. 2 PIELM architecture

$$\frac{\partial^p f_k}{\partial y^p} = n_k^p \frac{\partial^p \varphi}{\partial z^p} \tag{5}$$

We incorporate the information about the physics of the problem into the ELM by introducing physics-based training errors ξ_R, ξ_{G_1}, and ξ_{G_2}. ξ_R is defined as error in approximating the PDE. ξ_{G_1} and ξ_{G_2} denote the errors in approximating BCs.

$$\xi_R = \frac{\partial^4 \mathbf{f}}{\partial x^4} + 2\frac{\partial^4 \mathbf{f}}{\partial x^2 \partial y^2} + \frac{\partial^4 \mathbf{f}}{\partial y^4} - \mathbf{R}, \quad (x, y) \in \Omega \tag{6}$$

$$\xi_{G_1} = \mathbf{f} - \mathbf{G}_1, \quad (x, y) \in \partial\Omega \tag{7}$$

$$\xi_{G_2} = \frac{\partial \mathbf{f}}{\partial n} - \mathbf{G}_2, \quad (x, y) \in \partial\Omega \tag{8}$$

Now, if an ELM with N^* hidden nodes can solve this PDE with zero error, then it implies that there exist **c**, **m**, **n**, and **b**, such that

$$\xi_R = \mathbf{0} \tag{9}$$

$$\xi_{G_1} = \mathbf{0} \tag{10}$$

$$\xi_{G_2} = \mathbf{0} \tag{11}$$

Equations (9)–(11) lead to a system of linear equations of the type $A\mathbf{c} = \mathbf{b}$, which can be solved using the least-squares method (see Appendix). To find **c**, we use Moore–Penrose generalized inverse as it works well for singular and nonsquare A too.

It completes the mathematical formulation of PIELM. The key steps in its implementation are as follows:

(1) Assign the input layer weights randomly.
(2) Find the expressions for ξ_R, ξ_{G_1}, and ξ_{G_2}.
(3) Assemble the three sets of equations in the form of $A\mathbf{c} = \mathbf{b}$.
(4) The output layer weight vector is given by $pinv(A)\mathbf{b}$, where *pinv* refers to pseudo-inverse.

An important advantage of PIELM is that it provides a criterion to decide the number of neurons in the hidden layer. As mentioned earlier, if the number of hidden layer nodes is equal to the number of training samples, ELM can exactly solve the system of equations. Therefore, instead of choosing the number of hidden units by hit and trial, we initially set the number of hidden units the same as the number of constraints, i.e., $N^* = N_R + N_{G_1} + N_{G_2}$. For a lower number of neurons, the solutions are not stable. For a higher number, the order of accuracy doesn't improve (see Sec. 4.4). Therefore, PIELM reduces, to a certain extent, the arbitrariness of the number of neurons in typical deep PINNs. It is also to be noted that PIELM cannot be used directly on nonlinear problems in the present formulation.

4 Numerical Experiments

This section is divided into three parts. In the first part, we evaluate the performance of PIELM with PINN in solving biharmonic equations on regular and irregular domains. In the second, we solve the "benchmark" lid-driven cavity problem with PIELM and a fast FDM-based solver. Finally, we demonstrate the advantage of PIELM over conventional mesh-based methods by considering a very complicated computational geometry (map of Australia). It is challenging to create a mesh for such geometries by traditional methods. However, we show that PIELM solves such cases seamlessly due to its meshless nature. All the experiments are conducted in MATLAB 2017b environment running in an Intel Core i5 2.20GHz CPU and 8GB RAM Dell laptop.

4.1 Comparison Between Performance of PIELM and the Original PINN.

In this example, we supply a manufactured solution to the biharmonic equation (Eq. (1)) and then calculate the forcing terms and essential boundary conditions using the symbolic computation toolbox of MATLAB. As verification of code is a purely mathematical exercise, the solutions need not be physically realistic. To evaluate the performance of an algorithm, we compute the error in the computational domain, i.e., the difference between the predicted and the exact solution.

We refer to the studies by Adibi and Eshaghi [25] and Chen et al. [26] for the test cases considered in this section. The computational domains for the two test cases are as follows:

$$\partial\Omega_1 : x^2 + y^2 = \frac{1}{4} \tag{12}$$

$$\partial\Omega_2 : \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} \left(\frac{3}{5} + \frac{1}{4}\sin(2\theta)\right)\cos(\theta) \\ \left(\frac{3}{5} + \frac{1}{4}\sin(2\theta)\right)\sin(\theta) \end{Bmatrix}, \quad \theta \in [0, 2\pi] \tag{13}$$

Exact solutions for the corresponding problems are given by

$$u(x, y) = x^2 + y^2 + e^x, \quad (x, y) \in \Omega_1 \tag{14}$$

$$u(x, y) = x^2 + y^2 + e^x \cos(y), \quad (x, y) \in \Omega_2 \tag{15}$$

Figures 3 and 4 show the distribution of data points and PIELM predictions in the computational domain, respectively. The total number of hidden nodes is the same as the number of data points, i.e., 150. As the figure shows, with just 150 training data points, the PIELM predictions are very accurate, and the error is of error 10^{-6} for both the cases. The average computation time of PIELM is of the order 10^{-3} s. Now we test the same problem using the original PINN. The reference code for PINN is available online.[2] For this example, we consider a neural network with two hidden layers with ten neurons each (i.e., a total of 150 unknown parameters). The nonlinear activation used is a hyperbolic tangent, and the optimizer is L-BFGS. This PINN takes almost 8–10 s to reach the same accuracy level for the same sized input dataset. PIELM is at least a thousand times faster than the original PINN.

Please note that although our PIELM weights are trained using just 150 training data points, it predicts the solution for 5000 new data points (different from training data). It shows that PIELM has excellent generalization property, and it doesn't suffer from overfitting. It is a very desirable property because, as the order of PDE increases, the traditional deep neural network requires more layers, which makes them error prone due to the infamous vanishing gradients problem.

[2]https://github.com/maziarraissi/PINNs

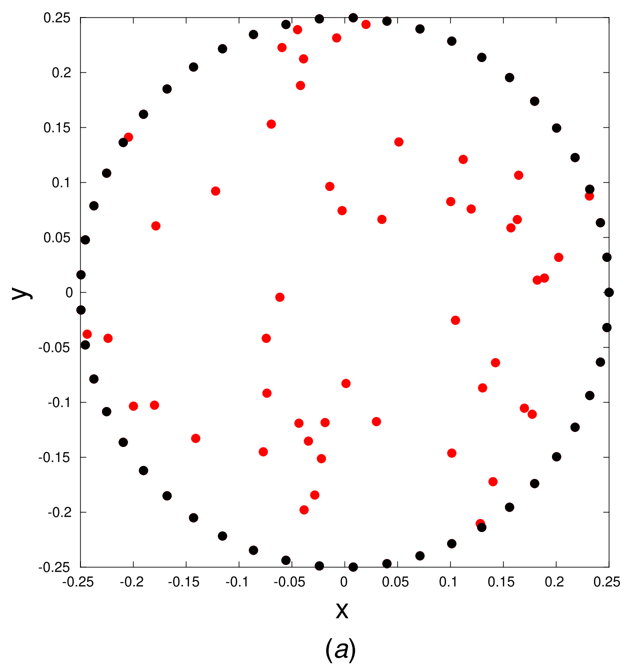
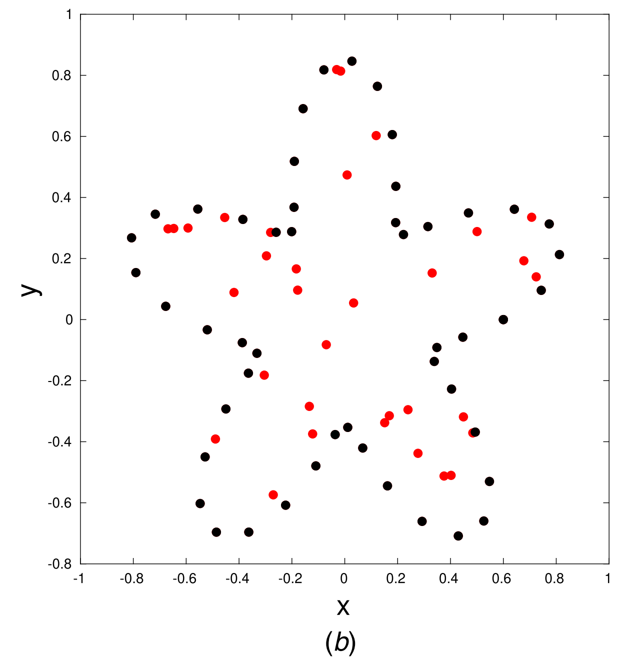



Fig. 3 Randomly distributed data points in the computational domain. Red (lying within the computational boundary) and black points (lying at the computational boundary) correspond to collocation and boundary points, respectively. (*a*) Circular domain and (*b*) irregular domain.

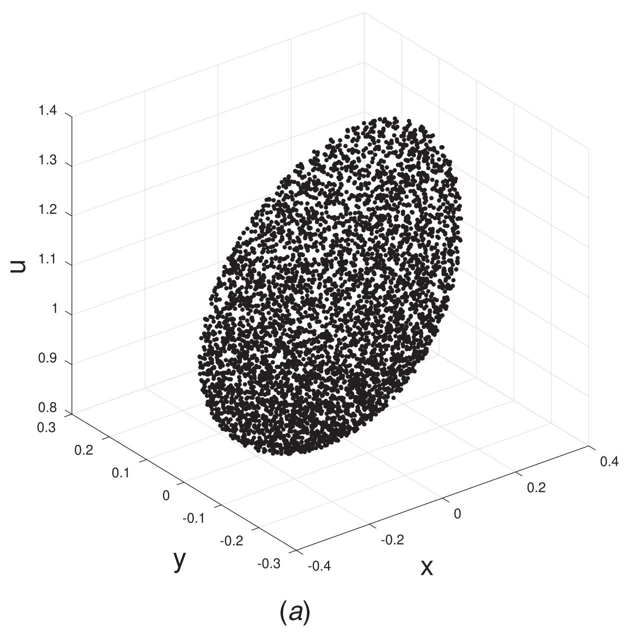
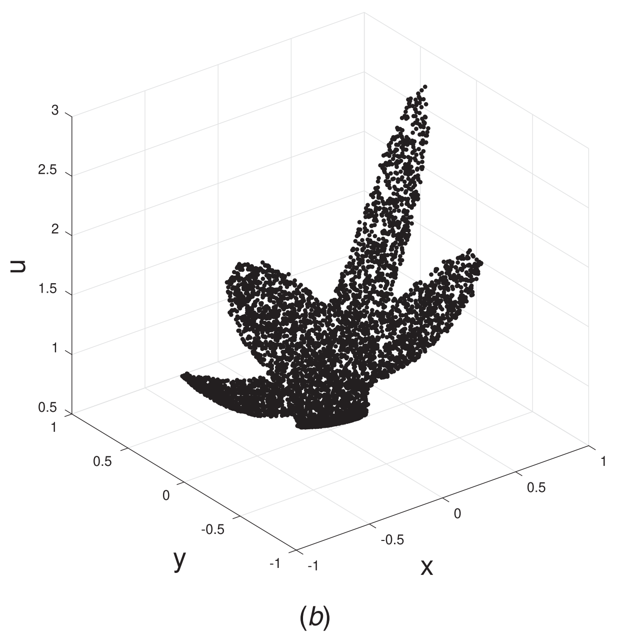



Fig. 4 PIELM solutions for the two cases: (*a*) circular domain and (b) irregular domain

4.2 Comparison Between the Performance of PIELM and a Fast Finite Difference Method-Based Solver in Solving the Lid-Driven CavityProblem.

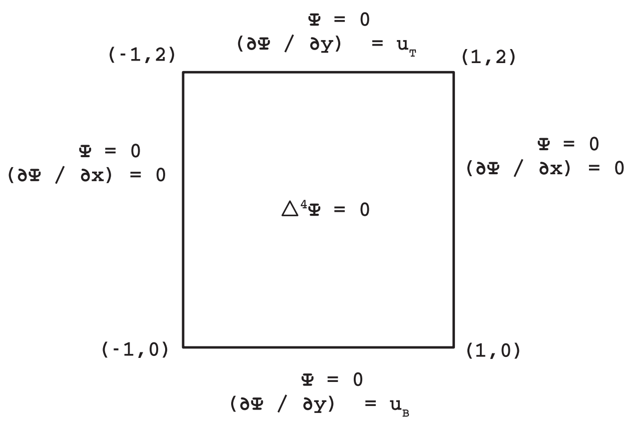
Consider the steady, isothermal, incompressible flow of a Newtonian fluid of constant density ρ and viscosity μ. The governing equations are Navier–Stokes and continuity equations. On scaling the velocity, pressure, and length with U, $\mu U/L$, and L, respectively, the nondimensional form of governing equations is given by

$$\mathrm{Re}(\mathbf{u}.\nabla\mathbf{u}) = -\nabla p + \nabla^2\mathbf{u} \tag{16}$$

where $\mathbf{u}$ is the dimensionless velocity, p is the dimensionless pressure, and $\mathrm{Re} = \rho UL/\mu$ is the Reynolds number measuring the relative importance of inertia to viscous forces. Under Stokes' flow conditions, Re is set to zero. Taking the curl of the remaining terms to eliminate the pressure and introducing the stream function ψ, via the continuity equation, yields the biharmonic equation

$$\nabla^4\psi = 0 \tag{17}$$

In this example, we use our PIELM to solve Eq. (17) for Stokes' flow on a square lid-driven cavity. Mathematically, the flow geometry and the boundary conditions are described in Fig. 5. Please note that, unlike the previous problem, we are not supplying an exact synthetic solution and deriving boundary conditions from it. The selection of continuous functions as exact solutions results in continuous boundary conditions also. In this case, we have discontinuities in the x component of velocity at the lid corners. As we have provided physically realistic boundary conditions, we expect PIELM to reciprocate by producing physically realistic flow structures that are previously reported in the literature[6].



$\Psi = 0$
$(\partial\Psi / \partial y) = u_T$
(-1,2)
(1,2)
$\Psi = 0$
$(\partial\Psi / \partial x) = 0$
$\Psi = 0$
$(\partial\Psi / \partial x) = 0$
$\Delta^4\Psi = 0$
(-1,0)
(1,0)
$\Psi = 0$
$(\partial\Psi / \partial y) = u_B$

Fig. 5 Description of the lid-driven cavity problem

We consider three different cases that give rise to three distinct flow structures as described in Gaskell et al. [6]. In the first case ($u_B = 0$, $u_T = 1$), the flow consists of a large eddy symmetrically situated at the center. In the second case ($u_B = -1$, $u_T = 1$), there is a sizeable circulatory motion containing two eddy centers and a saddle point at the center of the cavity. For the last case ($u_B = 1$, $u_T = 2$), there are two large eddies in the cavity.

The PIELM predictions for these three cases are shown in Fig. 6. As the figure shows, PIELM captures all these structures correctly using just 20×20 uniformly spaced data points. The computation time is nearly 0.5 s.

Among the three cases shown, the first case has received more attention in the literature than the general case in which both the lids move. Therefore, we choose this case to compare the speed of PIELM with the conventional finite difference algorithm described by Pozrikidis [27]. The standard vorticity-stream function approach has been used to solve this problem, and therefore, we are skipping the mathematical formulation here. For the details of the numerical method, the readers may refer to Chapter 13 of the reference. The numerical code is also available online.[3] We found that, for a 40×40 uniformly spaced dataset, the FDM solver converged to the solution in just 1 s while PIELM took almost 10–11 s. Therefore, we conclude that FDM is much faster than PIELM for rectangular geometry.

Figure 7 shows the distribution of stream function along the horizontal and vertical centerlines using FDM and PIELM. Due to discontinuous boundary conditions, the maximum errors are of orders 10^{-3} and 10^{-2} along with the horizontal and vertical directions, respectively. To fix this error, residual adaptive refinement [28] can be used to automatically increase the number of sample points near the region of sharp velocity jump.

4.3 Performance of PIELM in Solving the Biharmonic Equation on a Very Complicated Domain. Even though FDM performed better than PIELM in the previous case, typical mesh-based methods cannot be used when the computational domain is complicated, because, in such cases, mesh generation itself is an arduous task [8]. The calculation of derivatives for calculating fluxes also induces a lot of error in the solution. To elaborate this point, we consider a very complicated geometry, i.e., scaled map of Australia. Figure 8 shows the scaled map of Australia and the distribution of sampling points. We tried to create a mesh for this geometry using the PDE toolbox of MATLAB as well as in COMSOL. Both software failed in creating a mesh for this geometry. On the other hand, PIELM faces no such difficulty in solving this problem.

We repeat the test case of Adibi and Jaafar [25] on this complicated computational domain. For training and testing samples consisting of 711 and 1661 points, respectively, the PIELM solution and error is given in Fig. 9. The maximum error is of the order of 10^{-6}, and the total CPU time is of the order of 10^{-1} s. Therefore, in the case of complicated computational domains, PIELM offers an advantage over conventional mesh-based methods due to its meshless nature.

[3]http://dehesa.freeshell.org/FDLIB

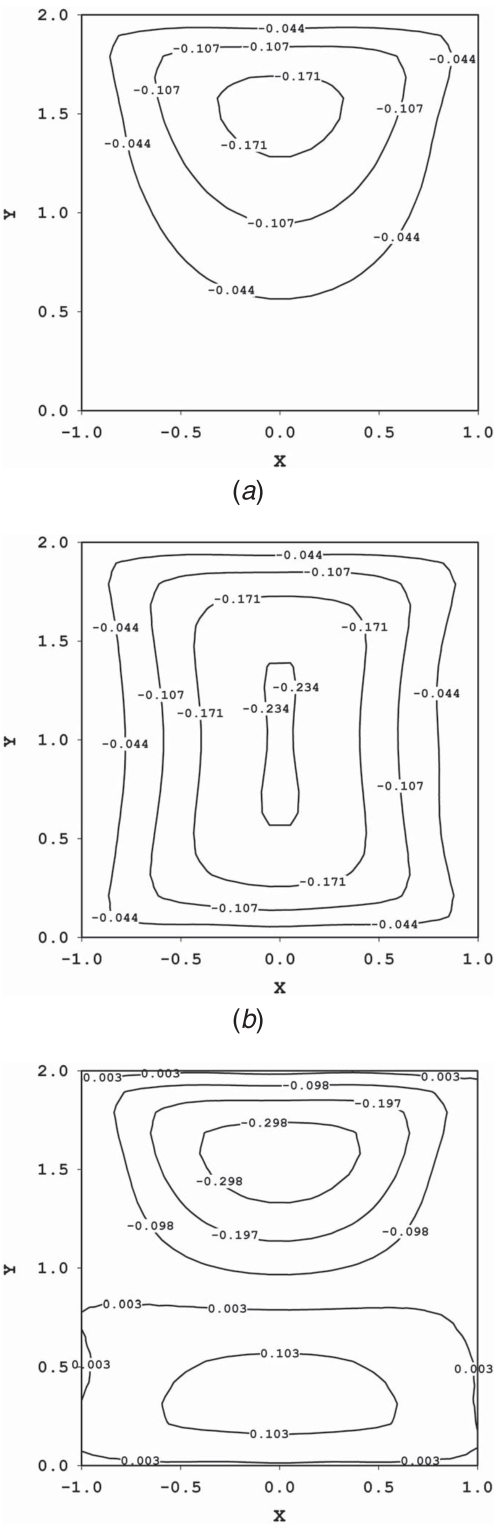
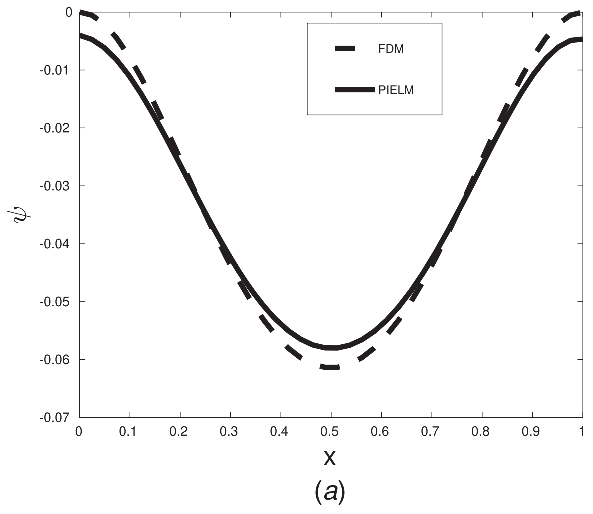


Fig. 6 PIELM predictions of flow structures in different cases of lid-driven cavity: (*a*) case 1: $u_B = 0$, $u_T = 1$; (*b*) case 2: $u_B = -1$, $u_T = 1$; and (c) case 3: $u_B = 1$, $u_T = 2$



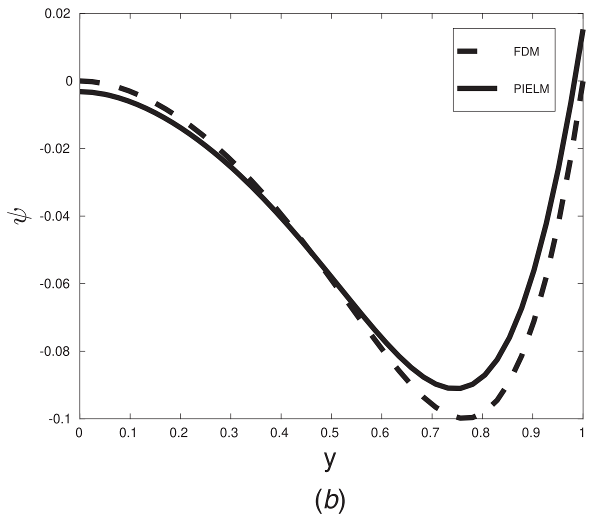
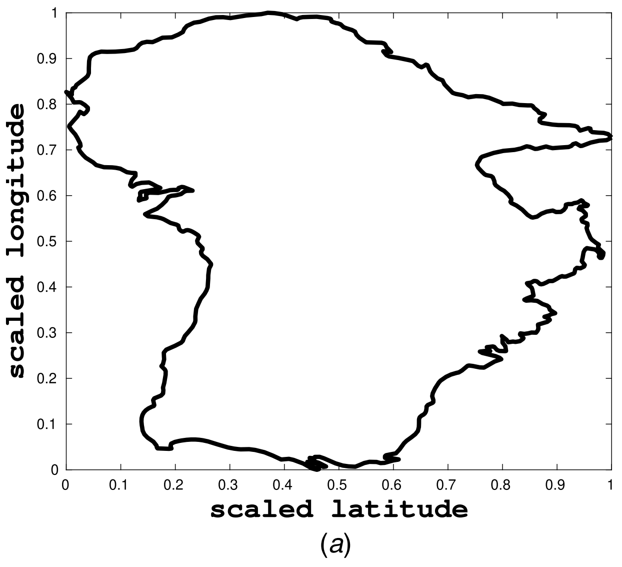


Fig. 7 Stream function distribution along horizontal and vertical centerlines: (a) along the horizontal centerline and (b) along the vertical centerline



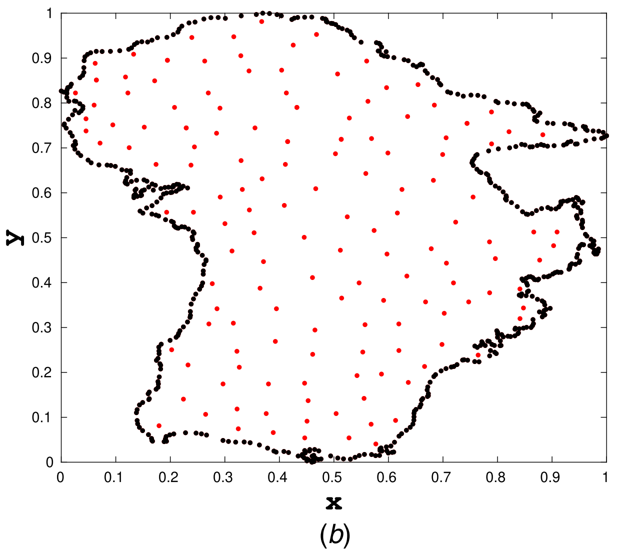


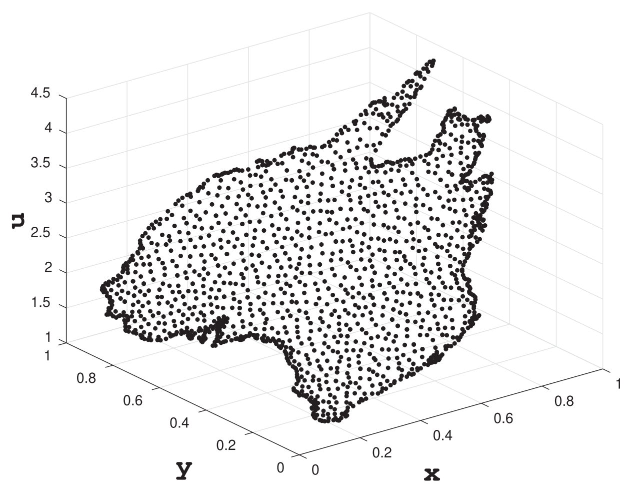
Fig. 8 A complicated geometry that is difficult to mesh: (a) scaled map of Australia and (b) training dataset

4.4 Effect of Number of Hidden Layers. Figure 10 shows the relationship between the generalization performance and the network size of PIELM for the first example discussed in Sec. 4.1. We have just changed the exact solution to $u(x, y) = x^2 + y^2 + e^{3x}$ (sharper gradient), so that the effect of the number of neurons can be clearly seen. As shown in the figure, the generalization performance of PIELM is stable on a wide range of hidden nodes. However, the performance tends to deteriorate for very few neurons. As we discussed earlier, setting the number of hidden units the same as the number of constraints (i.e., $N^* = N_R + N_{G_1} + N_{G_2}$) is a good idea because the performance doesn't improve on increasing the number of hidden layers. This aspect of PIELM reduces the arbitrariness in the selection of neural network architecture. The next logical question that follows is: How many sample points are sufficient? Currently, we try to find the minimum amount of data using hit and trial. We start with a small number and increase the number of sample points gradually to reach a desired level of accuracy.

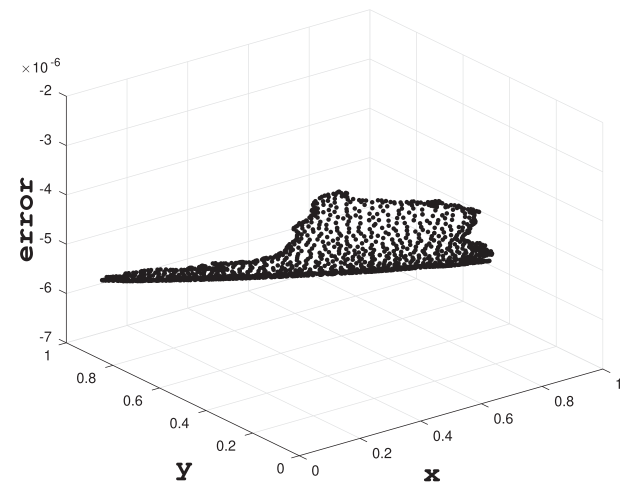
5 Comparison Between Mesh-Based Methods, PINN, and PIELM

In this section, we briefly summarize the key points of similarities and differences between the mesh-based methods, PINN, and PIELM.

(1) **Basis functions.** Piecewise polynomials (linear) are used as basis functions for conventional FEM. However, for PINN and PIELM, the basis functions are deep and shallow neural networks (nonlinear), respectively.
(2) **Unknowns.** Mesh-based methods solve for the quantities at the mesh points. PINN solves for all its weights and biases. PIELM solves for its parameters in the outer layer only.
(3) **Physics awareness.** Conventional methods try to mimic the physics of the problem by designing appropriate discretizations. For example, the central difference scheme is used for diffusion terms as it is a direction independent phenomenon. On the other hand, the upwind differencing scheme is used for advection terms because the direction of flow becomes essential there. In the case of PINN and PIELM, the information about physics is embedded in their physics-based cost functions and errors, respectively.
(4) **Solution methods.** Mesh-based methods iteratively solve a system of linear or nonlinear algebraic equations. PINNs tend to minimize the cost function iteratively using backpropagation. PIELM doesn't use any iterative training. It calculates the weights using simple matrix inversion, assuming that the physics-based errors are zero.



(*a*)



(*b*)

Fig. 9 Performance of PIELM in a complicated computational domain. The number of testing points are significantly larger than the number of training points. The error in the solution is of the order 10^{-6}. (*a*) PIELM solution and (*b*) error.

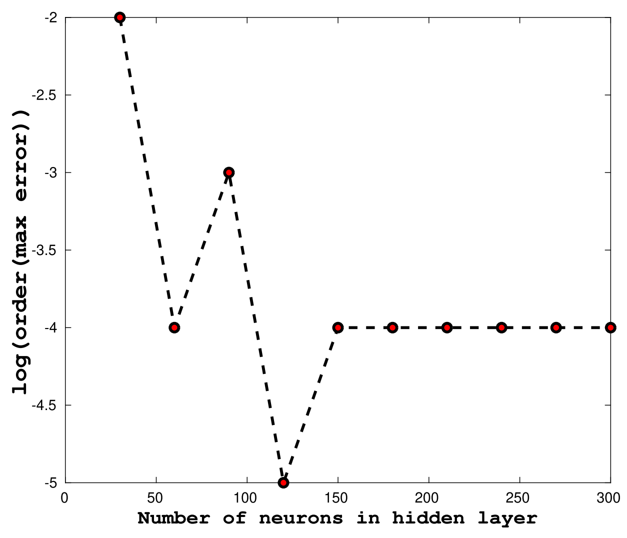


Fig. 10 Generalization performance of PIELM is stable on a wide range of hidden nodes

(5) **Sources of errors.** Typically, conventional methods suffer from issues like discretization errors, numerical instability, generation of complicated mesh, etc. Both PINN and PIELM are free from discretization error and mesh-generation issues due to their PDE embedding and meshless formulation. However, PINN still suffers from issues like vanishing gradients, local minima, etc. PIELM is a single-layer network, so it is free from vanishing gradients problem. As PIELM transforms the PDE into a system of linear equations, the primary source of error is the ill-behaved coefficient matrices.
(6) **Computation on regular domains.** Among the three, mesh-based methods are the fastest, followed by PIELM and PINN.
(7) **Computation on complicated domains.** For complicated domains, PIELM is the fastest among the three.

6 Conclusions

In this paper, we demonstrated the capability of a fast machine learning algorithm called PIELM in solving the biharmonic equation. PIELM is a combination of PINN and ELM that contains the desirable features of both its parent algorithms, namely physics awareness and speed. The main conclusions of this work are as follows:

(1) PIELM rapidly solves high-order linear PDEs without using prior data or parallel computation.
(2) PIELM reduces the arbitrariness in the selection of neural network architecture.
(3) PIELM is almost a thousand times faster than the original PINN and is free from optimization errors typically found in PINNs.
(4) On regular-shaped computational domains, PIELM is slower than mesh-based methods. However, it is readily applicable to very complicated geometries where meshing is computationally taxing.

Overall, PIELM improves the candidature of machine learning methods in solving linear PDEs by increasing their speed and robustness. Extension of this approach to nonlinear problems is the topic of future work.

Appendix: PIELM Equations for Biharmonic Equations

If $\mathbf{x}_f$, $\mathbf{y}_f$ denote collocation points vectors, $\mathbf{x}_{bc}$, $\mathbf{y}_{bc}$ denote boundary points vectors, $\mathbf{I}$ denotes bias vector, φ denotes the activation function, and "$\odot$" refers to Hadamard product, then we define $\boldsymbol{X_f} = [\mathbf{x}_f, \mathbf{y}_f, \mathbf{I}]$, $\boldsymbol{X_{bc}} = [\mathbf{x}_{bc}, \mathbf{y}_{bc}, \mathbf{I}]$, and $\boldsymbol{W} = [\mathbf{m}, \mathbf{n}, \mathbf{b}]$. The PIELM equations corresponding to Eqs. (6)–(8) are as follows:

$$\varphi''''(\boldsymbol{X_f}\boldsymbol{W}^T)\odot\mathbf{c}\odot(\mathbf{m}\odot\mathbf{m}\odot\mathbf{m}\odot\mathbf{m} + 2\mathbf{m}\odot\mathbf{m}\odot\mathbf{n}\odot\mathbf{n} + \mathbf{n}\odot\mathbf{n}\odot\mathbf{n}\odot\mathbf{n}) = R(\mathbf{x}_f, \mathbf{y}_f) \quad (A1)$$

$$\varphi(\boldsymbol{X_{bc}}\boldsymbol{W}^T)\mathbf{c} = G_1(\mathbf{x}_{bc}, \mathbf{y}_{bc}) \quad (A2)$$

$$\varphi'(\boldsymbol{X_{bc}}\boldsymbol{W}^T)\odot\mathbf{c}\odot\mathbf{m} = \frac{\partial}{\partial x}G_2(\mathbf{x}_{bc}, \mathbf{y}_{bc}) \quad (A3)$$

$$\varphi'(\boldsymbol{X_{bc}}\boldsymbol{W}^T)\odot\mathbf{c}\odot\mathbf{n} = \frac{\partial}{\partial y}G_2(\mathbf{x}_{bc}, \mathbf{y}_{bc}) \quad (A4)$$

These equations are combined to form a system of linear equations of the type $A\mathbf{c} = \mathbf{b}$, which can be solved using the least-squares method. To find $\mathbf{c}$, we use Moore–Penrose generalized inverse as it works well for singular and nonsquare $\boldsymbol{A}$ too.

References

[1] Weinan, E., and Liu, J.-G., 1996, "Vorticity Boundary Condition and Related Issues for Finite Difference Schemes," J. Comput. Phys., **124**(2), pp. 368–382.

[2] Kupferman, R., 2001, "A Central-Difference Scheme for a Pure Stream Function Formulation of Incompressible Viscous Flow," SIAM J. Sci. Comput., **23**(1), pp. 1–18.

[3] Harper, J., and Wake, G., 1983, "Stokes Flow Between Parallel Plates Due to a Trausversely Moving End Wall," IMA J. Appl. Math., **30**(2), pp. 141–149.

[4] Gaskell, P., Savage, M., Summers, J., and Thompson, H., 1995, "Modelling and Analysis of Meniscus Roll Coating," J. Fluid Mech., **298**, pp. 113–137.

[5] Canedo, E. L., and Denson, C. D., 1989, "Flow in Driven Cavities With a Free Surface," AIChE J., **35**(1), pp. 129–138.

[6] Gaskell, P., Savage, M., Summers, J., and Thompson, H., 1998, "Stokes Flow in Closed, Rectangular Domains," Appl. Math. Model., **22**(9), pp. 727–743.

[7] Timoshenko, S., and Goodier, J., 1970, *Theory of Elasticity*, Mcgraw-Hill Company Inc., New York.

[8] Berg, J., and Nyström, K., 2018, "A Unified Deep Artificial Neural Network Approach to Partial Differential Equations in Complex Geometries," Neurocomputing, **317**, pp. 28–41.

[9] Raissi, M., Perdikaris, P., and Karniadakis, G. E., 2019, "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations," J. Comput. Phys., **378**, pp. 686–707.

[10] Lagaris, I. E., Likas, A., and Fotiadis, D. I., 1998, "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations," IEEE Trans. Neural Netw., **9**(5), pp. 987–1000.

[11] Sirignano, J., and Spiliopoulos, K., 2018, "Dgm: A Deep Learning Algorithm for Solving Partial Differential Equations," J. Comput. Phys., **375**, pp. 1339–1364.

[12] Sun, L., Gao, H., Pan, S., and Wang, J.-X., 2019, "Surrogate Modeling for Fluid Flows Based on Physics-Constrained Deep Learning Without Simulation Data," arXiv:1906.02382.

[13] Nabian, M. A., and Meidani, H., 2020, "Physics-Driven Regularization of Deep Neural Networks for Enhanced Engineering Design and Analysis," ASME J. Comput. Inf. Sci. Eng., **20**(1), p. 011006.

[14] Dwivedi, V., and Srinivasan, B., 2019, "Physics Informed Extreme Learning Machine (PIELM)—A Rapid Method for the Numerical Solution of Partial Differential Equations," Neurocomputing.

[15] Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K., 2006, "Extreme Learning Machine: Theory and Applications," Neurocomputing, **70**(1–3), pp. 489–501.

[16] Huang, G.-B., Chen, L., and Siew, C. K., 2006, "Universal Approximation Using Incremental Constructive Feedforward Networks With Random Hidden Nodes," IEEE Trans. Neural Netw., **17**(4), pp. 879–892.

[17] Balasundaram, S., and Kapil, G., 2011, "Application of Error Minimized Extreme Learning Machine for Simultaneous Learning of a Function and Its Derivatives," Neurocomputing, **74**(16), pp. 2511–2519.

[18] Yang, Y., Hou, M., and Luo, J., 2018, "A Novel Improved Extreme Learning Machine Algorithm in Solving Ordinary Differential Equations by Legendre Neural Network Methods," Adv. Differ. Equ., **2018**(1), p. 469.

[19] Sun, H., Hou, M., Yang, Y., Zhang, T., Weng, F., and Han, F., 2019, "Solving Partial Differential Equation Based on Bernstein Neural Network and Extreme Learning Machine Algorithm," Neural Process. Lett., **50**, pp. 1–20.

[20] Vlahinos, A. and Scully, S. J., 2019, "Protection Device That Promotes Air Flow for Heat Transfer," US Patent Application 16/297,096.

[21] Rao, S. S., 2017, *The Finite Element Method in Engineering*, Butterworth-Heinemann, Oxford, UK.

[22] LeVeque, R. J., 2007, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*, Vol. 98, SIAM, Philadelphia.

[23] Versteeg, H. K., and Malalasekera, W., 2007, *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*, Pearson Education, Essex, UK.

[24] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M., 2017, "Automatic Differentiation in Machine Learning: A Survey," J. Mach. Learn. Res., **18**(1), pp. 5595–5637.

[25] Adibi, H., and Es'haghi, J., 2007, "Numerical Solution for Biharmonic Equation Using Multilevel Radial Basis Functions and Domain Decomposition Methods," Appl. Math. Comput., **186**(1), pp. 246–255.

[26] Chen, G., Li, Z., and Lin, P., 2008, "A Fast Finite Difference Method for Biharmonic Equations on Irregular Domains and Its Application to an Incompressible Stokes Flow," Adv. Comput. Math., **29**(2), pp. 113–133.

[27] Pozrikidis, C., 2011, *Introduction to Theoretical and Computational Fluid Dynamics*, Oxford University Press, Oxford, UK.

[28] Lu, L., Meng, X., Mao, Z., and Karniadakis, G. E., 2019, "Deepxde: A Deep Learning Library for Solving Differential Equations," arXiv:1907.04502.