Proceedings of the 4th International Conference on Computing Innovation and Applied Physics
DOI: 10.54254/2753-8818/87/2025.20346

Solving Differential Equations with Physics-Informed Neural Networks

Chenghao Dong[1,a,*]

[1]*Mathematical Institute, University of Oxford, Oxford OX2 6GG, UK*
a. chenghao.dong@kellogg.ox.ac.uk
**corresponding author*

Abstract: Solving differential equations is an extensive topic in various fields, such as fluid mechanics and mathematical finance. The recent resurgence in deep neural networks has opened up a brand new track for numerically solving these equations, with the potential to better deal with nonlinear problems and overcome the curse of dimensionality. The Physics-Informed Neural Network (PINN) is one of the fundamental attempts to solve differential equations using deep learning techniques. This paper aims to briefly review the application of PINNs and their variants in solving differential equations through a few simple examples, and to provide practitioners interested in this direction with a quick introduction to the relevant topic.

Keywords: neural networks, PINNs, differential equations, Fourier features.

1. Introduction

Differential equations are equations involving unknown functions of one or more independent variables and their derivatives. Solving these equations is an extensive topic in various fields such as fluid mechanics and mathematical finance, with new applications continually emerging. Since most differential equations cannot be solved analytically, the advent of computers in the mid-1900s eventually led to the development of numerical methods for solving complex equations, such as nonlinear ones or those defined over intricate geometries. Despite the popularity and power of traditional numerical methods involving finite differences and finite elements, limitations of these methods still exist, including having difficulty in handling nonlinear problems and obtaining fast solutions that are precise enough for high-dimensional problems.

The recent resurgence in deep neural networks has opened up a brand new track for numerically solving differential equations, especially under circumstances when traditional methods are prone to failure. Specifically, these methods involving neural networks particularly excel in handling nonlinear equations [1] as well as high-dimensional problems [2], and can adapt to complex geometries provided suitable sampling methods are developed [3].

This report will mainly focus on the implementation of solving differential equations with physics-informed neural networks (PINNs). In Section 2, we will introduce some preliminary knowledge relevant to the paper. In Section 3, we will apply the method to solving onevariable ordinary differential equations (ODEs) and discuss the limitations of the approach when solutions with high frequency get involved. The example of a 2D elliptic partial differential equation (PDE) will be displayed in Section 4 together with an extension of the method for solving 3D wave equations.

DOI: 10.54254/2753-8818/87/2025.20346

Finally, Section 5 will provide a simple summary of our work with some possible directions for improvements of the method. All codes in this paper are available through the link given in the Appendix.

2. PINNs: Physics-Informed Neural Networks

The Physics-Informed Neural Networks (PINNs) provide a general unsupervised framework for solving differential equations with deep neural networks. Over the recent decades, the method has been widely studied and plays a significant role in solving inverse problems [4] and equation discovery [5]. Given a differential equation with the initial and boundary conditions that the solution satisfies

$$\mathcal{D}[u(\mathbf{x})] = f(\mathbf{x}) \quad \text{for} \quad \mathbf{x} \in \Omega \subset \mathbb{R}^d, \tag{1}$$

$$\mathcal{B}_k[u(\mathbf{x})] = g_k(\mathbf{x}) \quad \text{for} \quad \mathbf{x} \in \Gamma_k \subset \partial\Omega, \tag{2}$$

where $\mathcal{D}$ and $\mathcal{B}_k$ are differential operators and $u(\mathbf{x})$ is the solution of the equation. PINNs aim to train a neural network, i.e., the output of a multilayer perceptron [6], $\varphi_\theta(\mathbf{x})$, to approximate the solution $u(\mathbf{x})$ by minimizing the loss function $\mathcal{L}(\theta)$ for a batch of points $\{\mathbf{x}_i\}_{i=1}^{N_p} \subset \Omega$ and $\{\mathbf{x}_{k,j}\}_{j=1}^{N_{b,k}} \subset \Gamma_k$,

$$\min_{\theta} \mathcal{L}(\theta) = \mathcal{L}_p(\theta) + \mathcal{L}_b(\theta), \tag{3}$$

$$\mathcal{L}_p(\theta) = \frac{1}{N_p}\sum_{i=1}^{N_p} \| \mathcal{D}[\varphi_\theta(\mathbf{x}_i)] - f(\mathbf{x}_i) \|^2 \tag{4}$$

$$\mathcal{L}_b(\theta) = \sum_k \frac{\lambda_k}{N_{b,k}} \sum_{j=1}^{N_{b,k}} \left\| \mathcal{B}\left[\varphi_\theta(\mathbf{x}_{k,j})\right] - g_k(\mathbf{x}_{k,j}) \right\|^2 \tag{5}$$

where $\lambda_k > 0$ are pre-specified parameters; $\mathcal{L}_p(\theta)$ and $\mathcal{L}_b(\theta)$ are referred to as the **physics loss** and the **boundary loss** respectively.

To avoid the problem of vanishing gradients, the activation function for each hidden neuron in PINNs should be non-linear and infinitely differentiable. Therefore, the **Tanh** activation will be chosen for all neurons in hidden layers throughout the experiments in this paper. The derivatives involved in the loss (4) and (5) can be easily obtained using modern learning frameworks such as PyTorch and TensorFlow with automatic differentiation [7, 8].

There are mainly two ways to impose the initial and boundary conditions on the neural network output φ_θ. One way is to apply the general formulation (3) and increase the value of λ_k, which leads to PINNs with **soft** conditions. Another choice is to use the neural network as a part of the solution ansatz so that the network's output will always satisfy the required boundary (resp., initial) conditions. The latter method eventually results in PINNs with **hard** conditions.

Once a hard constraint is asserted on the network's output, the boundary loss $\mathcal{L}_b(\theta)$ will no longer be needed as its contribution to the total loss $\mathcal{L}(\theta)$ will always be zero. Therefore, the problem will become fully unsupervised and the loss that the network aiming to minimize will be reduced to

$$\min_{\theta} \mathcal{L}(\theta) = \mathcal{L}_p(\theta) \tag{6}$$

if we choose to impose the initial and boundary conditions in a hard manner.

DOI: 10.54254/2753-8818/87/2025.20346

3. Solving Ordinary Differential Equations

3.1. Examples with Dirichlet Boundary Conditions

In this section, we will consider solving the following second-order ODE with Dirichlet boundary conditions (which is also referred to as the **type 1 condition** throughout this paper for clarity)

$$y''(x) = f(x, y, y') \quad \text{for} \quad x \in [a, b], \tag{7}$$

$$y(a) = y_a \quad \text{and} \quad y(b) = y_b. \tag{8}$$

PINNs with hard boundary conditions will be applied to obtain the solution of the equation. Let $\hat{\varphi}_\theta(x)$ be the original output of the network. The modified output satisfying the Dirichlet boundary conditions is given as

$$\varphi_\theta(x) = \hat{\varphi}_\theta(x) + \frac{b-x}{b-a} \cdot [\, y_a - \hat{\varphi}_\theta(a) \,] + \frac{x-a}{b-a} \cdot [\, y_b - \hat{\varphi}_\theta(b) \,], \tag{9}$$

with a reduced loss function defined over a series of sampled points $\{x_i\}_{i=1}^{N_p} \subset [a, b]$ given as

$$\mathcal{L}(\theta) = \frac{1}{N_p} \sum_{i=1}^{N_p} \left\| \varphi_\theta''(x_i) - f\left(x_i, \varphi_\theta(x_i), \varphi_\theta'(x_i)\right) \right\|^2. \tag{10}$$

The test examples we used to illustrate the method in Section 3.4 are listed in Table 1 for $a = 0$ and $b = 1$.

Table 1: Known solutions for Dirichlet boundary condition examples.

Index	Forcing Function	True Solution	Target Interval
i	$f(x, y, y')$	y_{true}	(a, b)
1	$-y$	$\sin(x)$	(0, 1)
2	$y^2 - [1 + x(1-x)]^2 - 2$	$1 + x(1-x)$	(0, 1)
3	y	e^x	(0, 1)
4	$e^x[(1 - 16\pi^2)\sin(4\pi x) + 8\pi\cos(4\pi x)]$	$e^x \sin(4\pi x)$	(0, 1)

3.2. Examples with Other Boundary Conditions

The method we used in the previous section for equations with type 1 (Dirichlet) conditions can also be easily generalized to those with other boundary conditions. For example, it can be applied to the same equation with the following **type 2** condition

$$y(a) = y_a \; and \; y'(b) = y_b \tag{11}$$

with ansatz

$$\varphi_\theta(x) = \hat{\varphi}_\theta(x) + y_a - \hat{\varphi}_\theta(a) + (x - a) \cdot [\, y_b - \hat{\varphi}'_\theta(b) \,], \tag{12}$$

and **type 3** condition

$$y(a) + y'(a) = y_a \; and \; y'(b) = y_b, \tag{13}$$

with ansatz

$$\varphi_\theta(x) = \hat{\varphi}_\theta(x) + y_a - [\, \hat{\varphi}_\theta(a) + \hat{\varphi}'_\theta(a) \,] + (x - a - 1) \cdot [\, y_b - \hat{\varphi}'_\theta(b) \,]. \tag{14}$$

The loss functions of these formulations remain the same as the one presented in the formula(10). Test examples used for these examples are summarized in Table 2.

DOI: 10.54254/2753-8818/87/2025.20346

Table 2: Known solutions for examples involving conditions of type 2 and 3.

Index	Forcing Function	True Solution	Target Interval
i	$f(x, y, y')$	y_{true}	(a, b)
5	$-y[y^2 + (y')^2]$	$\cos(x)$	$(0, 1)$
6	$-2xy' - 2y$	$2\exp(-x\text{^}2)$	$(0, 1)$
7	$1 - (y')^2$	$\ln[\cosh(x)]$	$(0, 1)$
8	$-y$	$\sin(x)$	$(0, 1)$

3.3. Example with ODE Systems

Finally, we generalize the method to solve systems of second-order ODEs. Consider the equation (7) and (8) once again with type 1 Dirichlet boundary conditions for some $y, f \in \mathbb{R}^d$, $d \geq 2$. The solution ansatz and the loss function are the same as the one presented in formulas (9) and (10) in vectorized form. The only change we need to specify is that the output layer of the network now consists of d neurons.

For the numerical experiment, we consider the following system of equations

$$u''(x) = x(u - 1) - v - (0.5\pi)^2 \sin(0.5\pi x) - 2, \tag{15}$$

$$v''(x) = v - x^2(1 - x) - 6x + 2, \tag{16}$$

for $x \in [0,1]$ with $y(x) = [u(x), v(x)]^{\mathrm{T}}$ under the Dirichlet boundary condition. The solution of the system is now given by $u(x) = x(1\text{-}x) + \sin(0.5\pi x)$ and $v(x) = x^2(1 - x)$.

3.4. Numerical Results

Solutions of the sample equations listed in Table 1, Table 2 and equations (15) – (16) are obtained by training a simple 3-layer neural network with only one hidden layer composed of 50 neurons using the Adam optimizer with a batch size of 32 [9]. For each model, only 1000 points (2000 points for solving the ODE system) are sampled from the interval.

For the convenience of comparing the convergence of solving different equations, the L_2 losses $L_2(\theta)$ are recorded instead of the physics loss $\mathcal{L}_p$, defined as

$$[L_2(\theta)]^2 = \int_{\Omega} \|\varphi_\theta(\mathbf{x}) - y_{\text{true}}(\mathbf{x})\|^2 \, d\mathbf{x} \tag{17}$$

The final results are displayed in the above Figure 1. Note that for most of the examples where the true solutions are monotonic functions over the target interval, the method performs a rapid convergence and achieves an accuracy below 10^{-2} within only 100 training epochs. However, for a solution with a relatively high latent frequency, for instance, y_4, the method tends to converge in a much slower way.

DOI: 10.54254/2753-8818/87/2025.20346

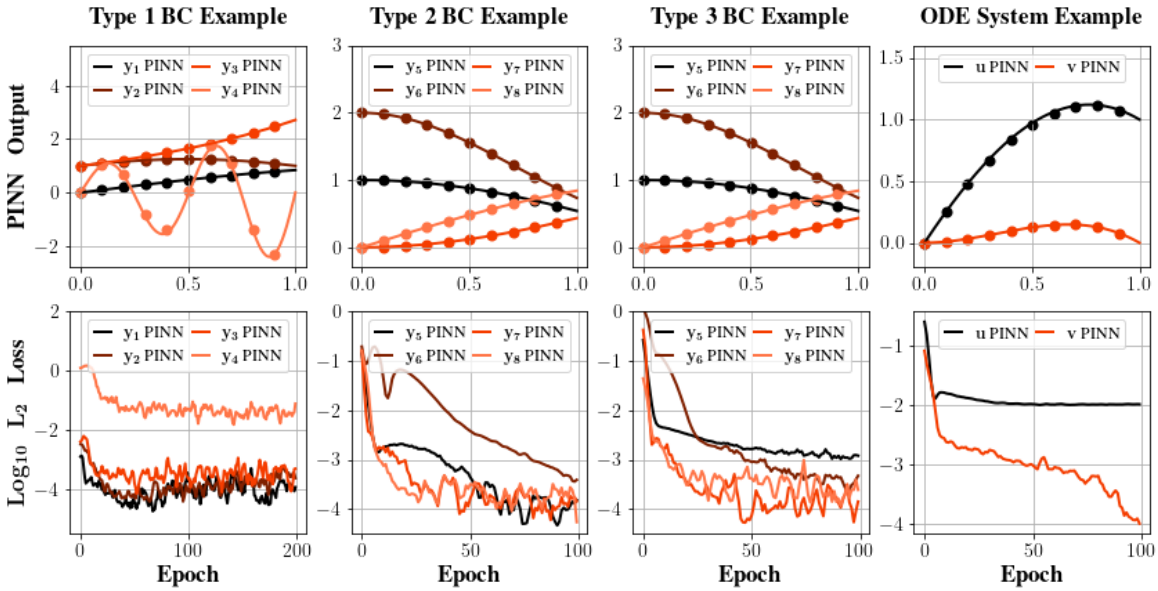

Figure 1: Solutions of Section 3 sample equations generated by PINNs with dots in figures representing the exact value.

3.5. Scaling to Higher Frequencies

In previous examples, we notice that the PINNs tend to converge faster for functions with lower frequencies. Indeed, literature suggests neural networks prioritize learning lower-frequency functions [10]. Furthermore, the capability of these networks to fit high-frequency functions is also bounded by the number of neurons and trainable parameters.

In fact, once the solution involves high-frequency terms, regular PINNs may take a significant amount of time to converge to adequate accuracy even for relatively simple equations. One solution to this challenge of accelerating convergence when scaling PINNs to higher frequencies is introducing Random Fourier Features (RFF) into the training process [11].

The effect of these Fourier features can be viewed as an additional initialization step for the network input. Instead of passing the value of the independent variables $\mathbf{x}$ directly into the network $\varphi_\theta(\mathbf{x})$, we first convert the input into a series of triangular signals with a pre-specified, fixed random Gaussian matrix G using the map

$$\mathrm{RFF}(\mathbf{x}) = [\ \cos(2\pi \mathrm{G}\mathbf{x}),\ \sin(2\pi \mathrm{G}\mathbf{x})\ ]^{\mathrm{T}}, \tag{18}$$

where the components of G are drawn independently from a normal distribution $N(0,\sigma^2)$.

We carry out the numerical experiment to study the effect of such an improvement by considering a simple equation $y''(x) = -(n\pi)^2 y$ with Dirichlet boundary conditions imposed over the interval $[0,1]$ whose solution is given by $y = \sin(n\pi x)$. The results are summarized in Figure 2.

The experiment mainly reveals several key observations. First, we see the network converges much faster for y with a lower frequency when $n = 1$. To achieve convergence for such a function, only a regular PINN with a structure of 3 layers and one hidden layer containing 16 neurons is needed, which in all provides $(1 + 1) \times 16 + (16 + 1) \times 1 = 49$ trainable parameters.

However, the structure soon becomes inexpressive for $n = 5$ as the function's frequency increases. We see the regular network starts to become expressive again when two more hidden layers are added (5 layers in total) to the architecture, and when the number of neurons for each hidden layer is increased to 32. The number of trainable parameters now becomes $(1 + 1) \times 32 + (32 + 1) \times 32 \times 2 + (32 + 1) \times 1 = 2209$.

DOI: 10.54254/2753-8818/87/2025.20346

In comparison, over two thousand new parameters are added to the model to strengthen the expressiveness of the network, while the frequency only increases from 0.5 to 2.5. Luckily, the situation can be alleviated by adding Fourier features to the network structure and evaluating $\varphi_\theta(\mathrm{RFF}(\mathbf{x}))$ instead. With a Gaussian matrix G sampled from $R^{8\times1}$, the error of the network's output quickly converges to a scale of $O(10^{-2})$ for a 3-layer network with a hidden layer of width 16. This time, only $(2 \times 8 + 1) \times 16 + (16 + 1) \times 1 - 49 = 240$ new trainable parameters are needed to strengthen the network expressiveness.

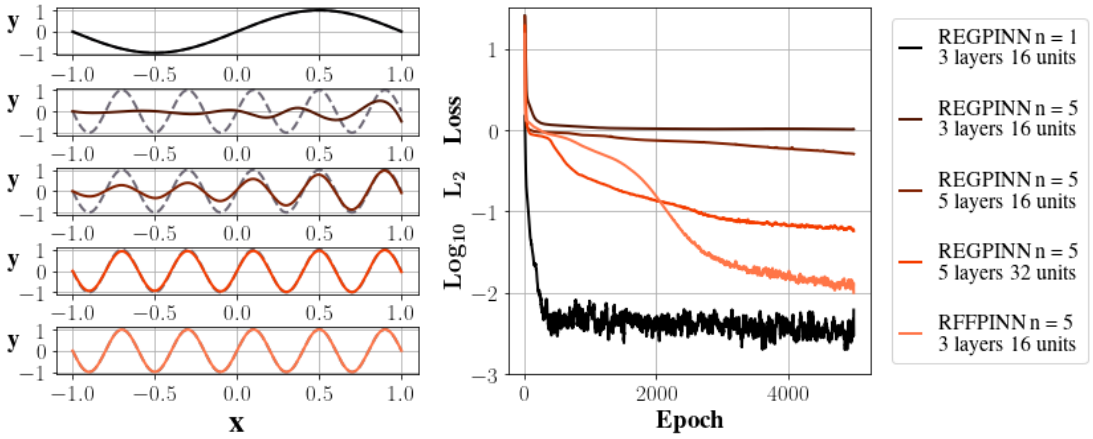

Figure 2: Apply PINNs to high-frequency functions. REGPINN stands for regular PINNs, and RFFPINN represents PINNs with random Fourier features.

4. Solving Partial Differential Equations

4.1. 2D Elliptic Equation

In this section, we apply the PINNs to solving 2D Laplace equations with Dirichlet boundary conditions over rectangular regions $\Omega = [a, b] \times [c, d]$,

$$u_{xx}(x, y) + u_{yy}(x, y) = f(x, y) \quad \text{for} \quad (x,y) \in \Omega, \tag{19}$$

and

$$u(a, y) = u_a(y), u(b, y) = u_b(y), \tag{20}$$

$$u(x, c) = u_c(x), \quad u(x, d) = u_d(x) \tag{21}$$

To be more specific, we specify the rectangular region to be $\Omega = [0,1] \times [0,1]$ and set the true solution to be $u_{true}(x, y) = \sin(2\pi x)\sin(3\pi y)$, so that it satisfies the equation for $f(x, y) = -13\pi^2 \sin(2\pi x)\sin(3\pi y)$ and $u_0(y) = u_1(y) = u_0(x) = u_1(x) = 0$.

The solution ansatz is given in the formula (22) to impose the boundary condition in a hard manner,

$$\begin{aligned}\varphi_\theta(x, y) = \hat{\varphi}_\theta(x, y) &+ \frac{b-x}{b-a} \cdot [u_a(y) - \hat{\varphi}_\theta(a, y)] + \frac{x-a}{b-a} \cdot [u_b(y) - \hat{\varphi}_\theta(b, y)] \\ &+ \frac{d-y}{d-c} \cdot [u_c(x) - \hat{\varphi}_\theta(x, c)] + \frac{y-c}{d-c} \cdot [u_d(x) - \hat{\varphi}_\theta(x, d)] \\ &- \frac{(b-x)(d-y)}{(b-a)(d-c)} \cdot [u_a(c) - \hat{\varphi}_\theta(a, c)] - \frac{(x-a)(d-y)}{(b-a)(d-c)} \cdot [u_b(c) - \hat{\varphi}_\theta(b, c)] \\ &- \frac{(b-x)(y-c)}{(b-a)(d-c)} \cdot [u_a(d) - \hat{\varphi}_\theta(a, d)] - \frac{(x-a)(y-c)}{(b-a)(d-c)} \cdot [u_b(d) - \hat{\varphi}_\theta(b, d)],\end{aligned} \tag{22}$$

with the loss function

DOI: 10.54254/2753-8818/87/2025.20346

$$\mathcal{L}(\theta) = \frac{1}{N_p}\sum_{i=1}^{N_p}\left\|\frac{\partial^2\varphi_\theta}{\partial x^2}(x_i, y_i) + \frac{\partial^2\varphi_\theta}{\partial y^2}(x_i, y_i) - f(x_i, y_i)\right\|^2. \quad (23)$$

We trained a 3-layer network over 2000 sampled points with a batch size of 32 using the Adam optimizer to obtain the solution of the equation (19) – (21); the loss changes are tracked and recorded in the right column of Figure 3 alongside the changes in the L_2 loss.

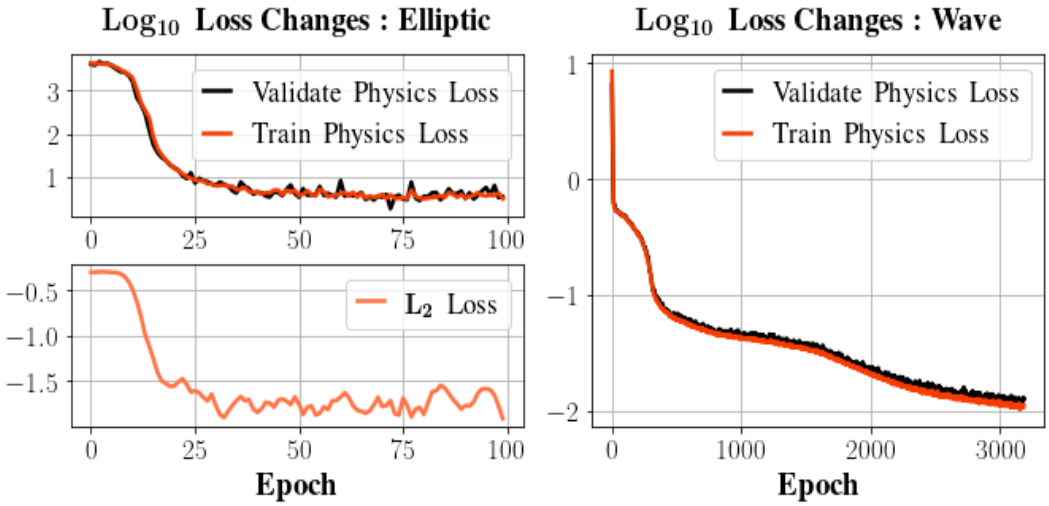

Figure 3: The plots on the left column give the changes in both the physics loss and the L_2 loss while training the PINN to solve the sample elliptic equation (19) – (21); The plots on the right column give the changes in physics losses while training the network to solve the sample wave equation in Section 4.2.

The output of the PINN and its error corresponding to the true solution are shown in Figure 4. For most of the points, the absolute error is controlled within a scale of $O(10^{-2})$, indicating a relatively good approximation.

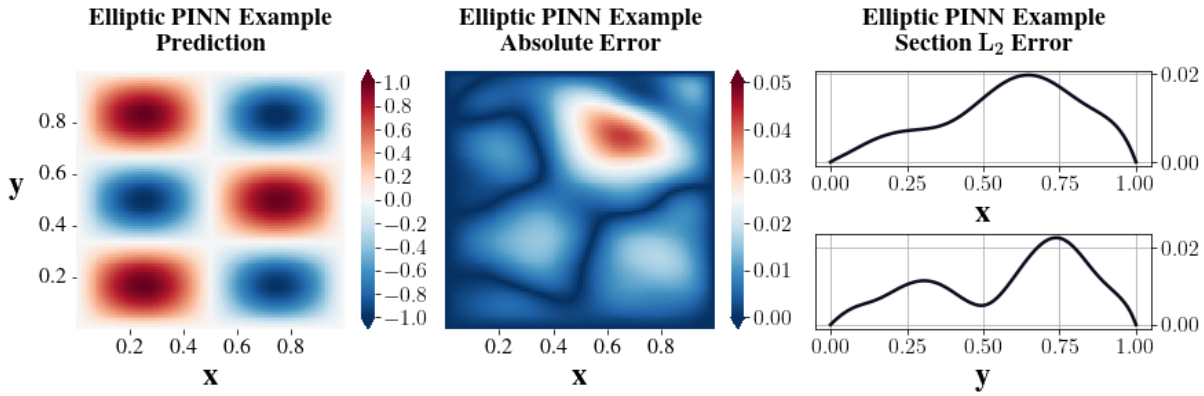

Figure 4: The solution of sample elliptic equation (19) – (21) generated by the PINN (plot 1) with prediction error displayed in plot 2 and 3.

4.2. 3D Wave Equation

The method is also generalized to solve 3D wave equations with 2 spatial dimensions. Consider finding $u(x,y,t)$ over a target region $\Omega = [0,a] \times [0,b] \times [0,T]$ which satisfies the equation,

DOI: 10.54254/2753-8818/87/2025.20346

$$u_{tt} = c^2(u_{xx} + u_{yy}) \text{ for } c > 0, \tag{24}$$

$$u(0,y,t) = u(a,y,t) = u(x,0,t) = u(x,b,t) = 0 \tag{25}$$

$$u(x,y,0) = g(x,y) \quad \text{and} \quad u_t(x,y,0) = \phi(x,y). \tag{26}$$

The analytical solution to this problem can be obtained via Fourier series expansions

$$u_{\text{true}}(x,y,t) = \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} \sin(\mu_m x)\sin(\nu_n y)[A_{mn}\cos(\lambda_{mn}t) + B_{mn}\sin(\lambda_{mn}t)], \tag{27}$$

where $\mu_m = \frac{m\pi}{a}$, $\nu_n = \frac{n\pi}{b}$, $\lambda_{mn} = c\sqrt{\mu_m^2 + \nu_n^2}$, $A_{mn} = \frac{4}{ab}\int_0^a\int_0^b g(x,y)\sin(\mu_m x)\sin(\nu_n y)\,dydx$ and $B_{mn} = \frac{4}{ab\lambda_{mn}}\int_0^a\int_0^b \phi(x,y)\sin(\mu_m x)\sin(\nu_n y)\,dydx$.

This time, the conditions are applied softly to train the network $\varphi_\theta(x,y,t)$, with a loss function

$$\begin{aligned}\mathcal{L}(\theta) = {} & \frac{\lambda_0}{N_p}\sum_{i=1}^{N_p}\left\|\left[\partial_{tt}^2\varphi_\theta - c^2\left(\partial_{xx}^2\varphi_\theta + \partial_{yy}^2\varphi_\theta\right)\right](x_i,y_i,t_i)\right\|^2 + \frac{\lambda_1}{N_{b,1}}\sum_{j=1}^{N_{b,1}}\left\|\varphi_\theta(0,y_{1,j},t_{1,j})\right\|^2 \\ & + \frac{\lambda_2}{N_{b,2}}\sum_{j=1}^{N_{b,2}}\left\|\varphi_\theta(a,y_{2,j},t_{2,j})\right\|^2 + \frac{\lambda_3}{N_{b,3}}\sum_{j=1}^{N_{b,3}}\left\|\varphi_\theta(x_{3,j},0,t_{3,j})\right\|^2 + \frac{\lambda_4}{N_{b,4}}\sum_{j=1}^{N_{b,4}}\left\|\varphi_\theta(x_{4,j},b,t_{4,j})\right\|^2 \\ & + \frac{\lambda_5}{N_{b,5}}\sum_{j=1}^{N_{b,5}}\left\|\varphi_\theta(x_{5,j},y_{5,j},0) - g(x_{5,j},y_{5,j})\right\|^2 + \frac{\lambda_6}{N_{b,6}}\sum_{j=1}^{N_{b,6}}\left\|\partial_t\varphi_\theta(x_{6,j},y_{6,j},0) - \phi(x_{6,j},y_{6,j})\right\|^2.\end{aligned} \tag{28}$$

Specifically, we solved the equation for a = b = c = T = 1 and trained the network with $\lambda_0 = 5$ and $\lambda_k = 20$ for k = 1,⋯ ,6. 2500 points from each of the five boundaries and the interior of the region were sampled to form a training data set of size 15000. The training loss changes were recorded in Figure 3, and the predictions of the PINN are given in Figure 5. Again, the scale of the absolute errors is approximately $O(10^{-2})$.

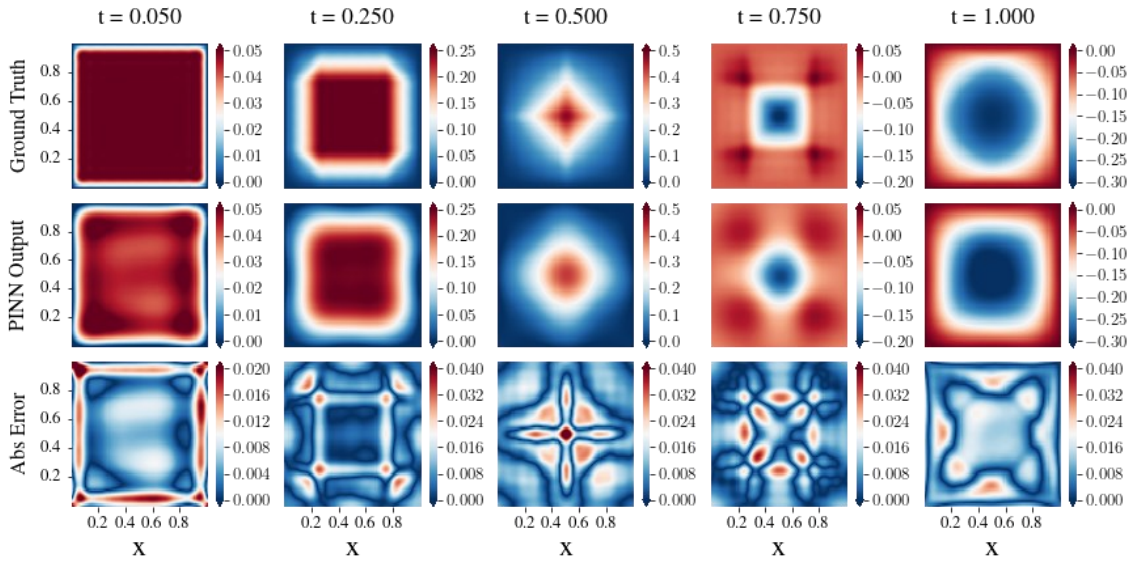

Figure 5: The solution of the sample wave equation generated by the PINN.

5. Conclusion

This paper mainly discusses the application of physics-informed neural networks, i.e., PINNs, to solving differential equations. We first apply the method to ODEs and ODE systems under different types of boundary conditions and find that the solution generated tends to converge slower for high-

DOI: 10.54254/2753-8818/87/2025.20346

frequency functions. We then fix the problem and improve the convergence pattern for networks fitting high-frequency solutions by adding random Fourier features to the network structure. Finally, we generalize our work to study the behaviour of PINNs over PDEs with two and three independent variables, where the general absolute errors between the network output and the ground truth are controlled within a scale of $O(10^{-2})$.

However, there are still many aspects that our study fails to cover. For example, most of our example networks are trained under hard conditions. Nevertheless, finding a corresponding solution ansatz can be challenging for complex boundary conditions. Clearly, employing soft conditions is a more versatile approach in the general sense. Also, for more complicated practical problems, PINNs usually suffer from problems of high computational cost and slow convergence.

To increase the flexibility of the method, one possible improvement is to consider conditioned PINNs, where the initial and boundary conditions, as well as other possible features of the equation, are also taken in as part of the network inputs to avoid retraining of similar models [12, 13]. In terms of reducing the computational cost, possible adjustments may be either employing more advanced network architectures [14], or letting networks prioritize learning sample points with higher loss values [15].

References

[1] *M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations". In: ArXiv abs/1711.10561 (2017). url: https: //api.semanticscholar. org/CorpusID:394392 (visited on 05/04/2024).*

[2] *W. Ee and B. Yu. "The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems". In: Communications in Mathematics and Statistics 6 (2017), pp. 1–12. url: https://api.semanticscholar. org/CorpusID:2988078 (visited on 05/04/2024).*

[3] *J. Berg and K. Nystr¨om. "A Unified Deep Artificial Neural Network Approach to Partial Differential Equations in Complex Geometries". In: ArXiv abs/1711.06464 (2017). url: https://api.semanticscholar. org/CorpusID: 38319575 (visited on 05/04/2024).*

[4] *J. Adler and O. Oktem. "Solving Ill-posed Inverse Problems Using Iterative Deep Neural Networks". In:¨ Inverse Problems 33.12 (Nov. 2017), p. 124007. issn: 1361-6420. doi: 10.1088/1361-6420/aa9581. url:http://dx.doi.org/10. 1088/1361-6420/aa9581 (visited on 05/04/2024).*

[5] *Z. Chen, Y. Liu, and H. Sun. "Physics-informed Learning of Governing Equations from Scarce Data". In: Nature Communications 12 (2020). url: https://api.semanticscholar.org/CorpusID:239455737 (visited on 05/04/2024).*

[6] *F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." In: Psychological review 65 6 (1958), pp. 386–408. url: https://api.semanticscholar.org/ CorpusID:12781225 (visited on 05/05/2024).*

[7] *A.G. Baydin et al. "Automatic Differentiation in Machine Learning: A Survey". In: Journal of Machine Learning Research 18 (Apr. 2018), pp. 1–43.*

[8] *PyTorch autograd.grad. url: https://pytorch.org/docs/stable/generated/torch.autograd.grad. html (visited on 05/04/ 2024).*

[9] *D.P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. 2017. arXiv: 1412.6980[cs.LG].*

[10] *N. Rahaman et al. "On the Spectral Bias of Neural Networks". In: International conference on machine learning. PMLR. 2019, pp. 5301–5310.*

[11] *M. Tancik et al. "Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains". In: NIPS '20., Vancouver, BC, Canada, Curran Associates Inc., 2020. isbn: 9781713829546.*

[12] *B. Moseley, A. Markham, and T. Nissen-Meyer. Solving the Wave Equation with Physics-Informed Deep Learning. June 2020. url: https://arxiv.org/abs/2006.11894 (visited on 05/05/2024).*

[13] *S. Wang, H. Wang, and P. Perdikaris. "Learning the Solution Operator of Parametric Partial Differential Equations with Physics-Informed Deep Nets". In: Science Advances 7 (Sept. 2021). doi: 10.1126/sciadv. abi8605.*

[14] *Y. Zhu et al. "Physics-Constrained Deep Learning for High-dimensional Surrogate Modeling and Uncertainty Quantification without Labeled Data". In: Journal of Computational Physics 394 (2019), pp. 56– 81. issn: 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.05.024. url: https://www. sciencedirect.com/science/article/pii/ S0021999119303559 (visited on 05/05/2024).*

DOI: 10.54254/2753-8818/87/2025.20346

[15] C. Wu et al. "A Comprehensive Study of Non-adaptive and Residual-based Adaptive Sampling for PhysicsInformed Neural Networks". In: Computer Methods in Applied Mechanics and Engineering 403 (Jan. 2023), p. 115671. doi: 10.1016/j.cma.2022.115671.

Appendix

All codes in this paper can be found via: https://github.com/abaaba337/MMSC-Computing-Case-Study-PINN.