Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Instituto de Matemática e Estatística

Vitor Bueno Entringe de Souza

Convergence analysis of physics-informed neural networks and comparison with finite difference methods

Rio de Janeiro
2024

Vitor Bueno Entringe de Souza

Convergence analysis of physics-informed neural networks and comparison with finite difference methods

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Orientadoras: Profª Dra. Cristiane Oliveira de Faria

Profª Dra. Karla Tereza Figueiredo Leite

Rio de Janeiro

2024

CATALOGAÇÃO NA FONTE

UERJ/REDE SIRIUS/BIBLIOTECA CTC/A

S729 Souza, Vitor Bueno Estringe de.
Convergence analysis of physics- informed neural network and comparison with finite difference / Vitor Bueno Estringe de Souza. – 2024.
71 f.: il.

Orientadoras: Cristiane Oliveira de Faria, Karla Tereza Figueiredo Leite.
Dissertação (Mestrado em Ciências Computacionais) - Universidade do Estado do Rio de Janeiro, Instituto de Matemática e Estatística.

1. Redes neurais (Computação) - Teses. 2. Equações diferenciais parciais - Teses. I. Faria, Cristiane Oliveira de. II. Leite, Karla Tereza Figueiredo. III. Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística. IV. Título.

CDU 004.032.26

Patricia Bello Meijinhos CRB7/5217 - Bibliotecária responsável pela elaboração da ficha catalográfica

Autorizo, apenas para fins acadêmicos e científicos, a reprodução total ou parcial desta dissertação, desde que citada a fonte.

_____ _____

Assinatura Data

Vitor Bueno Entringe de Souza

Convergence analysis of physics-informed neural networks and comparison with finite difference methods

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Aprovada em 08 de Março de 2024.
Banca Examinadora:

Profª Dra. Cristiane Oliveira de Faria (Orientadora)
Instituto de Matemática e Estatística - UERJ

Profª Dra. Karla Tereza Figueiredo Leite (Orientadora)
Instituto de Matemática e Estatística - UERJ

Profª Dra. Patrícia Nunes da Silva
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Norberto Mangiavacchi
Faculdade de Engenharia - UERJ

Prof. Dr. Fábio Pereira dos Santos
Universidade Federal do Rio de Janeiro

Prof. Dr. Rubens André Sucupira
Instituto de Matemática e Estatística - UERJ

Rio de Janeiro
2024

DEDICATÓRIA

I dedicate this work to my parents

ACKNOWLEDGMENT

First of all, I would like to thank my parents who have always supported me and given me everything I have needed throughout my life, as well as all my family who have always supported me. Nothing would have been possible without them.

To my advisors, Prof. Dr. Cristiane Oliveira de Faria and Prof. Dr. Karla Tereza Figueiredo, for all their support.

To all my friends who have been by my side during all the important moments in my life and who will always be there.

To this university, the professors, directors, coordinators and administration who provided the best environment in which to carry out this work.

To all the members of the panel for agreeing to assess this work.

To CAPES for its fundamental financial support through the master's scholarship.

ABSTRACT

SOUZA, Vitor Bueno Entringe de. V. B. E. S. *Convergence analysis of physics-informed neural networks and comparison with finite difference methods.* 2024. 71 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2024.

Partial differential equations (PDEs) are an important part of mathematics used to describe numerous physical phenomena. Analytical solutions are often not available, making the use of approximations necessary. Researchers are always trying to develop new techniques with the aim of improving understanding and providing more accurate approximations through more efficient calculations. Deep neural networks have recently been combined with prior physical knowledge from the problems to produce the physics-informed neural network (PINN), whose main contribution was a powerful way to incorporate unused knowledge into training these networks leveraging automatic differentiation techniques. This work is focused on analyzing the convergence of this method through the perspective of the discretizations used to train the networks and also the architecture of the networks, the errors analyzed from this perspective are referred to as estimation and generalization errors respectively. The generalization error was also approached by the notorious bias and variance decomposition in order to obtain a better understanding of its contribution to the error of the approximations. PINN approximations were compared with approximations obtained from finite difference methods (FDM) in three examples of differential equations: one dimensional heat equation, two dimensional heat equation and two dimensional advection diffusion equation. The experiments carried out suggested that there is a threshold of the improvement of the generalization and estimation error, suggesting that there is an optimal number of points and architectural parameters to be used. The comparisons with the FDM showed that PINN has an advantage especially in the two dimensional cases, as it does not propagate the error from one time step to the next, this is due to the fact that this technique approximates the solution of all the time steps at once, having an unprecedented advantage over FDM.

Keywords: PINN. Convergence analysis. Comparative study. FDM.

RESUMO

SOUZA,Vitor Bueno Entringe de. V. B. E. S. *Análise de convergência de redes neurais informadas por física e comparação com métodos de diferenças finitas*. 2024. 71 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2024.

As equações diferenciais parciais (EDPs) são uma parte importante da matemática usada para descrever inúmeros fenômenos físicos. Muitas vezes soluções analíticas não estão disponíveis, tornando necessário o uso de aproximações. Pesquisadores estão sempre tentando desenvolver novas técnicas com o objetivo de melhorar o entendimento e providenciar aproximações mais precisas através de cálculos mais eficientes. Redes neurais foram recentemente combinadas com conhecimento físico prévio dos problemas para produzir a rede neural informada pela física (PINN), cuja principal contribuição foi uma forma poderosa de incorporar conhecimento não utilizado no treinamento dessas redes, aproveitando técnicas de diferenciação automática. Este trabalho tem como foco analisar a convergência deste método através da perspectiva das discretizações utilizadas para treinar as redes e também da arquitetura das redes, os erros analisados sob esta perspectiva são denominados erros de estimativa e generalização respectivamente. O erro de generalização também foi abordado pela notória decomposição de vieses e variâncias, a fim de obter uma melhor compreensão da sua contribuição para o erro das aproximações. As aproximações PINN foram comparadas com aproximações obtidas a partir de métodos de diferenças finitas (FDM) em três exemplos de equações diferenciais: equação de calor unidimensional, equação de calor bidimensional e equação de difusão de advecção bidimensional. Os experimentos realizados sugeriram que existe um limiar de melhoria do erro de generalização e estimativa, sugerindo que existe um número ótimo de pontos e parâmetros arquiteturais a serem utilizados. As comparações com o FDM mostraram que o PINN tem uma vantagem principalmente nos casos bidimensionais, pois não propaga o erro de um passo de tempo para o próximo, isso se deve ao fato desta técnica aproximar a solução de todos os passos de tempo de uma só vez, tendo uma vantagem sem precedentes sobre o FDM.

Palavras-chave: PINN. Análise de convergência. Método de Diferenças Finitas. Estudo comparativo.

LIST OF ILLUSTRATIONS

Figure 1 - Errors of machine learning algorithms 14
Figure 2 - Differential control volume 19
Figure 3 - Overview of backpropagation 29
Figure 4 - Schematic representation of PINN architecture 32
Figure 5 - Bias and variance decomposition - 1D heat 41
Figure 6 - Bias and variance decomposition - 2D heat 42
Figure 7 - Finer discretization - 1D heat 44
Figure 8 - Wider network test - 1D heat 45
Figure 9 - Deeper network test - 1D heat 45
Figure 10 - Activation function test - 1D heat 46
Figure 11 - Comparison with FDM (time) - 1D heat 47
Figure 12 - Comparison with FDM (space) - 1D heat 48
Figure 13 - Visualization of training - 1D heat 49
Figure 14 - Absolute error - 1D heat 50
Figure 15 - Finer discretization - 2D heat 51
Figure 16 - Wider network test - 2D heat 52
Figure 17 - Deeper network test - 2D heat 53
Figure 18 - Activation function test - 1D heat 54
Figure 19 - Comparison with FDM (time) - 2D heat 55
Figure 20 - Comparison with FDM (space) - 2D heat 55
Figure 21 - Visualization of training - 2D heat 57
Figure 22 - Absolute error - 2D heat 58
Figure 23 - Finer discretization - Advection-Diffusion 58
Figure 24 - Residual point ratio test 60
Figure 25 - Wider network test - Advection-diffusion 61
Figure 26 - Deeper network test - Advection-diffusion 62
Figure 27 - Activation function test - Advection-diffusion 62
Figure 28 - Comparison with FDM (time) - Adv-Dif 63
Figure 29 - Comparison with FDM (space) - Adv-Dif 64
Figure 30 - Visualization of training - Adv-Dif 65
Figure 31 - Absolute error - Adv-Dif 66

LIST OF TABLES

Table 1 - Activation functions used in testing 36

SUMMARY

	INTRODUCTION	12
1	**THEORETICAL FOUNDATION**	17
1.1	**Heat Equation**	17
1.1.1	One Dimensional Heat Equation	20
1.1.2	Two Dimensional Heat Equation	21
1.2	**Two Dimensional Advection-Diffusion equation**	22
1.3	**Finite Differences Approximations**	22
1.3.1	One Dimensional Heat Equation	25
1.3.1.1	Forward Time/ Central Space Method (FTCS)	25
1.3.1.2	The Crank-nicolson Method	25
1.3.2	Two Dimensional Heat Equation	26
1.3.2.1	Forward Time/ Central Space Method (FTCS)	26
1.3.2.2	Alternating Direction Implicit Method (ADI)	26
1.3.3	Two Dimensional Advection-Diffusion equation	27
1.3.3.1	Forward Time/ Central Space Method (FTCS)	27
1.3.3.2	UpWind Method	27
1.4	**Class of NN**	28
1.5	**PINN**	29
2	**METHODOLOGY**	33
2.1	**Estimation error**	34
2.2	**Approximation error**	35
2.3	**Bias Variance Decompostion**	35
2.3.1	Decomposition of square loss	37
2.3.2	Methodology of the decomposition with PINN's	39
2.3.3	Experiments and Results	40
3	**CASE STUDY**	43
3.1	**One Dimensional Heat Problem**	43
3.1.1	Discretization influence - Heat 1D	43
3.1.2	Network Architecture Influence - Heat 1D	44
3.1.3	Numerical Results - Heat 1D	46
3.1.4	Visualization - Heat 1D	48
3.2	**Two Dimensional Heat Problem**	50
3.2.1	Discretization Influence - Heat 2D	50
3.2.2	Network Architecture Influence - Heat 2D	51
3.2.3	Numerical Results - Heat 2D	53
3.2.4	Visualization - Heat 2D	56

3.3 **Two Dimensional Advection-Diffusion** 56
3.3.1 Discretization Influence - Advection-Diffusion 56
3.3.2 Network Architecture - Advection-Diffusion 60
3.3.3 Numerical Results - Advection-Diffusion 61
3.3.4 Visualization - Advection-Diffusion 64
CONCLUSIONS 67
REFERENCES 69

INTRODUCTION

Partial Differential Equations (PDEs) are an important area of Mathematics used to describe physical and natural phenomena that involve continuous/discontinuous variations and changes in several independent or dependent variables. In most cases, the analytical solution are not available, so numerical methods based in discretization, such as the Finite Difference Method (FDM), Finite Element Method (FEM) and Finite Volume Method (FVM) are used to obtain approximation solutions for these complex systems (LEVEQUE, 2007).

However, the more faithful the modeling proposals are to reality, the need for new numerical methods that adapt to these requirements arises. Although there are many well-established methods in the area, researchers are always trying to develop new techniques that aim to improve understanding and provide more accurate and efficient calculations to approximate the solutions of these systems. Furthermore, the growth availability of more data and available Information Technology (IT) resources has impacted the most diverse areas, and here was no exception.

Among all new proposals, approaches using machine learning (ML) methods are increasingly used in all research areas. In simple terms, the standard ML approach is to obtain knowledge of the problem at hand by extracting and integrating the pattern and behavior of previously acquired data in order to be able to predict the outcome of new data. When analyzing these problems through this scope, it was noticed that limitations still remain when using purely data-driven models in real-world applications. These arise as a need for robustness, interpretability, and adherence to physical constraints.

These limitations are particularly pertinent to problems where the laws of physics and scientific principles govern the system's behavior under study. Often, in applications of this nature, data is scarce and noisy due to the high costs of experiments and the difficulty of measurement. These characteristics in data with applications like that yield models that do not have generalization capability (in the sense that it is unable to produce accurate approximations away from the data used in training) and fail to approximate the problem's solution. (HAO et al., 2022)

One of the possible causes of these generalization errors is the exclusive dependence on empirical data (generally not available in abundance) without any incorporation of the physical laws that generate the data. Since this approach marginalized an unprecedented amount of knowledge acquired through years of research, it sought to incorporate physical knowledge that was not used in the models to improve their general performance by increasing their generalization capacity.

Many methods have been proposed by researchers to integrate physical knowledge into machine learning techniques, this area of research has the name of Physics-Informed

Machine Learning. The use of deep neural networks to deal with PDEs was natural due to their well-known property of universal function approximators (HORNIK; STINCHCOMBE; WHITE, 1989). An early work (LAGARIS; LIKAS; FOTIADIS, 1998) proposed a method to approximate the solutions of differential equations using multilayer perceptrons, although it was a milestone in the field, its approximations and applicability could not be compared with solidified numerical methods. Many years later, this idea was revisited in the influential work of Raissi, Perdikaris, Karniadakis (2019), giving rise to the Physically Informed Neural Network Commonly referred to as PINNs (RAISSI; PERDIKARIS; KARNIADAKIS, 2019). This method addresses the problem of available data by making use of physical properties at points where there is no prior information about their output, minimizing the generalization error generated by data scarcity.

PINN's contribution was a simple yet powerful way to implement prior physical knowledge in the training of these networks using developments in automatic differentiation (BAYDIN et al., 2018) to provide solutions for various types of PDEs, which can be expressed in the most general form as:

$$\begin{cases} F(t,x,u(t,x);\gamma) &= 0 & t \in [0,T],\ x \in \Omega \\ I(x,u(0,x)) &= 0 & x \in \Omega \\ B(t,x,u(t,x)) &= 0 & t \in [0,T],\ x \in \partial\Omega \end{cases} \tag{1}$$

F is a general differential operator that can consist of temporal derivatives, spatial derivatives, linear and non-linear terms and γ its parameters, u represents the unknown solution, x is the position vector in the domain Ω, with boundary $\partial\Omega$, I and B denote, respectively, the initial and boundary conditions and can also consist of differential, linear or non-linear operators

The function $u(x)$ is approximated by a neural network parameterized by θ (Eq 2) (weights and bias). The objective is to find a set of parameters θ minimizing a loss function that depends on the residual of the differential equation and the boundary and initial condition of the problem. This approach aims to convert the problem of directly solving the governing equation into a loss function optimization problem.

$$u(x) \approx u_\theta(x) \tag{2}$$

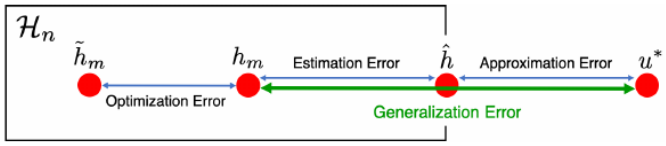
Since its proposal, many suggestions of enhancements were made to the framework, such as Conservative physics-informed neural networks (cPINN) (JAGTAP; KHARAZMI; KARNIADAKIS, 2020a), Extended physics-informed neural networks (XPINNs) (JAGTAP; KARNIADAKIS, 2020), Augmented Physics-Informed Neural Networks (APINNs) (HU et al., 2023) and many others. This work will focus on the naive approach first presented by Raissi et al (RAISSI; PERDIKARIS; KARNIADAKIS, 2019).

The neural network-based supervised learning error can be decomposed into three

components: approximation error, optimization error and estimation error. The approximation error refers to the expressiveness of the network and indicates how close the neural network can theoretically get to the solution. Estimation error can be understood as the error inherent in using a finite set of data to obtain an Approximation to a function defined in the continuous domain. Optimization error is the error obtained when updating network parameters with the objective of minimizing a loss function using some optimizer, can be seen as the distance between a gobal minimum and a local minimum of the loss function. (SHIN; DARBON; KARNIADAKIS, 2020)

The figure 1 shows an illustration of the total errors, where $\mathcal{H}_n$ is the class of the approximation function, u^* is the PDE solution, $\hat{h}$ is the function of the class $\mathcal{H}_n$ that was approximated with an infinite amount of data, as if it were possible to represent the entire domain of the problem instead of using a discretization. h_m is an approximation made with a finite number of data points m, and $\tilde{h}_m$ is the approximation obtained by applying an optimization process.

Figure 1 - Errors of machine learning algorithms



Legend: Illustration of the total errors extracted from (SHIN; DARBON; KARNIADAKIS, 2020)

The estimation error combined with the approximation error constitutes the generalization error. In machine learning this error refers to a measure of the model's accuracy on unseen data. When approximating solution for PDEs, the generalization error can be interpreted as the distance of the approximate solution generated by a network whose optimization process leads to the global minimum of the loss function in the parameter space and the PDE analytical solution(SHIN; DARBON; KARNIADAKIS, 2020).

Naturally, researchers are always trying to understand and improve the errors involved in these approximations and many studies are being done to mitigate these errors independently.

Some of the research addressing the approximation error aims to apply different architecture of neural networks used to approximate the solutions. Even using a regular MLP, determine the optimal number of parameters (weights and biases) to approximate a specific PDE can be a difficult task. Identifying and measuring the impact of a specific network architecture is a challenge, and many authors have used different architectures to approximate specific PDE solutions (YANG; ZHANG; KARNIADAKIS, 2020; JAGTAP;

KHARAZMI; KARNIADAKIS, 2020b; GAO; SUN; WANG, 2021; VIANA et al., 2021).

As the optimization error, PINNs have a training process that is significantly different from regular neural networks, so current optimizers and loss functions may not be ideal for this method. Many authors propose new loss functions, different initializations of the weight and bias of networks, and proposing the use of different optimization algorithms to calculate and update network parameters that improve the convergence speed and performance of networks (ZHU; LIU; YAN, 2021; YANG; ZHANG; KARNIADAKIS, 2020; JAGTAP; KHARAZMI; KARNIADAKIS, 2020b; NABIAN; GLADSTONE; MEIDANI, 2021).

The objective of this work is to evaluate the approximation to the solution of the PDE from the perspective of the estimation error and the approximation error (as in Figure 1). The estimation error is analyzed to understand how the approximation of the solution is affected by the discretization used to train the network, while the approximation error is analyzed to understand how the approximation of the solution is affected by the class of neural networks chosen. An optimal number of points and architectural parameters are found through the tests carried out. With these in hand, the objective is to compare the accuracy of the optimum PINN approximation with some finite difference methods in each problem.

Bias and variance decomposition is an important analysis for understanding the generalization capability of a machine learning algorithm (HASTIE et al., 2009; BISHOP, 2006). It has been suggested that the relationship of these two measurements in the context of neural networks follows the same trade-off relationship as most machine learning algorithms (GEMAN; BIENENSTOCK; DOURSAT, 1992).

Recent work has suggested that this relationship may not be present in the analysis of error obtained in neural networks (NEAL et al., 2018; NEAL, 2019). Although PINN's training is different from most neural network applications, it is of interest to know whether this decomposition is, in fact, valid for the PINN methodology and, if so, how this relationship can help to understand the error in approximations to solutions to PDE's.

PINns are usually trained with points obtained from some expected value of some random variable that follows a probability density function, that is in contrast with some of the well-established numerical methods, in these it is necessary to perform regular discretization in the PDE domain. Although PINNs do not require these conditions, in the interest of comparing them to some of these numerical methods in settings as equal as possible, these discretizations were used to produce Approximations. This is a way of understanding and comparing the PINN convergence as a function of the number of points with a discretization equal to that used in numerical methods.

This work is organized as follows. In section 1 the model problem that will be used as an example for PINNs and FDM is presented. In section 1.3 finite difference

approximations are presented together with the finite different equations used in the methods applied to the model problem. In section 1.4 the background knowledge of the neural networks used is presented followed by section 1.5 where the operation details of PINNs are detailed. Section 2 presents the methodology followed in the experiments which is divided in the sections 2.1 and 2.2 addressing the estimation and approximation error respectively from the perspective of PINNs. Section 2.3 presents the study and experiments of the bias and variance decomposition. The case study is presented in section 3 divided into: one dimensional heat problem in section 3.1.1, two dimensional heat problem in section 3.2 and the two dimensional addiction-diffusion problem in section 3.3. Each case study is separated in four sections each addressing: the approximation error, generalization error, comparisons with the FDM and visualization of the training process. Finally, section 3.3.4 presents some conclusions from the experiments.

1 THEORETICAL FOUNDATION

1.1 Heat Equation

The objective of this section is to deduce the heat equation, this was based on the works of (INCROPERA et al., 1996), where it can be found more details and in depth knowledge about the dynamics of heat transfer.

Heat transfer is thermal energy in motion, it will always occur when there is a difference in temperature in a region. The process of conduction occurs when there is a temperature difference between a solid or stationary fluid and heat is diffused through it.

It is possible to interpret the transfer of energy from particles with more energy to particles with less energy in a solid or stationary fluid due to the interaction of their particles. The temperature at any point is associated with the energy of the molecules in the vicinity of that point. This energy is related to the random translational movement, as well as the internal rotation and vibration movements of the molecules.

Higher temperatures are associated with higher molecular energy, when neighboring molecules collide there is a transfer of energy from molecules with more energy to molecules with less. In the presence of a temperature difference, energy transfer by conduction must occur in the direction of decreasing temperature.

It is possible to quantify the heat transfer process in terms of appropriate rate equations, these equations can be used to calculate the amount of energy being transferred per time unit, for conduction the rate equation is known as Fourier's law (equation 3)

$$q''_x = -k\frac{dT}{dx} \tag{3}$$

The heat flux q''_x is the heat transfer rate in the x direction per area unit perpendicular to the transfer direction and is proportional to the temperature gradient $\frac{dT}{dx}$ in this direction. The parameter k is the thermal conductivity of the material. The negative sign is a consequence of heat transfer in the direction of decreasing temperature

Fourier's law is phenomological, that is, this law was developed through the observation of phenomena, for this reason the rate equation can be seen as a generalization based on vast experimental evidence. It was observed that the amount of heat transferred in a cylindrical region with its insulated surface and its end at different temperatures is proportional to the sectional area of the cylinder, the difference in temperature of its ends and inversely proportional to the length of the cylinder.

This proportionality was transformed into equality when it was observed that different materials subjected to the same experiments under the same conditions presented different heat transfer rates, therefore a coefficient was added that describes the behavior

of the material resulting in the equation 4 when the section length tends to zero.

$$q_x = -kA\frac{dT}{dx} \tag{4}$$

The negative sign is necessary to indicate that heat is always transferred towards decreasing temperature. Heat flow is defined as in the equation 5 which is Fourier's law as it is widely known

$$q''_x = \frac{q_x}{A} = -k\frac{dT}{dx} \tag{5}$$

As written, Fourier's law implies that heat flow is a directional magnitude, in particular the direction of q''_x is normal to the cross-sectional area A, more generally the direction of flow will always be normal to a surface of constant temperature.

Recognizing heat flow as a vector quantity, it is possible to write a more general form of the Fourier law as in equation 6

$$q'' = -k\left(i\frac{\partial T}{\partial x} + j\frac{\partial T}{\partial y} + k\frac{\partial T}{\partial z}\right) \tag{6}$$

It is possible to decompose the equation 6 into three expressions, each relating the heat flow through a surface to the temperature gradient in a direction perpendicular to the surface as in the equation 7

$$q''_x = -k\frac{\partial T}{\partial x} \qquad q''_y = -k\frac{\partial T}{\partial y} \qquad q''_z = -k\frac{\partial T}{\partial z} \tag{7}$$

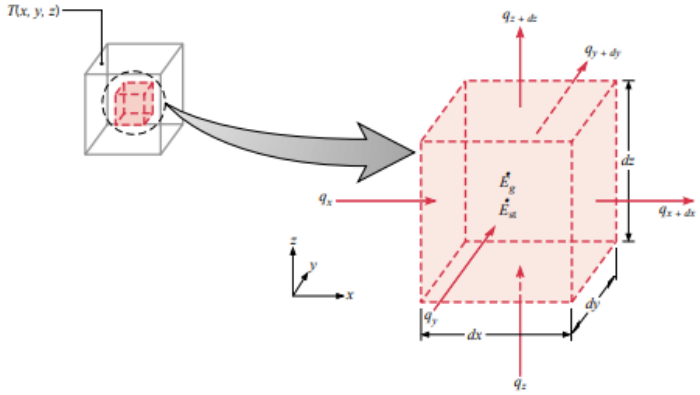
Fourier's law is fundamental to heat transfer by conduction. This expression defines an important property of materials, thermal conductivity, and is applicable to all matter, regardless of its state.

To consider the general way in which the temperature distribution can be determined, it is necessary to follow the energy conservation methodology. First defining a differential control volume considering a homogeneous medium within which there is no macroscopic movement and the temperature distribution is represented in Cartesian coordinates, an infinitesimally small volume ($dxdydz$) is defined as in figure 2.

If there are temperature gradients, there will be conductive heat transfer across each of the control surfaces. The heat transfer rates are perpendicular to each of the control surfaces at the coordinate positions x, y and z and are denoted by the terms q_x, q_y and q_z respectively. The rate of heat transfer by conduction at the opposite surface can be expressed as a Taylor expansion ignoring higher order terms as in equation 8

$$\begin{aligned} q_{x+dx} &= q_x + \tfrac{\partial q_x}{\partial x}dx \\ q_{y+dy} &= q_y + \tfrac{\partial q_y}{\partial y}dy \\ q_{z+dz} &= q_z + \tfrac{\partial q_z}{\partial z}dz \end{aligned} \tag{8}$$

Figure 2 - Differential control volume



Legend: Illustration extracted from (INCROPERA et al., 1996)

These equations (Eq. 8) state that the x component of the heat transfer rate at position $x + dx$ is equal to the value of the x component plus the amount it changes multiplied by dx

There may also be variations in the thermal energy stored by the matter within the control volume, the term referring to the energy accumulation can be written as equation 9

$$E_{acu} = \rho c_p \frac{\partial T}{\partial t} dxdydz \tag{9}$$

The expression refers to the variation in thermal energy accumulated by matter, $\rho c_p \partial T \partial t$ is the rate of variability of sensible energy in the medium in relation to time per unit volume

The last step in the energy conservation methodology is to represent the conservation using the rate equations developed, knowing that the rate of heat conduction that constitutes the energy input as E_{en} and the energy exit as E_{ex}, the conservation can be exrpessed as equation 10.

$$E_{en} - E_{ex} = E_{acu} \Rightarrow q_x + q_y + q_z - q_{x+dx} - q_{y+dy} - q_{z+dz} = \rho c_p \frac{\partial T}{\partial t} dxdydz \tag{10}$$

Using the terms of the equations 8 and 9 it is possible to write the equation 10 as 11

$$-\frac{\partial q_x}{\partial x} dx - \frac{\partial q_y}{\partial y} dy - \frac{\partial q_z}{\partial z} dz = \rho c_p \frac{\partial T}{\partial t} dxdydz \tag{11}$$

The rate of heat conduction by conduction can be determined by Fourier's law, where each heat flow component is multiplied by the area of the respective control surface

to obtain the heat transfer rate as in equation 12

$$\begin{aligned} q_x &= -kdydz\frac{\partial T}{\partial x} \\ q_y &= -kdxdz\frac{\partial T}{\partial y} \\ q_z &= -kdxdy\frac{\partial T}{\partial z} \end{aligned} \tag{12}$$

To obtain the heat equation it is necessary to replace all terms in equation 11 by the terms defined by the Fourier's law in equation 12 and divide by the dimensions of the control volume resulting in the equation 13

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right) + \frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right) = \rho c_p \frac{\partial T}{\partial t} \tag{13}$$

The equation 13 is the general form of heat diffusion in Cartesian coordinates. Often called the heat equation, it provides the basic tool for heat conduction analysis. Through its solution it is possible to obtain a temperature distribution as a function of time.

It is often possible to work with a simplified version of Equation 13, if the thermal conductivity is constant, the heat equation can be written as Equation 14, where $\alpha^{-1} = k/\rho c_p$ is the thermal diffusivity.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} = \alpha \frac{\partial T}{\partial t} \tag{14}$$

the solution to the equation 13 depends on the physical conditions that exist in the border regions of the medium, and also, if the situation varies as a function of time, the solution will also depend on the conditions of the medium at some initial instant. There are many possibilities of border conditions (or boundary conditions) that can be expressed mathematically.

The boundary condition that will be used the examples is the Dirichlet boundary condition, which corresponds to keeping the edges of the problem surface at a fixed temperature.

1.1.1 One Dimensional Heat Equation

The problem used as an example in one dimension is the homogeneous one-dimensional heat equation with the Dirichlet boundary condition and initial condition given by a func-

tion of x. The problem set up is in display in the Eq. 15.

$$\begin{cases} \dfrac{\partial u}{\partial t} = \alpha \dfrac{\partial^2 u}{\partial x^2} \qquad x \in (0, L),\ t > 0 \\ \\ u(t, 0) = u(t, L) = 0 \qquad t > 0 \\ u(0, x) = sen(\pi x) \qquad x \in (0, L) \end{cases} \tag{15}$$

The goal is to model the flow of heat (thermal energy) in an object model with a one-dimensional differential equation, $u(t, x)$ denotes the temperature at point x at time t, where x is bounded to the finite length of the object, α is the heat conduction coefficient or thermal diffusivity of the object ((LEVEQUE, 2007)).

In this particular setting it will be used $\alpha = 1$, $L = 1$ and it is of interest to know how the temperature at each point x varies as a function of t up to 0.2.

1.1.2 Two Dimensional Heat Equation

The problem used as an example in two dimensions is the extension of the problem in one dimension, the homogeneous two-dimensional heat equation with the Dirichlet boundary condition and initial condition given by a function of x and y. The problem set up is in display in the Eq. 16.

$$\begin{cases} \dfrac{\partial u}{\partial t} = \alpha \left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} \right) \qquad x, y \in R,\ t > 0 \\ u(t, x, 0) = u(t, x, L) = 0 \qquad 0 \leq x \leq L,\ t > 0 \\ u(t, 0, y) = u(t, L, y) = 0 \qquad 0 \leq y \leq L,\ t > 0 \\ u(0, x, y) = sen(\pi x) sen(\pi y) \qquad x, y \in R \\ \text{Where } R = [0, L]\ x\ [0, L] \end{cases} \tag{16}$$

The goal is to model the flow of heat (thermal energy) in an object model with a one-dimensional differential equation, $u(t, x)$ denotes the temperature at point x at time t, where x is bounded to the finite length of the object, *alpha* is the heat conduction coefficient or thermal diffusivity of the object ((LEVEQUE, 2007)).

In this particular setting it will be used $\alpha = 1$, $L = 1$ and it is of interest to know how the temperature at each point x varies as a function of t up to 0.2 .

1.2 Two Dimensional Advection-Diffusion equation

The final exemple that will be addressed is the time dependent two dimensional advection-diffusion equation, which is the basis for many physical and chemical phenomena. It is an essential PDE for modeling the transport of long-range air pollution and wind-flow in the atmosphere (SALMAN et al., 2022), the pollutants into the aquatic environment (WONGSAIJAI; SUKANTAMALA; POOCHINAPAN, 2020), among others. The general form of this PDE is on display in equation 17

$$\frac{\partial u}{\partial t} - D_x\frac{\partial^2 u}{\partial x^2} - D_y\frac{\partial^2 u}{\partial y^2} + V_x\frac{\partial u}{\partial x} + V_y\frac{\partial u}{\partial y} = 0 \tag{17}$$

This is the time dependent advection diffusion PDE in a two dimensional form. While the diffusion equation, as stated previously, is parabolic, advection is hyperbolic, the advection diffusion equation, naturally, combines both. The solution to the problem is the function $u(t, x, y)$ that yields the fluid mass/concentration of a certain point (x, y) in the space domain at some moment in time t. D_x and D_y represents the coefficients of diffusion along the respective axes and V_x and V_y is the velocity in the respective axes.

To test and compare the approximations provided by PINN the problem chosen has the analytical solution on display in equation 18 (NOYE; TAN, 1989). The space domain considered will be a squared region $x \in [0, 2]$ and $y \in [0, 2]$, while the initial condition and boundary conditions are obtained directly from the equation. Dirichlet boundary conditions are assumed for this squared region. The constants D_x and D_y are equal to 0.01 and V_x and V_y are equal to 0.8. The goal is to model the phenomena up to $t = 1.25$.

$$u(t, x, y) = \frac{1}{4t+1}\, exp\left[-\frac{(x - ut - 0.5)^2}{D_x(4t+1)} - \frac{(y - vt - 0.5)^2}{D_y(4t+1)}\right] \tag{18}$$

This specific problem is addressed by several works that propose different finite difference schemes and serves as a reference for new methods in the area. (SHALLAL; JUMAA, 2016; KARAA; ZHANG, 2004; WONGSAIJAI; SUKANTAMALA; POOCHINAPAN, 2020; TIAN; GE, 2007; KALITA; DALAL; DASS, 2002) This problem was also solved in the work of Salman et al. (2022) using the PINN methodology, the objective is to provide further analysis on how the accuracy of the approximation is conditioned to the discretization of the domain and the architecture of the network used.

1.3 Finite Differences Approximations

The purpose of these methods is to approximate solutions to differential equations, this is done by making some discrete approximation to the functions that solve the diffe-

rential equation. This equation represents a relationship between several of its derivatives in some given region of space and/or time along with some boundary conditions along the edges of this domain. (LEVEQUE, 2007)

In general, analytical solutions to these problems are rarely available, thus it is often necessary to approximate its solution. A finite difference method does that by replacing the derivatives in the differential equation with some finite difference approximations. This approach provides a large but finite set of algebraic equations to be solved in place of the differential equation. (LEVEQUE, 2007)

A frequently used method for approximation is the Taylor series expansion of a function, this is what will be focused on at frist. The notation and development of this section were based on the works of (HOFFMANN; CHIANG, 1998).

Given a function $f(x)$, $f(x + \Delta x)$ can be expanded into a Taylor series about x as Eq 19

$$f(x + \Delta x) = f(x) + \sum_{n=1}^{\infty} \frac{(\Delta x)^n}{n!} \frac{\partial^n f}{\partial x^n} \tag{19}$$

Solving for $\frac{\partial f}{\partial x}$ it is obtained Eq 20

$$\frac{\partial f}{\partial x} = \frac{f(x + \Delta x) - f(x)}{\Delta x} - \frac{\Delta x}{2!} \frac{\partial^2 f}{\partial x^2} - \frac{(\Delta x)^2}{3!} \frac{\partial^3 f}{\partial x^3} + \dots \tag{20}$$

It is common to sum all the terms with factors of Δx and higher and represent them as $O(\Delta x)$, these terms are read as terms of order Δx, which means that the behaviour of the rest of the equation as $\Delta x \to 0$ is linear in this case.

$$\frac{\partial f}{\partial x}|_i = \frac{f_{i+1} - f_i}{\Delta x} + O(\Delta x) \tag{21}$$

Equation 21 is an Approximation for the frist partial derivative of f with respect to x using the subscript i to represent the discrete points in the x direction. This equation is known as the frist forward difference Approximation of $\frac{\partial f}{\partial x}$ of order Δx

$$\frac{\partial f}{\partial x}|_i = \frac{f_i - f_{i-1}}{\Delta x} + O(\Delta x) \tag{22}$$

The same can be done considereing the Taylor series expasion of $f(x - \Delta x)$ and solving for $\frac{\partial f}{\partial x}$. This equation (equation 22) is know as the frist backward difference Approximation of $\frac{\partial f}{\partial x}$ of order Δx.

Subtraction the Taylor series expansion that originated both difference approximation (equations 21 and 22) and solving for $\frac{\partial f}{\partial x}$ results in the equation know as the central difference Approximation of order $(\Delta x)^2$ (equation 23) which means that the behaviour of the rest of the equation as $\Delta x \to 0$ is quadratic.

$$\frac{\partial f}{\partial x}|_i = \frac{f_{i+1} - f_{i-1}}{2\Delta x} + O(\Delta x)^2 \tag{23}$$

Similarly, it is possible to consider both Taylor series expansions of $f(x+\Delta x)$ and $f(x+2\Delta x)$, subtracting them and solving for $\frac{\partial^2 f}{\partial x^2}$ to get the equation 24. This equation represents the forward differentce approximation to the second derivative of f with respect to x and is of order (Δx).

$$\frac{\partial^2 f}{\partial x^2}|_i = \frac{f_{i+2} - 2f_{i+1} + f_i}{(\Delta x)^2} + O(\Delta x) \tag{24}$$

Likewise, an approximation can be produced using the Taylor series expansions of $f(x-\Delta x)$ and $f(x-2\Delta x)$, resulting in Eq 25, the backward difference Approximation of of $frac\partial^2 f\partial x^2$.

$$\frac{\partial^2 f}{\partial x^2}|_i = \frac{f_i - 2f_{i-1} + f_{i-2}}{(\Delta x)^2} + O(\Delta x) \tag{25}$$

To obtain a central difference approximation of the second derivative it is only necessary to add both equations (25 e 24), resulting in equation 26.

$$\frac{\partial^2 f}{\partial x^2}|_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{(\Delta x)^2} + O(\Delta x)^2 \tag{26}$$

The finite difference approximations discussed are used to replace derivatives that appear in partial differential equations. The one dimensional heat equation (equation 15) will be used as an example. It is necessary to approximate the PDE by a finite difference equation in a domain with equal grid spacing, the subscript i will be used to represent the cartesian coordinate of x while the superscript n will be used to represent the time. In equation 27 it is possible to see the approximation of the first-order finite difference Approximation in time and the second-order central difference Approximation in space.

$$\frac{\partial u}{\partial t} = \frac{u_i^{n+1} - u_i^n}{\Delta t} + O(\Delta t); \qquad \frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1}^n - u_i^n + u_{i-1}^n}{(\Delta x)^2} + O(\Delta x)^2 \tag{27}$$

The finite difference formulation of equation 27 takes the form of Eq 28. It is important to note that the spatial approximations are applied at the time level n, but there is also a formulation that the spatial approximations are applied at the time level $n+1$, taking the form of the equation 28.

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = k\frac{u_{i+1}^{n+1} - u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2} + O[\Delta t, (\Delta x)^2] \tag{28}$$

Equation 27 is classified as an explicit formulation while Eq 28 is classified as an implicit formulation. The number of unknowns is the main difference between both formulations, the explicit one contains only one unknown (u_i^{n+1}) while the implicit one involves three unknowns $(u_{i-1}^{n+1}, u_i^{n+1}, u_{i+1}^{n+1})$. Naturally, the process to obtain the solutions is different in both cases.

Using the explicit formulation it is possible to solve the one unknown directly at

each grid point. When using the implicit formulation, it is necessary to write the finite difference equation for all spatial grid points at $n+1$ time levels to give the same number of equations as there are unknowns to solve simultaneously through a system.

1.3.1 One Dimensional Heat Equation

1.3.1.1 Forward Time/ Central Space Method (FTCS)

As seen in Eq 27, the second-order PDE was transformed into a finite difference equation by replacing $\frac{\partial u}{\partial t}$ with a forward difference approximation of order Δt (equation. 21) and replacing $\frac{\partial^2 u}{\partial x^2}$ with a second-order central difference of order $(\Delta x)^2$ (equation 26). Due to the fact that this equation has only one unknown u_i^{n+1}, it can be calculated from the algebraic equation in Eq 29.

$$u_i^{n+1} = u_i^n + \alpha \frac{\Delta t}{(\Delta x)^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \tag{29}$$

The dependent variable at the time level n is known from a previous solution or given as an initial condition, the calculated values at $n+1$ depend only on the past history. To start the solution, an initial condition and two boundary conditions must be specified. As each finite difference equation involves only one unknown, the resulting equations at the time level $n+1$ are solved independently to give the values of the unknown.

1.3.1.2 The Crank-nicolson Method

It is also possible to replace the second-order term in Eq 15 by the average of the central difference at time levels n and $n+1$ giving the discretized equation the form of Eq 30

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \left(\frac{1}{2}\right) \left[\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2} + \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2}\right] \tag{30}$$

This method is defined as implicit, as there are more than one unknown in the finite difference equation. As a consequence, It is necessary to solve a set of equations simultaneous, which require more calculations per time step.

The advantage of this method lies in its stability, being unconditionally stable means that it is possible to use larger time steps.

1.3.2 Two Dimensional Heat Equation

1.3.2.1 Forward Time/ Central Space Method (FTCS)

For the heat equation in two dimensions, an explicit equation can be obtained using forward differentiation for the time derivative and central differentiation for the space derivatives, giving an equation in the form of equation 31

$$\frac{u_{ij}^{n+1} - u_{ij}^{n}}{\Delta t} = \alpha \left[\frac{u_{i+1j}^{n} - 2u_{ij}^{n} + u_{i-1j}^{n}}{(\Delta x)^2} + \frac{u_{ij+1}^{n} - 2u_{ij}^{n} + u_{ij-1}^{n}}{(\Delta y)^2} \right] \tag{31}$$

Since the only unknown in this formulation is the term u_{ij}^{n+1}, each grid point can be easily calculated by the equation 32

$$u_{ij}^{n+1} = u_{ij}^{n} + \frac{\alpha \Delta t}{(\Delta x)^2}[u_{i+1j}^{n} - 2u_{ij}^{n} + u_{i-1j}^{n}] + \frac{\alpha \Delta t}{(\Delta y)^2}[u_{ij+1}^{n} - 2u_{ij}^{n} + u_{ij-1}^{n}] \tag{32}$$

1.3.2.2 Alternating Direction Implicit Method (ADI)

Implicit methods for solving two-dimensional PDEs tend to yield systems at each time step that are very large and time-consuming to obtain a solution, many methods have tried to overcome this difficulty by introducing other approximations to the diffusion term, one of the most popular methods of the Alternating Direction Implicit method.

The ADI method is known as a simple but effective methof for solving two-dimensional parabolic problems (LEVEQUE, 2007). The main idea is to divide the direction in which the implicit terms are used, first solving for the x-direction implicitly and the y-direction explicitly and then, using the results obtained, solving for the y-direction implicitly and the x-direction explicitly

Initially, the PDE is approximated using the forward differencing for the time derivative and the central differencing for the space derivatives, from which two equations emerge to be solved sequentially, at the time level n, $n + 1/2$ and for $n + 1$. Equation 33 is implicit in the x-direction and explicit in the y-direction, the solution in this stage is called the x-sweep.

$$\frac{u_{ij}^{n+\frac{1}{2}} - u_{ij}^{n}}{\frac{\Delta t}{2}} = \alpha \left[\frac{u_{i+1j}^{n+\frac{1}{2}} - 2u_{ij}^{n+\frac{1}{2}} + u_{i-1j}^{n+\frac{1}{2}}}{(\Delta x)^2} + \frac{u_{ij+1}^{n} - 2u_{ij}^{n} + u_{ij-1}^{n}}{(\Delta y)^2} \right] \tag{33}$$

Equation 34 is implicit in the y-direction and explicit in the x-direction, the solution in this stage is called the y-sweep.

$$\frac{u_{ij}^{n+1} - u_{ij}^{n+\frac{1}{2}}}{\frac{\Delta t}{2}} = \alpha \left[\frac{u_{i+1j}^{n+\frac{1}{2}} - 2u_{ij}^{n+\frac{1}{2}} + u_{i-1j}^{n+\frac{1}{2}}}{(\Delta x)^2} + \frac{u_{ij+1}^{n+1} - 2u_{ij}^{n+1} + u_{ij-1}^{n+1}}{(\Delta y)^2} \right] \tag{34}$$

This method produces two sets of tridiagonal systems to be solved in sequence for each equation.

1.3.3 Two Dimensional Advection-Diffusion equation

1.3.3.1 Forward Time/ Central Space Method (FTCS)

As previously, for the two dimensional advection-diffusion equation an explicit equation can be obtained using forward differentiation for the time derivative and second order central differentiation for the space derivatives, giving an equation in the form of equation 35

$$\begin{aligned} \frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} - D \left[\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{\Delta x^2} + \frac{u_{i,j+1-}^{n} 2u_{i,j}^{n} + u_{i,j-1}^{n}}{\Delta y^2} \right] + \\ + V \left[\frac{u_{i+1,j}^{n} - u_{i-1,j}^{n}}{2\Delta x} + \frac{u_{i,j+1}^{n} - u_{i,j-1}^{n}}{2\Delta y} \right] = 0 \end{aligned} \tag{35}$$

1.3.3.2 UpWind Method

Another explicit method implemented is the UpWind method. This is method similar to the FTCS method as it also uses the forward differentiation for the time derivative and the second order centered space for the diffusion term, it differs in the advection term where, instead of also using a second-order centered space approximation, it uses a first-order backward space approximation. The equation 36 is the result.

$$\begin{aligned} \frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} - D \left[\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{\Delta x^2} + \frac{u_{i,j+1-}^{n} 2u_{i,j}^{n} + u_{i,j-1}^{n}}{\Delta y^2} \right] + \\ + V \left[\frac{u_{i,j}^{n} - u_{i-1,j}^{n}}{\Delta x} + \frac{u_{i,j}^{n} - u_{i,j-1}^{n}}{\Delta y} \right] = 0 \end{aligned} \tag{36}$$

1.4 Class of NN

Various types of architectures were used in the PINN framework (CUOMO et al., 2022), in the original work (RAISSI; PERDIKARIS; KARNIADAKIS, 2019) PINN was proposed using a fully connected feed forward deep neural network. A generic deep neural network with L layers can be represented as the composition of L functions $f_i(x_i, \theta_i)$, where x_i denotes the inputs of layer i, the approximation $u_\theta(x)$ can be expressed as Eq 37.

$$u_\theta(x) = f_L \circ f_{L-1} \circ \cdots \circ f_1(x) \tag{37}$$

The layer composition is expressed by $\circ$ and should be read as $f_{(l)} \circ f_{l-1}(x) = f_l(f_{l-1}(x))$. The network input is $x \in \mathbb{R}^d$ and let $g_0(x) = x$, it is possible to define each layer f_i by Eq 38.

$$f_i(x) = W_i \,.\, g_{i-1} \;+\; b_i, \qquad g_i(x) = \sigma(f_i(x)), \qquad i = 1, 2, \ldots, L \tag{38}$$

Each hidden layer of the network has size $(K_1, \ldots, K_L)$, and $W_i \in \mathbb{R}^{K_{i-1} \times K_i}$ is the matrix of weights, $b_i \in \mathbb{R}^{K_i}$ is the bias of each layer and σ is the an element wise activation function. In this setup, $\theta = (W_1, b_1, \ldots, W_L, b_L)$ represents all trainable parameters of the network.

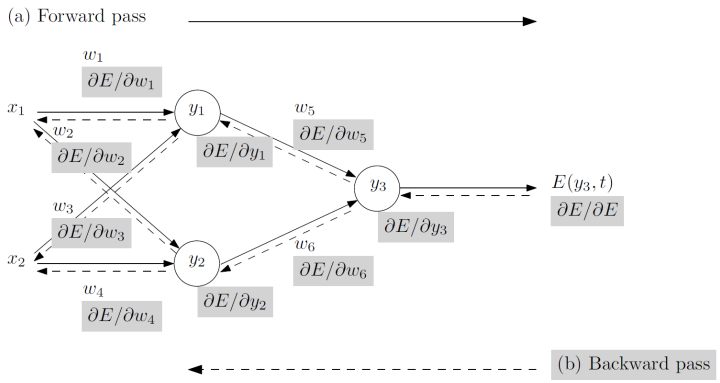
The network parameters are estimated by minimizing a loss function ($\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$) through some gradient-based optimization process. This process aims (in the best case) to find a global minimum of the loss function, thus finding a set of parameters θ^* that minimizes the loss, as indicated in equation 39.

$$\theta^* = argmin_\theta \mathcal{L}(\theta) \tag{39}$$

Backpropagation is the algorithm that models the learning in neural networks as a gradient descent in the parameters space looking for a minimum of the loss function. These gradients used to perform the descendent step are calculated using backward propagation of the value of the loss function, making use of the chain rule of differentiation, to compute partial derivatives of the loss function with respect to each weight, as it is illustrated in figure 3. This is a specific case of the so called reverse mode automatic differentiation, were in this general form it is possible to calculate the sensitivity of a considered output y_i with respect to changes in x_i, that is, the derivative of an output y_i with respect to some input x_i in the most general form of a computational graph (BAYDIN et al., 2018).

In the figure 3, the forward pass (a) the inputs x_i are fed forward, generating outputs y_i, which are ultimately used in a loss function E to calculate the error to the real value t. The backward pass (b) is where the gradient of the loss function E with respect to each of the weights (w_i) is calculated using the intermediate values of the

Figure 3 - Overview of backpropagation



Legend: Figure extracted from Baydin et al., 2018

computational graph and the differentiation chain rule (BAYDIN et al., 2018)

In the ith iteration, the model is updated according to the equation 40, where η is the learning rate, which can be seen as the step size of the update in each iteration of the training

$$\theta^{i+1} = \theta^i - \eta \bigtriangledown \mathcal{L} \tag{40}$$

1.5 **PINN**

Physical knowledge refers to understanding the fundamental laws and principles of the physical world. In many domains of science and engineering, real-world physics can be modeled using differential equations based on different domain-specific assumptions and simplifications These equations arise whenever a universal law is expressed through variables and their derivatives (TENENBAUM; POLLARD, 1985). Differential equations represent precise physical laws that can describe various scientific phenomena. These are known as governing equations.

Physics-informed Machine Learning (PIML) is a direction of ML that aims to leverage prior physical knowledge to improve performance on a set of tasks that involve a physical mechanism. A fundamental question for PIML is how this prior physical knowledge is integrated into machine learning models, previous works showed that it can be incorporated through modifications in one or more features of machine learning models. (HAO et al., 2022)

The PINN approach considers the solution to a PDE to be in the form of a deep neural network $u_\theta(x)$, this network is used to calculate the approximate solution of a

generic differential equation of the form:

$$\begin{cases} F(t,x,u_\theta(t,x);\gamma) &= 0 & t \in [0,T],\ x \in \Omega \\ I(x,u_\theta(0,x)) &= 0 & x \in \Omega \\ B(t,x,u_\theta(t,x)) &= 0 & t \in [0,T],\ x \in \partial\Omega \end{cases} \tag{41}$$

In the equation 41, θ represents the network parameters, F is a general differential operator that can consist of temporal derivatives, spatial derivatives, linear and non-linear terms, x is the position vector in the domain Ω, with boundary $\partial\Omega$, I and B denote, respectively, the initial and boundary conditions and can also consist of differential, linear or non-linear operators.

PINN's make the incorporation of the prior physical knowledge using the propriety of reverse mode automatic differentiation (BAYDIN et al., 2018) to differentiate neural networks with respect to their input, calculating the residual of the differential equation and incorporate it as a soft constraint In the loss function being minimized during the training of the network, that way the approximation adheres to the physical condition of the problem described by the same differential equation. (RAISSI; PERDIKARIS; KARNIADAKIS, 2019)

The main idea of automatic differentiation is that all numerical calculations are ultimately compositions of a finite set of elementary operations for which the derivatives are known, combining the derivatives of the constituent operations via the chain rule gives the derivative of the overall composition (BAYDIN et al., 2018). This idea is illustrated in figure 3.

The PINN loss can be defined by three nonnegative terms, the initial condition term defined in the spatial domain, the boundary condition term, defined in the boundary of the space domain and the entire time domain and the residual defined over the the entire spatial and time domain as in equation 42

$$\begin{aligned} L(\theta) = \frac{1}{|[0,T]\ x\ \Omega|} \int_{[0,T]\ x\ \Omega} (F(t,x,u_\theta(t,x);\gamma))^2 dtdx\ + \frac{1}{|\Omega|}\int_\Omega (I(x,u_\theta(0,x)))^2 dx + \\ + \frac{1}{|[0,T]\ x\ \partial\Omega|}\int_{[0,T]\ x\ \partial\Omega} (B(t,x,u_\theta(t,x)))^2 dtdx \end{aligned} \tag{42}$$

To calculate the equation 42 it is necessary to evaluate several integral terms that include calculations of high-order derivatives of u_θ using automatic differentiation. Therefore, it is necessary to perform some sort of discretization in order to approximate the values of 42, each integral is approximate through point obtained from their own region of the domain (RAISSI; PERDIKARIS; KARNIADAKIS, 2019; HAO et al., 2022; NABIAN; GLADSTONE; MEIDANI, 2021).

The equation 42 is approximated by the equation 43

$$L(\theta) = L_{ic}(\theta) + L_{bc}(\theta) + L_r(\theta) \tag{43}$$

And the value of each term is obtained as equation 44, where $\{0, x_{in}^i\}_{i=1}^{N_{ic}}$, $\{t_{bc}^i x_{bc}^i\}_{i=1}^{N_{bc}}$ and $\{t_r^i x_r^i\}_{i=1}^{N_r}$ represents respectively the set of points sampled in the initial condition, boundary condition and the points in the rest of the domain where the residual of the differential equation is calculated, each contaning N_{ic},N_{bc} and N_r points respectively. For simplicity, x represents all the dimensions of the spatial domain and t the time domain.

$$\begin{array}{rcl} L_{ic}(\theta) & = & \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} (I(x, u_\theta(0, x_{ic}^i)))^2 \\ L_{bc}(\theta) & = & \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} (t_{bc}^i, x_{bc}^i, u_\theta(t_{bc}^i, x_{bc}^i))^2 \\ L_r(\theta) & = & \frac{1}{N_r} \sum_{i=1}^{N_{ic}} (F(t_r^i, x_r^i, u_\theta(t_r^i, x_r^i)))^2 \end{array} \tag{44}$$

The loss function that is minimized during training is a composition of terms defined by the PDE of the problem as in equation 43. The optimal parameters θ_* (equation 45) reached by the minimization process represents the network found to approximate the solution to the PDE. Thus, approximating the solution to the PDE is reduced to an optimization problem where the initial and boundary conditions and the residual of the differential equation can be seen as constraints.

$$\theta^* = argmin_\theta (L_{ic}(\theta) + L_{bc}(\theta) + L_r(\theta)) \tag{45}$$

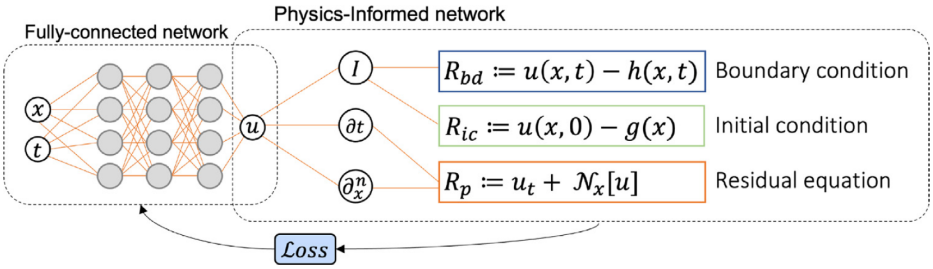
The backpropagation algorithm used to update network parameters is a special case of reverse mode automatic differentiation. During the training, the network loss is calculated as a function of its parameters, and backpropagation (BAYDIN et al., 2018) is used to calculate the gradients of the loss function with respect to the network parameters necessary to perform the updates during the training.

While the backpropagation algorithm is used to update the network parameters during PINN training, the reverse mode automatic differentiation algorithm is also applied in the residual term of the loss function to calculate the derivatives of the network's output with respect to its inputs and obtain the residual of the differential equation, this term acts as a regularizer during training of the network and ensures that physical knowledge is being incorporated into network training. The dynamic of the PINN training is represented in figure 4.

In figure 4, the loss function being minimized is the composition of the boundary condition, the initial condition, and the residual of the differential equation, in the same way as stated previously, although using a slightly different notation.

The residual term formulated by the PDE differential operator that describes the problem is calculated at each point in the domain designated as a residual point, that is, a point that is neither in the initial condition nor in the boundary condition and therefore does not have an output defined by the problem.

Figure 4 - Schematic representation of PINN architecture



Legend: Figure extracted from (LORENZO et al., 2023)

Taking the one dimensional heat equation as an example (equation 15), the term in equation 46 is the one responsible for minimizing the residual of the differential equation in the predefined points.

$$L_r(\theta) := \frac{1}{N_r}\sum_{i=i}^{N_r}\left(\frac{\partial u_\theta(t_r^i, x_r^i)}{\partial t} - \alpha\frac{\partial^2 u_\theta(t_r^i, x_r^i)}{\partial x^2}\right) \tag{46}$$

As the solution u is approximated by a neural network, the training process makes use of automatic differentiation to calculate the first and second order derivatives present in the equation 46, while the initial and boundary conditions are imposed on the network training as constraints.

In order to measure and compare the effects of the points used as training, the discretizations were carried out in a similar way to those done with previously established numerical methods. Due to the nature of this approach, it is optional to define a specific step on each dimension and form a regular spaced grid discretization to obtain the points used in training. However, for this study, regularly spaced grid training points will be used to directly compare with other numerical methods. Unlike these methods, it is possible to use random points in the domain to obtain a solution with neural networks. The work of Shin, Darbon and Karniadakis (2020) (SHIN; DARBON; KARNIADAKIS, 2020) is concerned with the convergence analysis when the points used to train the network are obtained from some probability distribution function.

2 METHODOLOGY

The first step is to evaluate the approximation to the solution of the PDE from the perspective of the estimation error. This was done using different discretizations with regular spacing to train the same neural network (same architecture and initial parameters) and obtain approximations in predefined points. The objective is to verify the influence of a finer discretization in each domain independently on the generalization capacity of the network.

The next step is to evaluate the approximation to the solution of the PDE from the perspective of the approximation error (as illustrated in figure 1) using different architectural parameters of the network and verify their influence. Each modification was made independently, and the error was obtained through the average of five models to mitigate the randomness of parameter initialization.

Finally, through the tests carried out, an optimal number of points is found along with optimal architectural parameters. With these in hand, the objective is to compare the accuracy of the optimum PINN approximation with some finite difference methods in each problem stated previously.

All applications of PINN were made in a TensorFlow environment (ABADI et al., 2016). The purpose of the finite difference methods is to approximate the solutions of differential equations. These methods proceed by replacing the derivatives in the differential equation with final difference approximations. This gives a large but finite set of algebraic equations to solve instead of the differential equation (LEVEQUE, 2007).

Two finite difference methods were chosen to compare the results obtained with PINNs in the one-dimensional heat example: one explicit with the forward time-centered space method (FTCS) and one implicit using the Crank-Nicolson method. Two FDM were also chosen to compare the results obtained with PINNs in the two-dimensional heat example: the forward time-centered space method (FTCS) and the alternating-direction implicit (ADI) method. Finally, in the advection diffusion problem, two explicit methods were used to compare the approximations to the solutions of the PDE: the forward time-centered space method (FTCS) and the Upwind method. ((HOFFMANN; CHIANG, 1998), (LEVEQUE, 2007))

One of the main differences in the compared methods is that PINN can generate an approximation of the solution at any point in the domain, regardless of whether this point was used to train the network. In other words, it is possible to use a certain discretization to train the network and then obtain an approximation of the solution at completely different points. This is a freedom that is simply not possible in FDM, in this method it is only possible to obtain an approximation at the pre-defined points of the discretization used.

During comparisons of the approximations to the solutions, the discretizations used in implementing the finite difference methods were also used in the PINN method, although it does not need to comply with any type of restriction in the discretizations used. In the FDM it is only possible to approximate solutions at these points, therefore for the comparison, the discretization used to train the network by the PINN approach were also used to generate an approximation to the solution and compare the approximations.

Finally, a study was carried out on the bias and variance decomposition of the expected PINN loss along with applications to the one-dimensional and two-dimensional heat problem to observe the behavior of these measurements as the width of the networks used increases.

The error metric used to compare the approximations with the analytical solution of the differential equation is the L_2 relative norm given by equation 47. $y(x)$ is the analytical solution and $u_\theta(x)$ is the approximation parametrized by θ, all the inputs are resumed in the x variable for simplicity. The L_2 is calculated in a chosen discretization of the domain containing n points, and is a way of calculating the relative distance of the approximations to the actual solution in the n chosen points.

$$L_2(y(x), u_\theta(x)) = \frac{\sqrt{\sum_{i=1}^{n}(y(x_i) - u_\theta(x_i))^2}}{\sqrt{\sum_{i=1}^{n}(y(x_i))^2}} \tag{47}$$

2.1 **Estimation error**

The estimation error is intrinsic to any approximation of a function defined in the continuous domain through a finite amount of data. Naturally, this error must be considered when estimating solutions for PDE through neural networks. The question is how many points are needed to approximate the problem adequately. It is also of interest to know how a finer discretization in each domain influences the approximation obtained.

It is common to find PINNs being trained with domain points obtained from some probability distribution function (HAO et al., 2022). In this analysis of estimation errors, the points used in training were obtained from a regular discretization in the domain, in the same way as is done in some finite difference methods. This approach aims to compare PINNs with FDM under similar conditions.

As the neural network does not need to obey any constraints on the step ratio used in the space and time domain, convergence problems are not expected using a combination that would not work in some finite difference methods.

Aiming for good consistency during the measurement of the error obtained with the trained networks, each resulting network was requested to generate an approximation in the same points of the domain, regardless of whether these points were used at some

moment of the training. This approach measures the influence of the points used during training on the network's generalization capability.

2.2 Approximation error

The approximation error aims to measure the expressiveness of the neural network used to solve the problem. Although neural networks can be used as universal function approximators (HORNIK; STINCHCOMBE; WHITE, 1989), the process of finding the architecture necessary to approximate a solution to a PDE is not a simple task. The objective is to find an architecture that fits the problem in a way that it has the expressiveness to incorporate the complexity of the problem without having excessive parameters that could compromise training.

In this work only architectures based on a multilayer perceptron were considered, therefore the parameters of these architectures that will be analyzed are the number of neurons in the hidden layer, the number of hidden layers and the activation function used in each neuron.

Naturally, the activation function has a major impact on the training of PINNs, because, as the high-order derivatives must be calculated using backpropagation, it is necessary to have a smooth activation function.

Different MLP architectures were used to produce approximations to the PDEs solutions. The objective of the experiments is to independently evaluate the impact of more neurons in the hidden layers, more hidden layers, and different activation functions on the obtained approximation.

The different activation functions tested in the experiments are displayed in table 1 along with their respective formulations

All networks were trained and evaluated using the same points. All errors presented in the plots were obtained from the average of five models trained with the same architecture, but compiled with different initial values for the trainable parameters. As another atempt to mitigate the randomness of the initial value while maintaining some coherence with how the procedure is currently used in practice, the initial values of the trainable parameters were obtained from a normal distribution function with a mean of 0 and a standard deviation equal to 0.25.

2.3 Bias Variance Decompostion

An important analysis for understanding the generalization capability of a machine learning algorithm is the Bias and Variance decomposition (HASTIE et al., 2009;

Table 1 - Activation functions used in testing

Function	Formulation
Hyperbolic Tangent	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU)	$\begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
Exponential linear unit (ELU)	$\begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
Gaussian Error Linear Unit (GELU)	$\frac{1}{2}x\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$
Scaled exponential linear unit (SELU)	$\lambda\begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
Sigmoid	$\frac{1}{1 + e^{-x}}$
Swish	$\frac{x}{1 + e^{-x}}$

Legend: The hyperparameter α in the linear exponential unit (ELU) has been set to 1. This hyperparameter controls the value of ELU saturates for negative net entries; more information is available in (CLEVERT; UNTERTHINER; HOCHREITER, 2015). The Gaussian Error Linear Unit (GELU) function $erf()$ represents the Gauss error function; more information is available in (HENDRYCKS; GIMPEL, 2016). The scaled exponential linear unit (SELU) λ is fixed at 1.05070098 and α at 1.67326324, more information available at (KLAMBAUER et al., 2017). More information about the Swish function is available at (RAMACHANDRAN; ZOPH; LE, 2017)

BISHOP, 2006). In most machine learning algorithms it is possible to observe a tradeoff relationship between the variance and bias, and it has been suggested that this relationship was also present in neural networks (GEMAN; BIENENSTOCK; DOURSAT, 1992) in the sense that, as the complexity (number of parameters) of the network increases the bias decreases while the variance increases and the statement "The price to pay for achieving low bias is high variance" (GEMAN; BIENENSTOCK; DOURSAT, 1992) was also true for neural networks. Although this was a common assumption for many, recent works suggested that this relationship might not be present when analyzing errors obtained in work based on neural networks. (NEAL et al., 2018; NEAL, 2019)

The tradeoff relationship is connected with the notion of underfitting and overfitting the training data, and it all revolves around the generalization capacity of the network. In the under-parameterized algorithms, there is a tendency to obtain a model that is not capable of capturing the true complexity of the problem and, therefore, yielding a model that underfits the data used in training. On the other hand, the over-parameterized algorithms have tendencies to obtain a model that has the expressivity needed to capture the problem complexity, but it may yield a model that will overfit the training data, capturing also the random fluctuations present in the data used in training that does not correspond to the problem complexity.

The underfitting setting of an algorithm is the one that tends to have a high bias and low variance, while the overfitting setting is the one that tends to have low bias and high variance. Ideally, the right complexity of the algorithm for the problem is the one with the intermediate value; that way, it is not underfitting nor overfitting the training data. This ideal setting has the best generalization capability, where it can extrapolate beyond the data used in its training and provide good approximation in data that was not present during the model's training. (HASTIE et al., 2009; BISHOP, 2006)

Recent work exploring the behavior of these quantities in neural networks have refuted an early claim that the tradeoff relationship was also present in these models. More specifically, it was suggested that the variance of neural networks have a tendency of increasing as its width (number of neurons in hidden layers) increases, losing its generalization capability. Through many experiments in recent works, it was suggested that, ultimately, the variance of the networks tends to decrease as their width increases, and their generalization capabilities tend to increase as well. (NEAL et al., 2018; YANG et al., 2020; NEAL, 2019)

2.3.1 Decomposition of square loss

The early decomposition in the works of (GEMAN; BIENENSTOCK; DOURSAT, 1992) is made in the context of a supervised learning regression problem, where the

parametric learner has the task of minimizing a mean squared loss function. The task is to learn a function $u : \mathbb{R}^d \to \mathbb{R}^c$ based on training samples $\tau = \{(x_i, y_i)\}_{i=1}^n$ drawn from some joint distribution P such that the mean squared loss $\mathbb{E}_{x,y}[(y - u_\theta(x, \tau))^2]$ is minimal, where $(x, y) \sim P$ (YANG et al., 2020). The learned function is denoted as $u_\theta(x, \tau)$ to make the dependence on the training data τ clear. Also it is in evidence in the notation the parameterization θ of the network, this value will not be present for now for simplicity. In this notation the mean squared error loss is expressed as the expected value over the test dataset, as equation 48.

$$\frac{1}{n}\sum_{i=1}^{n}[(y_i - u_i(x_i, \tau))^2] = \mathbb{E}_{x,y}[(y - u(x, \tau))^2] \tag{48}$$

And its decomposition into bias and variance begins by taking the expected prediction of the test dataset over all the models trained with all the possible training datasets τ.

$$\begin{aligned}
\mathbb{E}_{x,y}\mathbb{E}_\tau[(y - u(x,\tau))^2] &= \mathbb{E}_{x,y}\mathbb{E}_\tau[((y - \mathbb{E}_\tau[u(x,\tau)]) + (\mathbb{E}_\tau[u(x,\tau)] - u(x,\tau))^2] \\
&= \mathbb{E}_{x,y}\mathbb{E}_\tau[(y - \mathbb{E}_\tau[u(x,\tau)])^2] + \mathbb{E}_{x,y}\mathbb{E}_\tau[\mathbb{E}_\tau[u(x,\tau)] - u(x,\tau))^2] + \\
&\quad +2.\mathbb{E}_{x,y}\mathbb{E}_\tau[(y - \mathbb{E}_\tau[u(x,\tau)])].\mathbb{E}_{x,y}\mathbb{E}_\tau[\mathbb{E}_\tau[u(x,\tau)] - u(x,\tau))] \\
&= \mathbb{E}_{x,y}[(y - \mathbb{E}_\tau[u(x,\tau)])^2] + \mathbb{E}_x\mathbb{E}_\tau[\mathbb{E}_\tau[u(x,\tau)] - u(x,\tau))^2]
\end{aligned}$$

The term $\mathbb{E}_{x,y}\mathbb{E}_\tau[(\mathbb{E}_\tau[u(x,\tau)] - u(x,\tau))]$ is equal to zero due to the linearity of the expected value, resulting $\mathbb{E}_{x,y}[(\mathbb{E}_\tau[u(x,\tau)] - \mathbb{E}_\tau[u(x,\tau)])]$. The remaining terms, getting rid of the expected value that have no longer use in the expression, results in the measures of interest.

The term $\mathbb{E}_{x,y}[(y - \mathbb{E}_\tau[u(x,\tau)])^2]$ is the bias while the therm $\mathbb{E}_x\mathbb{E}_\tau[\mathbb{E}_\tau[u(x,\tau)] - u(x,\tau))^2]$ is the variance. The bias is a measure of how close the central tendency of a learner is to the true function, the variance is a measure of fluctuations of a learner around its central tendency, where the fluctuations result from different samplings in training (GEMAN; BIENENSTOCK; DOURSAT, 1992; NEAL, 2019; YANG et al., 2020; HASTIE et al., 2009)

When the decomposition was first introduced in the context of machine learning, it was suggested that the relationship of these two measurements was that of a tradeoff, where for one to increase the other would have to decrease. Although this seems to be true in many ML algorithms, in the context of neural networks the experiments first presented by (GEMAN; BIENENSTOCK; DOURSAT, 1992) were a topic of debate in later studies, suggesting that this trade-off relation was in fact not present in the neural networks. (NEAL, 2019; YANG et al., 2020; NEAL et al., 2018).

2.3.2 Methodology of the decomposition with PINN's

It was stated previously that the training of PINN is not as straightforward as other machine learning problems. In a common supervised learning regression problem, the training data consists of the pairs of the input x and its respective output y, where the outcome of the problem is, by definition, known. During the training of PINNs, however, to produce the approximation to the solution of the PDE, the training does not minimize a simple loss function; instead, it minimizes a loss function that depends on the derivatives of the same neural network that is being trained, minimizing the residuals of the PDE which the solution is being approximated.

Thus, on account of the difference in the loss function that is being minimized, the decomposition of the loss into bias and variance has to be analyzed further. The decomposition cannot be done considering the values of the solution that are not available during training; the decomposition must be done with the values that are being minimized; therefore, the fixed values, coming from specific conditions of the problem and the residuals calculated during training.

Each region of the domain of the PDE has to obey some condition established by the problem. During the problem studied in section 1.1, for example, the initial and boundary conditions were given as fixed values, while the points on the inside of the domain, the collocation points, were the ones where the residual of the PDE was calculated. As a consequence, the loss function minimized treated the initial and boundary condition points the same way as a supervised learning problem (minimizing the mean squared loss between the ground truth value and the approximated value) and minimized the residual of the PDE in the collocation points (calculating the derivatives of the network concerning its input with automatic differentiation). The value given by this custom loss function is the one that is being decomposed into bias and variance, not the value yield as the approximation of the solution.

Using this specific problem as an exemple, the term of the loss function that is responsible for minimizing the distance of the predictions of the initial and boundary conditions are the same as a supervised learning problem and takes the form of equation 49. Where $u(t, x)$ is the approximations of a given input, y is the true value and $\{(t_1^u, x_2^u), (t_2^u, x_3^u), \dots (t_{N_b}^u, x_{N_b}^u)\}$ are the points in the initial and boundary condition.

$$\frac{1}{N_b} \sum_{i=1}^{N_b} (u(t_i^u, x_i^u) - y_i)^2 \tag{49}$$

The difference in the way the loss is calculated in the colocation points is that it does not minimize the distance of the approximation to the true values; instead, it minimizes the value of the residual of the PDE in these points. However, it is similar in that it minimizes the distance of this value to zero. Therefore it can take the form

of equation 50, where the derivatives are calculated using automatic differentiation and $\{(t_1^f, x_2^f), (t_2^f, x_3^f), \dots (t_{N_r}^f, x_{N_r}^f)\}$ are the collocation points.

$$\frac{1}{N_r}\sum_{i=i}^{N_r}\left[\left(\frac{\partial u_(t_i^f, x_i^f)}{\partial t} - \alpha\frac{\partial^2 u(_i^f, x_i^f)}{\partial x^2}\right) - 0\right]^2 \tag{50}$$

This term (equation 50), put in this redundant way, makes clear that, in each region a mean squared loss is being minimized and, therefore, its decomposition into bias and variance is valid just as it was present previously, even though the true values of the problem are not present in the minimization process of the collocation points. Instead of the true value, it is necessary to compare them with the value zero, to which its distance is being minimized by minimizing the residual.

2.3.3 Experiments and Results

To approximate the expected value over the training sets, it is required, by definition, different training sets to be extracted from the same distribution. Due to the scarcity of data in usual machine learning applications, these calculations usually rely on bootstrap training sets, which simulate obtaining new data. During the application of PINN, it is possible to have total control over the domain discretization and, therefore, the data used in training. For this reason, one of the main differences is the access to the original distribution from which the data was extracted. Since obtaining new data sets is not a problem, the expected values can be calculated by averaging the loss of trained networks (trained with different data sets extracted from the same distribution) on a distinct data set called the test data set. This test data set evaluates all trained networks on the same points, obtaining the expected loss value on a fixed test set.

The denotation $u(x, \tau)$ is the trained network estimated by a given training set τ; for simplicity, all the inputs are summarized as the vector x. For all experiments, to estimate the value of the bias and variance it was used 30 training sets $\tau_1, \tau_2, \dots, \tau_{30}$ to generate the corresponding trained networks $u(x, \tau_1), u(x, \tau_2), \dots, u(x, \tau_{30})$. The expected value $\mathbb{E}_\tau[u(x, \tau)]$ is estimated taking the average of the loss of all the trained networks $\overline{u}(x) = 1/30 \sum_{i=1}^{30} u(x, \tau_i)$. Bias and variance are estimated via the formulas (GEMAN; BIENENSTOCK; DOURSAT, 1992).

$$\text{Bias} \simeq \mathbb{E}_{x,y}[(y - \overline{u}(x))^2] \tag{51}$$

$$\text{Variance} \simeq \mathbb{E}_x[1/30 \sum_{i=1}^{30}(\overline{u}(x) - u(x, \tau_i))^2] \tag{52}$$

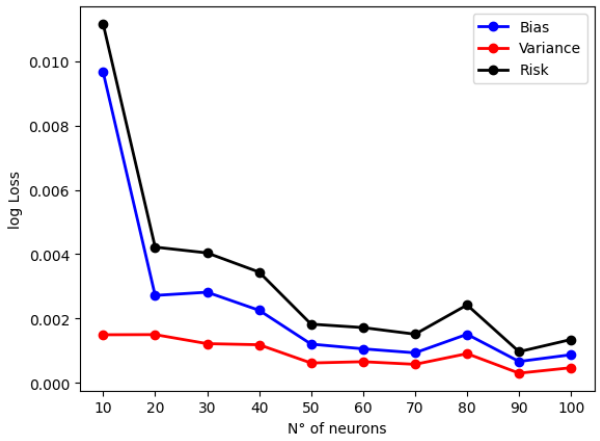
And the expected value $\mathbb{E}_{x,y}$ and $\mathbb{E}_x$ are the average over the loss of each point on the test dataset

To mitigate the effect of the random initialization of the parameters, each result was obtained from an average of five different networks trained with the same sets. The initial values of the trainable parameters were obtained from a normal distribution function with a mean of 0 and a standard deviation equal to 0.25.

The blue and red lines represent bias and variance, respectively, while the black line is the sum of both, often called the risk curve. The black line represents the expected loss of the network with a dataset that was not used during training (test dataset).

In the one-dimensional heat problem, all the networks were trained for 5000 epochs with Adam optimizer and a fixed learning rate of 10^{-4}. All architectures contained one hidden layer, while the number of neurons varied from 10 to 100 to vary the network's complexity (number of parameters). The activation function used was GELU (HENDRYCKS; GIMPEL, 2016). All the training datasets and the test data set contained 50 points from the initial condition, 300 points from the boundary condition, and 1000 residual points. The results are displayed in the Figure 5.

Figure 5 - Bias and variance decomposition - 1D heat

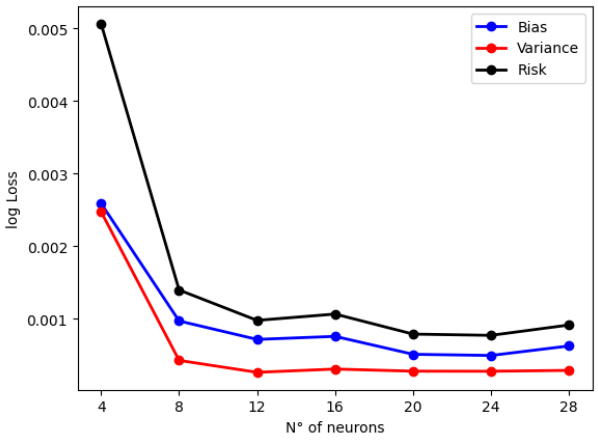


Legend: Loss value in logarithmic scale (y-axis). Number of neurons used (x-axis)

In the two-dimensional heat problem, all the networks were trained for 5000 epochs with Adam optimizer and a fixed learning rate of 10^{-2} until 1000 epochs, 10^{-3} until 3000, and 10^{-4} until 5000 epochs. All architectures contained three hidden layers, while the number of neurons varied from 4 to 28 to vary the network's complexity (number of parameters). All the training datasets and the test data set contained 500 points from

the initial condition, 1500 points from the boundary condition, and 5000 residual points. The results are displayed in Figure 6.

Figure 6 - Bias and variance decomposition - 2D heat



Legend: Loss value in logarithmic scale (y-axis). Number of neurons used (x-axis)

D

Observing the results, the bias decreases as the network width increases, making it clear that wider networks are, in fact, better able to capture the complexities of the problem. On the other hand, the variance follows the results presented in recent studies, which shows that it does not tend to increase as the network width increases. With these observations, it is possible to suggest that the training of these networks is not compromised when using wider networks and, therefore, the approximations should not be affected either.

It is also possible to suggest that, after a threshold value of the neurons, there is no significant increase in the training of the networks, which means that using values greater than the threshold does not result in a better training process and, therefore, does not result in a better approximation.

3 CASE STUDY

3.1 One Dimensional Heat Problem

3.1.1 Discretization influence - Heat 1D

In order to measure the influence of the domain discretization on the obtained approximation, the same neural network (same architecture and same initial weights) was trained using different types of discretization. The network consisted of eight densely connected hidden layers with twenty neurons each; the activation function used was the Gaussian Error Linear Unit (GELU) (HENDRYCKS; GIMPEL, 2016), trained with the Adam optimizer for 2000 epochs. For this problem, the input layer had two dimensions, one for space and one for time, and the output layer had one dimension. This network had a total of 3021 trainable parameters.

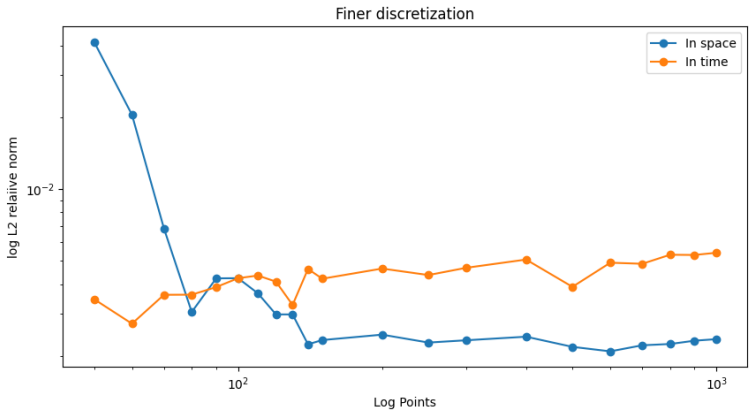
Two tests were performed to independently measure the influence of a finer discretization in the domain of space and time. In the first, the number of points used in the time domain was fixed while the number of points in the spatial domain was varied, and in the second, the opposite was done. As the neural network does not need to obey any constraints on the step ratio used in the space and time domain, convergence problems are not expected using a combination that would not work in some finite difference methods.

It is possible to see the results of both experiments in the graph of Fig. 7. Each point on the graph is a particular discretization, on the x-axis is the total number of points in the discretization and on the y-axis is the L_2 relative norm of the approximation with the analytical solution (both axes in logarithmic scale).

In the time domain refinement experiment (orange line in Figure 7), the spatial domain was fixed at 10 points while the spatial domain varied from 5 points to 15, adding 1 point, and then 15 points to 100, adding 5 points each time. In the spatial domain refinement experiment (blue line in figure 7), the spatial domain varies from 5 points to 15, adding 1 point, and then from 15 points to 100, adding 5 points at a time, while the time domain was set at 10 points.

For each discretization used to train the network, the relative L_2 norm was calculated using the approximation obtained with fifteen points in the space domain and fifty in the time domain for a total of seven hundred and fifty points. The network approximations were compared with the analytical solution on the same points.

Figure 7 - Finer discretization - 1D heat



Legend: The orange line refining the space domain and the blue line the time domain. Each totaling 21 different discretizations.

3.1.2 Network Architecture Influence - Heat 1D

To discover the best architecture for the network used to approximate the solution of the one-dimension heat equation, described in equation 15, a multilayer perceptron with the Gaussian Error Linear Unit (GELU) (HENDRYCKS; GIMPEL, 2016) as its activation functions were initially used. For the test aimed at finding the optimal number of neurons in the hidden layers, the number of hidden layers was fixed at three and the number of neurons in these layers was varied.

It is possible to see the results in Figure 8.

The graph in figure 8 shows the number of neurons used in the network on the x-axis and the L_2 relative norm on a logarithmic scale on the y-axis. The experiment was carried out ranging from 1 to 30 neurons in the hidden layers, adding 1 at a time and then from 5 to 100 neurons, adding five at a time.

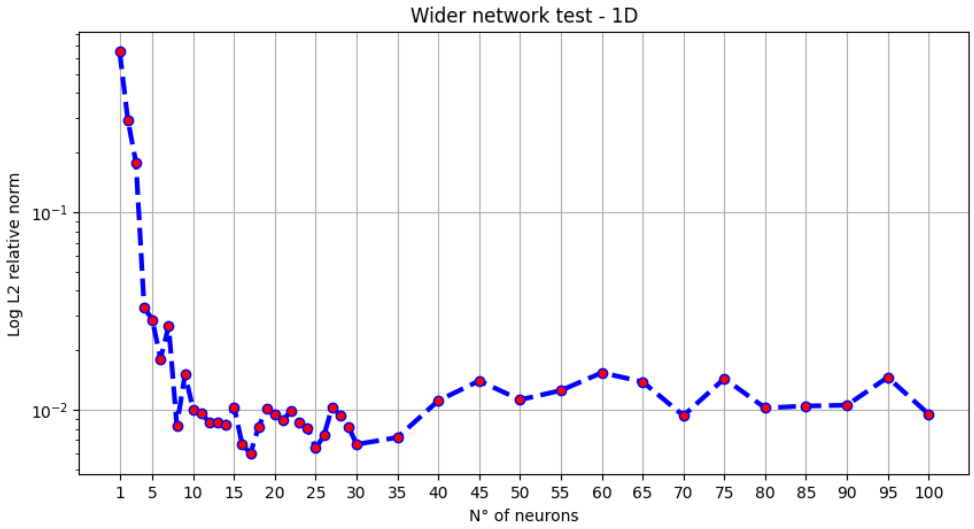
The second test, aiming to discover the impact of the number of hidden layers, was performed with a fixed number of twenty neurons per layer and varying the number of layers.

The results are on display in the Figure 9.

The graph in Figure 9 shows the number of hidden layers used in the network on the x-axis and the L_2 relative norm on a logarithmic scale on the y-axis. The experiment was carried out by varying the number of hidden layers from 1 to 16 hidden layers, adding 1 at a time.

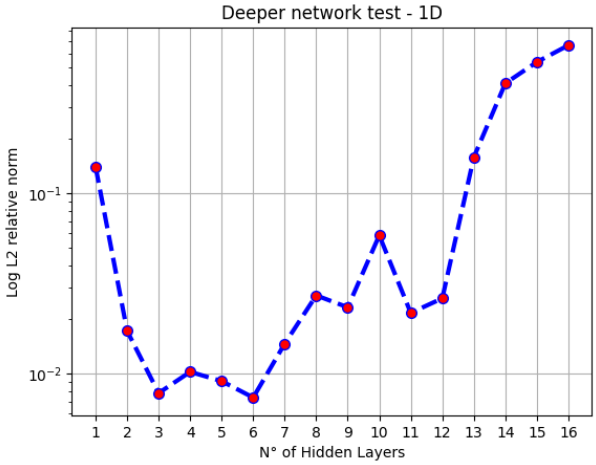
A final test was made to evaluate which was the best activation function. The number of hidden layers was fixed at three and the number of neurons per layer was fixed

Figure 8 - Wider network test - 1D heat



Legend: Experiment where the number of neurons in the hidden layer was varied for the one-dimensional heat problem

Figure 9 - Deeper network test - 1D heat

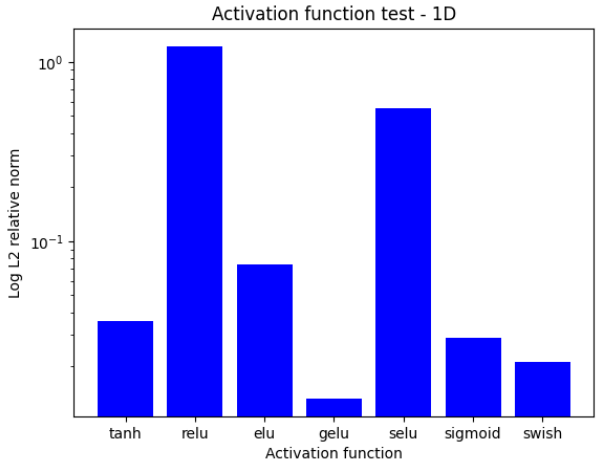


Legend: The graph shows the experiment where the network architecture was varied from 1 to 16 hidden layers, adding 1 each time.

at twenty.

The functions used, as well as the average error obtained, are shown in Figure 10; the formulas and details of each one can be seen in the table 1. It is common to use hyperbolic tangent when approximating functions by neural networks, but this test showed that using the newly proposed Gaussian error linear unit (GELU) ((HENDRYCKS; GIMPEL, 2016)) as the activation function, it is possible to obtain a better approximation.

Figure 10 - Activation function test - 1D heat



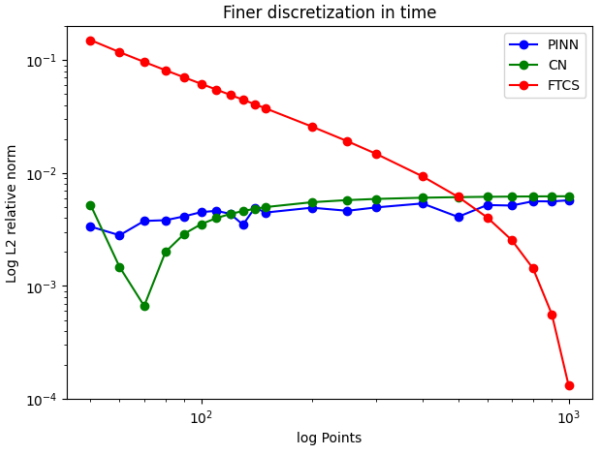
Legend: Experiment done with the same network architecture but with different activation functions

3.1.3 Numerical Results - Heat 1D

Two finite difference methods were chosen to compare the results obtained with PINNs in the one-dimensional example: one explicit with the forward time-centered space method (FTCS) and one implicit using the Crank-Nicolson (CN) method (HOFFMANN; CHIANG, 1998; LEVEQUE, 2007). The FTCS method has to obey a condition related to the step size in each dimension to produce an approximation, the Crank-Nicolson method, on the other hand, is made in such a way as to be able to produce an approximation regardless of the chosen steps.(LEVEQUE, 2007).

The discretizations used in implementing the finite difference methods were also used to train the network and to evaluate it. The plot of Fig. 11 shows the L2 relative norm obtained with the Crank-Nicolson (CN) method, the FTCS method, and also the error obtained with the PINN when more points were added in the time domain.

Figure 11 - Comparison with FDM (time) - 1D heat



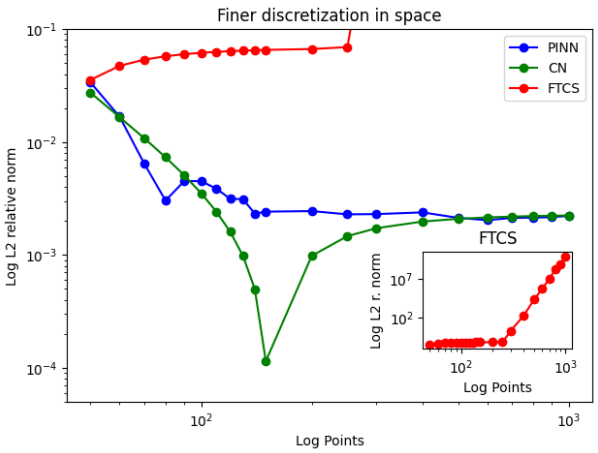
Legend: Plot relating the number of points to the L2 relative norm, both axes in logarithmic scale. Each point is a distinct discretization for a total of 21 different discretizations

In this experiment, the space domain was fixed at 10 points, while the time domain ranged from 5 points to 15, adding 1 point and then from 15 points to 100, adding 5 points each time. The CN and PINN methods produced very close approximations. In contrast, the FTCS method produced better approximations as the number of points in the time domain increased.

Regarding the experiment comparing refinement in the spatial domain, the graph in Figure 12 shows the L_2 relative norm when a finer discretization in the space domain is done while the time domain is fixed.

The experiment shown in figure 12 was carried out with the time domain fixed at 10 points while the space domain varied from 5 points to 15, adding 1 point, and then from 15 points to 100, adding 5 points each time. The first observation is that, for some of the discretizations used, the FTCS method did not converge. The convergence of this method depends on a ratio of the steps taken in each dimension, while the CN and the PINN can approximate a solution regardless of the chosen steps. In this setting, PINN shows that it can produce stable approximations, while the CN method has an optimal discretization capable of producing the best approximation.

Figure 12 - Comparison with FDM (space) - 1D heat



Legend: Plot relating the number of points to the L_2 relative norm, both axes in logarithmic scale. Each point is a distinct discretization for a total of 21 different discretizations

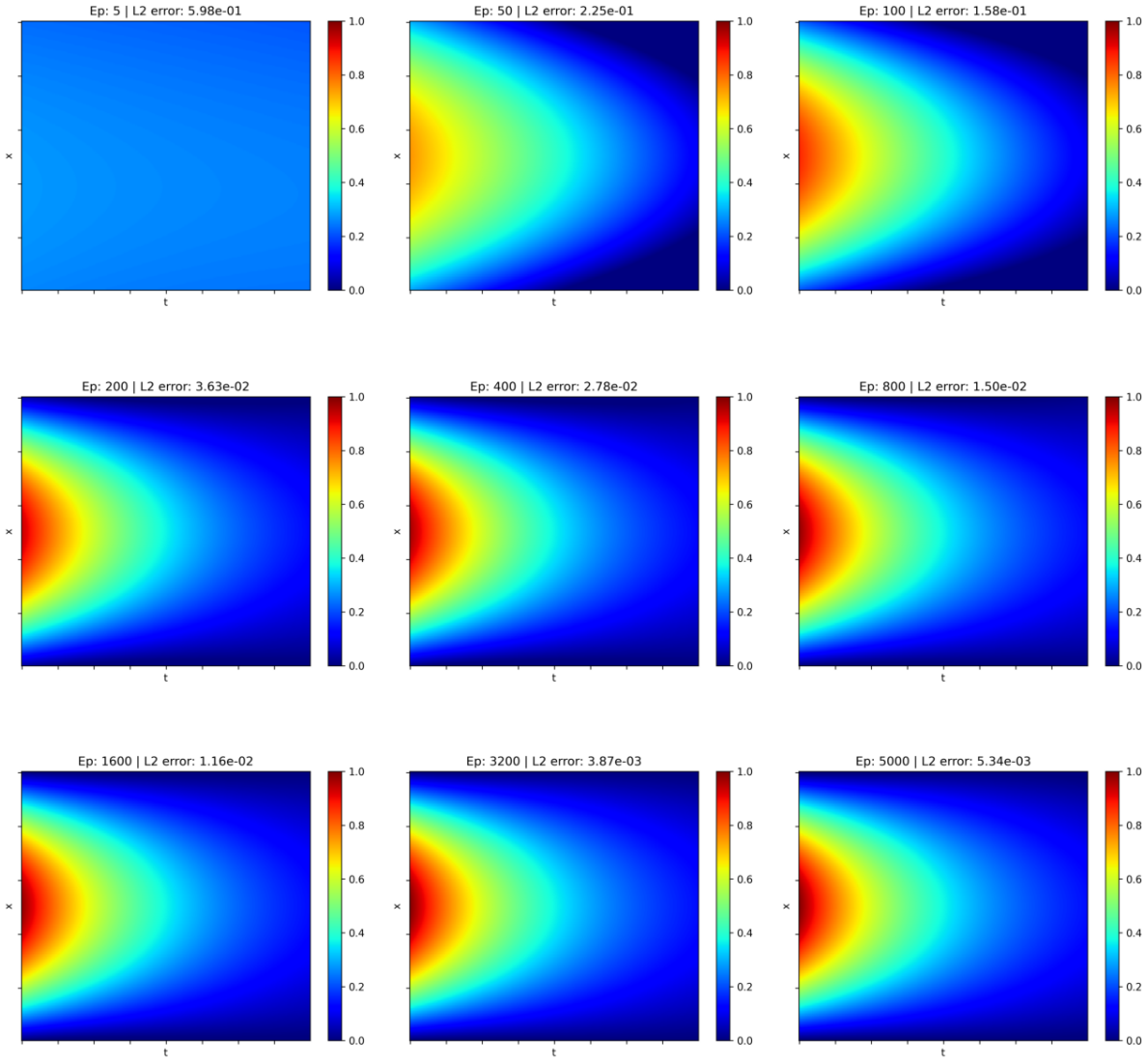
3.1.4 Visualization - Heat 1D

As a way of visualizing the training process, fidure. 13 shows the approximations of the network after some training epochs. The number of training epochs for each solution is displayed above each plot, along with the L2 relative norm obtained with each approximation. The presented solutions were obtained from a grid with 100 points in the space domain and 400 in the time domain.

The network was composed of 3 hidden layers with 20 neurons each, totaling 921 trainable parameters, each neuron having GELU as an activation function. The network was trained for 5000 epochs with a learning rate of 10^{-2} to 1000 epochs, 10^{-3} from 1000 to 3000 epochs, and 5.10^{-4} from 3000 to 5000 epochs. The absolute error of the network after 5000 epochs of training is displayed in figure 14.

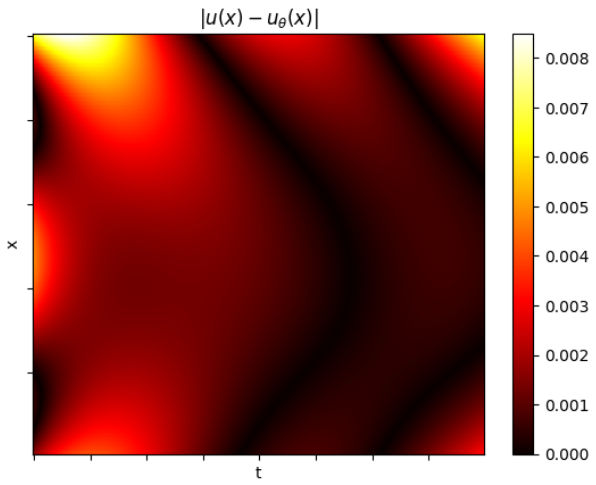
In the absolute error of the one-dimensional problem (figure 14), it is possible to observe that all points in the domain have agreement of at least two decimal places.

Figure 13 - Visualization of training - 1D heat



Legend: Solutions obtained during the training process of a physics informed neural networks

Figure 14 - Absolute error - 1D heat



Legend: Absolute error for the one dimension heat equation

3.2 **Two Dimensional Heat Problem**

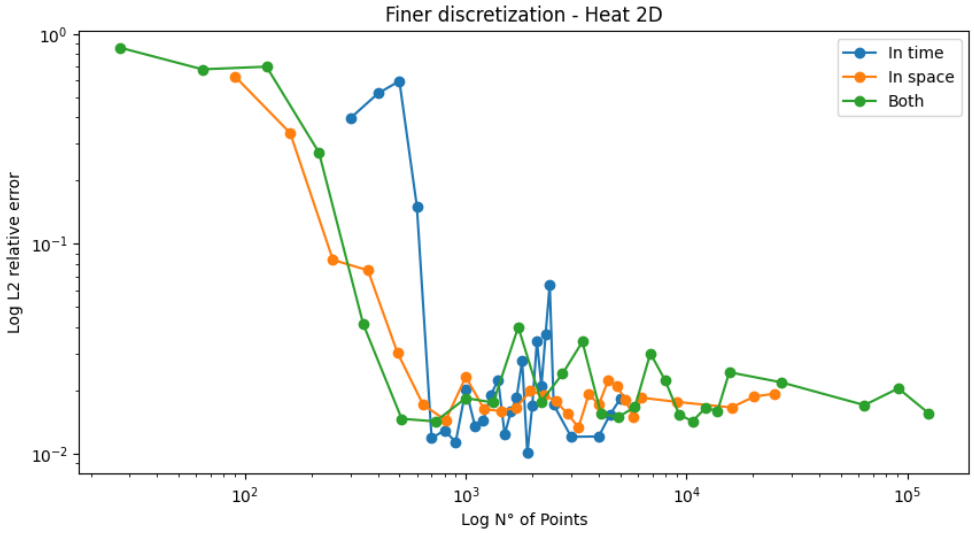
3.2.1 Discretization Influence - Heat 2D

Just as was done for the 1D evaluation, the influence of domain discretization on the obtained approximation was also measured for this case, using the same neural network (same architecture and same initial weights) trained with different types of discretization. The network consisted of ten densely connected hidden layers with thirty neurons each; the activation function used was the Gaussian Error Linear Unit (GELU) (HENDRYCKS; GIMPEL, 2016), trained with the Adam optimizer for 5000 epochs. For this problem, the input layer had three dimensions, two for space and one for time, and the output layer had one dimension. This network had a total of 2941 trainable parameters.

To independently measure the influence of a more refined discretization in the space and time domain, two tests were also carried out for the 2D case, following the same methodology used in the 1D case. At first, the number of points used in the time domain was fixed while the number of points in the spatial domain was varied, and in the second, the opposite was done. Again, for this case, as the neural network does not need to obey any restrictions on the step ratio used in the space and time domain, convergence problems are not expected using a combination that would not work in some finite difference methods. It is possible to see the results of both experiments in Figure 15. Each point on the graph is a particular discretization, on the x-axis is the total number of points in the discretization, and on the y-axis is the L_2 relative norm of the approximation

with the analytical solution (both axes in logarithmic scale)

Figure 15 - Finer discretization - 2D heat



Legend: The orange line is refining the space domain, the blue line the time domain, and the green line both. Each experiment has a total of 27 different discretizations.

In the time domain refinement experiment (blue line in figure 15), each spatial dimension was fixed at 10 points while the time domain was ranged from 3 points to 25, adding 1 point, and then from 25 points to 50, adding 5 points each time. In the spatial domain refinement experiment (orange line in figure 15), each spatial dimension ranged from 3 points to 25, adding 1 point, and then from 25 points to 50, adding 5 points at a time, while the domain time was set at 10 points. Finally, in the experiment where both domains are varied (green line in figure 15) at the same time, each spatial dimension and time varies from 3 points to 25, adding 1 point, and then from 25 points to 25 points. 50, adding 5 points at a time.

The relative L_2 norm was calculated using the solution obtained with fifteen points in each space domain and fifty in the time domain for a total of eleven thousand two hundred and fifty points. The values provided by the networks were then compared with the analytical solution on the same points.

3.2.2 Network Architecture Influence - Heat 2D

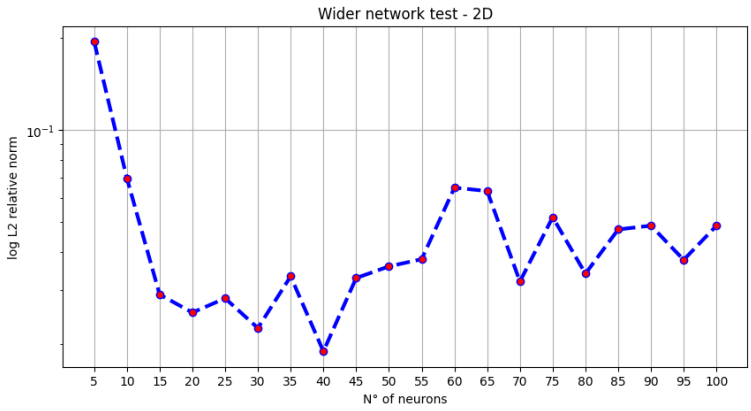
To discover the best architecture for the network used to approximate the solution, a multilayer perceptron with the Gaussian Error Linear Unit (GELU) (HENDRYCKS;

GIMPEL, 2016) as its activation functions was initially used.

For the test aimed at finding the optimal number of neurons in the hidden layers, the number of hidden layers was initially fixed at four, and the number of neurons in these layers was varied; this initial parameter was chosen since it would provide the network with enough expressibility while not making the training impractical.

It is possible to see the results in Figure 16.

Figure 16 - Wider network test - 2D heat



Legend: Experiment where the number of neurons in the hidden layer was varied for the two-dimensional heat problem

The graph in Figure 16 shows the number of neurons used on the x-axis and the L_2 relative norm in logarithmic scale on the y-axis. The number of neurons was varied from 1 to 100 neurons in the hidden layers, adding 5 each time.

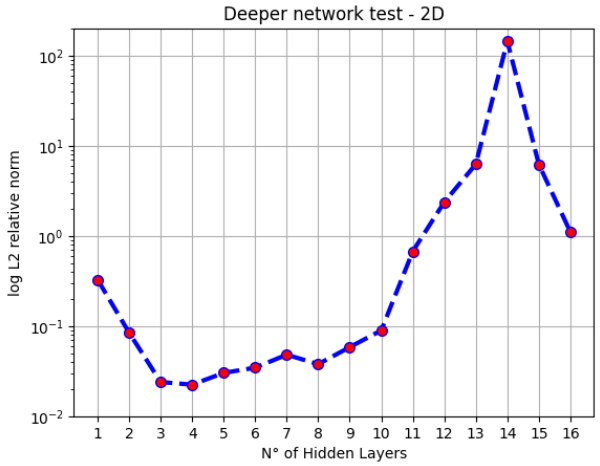
The second test, aiming to discover the impact of the number of hidden layers, was performed with a fixed number of thirty neurons in each hidden layer while varying the number of layers.

The previous test showed that the error was very similar to the minimum obtained with thirty neurons, suggesting that thirty neurons in the hidden layers provide the network with the necessary expressiveness. The results are displayed in the Figure 17.

The graph in Figure 17 shows the number of hidden layers used on the x-axis and the L_2 relative norm in logarithmic scale on the y-axis. The number of hidden layers was varied from 1 to 16, adding 1 at a time.

A final test was made to evaluate the best activation function. The number of hidden layers was fixed at three and the number of neurons per layer was fixed at thirty. These parameters were chosen since the average error obtained in the previous test suggested that the network with this parametrization contains the necessary expressiveness for the problem.

Figure 17 - Deeper network test - 2D heat



Legend: Experiment where the number of hidden layers of the network was varied for the two-dimensional heat problem

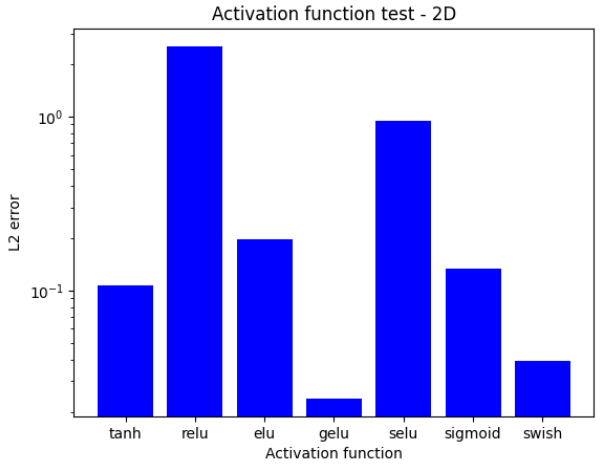
The functions used, as well as the average error obtained, are shown in Figure 18; the formulas and details of each one can be seen in the table 1. It is common to use hyperbolic tangent when approximating functions by neural networks, but this test showed that using the newly proposed Gaussian error linear unit (GELU) ((HENDRYCKS; GIMPEL, 2016)) as the activation function, it is possible to obtain a better approximation.

3.2.3 Numerical Results - Heat 2D

Two finite difference methods were chosen in the two-dimensional example to compare the results obtained with PINNs: one explicit with the forward time-centered space method (FTCS) and one implicit using the Alternating Direction Implicit (ADI) Method (HOFFMANN; CHIANG, 1998; LEVEQUE, 2007). The FTCS method has to obey a condition related to the step size in each dimension to produce a solution; the ADI method, on the other hand, is made in such a way as to be able to produce a solution regardless of the chosen steps (LEVEQUE, 2007).

As in the one-dimensional example, the discretizations used in implementing the finite difference methods were also used to train the network and to evaluate it. The plot of Fig. 19 shows the L2 relative norm obtained with the Alternating Direction Implicit (ADI) Method, the FTCS method, and also the error obtained with the PINN when more

Figure 18 - Activation function test - 1D heat



Legend: Experiment done with the same network architecture but with different activation functions

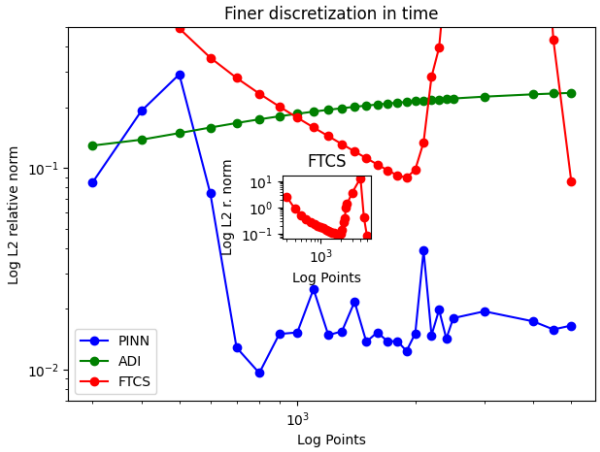
points were added in the time domain.

In the time domain refinement experiment, (figure 19) both dimensions of the spatial domain were fixed with 10 points, while the time domain varied from 3 points to 25, adding 1 point, and after 25 points to 50, adding 5 points each time. It is possible to see that, while the FTCS method was not able to produce approximations for all discretizations and the CN method did not improve at all, the PINN approximations were able to provide consistently better results in almost all tested discretizations.

The test refining the space domain is displayed in Figure 20, this plot shows the L_2 relative error when a finer discretization in the space domain is done while the time domain is fixed.

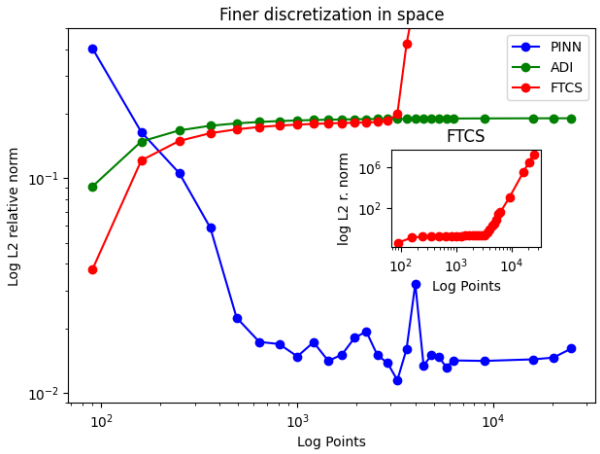
In this experiment shown in Figure 20, the time domain was fixed at 10 points while both dimensions of the space domain ranged from 3 points to 25, adding 1 point, and then from 25 points to 50, adding 5 points each time. The first observation is that, for some of the discretizations used there, the FTCS method did not converge, while the ADI and the PINN can approximate a solution regardless of the chosen steps. However, PINN approximations managed to be much closer to the solution.

Figure 19 - Comparison with FDM (time) - 2D heat



Legend: Plot relating the number of points to the L_2 relative norm, both axes in logarithmic scale. Each point is a distinct discretization, totaling 27 different discretizations.

Figure 20 - Comparison with FDM (space) - 2D heat



Legend: Plot relating the number of points to the L_2 relative norm, both axes in logarithmic scale. Each point is a distinct discretization, totaling 27 different discretizations

3.2.4 Visualization - Heat 2D

Figure 21 shows the approximations of the network after some training epochs for specific time steps (0, 0.05, 0.1, 0.15). The displayed approximations were produced after 0, 50, 100, 1000, 2000 and 5000 epochs. The presented solutions were obtained from a grid with 50 points in each dimension of the space domain and 25 points in the time domain. As demonstrated in the estimation test, this discretization is capable of expressing the complexity of the problem domain.

The network was composed of four hidden layers with forty neurons each, totaling 2941 trainable parameters, each neuron having GELU as an activation function. The network was trained for 5000 epochs with a learning rate of 10^{-2} to 1000 epochs, 10^{-3} from 1000 to 3000 epochs, and 5.10 -4 from 3000 to 5000 epochs. Figure 22 displays the absolute error at selected time steps.

In the absolute error of the two-dimensional problem (figure 22), it is possible to observe that, in the time slices of the domain that was shown, the one that presented the greatest error was the initial condition, with agreement of at least one decimal place, while the other time slices managed to obtain agreement of at least two decimal places.

3.3 **Two Dimensional Advection-Diffusion**

3.3.1 Discretization Influence - Advection-Diffusion

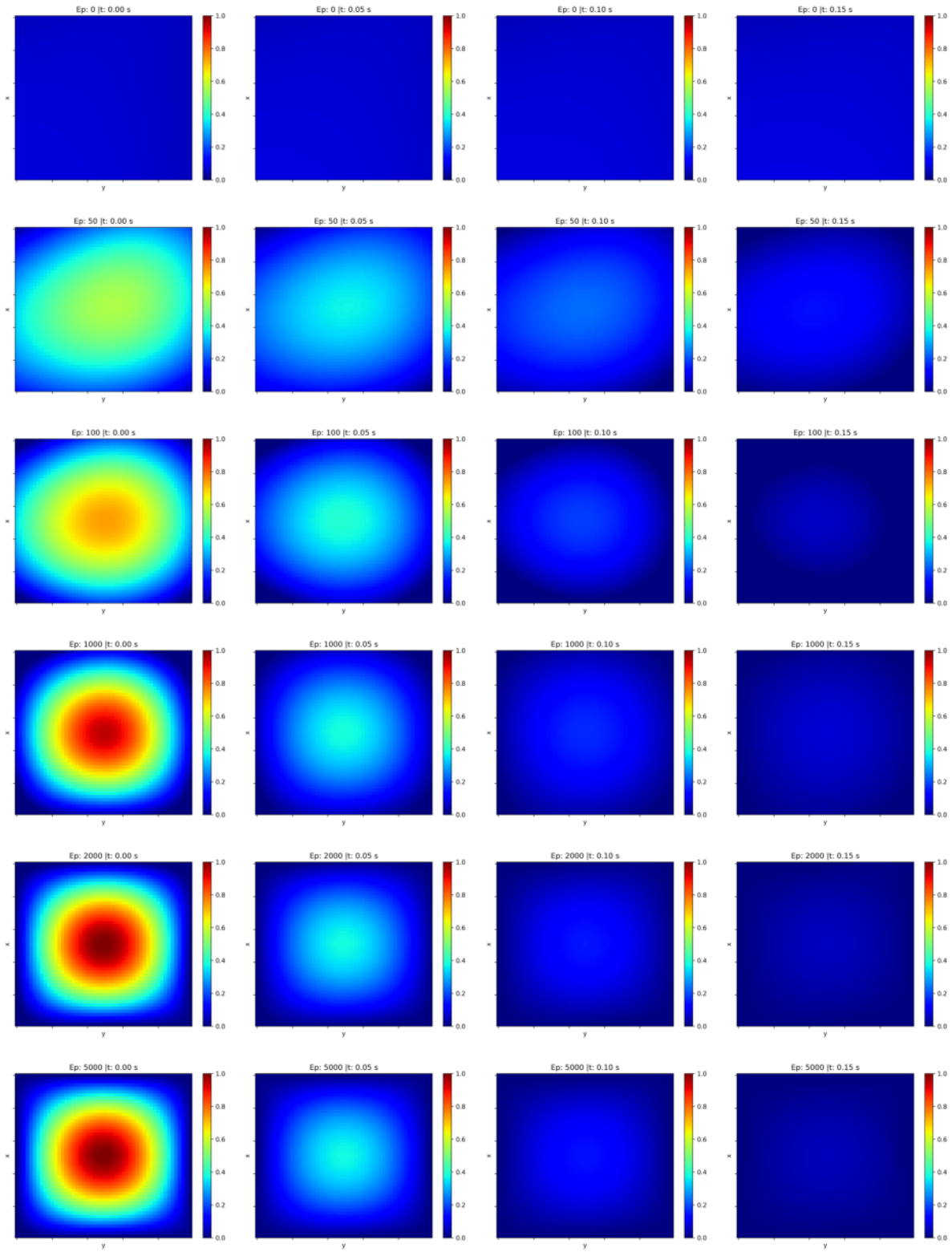
As previously performed in experiments with one-dimensional and two-dimensional heat equation, to measure the influence of domain discretization on the approximation of the solution to the advection-diffusion equation, the same neural network (same architecture and same initial weights) was trained using different types of discretization. The network had a multilayer perceptron with four hidden layers and ten neurons each. The activation function was the Gaussian Error Linear Unit (GELU) (HENDRYCKS; GIMPEL, 2016); this network contained 381 trainable parameters and was trained for 5000 epochs in each experiment.

This time, besides the two tests refining the domain in time and space independently, a test where both domains were refined together was also performed. In the independent tests, first, the number of points used in the time domain was fixed while the number of points in the spatial domain was varied, and in the second, the opposite was done. Likewise, the number of points was increased, in the same way, in both domains in the experiment where both were being refined.

The results are displayed in Figure 23.

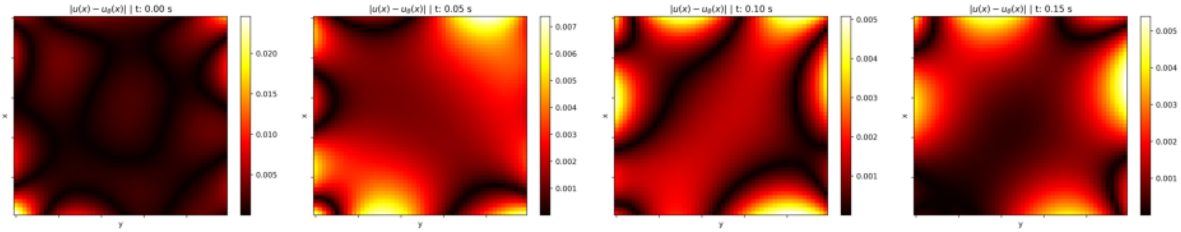
In the time domain refinement experiment (blue line in figure 23), each spatial

Figure 21 - Visualization of training - 2D heat



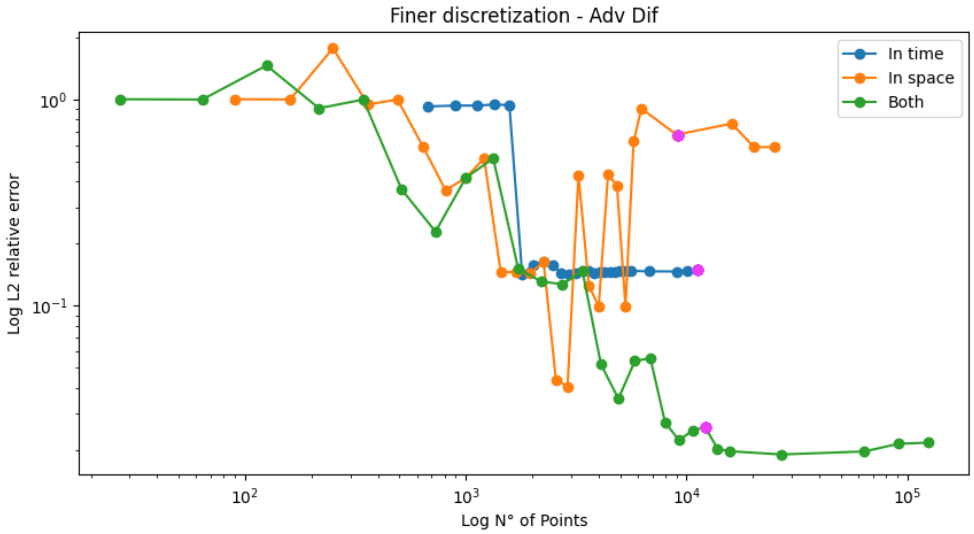
Legend: Solutions obtained during the training process of physics-informed neural networks for the two dimension heat equation at times 0, 0.05, 0.1, 0.15 after 0, 50, 100, 1000, 2000 and 5000 epochs

Figure 22 - Absolute error - 2D heat



Legend: Absolute error for the two dimensions heat equation at times 0, 0.05, 0.1, 0.15

Figure 23 - Finer discretization - Advection-Diffusion



Legend: The orange line is refining the space domain, the blue line the time domain, and the green line both. Each experiment has a total of 27 different discretizations.

dimension was fixed at 10 points while the time domain was ranged from 3 points to 25, adding 1 point, and then from 25 points to 50, adding 5 points each time. In the spatial domain refinement experiment (orange line in figure 23), each spatial dimension ranged from 3 points to 25, adding 1 point, and then from 25 points to 50, adding 5 points at a time, while the domain time was set at 10 points. Finally, in the experiment where both domains are varied (green line in figure 23) at the same time, each spatial dimension and time varied from 3 points to 25, adding 1 point, and then from 25 points to 25 points. 50, adding 5 points at a time

The relative L_2 norm was calculated using the solution obtained with fifteen points in each space domain and fifty in the time domain for a total of eleven thousand two hundred and fifty points. The values provided by the networks were compared with the analytical solution on the same points.

One additional experiment was made to study the two-dimensional advection-diffusion equation. The motivation was the discrepancy in the L_2 relative norm in the different discretization refinements as the total number of points increased. It was suggested that this discrepancy was due to the difference in how the points were distributed.

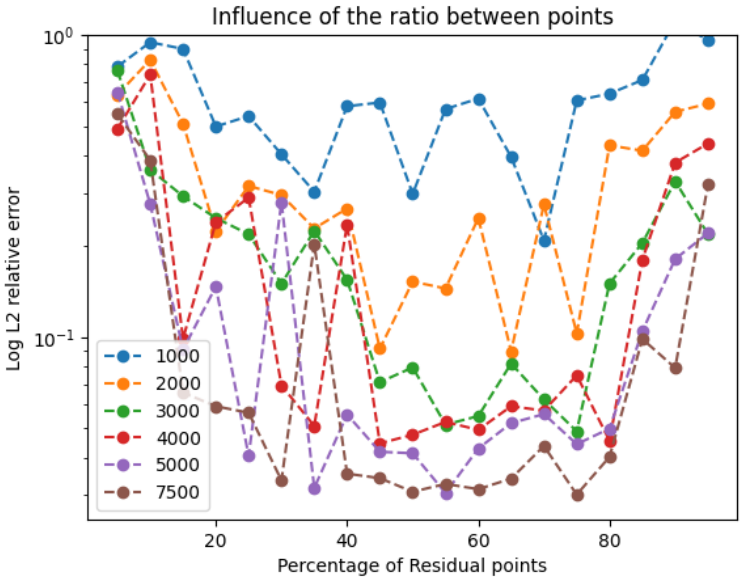
In order to measure the influence of different ratios of collocation points and initial/boundary condition points, an experiment was carried out as follows: with the total number of points fixed, the percentage of collocation points was varied, each discretization was extracted from a uniform distribution in order to have complete control over the ratio. The same network was used in all experiments (same architecture and same initial weights). For each ratio used, five discretizations obtained from some random distribution were taken; each one was used to train a network and generate an approximation at the same predefined points; from that, the average of the L_2 relative norm for the five approximations was calculated for each ratio. This average of five approximations was made to mitigate the randomness of the discretizations used to train the networks.

The predefined points were obtained from a regular discretization with fifteen points in each direction in the spatial domain and fifty in the time domain. The approximation at these points was used to calculate the L_2 relative norm, which was used to compare the approximations with the analytical solutions.

This experiment shows that it is important to be aware of the ratio between initial/boundary condition points and residual points; it also suggests that their importance increases as less data is used. When analyzing the ratio of the discretizations highlighted in the experiment in figure 23 (pink dots), the last discretization made in the time domain refinement contained 8281 placement points (73% of the total), while the closest in the spatial domain refinement contained 7,056 placement points (78% of the total), and the closest in refinement in both domains contained 9,702 placement points (79% of the total).

Although the ratios for the highlighted discretizations are different (figure 23), this

Figure 24 - Residual point ratio test



Legend: Each line represents a total of points while the percentage of residual points varies (x-axis)

does not explain the difference in the approximations yield. The discretizations that varied from 70% to 80% of the colocation points in figure 24 do not produce approximations as different as those seen in the Figure 23.

3.3.2 Network Architecture - Advection-Diffusion

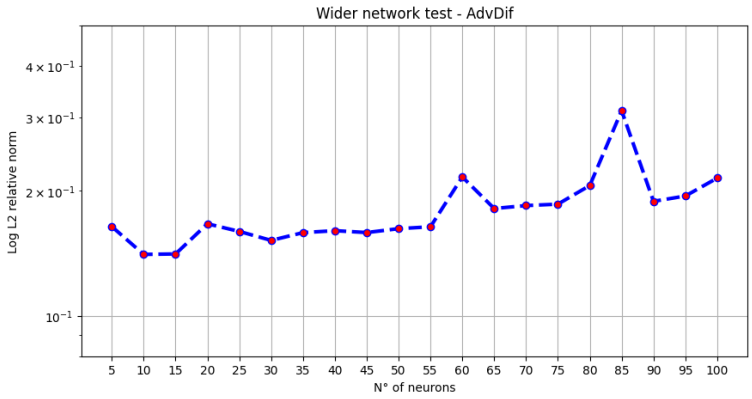
As in the previous cases, the first experiment aimed to find the optimum number of neurons in the hidden layer of the multilayer perceptron used to approximate the solution. For this objective, the number of hidden layers was fixed at three, while the number of neurons in each hidden layer was varied.

The activation function used in each neuron was the Gaussian Error Linear Unit (GELU) (HENDRYCKS; GIMPEL, 2016), and each network was trained for 5000 epochs. This number of hidden layers was chosen since it would provide enough expressibility to the network while not making the training impractical. It is possible to see the results in Figure 25.

The graph in Figure 25 shows the number of neurons used on the x-axis and the L_2 relative norm in the logarithmic scale on the y-axis. The number of neurons varied from 1 to 100 in the hidden layers, adding 5 at a time.

The second test, to discover the optimum number of hidden layers, was performed

Figure 25 - Wider network test - Advection-diffusion



Legend: Experiment where the number of neurons in the hidden layer was varied for the two-dimensional Advection-Diffusion problem

with a fixed number of neurons in each hidden layer while the number of hidden layers was varied.

Each hidden layer contained 10 neurons using the Gaussian Error Linear Unit (GELU) (HENDRYCKS; GIMPEL, 2016) as the activation function. Each network was trained for 5000 epochs. The results are displayed in Figure 26.

The graph in Figure 26 shows the number of hidden layers used on the x-axis and the L_2 relative norm in the logarithmic scale on the y-axis. The number of hidden layers was varied from 1 to 16, adding 1 at a time.

A final test was made to evaluate the best activation function, performed in the same way as the previous ones. The number of hidden layers was fixed at four, and the number of neurons per layer was fixed at ten. These parameters were chosen since the average error obtained in the previous test suggested that the network with this parametrization contains the necessary expressiveness for the problem.

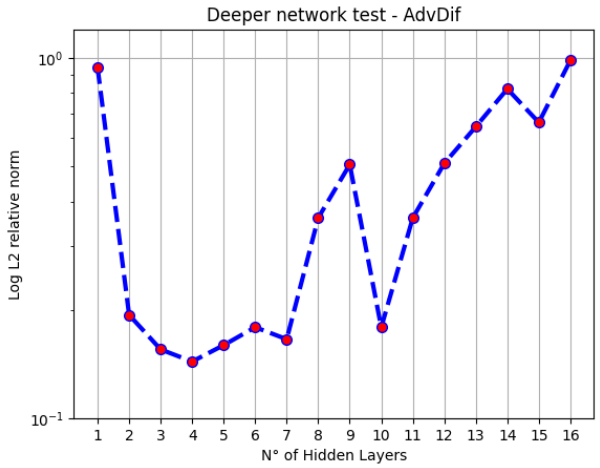
The results are on display in the figure 25.

The functions used (x-axis), as well as the L_2 relative norm (y-axis in logarithmic scale), are shown in Figure 27. The formulas and details of each one can be seen in table 1.

3.3.3 Numerical Results - Advection-Diffusion

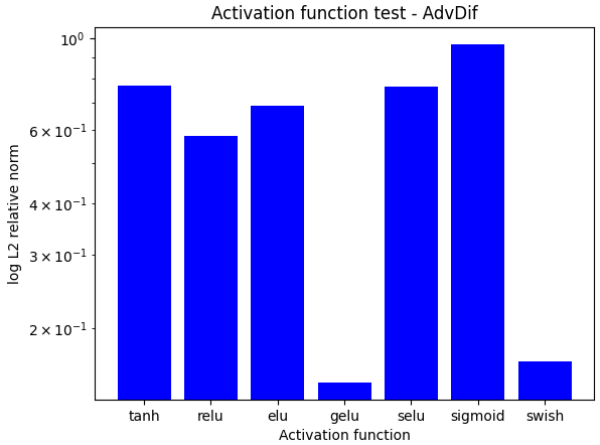
Two explicit finite difference methods were chosen in the two-dimensional advection-diffusion example to compare the results obtained with PINNs: the Forward time-centered

Figure 26 - Deeper network test - Advection-diffusion



Legend: Experiment where the number of hidden layers of the network was varied for the two-dimensional Advection-Diffusion problem

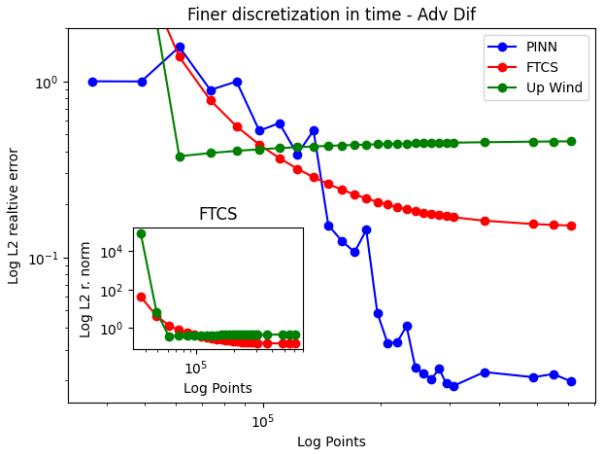
Figure 27 - Activation function test - Advection-diffusion



Legend: Experiment done with the same network architecture but with different activation functions

space method (FTCS) and the Upwind method. Both methods must obey a condition related to the step size in each dimension to produce a solution. As before, the discretizations used in implementing the finite difference methods were also used to train the network and to evaluate it. The plot of Figure 28 shows the l_2 relative norm obtained with the FTCS Method, the UpWind method, and also the error obtained with the PINN when more points were added in the time domain.

Figure 28 - Comparison with FDM (time) - Adv-Dif



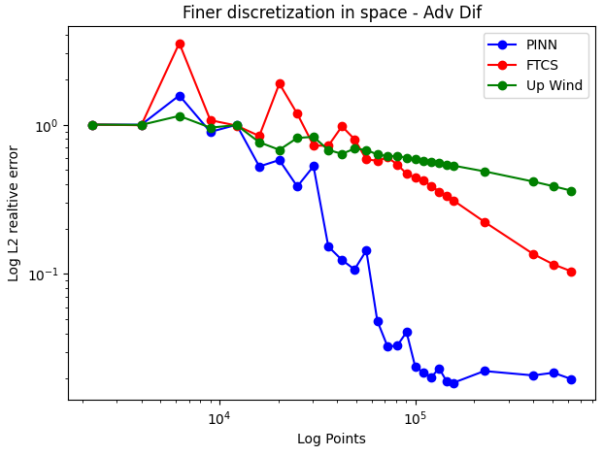
Legend: Plot relating the number of points to the L_2 relative norm, both axes in logarithmic scale. Each point is a distinct discretization, totaling 27 different discretizations.

In the time domain refinement experiment (Figure 19), both dimensions of the spatial domain were fixed with 35 points, while the time domain varied from 30 points to 250, adding 10 points, and after 25 points to 50, adding 50 points each time. It is possible to see that, while the FTCS method was not able to produce approximations for all discretizations and the CN method did not improve at all, the PINN approximations were able to provide consistently better results in almost all tested discretizations.

The test refining the space domain is displayed in Figure 29; this plot shows the L_2 relative error when a finer discretization in the space domain is done while the time domain is fixed.

In this experiment 29, the time domain was fixed at 10 points while both dimensions of the space domain ranged from 3 points to 250, adding 1 point, and then from 25 points to 50, adding 5 points each time.

Figure 29 - Comparison with FDM (space) - Adv-Dif



Legend: Plot relating the number of points to the L_2 relative norm, both axes in logarithmic scale. Each point is a distinct discretization, totaling 27 different discretizations

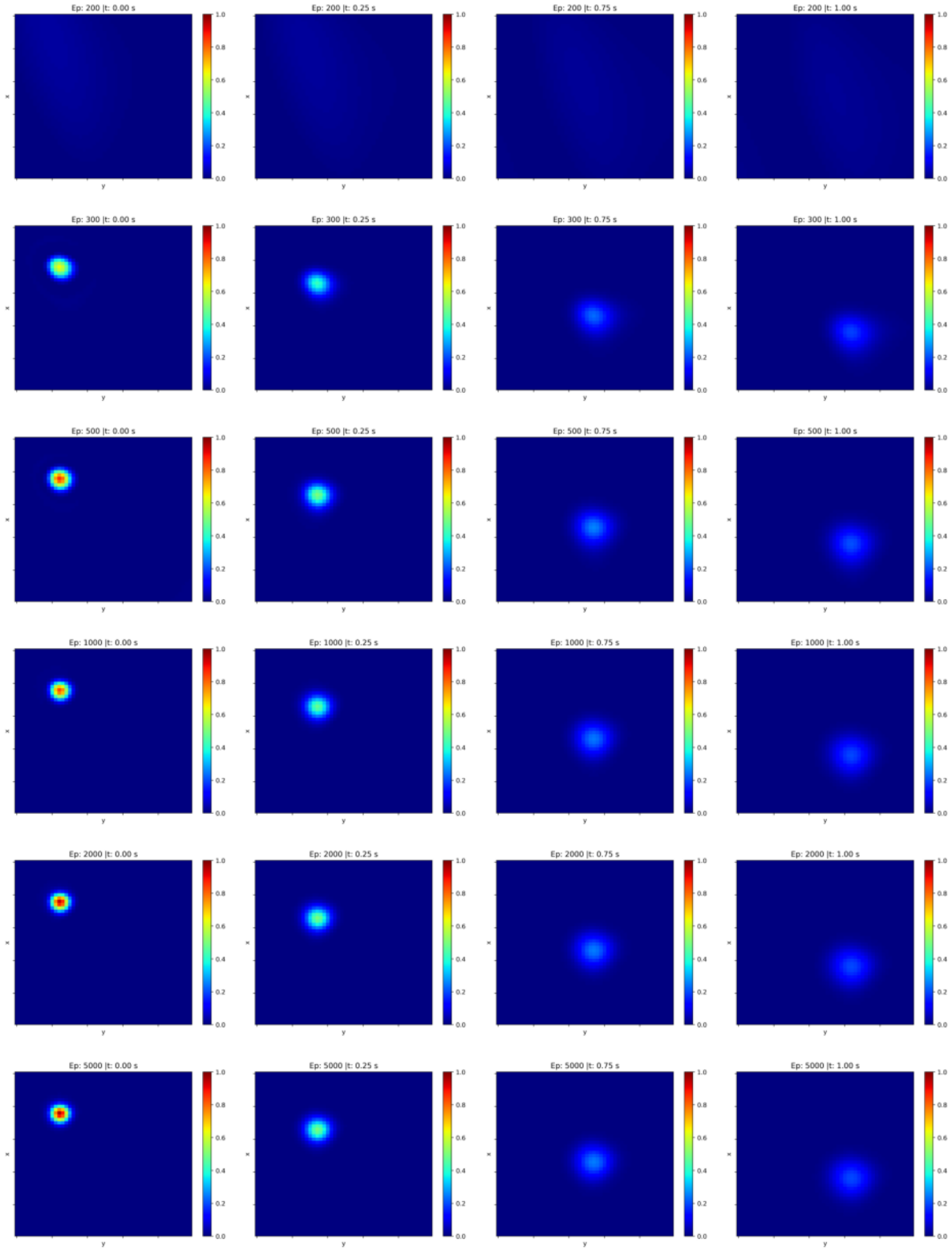
3.3.4 Visualization - Advection-Diffusion

The figure 30 shows the network approximations after some training epochs for specific time intervals (0, 0.25, 0.75, 1). The displayed approximations were produced after 0, 200, 500, 1000, 2000, and 5000 epochs. The approximations presented were obtained from a network trained with a grid with 50 points in each dimension of the spatial domain and 100 points in the time domain.

The network was composed of four hidden layers with ten neurons each, totaling 381 trainable parameters, each neuron having GELU as an activation function. The network was trained for 5000 epochs with a learning rate of 10^{-2} to 1000 epochs, 10^{-3} from 1000 to 3000 epochs, and 5.10 -4 from 3000 to 5000 epochs. Figure 31 displays the absolute error at selected time steps.

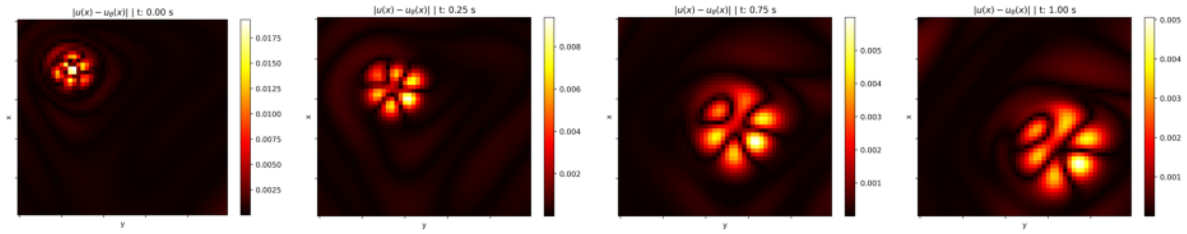
In the absolute error of the advection-diffusion problem (figure 31), once again it is possible to observe that, in the time slices of the domain that was shown, the one that presented the greatest error was the initial condition, with agreement of at least one decimal place, while the other time intervals managed to obtain agreement of at least two decimal places.

Figure 30 - Visualization of training - Adv-Dif



Legend: Solutions obtained during the training process of a physics-informed neural networks for the two dimension Advection-Diffusion equation at times 0, 0.25, 0.75 and 1 after 200, 300, 500, 1000, 2000 and 5000 epochs

Figure 31 - Absolute error - Adv-Dif



Legend: Absolute error for the two dimensions Advection-Diffusion equation at times 0, 0.25, 0.75 and 1

CONCLUSIONS

When addressing the estimation error in both heat problems, it was possible to conclude that, as the two-dimensional problem is more complex, more points are needed to obtain an adequate approximation. The error obtained with finer discretization in time and space indicates that, after a specific limit, the approximation obtained stops improving, suggesting that there is an optimal number of points to be used in training.

However, experiments with the advection-diffusion problem suggest that more data does not necessarily translate into a better approximation. Although the experiment varying the ratio of initial/limit points and residual points showed that it is important to be aware of this aspect, it does not explain the difference in the approximations obtained. It suggests that it is necessary to know the critical points of the problem domain to extract a discretization that prioritizes points in these regions; that way, the network will produce better approximations with these points.

As for the approximation error, all experiments that increased the number of hidden layers showed that deep networks are much more challenging to train. After some threshold, the approximations produced could not be compared with the solution of PDEs due to the fact that the networks were unable to learn. In all cases, it is possible to visualize an ideal number of hidden layers that achieve the expressiveness necessary to incorporate the complexity of the solution without suffering difficulties in the training process.

The experiment increasing the number of neurons in the hidden layers showed that, in all cases, wider networks do not offer greater training challenges, producing relatively close approximations while increasing the number of neurons. Since it is possible to see a limit at which approximations stop improving, it is suggested that there is an ideal number of neurons that can be used in each problem.

Regarding the decomposition of bias and variance, the results align with recent work on the behavior of these measurements as the width of neural networks increases. It suggests that this decomposition is, in fact, valid despite the differences that PINN training proposes. This analysis suggests that there is no increase in loss due to generalization error induced by the use of wider networks, which also suggests that the performance of the approximations should not be impacted by this aspect either.

Comparisons with the chosen finite difference method in the one-dimension heat problem showed that the PINNs are able to provide similar approximations to the Crank Nicholson method in these configurations. However, the implicit method clearly showed an optimal discretization as the number of points in both comparisons increased, and both converged for a similar error.

As for the FTCS, this method presented a significant increase in approximations as

the number of points in the time domain increased, the opposite happened when the number of points in the space domain increased, leading to non-convergence. In comparison to the approximations offered by PINN, there are no issues with convergence. However, there is a limitation that a finer discretization does not provide better approximations in both experiments.

The comparison with the FDM in the two-dimensional heat problem shows very different results; in the configurations used, the PINN showed a clear advantage over the two FDMs used (the explicit: FTCS and the implicit: ADI) in almost all scenarios. While the FTCS method had strict convergence constraints, the ADI method produced a solution in all cases, but those could not match the performance of the PINN approximations.

In the two-dimensional Advection-Diffusion problem, the results were similar to the two-dimensional heat problem; the approximations provided by PINN were much closer to the analytical solution in the majority of the discretizations tested. It is also important to state that this specific configuration of refinement in both spatial and time domains was chosen due to the fact that in discretizations similar to those used in the two-dimensional heat problem, both methods were unable to produce an approximation.

The visualization of the training process shows that the network approximates the PDE solution at all points simultaneously, unlike FDM, which can only produce a solution at time step n if the solution at time step $n-1$ is available.

It is possible to conclude that this creates an advantage for PINNs when compared to FDM, since errors from previous time steps are not perpetuated to the next steps. It can be argued that the reason why FDM methods produced a relatively large error approximating the solution of the two-dimensional heat equation is due to the propagation of the error throughout the time domain.

This independence of the approximate solutions at each time step gives PINn a unique advantage in terms of stability, and it is possible to argue that this is the reason for its better performance in the two-dimensional problem, where error propagation plays an even more crucial role when compared to the one-dimensional problem.

The future of the research is to improve the comprehensive understanding of the convergence of this method, exploring its different aspects in relation to the quality of the obtained result. The aim is to meticulously identify the vulnerabilities of the method, providing substantial improvements that elevate the solution from approximate solutions. The resulting approximations will be subjected to rigorous analysis, carefully comparing them with the most widely used methods in approaching the solutions of differential equations. This process will provide the community with an immediate assessment of the potential for implementing this method on specific problems.

REFERENCES

ABADI, M. et al. {TensorFlow}: a system for {Large-Scale} machine learning. In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. [S.l.: s.n.], 2016. p. 265–283.

BAYDIN, A. G. et al. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, Microtome Publishing, v. 18, p. 1–43, 2018.

BISHOP, C. Pattern recognition and machine learning. *Springer google schola*, Springer, v. 2, p. 531–537, 2006.

CLEVERT, D.-A.; UNTERTHINER, T.; HOCHREITER, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

CUOMO, S. et al. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, Springer, v. 92, n. 3, p. 88, 2022.

GAO, H.; SUN, L.; WANG, J.-X. Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, Elsevier, v. 428, p. 110079, 2021.

GEMAN, S.; BIENENSTOCK, E.; DOURSAT, R. Neural networks and the bias/variance dilemma. *Neural computation*, MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , v. 4, n. 1, p. 1–58, 1992.

HAO, Z. et al. Physics-informed machine learning: A survey on problems, methods and applications. *arXiv preprint arXiv:2211.08064*, 2022.

HASTIE, T. et al. *The elements of statistical learning: data mining, inference, and prediction*. [S.l.]: Springer, 2009. v. 2.

HENDRYCKS, D.; GIMPEL, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

HOFFMANN, K. A.; CHIANG, S. T. *Computational Fluid Dynamics Volume 1*. [S.l.]: Engineering Education System, 1998.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, p. 359–366, 1989.

HU, Z. et al. Augmented physics-informed neural networks (apinns): A gating network-based soft domain decomposition methodology. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 126, p. 107183, 2023.

INCROPERA, F. P. et al. *Fundamentals of heat and mass transfer*. [S.l.]: Wiley New York, 1996. v. 6.

JAGTAP, A. D.; KARNIADAKIS, G. E. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, Brown Univ., Providence, RI (United States), v. 28, n. 5, 2020.

JAGTAP, A. D.; KHARAZMI, E.; KARNIADAKIS, G. E. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, Elsevier, v. 365, p. 113028, 2020.

JAGTAP, A. D.; KHARAZMI, E.; KARNIADAKIS, G. E. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, Elsevier, v. 365, p. 113028, 2020.

KALITA, J. C.; DALAL, D.; DASS, A. K. A class of higher order compact schemes for the unsteady two-dimensional convection–diffusion equation with variable convection coefficients. *International Journal for Numerical Methods in Fluids*, Wiley Online Library, v. 38, n. 12, p. 1111–1131, 2002.

KARAA, S.; ZHANG, J. High order adi method for solving unsteady convection–diffusion problems. *Journal of Computational Physics*, Elsevier, v. 198, n. 1, p. 1–9, 2004.

KLAMBAUER, G. et al. Self-normalizing neural networks. *Advances in neural information processing systems*, v. 30, 2017.

LAGARIS, I. E.; LIKAS, A.; FOTIADIS, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, IEEE, v. 9, n. 5, p. 987–1000, 1998.

LEVEQUE, R. J. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. [S.l.]: SIAM, 2007.

LORENZO, D. D. et al. Physics informed and data-based augmented learning in structural health diagnosis. *Computer Methods in Applied Mechanics and Engineering*, Elsevier, v. 414, p. 116186, 2023.

NABIAN, M. A.; GLADSTONE, R. J.; MEIDANI, H. Efficient training of physics-informed neural networks via importance sampling. *Computer-Aided Civil and Infrastructure Engineering*, Wiley Online Library, v. 36, n. 8, p. 962–977, 2021.

NEAL, B. On the bias-variance tradeoff: Textbooks need an update. *arXiv preprint arXiv:1912.08286*, 2019.

NEAL, B. et al. A modern take on the bias-variance tradeoff in neural networks. *arXiv preprint arXiv:1810.08591*, 2018.

NOYE, B.; TAN, H. Finite difference methods for solving the two-dimensional advection–diffusion equation. *International Journal for Numerical Methods in Fluids*, Wiley Online Library, v. 9, n. 1, p. 75–98, 1989.

RAISSI, M.; PERDIKARIS, P.; KARNIADAKIS, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, Elsevier, v. 378, p. 686–707, 2019.

RAMACHANDRAN, P.; ZOPH, B.; LE, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

SALMAN, A. et al. Deep learning solver for solving advection-diffusion equation in comparison to finite difference methods. *SSRN Electronic Journal*, 01 2022.

SHALLAL, M.; JUMAA, B. F. Numerical solutions based on finite difference techniques for two dimensional advection-diffusion equation. *British Journal of Mathematics Computer Science*, v. 16, p. 1–11, 01 2016.

SHIN, Y.; DARBON, J.; KARNIADAKIS, G. E. On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes. *arXiv preprint arXiv:2004.01806*, 2020.

TENENBAUM, M.; POLLARD, H. *Ordinary differential equations: an elementary textbook for students of mathematics, engineering, and the sciences*. [S.l.]: Courier Corporation, 1985.

TIAN, Z.; GE, Y. A fourth-order compact adi method for solving two-dimensional unsteady convection–diffusion problems. *Journal of Computational and Applied Mathematics*, Elsevier, v. 198, n. 1, p. 268–286, 2007.

VIANA, F. A. et al. Estimating model inadequacy in ordinary differential equations with physics-informed neural networks. *Computers & Structures*, Elsevier, v. 245, p. 106458, 2021.

WONGSAIJAI, B.; SUKANTAMALA, N.; POOCHINAPAN, K. A mass-conservative higher-order adi method for solving unsteady convection–diffusion equations. *Advances in Difference Equations*, SpringerOpen, v. 2020, n. 1, p. 1–24, 2020.

YANG, L.; ZHANG, D.; KARNIADAKIS, G. E. Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal on Scientific Computing*, SIAM, v. 42, n. 1, p. A292–A317, 2020.

YANG, Z. et al. Rethinking bias-variance trade-off for generalization of neural networks. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2020. p. 10767–10777.

ZHU, Q.; LIU, Z.; YAN, J. Machine learning for metal additive manufacturing: predicting temperature and melt pool fluid dynamics using physics-informed neural networks. *Computational Mechanics*, Springer, v. 67, p. 619–635, 2021.