Aalto University
School of Science

Bachelor's Programme in School of Science

Physics-informed neural networks

Accuracy and convergence

Aamos Vaara

Bachelor's Thesis
2024

Aalto University
School of Science

Author Aamos Vaara

Title Physics-informed neural networks — Accuracy and convergence

Degree programme School of Science

Major Computer science

Teacher in charge Prof. Lauri Savioja

Advisor M.Sc Katsiaryna Haitsiukevich, Dr. Anssi Laukkanen

Collaborative partner VTT Technical Research Centre of Finland

Date 06.05.2024 **Number of pages** 28+1 **Language** English

Abstract

Physics-informed neural networks (PINNs) are a promising method for solving partial differential equations. PINNs have their own unique advantages compared with the traditional numerical solvers, such as being a mesh-free method and the ability to incorporate scattered measurement information to the solution process. In some scenarios, PINNs are the state-of-the-art method for solving the problem. Nevertheless, as with all methods, they have their own pitfalls. PINNs suffer from spectral bias, causality violations, increased complexity of the loss landscape and unbalanced loss terms to name a few. This thesis presents theory behind why these issues exists, and introduce methods to alleviate them. Examples of these methods include, curriculum training, causal weighting of the loss function, dynamic balancing of the loss terms using neural tangent kernel, Gaussian Fourier feature embedding and Wavelet activation function. This thesis is a literature review, however some methods are implemented and tested to achieve a more direct comparison between them. The methods tested include Gaussian Fourier feature mapping, NTK weight balancing and Wavelet activation function. The aforementioned methods, are investigated numerically using convection partial differential equation. Traditional PINNs failed to find satisfactory results for the convection problem, while combining dynamic weight balancing and Wavelet activation function achieved up to four magnitudes lower mean squared error compared with traditional PINN.

Keywords Convergence, Multi-objective optimization, Neural tangent kernel, Physics-informed neural network, Spectral bias

**Aalto-yliopisto
Perustieteiden
korkeakoulu**

Tekijä Aamos Vaara

Työn nimi Fysiikkainformoidut neuroverkot — Tarkkuus ja suppeneminen

Koulutusohjelma Perustieteiden korkeakoulu

Pääaine Tietotekniikka

Vastuuopettaja Prof. Lauri Savioja

Työn ohjaajat M.Sc Katsiaryna Haitsiukevich, Dr. Anssi Laukkanen

Yhteistyötaho VTT

Päivämäärä 06.05.2024 **Sivumäärä** 28+1 **Kieli** englanti

Tiivistelmä

Toisin kuin klassiset neuroverkot, fysiikkainformoidut neuroverkot eivät tarvitse dataa ratkaisufunktiosta, paitsi alkuehdoista. Verkkoa voidaan kouluttaa niin, että verkon tuloste toteuttaa osittaisdifferentiaaliyhtälön. Tämä voidaan toteuttaa derivoimalla verkon tulostetta ja sovittamalla se osittaisdifferentiaaliyhtälöön. Fysiikkainformoidun neuroverkon tappiofunktiossa on yleensä kolme termiä, jotka kuvaavat, kuinka hyvin verkko toteuttaa yhtälön, reunaehdot ja alkuehdot. Tappiofunktio on siis enemmän informatiivinen kuin yleensä neuroverkkojen tappiofunktio regressio-ongelmissa. Työn tavoitteena on tutkia fysiikkainformoitujen neuroverkkojen yleisiä ongelmia ja mahdollisia menetelmiä, joilla voidaan lieventää näiden ongelmien vaikutusta neuroverkkojen kouluttamisessa. Tässä työssä käsitellään monimutkaisempaa optimointiympäristöä, syy-yhteyksien rikkomista, spektrivinouma (Eng. Spectral bias), tappiofunktion termien epätasapainoista tilaa (Eng. Unbalanced loss terms) sekä spektrivinouma neurotangenttiytimen (Eng. Neural tangent kernel) kautta. Työ on pääosin kirjallisuuskatsaus, jossa on lisäksi numeerinen testiosio. Numeerisessa testiosiossa tutkitaan Gaussista Fourier-piirrekuvausta (Eng. Gaussian Fourier-feature mapping), Wavelet-aktivaatiofunktiota sekä tappiofunktion tasapainottamista neurotangenttiytimeen pohjautuvalla menetelmällä.

Eteenpäin suuntautuvat neuroverkot kärsivät spektrivinoumasta, mikä tarkoittaa, että tämän tyyppiset neuroverkot suosivat optimoinnin aikana alhaisia taajuuksia ratkaisufunktiosta. Tämän vuoksi neuroveron tuottama ratkaisu saattaa olla liian yksinkertainen, sillä verkko ei ole oppinut korkeamman taajuuden ominaisuuksia funktiosta. Kyseinen ilmiö toistuu myös fysiikkainformoiduissa neuroverkoissa. Gaussinen Fourier-piirrekuvaus pyrkii muokkaamaan taajuuksia, joita neuroverkko suosii, jotta verkko oppisi myös korkeammat taajuudet. Wavelet-aktivaatiofunktio perustuu Fourier-piirrekuvaukseen, mutta toimii aktivaatiofunktiona ja pyrkii samaan kuin Gaussinen Fourier-piirrekuvaus. Monimutkaisempi optimointiympäristö johtuu fysiikkainformoidun neuroverkon monimutkaisemmasta tappiofunktiosta, joka koostuu useiden termien summasta. Näiden termien tasapainottaminen optimoinnin aikana on keskeistä, jotta sopiva paikallinen minimi voidaan löytää. Termejä voidaan tasapainottaa neurotangenttiytimen avulla, josta saadaan informaatiota jokaisen termin suppenemisnopeudesta. Neurotangenttiytimeen pohjautuvalla tappiofunktion tasapainottamisella pyritään tasaamaan suppenemisnopeudet eri termeillä.

Menetelmiä tutkittiin käyttäen konvektion osittaisdifferentiaaliyhtälöä, jossa on korkean taajuuden ominaisuuksia. Jokainen menetelmäyhdistelmä laskettiin viisi kertaa. Tuloksista on havaittavissa, että klassinen fysiikkainformoitu neuroverkko ei kykene ratkaisemaan ongelmaa. Tappiofunktion tasapainottaminen ei korjaa spektrivinoumaa, joten keskineliövirhe on hyvin samanlainen kuin klassisella fysiikkainformoidulla neuroverkolla. Sen sijaan, jos tappiofunktion tasapainottamisen yhdistää menetelmään, joka pyrkii muokkaamaan spektrivinoumaa, menetelmä vähentää hajontaa tuloksissa. Wavelet-aktivointifunktion ja tappiofunktion tasapainottaminen tuotti lähes neljän magnitudin eron klassiseen fysiikkainformoituun neuroverkkoon keskineliövirheessä.

Avainsanat Fysiikkainformoitu neuroverkko, monitavoiteoptimointi, neurotangenttiydin, spektrivinouma, suppeneminen

Contents

Abbreviations

CAN-PINN	Coupled-automatic-numerical differentiation PINN
CFD	Computational fluid dynamics
FEM	Finite element method
FNO	Fourier neural operator
FVM	Finite volume method
MLP	Multilayer perceptron
MSE	Mean squared error
NTK	Neural tangent kernel
PDE	Partial differential equation
PINO	Physics-informed neural operators
PINN	Physics-informed neural network
SA-PINN	Self-adaptive PINN
Seq2Seq	Sequence-to-sequence
SGD	Stochastic gradient descent
SGDM	Stochastic gradient descent with momentum
SPINN	Separable PINN

1 Introduction

The ability to simulate physical systems is a cornerstone of modern engineering. Typically, analytical solutions cannot be found, as is the case for many well-known equations such as the Navier-Stokes equations. Therefore, traditionally, solving partial differential equations is reliant on numerical methods such as the finite volume method (FVM) or the finite element method (FEM). These methods typically introduce a dense mesh to approximate derivatives, making them computationally demanding.

In addition to traditional techniques for solving PDEs, neural network based methods have been developed for this task. Neural networks are a promising method as they are universal approximators of continuous functions and non-linear continuous operators [18, 4], given that the network has sufficient size. Neural networks as approximators for a PDE solution can work with an arbitrary set of points inside the problem domain, which means that they do not require the problem domain to be discretized, making them a mesh-free method [26]. Unlike numerical methods, that are limited to the grid on which the solution was constructed, one can evaluate any point within the domain using a trained model. Moreover, integrating sparse measurements into the solution process is more straightforward with neural networks compared with other traditional methods. Aforementioned property is especially useful when addressing ill-posed problems. These properties of neural networks make them a promising alternative method to solving PDEs. In certain domains, such as weather forecasting and protein structure prediction, deep learning methods provide state-of-the-art solutions [15, 25, 23] and [11].

One disadvantage of deep learning methods is that they require vast amount of data to achieve acceptable performance. Thus, employing deep learning methods in some fields has been challenging, especially when the cost of data acquisition is high. For example, gathering just one data point from metal fatigue testing costs several hundred to thousands of euros. It is evident then, that data in such domains are typically scarce and inherently noisy due to experimental data gathering. Even with ample data, physical system dynamics modelled by neural networks may exhibit violation of the underlying physical constraints or governing laws due to the presence of noise in the training data [1]. Traditional data-driven deep learning methods can only be as good as the data provided to the model. These issues limit their utility in computational engineering domains as noted in [1].

Physics-informed neural networks (PINNs) were first introduced in [14]. The form of PINNs that this thesis considers was introduced in [29] to address the limited data availability and violations of governing physical laws. Data is not the only information available about these systems; there is also extensive knowledge on the physics that governs the system. The idea of PINNs is to leverage the partial differential equation (PDE) derived from the first principles provided by physics. The PDE is incorporated into the loss function itself and reinforces the PINN to converge on solutions that satisfy the underlying PDE [29]. Incorporating physical prior knowledge into the loss function limits the space of possible solutions, enabling the network to converge faster and generalize better, even with only a few training samples [29]. As the authors in [29] note, "there exist a vast amount of prior knowledge that is currently not being utilized in modern machine learning practice." PINNs have made it feasible to employ deep learning methods in new fields. Some examples include fluid mechanics [2, 32], material science [27, 39] and power systems [10].

PINNs incorporate physics into the loss function, yet they face challenges that hinder their ability to solve the given problem. Some of these challenges include a more complex loss landscape [13] and addressing multi-scale losses, where different terms of loss function may have drastically varying scales that cannot be balanced during the pre-processing phase [38]. PINNs are also biased to learn low-frequency features, of the target function, corresponding to a less complex function, known as spectral bias [7], causality violation [35] and the incapability of vanilla PINNs to extrapolate solutions beyond the training domain [8].

Vanilla PINNs used for standard forward problems, in the setting of scientific computation, do not meet the computational performance and accuracy requirements [19]. Nevertheless, as noted in [29], traditional methods have been studied for over 50 years, and PINNs were not intended to replace traditional methods of solving PDEs but rather to compliment them [29]. It is then unsurprising, that traditional methods tend to outperform PINNs with well-posed forward problems in terms of accuracy and performance. PINNs are one method among many, and each method has its own domain where it is applicable. PINNs may not be the optimal method for every scenario. However, PINNs are a versatile method. PINNs can handle ill-posed forward and inverse problems. Even problems that have only a partly known PDE. In some setting, PINNs perform better, in terms of accuracy and efficiency, than any existing solvers. As is the case in computational fluid dynamics (CFD) when scattered spatio-temporal data is available for the problem [2].

The goal of this thesis is to investigate what are the common issues found with PINNs and how to potentially address them. This thesis is an overview of the known pitfalls of PINNs and proposed solutions that aim to mitigate these problems. This thesis will examine the adaptive weight balancing of multiple loss terms, training schemes, Fourier feature mapping and modified activation function.

This thesis follows the style of [38], which is an excellent guide on training PINNs. However, the angle at which the methods are presented differ.

The rest of this study is broken down as follows. The Section 2 introduces the formalization of physics-informed neural networks. Section 3 delves deeper into previously mentioned problems and provides methods for addressing them. Section 4 contains numerical testing of a few select methods. Lastly, Section 5 contains discussion and future works.

2 Physics-informed neural networks

Physics-informed neural networks differ from standard neural networks primary by their loss function. PINNs also have a more restrictive set of activation functions, where differentiability is the limiting factor for the possible activation functions. This requirement is due to the evaluation of the loss function, which typically employs the use of automatic differentiation, as will be evident in the following subsections.

2.1 Formalization

The following is a formalization of PINNs closely following the paper [38] that is based on [29].

General form of parameterized and non-linear partial differential equation can be written as

$$\dot{u} + \mathcal{N}[u] = 0, \quad t \in [0, T], x \in \Omega. \tag{1}$$

With initial and boundary conditions as

$$\begin{aligned} u(0, x) &= g(x), \quad x \in \Omega \\ B[u] &= 0, \qquad t \in [0, T], x \in \partial\Omega, \end{aligned} \tag{2}$$

where $\mathcal{N}[\cdot]$ is a linear or non-linear differential operator, u being the unknown solution to the PDE, $\dot{u} = \frac{\partial u}{\partial t}$ is the derivative with respect to t and $B[\cdot]$ is a boundary operator.

The unknown function $u(t, x)$ from PDE (1) – (2) provided by a deep neural network is denoted as $u_\theta(t, x)$, where θ denotes the parameters of the PINN.

The extent to which the network satisfies the partial differential equation is given by the PDE residual, which is defined as

$$R_\theta(t, x) = \dot{u_\theta} + \mathcal{N}[u_\theta(t, x)]. \tag{3}$$

Physics-informed neural networks are trained by minimizing the following loss function

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathrm{ic}}(\theta) + \mathcal{L}_{\mathrm{bc}}(\theta) + \mathcal{L}_{\mathrm{r}}(\theta), \tag{4}$$

where $\mathcal{L}_{\mathrm{ic}}(\theta)$ stands for initial condition loss, $\mathcal{L}_{\mathrm{bc}}(\theta)$ denotes the boundary loss and finally $\mathcal{L}_{\mathrm{r}}(\theta)$ is the residual loss of the PDE defined earlier.

Aforementioned loss terms are defined more precisely as

$$\mathcal{L}_{\text{ic}}(\theta) = \frac{1}{N_{\text{ic}}}\sum_{i=1}^{N_{\text{ic}}}\left|u_\theta\left(0,\ x_{i,\ \text{ic}}\right) - g(x_{i,\ \text{ic}})\right|^2 \quad \propto \int_\Omega \left|u_\theta\left(0,x\right) - g\left(x\right)\right|^2 dx \tag{5}$$

$$\mathcal{L}_{\text{bc}}(\theta) = \frac{1}{N_{\text{bc}}}\sum_{i=1}^{N_{\text{bc}}}\left|B\left[u_\theta\right]\left(x_{i,\ \text{bc}}, t_{i,\ \text{bc}}\right)\right|^2 \quad \propto \int_0^T\int_{\Omega_{\text{bc}}} \left|B\left[u_\theta\right]\left(x,t\right)\right|^2 dxdt \tag{6}$$

$$\mathcal{L}_{\text{r}}(\theta) = \frac{1}{N_{\text{r}}}\sum_{i=1}^{N_{\text{r}}}\left|R_\theta\left(t_{i,\text{r}},\ x_{i,\text{r}}\right)\right|^2 \quad \propto \int_0^T\int_\Omega \left|R_\theta\left(t,\ x\right)\right|^2 dxdt. \tag{7}$$

Here $\left\{x_{i,\text{ic}}\right\}_{i=1}^{N_{\text{ic}}}$ and $\left\{t_{i,\text{r}}, x_{i,\text{r}}\right\}_{i=1}^{N_{\text{r}}}$ can be points of mesh or points randomly sampled from the domain. The points $\left\{t_{i,\text{bc}}, x_{i,\text{bc}}\right\}_{i=1}^{N_{\text{bc}}}$ are sampled from the boundary domain Ω_{bc}. Figure 1 showcases a possible one sample of the previously discussed points.

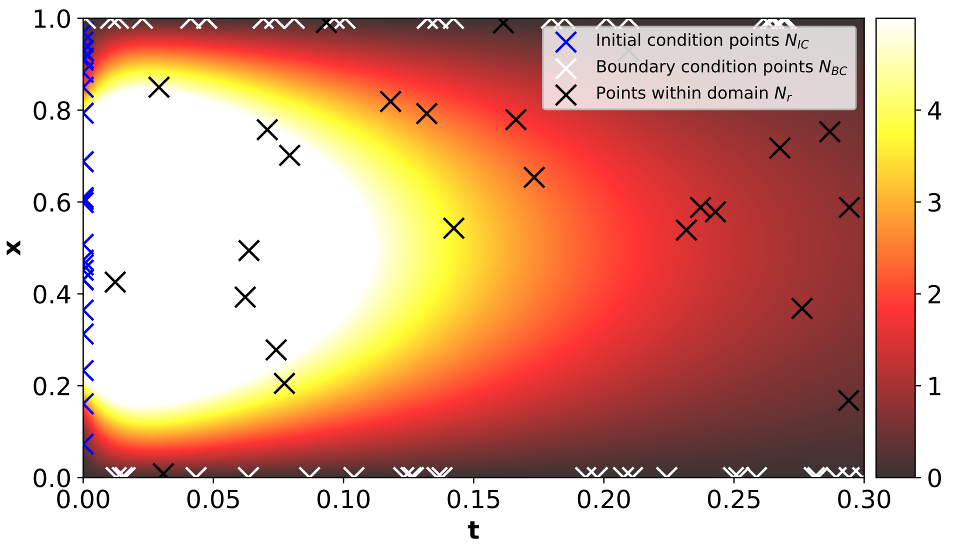


Figure 1: One sample of points $\left\{x_{i,\text{ic}}\right\}_{i=1}^{N_{\text{ic}}}$, $\left\{t_{i,\text{r}}, x_{i,\text{r}}\right\}_{i=1}^{N_{\text{r}}}$ and $\left\{t_{i,\text{bc}}, x_{i,\text{bc}}\right\}_{i=1}^{N_{\text{bc}}}$ from 1D heat equation solution.

It is important to note that in data-free PINNs there is no data on the actual solution $u(t,x)$ on the points $\left\{t_{i,\text{r}}, x_{i,\text{r}}\right\}_{i=1}^{N_{\text{r}}}$ and $\left\{t_{i,\text{bc}}, x_{i,\text{bc}}\right\}_{i=1}^{N_{\text{bc}}}$. The network is evaluated on these points, and its ability to satisfy the boundary conditions and the PDE is measured by the loss terms $\mathcal{L}_{\text{bc}}(\theta)$ and $\mathcal{L}_{\text{r}}(\theta)$. Data is available regarding the initial condition as $u(0,x) = g(x)$ is given.

In data-driven methods the loss is usually defined simply as

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left|u_\theta\left(t_i,x_i\right) - u\left(t_i,\ x_i\right)\right|^2, \tag{8}$$

where $u(t,x)$ is the solution provided by other methods or gathered by experiments. In the case where data on $u(t,x)$ exists, it can be incorporated into the loss function defined in (4) by simply adding the data-driven loss (8) to (4). An example of this approach is a physics-informed neural operators (PINO) that combines data and physics loss to approximate the solution operator [16].

The loss function defined in (4) effectively regularizes the space of admissible solutions, disregarding solutions that do not satisfy the underlying PDE, boundary and initial conditions [29]. In other words, solutions that adhere to the physical constraints are favoured. This stands in stark contrast to the data-driven methods, where the loss function provides information only on the point-wise difference of the solution given by the neural network and the actual solution. An important thing to notice also is the fact that in order to train the data-driven model one requires a sample of the target function $u(t,x)$ from the whole domain. Whereas for PINNs, only points from the initial conditions with targets are needed. However, in most cases, there is some data of the solution $u(t,x)$ outside of the initial conditions. In these scenarios, the loss of a data-driven method is augmented to the loss defined in (4) as descried earlier.

2.2 Example: 1D heat equation

For a concrete example consider a 1D heat equation on domain $x \in [0,L]$ and $t \in [0,T]$ defined as

$$\dot{u} = \alpha \frac{\partial^2 u}{\partial x^2}. \tag{9}$$

For completeness it is written in the form earlier used in (1)

$$\dot{u} - \alpha \mathcal{N}[u] = 0, \tag{10}$$

with boundary condition

$$u(0,t) = u(L,t) = 0 \tag{11}$$

and initial condition as $u(x,0) = sin(\frac{\pi x}{L}), x \in [0,L]$. Differential operator $\mathcal{N}[\cdot]$ is defined as a second order derivative of u with respect to x. For simplicity, the thermal diffusivity term is set to $\alpha = 1$.

Loss in training is defined here as

$$\mathcal{L}_{\text{ic}} = \frac{1}{N_{\text{ic}}} \sum_{i=1}^{N_{\text{ic}}} \left| u_\theta\left(0, x_{i,\text{ic}}\right) - \sin\left(\frac{\pi x_{i,\text{ic}}}{L}\right) \right|^2 \tag{12}$$

$$\mathcal{L}_{\text{bc}} = \frac{1}{N_{\text{bc}}} \sum_{i=1}^{N_{\text{bc}}} \left| u_\theta\left(t_{i,\text{bc}}, x_{i,\text{bc}}\right) \right|^2. \tag{13}$$

The boundary points $(t_{\text{bc}}, x_{\text{bc}})$ can be formed by randomly sampling sets $\left\{x_{i,\text{bc}}\right\} \in \{0, L\}$ and $\left\{t_{i,\text{bc}}\right\} \in [0,T]$. Similarly, for initial condition points $(0, x_{\text{ic}})$ can be formed by sampling the set $\left\{ x_{i,\text{ic}}\right\} \in [0,L]$ randomly.

Typically, the computation of the PDE residual loss is performed by using automatic differentiation. However, there are variants of PINNs that use the finite difference method in addition to automatic differentiation as is with CAN-PINNs demonstrated in [5]. For the sake of demonstrating concretely where the PDE residual loss comes from, the finite difference method was chosen.

$$\frac{\partial u_\theta}{\partial t} \approx \frac{u_\theta\left(t_r + \Delta t, x_r\right) - u_\theta\left(t_r, x_r\right)}{\Delta t} \qquad = K_t(t_r, x_r) \tag{14}$$

$$\frac{\partial^2 u_\theta}{\partial x^2} \approx \frac{u_\theta\left(t_r, x_r + \Delta x\right) - 2u_\theta\left(t_r, x_r\right) + u_\theta\left(t_r, x_r - \Delta x\right)}{\Delta x^2} \qquad = K_{xx}(t_r, x_r). \tag{15}$$

Then the PDE residual $R_\theta(t,x)$ is defined as

$$R_\theta(t,x) = K_t(t,x) - K_{xx}(t,x). \tag{16}$$

Subsequently the PDE residual loss is therefore defined as

$$\mathcal{L}_{\mathrm{r}}(\theta) = \frac{1}{N_r} \sum_{i=1}^{N_{\mathrm{r}}} \left| K_t\left(t_{i,\mathrm{r}}, x_{i,\mathrm{r}}\right) - K_{xx}\left(t_{i,\mathrm{r}}, x_{i,\mathrm{r}}\right) \right|^2. \tag{17}$$

Residual points $\{t_{i,r}, x_{i,r}\}$ can be randomly sampled from the whole domain.

As was mentioned previously, the terms $K_{xx}(t,x)$ and $K_t(t,x)$ can be obtained from the automatic differentiation, eliminating the need to use the finite difference method. Calculation of PDE residual for 1D heat equation using automatic differentiation in PyTorch is demonstrated in Listing 1.

```python
def u(model, X):
    return model(X)

def PDE_residual(model, t,x):
    x = torch.autograd.Variable(x, requires_grad=True)
    t = torch.autograd.Variable(t, requires_grad=True)
    X = torch.cat((x,t), dim = 1)
    u_val = u(model,X)
    u_x = torch.autograd.grad(u_val.sum(),x,create_graph
        =True)[0]
    u_t = torch.autograd.grad(u_val.sum(),t,create_graph
        =True)[0]
    u_xx = torch.autograd.grad(u_x.sum(),x,create_graph=
        True)[0]
    return u_t - u_xx
```

Listing 1: Example code block for defining the PDE residual function in PyTorch.

3 Challenges with PINNs

This Section introduces the increased complexity of loss landscape, unbalanced loss terms and delves deeper into spectral bias. The subsections will introduce a problem and then showcase a possible method for addressing it. This thesis focuses mainly on PINNs that are based on fully-connected multi-layer neural networks (MLPs). However, there are other types of promising PINN architectures, such as PINNsFormer that is based on the Transformer architecture [40]. Nevertheless, these PINN types are outside the scope of this thesis and are only briefly discussed in Sections 3.3 and 5.

3.1 Increased complexity of loss landscape

The loss landscape describes the loss in terms of the network parameters, from which it is evident that the model architecture, loss function, and the problem itself plays a role in determining how complex the landscape is. Including the PDE residual into the loss function may increase the complexity of the loss landscape [13], thereby making finding the global minima—that is, optimizing the model—more challenging.

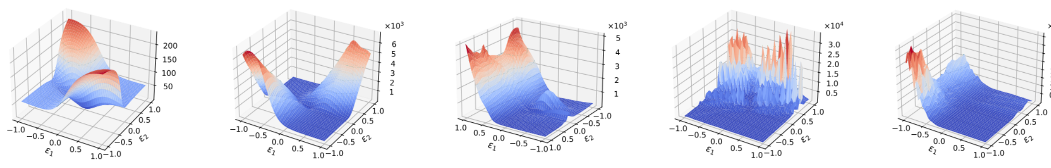
One proposed solution to this problem is curriculum regularization, as demonstrated in [13] using convection problem of the following form

$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0 \quad x \in \Omega, t \in [0, T] \tag{18}$$

$$u(x, 0) = sin(x) \tag{19}$$

$$u(0, t) = u(2\pi, t), \tag{20}$$

where $\frac{\partial u}{\partial t}$ is the derivative of $u(x, t)$ with respect to t. Correspondingly, $\frac{\partial u}{\partial x}$ is the derivative of $u(x, t)$ with respect to x. Lastly, β is the convection coefficient.



(a) $\beta = 1.0$ (b) $\beta = 10.0$ (c) $\beta = 20.0$ (d) $\beta = 30.0$ (e) $\beta = 40.0$

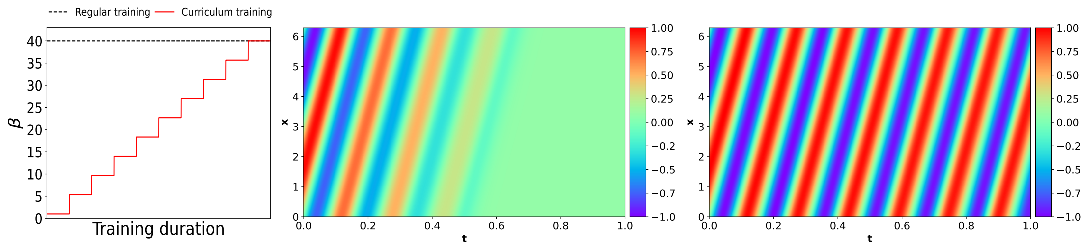
Figure 2: Loss landscape of trained model on Equation (18) – (20) with varying β values. Figure is from the paper [13].

From the Figure 2 it is clear that the loss landscape becomes increasingly more complex as the β coefficient increases. The same phenomena can be observed with Navier-Stokes equations with increasing Reynolds numbers [38].

In contrast to randomly initializing the model parameters and starting at a random point in the loss landscape, the model can be trained for β_1 for which $\beta_1 < \beta_2$. Since β_1 has a less complex loss landscape, compared to β_2, it is more straightforward to find the optimal parameters for β_1, denoted as $\theta_1^\star$. Furthermore, the optimal parameters $\theta_1^\star$

are closer to β_2 optimal parameters, $\theta_2^\star$, the smaller the difference is between β_1 and β_2. In other words, given $|\beta_1 - \beta_2| < \epsilon$ where ϵ is sufficiently small then $\theta_1^\star \approx \theta_2^\star$. Therefore, the parameters $\theta_1^\star$ can act as a better starting point for the model than random initialization.

Curriculum regularization extends this idea to taking multiple steps between the target β and initial β_0. This training scheme is visualized in Figure 3 using the convection equation (18) – (20). PINNs are known to struggle with equation (18) – (20) when β coefficient is large. However, finding sufficient parameters $\theta^\star$ for a low β coefficient with PINNs is relatively easy. The authors in [13] observed that the error is almost two magnitudes lower using curriculum regularization compared with a regular training scheme.



(a) β during training **(b)** Regular training **(c)** Curriculum training

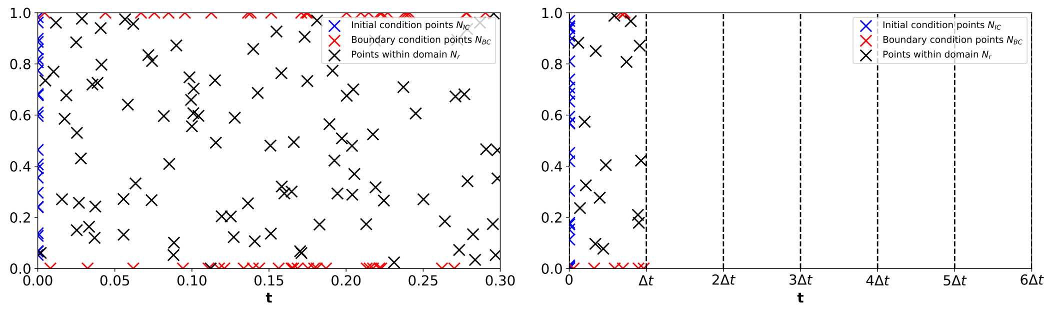
Figure 3: PINN trained on different training schemes for equation (18) – (20) with $\beta = 40$.

However, the step size from β_n to β_{n+1} is not well-defined [7]. The step size, $\Delta\beta = |\beta_n - \beta_{n+1}|$, is in relation to how different the loss landscape is between β_n and β_{n+1}. Curriculum regularization effectively introduces a new hyper parameter for the step size $\Delta\beta$ to the model that requires trial and error to optimize, as noted in [7]. Furthermore, it is more complex to implement when PDE has multiple coefficients. One has to know which set of coefficients are easy to solve and what are the step sizes with the coefficients such that the network parameters of the old coefficient set are reasonably close to the new optimal network parameters for the new coefficient set.

3.2 Causality violations

Time dependent PDEs such as convection problem (18) – (20) have inherent causality. The solution propagates from the initial condition further in time accordingly to the PDE. Continuous-time PINNs, optimized with a method based on gradient descent, have a tendency of minimizing $\mathcal{L}_\mathrm{r}(\theta)$ for larger time values t [35]. This is due to the PDE residual loss $\mathcal{L}_\mathrm{r}(\theta)$ potentially having a more complex optimization environment near points where accurate approximations exists, compared with to the vicinity of points that do not have accurate approximations. For example, the convection problem (18) – (20), any constant function minimizes the PDE residual loss $\mathcal{L}_\mathrm{r}(\theta)$ and the boundary loss $\mathcal{L}_\mathrm{bc}(\theta)$, but when there are accurate approximations of the previous time step, such as close to the initial conditions, there exists only one function that minimizes both the PDE residual loss $\mathcal{L}_\mathrm{r}(\theta)$ and the boundary loss $\mathcal{L}_\mathrm{bc}(\theta)$. Since the loss function (4) is minimized simultaneously for all the time points in regular training, the network is biased on minimizing $\mathcal{L}_{\mathrm{r}}(\theta)$ before the predictions at previous time steps are accurate, which may lead the model violating temporal causality [35].

One solution to respecting temporal causality is to segment time into smaller segments. This idea was introduced in [20] but this thesis considers the sequence-to-sequence (seq2seq) method demonstrated in [13]. In seq2seq, time is segmented into segments Δt and the model is trained on previous segment before moving to the next segment $2\Delta t$.



(a) Regular sampling **(b)** Seq2Seq sampling

Figure 4: Comparison between normal sampling and seq2seq sampling

The scheme demonstrated in Figure 4b aims at having the PINN learn the previous time segment before moving onto the next one, alleviating the problem of propagating the solution before previous time step predictions are accurate.

Other method is to formulate weights for each term in the PDE residual loss $\mathcal{L}_{\mathrm{r}}(\theta)$ such that the PINN is forced to minimize the previous time step losses before moving onto next time step. This method was demonstrated in [35] and given as

$$\mathcal{L}_{\mathrm{r}}(\theta) = \frac{1}{N_{\mathrm{r}}} \sum_{i=1}^{N_{\mathrm{r}}} w_i \mathcal{L}_{\mathrm{r}}(t_i\,, \theta)\,. \tag{21}$$

The weights are defined as

$$w_i = \exp\left(-\,\epsilon \sum_{k=1}^{i-1} \mathcal{L}_{\mathrm{r}}(t_k, \theta)\right), \text{ for } i = 2,3,\ldots,N_{\mathrm{r}}, \tag{22}$$

here $\mathcal{L}_{\mathrm{r}}(t_i\,, \theta) = \left|R_\theta\left(t_{i,\mathrm{r}}, x_{i,\mathrm{r}}\right)\right|^2$ but the elements must be ordered in such a way that $t_i \leq t_{i+1}$ in order to achieve the desired outcome. The total loss is given by

$$\mathcal{L}_{\mathrm{r}}(\theta) = \frac{1}{N_{\mathrm{r}}} \sum_{i=1}^{N_{\mathrm{r}}} \exp\left(-\,\epsilon \sum_{k=1}^{i-1} \mathcal{L}_{\mathrm{r}}(t_k, \theta)\right) \mathcal{L}_{\mathrm{r}}(t_i\,,\theta)\,. \tag{23}$$

The weights in equation (22) increase relative to the sum of the PDE residual loss of the previous time steps [38]. Therefore, the term $\mathcal{L}_{\mathrm{r}}(t_i, \theta)$ won't be minimized until the sum of the PDE residuals from the previous time steps $t_0, t_1, \ldots, t_{i-1}$ are sufficiently small [35]. The parameter ϵ is a user defined hyperparameter referred to as the causality parameter [35]. The causality parameter determines effectively what is sufficiently small before moving onto the next time step. Having insufficiently small ϵ may fail to impose temporal causality and too small ϵ may make the optimization problem harder [38].

Both of these methods aim at respecting temporal causality and can be employed at the same time. Authors in [38] see the weighting of PDE residual loss demonstrated in equations (21) – (23) as a crucial enhancement for sequence-to-sequence type training scheme as the model may still violate temporal causality within each Δt window.

These methods however have ambiguity as to what is a proper Δt or ϵ such that temporal causality is respected while keeping computational costs reasonable.

An alternative approach, which is outside the scope of this thesis but briefly mentioned here, was demonstrated in [9]. One can use an ensemble of PINNs, with different weights, to determine whether to include the point (x_k, t_k) into the loss function $\mathcal{L}_\mathrm{r}(\theta)$ based on ensemble agreement as a criterion [9]. Furthermore, this method yields an approximation of certainty for the solution on each point $u(t, x)$ [9].

3.3 Spectral bias

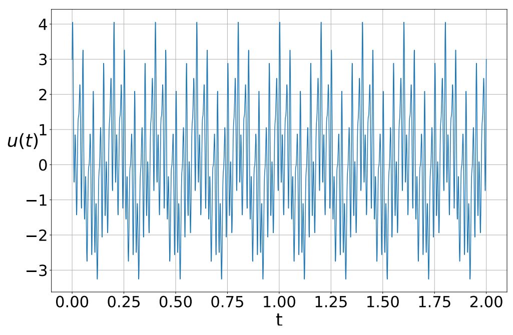
Spectral bias refers to the tendency of fully-connected feedforward neural networks to converge faster to the lower frequency features of the target function compared with the higher frequency features during training [28]. This means that the components of the target function corresponding to lower frequencies, meaning lower complexity, are learned first [3].

Figure 5 displays this phenomena. The function u is created by superposing sin functions with frequencies 5, 20, 40, 60 and 80 with random phases. A fully-connected feedforward neural network is employed to learn the function u and from approximations given by the network the frequency spectrum is compared to the reference seen in Figure 5b. From the Figure 5c the different convergence rates between the frequencies can be observed.

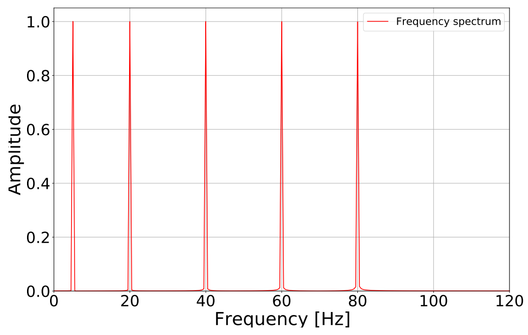
Difficulty in learning to approximate the solution of the convection equation (18) – (20) is related to spectral bias as β coefficient gains larger values, the target function gradually gains higher frequency features. Overcoming spectral bias with increasing β is progressively more difficult. Making optimizing harder for higher β coefficient values which can be seen as the complexity of loss landscape shown in Figure 2.

The paper [7] showed how the choice of the optimizer can alleviate the problem with spectral bias. Stochastic gradient descent with momentum added to it (SGDM) will converge faster to higher frequency features and slower to low frequencies, when compared with stochastic gradient descent without momentum (SGD) [7]. Thus, the optimizer can, to some extent, even out the convergence rates between low frequency and high frequency features. Through numerical experimentation, it was demonstrated in [7] that SGDM and Adam both converge to the solution, but Adam achieves this faster.

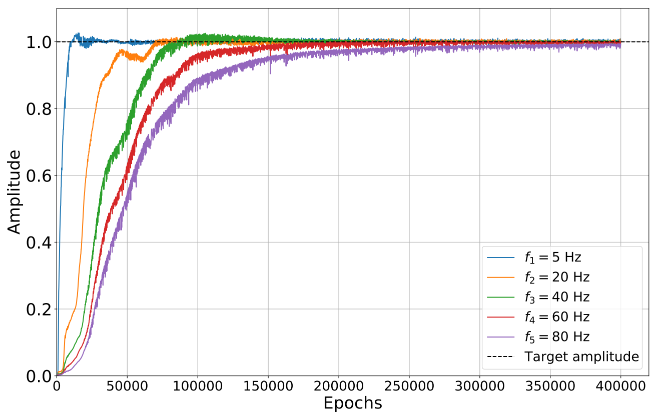
Architecture of the PINN also plays a role. It is well-known that the expressivity of a neural network increases as the width and/or depth increases. This means that the



(a) The function u



(b) Frequency spectrum of u



(c) Amplitude of different frequencies during training. Amplitude of 1 is the target for all frequencies.

Figure 5: Spectral bias of fully-connected feedforward neural network

larger the neural network is, the more complex target functions it can learn. Authors in [28] showed how increasing the depth or width of the network helps the network to converge to higher frequencies [28]. Moreover, depth of the network had a larger impact compared with width [28]. However, this approach has its drawbacks since larger networks are more challenging to optimize.

Another alternative to improve the expressivity of PINNs is to choose a more advanced architecture such as PINNsFormer [40]. PINNFormer introduces pseudo-sequences during training encouraging the network to respect the temporal structure and causality [40]. Yet, even a transformer based PINNs exhibit spectral bias [34]. However, the effects of changing the architecture from an MLP to a transformer on spectral bias require further research [34]. In Section 3.5 spectral bias is revisited through neural tangent kernel and Fourier features.

3.4 Unbalanced loss terms

PINNs loss function (4) may have varying scales between the terms [38]. Even when the inputs are scaled through non-dimensionalization into the same range the problem remains since the physics loss and data-driven loss may scale differently depending on the PDE [30, 38]. Nevertheless, scaling the inputs to a more suitable range is recommended as this accelerates convergence, especially with gradient-based optimization methods. If the scales differ greatly, the model may minimize some terms in the loss function (4) while disregarding the other terms. As the aim is to maximize the information gained from the loss function, the model must take into account all the terms. This problem corresponds to a multi-objective optimization problem. One solution is to add scalar terms λ_{ic}, λ_{bc} and λ_{r} to the loss terms in order to balance them.

$$\mathcal{L}(\theta) = \lambda_{\mathrm{ic}}\mathcal{L}_{\mathrm{ic}}(\theta) + \lambda_{\mathrm{bc}}\mathcal{L}_{\mathrm{bc}}(\theta) + \lambda_{\mathrm{r}}\mathcal{L}_{\mathrm{r}}(\theta). \tag{24}$$

The choice for parameters $\lambda_{\mathrm{ic}}, \lambda_{\mathrm{bc}}, \lambda_{\mathrm{r}}$ are important for the convergence of PINNs. In theory, given any choice of parameters would lead to the same Pareto optima as all the loss terms converge to zero [30]. In practice however, finite-sized neural networks trained through a gradient-based method converge to local optima [30]. The choice of the scalars λ_{ic}, λ_{bc} and λ_{r} has an effect on which local optima the model settles on, thereby the scalars have an effect on how accurate the model is.

Vanilla PINNs use $\lambda_{\mathrm{ic}} = \lambda_{\mathrm{bc}} = \lambda_{\mathrm{r}} = 1$ which frequently leads to optimization failures [30]. Moreover, the optimal scalar terms may vary between problems and thus require manually tuning them for each problem individually making this approach impractical [38, 30].

Most methods that dynamically assign the scalars during training focus on balancing the convergence rate of each loss term. Aiming to roughly have an equal convergence rate between all the loss terms and not that the magnitude of each loss term is equal.

One method to achieve this is to compute the norms of the gradient of the loss terms as demonstrated in [38].

$$\begin{aligned}
\hat{\lambda}_{\mathrm{ic}} &= \frac{||\nabla_\theta \mathcal{L}_{\mathrm{ic}}(\theta)||_2 + ||\nabla_\theta \mathcal{L}_{\mathrm{bc}}(\theta)||_2 + ||\nabla_\theta \mathcal{L}_{\mathrm{r}}(\theta)||_2}{||\nabla_\theta \mathcal{L}_{\mathrm{ic}}(\theta)||_2} \\
\hat{\lambda}_{\mathrm{bc}} &= \frac{||\nabla_\theta \mathcal{L}_{\mathrm{ic}}(\theta)||_2 + ||\nabla_\theta \mathcal{L}_{\mathrm{bc}}(\theta)||_2 + ||\nabla_\theta \mathcal{L}_{\mathrm{r}}(\theta)||_2}{||\nabla_\theta \mathcal{L}_{\mathrm{bc}}(\theta)||_2} \\
\hat{\lambda}_{\mathrm{r}} &= \frac{||\nabla_\theta \mathcal{L}_{\mathrm{ic}}(\theta)||_2 + ||\nabla_\theta \mathcal{L}_{\mathrm{bc}}(\theta)||_2 + ||\nabla_\theta \mathcal{L}_{\mathrm{r}}(\theta)||_2}{||\nabla_\theta \mathcal{L}_{\mathrm{r}}(\theta)||_2}
\end{aligned}$$

The global weights $\lambda_{\mathrm{ic}}, \lambda_{\mathrm{bc}}, \lambda_{\mathrm{r}}$ can be updated as a moving average defined in [38] as

$$\lambda_{\mathrm{new}} = \alpha\lambda_{\mathrm{old}} + (1-\alpha)\hat{\lambda}_{\mathrm{new}}, \tag{25}$$

where $\alpha \in (0, 1)$. One can initialize the weighting coefficients to be $\lambda_{\mathrm{ic}} = \lambda_{\mathrm{bc}} = \lambda_{\mathrm{r}} = 1$ at the start of the training and update them using the moving average (25). The weight updates can be done in any frequency and has a small impact on the computational efficiency [38].

Other method of dynamically computing the weights utilizes neural tangent kernel (NTK) matrix. In the case of physics-informed network, the NTK matrices can be calculated by formulas given in the paper [37] as follows

$$
\begin{aligned}
K_{\text{ic}} &= \left[\left\langle \frac{\partial u}{\partial \theta}\left(0, x_{i,\text{ic}}\right), \frac{\partial u}{\partial \theta}\left(0, x_{j,\text{ic}}\right) \right\rangle\right]\\
K_{\text{bc}} &= \left[\left\langle \frac{\partial u}{\partial \theta}\left(t_{i,\text{bc}}, x_{i,\text{bc}}\right), \frac{\partial u}{\partial \theta}\left(t_{j,\text{bc}}, x_{j,\text{bc}}\right) \right\rangle\right]\\
K_{\text{r}} &= \left[\left\langle \frac{\partial R_\theta}{\partial \theta}\left(t_{i,\text{r}}, x_{i,\text{r}}\right), \frac{\partial R_\theta}{\partial \theta}\left(t_{j,\text{r}}, x_{j,\text{r}}\right) \right\rangle\right].
\end{aligned}
$$

Where R_θ was defined in (3). The weighting coefficients λ_{ic}, λ_{bc} and λ_{r} are updated as

$$
\begin{aligned}
\hat{\lambda}_{\text{ic}} &= \frac{\operatorname{Tr}\left(K_{\text{ic}}\right) + \operatorname{Tr}\left(K_{\text{bc}}\right) + \operatorname{Tr}\left(K_{\text{r}}\right)}{\operatorname{Tr}\left(K_{\text{ic}}\right)}\\
\hat{\lambda}_{\text{bc}} &= \frac{\operatorname{Tr}\left(K_{\text{ic}}\right) + \operatorname{Tr}\left(K_{\text{bc}}\right) + \operatorname{Tr}\left(K_{\text{r}}\right)}{\operatorname{Tr}\left(K_{\text{bc}}\right)}\\
\hat{\lambda}_{\text{r}} &= \frac{\operatorname{Tr}\left(K_{\text{ic}}\right) + \operatorname{Tr}\left(K_{\text{bc}}\right) + \operatorname{Tr}\left(K_{\text{r}}\right)}{\operatorname{Tr}\left(K_{\text{r}}\right)}.
\end{aligned}
$$

Updating the global weighting coefficients can be done by using the moving average defined in (25). The weighting coefficients updated through NTK aim at having equal convergence rates between the weighted loss terms [38, 37].

The method of weight balancing using gradient norms is effective in practice; however, it lacks any theoretical justification [37]. It is also less stable than a method based on neural tangent kernel (NTK) matrix [38] and does not provide insight at appropriate learning rate for the model in contrast to NTK [37]. If the full NTK matrix K is computed, the formulation can be seen in [37], requirements for the learning rate can be formalized. More precisely, to ensure the stability of a gradient descent optimization the learning rate should be set to be less than $2/\lambda_{\max}$, where $\lambda_{\max}$ is the largest eigenvalue of the NTK matrix [37].

The performance in practice is quite similar between the method utilizing gradient norms and NTK but as noted NTK is more stable [38].

3.5 Spectral bias through NTK

Neural tangent kernel (NTK) analysis for PINNs is complex and challenging analytically. However, there has been research of NTK for simpler systems, such as the regression of functions with deep fully-connected neural networks. In this context, it has been shown that the ability to learn the target function can be described by the corresponding eigenspace of the neural tangent kernel [36]. Furthermore, these networks have a tendency to converge to the target function along the eigenvectors of the NTK that corresponds to larger eigenvalues [36]. For MLPs, there is an inverse relationship between the eigenvalues of the NTK and the frequency of the related eigenfunctions: as the frequency of the eigenfunction increases the corresponding eigenvalue decreases [36]. Thus, the network appears to be biased toward low-frequency solutions [36]. In fact, the authors in [36] argued that the observed spectral bias of MLPs is equivalent to NTK eigenvector bias.

The aim of the Fourier features is to modify the eigenspace of NTK [36, 38]. By modifying the eigenspace of NTK, it is possible to adjust the frequencies to which the network is biased on learning. Research on analytical analysis of NTK is limited due to computing analytical NTK for deeper networks has proven to be challenging. However, the effectiveness of Fourier features has been displayed analytically in these simpler cases in modulating the NTK eigenspace such as in [36]. In more complex scenarios, Fourier features have been shown numerically to be highly beneficial in a multitude of studies such as in [36, 17, 31, 38].

Gaussian Fourier feature embedding was originally proposed in the paper [33] as

$$\gamma(v) = \begin{bmatrix} \cos(2\pi B v) \\ \sin(2\pi B v) \end{bmatrix}, \tag{26}$$

where B is drawn from a Gaussian distribution $N(0, \sigma^2)$ [33]. The constant 2π is usually left out as is done in [38, 36]. The hyperparameter σ modifies the eigenspace of the NTK and by this the frequencies to which the network is biased on learning [36]. Optimal σ is such that the bandwidth of the NTK is equal to the target function signals [38]. In the case where there is no prior knowledge of the target function signal spectrum, the parameter σ can be searched from the domain $\sigma > 0$.

Fourier features can also enforce periodic boundary conditions strictly [38]. In the paper [38] this was done for (18) – (20) by using the following mapping

$$\gamma(v) = [\cos(\omega_x x), \sin(\omega_x x), \cos(\omega_t t), \sin(\omega_t t)], \tag{27}$$

where ω_x, ω_t are trainable parameters. With domain knowledge it is possible to engineer the Fourier feature mapping $\gamma(v)$ to a suitable form such as was done in [17] for elastodynamics.

The Fourier features maps the inputs using $\gamma(v)$ but even activation functions that are inspired by Fourier transform have been introduced. Authors in [40] introduced the Wavelet activation function as

$$W(x) = \omega_1 \sin(x) + \omega_2 \cos(x). \tag{28}$$

Wavelet closely resembles the basic Fourier feature embedding introduced in [33]. The distinction being that ω_1 and ω_2 are trainable parameters. Authors in [40] note that Wavelet does not lack expressivity as it can approximate any function. In Section 4, Wavelet is tested as activation function for (18) – (20) and compared with the traditional tanh activation function.

4 Numerical tests

This Section compares numerically Wavelet activation function, Gaussian Fourier feature embedding and NTK-based loss term balancing method. Due to the computational cost, all the methods presented in this thesis were not tested. However, the time scheduling approaches can be applied on top of all the methods tested in this Section. The main benefits of time scheduling approaches can be observed with simulations that have longer time horizons.

The equation used throughout the experiments is the convection equation (18) – (20)

$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0 \quad x \in \Omega, t \in [0, T]. \tag{29}$$

Initial and boundary conditions are defined as

$$u(x, 0) = \sin(x) \tag{30}$$
$$u(0, t) = u(2\pi, t). \tag{31}$$

This problem has an analytical solution of the form $u(x,t) = \sin(x - \beta t)$.

The convection problem was chosen for testing as it is challenging for PINNs and consequently it is used by many authors to demonstrate their methods. Furthermore, this problem for PINNs is increasingly challenging as the convection coefficient β grows in value since the target function, u, gains higher frequency features, which is evident from the analytical solution. The difficulty of the task can be scaled by scaling the coefficient β. In this thesis, a challenging $\beta = 90$ was chosen.

A data-free version of PINN is used with a loss function of the form (24) where, there are no data points of the actual solution except on initial conditions.

For each test scenario the relative L2 error and mean squared error (MSE) are computed. The relative L2 error and MSE are defined as

$$\text{Rel.L2} = \frac{||\hat{u} - u||_2}{||u||_2} \tag{32}$$
$$\text{MSE} = \frac{||\hat{u} - u||_2^2}{N}, \tag{33}$$

where $\hat{u}$ is the prediction provided by the PINN, u is the ground-truth from analytical solution and N is the number of points of prediction.

All the tests are conducted with 200000 epochs using $N_\text{r} = 2500$, $N_\text{ic} = 100$ and $N_\text{bc} = 100$. For each epoch, the points are resampled from the domain. The network employed is an MLP with 4 hidden layers and 200 neurons each with tanh activation between the layers. The network weights are learned by Adam optimizer [12] with starting learning rate of 10^{-3} and exponentially decaying learning rate every 5000 epochs. Each test scenario is computed 5 times from which the average mean squared

Fourier	NTK	Wavelet	MSE	Rel. L2 error
✓	✓	✓	0.539 ± 4.68	97.5 ± 39.9
✗	✓	✓	0.0857 ± 0.305	40.9 ± 7.23
✓	✗	✓	9.76 ± 135	377 ± 258
✓	✓	✗	8.23 ± 120	340 ± 247
✓	✗	✗	400 ± 7920	1850 ± 2390
✗	✓	✗	2110 ± 1520	6490 ± 235
✗	✗	✓	0.940 ± 12.1	117 ± 79.5
✗	✗	✗	2300 ± 3030	6760 ± 450

Table 1: Numerical tests from the 5 runs. The mean ± standard deviation are given for MSE and Rel.L2 error across the 5 runs. All the numbers should be multiplied by 10^{-4}.

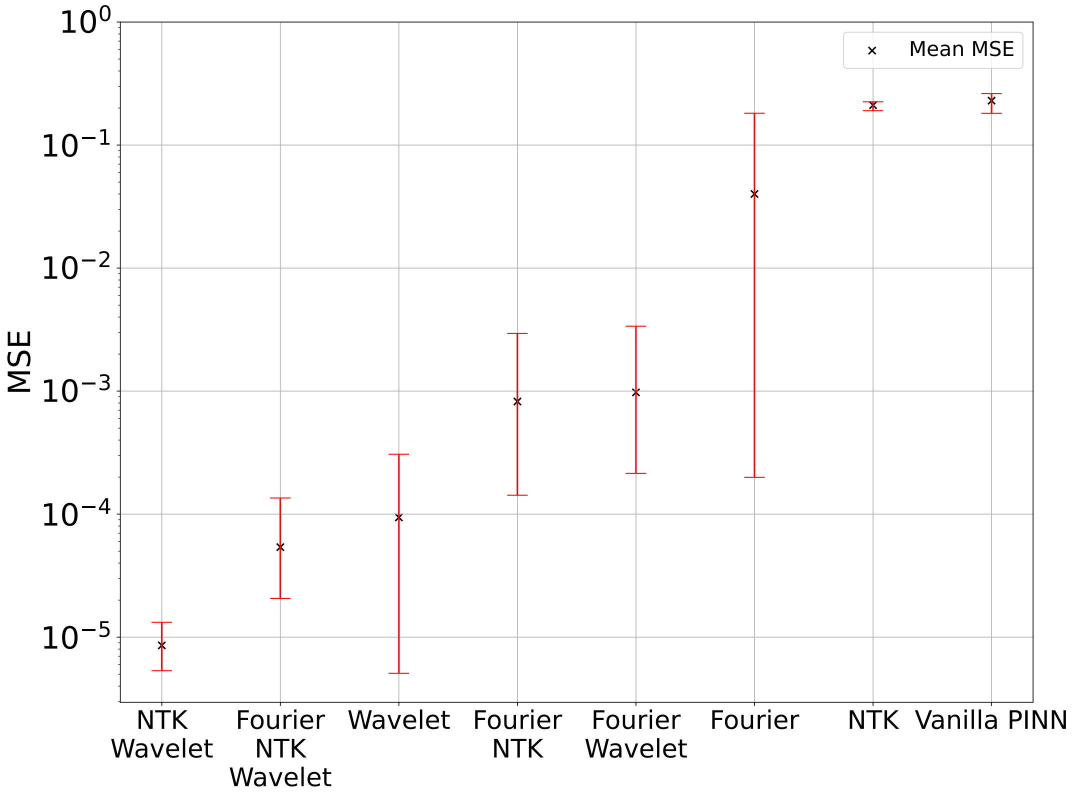


Figure 6: Error bars top is the highest MSE and bottom is lowest MSE observed from the 5 test runs.

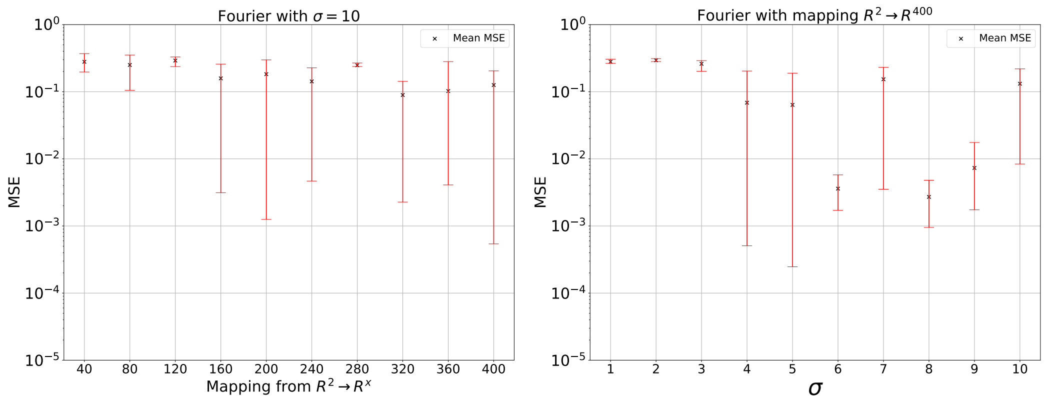
error, relative L2 error and their sample standard deviation are computed. All the numbers on Table 1 should be multiplied by 10^{-4}.

From the Table 1 it can be observed that all the method combinations had better results than the vanilla PINN, when measured by the average MSE error. Furthermore, NTK in each test case lowers the average MSE and variance. Including NTK seems to stabilize the convergence of these methods.

Wavelet activation function had the largest single effect on MSE although the variance is quite high in comparison with the mean MSE. Including NTK with wavelet brought down the average MSE by over a factor of 10^{-1}.

Gaussian Fourier feature mapping has two user parameters σ and dimension to which B maps the input to, $B : R^2 \longrightarrow R^N$. These parameters were found using hyperparameter optimization for when Fourier feature mapping was the only method enabled. Then these parameter choices were used for each combination of methods. It might be possible that the optimal parameter choices would be different for each method combination. Wavelet might alter what frequencies the network is biased to learn. From which, it stands to reason that the Gaussian Fourier feature mapping with Wavelet activation function would have optimal parameters that differ from the network which has tanh as activation function.

The sample size of 5 runs is admittedly small and might not reveal the whole picture of these methods. In addition to this, it is unlikely that the optimal parameters for Fourier feature mapping were found during the hyperparameter search. Figure 7 demonstrates the sensitivity of Gaussian Fourier feature mapping, when one parameter is fixed. There are 3 runs for each choice of parameters.



(a) Fourier feature mapping with fixed σ **(b)** Fourier feature mapping with fixed mapping

Figure 7: Gaussian Fourier feature mapping with different parameters while fixing one. Error bars top is the highest MSE and bottom is lowest MSE observed from the 3 test runs.

The absolute error between the reference solution and the best run of each method combination can be found from Appendix A.1.

5 Discussion and future work

This thesis studied challenges for PINN training such as loss landscape, causality violations, a multi-objective optimization problem, spectral bias from architecture and optimizer perspective and finally spectral bias from the viewpoint of NTK. Various methods are presented to alleviate these challenges. Finally, the methods: Gaussian Fourier feature mapping, NTK weight balancing, Wavelet activation function and their combinations were numerically benchmarked on convection equation (18) – (20). It is evident that successfully building a PINN that solves the PDE can be challenging and may require a combination of various methods in order to achieve satisfactory convergence. Moreover, some methods require user defined parameters and, in absence of domain knowledge, may require extensive hyperparameter search. Nevertheless, the methods improve the results from vanilla PINN considerably as demonstrated in the numerical tests Section 4. Of the benchmarked methods, the combination of NTK weight balancing and Wavelet activation function proved to be especially effective. On the other hand, NTK weight balancing alone did not yield any significant improvement over the vanilla PINN. In most of the best performing method combinations the Wavelet activation function was present.

This thesis contributes to the literature by bringing together various methods and challenges with PINNs. Furthermore, the numerical tests provide further insights and a more direct comparison between methods and their combined effects.

Prospective research subjects are related to improving the understanding of the neural tangent kernel of PINNs and behaviour of different PINN types. To begin with, the connection between NTK and spectral bias was studied in [36]. Examples of different PINN types include separable PINN (SPINN) [6] and Coupled-Automatic-Numerical differentiation method (CAN-PINN) [5] to tackle the practicalities of computing the PDE loss and reduce the computational cost [6, 5]. Transformer based PINN can be used to improve generalization and accuracy [40]. In cases where some regions in the solution may have more complex patterns, self-adaptive PINN (SA-PINN) using adaptive weights can be useful [21]. Physics-informed neural operators (PINO) [16] has taken an alternative approach by combining data and PDE constraints to learn the solution operator. This is achieved by utilizing Fourier neural operator (FNO) framework. For example, the convection equation would require a different PINN for each different β coefficient while a single PINO model can handle several different β coefficients. Since, the model has learned the solution operator and not directly the solution function itself.

In addition, the improved sampling of the points for the loss functions is under active research. Recent advances on this front are importance sampling [22] and direct mesh refinement approach [24], to name a few.

PINNs provide a new approach for solving PDEs with its own unique advantages compared with traditional solvers. PINNs can compliment existing solvers [29, 2], especially when existing solvers are unable to solve the problem [2]. Furthermore, in some scenarios PINNs are the best existing method [2]. Continued research on PINNs hold the promise to be able to supersede existing solvers with PINNs in some domains.

References

[1] Gregory Barber, Mulugeta A. Haile, and Tzikang Chen. *Physical Constraint Embedded Neural Networks for inference and noise regulation*. 2021. DOI: https://doi.org/10.48550/arXiv.2105.09146. arXiv: 2105.09146 [cs.LG].

[2] Shengze Cai et al. "Physics-informed neural networks (PINNs) for fluid mechanics: A review". In: *Acta Mechanica Sinica* 37.12 (2021), pp. 1727–1738. DOI: https://doi.org/10.1007/s10409-021-01148-1.

[3] Yuan Cao et al. *Towards Understanding the Spectral Bias of Deep Learning*. 2020. arXiv: 1912.01198 [cs.LG].

[4] Tianping Chen and Hong Chen. "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems". In: *IEEE Transactions on Neural Networks* 6.4 (1995), pp. 911–917. DOI: 10.1109/72.392253.

[5] Pao-Hsiung Chiu et al. "CAN-PINN: A fast physics-informed neural network based on coupled-automatic–numerical differentiation method". In: *Computer Methods in Applied Mechanics and Engineering* 395 (May 2022), p. 114909. ISSN: 0045-7825. DOI: 10.1016/j.cma.2022.114909. URL: http://dx.doi.org/10.1016/j.cma.2022.114909.

[6] Junwoo Cho et al. *Separable Physics-Informed Neural Networks*. 2023. DOI: https://doi.org/10.48550/arXiv.2306.15969. arXiv: 2306.15969 [cs.LG].

[7] Ghazal Farhani, Alexander Kazachek, and Boyu Wang. *Momentum Diminishes the Effect of Spectral Bias in Physics-Informed Neural Networks*. 2022. DOI: https://doi.org/10.48550/arXiv.2206.14862. arXiv: 2206.14862 [cs.LG].

[8] Lukas Fesser, Luca D'Amico-Wong, and Richard Qiu. *Understanding and Mitigating Extrapolation Failures in Physics-Informed Neural Networks*. 2023. DOI: https://doi.org/10.48550/arXiv.2306.09478. arXiv: 2306.09478 [cs.LG].

[9] Katsiaryna Haitsiukevich and Alexander Ilin. "Improved Training of Physics-Informed Neural Networks with Model Ensembles". eng. In: *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2023, pp. 1–8. ISBN: 1665488670. DOI: DOI:10.1109/IJCNN54540.2023.10191822.

[10] Bin Huang and Jianhui Wang. "Applications of physics-informed neural networks in power systems-a review". In: *IEEE Transactions on Power Systems* 38.1 (2022), pp. 572–588. DOI: 10.1109/TPWRS.2022.3162473.

[11] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589. DOI: https://doi.org/10.1038/s41586-021-03819-2.

[12] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. DOI: https://doi.org/10.48550/arXiv.1412.6980. arXiv: 1412.6980 [cs.LG].

[13] Aditi S. Krishnapriyan et al. *Characterizing possible failure modes in physics-informed neural networks*. 2021. DOI: https://doi.org/10.48550/arXiv.2109.01050. arXiv: 2109.01050 [cs.LG].

[14] I.E. Lagaris, A. Likas, and D.I. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations". In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 987–1000. ISSN: 1045-9227. DOI: 10.1109/72.712178. URL: http://dx.doi.org/10.1109/72.712178.

[15] Remi Lam et al. *GraphCast: Learning skillful medium-range global weather forecasting*. 2023. DOI: https://doi.org/10.48550/arXiv.2212.12794. arXiv: 2212.12794 [cs.LG].

[16] Zongyi Li et al. *Physics-Informed Neural Operator for Learning Partial Differential Equations*. 2023. DOI: https://doi.org/10.48550/arXiv.2111.03794. arXiv: 2111.03794 [cs.LG].

[17] Ruihua Liang et al. "Solving elastodynamics via physics-informed neural network frequency domain method". In: *International Journal of Mechanical Sciences* 258 (2023), p. 108575. ISSN: 0020-7403. DOI: https://doi.org/10.1016/j.ijmecsci.2023.108575. URL: https://www.sciencedirect.com/science/article/pii/S0020740323004770.

[18] Lu Lu et al. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators". In: *Nature Machine Intelligence* 3.3 (Mar. 2021), pp. 218–229. ISSN: 2522-5839. DOI: https://doi.org/10.48550/arXiv.1910.03193. URL: http://dx.doi.org/10.1038/s42256-021-00302-5.

[19] Stefano Markidis. "The old and the new: Can physics-informed deep-learning replace traditional linear solvers?" In: *Frontiers in big Data* 4 (2021), p. 669097. DOI: DOI:10.3389/fdata.2021.669097.

[20] Revanth Mattey and Susanta Ghosh. "A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations". In: *Computer Methods in Applied Mechanics and Engineering* 390 (2022), p. 114474. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2021.114474. URL: https://www.sciencedirect.com/science/article/pii/S0045782521006939.

[21] Levi McClenny and Ulisses Braga-Neto. *Self-Adaptive Physics-Informed Neural Networks using a Soft Attention Mechanism*. 2022. DOI: https://doi.org/10.48550/arXiv.2009.04544. arXiv: 2009.04544 [cs.LG].

[22] Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. "Efficient training of physics-informed neural networks via importance sampling". In: *CoRR* abs/2104.12325 (2021). arXiv: 2104.12325. URL: https://arxiv.org/abs/2104.12325.

[23] Tung Nguyen et al. *ClimaX: A foundation model for weather and climate*. 2023. DOI: https://doi.org/10.48550/arXiv.2301.10343. arXiv: 2301.10343 [cs.LG].

[24] Shikhar Nilabh and Fidel Grandia. *Efficient Training of Physics-Informed Neural Networks with Direct Grid Refinement Algorithm*. 2023. DOI: https://doi.org/10.48550/arXiv.2306.08293. arXiv: 2306.08293 [cs.LG].

[25] Jaideep Pathak et al. *FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators*. 2022. DOI: https://doi.org/10.48550/arXiv.2202.11214. arXiv: 2202.11214 [physics.ao-ph].

[26] Michael Penwarden et al. "Multifidelity modeling for Physics-Informed Neural Networks (PINNs)". In: *Journal of Computational Physics* 451 (2022), p. 110844. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2021.110844. URL: https://www.sciencedirect.com/science/article/pii/S0021999121007397.

[27] GP Purja Pun et al. "Physically informed artificial neural networks for atomistic modeling of materials". In: *Nature communications* 10.1 (2019), p. 2339. DOI: https://doi.org/10.1038/s41467-019-10343-5.

[28] Nasim Rahaman et al. *On the Spectral Bias of Neural Networks*. 2019. arXiv: 1806.08734 [stat.ML].

[29] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. 2017. DOI: https://doi.org/10.48550/arXiv.1711.10561. arXiv: 1711.10561 [cs.AI].

[30] Franz M. Rohrhofer et al. "Data vs. Physics: The Apparent Pareto Front of Physics-Informed Neural Networks". In: *IEEE Access* 11 (2023), pp. 86252–86261. DOI: 10.1109/ACCESS.2023.3302892.

[31] Chao Song and Yanghua Wang. "Simulating seismic multifrequency wavefields with the Fourier feature physics-informed neural network". In: *Geophysical Journal International* 232.3 (Oct. 2022), pp. 1503–1514. ISSN: 0956-540X. DOI: 10.1093/gji/ggac399. eprint: https://academic.oup.com/gji/article-pdf/232/3/1503/47715618/ggac399.pdf. URL: https://doi.org/10.1093/gji/ggac399.

[32] Luning Sun et al. "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data". In: *Computer Methods in Applied Mechanics and Engineering* 361 (2020), p. 112732. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2019.112732. URL: https://www.sciencedirect.com/science/article/pii/S004578251930622X.

[33] Matthew Tancik et al. *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*. 2020. DOI: https://doi.org/10.48550/arXiv.2006.10739. arXiv: 2006.10739 [cs.CV].

[34] Bhavya Vasudeva et al. *Simplicity Bias of Transformers to Learn Low Sensitivity Functions*. 2024. DOI: https://doi.org/10.48550/arXiv.2403.06925. arXiv: 2403.06925 [cs.LG].

[35] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. "Respecting causality for training physics-informed neural networks". eng. In: *Computer methods in applied mechanics and engineering* 421.C (2024), pp. 116813–. ISSN: 0045-7825. DOI: DOI:10.1016/j.cma.2024.116813.

[36] Sifan Wang, Hanwen Wang, and Paris Perdikaris. "On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks". In: *Computer Methods in Applied Mechanics and Engineering* 384 (2021), p. 113938. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2021.113938. URL: https://www.sciencedirect.com/science/article/pii/S0045782521002759.

[37] Sifan Wang, Xinling Yu, and Paris Perdikaris. "When and why PINNs fail to train: A neural tangent kernel perspective". In: *Journal of Computational Physics* 449 (2022), p. 110768. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2021.110768. URL: https://www.sciencedirect.com/science/article/pii/S002199912100663X.

[38] Sifan Wang et al. "An expert's guide to training physics-informed neural networks". In: *arXiv preprint arXiv:2308.08468* (2023). DOI: https://doi.org/10.48550/arXiv.2308.08468.

[39] Enrui Zhang et al. "Analyses of internal structures and defects in materials using physics-informed neural networks". In: *Science advances* 8.7 (2022), eabk0644.

[40] Zhiyuan Zhao, Xueying Ding, and B. Aditya Prakash. *PINNsFormer: A Transformer-Based Framework For Physics-Informed Neural Networks*. 2023. arXiv: 2307.11833 [cs.CE].

A Appendix

A.1 More results

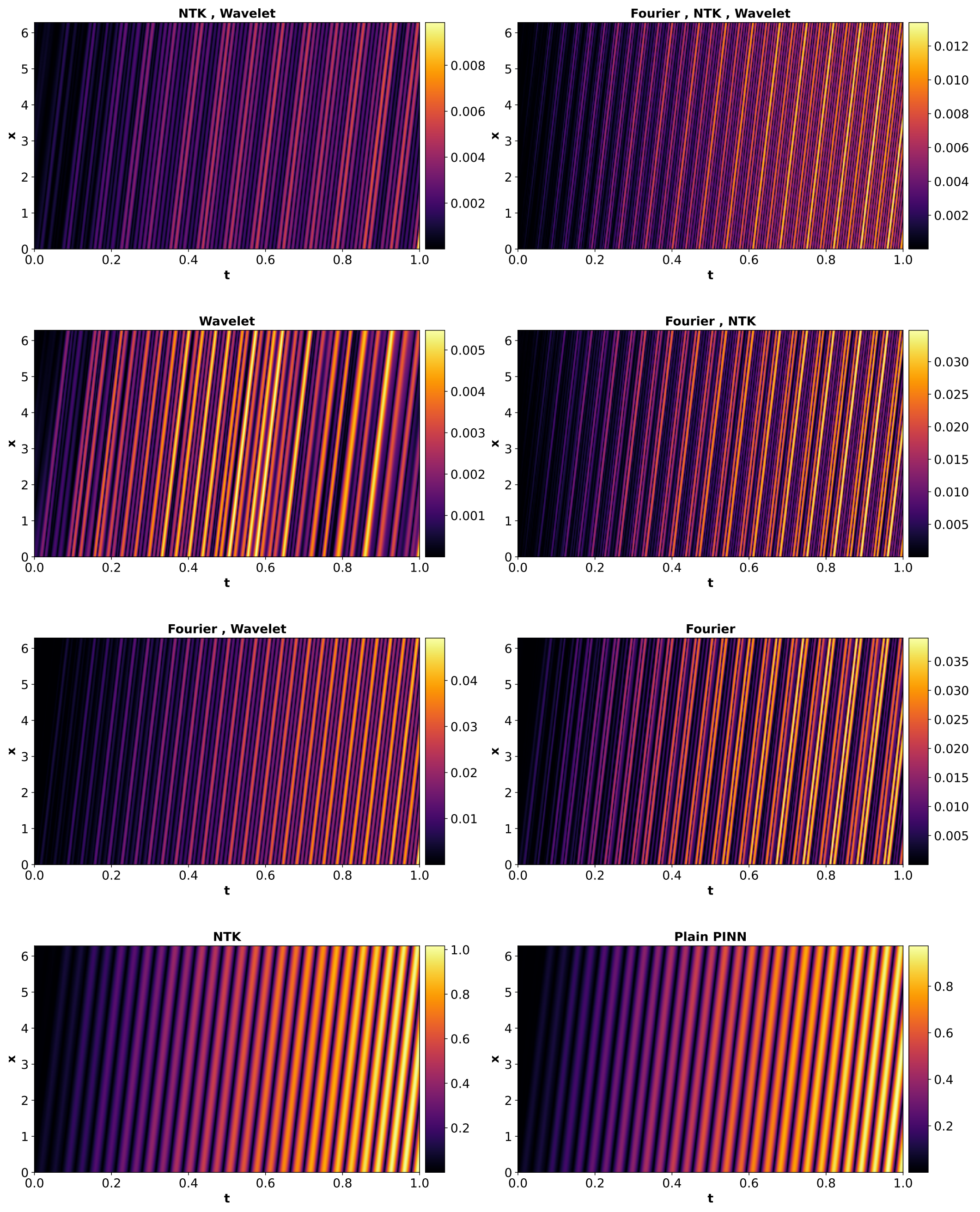


Figure A1: Absolute error between exact solution and the best estimated solution of the corresponding run.