Using Physics Informed Neural Network (PINN) for modeling the heat conduction in 1D and 2D geometries

Amir Nazemi[a], Ali Bahrami[a]

[a]*Department of Mechanical Engineering, University of Michigan, Ann Arbor, 48109, MI, USA*

Abstract

Currently, machine learning has been utilized in a wide range of areas. Machine learning has proven to be highly beneficial and effective in the realm of science and physics, leading to the emergence of a new discipline known as scientific machine learning. A wide array of machine learning branches has been applied in the field of scientific machine learning. A significant advancement in this domain occurred after the development of physics-informed machine learning. A Physics Informed Neural Network (PINN) is a neural network designed to accurately represent the solution of linear or nonlinear partial differential equations. This study employs an application of Physics-Informed Neural Networks (PINN) to address a heat transfer problem. The performance of Physics-Informed Neural Networks (PINN) in heat conduction applications is examined through three case studies: 1D steady-state, 1D transient, and 2D transient heat conduction. To ensure that the model prediction is reasonably accurate, a technique called self-adaptive weighting method has been incorporated into the PINN to significantly decrease the model's prediction error. Finally, another application of the PINN, which is physics discovery, was investigated. The study demonstrated that the PINN can achieve an accuracy of 93.2% in predicting the thermal diffusivity coefficient of a material, given a set of experimental data of a material with the unknown corresponding parameter.

Keywords: Scientific Machine Learning, Physics Informed Neural Network, Heat Transfer, Heat Conduction, Physics Discovery

1. Introduction

In contemporary times, the utilization of computers for simulating complex processes proves instrumental in enhancing the analysis of technological and physics-based phenomena, thereby optimizing efficiency. Physicists engage in the modeling of these intricate processes, culminating in the derivation of governing equations. The complexity of a given process directly correlates with the challenge posed in solving its governing equation. Notably, within the realm of fluid mechanics, the equations governing fluid flow and heat transfer exhibit a high degree of complexity, necessitating the application of diverse numerical methods for resolution. Over the past five decades, significant strides have been made in comprehending multi-scale physics across various disciplines. This progress has been achieved through the numerical solution of partial differential equations (PDEs) employing diverse methodologies such as finite differences, finite elements, spectral, and meshless methods [1; 2; 3].

Despite considerable advancements, the endeavor to model and predict the evolution of nonlinear multi-scale systems characterized by inhomogeneous cascades-of-scales encounters formidable challenges when relying on classical analytical or computational tools. These challenges give rise to exorbitant costs and multiple sources of uncertainty. Additionally, addressing inverse problems, such as inferring material properties in functional materials or unraveling missing physics in reactive transport, is frequently cost-prohibitive. These tasks demand intricate formulations, innovative algorithms, and sophisticated computer codes. Importantly, tackling real-life physical problems afflicted by missing, incomplete, or noisy boundary conditions through traditional approaches stands as a currently insurmountable challenge. The need for novel methodologies, integrating cutting-edge scientific techniques and advanced computational strategies, is evident to overcome the limitations posed by existing paradigms and unlock new frontiers in our ability to understand and manipulate complex, multi-scale systems [1; 4].

Observational data plays a pivotal role in contemporary research, particularly with the imminent deployment of over a trillion sensors, encompassing airborne, seaborne, and satellite remote sensing. This vast array of multi-fidelity observations, characterized by their volume, velocity, and variety, presents an opportunity for exploration through data-driven methods. However, integrating such data seamlessly into existing physical models remains challenging, given the spatiotemporal heterogeneity and lack of universally acceptable models [1; 5].

The emergence of machine learning (ML) addresses this challenge by leveraging its capacity to explore expansive design spaces, identify multi-dimensional correlations, and tackle ill-posed problems. ML, especially deep learning approaches, can automatically extract features from massive multi-fidelity observational data, offering tools for detecting climate extremes and statistically predicting dynamic variables like precipitation or vegetation productivity. Additionally, deep learning facilitates the linkage of these features with existing approximate models, paving the way for the development of new predictive tools [1].

Despite the widespread use of machine learning (ML), many ML approaches struggle to extract interpretable knowledge from vast datasets. While data-driven models may fit observations well, their predictions can be physically inconsistent due to issues like extrapolation or observational biases, limiting generalization performance. Thus, there is a crucial need to integrate fundamental physical laws and domain knowledge into ML models, 'teaching' them governing rules to provide more informative priors for enhanced predictive accuracy [1].

An illustration of the evolving learning approach is seen in 'physics-informed neural networks' (PINNs), a type of deep learning algorithm. PINNs seamlessly integrate data and mathematical operators, accommodating partial differential equations (PDEs) with or without missing physics. The main goal is to use prior knowledge or constraints for creating more understandable machine learning (ML) methods. These methods prove robust even with imperfect data, like missing values or noise, delivering accurate and physically consistent predictions, especially in tasks involving extrapolation or generalization [1; 4; 5].

In this study, we delve into the application of Physics-Informed Neural Networks (PINNs) as part of a course project, focusing on solving the heat conduction equation in both 1D and 2D geometries. The primary goals include gaining a foundational understanding of PINN, mastering its implementation using TensorFlow 2, and navigating the challenges inherent in solving governing equations for physical phenomena. Throughout the report, we detail the problem's Partial Differential Equation (PDE) and boundary conditions in Section 2.1, elucidate the background of PINN in Section 2.2, and explore the PINN structure with adaptive loss and for physics discovery in Sections 2.3 and 2.4. The information about training parameters are also explained in 2.5. The results and conclusions are subsequently presented in Sections 3 and 4, respectively.

2. Background and Methodology

In this section, the various facets of the problem, encompassing the governing equations for heat conduction, the formulation of equations for the PINN model, the incorporation of Adaptive Loss in PINN, and the exploration of Physics Discovery in the context of heat conduction will be comprehensively addressed. Additionally, we elucidate the methodology employed for implementing these models in TensorFlow 2.

2.1. Heat Conduction Equation

Heat conduction represents a mode of heat transfer characterized by the diffusion mechanism, wherein heat is propagated through a medium. A prominent illustration of this phenomenon is observed in solids, where heat is transferred in response to a temperature gradient. The governing equation for heat conduction in Cartesian coordinates is expressed in its general form as depicted in Equation (1). This equation encapsulates the fundamental principles underlying heat conduction, providing a foundational understanding of the thermal dynamics in materials experiencing temperature variations [3].

$$\frac{\partial}{\partial \tau}(\rho c_p T) = \frac{\partial}{\partial X}\left(k\frac{\partial T}{\partial X}\right) + \frac{\partial}{\partial Y}\left(k\frac{\partial T}{\partial Y}\right) + \frac{\partial}{\partial Z}\left(k\frac{\partial T}{\partial Z}\right) + \dot{Q} \quad (1)$$

where T denotes temperature, and X, Y, Z represent the Cartesian coordinates in space. The variable τ signifies time, while k stands for conductivity, and ρ and c_p denote density and specific heat capacity, respectively. The term $\dot{Q}$ refers to the heat source. To increase the perfomance of PINN, it is preferred to use the normalized form of equations. Therefore, the form of equation (1) is as modified as below.

$$\frac{\partial u}{\partial t} = \alpha\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right] + \frac{1}{\rho c_p}\dot{Q} \quad (2)$$

where $u = (T - T_0)/(T_0 - T_1)$, where T_0 represents the initial temperature, T_1 denotes the minimum temperature in the system, and x, y, z are the transformed coordinate components ranging from -1 to 1. The variable t signifies scaled time, ranging from 0 to 1, while $\alpha = k/(\rho c_p)$ characterizes the diffusivity of the material.

In this project, our focus is exclusively on addressing the 1D steady state, 1D transient, and 2D transient scenarios of heat conduction through the application of Physics-Informed Neural Networks (PINNs), aiming to derive surrogate models for these systems. In what follows, we elaborate on the equations, as well as the appropriate initial and boundary conditions for each of these cases.

I. 1D Steady State:

In this case, the temporal, y, and z spatial terms of Equation (2) are disregarded. Moreover, for this case and all subsequent cases, the heat source term is omitted. The simplified equation is expressed as follows:

$$\alpha\frac{\partial^2 u}{\partial x^2} = 0 \quad (3)$$

The boundary conditions for this equation are defined as $u = 0$ at $x = -1$ and $u = 1$ at $x = 1$. The visual representation of this case is presented in Figure 1a. The analytical solution for this equation is given by $u = 0.5(x + 1)$ for x within the range of -1 to 1.

II. 1D Transient:

In this case, the spatial terms y and z in Equation (2) are disregarded.

$$\frac{\partial u}{\partial t} = \alpha\frac{\partial^2 u}{\partial x^2} \quad (4)$$

The boundary conditions for this equation are specified as $u = 0$ at $x = -1$ and $u = 0$ at $x = 1$. Additionally, the initial condition for this problem is set as $u = 1$ at $t = 0$. The visual representation of this case is depicted in Figure 1b. The solution to this equation is computed using the Finite Difference (FD) implicit method, with the method's details provided in Appendix A.

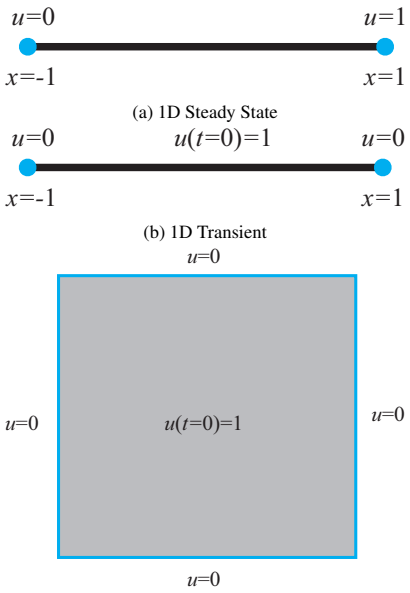


Figure 1: The schematic figure of the problem for different cases

III. 2D Transient:

In this case, the spatial term z in Equation (2) is disregarded.

$$\frac{\partial u}{\partial t} = \alpha\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right] \tag{5}$$

The boundary condition for this equation is specified as $u = 0$ at $x = -1$, $x = 1$, $y = -1$, and $y = 1$. Additionally, the initial condition for this problem is set as $u = 1$ at $t = 0$. The visual representation of this case is illustrated in Figure 1c. The solution to this equation is computed using the Finite Difference (FD) implicit method, with the method's details provided in Appendix Appendix A.

It is crucial to emphasize that all subsequent formulations are presented based on the 2D Transient case, and deriving equivalent equations for other cases is straightforward.

2.2. PINN

Typically, engineering systems with a PDE constraint can be formulated as:

$$u_t + \mathcal{N}[u] = 0, x, y \in \Omega, t \in [0, T] \tag{6}$$

where $u(x, y, t)$ denotes the latent (hidden) solution, $\mathcal{N}[.]$ is a nonlinear/linear differential operator, Ω is a subset of R^d, and x, y and t are the space and time with 0 and T representing the spatial domain and time span. In Physics-Informed Neural Networks (PINNs), the unknown parameters 0, which represent the weights and biases, are used to infer the latent function $u(x, y, t)$ (e.g., temperature field $T(x, y, t)$ in a heat transfer issue) using a feed-forward neural network. To achieve an optimal set of parameters, an optimization problem may be solved using gradient descent. This involves minimizing a composite loss function, which is expressed in the following form:

$$\mathcal{L}(\theta) := \mathcal{L}_{PDE}(\theta) + \mathcal{L}_{BC}(\theta) + \mathcal{L}_{IC}(\theta) \tag{7}$$

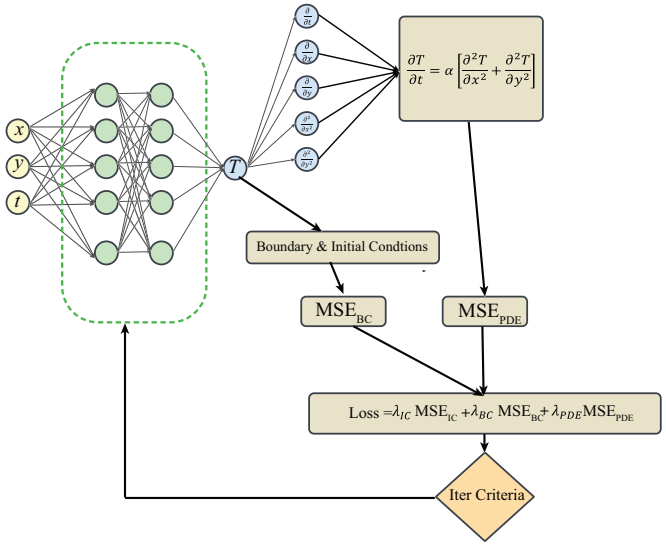


Figure 2: The flowchart of PINN algorithm

$\mathcal{L}_{PDE}(\theta)$ represents the loss term that imposes the governing rules and physics dictated by partial differential equations (PDEs). The PDE ($u_t + \mathcal{N}[u] = 0$) residuals are penalized at certain collocation points ($x_{PDE}, y_{PDE}, t_{PDE}$), which are frequently chosen in a random manner. $\mathcal{L}_{IC}(\theta)$ and $\mathcal{L}_{BC}(\theta)$ represent the losses related to the initial and boundary conditions, respectively. To reduce the losses during training, it is necessary to define initial and boundary locations, similar to those in $\mathcal{L}_{PDE}(\theta)$. A well-trained neural network $f_\theta(x, y, t)$ with a close-to-zero $\mathcal{L}(\theta)$ can accurately represent the solution of the PDEs for the specific task in mind. The mean squared error (MSE) is the standard loss function used for the loss terms in PINNs [6]. For the current study, losses can be written as equations (8). Also, a summary of the network architecture can be found in Figure 2.

$$\mathcal{L}_{PDE}(\theta) = \text{MSE}\left[\frac{\partial u}{\partial t} - \alpha\left(\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2}\right)\right] \tag{8a}$$

$$\begin{aligned}\mathcal{L}_{BC}(\theta) = &\text{MSE}[u(-1, y, t) - 0] \\ &+ \text{MSE}[u(+1, y, t) - 0] \\ &+ \text{MSE}[u(x, -1, t) - 0] \\ &+ \text{MSE}[u(x, +1, t) - 0]\end{aligned} \tag{8b}$$

$$\mathcal{L}_{IC}(\theta) = \text{MSE}[u(x, y, 0) - 1] \tag{8c}$$

2.3. PINN - Self Adaptive Loss

The basic Physics-Informed Neural Network (PINN) method, as explained earlier, has demonstrated remarkable performance in solving many linear and nonlinear PDEs. However, it may yield imprecise approximations or fail to converge completely when used to certain 'stiff' PDEs. A great amount of evidence has accumulated suggesting that this arises due to the limitations of gradient descent applied to the multi-part or

multi-objective loss function, such as [7; 8; 9; 10]. This phenomenon arises due to the nature of gradient descent, which is an algorithm that prioritizes certain components over others, leading to an imbalanced rate of descent among the various loss components. As a result, convergence to the proper solution is hindered.

All prior approaches in the literature utilize a linearly-scalarized function as a weight, with the sole distinction lying in the manner in which the weights are updated. However, the self-adaptive weighting approach (SA-PINNs) differs fundamentally in that it assigns weights to individual training points in the various loss components, rather than updating the weights for the entire loss component simultaneously. Out of the methods cited before, the Minimax weighting scheme [11] is the most similar to SA-PINNs because it also updates its weights using gradient ascent. However, in this scheme, the weights are applied to all loss components as a whole, whereas in the self-adaptive weighting method, each training point is weighted individually using its own weight variable. This strategy has a resemblance to attention methods employed in computer vision [12; 13]. The suggested technique aligns with the neural network principle of self-adaptation by updating the weights in the loss function by gradient descent, rather than hard-coding them to specific parts of the solution. The proposed self-adaptive PINN utilizes the loss functions in equations (9).

$$\begin{aligned}\mathcal{L}(\theta, \lambda_{PDE}, \lambda_{BC}, \lambda_{IC}) \coloneqq & \mathcal{L}_{PDE}(\theta, \lambda_{PDE}) \\ & + \mathcal{L}_{BC}(\theta, \lambda_{BC}) \\ & + \mathcal{L}_{IC}(\theta, \lambda_{IC})\end{aligned} \tag{9a}$$

$$\mathcal{L}_{PDE}(\theta, \lambda_{PDE}) = \mathrm{MSE}\left(\lambda_{PDE}\left(\frac{\partial u}{\partial t} - \alpha\left(\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2}\right)\right)\right) \tag{9b}$$

$$\begin{aligned}\mathcal{L}_{BC}(\theta, \lambda_{BC}) = & \mathrm{MSE}(\lambda_{BC_1}(u(-1, y, t) - 0)) \\ & + \mathrm{MSE}(\lambda_{BC_2}(u(+1, y, t) - 0)) \\ & + \mathrm{MSE}(\lambda_{BC_3}(u(x, -1, t) - 0)) \\ & + \mathrm{MSE}(\lambda_{BC_4}(u(x, +1, t) - 0))\end{aligned} \tag{9c}$$

$$\mathcal{L}_{IC}(\theta, \lambda_{IC}) = \mathrm{MSE}(\lambda_{IC}(u(x, y, 0) - 1)) \tag{9d}$$

where $\lambda_{PDE} = \left(\lambda^1_{PDE}, \ldots, \lambda^{n_{PDE}}_{PDE}\right), \lambda_{BC_i} = \left(\lambda^1_{BC_i}, \ldots, \lambda^{n_{BC}}_{BC_i}\right)$, and $\lambda_{IC} = \left(\lambda^1_{IC}, \ldots, \lambda^{n_{IC}}_{IC}\right)$ are trainable, nonnegative self-adaptation weights for the collocation, initial, and boundary points, respectively. The self-adaptation mask function λ is a nonnegative and differentiable function defined on the interval $[0, \infty)$, and it is strictly growing. One important characteristic of self-adaptive PINNs is that the loss function $\mathcal{L}(\theta, \lambda_{PDE}, \lambda_{BC}, \lambda_{IC})$ is minimized with respect to the network weights w, as normal, but is maximized with respect to the self-adaptation weights $\lambda_{PDE}, \lambda_{BC}, \lambda_{IC}$. The steps for gradient descent/ascent that relate to this are as equations (10).

$$\theta^{k+1} = \theta^k - \eta_k \nabla_\theta \mathcal{L}(\theta, \lambda_{PDE}, \lambda_{BC}, \lambda_{IC}) \tag{10a}$$

$$\lambda^{k+1}_{PDE} = \lambda^k_{PDE} + \rho^k \nabla_{\lambda_{PDE}} \mathcal{L}(\theta, \lambda_{PDE}, \lambda_{BC}, \lambda_{IC}) \tag{10b}$$

$$\lambda^{k+1}_{BC_i} = \lambda^k_{BC_i} + \rho^k \nabla_{\lambda_{\mathrm{BC}_i}} \mathcal{L}(\theta, \lambda_{PDE}, \lambda_{BC}, \lambda_{IC}) \tag{10c}$$

$$\lambda^{k+1}_{IC} = \lambda^k_{IC} + \rho^k \nabla_{\lambda_{IC}} \mathcal{L}(\theta, \lambda_{PDE}, \lambda_{BC}, \lambda_{IC}) \tag{10d}$$

where $\eta_k = 10^{-3}$ is the learning rate for the neural network weights at step k, $\rho^k = 0.2$ is a separate learning rate for the self-adaption weights.

It is evident that the sequences of weights λ^k_{PDE}, $k = 1, 2,$, λ^k_{BC}, $k = 1, 2,$, λ^k_{IC}, $k = 1, 2$, (and the related mask values) exhibit a consistent upward trend, as long as the corresponding unmasked losses are not equal to zero. Moreover, the size of the gradients $\nabla_{\lambda_{PDE}} \mathcal{L}$, $\nabla_{\lambda_{\mathrm{BC}_i}} \mathcal{L}$, $\nabla_{\lambda_{IC}\mathcal{L}} \mathcal{L}$, and hence the updates, is greater when the corresponding unmasked losses are bigger. Furthermore, the extent of updates may be regulated by defining a schedule for the learning rates ρ^k, so introducing further adaptability. This imposes more severe penalties on the network for not closely matching the residual, boundary, and initial points. It should be noted that any of the weights can be assigned fixed, non-trainable values, if desired. By setting $\lambda^k_{BC_i} \equiv 1$, just the weights of the initial and residue points will be trained. If the data is composed of noisy observations, caution must be taken to properly weigh them in order to prevent overfitting. However, it should be noted that this issue is still an ongoing research challenge. It is important to initialize the self-adaptive weights at the start of training. The weights can be initialized to either 1 (no weighting) or to a different number, depending on the particular circumstances. Alternatively, they can be randomly initialized within a certain range, mirroring the common practice of initializing neural network weights. Prior knowledge is crucial in this context. For instance, if it is known that the initial conditions in a problem are difficult to accommodate, the weights for the initial conditions could be initialized to a higher value compared to the other weights. Alternatively, the initial condition weights could be initialized at the same value as the other weights but with a larger learning rate. In this study, all weights are initialized randomly in the interval of $[0, 1)$. The gradient ascent/descent step may be readily accomplished using readily available neural network software by negating the values of $\nabla_{\lambda_{PDE}} \mathcal{L}$, $\nabla_{\lambda_{\mathrm{BC}_i}} \mathcal{L}$, and $\nabla_{\lambda_{IC}} \mathcal{L}$. For this study, the described method utilized using Tensorflow 2.7 and employ a predetermined number of iterations of the Adam optimizer [14]. Also, a summary of the network architecture can be found in Figure 3.

2.4. Physics Discovery

PINN can be extensively utilized to uncover the fundamental principles of a given event, such as the underlying PDE of a partially unknown problem or the characteristics of a material employed in a recognized process. In this study, it was assumed a heat conduction on an unknown material, which corresponds to $\mathcal{N}[u, \alpha_1] = -\alpha_1\left(\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2}\right)$. The current challenge is in the data-driven exploration of partial differential equations for physics discovery, possessing the following question: given a small set of scattered and potentially observations (which can be experimental or numerical) of the hidden state $u(x, y, t)$ of

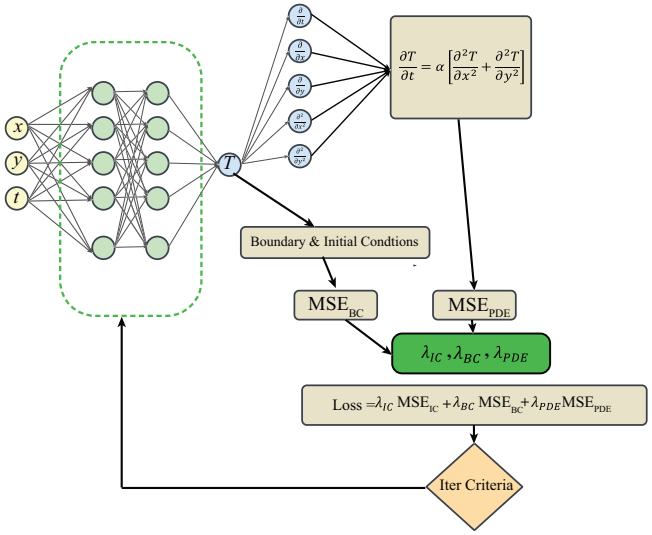


Figure 3: The flowchart of PINN algorithm with adaptive loss

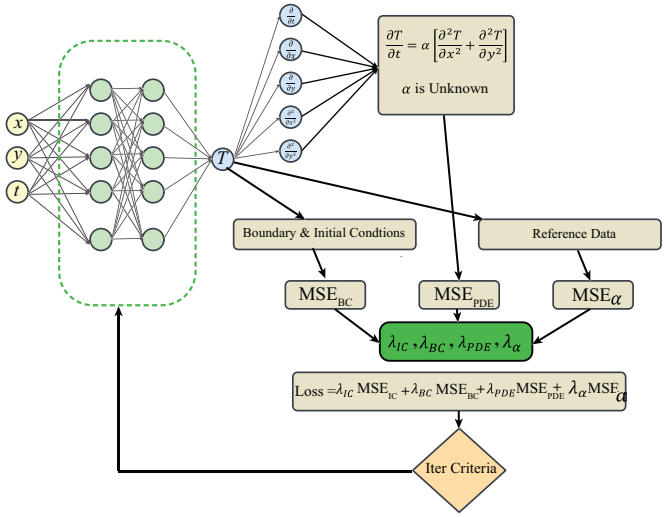


Figure 4: The flowchart of PINN algorithm with adaptive loss for Physics Discovery

a system, what are the parameters α_1 that best describe the observed data? Therefore, one more term is added to the total loss, which leads to equations (11).

$$\begin{aligned}\mathcal{L}(\theta, \lambda_{PDE}, \lambda_{BC}, \lambda_{IC}, \lambda_{data}, \alpha_1) := \mathcal{L}_{PDE}(\theta, \lambda_{PDE}, \alpha_1) \\ + \mathcal{L}_{BC}(\theta, \lambda_{BC}) \\ + \mathcal{L}_{IC}(\theta, \lambda_{IC}) \\ + \mathcal{L}_{data}(\theta, \lambda_{data})\end{aligned} \tag{11a}$$

$$\mathcal{L}_{PDE}(\theta, \lambda_{PDE}, \alpha_1) = \text{MSE}\left(\lambda_{PDE}\left(\frac{\partial u}{\partial t} - \alpha_1\left(\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2}\right)\right)\right) \tag{11b}$$

$$\begin{aligned}\mathcal{L}_{BC}(\theta, \lambda_{BC}) = \text{MSE}(\lambda_{BC_1}(u(-1, y, t) - 0)) \\ + \text{MSE}(\lambda_{BC_2}(u(+1, y, t) - 0)) \\ + \text{MSE}(\lambda_{BC_3}(u(x, -1, t) - 0)) \\ + \text{MSE}(\lambda_{BC_4}(u(x, +1, t) - 0))\end{aligned} \tag{11c}$$

$$\mathcal{L}_{IC}(\theta, \lambda_{IC}) = \text{MSE}(\lambda_{IC}(u(x, y, 0) - 1)) \tag{11d}$$

$$\begin{aligned}\mathcal{L}_{\text{data}}(\theta, \lambda_{\text{data}}) = \text{MSE}(\lambda_{\text{data}}(u(x^{\text{obs}}, y^{\text{obs}}, t^{\text{obs}}) \\ - u^{\text{obs}}(x^{\text{obs}}, y^{\text{obs}}, t^{\text{obs}})))\end{aligned} \tag{11e}$$

where $x^{obs}, y^{obs}, t^{obs}$ are the spatial and time coordinates of the observed points and $u^{obs}\left(x^{obs}, y^{obs}, t^{obs}\right)$ is the temperature at the observed points. By initializing α_1 with a random number (here is 0.5), similarly, an optimal set of parameters can be obtained via an optimization problem, i.e., using Adam optimizer, to minimize the corresponding composite loss function, and get reach to the actual value of α_1. Also, a summary of the network architecture can be found in Figure 4.

2.5. Training

The in-house algorithm utilizes an implicit finite difference approach to simulate the heat transfer of the previously mentioned issue and solve the relevant heat transfer partial differential equations (PDEs). The baseline, self-adaptive method, and physics discovery PINN models all utilize networks with a certain number of hidden layers (N_{layer}) and neurons (N_{neuron}), as outlined in Table 1. These hidden layers employ a hyperbolic tangent activation function. The output layer lacks an activation function, meaning it utilizes a linear activation function with a slope of 1. The training of the PDE involves the usage of the ADAM optimizer with a learning rate of 0.001. A total of (N_{epoch}) epochs are implemented, which may be found in Table 1. The models' performance is evaluated using a test dataset consisting of n_{PDE}, n_{BC}, n_{IC} points, which is shown in Figures 8e and 9e. for both 1D transient and 2D transient cases. For 1D steady state, $n_{PDE} = 1000$, $n_{BC} = 2$, for 1D transient case $n_{PDE} = 50000$, $n_{BC} = 1000$ per BC, $n_{IC} = 1000$, and 2D transient case, $n_{PDE} = 30000$, $n_{BC} = 1000$ per BC, $n_{IC} = 1000$. The construction and training of all models are carried out in Python using the TensorFlow framework.

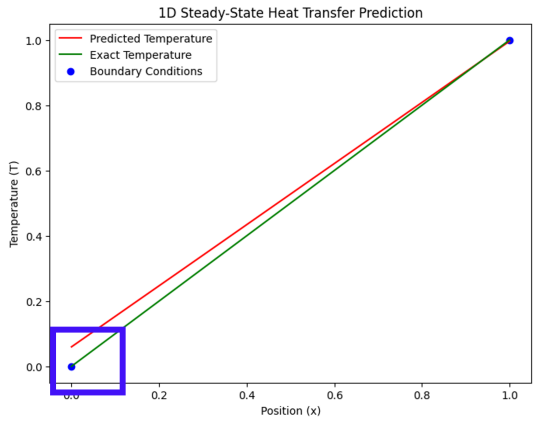
3. Results

PINN algorithm: After training the 1D steady-state and transient heat transfer problems, the temperature distribution plot and evolution of loss values along the epochs is illustrated in Figure 5. As shown, the baseline PINN was unable to predict the boundary condition at $x = 0$ well in 1D steady-state heat transfer problem. This is also confirmed in Figure 5b, which shows higher loss values of the corresponding boundary condition as well as its loss reduction rate, meaning that the optimizer was unable to minimize the corresponding loss, well.

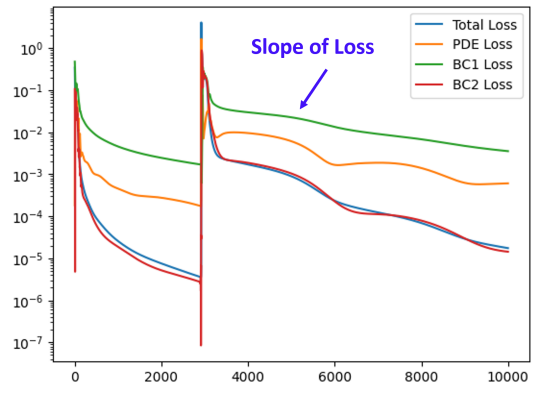
The same flaws can be found in 1D transient heat transfer problem. As shown, both boundary conditions as well as the initial condition could not be satisfied in the training process.

	Layer	1D Steady State	1D Transient	2D Transient
PINN	N_{1st}	1×1	1×2	1×3
	N_{Hi}	2×10	8×32	10×40
	N_{last}	1×1	1×1	1×1
	N_{Epochs}	10000	13000	1000
SA-PINN	N_{1st}	1×1	1×2	1×3
	N_{Hi}	2×10	8×20	10×20
	N_{last}	1×1	1×1	1×1
	N_{Epochs}	5000	900	1000
SA-PINN-PD	N_{1st}	-	1×2	-
	N_{Hi}	-	8×32	-
	N_{last}	-	1×1	-
	N_{Epochs}	-	2500	-

Table 1: The Information of Neural Networks for training the PINN, Self Adaptive Loss PINN (SA-PINN), and SA-PINN for Physics Discovery (SA-PINN-PD). The $N \times M$ in the table means N layers with M Neurons.



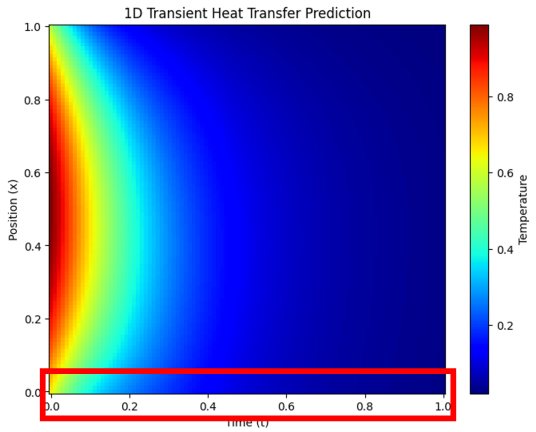
(a)



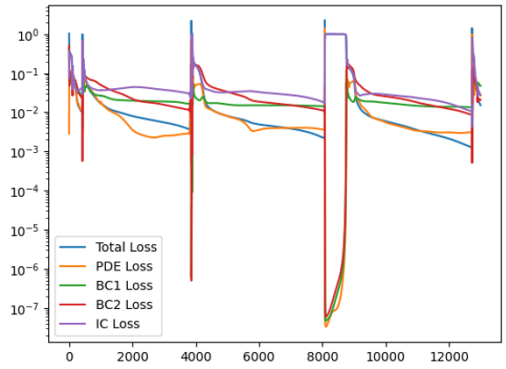
(b)

Figure 5: The results of training for 1D steady state case (a) temperature distribution (b) loss values vs epochs

It is shown in Figure 6a that the boundaries at initial time steps could not be imposed to the value set in the problem. Moreover,



(a)



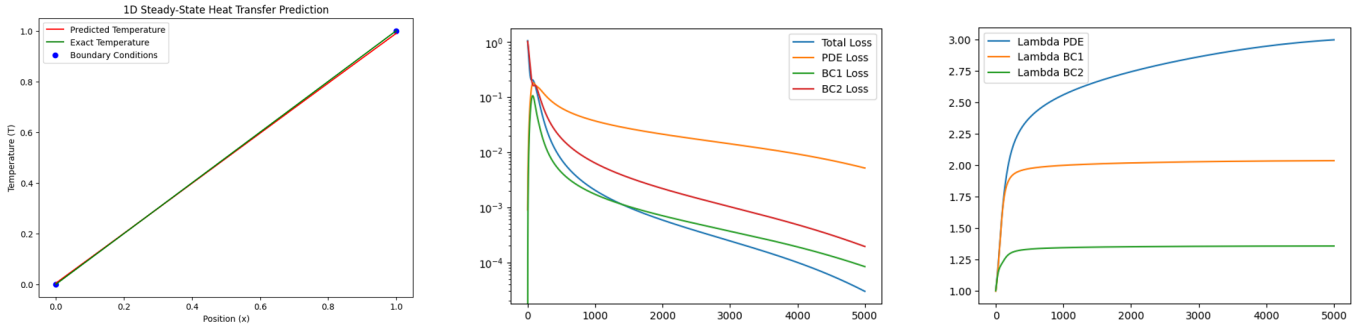
(b)

Figure 6: The results of training for 1D Transient case (a) temperature distribution (b) loss values vs epochs

the initial condition was unable to reach to its imposed value near the boundary condition x=1. The loss evolution (Figure 6) also confirms that the optimizer was unable to minimize the boundary and initial conditions' loss functions.

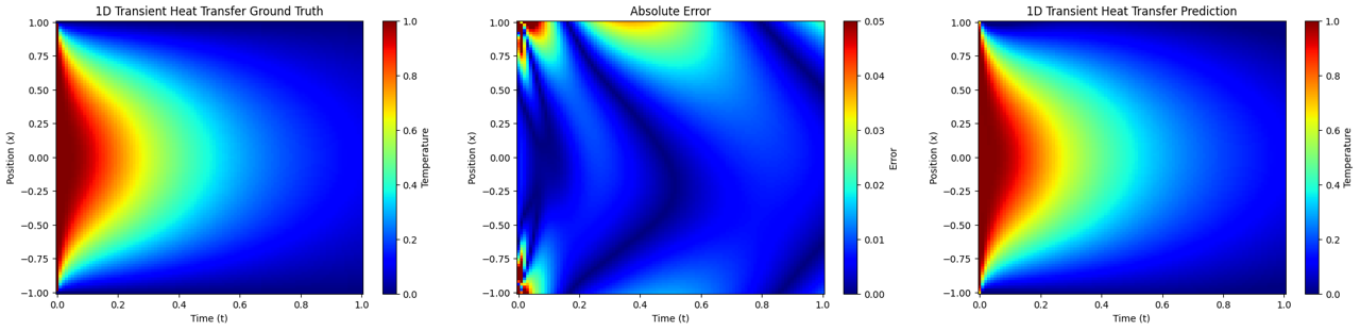
Self-adaptive weighting method: After training the 1D steady-state, 1D transient, and 2D transient heat transfer problems, the temperature distribution plot and evolution of loss values along the epochs are illustrated in Figures 7, 8, 9.

As it can be seen, implementing self-adaptive weighting method into the baseline PINN significantly increased PINN's performance in 1D steady state case. With this method, boundary conditions were imposed to their corresponding values during the training, as illustrated in Figure 7a. This is also confirmed in Figure 7b showing the rate of loss values decrease that roughly is the same for all PDE, and BC losses by using self-adaptive weighting method. The change in weight of each loss function is also illustrated in Figure 7c.

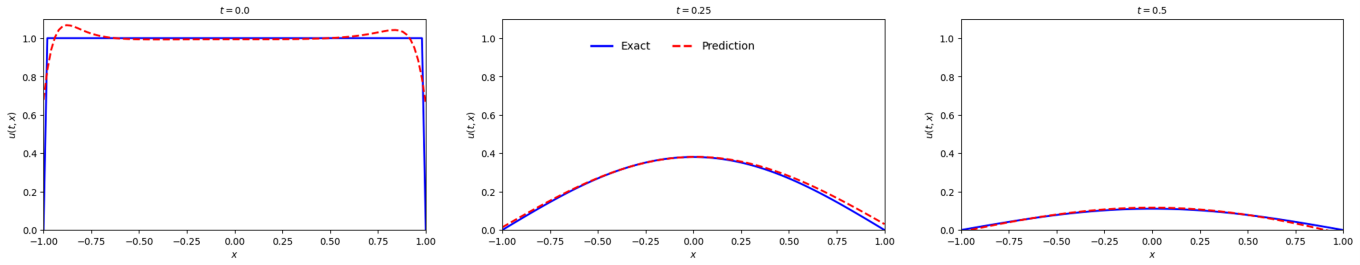


(a) (b) (c)

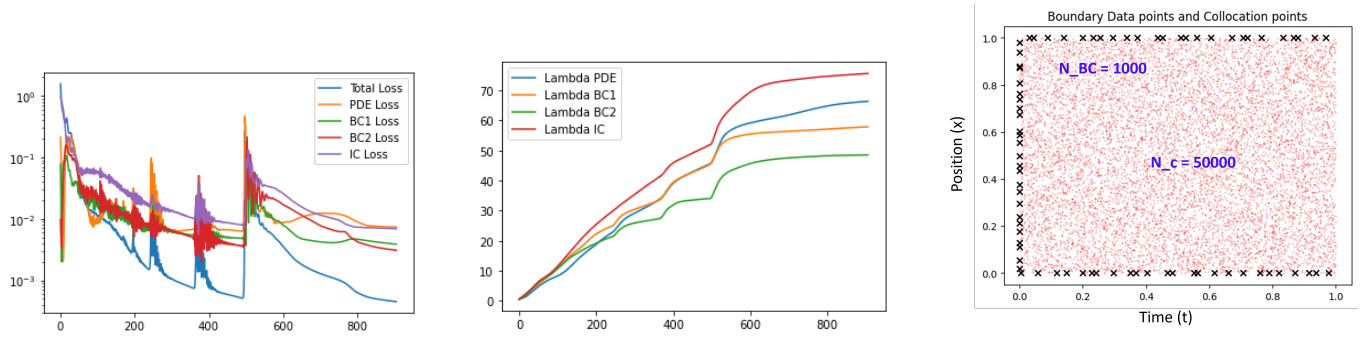
Figure 7: The results of training for 1D steady state case with SA-PINN algorithm (a) temperature distribution (b) loss values vs epochs (c) self-adaptive weights vs epochs



(a)

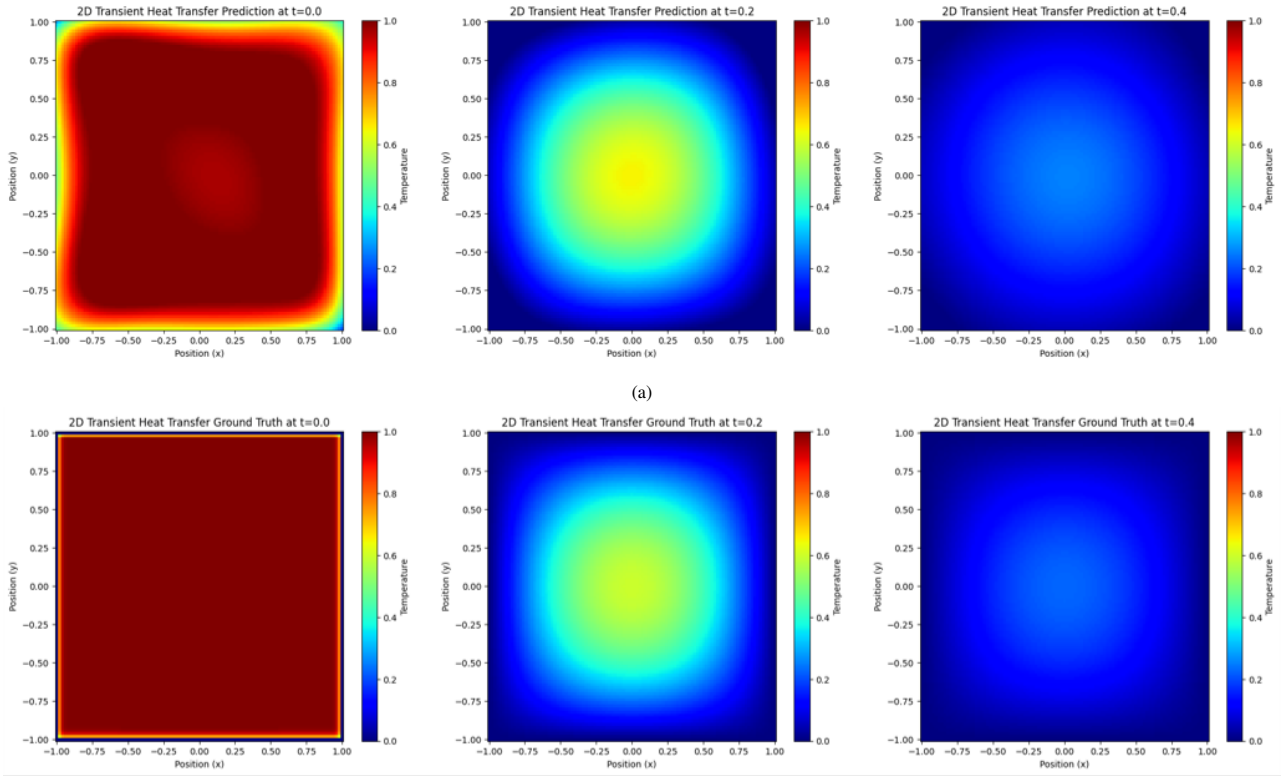


(b)



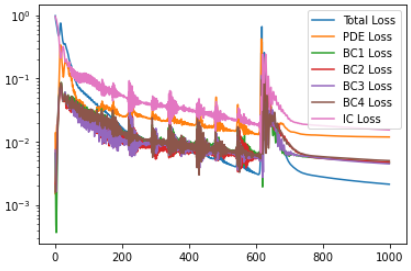
(c) (d) (e)

Figure 8: The results of training for 1D transient case with SA-PINN algorithm (a) temperature distribution for all times (b) Comparison of SA-PINN and FDM results for some time sections (c) loss values vs epochs (d) self-adaptive weights vs epochs (e) data set samples

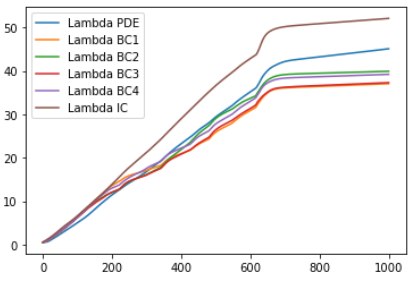


(a)

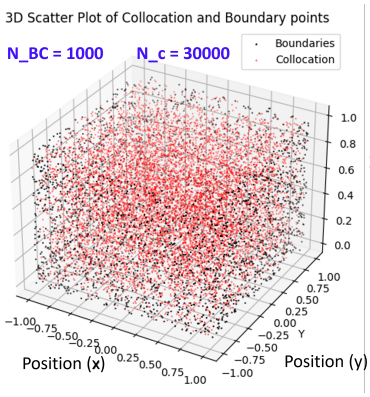
(b)



(c)

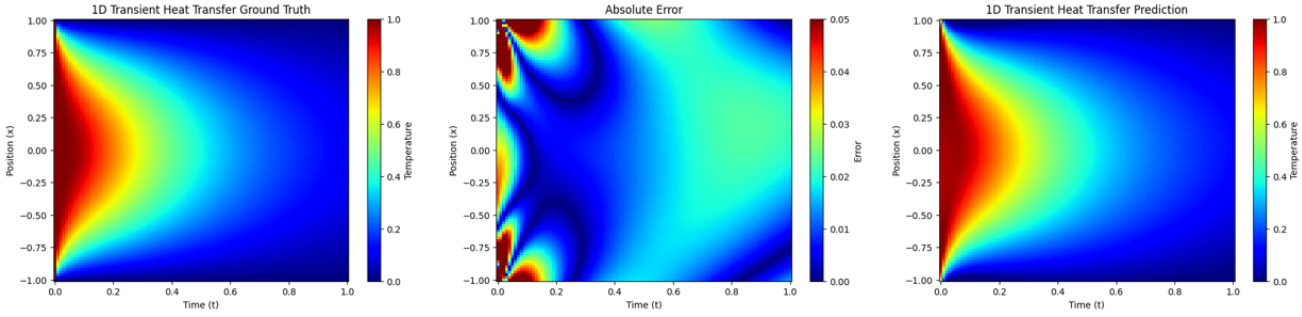


(d)



(e)

Figure 9: The results of training for 2D transient case with SA-PINN algorithm (a) temperature distribution of SA-PINN in some time sections (b) temperature distribution of FDM in some time sections (c) loss values vs epochs (d) self-adaptive weights vs epochs (e) data set samples



(a)

Figure 10: The results of training for 1D steady state case with SA-PINN algorithm for Physics Discovery (a) temperature distribution for SA-PINN and FDM results

The same increase in performance of the model prediction was seen for the 1D transient heat transfer problem. Using self-adaptive weighting method, initial and boundary conditions were imposed to their corresponding values very well, as illustrated in Figures 8, 8b . As shown in Figure 8c optimizer minimized all loss functions well with approximately same rate of minimization. The change in weight of each loss function is also illustrated in Figure 8d.

The PINN was also trained for a 2D transient heat transfer problem. Due to the low performance of the computer used for training, number of layers, neurons, or epochs could not be increased to reach to higher accuracy of the PINN model. Nevertheless, the limited number of epochs still could give a reasonably acceptable accuracy for the model's prediction. Figures 9a and 9b can show the comparison of FDM results with PINN results in three different time stamps.

PINN in the application of Physics Discovery: As mentioned earlier, PINN is implemented to investigate the performance of the PINN in the application of physics discovery. Here, PINN was used to predict the diffusivity coefficient of a 1D transient heat transfer problem. The actual value of the diffusivity coefficient, from which data points collected in order to train PINN, is 1.0. The initial value set in the process of training is 0.5. After training, the distribution and profile of temperature in domain is illustrated in Figure 10. Also the PINN prediction and error can be found in Table 2. As pointed out, PINN can be acceptably used in predicting the thermal diffusivity coefficient of the problem's material.

Correct PDE	$u_t = 1u_{xx}$
Initial PDE	$u_t = 0.5u_{xx}$
SA-PINN-PD	$u_t = 0.932u_{xx}$

Table 2: The result of SA-PINN for Physics Discovery (SA-PINN-PD) of the 1D transient heat transfer. The error is 6.8%.

4. Conclusion

In this project, we utilized Physics-Informed Neural Network (PINN) and Self-Adaptive Loss PINN to tackle the partial differential equations (PDEs) governing heat conduction in 1D steady state, 1D transient, and 2D transient scenarios. As previously highlighted, our results exhibit strong agreement with solutions obtained through the finite difference method for these PDEs. Notably, PINN displays promising efficacy in solving intricate PDEs like Navier-Stokes and Heat Transfer equations, a remarkable feat considering its meshless nature. Furthermore, PINN excels in unraveling unknown characteristics within data, such as material properties, a challenging task for traditional physics algorithms. Therefore, we affirm that physics-informed machine learning seamlessly integrates data and mathematical physics models, even in partially understood, uncertain, and high-dimensional contexts. This integration stands as a valuable and innovative tool for handling both data and complex physical phenomena.

Appendix A. FDM Algorithm and Formulation

In this appendix, we elucidate the finite difference method employed for solving the 2D transient case of heat conduction, focusing on an implicit algorithm. The approach utilizes a 1st-order approximation for the temporal term and a 2nd-order approximation for the spatial terms. Implicit methods are particularly advantageous in heat conduction simulations, as they offer increased stability and efficiency compared to explicit methods. The implicit nature of the algorithm allows for larger time steps, facilitating more computationally efficient simulations while ensuring accuracy in capturing the dynamic behavior of the system.

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \alpha \left[\frac{u_{i+1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n+1}}{\Delta x^2} + \frac{u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}}{\Delta y^2} \right] \tag{A.1}$$

The equation (A.1) gives rise to a system of algebraic equations, $Au = b$, where A is the coefficient matrix, u represents the unknown temperatures, and b is the vector comprising the right-hand side of the algebraic equations. In the context of a 2D scenario with the selection of nodes in both x and y directions as $n_x = n_y = 100$, the dimensions of vector u become $n_x n_y \times 1 = 10000 \times 1$, while the matrix A expands to $n_x n_y \times n_x n_y = 10000 \times 10000$. Consequently, solving an almost sparse matrix of this scale, such as A, becomes computationally challenging. A more viable approach involves employing a semi-implicit algorithm for the 2D case. In this configuration, the solution for one line in the $x(y)$ direction is derived using an implicit method, while the solution in the $y(x)$ direction is obtained by marching in that specific direction. By considering the implicit direction in x and the marching direction in y, the equation (A.1) undergoes modification for enhanced computational efficiency.

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \alpha \left[\frac{u_{i+1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n+1}}{\Delta x^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}}{\Delta y^2} \right] \tag{A.2}$$

Here, $u_{i,j-1}$ is determined based on the updated values from the preceding line, specifically at iteration time $n+1$. The semi-implicit algorithm is as below.

$\gamma \leftarrow \alpha \times \frac{\Delta t}{\Delta x^2}$
for $0 \le i \le nx - 2$ **do**
  $A[i,i] \leftarrow 1 + 4 \times \gamma$
  if $i \ge 1$ **then**
    $A[i, i-1] \leftarrow -\gamma$
  end if
  if $i <$ nx $- 3$ **then**
    $A[i, i+1] \leftarrow -\gamma$
  end if
end for

for $0 \leq k \leq k_max$ **do**
  for $0 \leq j \leq ny - 2$ **do**
    $R[:] \leftarrow \gamma u^{k-1}_{1\leq i\leq nx-2,j+1} + \gamma u^{k}_{1\leq i\leq nx-2,j-1} + u^{k-1}_{1\leq i\leq nx-2,j}$
    $R[0] \leftarrow R[0] + \gamma u^{k}_{x=-1}$
    $R[\mathrm{nx}-3] \leftarrow R[\mathrm{nx}-3] + \gamma u^{k}_{x=1}$
    $u' \leftarrow \mathrm{solve}(A, R)$
    $u^{k}_{1\leq i\leq nx-2,j} \leftarrow u'$
    $u^{k}_{0,j} \leftarrow u^{k}_{x=-1}$
    $u^{k}_{nx-1,j} \leftarrow u^{k}_{x=1}$
    if $j = 0$ **or** $j = \mathrm{ny} - 2$ **then**
      $u^{k}_{:,0} \leftarrow u^{k}_{y=-1}$
      $u^{k}_{:,ny-1} \leftarrow u^{k}_{y=1}$
    end if
  end for
end for

References

[1] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nature Reviews Physics 3 (6) (2021) 422–440.

[2] N. R. Council, et al., Assessing the reliability of complex models: mathematical and statistical foundations of verification, validation, and uncertainty quantification, National Academies Press, 2012.

[3] N. Zobeiry, K. D. Humfeld, A physics-informed machine learning approach for solving heat transfer equation in advanced manufacturing and engineering applications, Engineering Applications of Artificial Intelligence 101 (2021) 104232.

[4] 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, Journal of Scientific Computing 92 (3) (2022) 88.

[5] R. Maziar, P. Paris, K. G. Em, Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations, arXiv preprint arXiv: 1711.10566 (2017).

[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, Journal of Computational physics 378 (2019) 686–707.

[7] Y. Shin, J. Darbon, G. E. Karniadakis, On the convergence and generalization of physics informed neural networks, arXiv e-prints (2020) arXiv–2004.

[8] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM Journal on Scientific Computing 43 (5) (2021) A3055–A3081.

[9] S. Wang, X. Yu, P. Perdikaris, When and why pinns fail to train: A neural tangent kernel perspective, Journal of Computational Physics 449 (2022) 110768.

[10] C. L. Wight, J. Zhao, Solving allen-cahn and cahn-hilliard equations using the adaptive physics informed neural networks, arXiv preprint arXiv:2007.04542 (2020).

[11] D. Liu, Y. Wang, A dual-dimer method for training physics-constrained neural networks with minimax architecture, Neural Networks 136 (2021) 112–125. doi:https://doi.org/10.1016/j.neunet.2020.12.028.
URL https://www.sciencedirect.com/science/article/pii/S0893608020304536

[12] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, X. Tang, Residual attention network for image classification, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 3156–3164.

[13] Y. Pang, J. Xie, M. H. Khan, R. M. Anwer, F. S. Khan, L. Shao, Mask-guided attention network for occluded pedestrian detection, in: Proceedings of the IEEE/CVF international conference on computer vision, 2019, pp. 4967–4975.

[14] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).