Master's Thesis Report

Physics-Informed Neural Networks for Solving Ordinary Differential Equations in Stiffness Regimes: A Multi-Head Architecture and Transfer Learning Approach

By

Emilien Seiler

Master's student at EPFL

Visiting student in Harvard

Supervised by

Prof. Pavlos Protopapas, Harvard

Prof. Hesthaven S Jan, EPFL

EPFL

Department of Mathematics

Computational Science and Engineering

March 15, 2024

Physics-Informed Neural Networks for Solving Stiffness Regimes in Ordinary Differential Equations using Multi-Head Architecture and Transfer Learning

Abstract

This thesis explores the use of Physics Informed Neural Networks to solve stiff linear and non-linear ordinary differential equations. Physics Informed Neural Networks integrate governing equations into neural network structures via automatic differentiation, and stiffness introduces difficulty in encoding the solution during training. Behaviors such as rapid transient phases can be particularly challenging to encode. We extend previous methodologies to tackle stiffness with a novel transfer-learning-based approach. The approach consists of training a multi-head architecture in a non-stiff regime and transferring it to a stiff regime without the need for retraining the model. The present approach is compared to both vanilla Physics-Informed Neural Networks and numerical methods, such as RK45 and Radau methods. Experiments were conducted on two linear examples and extended to one non-linear example using perturbation theory. Our analysis indicates that transfer learning from a less stiff regime can be used to compute a solution in a stiffer regime, thereby reducing the complications associated with training in stiff systems. Transfer learning has been achieved on the Duffing equation, from a stiffness ratio less than 100 to a regime where the stiffness ratio is greater than 5000, outperforming the vanilla PINNs model. The average absolute error has been maintained at less than 10^{-3}. The approach provides competitive computational efficiency, especially when modifying initial conditions or force functions within a stiff domain. It is 70 and 7 times faster than using the Radau method for investigating linear and non-linear equations, respectively. However, challenges persist in transferring to very stiff regimes too far from the training one and handling all forms of non-linear equations.

Acknowledgments

First, I would like to thank my thesis advisor, Prof. Pavlos Protopapas, for giving me the opportunity to do my master's thesis in his laboratory at Harvard University. Thank you for your advice that guided me throughout this six-month project. I would also like to thank Prof. Jan Hesthaven for agreeing to supervise my thesis from Switzerland and for his trust and guidance. I would especially like to thank Wanzhou for his friendly presence in the lab and his sharp expertise. Thanks to my friends Felix and Titouan for sharing this Boston experience with me, thanks for the chess and ping-pong games. And, of course, thanks to my parents for allowing me to have such rich experiences. Thank you to my grandparents and brothers. Finally, a very special thank to Mahlia for accompanying and supporting me in all of our adventures.

Table of Contents

Introduction

Deep learning has emerged as a powerful paradigm in machine learning, characterized by the use of neural networks with multiple layers to learn intricate representations of data. In recent years, deep learning has revolutionized various fields: image classification [1], object detection [2], or natural language processing [3].

In 1998, Lagaris et al. introduced neural networks for solving differential equations and paved the way for today's Physics Informed Neural Networks (PINNs) [4]. These networks embed governing equations into the neural network architecture by minimizing the residual loss function using automatic differentiation. These models have demonstrated impressive outcomes in addressing physical problems, such as the Navier-Stokes equation [5], the Schrödinger equation [6], and problems in kinetic chemistry [7].

In the scope of differential equations, stiff problems present significant difficulties due to their diverse timescales. The challenge is exacerbated by the fact that the stiffness of an equation lacks a precise mathematical definition, adding complexity to the evaluation process. Hairer et al. offered a comprehensive exploration of numerical methods for tackling stiff problems, shedding light on the intricate nature of the issue [8]. Stiff equations manifest in various domains such as chemical kinetics [9] and electrical engineering [10]. Dahlquist noted in 1985 "Around 1960, things became completely different and everyone became aware that the world was full of stiff problems" [11].

This thesis focuses on addressing stiff Ordinary Differential Equation (ODE) resolutions using PINNs. Despite the successful application of PINNs in various studies, Wang et al. investigated a fundamental mode of failure of PINNs that is related to stiffness [12]. The investigation revealed that stiffness could induce gradient pathologies and pose challenges in optimization, leading to the inefficacy of stochastic gradient descent-based optimization. Optimizing stiffness problems proves challenging within the conventional PINNs architecture. In 2021, Ji et al. utilized quasi-steady-state assumptions to mitigate stiffness in ODE systems [13]. Baty et al. introduced in 2023 straightforward approaches to enhance training in stiff regime [14]. However, these advancements always require retraining the model if there is a slight change in the stiffness regime or parameters of the equation.

On the other hand, Desai et al proposed in 2022 a transfer learning framework for PINNs, enabling one-shot inference for linear systems of ODEs [15]. This implies that highly accurate solutions for

numerous unknown differential equations can be obtained instantaneously without retraining the entire network. A non-linear extension of this approach using perturbation theory was presented by Lei et al. in 2023 [16].

How can these methodologies be adapted to effectively tackle stiffness, aiming to rival numerical methods in both computational efficiency and accuracy for some linear and non-linear stiff ODEs? The objective is to evaluate these transfer learning methods on stiff problems and compare them with vanilla PINNs and numerical methods.

Firstly, this paper will introduce the mathematical concepts that will be used, including ODEs, the notion of stiffness, and PINNs. Secondly, it will develop the training procedure, which includes defining the utilized loss function, selecting the model architecture and setting up the training. Then, we will present the methods and results for linear and non-linear ODEs. Finally, we will discuss the study's results and their implications, as well as its limitations.

Mathematical Overview

Ordinary Differential Equation

An ODE is an equation containing an unknown function of one real or complex variable t, its derivatives, and some given functions of t. The unknown function is generally represented by a variable y, which, therefore, depends on t.

$$F(t, y, y', \ldots, y^{(n)}) = 0 \tag{1}$$

where y is a function of t, $y^{'} = \frac{dy}{dt}$ is the first derivative with respect to t, and $y^{(n)} = \frac{d^n y}{dt^n}$ is the n^{th} derivative with respect to t.
Various types of ODEs are characterized by the following properties:

- The order of a differential equation is determined by the highest degree of derivative present in the equation.
- An ODE is termed as a single ODE when there is only one unknown, and it becomes a system of ODEs when multiple unknowns are involved. In the case of a system, the equations are interrelated or coupled, necessitating simultaneous solution computation.
- Constant coefficients, which remain unaffected by the independent variable, represent fixed values in front of the unknown function and its derivatives. On the other hand, non-constant coefficients allow these terms to be functions of the independent variable t.
- A homogeneous ODE exclusively revolves around the function y and its derivatives $y^{(n)}$ in all terms. Conversely, a non-homogeneous differential equation introduces an additional term on the right side, denoted as the forcing function f, which may be a function of t.
- Linearity in a differential equation arises when the highest order of the dependent variable or its derivatives are equal to one. In contrast, non-linear differential equations encompass all other cases.

This thesis focuses on solving both linear and non-linear systems of first-order homogeneous and non-homogeneous ODEs featuring constant coefficients.

Vectorized Equation Representation

We introduce a vectorized equation representation that can generalize to any number and form of linear ODE. Subsequently, we employ this representation and notation for all linear ODEs presented in the following chapters. Consider the subsequent system of coupled first-order ODEs as an example for vectorization:

$$\begin{cases} \frac{dy_1}{dt} + a_1 y_1 + a_2 y_2 = f_1(t) \\ \frac{dy_2}{dt} + a_3 y_1 + a_4 y_2 = f_2(t) \end{cases} \tag{2}$$

with t the independent variable, $y_1(t)$ and $y_2(t)$ two unknown functions and $f_1(t)$ and $f_2(t)$ the forcing functions. We aim to write this and any other system of linear first-order ODEs in the following form:

$$\dot{y} + Ay = f \tag{3}$$

with y a vector, $\dot{y}$ a vector of the derivative of y with respect to the independent variable t, A represents a matrix of coefficients for the non derivative terms and f is a vector of representing the forcing function terms. Given this formulation we can rewrite the example as:

$$\begin{pmatrix} \frac{dy_1}{dt} \\ \frac{dy_2}{dt} \end{pmatrix} + \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} f_1(t) \\ f_2(t) \end{pmatrix} \tag{4}$$

Transformation of a Second-Order Simple ODE into a System of First-Order ODEs

A second-order simple ODE can be transformed into a system of two first-order ODEs following this formula.

$$f(t, y, y', y'') = 0 \in \mathbb{R} \text{ after } \begin{cases} y_1 = y \\ y_2 = y' \end{cases} \Rightarrow F(t, y_1, y_2, y_1', y_2') = \begin{cases} y_1' = y_2 \\ f(t, y_1, y_2, y_2') = 0 \end{cases} \in \mathbb{R}^2 \tag{5}$$

Stiffness

This study focuses on the resolution of stiff ODEs. In this section, we delve into the phenomenon of stiffness within ODEs and explore its impact on equation resolution through the application of

numerical analysis methods.

The term "phenomenon" seems more appropriate than "property", as the latter suggests that stiffness can be precisely defined mathematically, which, unfortunately, is proving difficult. Although specialists understand the intuitive meaning, its precise mathematical definition is the subject of ongoing controversy. The most pragmatical opinion is also historically the first one; *Stiff equations are equations where certain implicit methods, in particular BDF, perform better, usually tremendously better, than explicit ones* [17]. or more recently; *An ordinary differential equation problem is stiff if the solution being sought is varying slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results* [18].

We will discuss simple statements that attempt to describe stiffness and highlight what are probably the best properties for a "definition" of stiffness :

- **Eigenvalue Distribution**

  Stiffness occurs when eigenvalues of the Jacobian matrices of the system cover a wide range of magnitudes [8]. To quantify this notion, we introduce the notion of Stiffness Ratio, denoted as SR [19]; with J being the Jacobian of the system. SR is defined as follows:

$$SR = \frac{\left|Re(\bar{\lambda})\right|}{|Re(\underline{\lambda})|} \text{ with } |Re(\underline{\lambda})| \leq |Re(\lambda_i)| \leq \left|Re(\bar{\lambda})\right| \tag{6}$$

  $\lambda_t \in \mathbb{C}$ with i=0,1,...,n are the eigenvalues of J

  The higher is the Stiffness Ratio, the stiffer is the equation. When $SR < 20$ the problem is not stiff, up to $SR \simeq 1000$ the problem is classified as stiff, and when $SR \geq 100000$, the problem is very stiff [20].

- **Disparate Time Scales and Transient Phase**

  The stiffness behavior occurs when, on a given time scale, some elements change much more rapidly than others [21]. This conduct may lead to a phenomenon characteristic of stiffness which is a rapid transient phase.

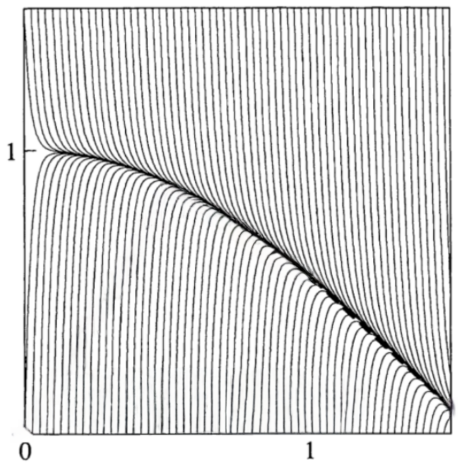

Figure 1: Example from Hairer et al. [8], solution curve of equation $y^{'} = -50(y - cos(x))$

The phenomenon is illustrated in Figure 1. Each curve corresponds to the solution of equation $y^{'} = -50(y - cos(x))$ with a different initial condition. We can see that all solutions reach a smooth solution in the vicinity of $y \simeq cos(x)$ after a more or less rapid transient phase. In the vicinity of these transient phase, the rate of change of the solution is significantly higher in comparison to other time scales. Such transients are a typical behaviour associated with stiffness, but as other properties it is neither sufficient or necessary. For instance, the disparity in time scales can arise from really small oscillations, where the amplitude of the oscillations is notably lower compared to the remainder of the solution. These phenomenons suggests that there is a notable stiff behaviour in this time scale.

- **Restricted Stability Regions for Explicit Methods**

  Stiff equations often have very fast transients or oscillations that occur on a much smaller time scale compared to the overall dynamics of the system. Because of these behaviours, explicit methods typically require small time steps to ensure stability and accuracy. These methods are subject to stability conditions, which limits the allowable time step size based on the spatial and temporal discretization. This increases the computational cost significantly, making explicit methods like Forward Euler or Runge Kutta 4.5 less efficient [22]. On the other hand, implicit methods like Radau are unconditionally stable and designed to handle stiff equations more effectively [23]. This criterion can be regarded as a consistent factor present in stiff problems. Indeed, explicit methods consistently face challenges when confronted to stiff problems.

An equation per se is not stiff. A stiff regime may be created by a particular initial value, an equation's parameter or an integration interval for example. While the eigenvalues of the Jacobian certainly contribute to the stiffness of the equation, they often cannot be the only criterion for characterizing stiffness [24]. Indeed, while the Stiffness Ratio serves as an indicator to measure the stiffness within an equation, it cannot be extrapolated for an inter-equation comparison because it only catches the impact of the eigenvalues. Quantities such as the dimension of the system, the initial conditions, the forcing function, the smoothness of the solution or the integration interval must also be considered.

Physics Informed Neural Network

Artificial Neural Networks constitute a class of machine learning models inspired by the structural intricacies of the human brain, commonly employed in tasks involving regression or classification. The fundamental component of these models is the node, whose operational principles are explain in the equation below.

$$u = \sigma(\mathbf{w}\mathbf{x}^T + b) \text{ with } \mathbf{w}, \mathbf{t} \in \mathbb{R}^n \text{ and } b, u \in \mathbb{R} \tag{7}$$

A node receives input signals $\mathbf{t}$ from other nodes or the input layer of the network. These inputs are multiplied by weights $\mathbf{w}$, adjusting the impact of each input on the node's output u. The weighted inputs are summed up, and an additional bias term b is added. Learning in an Artificial Neural Network involves adjusting these weights to enhance the network's performance. The resultant value undergoes processing through an activation function σ. This function introduces non-linearity, enabling the network to capture complex relationships. The output of the node, post-activation, is transmitted to the nodes in the subsequent layer of the network.
The effectiveness of neural networks relies on the stacking of multiple nodes with different weights within hidden layers and subsequently stacking several hidden layers in a row, fostering inter-connectivity among nodes.

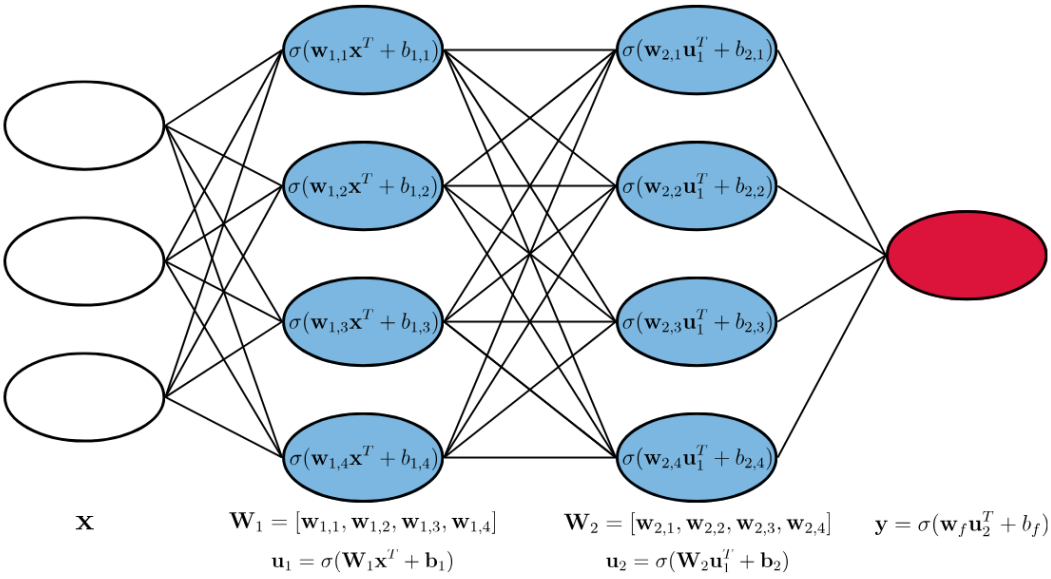

Figure 2: Artificial Neural Network with 2 fully connected hidden layers (in blue) each with 4 nodes. The white and red circles represent the input and output layers of the network

The Figure 2 describes the process of a network with two layers, the output is transformed by each hidden layer in succession. A layer can be seen has a matrix multiplication with W_i being the weight of the i^{th} layer followed by the activation function.

$$\mathbf{u}_i = \sigma(\mathbf{W}_i\mathbf{u}_{i-1} + \mathbf{b}_i) \text{ with } \mathbf{u}_0 = x,\ \mathbf{W}_i \in \mathbb{R}^{n\times n} \text{ and } \mathbf{u}_i,\ \mathbf{x},\ \mathbf{b}_i \in \mathbb{R}^n \tag{8}$$

Neural networks are recognized as universal approximators, theoretically capable of approximating any function, regardless of the number of hidden layer and number of nodes [25].

This concept can be extended to solutions of differential equations. For this task, these models are termed PINNs, a concept introduced in the literature in 1998 by Lagaris et al. [4]. In the context of solving ODEs, neural networks aim to minimize the residual of the equations during training to converge to the optimal representation of the underlying function.

Training Procedure

Loss Function

For both linear and non-linear ODEs, we make use of custom loss functions in the training procedure. At a high level, the goal is to minimize the residuals of the differential equations and satisfy the initial conditions. For the following section, we will denote the PINNs approximation of the true solution y as u.

Linear ODEs

For linear ODEs, we will use the vectorized representation present in Equation 3. The loss function is composed of two components: the minimization of the residuals of the equation and the minimization of the initial conditions. For a system of n linear ODEs, it is formulated as follows:

$$L = L_R + L_{IC} = \frac{1}{N} \sum_{t \in [t_1, ..., t_N]} [(\dot{u}_t + Au_t - f)^2] + (u_0 - y_0)^2 \tag{9}$$

with N the number of training data, $[t_1, ..., t_N]$ the training data, u_t and $\dot{u}_t$ the PINNs approximation and its derivative evaluated in t, u_0 the network evaluated in $t = 0$ and y_0 the initial condition. u_t, $\dot{u}_t$, f, u_0, $y_0 \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$. The first term L_R is the weighted some of the residual of the differential equation in each training data. The second term L_{IC} is simply the square initial condition error. The loss is convex because the convexity is invariant under affine maps. It is ultimately this convex function which we want to minimize during training.

Non-Linear ODEs

We cannot use the vectorized representation because of the non-linearity. For a system of n non-linear ODEs, the loss function is formulated as follows:

$$L = L_R = \frac{1}{N} \sum_{t \in [t_1, ..., t_N]} F(t, u_t, \dot{u}_t)^2 \tag{10}$$

$F \in \mathbb{R}^n$ is the non-linear ODE representation of Equation 1.

Initial conditions are not in the loss. For non-linear ODE we will make use of a reparametrization

approach that involves neural form described by Lagaris et al. [26]. In other words, we will rewrite the network output in a way such that it satisfies the initial value condition by construction, hence simplifying the loss function. We want to construct a neural network based model, such that initial conditions are exactly satisfied unconditionally and without sacrificing the approximation quality of the architecture. We introduce the operator $P^0_{0|t}\phi(t)$:

$$P^0_{0|t}\phi(t) = \phi(0)(1 + h(t) - h(0)) \text{ with } h(t) \text{ is any bounded function for } t \geq 0 \tag{11}$$

For example if $h(t) = e^{-t} \Rightarrow P^0_{0|t}\phi(t) = \phi(0)e^{-t}$
Using this operator, we can construct a neural form that would satisfy initial conditions. We denote u_r the reparameterized neural network based model:

$$\begin{aligned} u_r(t) &= P^0_{0|t}y_t + (1 - P^0_{0|t})u_t \\ u_r(t) &= u_t + (y_0 - u_0)(1 + h(t) - h(0)) \\ \text{with } h(t) = e^{-t} &\Rightarrow u_r(t) = u_t + (y_0 - u_0)e^{-t} \end{aligned} \tag{12}$$

This model u_r satisfies unconditionally the initial condition $u_r(0) = u_0 + (y_0 - u_0)e^0 = y_0$
The benefits of this method are that the solution is potentially easier to train, as initial conditions are satisfied by construction, and the network only needs to learn the behaviour in the interior. It may converge to the correct solution faster. But this solution is more complex since it requires modifications to the network architecture. Note that this approach is not used for linear ODEs because we would need a simple architecture to perform weight derivation that involves linear algebra.

Model Architecture

In any deep learning undertaking, it is crucial to consider the architecture of the model and select one which suits the given problem. In this context, our aim is for the neural network to approximate the solution of the ODE. For a given family of equations, the solution varies according to the parameters of the equation, or the initial conditions. Our aim is for the model to learn a generalized version of the solution that can adapt to different parameters. Instead of training our model exclusively on a specific equation, we opt for an architecture that allows the learning of a set of equations simultaneously, all belonging to the same family but with distinct parameters and initial

conditions.

Single-Head Approach

Before delving into our final architecture, we consider its simplified version, namely the single-head approach.

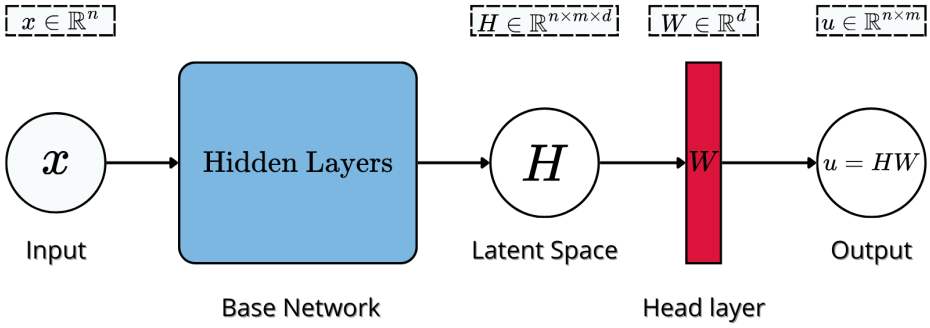

Figure 3: PINNs Single-Head Approach Diagram

As illustrated in Figure 3, the network takes x as input. This input goes through multiple hidden layers of varying sizes, and after each hidden layer, the activation function introduces non-linearity into the model. The output of the base network, denoted as H, has dimensions $n \times m \times d$, where m is the number of ODEs in the system, and d is the dimension of the last hidden layer. Each $m \times d$ matrix serves as the latent space representation of the ODE at each evaluation point.
In the case of a single-head approach, the model is trained on a single set of equation parameters and initial conditions. The head layer is a simple linear layer, which can be represented as a matrix multiplication. The final output of the network is expressed as $u = HW$, where W corresponds to the weights of this last layer. In this configuration, H only represents a unique set of equation's parameters.

Multi-Head Approach

With the multi-head approach, we want to construct H in such a way that it embodies a generalization of the equation. The multiple heads of the method are characterized by diverse parameters sets such as equation coefficients, forcing functions, or initial conditions.

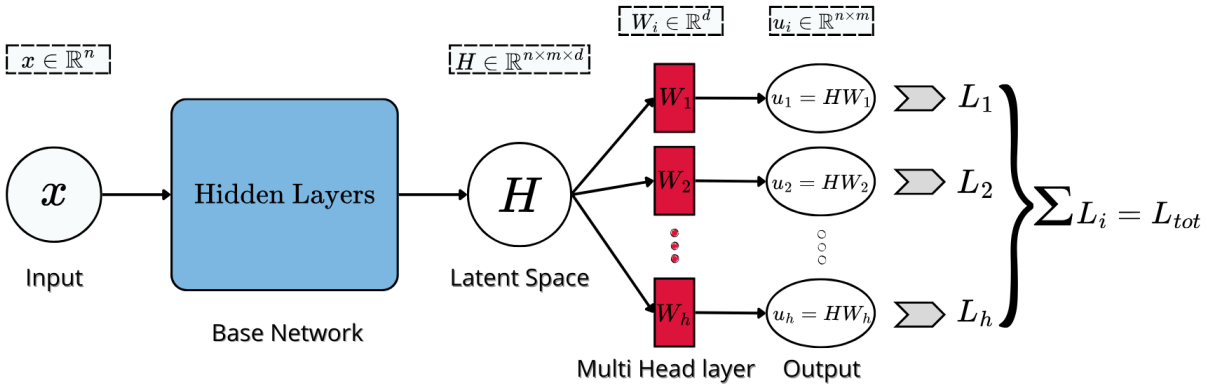

Figure 4: PINNs Multi-Head Approach Diagram

As shown on Figure 4, the first part of the network is similar to the single head approach. The input x given to the network will flow through all hidden layers. In this architectural configuration, there are now multiple output layers, each characterizing a distinct head (h heads as illustrated in the Figure). Each layer possesses its unique set of weights W_i and generates its individual output $u_i = HW_i$. For each of these h heads, we calculate a loss function L_h and aggregate them to determine the overall loss, L_{tot}.

Each head uses the same H to calculate u_i. Thus, variations in the training conditions (initial conditions and equation parameters) through the different heads, will be encoded in a single $m \times d$ matrix for each evaluation point. This results into a tensor H $(n \times m \times d)$ which captures the general equation form in all evaluation points. During the training, it is observed that the network learns to shift the W_i value to accommodate the different conditions as it converges on a fix H. As the network gets sufficiently trained, it can be confirmed that at much later epochs H stabilizes, while the W_i values are the ones fluctuating.

Ultimately, we want to save this fixed value of H after training and compute W for an untrained head such that we can construct the solution u by multiplying the learned H^{fix} with a derived set of weights W.

Training Setup

Layer Architecture

Tables 1a and 1b show the number of layers and nodes in the network architecture.

Network Linear ODE Architecture	
Layer	*(Input, Output)*
Input Layer 1	(1, 128)
Hidden Layer 2	(128, 128)
Hidden Layer 3	(128, 132)
Output Layer	(132, m)

(a) Network Architecture for Linear ODEs, m is the dimension of the ODEs system

Network Non-Linear ODE Architecture	
Layer	*(Input, Output)*
Hidden Layer 1	(1, 128)
Hidden Layer 2	(128, 128)
Hidden Layer 3	(128, 256)
Hidden Layer 4	(256, 512)
Output Layer	(512, m)

(b) Network Architecture for Non-Linear ODEs, m is the dimension of the ODEs system

Table 1: Comparison of Linear and Non-Linear ODE Networks Architectures

For the linear ODE scenario, we adopt a 3-layered neural network architecture. The first two hidden layers consist of 128 nodes each, while the third hidden layer comprises 132 nodes. The input layer, tailored for ODEs, consistently features one node, representing the single dependent variable. Meanwhile, the output layer consists of m nodes, aligning with the m equations characterizing the ODE system. The exact number of nodes chosen were based on empirical experiments. The crucial role of the output layer lies in determining the dimensionality of the latent space representation, denoted as H. To ensure an accurate and generalized representation, a larger number of nodes in this layer is preferred. However, a balance must be struck, as an excessive number of nodes could lead to increased computational time in subsequent processes.

The architecture for non-linear ODEs demands increased complexity. This is reflected in a higher number of layers and nodes compared to the linear counterpart. The augmented model with additional layers and nodes accommodates the heightened intricacies of non-linear ODEs, enabling a more robust approximation of their solutions. This expanded architecture is essential for capturing the nuanced relationships within the non-linear system.

Activation Function

After each hidden layer, the SiLU activation function is applied to introduce non-linearity to the model. The SiLU activation function is mathematically represented as:

$$SiLU(t) = t\sigma(t) \text{ with } \sigma \text{ the sigmoid function} \tag{13}$$

The SiLU function is smooth and continuous, which aids in optimization during the training process [27].

Data Sampling

When training PINNs, we have no data in the form of supervised learning pairs. Thus, we create our own data by sampling and evaluating the loss function. For all trainings, we use an equally-spaced noisy sampling technique. Points are selected at equal intervals within a given range and random noise is added to the position of each point. We ensure that every region of the input space is sampled. The noise prevent overfitting to the grid structure and promote more flexible learning, especially in regions with complex patterns.

Optimization

We opt for the ADAM optimizer during the training of our network. To help the training, a decay factor is used to regularize the optimization process. It scales down the gradients during training, preventing overfitting, avoiding numerical instability, and improving convergence. The hardware that was used to train all our networks is an NVIDIA GeForce GTX 1050 GPU. The relevant specification details of the GPU are included in Table 2 below.

GPU Specification	
Hardware Metric	*Numeric Value*
Memory Size	7GB
Cores	640 cores

Table 2: NVIDIA GeForce GTX 1050 GPU

Metric

To assess the effectiveness of our architecture, we employ two distincts metrics involving the PINNs solution u and the numerical solution y. The first numerical method under consideration is RK45, the explicit Runge-Kutta method with a step size control algorithm, which is not explicitly designed for stiff problems [22]. The second numerical approach is the Radau method, an implicit Runge-Kutta method of order 5 with adaptative step size. This last method has shown really good performance on stiff problems [23].

The first metric, Mean Absolute Error, is defined as follows:

$$MAE = \frac{1}{N} \sum_{t \in [t_1, \ldots, t_N]} \Big[|u(t) - y(t)| \Big]$$

It provides a measure of the overall performance of the method across all time scales. The absolute value ($|.|$) ensures a positive difference between the solutions.

The second metric, Maximum Absolute Error, is expressed as:

$$MaxAE = \max_{[t_1,\ldots,t_N]} |u(t) - y(t)|$$

It allows us to identify the largest error in the network. This is particularly crucial in the context of stiff equation in the vicinity of transient phase and initial conditions, providing insights into the maximum deviation between the predicted and numerical solutions.

Source code:

The source code of the thesis project is available at `https://github.com/eseiler18/Thesis`

Linear ODE

In this section, our focus is directed towards ODE of the form $\dot{y} + Ay = f$. We aim to address the resolution of these ODEs within the realm of stiff domains. In each illustrative example, we designate α as the stiffness parameter of the ODEs, wherein a higher α implies increased stiffness. Initially, we present the methods for approximating the solution through PINNs. Subsequently, we present our findings derived from the application of these methods. The comparison of these methods will be conducted from the perspectives of both accuracy and computational time, juxtaposed against two numerical approaches: RK45 and Radau.

Methods

In the context of linear ODE, two distinct methodologies will be employed to address the resolution of the equation within a stiff domain. The initial approach involves a direct attempt to learn the solution within the stiff regime. The second method relies on a form of transfer learning, wherein the model undergoes training in a non-stiff regime of the equation. Subsequently, this pre-trained model is utilized to ascertain a closed-form solution within a stiff domain.

Learning Stiffness: *Vanilla PINNs*

The initial approach involves to train a PINNs directly within a stiff domain. In this method, we employ the Single-Head architecture explained in Figure 3 and try to learn the solution within a specific stiff regime, utilizing a fixed α and a singular set of initial conditions.
It is noteworthy that a drawback of this method becomes evident: with each change in stiffness domain or initial conditions, the model needs to relearn the solution. Subsequently, we will delve into the intricacies and challenges associated with the training of PINNs in stiff domains, revealing complexities and time-consuming aspects in the subsequent sections.

Transfer Learning to Stiff Regime: *One Shot Transfer Learning*

The proposed methodology revolves around the judicious utilization of the multi-head architecture called One Shot Transfer Learning [15]. One can train the model effectively in a non-stiff regime. This involves training the multi-head specifically for various small values of α. Then, the trained

latent space H encodes the properties of the equation and variations with respect to α within the non-stiff domain. Subsequently, it can be extracted by completing a forward pass through the trained network to save H. The weight-frozen version of H will be denoted as H^{fix}.

Drawing upon the network's architecture $u = HW$, we posit the existence of a layer $\widetilde{W}$ such that, for a stiff domain, $u = H^{fix}\widetilde{W}$. We deduce our new set of weights, by analytically determining the weights that minimize the ODE loss function corresponding to the stiff problem. This $\widetilde{W}$ corresponds to the generalized weights that satisfy the stiff differential equation, align with the training form, and are encapsulated by the latent space H^{fix}. Ultimately, we employ these weights in conjunction with the acquired H^{fix} to compute the stiff solution of the equation, thus finding $u = H^{fix}\widetilde{W}$.

Derivation of $\widetilde{W}$ for linear ODE

The convex nature of our network's loss function, as depicted in Equation 9, allows for analytical determination of the relevant weights, denoted as $\widetilde{W}$. Analytical derivation furnishes us with a closed-form representation of $\widetilde{W}$, a result not attainable through approaches such as empirically fine-tuning the last hidden layer of the network. The derivation starts with the loss function associated with the training of a PINNs on a linear ODE:

$$L = \frac{1}{N} \sum_{t \in [t_0, \dots, t_f]} [(\dot{u}_t + \widetilde{A} u_t - \widetilde{f})^2] + (u_0 - \widetilde{y_0})^2 \tag{14}$$

Here $\widetilde{A}$, $\widetilde{f}$ and $\widetilde{y_0}$ denote the set of new parameters in the regime where we want to compute the solution.

Now, since we know that $u = H^{fix}W$, we substitute into the loss function such that the loss is dependent on H^{fix} and W.

$$L(W) = \frac{1}{N} \sum_{t \in [t_0, \dots, t_f]} [(\dot{H_t^{fix}} W + \widetilde{A} H_t^{fix} W - \widetilde{f})^2] + (H_0^{fix} W - \widetilde{y_0})^2 \tag{15}$$

With H_t^{fix} is a $m \times d$ matrix which corresponds to the learnt latent space associated with the evaluation point t. We also denoted that $\dot{u}_t = \dot{H}^{fix} W$ because W is independent of the evaluation point t.

We want to find $\widetilde{W}$ that minimizes the loss function. In other words, we want to calculate the least

square solution of $L(W)$ finding $\widetilde{W}$ such that $\Delta_{\widetilde{W}} L = 0$, according to matrix derivation [28]

$$\begin{aligned}\Delta_W L = \frac{2}{N} \sum_{t\in[t_0,\dots,t_f]} &\{[(\dot{H}_t^{fix})^T + (H_t^{fix})^T \widetilde{A}^T][\dot{H}_t^{fix} W + \widetilde{A} H_t^{fix} W - \widetilde{f}]\} \\ &+2(H_0^{fix})^T [H_0^{fix} W - \widetilde{y_0}]\end{aligned} \tag{16}$$

Which gives after development:

$$\begin{aligned}\Delta_W L = \frac{2}{N} \sum_{t\in[t_0,\dots,t_f]} &\Big[\Big(\dot{H}_t^{fix}\Big)^T \dot{H}_t^{fix} W \\ &+ \Big(\dot{H}_t^{fix}\Big)^T \widetilde{A} H_t^{fix} W \\ &- \dot{H}_t^{fix} \widetilde{f} \\ &+ \Big(H_t^{fix}\Big)^T \widetilde{A}^T \dot{H}_t^{fix} W \\ &+ \Big(H_t^{fix}\Big)^T \widetilde{A}^T \widetilde{A} H_t^{fix} W \\ &- \Big(H_t^{fix}\Big)^T \widetilde{A}^T \widetilde{f}\Big] \\ &+ 2\Big(H_0^{fix}\Big)^T H_0^{fix} W - 2\Big(H_0^{fix}\Big)^T \widetilde{y_0}\end{aligned} \tag{17}$$

After equalizing to zero and isolating $\widetilde{W}$, we find that $\Delta_W L(\widetilde{W}) = 0$ if:

$$\begin{aligned}\widetilde{W} &= M^{-1}\Big((H_0^{\text{fix}})^T \widetilde{y_0} + \frac{1}{N}\sum_t \Big[(\dot{H}_t^{\text{fix}})^T \widetilde{f} + (H_t^{\text{fix}})^T \widetilde{A}^T \widetilde{f}\Big]\Big) \text{ with} \\ M = \frac{1}{N}\sum_t &\Big[(\dot{H}_t^{\text{fix}})^T \dot{H}_t^{\text{fix}} + (\dot{H}_t^{\text{fix}})^T \widetilde{A} H_t^{\text{fix}} + (H_t^{\text{fix}})^T \widetilde{A}^T \dot{H}_t^{\text{fix}} + (H_t^{\text{fix}})^T \widetilde{A}^T \widetilde{A} H_t^{\text{fix}}\Big] \\ &+(H_0^{\text{fix}})^T H_0^{\text{fix}}\end{aligned} \tag{18}$$

In order to perform transfer learning on ODEs, it is first necessary that a multi-head network is sufficiently trained on varying y_0, A, and f values. These values will ensure that the equation operates within a non-stiff regime. After training, a single forward pass must be completed through the trained model in order to obtain H^{fix}. We can compute the generalized weights $\widetilde{W}$ using Equation 18 for a chosen $\widetilde{y_0}$, $\widetilde{A}$, and $\widetilde{f}$ that operates in a stiffer regime. Finally, with H^{fix} and $\widetilde{W}$, the desired differential equation solution can be computed in "one-shot" by following u = $H^{fix}\widetilde{W}$.

The pivotal observation and the most potent aspect of this transfer learning approach lie in the fact that the network training process is required only once, until a new equation form needs to be

addressed. Assuming the persistence of the same equation form, parameters such as $\widetilde{y_0}$, $\widetilde{A}$, and $\widetilde{f}$ can be freely altered to new values without the need for additional training.

We conclude this section by highlighting which parts of Equation 18 need to be recomputed depending on what type of transfer learning is to be performed. First, if we want to change the initial condition y_0, then we simply need to recompute the $(H_0^{fix})^T y_0$ term. Next, if we choose to change A, we need to recompute M (and invert it) and $\frac{1}{N}\sum_t (H_t^{fix})^T A^T f$ as both of these pieces depend on A. Finally, if we choose to change f, we only need to calculate $\frac{1}{N}\sum_t \dot{H}_t^{fix} f$, $\frac{1}{N}\sum_t (H_t^{fix})^T A^T f$ given their dependence on f. It is important to note that adjusting y_0 or f involves recalculations through a few matrix multiplications and additions, while modifying A is more resource-intensive, requiring the recomputation and inversion of the M matrix.

Results

Below, we present results on two different examples of linear stiff ODEs. In each case, we present the system being solved, examine the stiffness regime and solve it accordingly to the methods discussed before. The stiffness parameter is always α and numerical solutions are compute using *scipy.integrate.solve_ivp* with the Radau method.

Damped Harmonic Oscillator

The first example is the Damped Harmonic Oscillator:

$$\frac{d^2y}{dt^2} + 2\alpha\omega_0\frac{dy}{dt} + \omega_0^2 y = 0 \tag{19}$$

with ω_0 the undamped angular frequency of the oscillator and α the damping ratio. This second-order ODE describes the motion of an harmonic oscillator slowed by the frictional force that is proportional to the velocity of the motion. We take $\omega_0 = 1$ for simplicity and transform it into a system of two first-order ODE according Equation 5.

For simplicity, we will substitute 2α with α while keeping the rest of the equation unchanged. Then $A = \begin{bmatrix} 0 & -1 \\ 1 & \alpha \end{bmatrix}$ and $f = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ is the vectorized representation of this system of ODEs:

$$\begin{cases} \frac{dy_1}{dt} - y_2 = 0, \quad y_1(0) = 1 \\ \frac{dy_2}{dt} + y_1 + \alpha y_2 = 0, \quad y_2(0) = 0.5 \end{cases}, \quad \text{for } 0 \leq t \leq 10. \tag{20}$$

The Stiffness Ratio of this system of ODEs is proportional to α^2 (Derivation in the Appendix). In this scenario, the Stiffness Ratio serves as a reliable indicator of the equation's stiffness, and the solution demonstrates a rapid transient phase in the vicinity of $y_2(0)$, which becomes more significant with increasing α.

Learning Stiffness: *Vanilla PINNs*

In this part, the single-head approach is used to try to directly learn the solution in the stiff domain. Here are several results with increasing stiffness value.

- $\alpha = 10$ and $SR \cong 100$

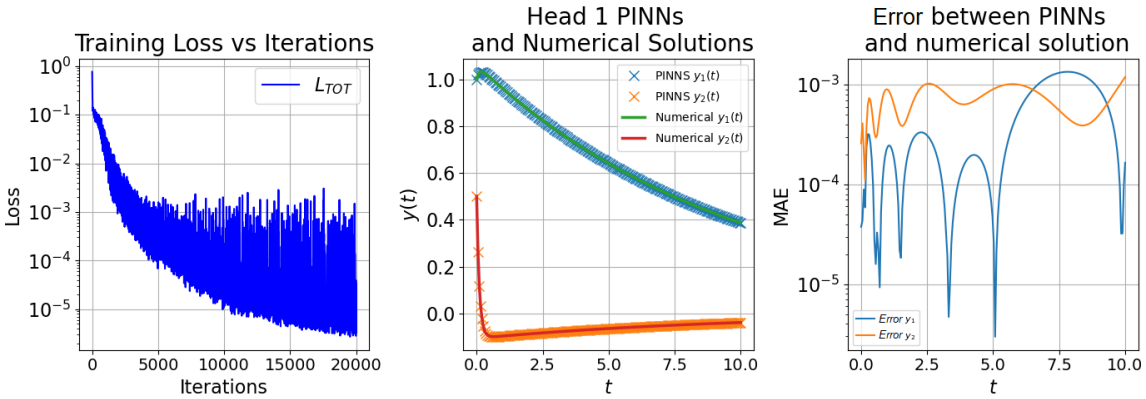

Figure 5: Training results for Equation 20 with $\alpha = 10$ after 20000 iterations with $l_r = 10^{-4}$. The first plot shows the evolution of the loss during training, the second plot shows the network learned solution versus the numerical solution and the third plot highlights the absolute error (absolute difference between network and numerical solution)

We observe that after 20000 iterations, the loss value reaches around 10^{-5}. It can be observed that the loss exhibits pronounced oscillations, a characteristic behavior indicative of training within a stiff regime. However, it is noteworthy that the overall trajectory of the loss remains downward, with the final model situated within the lower range of these oscillations. Additionally, by plotting the network-learned solutions on top of the true solutions it becomes clear that the network has learned the solution. The error plot correspond to the Absolute Error it reach 10^{-3} and 10^{-4} for y_1 and y_2 respectively.

- $\alpha = 20$ and $SR \cong 400$

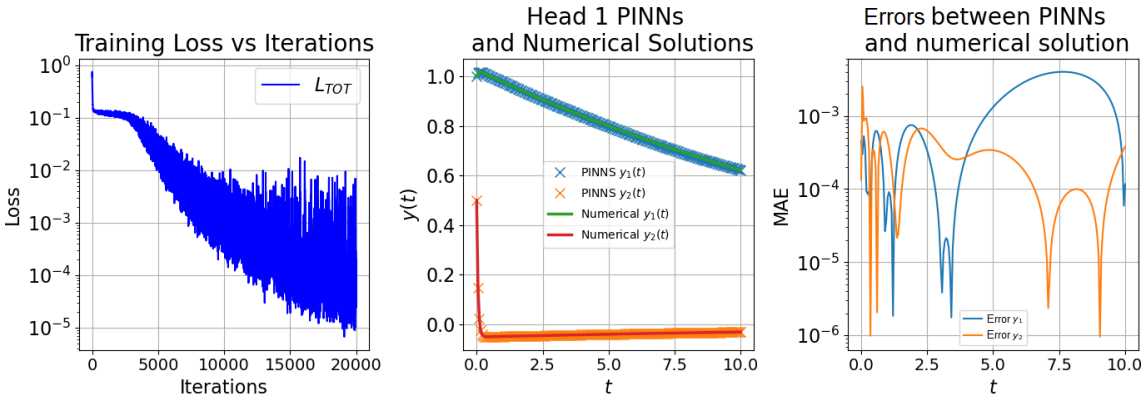

Figure 6: Training results for Equation 20 with $\alpha = 20$ after 20000 iterations with $l_r = 10^{-5}$

We observe that after 20000 iterations, the loss value reaches around 10^{-5}. The oscillations are even bigger and the learning rate must have been reduced compare to the previous training. However, the model managed to learn the solution and the Absolute Error reach around 10^{-3} and 10^{-4}.

- $\alpha = 40$ and $SR \cong 1600$

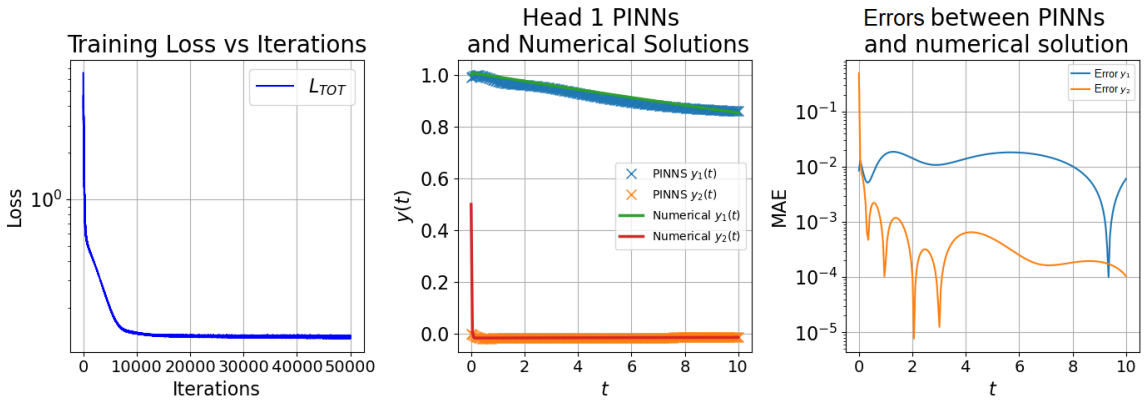

Figure 7: Training results for Equation 20 with $\alpha = 40$ after 50000 iterations with $l_r = 10^{-6}$

We observe that after 10000 iterations, the network faced difficulty to learn more the equation. Indeed, the loss value is stuck and there is no clear improvement until 50000 iterations. The solution y_2 has non-negligible errors of order of 10^{-2} and it fails to encode the transient phase of y_2 so the initial condition is not satisfied.

- $\alpha = [10, 50]$ and $SR \cong [100, 2500]$

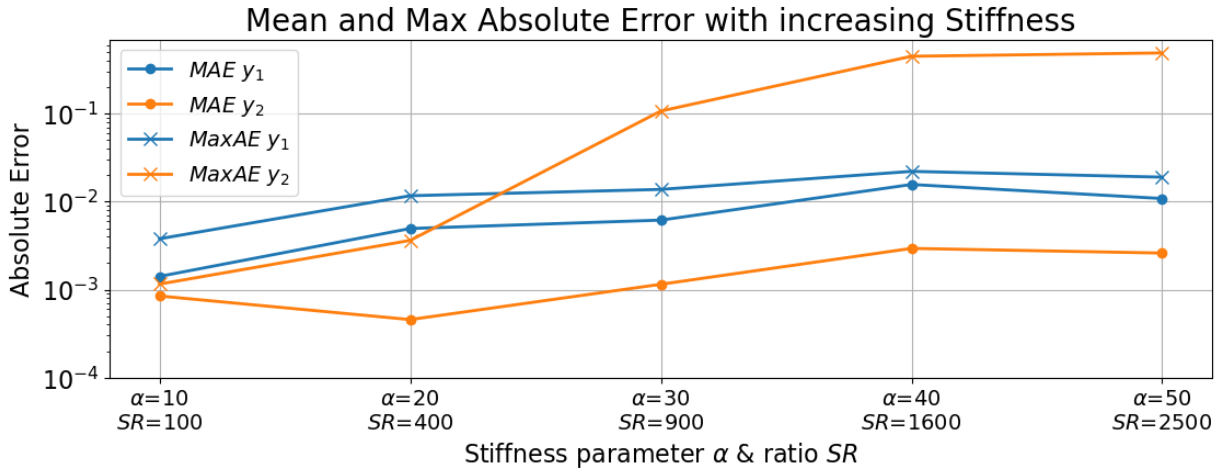

Figure 8: Mean Absolute Error and Maximum Absolute Error of Equation 20 by training a PINNs as α increase from 5 to 50

The figure 8 shows the evolution of the Mean Absolute Error (MAE) and the Maximum Absolute Error ($MaxAE$) as the stiffness increases. From a general perspective, the error tends to increase with the growth in stiffness. In the low stiffness regime, the model demonstrates effective encoding

of the solution, as indicated by metrics in the range of 10^{-4} to 10^{-2} for $\alpha = 10, 20$. Regarding y_2, there is a disparity between MAE and $MaxAE$. This difference can be attributed to the model's challenge in capturing the transient phase within the time scale of the initial condition $y_2(0) = 0.5$ (see Figure 18), resulting in $MaxAE = 0.5$ from $\alpha = 40$. For y_1, both metrics exhibit similar behavior, with an increasing trend observed as α rises.

Acquiring solutions within a stiff domain appears challenging with the single-head architecture. For stiffness values deemed reasonable $\alpha < 20$, a solution may necessitate extensive training, leading to instability in the loss. Moreover, for domains characterized by higher stiffness $\alpha > 20$, the model struggles to effectively encode the solution's stiffness attributes, like the transient phase.

Transfer Learning to Stiff Regime: *One Shot Transfer Learning*

In this part, the multi-head approach is used to learn a latent space representation of the equation in a non-stiff regime and use it to transfer learning to the solution in the stiff domain.

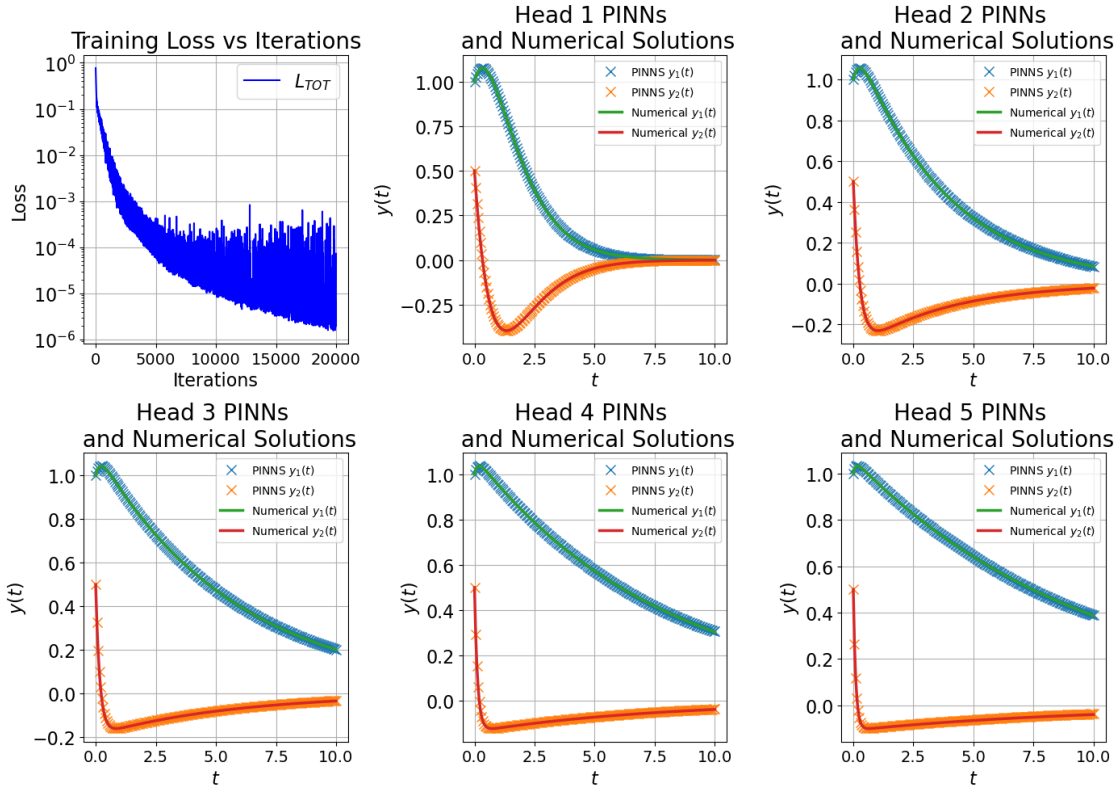

Figure 9: Training results for Equation 20 after 20000 iterations with $l_r = 10^{-4}$, using five heads, $\alpha \in \{2, 4, 6, 8, 10\}$ and $SR \cong \{1, 16, 36, 64, 100\}$. The 1^{st} plot illustrates the loss vs. iterations. The remaining five plots show the network learned solutions (blue/orange) versus the true solutions (green/red) for each head's equation configurations.

Figure 9 shows the results of training on Equation 20 in a non-stiff domain. The α and SR of each head are $[2, 4, 6, 8, 10]$ and $[1, 16, 36, 64, 100]$. It is evident that we have sufficiently trained the network, as our loss reaches around 10^{-5} and 10^{-6}. The network learned solutions, and numerical solutions align very closely across all five heads.

Now we can extract the H_{fix} latent representation of this training session and compute $\widetilde{W}$ in a stiff regime according to equation 18. Only A is dependant of the stiff parameter α in equation 20 so we just need to change it to $\widetilde{A}$ (this will result in recomputing the M matrix and invert it).

- $\alpha = 20$ and $SR \cong 400$

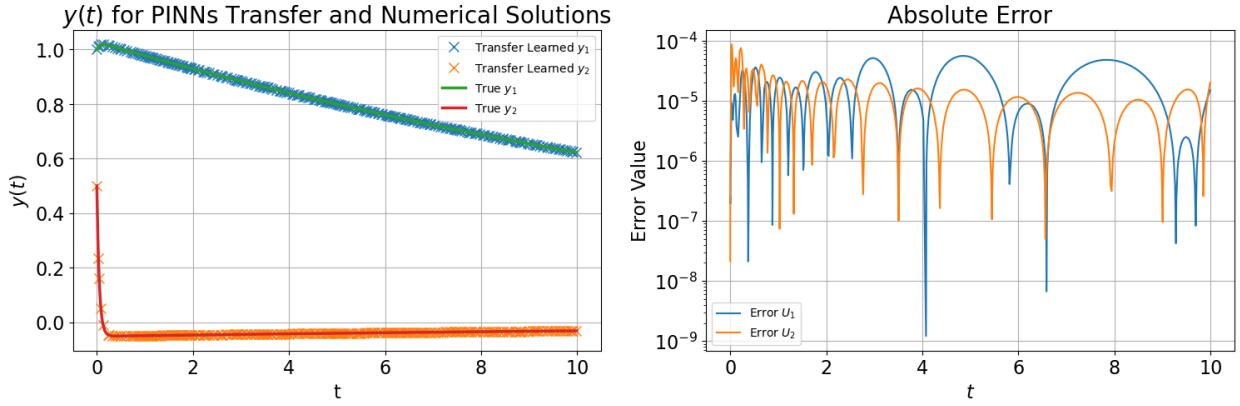

Figure 10: Transfer learning results to $\alpha = 20$. The left subplot shows the transfer learned solutions (blue/orange) versus the true solutions (green/red) and the right subplot shows the absolute error.

The obtained solution aligns well with the numerical solution, and the initial conditions are satisfied. The error plot reflects the absolute error between PINNs and numerical solution, which is approximately 10^{-4} and 10^{-5}.

- $\alpha = 40$ and $SR \cong 1600$

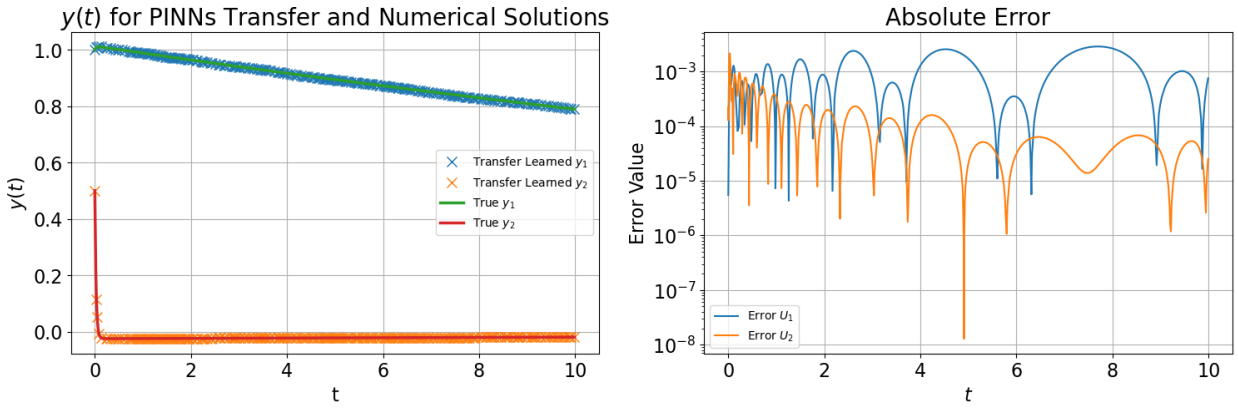

Figure 11: Transfer learning results to $\alpha = 40$. The left subplot shows the transfer learned solutions (blue/orange) versus the true solutions (green/red) and the right subplot shows the absolute error.

The obtained solution aligns well with the numerical solution, and the initial conditions are satisfied. The error plot reflects the absolute error between PINNs and numerical solution, which is approximately 10^{-3} and 10^{-4}.

- $\alpha = 60$ and $SR \cong 3600$

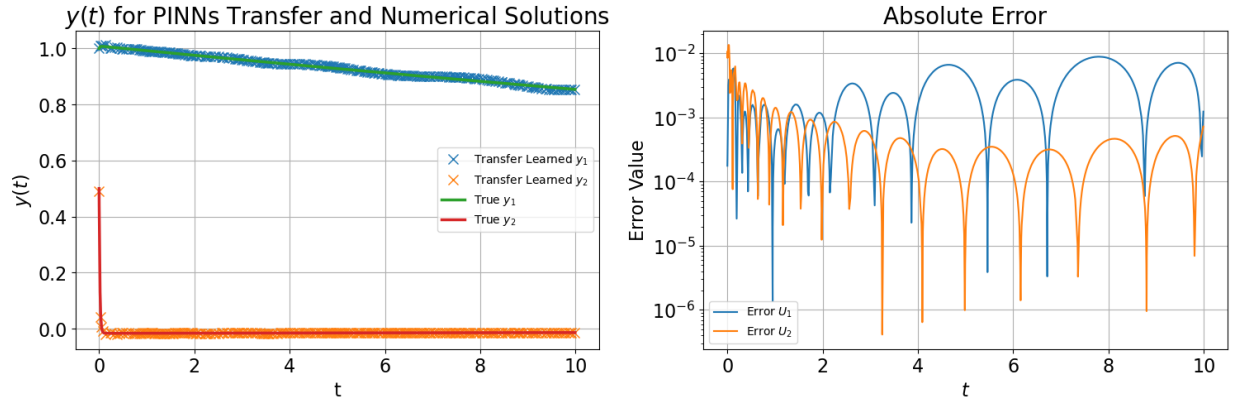

Figure 12: Transfer learning results to $\alpha = 60$. The left subplot shows the transfer learned solutions (blue/orange) versus the true solutions (green/red) and the right subplot shows the absolute error.

The obtained solution aligns with the numerical solution, and the initial conditions are satisfied. It is starting to have some oscillations on y_1. The error plot reflects the absolute error between PINNs and numerical solution, which is approximately 10^{-2} and 10^{-3}.

- $\alpha = 100$ and $SR \cong 10000$

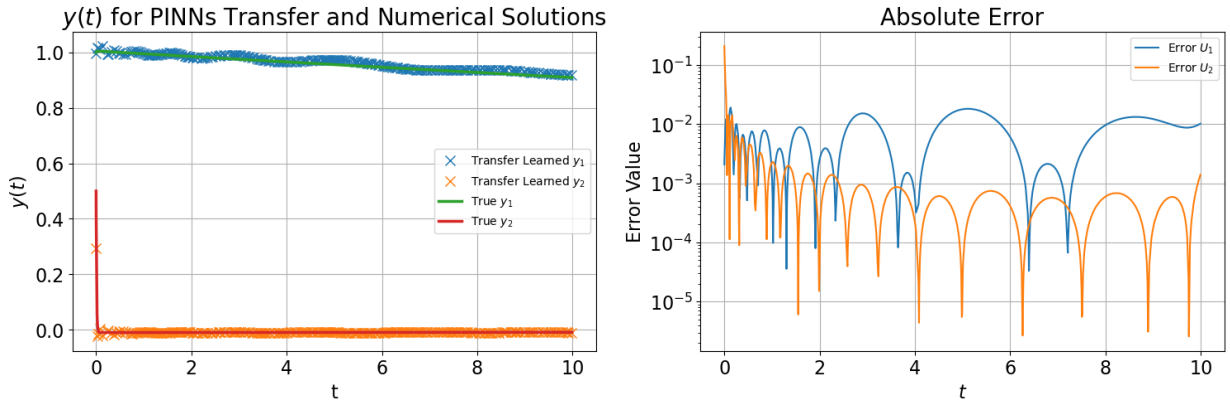

Figure 13: Transfer learning results to $\alpha = 100$. The left subplot shows the transfer learned solutions (blue/orange) versus the true solutions (green/red) and the right subplot shows the absolute error.

The obtained solution does not align with the numerical solution, and the initial conditions are not completely satisfied. There are non negligible oscillations on y_1. The error plot reflects the absolute error between PINNs and numerical solution, which is approximately around 10^{-1} and 10^{-3}. It appears that we have approached the stiff limit, where a transfer can be made without loss of precision.

- $\alpha \in [10, 150]$ and $SR \in [100, 22500]$

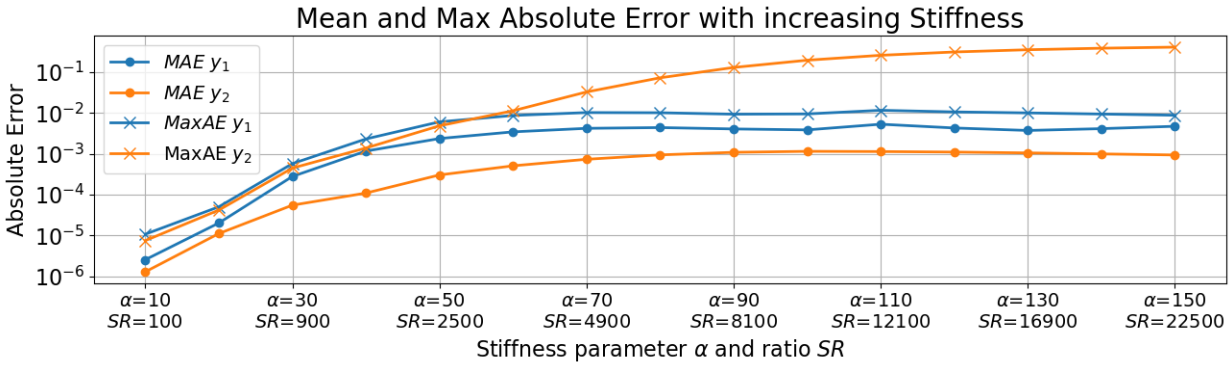

Figure 14: Mean Absolute Error and Maximum Absolute Error of Equation 20 using Transfer Learning as α increase from 10 to 150

As depicted in Figure 14, a general observation indicates an increase in error with the rise in stiffness. The MAE rises from 10^{-5} until reaching a plateau within the range of 10^{-3} for both solutions. Upon examining $MaxAE$ of y_2, it is evident that the challenging stiff behavior is still to

capture the transient phase of y_2. The value 10^{-2} of the $MaxAE$ of y_2 is reached at $\alpha = 60$, and the maximum 0.5 is reached at $\alpha = 130$. On the other side, the $MaxAE$ of y_1 stabilizes around the reasonable value of 10^{-2}. We can transfer to a stiffer regime, around $\alpha \simeq 60$, in comparison to the training regime where $\alpha < 10$, without experiencing a substantial loss in accuracy.

- Computational Time comparative studies with numerical methods: *increasing α stiffness parameter*

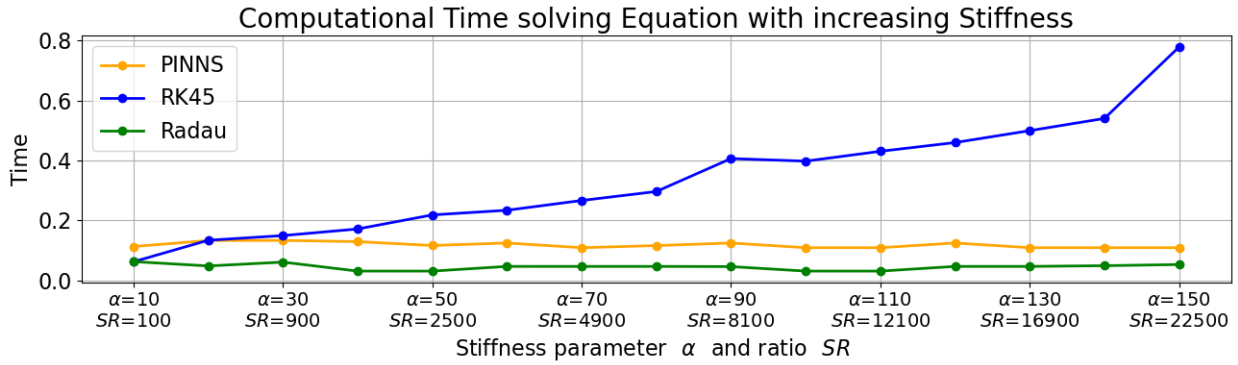

Figure 15: Comparison of computational time solving equation 20 with PINNs, RK45 and Radau with increasing stiffness

One can observe from Figure 15 that the computation time of RK45 increases significantly as the stiffness of the equation rises. This can be explained by its explicit nature. The transfer learning method employing PINNs competes effectively with the implicit Radau method, demonstrating computation times of comparable magnitude and relatively independant of the stiffness of the equation.

- Computational Time comparative studies with numerical methods: *change initial y_0 condition within a stiff domain*

In this context, our focus is on modifying the initial conditions within the stiff domain. To achieve this, we require a latent space representation H_{fix} capable of handling various initial conditions. Consequently, we trained a multi-head model under non-stiff regimes, each characterized by distinct initial conditions.

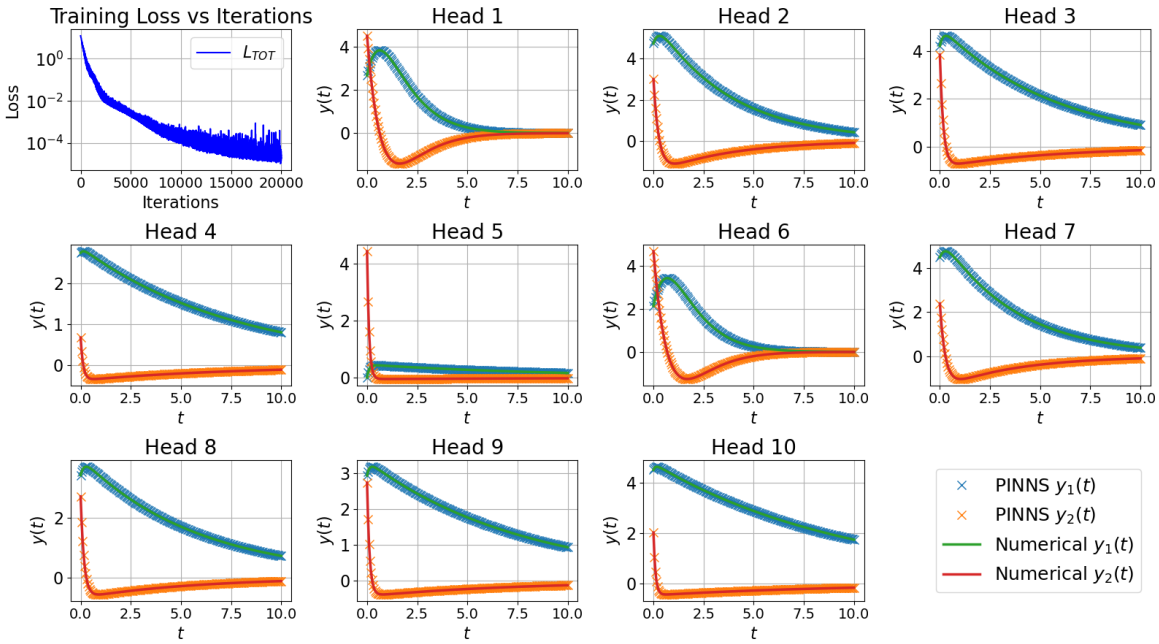

Figure 16: Training results for Equation 20 after 20000 iterations with $l_r = 10^{-4}$, using ten heads, $\alpha \in \{2, 4, 6, 8, 10\}$, $y_1(0) \in [0, 5]$ and $y_2(0) \in [0, 5]$

The figure 42 shows the training outcome of the multihead with variying initial conditions. The network training is successful, reflected by the achieved low losses of approximately 10^{-4}. The learned solutions closely match the numerical solutions across all ten heads.

Now, we can extract the H_{fix} latent representation of this training session and compute the M in a stiff regime according to equation 18. Then, we will compute several solutions with various initial conditions different from the training one. We do not need to recompute the M matrix because it is independent from the initial conditions.

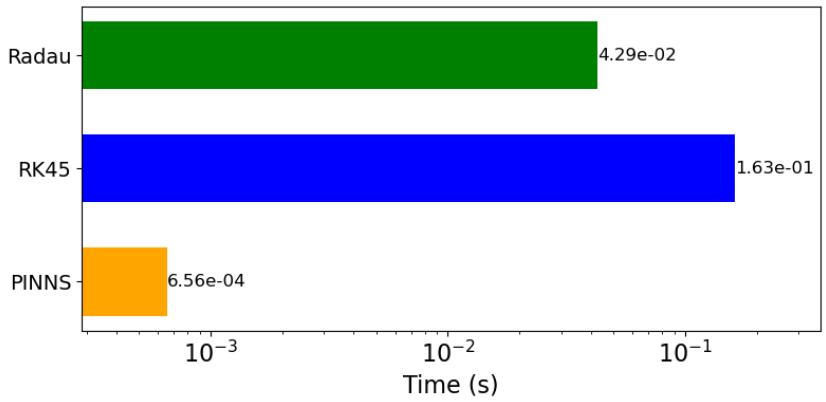

Figure 17: Average computational time solving 1000 stiff equations with $\alpha = 60$ by changing initial conditions at each iterations

The figure 17 show that the transfer learning PINNs method was $\cong$ 70 time faster than the Radau Method and 300 time that RK45. These experiments were conducted by aligning the tolerance of the numerical methods to the same order of magnitude of precision as the PINNs at this stiffness regime, specifically, 10^{-3}.

System of ODEs with not constant force function

The second example, discussed in [29] is the following system of ODEs:

$$\begin{cases} \frac{dy_1}{dt} + 2y_1 - y_2 = 2\sin(\omega t), \quad y_1(0) = 2 \\ \frac{dy_2}{dt} - (\alpha - 1)y_1 + \alpha y_2 = \alpha(\cos(\omega t) - \sin(\omega t)), \quad y_2(0) = 10 \end{cases}, \quad \text{for } 0 \leq t \leq 10. \tag{21}$$

Here, the force function is not constant and depends on the stiffness parameter α. The Stiffness Ratio is proportional to α (see Appendix for derivation). Using this value to compare the stiffness of this equation with the previous example is not feasible because this value only catch the impact of the eigenvalues. When examining the time needed to solve the equation through an explicit method, it is comparable to that of Equation 20 for the corresponding value of α. Even though the Stiffness Ratio in the prior example was α^2, another contributing factor to the increased stiffness of this equation is certainly the force function and the initial conditions. For the next part, $\omega = 1$ when no other values are mentioned.

Learning Stiffness: *Vanilla PINNs*

As in the previous example, the application of the single-head model to learn the solution in the stiffness regime yielded inconclusive results. Below is provided the evolution of MAE and $MaxAE$ with the method as the stiffness increase.

- $\alpha = [10, 40]$

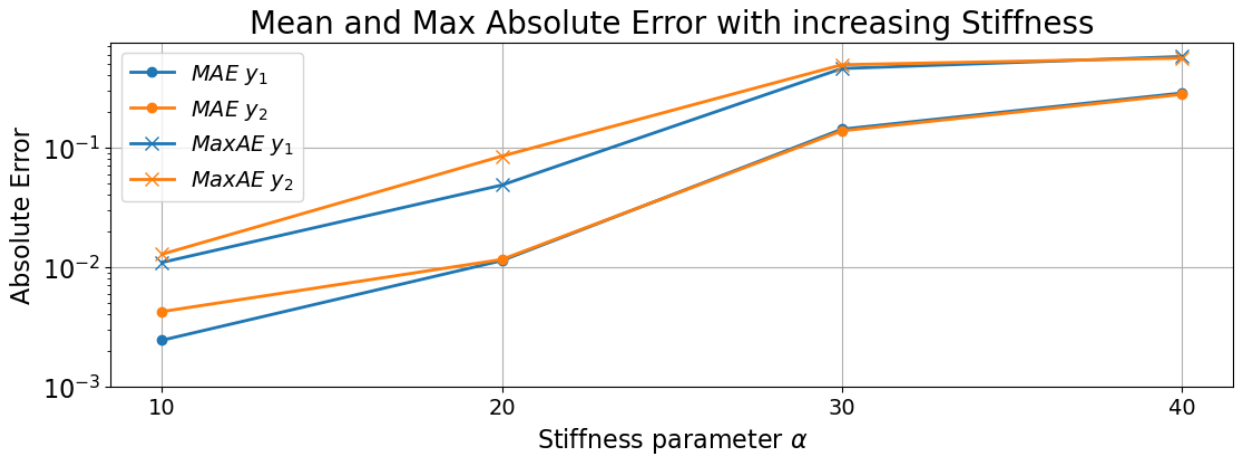

Figure 18: Mean Absolute Error and Maximum Absolute Error by training a PINNs on Equation 21 as α increases from 10 to 40

In the least stiff regime with $\alpha = 10$, the model is reasonably capable of encoding the solution, as evidenced by metrics ranging from 10^{-3} to 10^{-2}. Nevertheless, it quickly starts to exhibit a noteworthy error, surpassing 10^{-2} and 10^{-1} for MAE and $MaxAE$, respectively, as $\alpha > 10$. The model is therefore unable to learn and capture the stiffness in these regimes.

Transfer Learning to Stiff Regime: *One Shot Transfer Learning*

The multi-head approach is used to learn a latent space representation of the equation in a non-stiff regime and use it to transfer learning to the solution in the stiff domain.

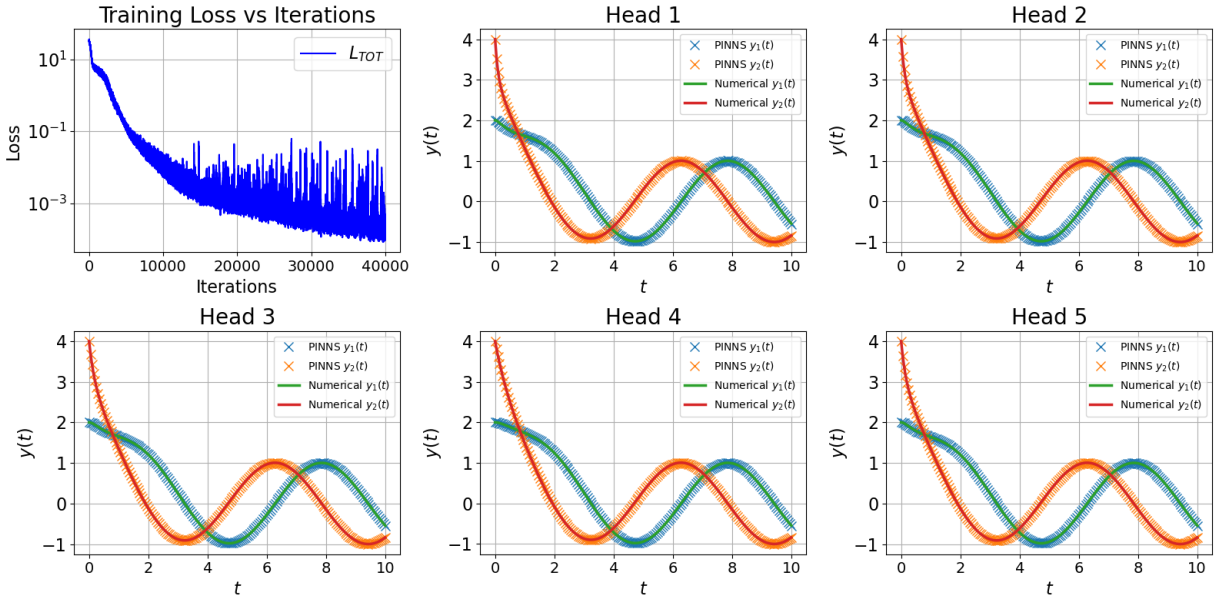

Figure 19: Training results for Equation 21 after 40000 iterations with $l_r = 10^{-4}$, using five heads, $\alpha \in \{2, 4, 6, 8, 10\}$ ($SR \cong$ same). The 1^{st} plot illustrates the loss vs. iterations. The remaining five plots show the network learned solutions (blue/orange) versus the true solutions (green/red) for each head's equation configurations.

Figure 19 shows the results of training on Equation 21 in a non-stiff domain. The α of each head are $[2, 4, 6, 8, 10]$. The loss reaches around 10^{-3} and 10^{-4}. The network learned solutions, and numerical solutions align very closely across all five heads.

Now we can extract the H_{fix} latent representation of this training session and compute $\widetilde{W}$ in a stiff regime according to equation 18. A and f are dependant of the stiff parameter α in equation 21.

- $\alpha = 20$

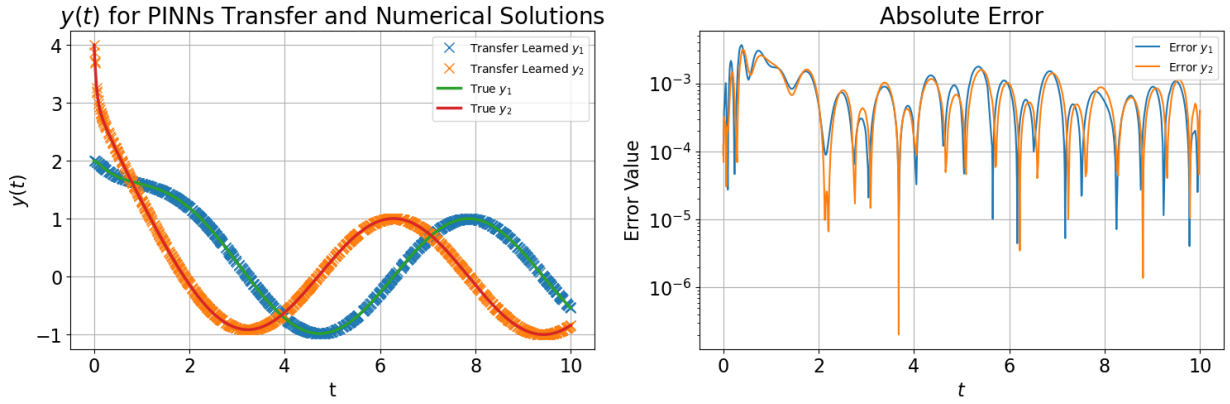

Figure 20: Transfer learning results to $\alpha = 20$. The left subplot shows the transfer learned solutions (blue/orange) versus the true solutions (green/red) and the right subplot shows the absolute error.

The obtained solution aligns well with the numerical solution, and the initial conditions are satisfied. The error plot reflects the absolute error between PINNs and numerical solution, which is approximately between 10^{-3} and 10^{-5}.

- $\alpha = 50$

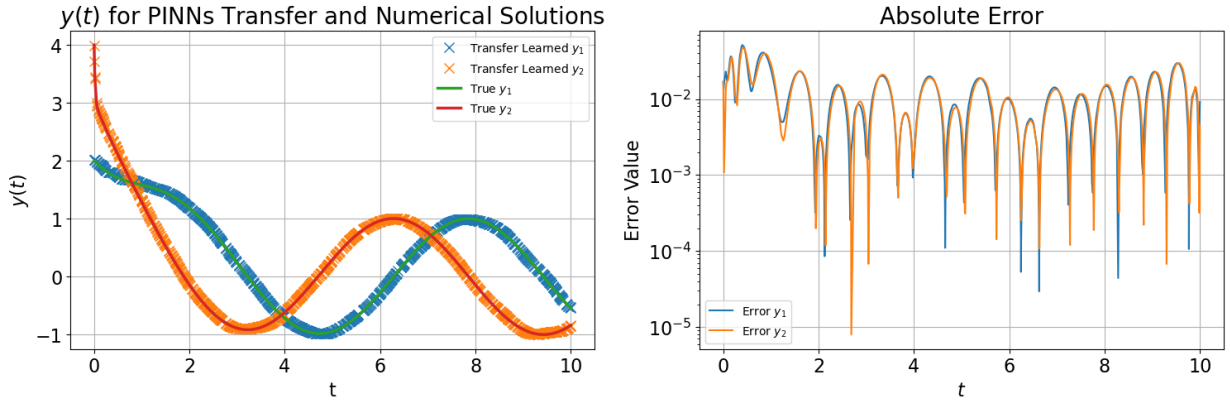

Figure 21: Transfer learning results to $\alpha = 50$. The left subplot shows the transfer learned solutions (blue/orange) versus the true solutions (green/red) and the right subplot shows the absolute error.

The obtained solution aligns well with the numerical solution, and the initial conditions are satisfied. The error plot reflects the absolute error between PINNs and numerical solution, which is approximately between 10^{-2} and 10^{-3}.

- $\alpha = 100$

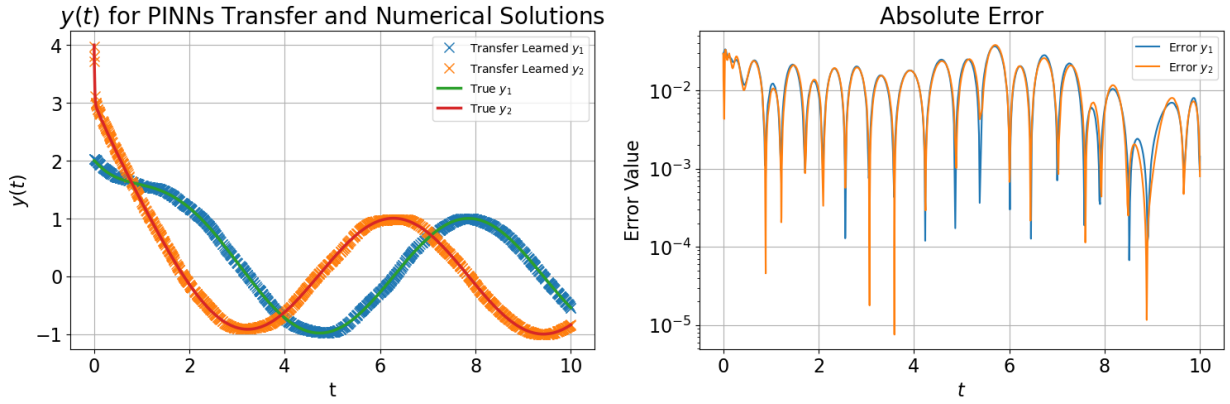

Figure 22: Transfer learning results to $\alpha = 100$. The left subplot shows the transfer learned solutions (blue/orange) versus the true solutions (green/red) and the right subplot shows the absolute error.

The obtained solution aligns well with the numerical solution, and the initial conditions are satisfied. The error plot reflects the absolute error between PINNs and numerical solution, which is approximately around 10^{-2}.

- $\alpha = 150$

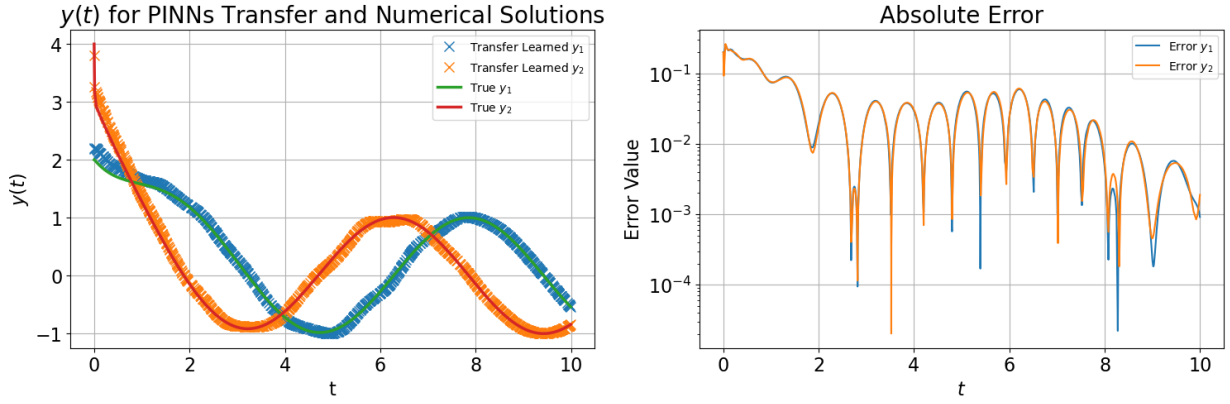

Figure 23: Transfer learning results to $\alpha = 150$. The left subplot shows the transfer learned solutions (blue/orange) versus the true solutions (green/red) and the right subplot shows the absolute error.

The obtained solution does not align well with the numerical solution. We can clearly see some deviation from the numerical solution. The initial conditions are not adhered to perfectly, with an error exceeding 10^{-1}.

- $\alpha \in [10, 150]$

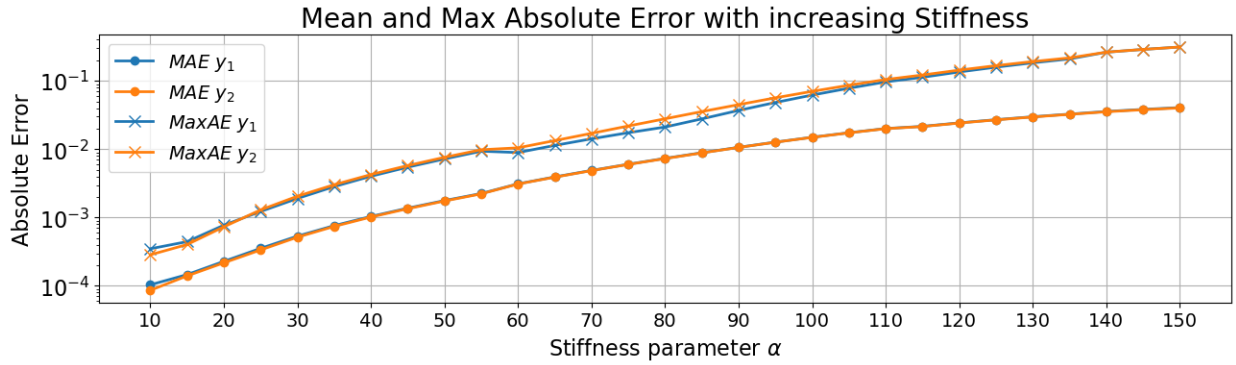

Figure 24: Mean Absolute Error and Maximum Absolute Error of Equation 21 as α increase from 10 to 150

We can see on Figure 24 that from a general perspective, the error tends to increase with the growth in stiffness. It is worth noting that for this equation, the magnitude of the solution is on the order of unity. Therefore, an error of 10^{-2} is more reasonable compared to the previous example. The initial noteworthy observation is the similarity in the errors of y_1 and y_2, with relatively constant differences between MAE and $MaxAE$. The MAE remains within the range of 10^{-3} to 10^{-2} up to $\alpha = 90$. Subsequently, it starts to show a notable increase, reaching values around 10^{-1} by the value $\alpha = 150$. The $MaxAE$ follows a similar increasing trend as MAE but with a scaling factor of approximately less than 10.

We can transfer to a stiffer regime, around $\alpha \simeq 90$, in comparison to the training regime where $\alpha < 10$, without experiencing a substantial loss in accuracy.

- Computational Time comparative studies with numerical methods: *increasing α stiffness parameter*

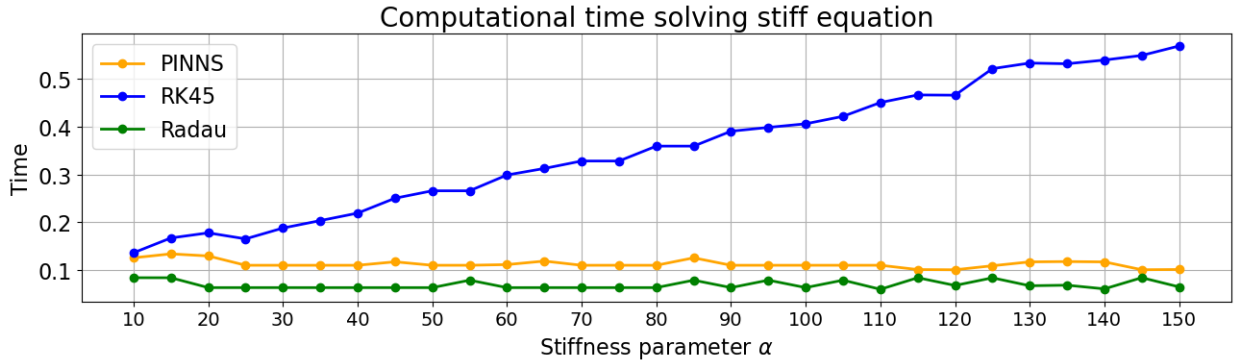

Figure 25: Comparison of computational time solving equation 21 with PINNs, RK45 and Radau with increasing stiffness

One can observe from Figure 25 similary to the previous example the computation time of RK45 increases significantly as the stiffness of the equation rises. The PINNs transfer learning method competes effectively with the implicit Radau method.

- Computational Time comparative studies with numerical methods: *change ω in the force function within a stiff domain*

In this context, our focus is on modifying the force within the stiff domain. To achieve this, we require a latent space representation H_{fix} capable of handling different force function, different value of ω in equation 21. Consequently, we trained a multi-head model under non-stiff regimes with different ω value.

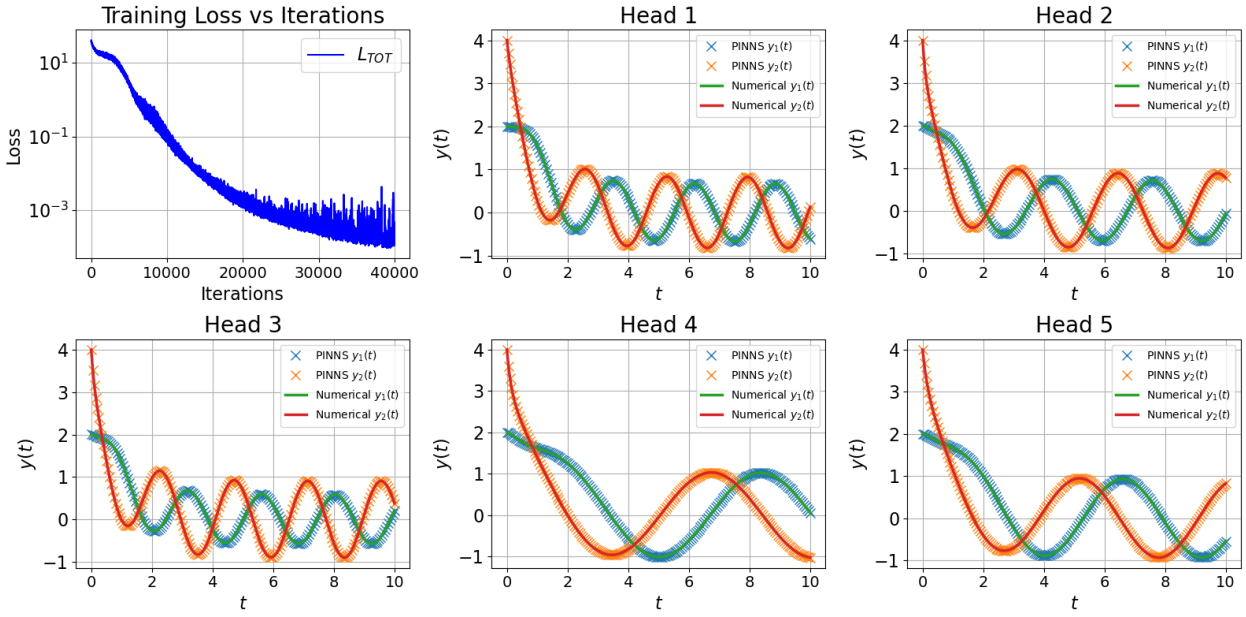

Figure 26: Training results for Equation 21 after 40000 iterations with $l_r = 10^{-4}$, using five heads, $\alpha \in \{1, 2, 3, 4, 5\}$ and $\omega \in [0, \pi]$

The figure 26 shows the training outcome of the multihead with different ω value. The network training is successful, reflected by the achieved low losses of approximately 10^{-4}. The network can handle variation of frequencies in the force function. The learned solutions closely match the numerical solutions.

Now, we can extract the H_{fix} latent representation of this training session and compute the M in a stiff regime according to equation 18. Then, we will compute several solutions with various ω, different from the training ones. We do not need to recompute the M matrix.

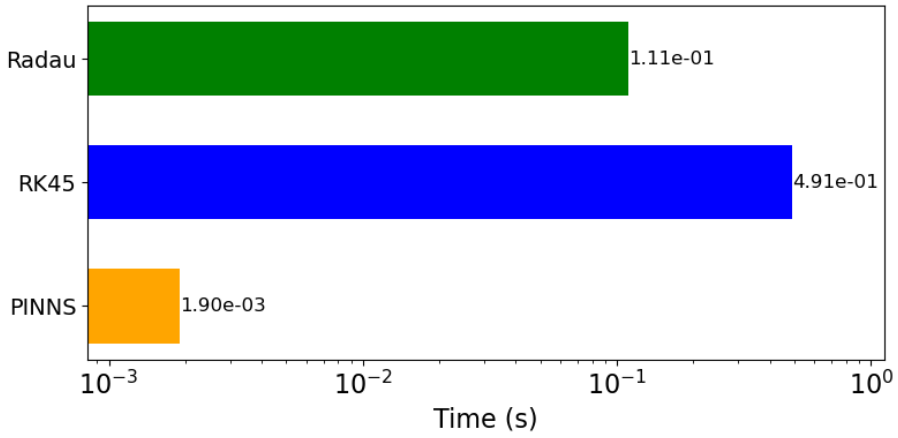

Figure 27: Average computational time solving 1000 stiff equations with $\alpha = 90$ by changing ω in the force function at each iteration with $\omega \in [0, \pi]$

The figure 27 shows that the transfer learning PINNs method is about 60 time faster than the Radau Method and about 250 time that RK45.

Non-linear ODE

In this section, we will now turn our attention to non-linear ODEs. These equations are more intricate to solve compared to linear counterparts and lack vectorized forms. Initially, we will present various methods employed for solving these equations within a stiff domain before delving into the results.

Methods

Three distinct methodologies will be employed to address the resolution of the equation within a stiff domain. The first fapproach is the same as the linear one trying to learn the stiffness. The second and third method connect the idea of transfer learning of the linear method by integrating the difficult brought by the non-linearity.

Learning Stiffness: *Vanilla PINNs*

As explained previously, we will try to learn the solution of the ODE by training a PINNs directly in the stiff regime. The only changes compared to the linear approach are: more layers in the network architecture (refer to Table 1b), and the reparametrization approach used to handle the initial conditions.

Transfer Learning to Stiff Regime: *LBFGS Transfer Learning*

We will learn a latent space representation of the equation H_{fix} in a non-stiff regime using Table 1b for the layer architecture and the reparametrization approach to handle the initial conditions. Because of the non-linearity, we can not derived a least square approximation of $\widetilde{W}$, we have to fine tuned this last layer iteratively. The optimization algorithm used to learn this last layer is Limited-memory Broyden–Fletcher–Goldfarb–Shanno [30] (LBFGS). It belongs to the family of quasi-Newton methods and stands out for its efficient memory usage by approximating the Hessian matrix with limited storage. It is a good trade off between computational time and accuracy order to learn the last layer $\widetilde{W}$.

Transfer Learning to Stiff Regime: *Perturbation Approach*

The objective of this method [16] is to approximate the non-linear ODE by some of linear ODEs that will be solved using the linear one shot transfer learning approach. We will focus on non-linear ODEs of the form:

$$\dot{y} + Ay + \beta g(y) = f(t) \tag{22}$$

with y a vector, $\dot{y}$ a vector of the derivative of y, A the matrix of the coefficient of linear part, $g : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is a non-linear function, $\beta \in [0, 1]$ is the perturbation coefficient and f the forcing function terms.

To transform this non-linear system of ODEs into several linear ones, a perturbation expansion is used. We first make the assumption $y = \sum_{i=0}^{\infty} \beta^i Y_i \simeq \sum_{i=0}^{p} \beta^i Y_i$ with $p \in \mathbb{N}$. It implies that $\dot{y} = \sum_{i=0}^{p} \beta^i \dot{Y}_i$, by inserting these into equation 22:

$$\sum_{i=0}^{p} \beta^i \dot{Y}_i + A \sum_{i=0}^{p} \beta^i Y_i + \beta g(\sum_{i=0}^{p} \beta^i Y_i) = f(t) \tag{23}$$

Then, by collecting the different power of β:

$$\begin{aligned}
&\beta^0 \Rightarrow \dot{Y_0} + AY_0 = f(t) \\
&\beta^1 \Rightarrow \dot{Y_1} + AY_1 = F_1(Y_0) \\
&\beta^2 \Rightarrow \dot{Y_2} + AY_2 = F_2(Y_0, Y_1) \\
&\quad \vdots \\
&\beta^p \Rightarrow \dot{Y_p} + AY_p = F_p(Y_0, \ldots, Y_{p-1})
\end{aligned} \tag{24}$$

with F_i is a function that collects the β^i term of g. By construction F_i depends only on the previous solutions Y_0, ..., Y_{i-1}. We have transformed the non-linear system of ODEs (22) into $p + 1$ linear system of ODEs of the form:

$$\forall i = 0, 1, \ldots, p : \quad \dot{Y_i} + AY_i = F_i(Y_0, Y_1, \ldots, Y_{i-1}) \quad \text{with} \quad F_i = \begin{cases} f(t), & \text{if } i = 0 \\ \beta^i \text{ term of } g, & \text{otherwise} \end{cases} \tag{25}$$

The initial conditions for each system are now:

$$Y_i(0) = \frac{y_0}{\sum_{j=0}^{p} \beta^j} \quad \forall i \in [0, p] \tag{26}$$

The approach involves solving these systems in an iterative way. We can employ the transfer learning method described by Equation 18 to compute the PINNs solution U_i of each systems. Then the final PINNs solution is calculated with $u = \sum_{i=0}^{p} \beta^i U_i$.

In the context of stiff equation, we want to:

- Train a multi-head model on the linear part of the equation 22$\longrightarrow \dot{y} + Ay = F_0$ in a non-stiff regime.
- Extract the latent representation H_{fix}
- Use the equation 18 to compute iteratively the solution $U_i = H_{fix}\widetilde{W}_i$ of the p systems of equation 25 in a stiff regime
- Compute the final solution with $u = \sum_{i=0}^{p} \beta^i U_i$

Note that because the systems are linear, we use the architecture of Table 1a and the learning of initial conditions for the training part.

Results

Below, we present training and PINNs results of a non-linear stiff ODEs. The stiffness parameter is always α. In all cases, the true or actual solutions are computed using *scipy.integrate.solve_ivp* with the Radau method.

Duffing Equation

The non-linear example is the Duffing equation. It is a non-linear second-order differential equation used to model certain damped and driven oscillators. The equation is given by:

$$\frac{d^2y}{dt^2} + \alpha\frac{dy}{dt} + \delta y + \beta y^3 = \gamma cos(\omega t) \tag{27}$$

which can be rewritten in a system of two first-order ODEs as explained in Equation 5:

$$\begin{cases} \frac{dy_1}{dt} - y_2 = 0 \\ \frac{dy_2}{dt} + \delta y_1 + \alpha y_2 + \beta y_1^3 = \gamma \cos(\omega t), \end{cases} \tag{28}$$

By investigating the eigenvalues of this system, it appears that the Stiffness Ratio is proportional to α^2 (see Appendix for the derivation). We assign values to the parameters as follows: $\delta = 0.1$, $\gamma = 1$, $\omega = 1$, and let $\beta \in [0, 1]$ govern the degree of non-linearity. The resulting system of ODEs to be solved is then as follows:

$$\begin{cases} \frac{dy_1}{dt} - y_2 = 0 & y_1(0) = 1 \\ \frac{dy_2}{dt} + 0.1y_1 + \alpha y_2 - \beta y_1^3 = \cos(t) & y_2(0) = 0.5 \end{cases} \quad \text{for } 0 \leq t \leq 5. \tag{29}$$

The default value for β is 0.5. If modified, it will be mentioned.

Learning Stiffness: *Vanilla PINNs*

- $\alpha = 10$ and $SR \cong 100$

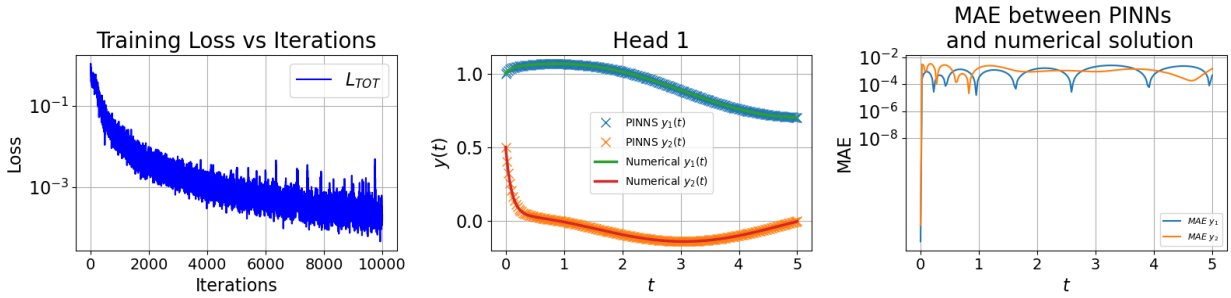

Figure 28: Training results for Equation 29 with $\alpha = 10$ after 10000 iterations with $l_r = 10^{-5}$

We observe that after 10000 iterations, the loss value reaches around 10^{-4}. Additionally, by plotting the network-learned solutions on top of the true solutions it becomes clear that the network has learned the solution. The error plot represents the Absolute Error, exhibiting notably low values for the initial conditions due to the reparametrization. It reaches around 10^{-3} for both y_1 and y_2 across the remaining time scale.

- $\alpha = 30$ and $SR \cong 100$

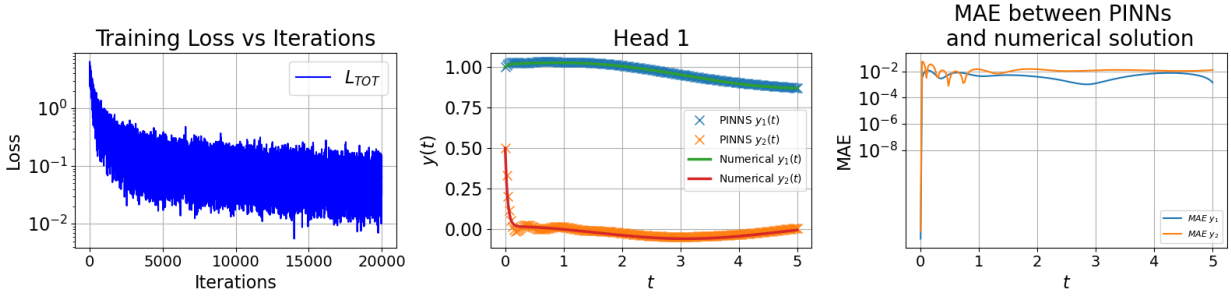

Figure 29: Training results for Equation 29 with $\alpha = 30$ after 20000 iterations with $l_r = 10^{-6}$

We observe that after 20000 iterations, the loss value reaches around 10^{-2}. We notice significant oscillations that highlight the challenge for the network to learn the stiffness of the equation. The error magnitude is 10^{-2}, which is non-negligible and evident in the plots of both the PINNs and numerical solutions.

As concluded for linear equations, it appears challenging for single-head networks to capture the stiffness behavior of the equation, even with the reparametrization of the initial conditions. For high stiffness values, a solution may necessitate extensive training, resulting in loss instability and computational expense.

- $\alpha \in [10, 40]$

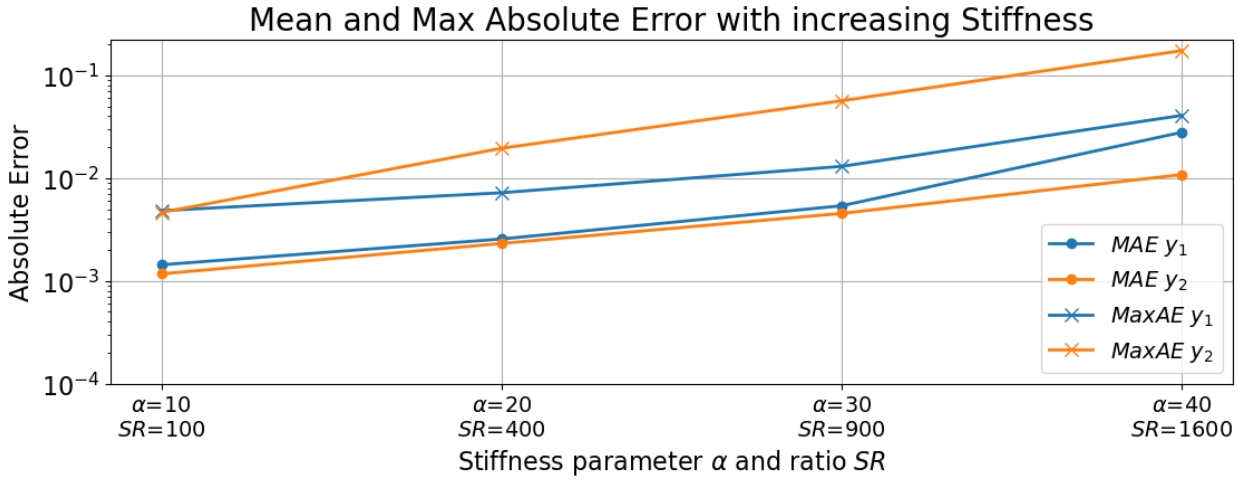

Figure 30: Mean Absolute Error and Maximum Absolute Error solving training a PINNs on Equation 29 as α increase from 10 to 40

In the least stiff regime with $\alpha = 10$, the model is reasonably capable of encoding the solution, as evidenced by metrics ranging from 10^{-3} to 10^{-2}. Nevertheless, it quickly starts to exhibit a noteworthy error, surpassing 10^{-2} and 10^{-1} for MAE and $MaxAE$, respectively, as $\alpha > 10$. The model is unable to learn a solution that is in a too stiff regime.

Transfer to Stiff Regime: *LBFGS Transfer Learning*

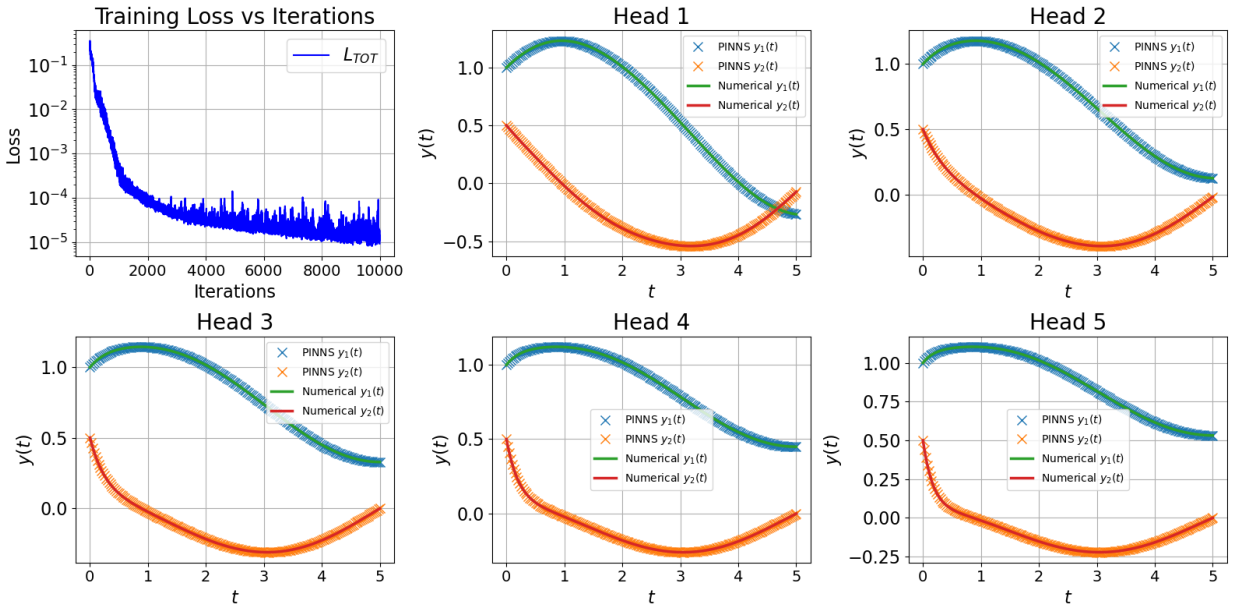

Figure 31: Training results for Equation 29 after 10000 iterations with $l_r = 10^{-5}$, using five heads, $\alpha \in \{2, 3, 4, 5, 6\}$ ($SR \cong \{4, 9, 16, 25, 36\}$. The first plot illustrates the loss vs. the number of iterations. The remaining five plots show the network learned solutions (in blue/in orange) versus the true solutions (in green/in red) for each head's equation configurations.

Figure 31 shows the results of training on Equation 29 in a non-stiff domain. The α of each head are $[2, 3, 4, 5, 6]$. The loss reaches around 10^{-4} and 10^{-5}. The network learned solutions, and numerical solutions align very closely across all five heads.

Now, we freeze the hidden layer of this trained model H^{fix} and fine tuned the last layer W in a stiffer regime with LBFGS..

- $\alpha = 10$ and $SR \cong 100$

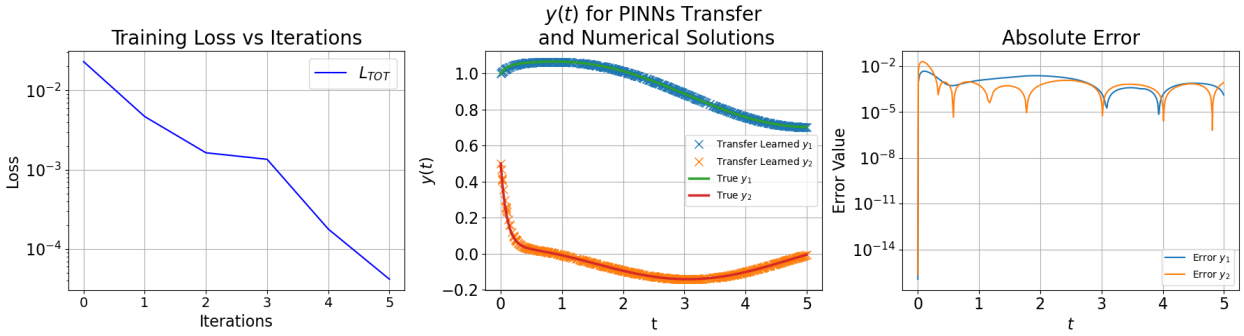

Figure 32: Transfer learning with LBFGS results for Equation 29 with $\alpha = 10$ after 5 iterations with $l_r = 1$

We observe that after 5 iterations, the loss value drops below 10^{-4}. Transfer learning works effectively, and the PINNs match the numerical method solutions. The error plot shows notably low values for the initial conditions due to the reparametrization. It reaches around 10^{-3} and 10^{-4} for both y_1 and y_2 over the remaining time scale.

- $\alpha = 30$ and $SR \cong 900$

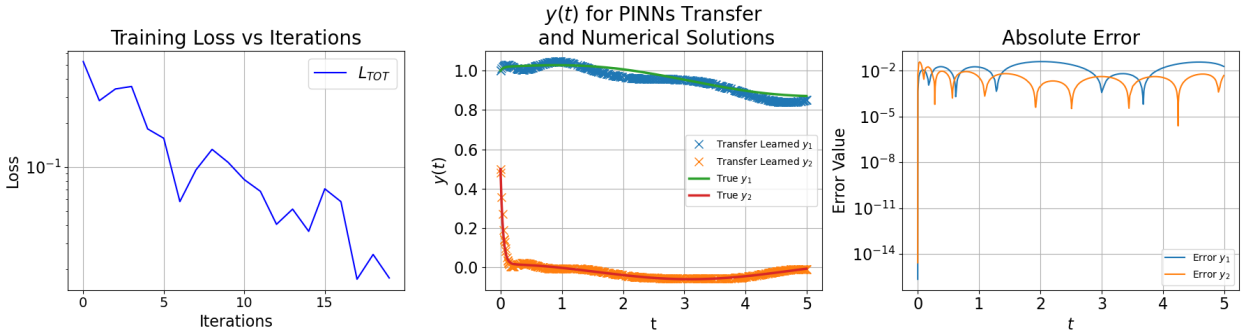

Figure 33: Transfer learning with LBFGS for Equation 29 with $\alpha = 30$ after 20 iterations with $l_r = 0.5$

We observe that after 20 iterations, the loss value does not even reach 10^{-2}. By examining PINNs, numerical solutions, and error plot, it is evident that the transfer learning does not work as before. The error of magnitude 10^{-2} is not negligible.

- $\alpha \in [5, 40]$

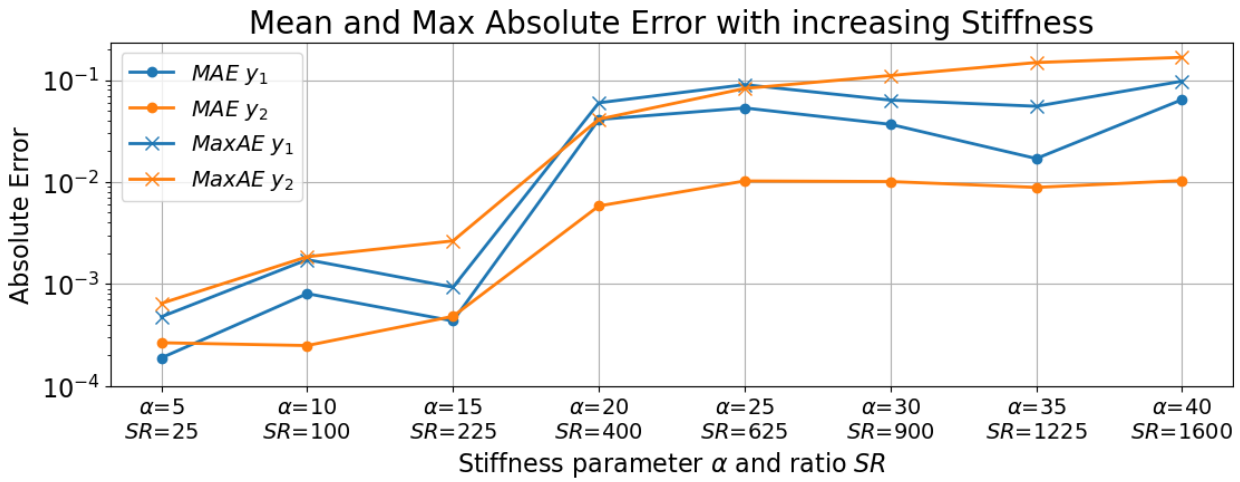

Figure 34: Mean Absolute Error and Maximum Absolute Error solving Equation 29 using LBFGS transfer learning as α increase from 5 to 40

In the least stiff regime $\alpha \in [5, 15]$, the LBFGS optimization demonstrated effective performance, as indicated by metrics ranging from 10^{-4} to 10^{-3}. The error magnitude observed in this regime is lower than that associated with learning the stiffness method in the stiff regime, when comparing with Figure 30. Nevertheless, it quickly starts to exhibit a noteworthy error, in the range of 10^{-2} and 10^{-1}, as $\alpha > 20$. The disparity between $MaxAE$ and MAE for y_2, coupled with the convergence of $MaxAE$ approaching 0.5, indicates a challenge to encode the transient phase requirements of this solution due to stiffness. The LBFGS optimization of the last layer is therefore unable to learn and to capture the stiffness in these regimes.

Transfer to the stiff regime: *Perturbation Approach*

After the expansion $y_1 \simeq \sum_{i=0}^{p} \beta^i X_i$ and $y_2 \simeq \sum_{i=0}^{p} \beta^i Y_i$ with $p \in \mathbb{N}$. The component of the $p+1$ systems outlined in Equation 25 for the Duffing Equation 29 are (see details in the Appendix):

$$A = \begin{bmatrix} 0 & -1 \\ 0.1 & \alpha \end{bmatrix}, \quad g = \begin{bmatrix} 0 \\ -y_1^3 \end{bmatrix}$$

$$F_i = \begin{cases} \begin{bmatrix} 0 \\ cos(t) \end{bmatrix} & \text{if } i = 0 \\ \begin{bmatrix} 0 \\ -\sum_{a+b+c=i-1}^{0<a,b,c<p} \phi(a,b,c) X_a X_b X_c \quad \text{with } \phi(a,b,c) = \begin{cases} 6 \text{ if } a \neq b \neq c \\ 1 \text{ if } a = b = c \\ 3 \text{ otherwise} \end{cases} \end{bmatrix} & \text{otherwise} \end{cases}$$

Following the first step of the approach, the training outcome of the linear form in a non-stiff regime is:

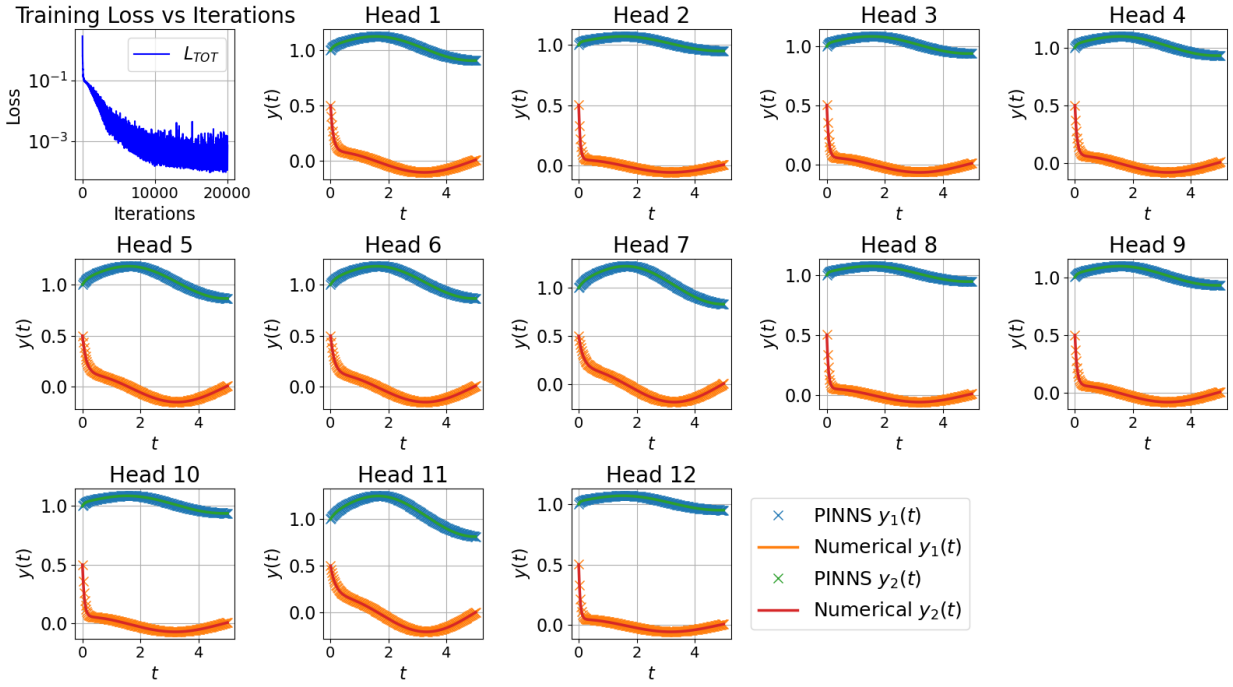

Figure 35: Training results for the linear form after 20000 iterations with $l_r = 10^{-5}$, using 12 heads, $\alpha \in [0, 10]$. The first plot illustrates the loss vs. the number of iterations. The remaining plots show the network learned solutions (in blue/in orange) versus the true solutions (in green/in red) for each head's equation configurations.

The loss reaches about 10^{-4}. The network learned solutions, and numerical solutions align very closely across all twelve heads.

We extracted H^{fix} of this training session. Then, we used it to solve iteratively the $p+1$ systems characterised by F_i and we reconstructed the final solution.

- $\alpha = 50$, $\beta = 0.5$, $p = 10$ and $SR \cong 2500$

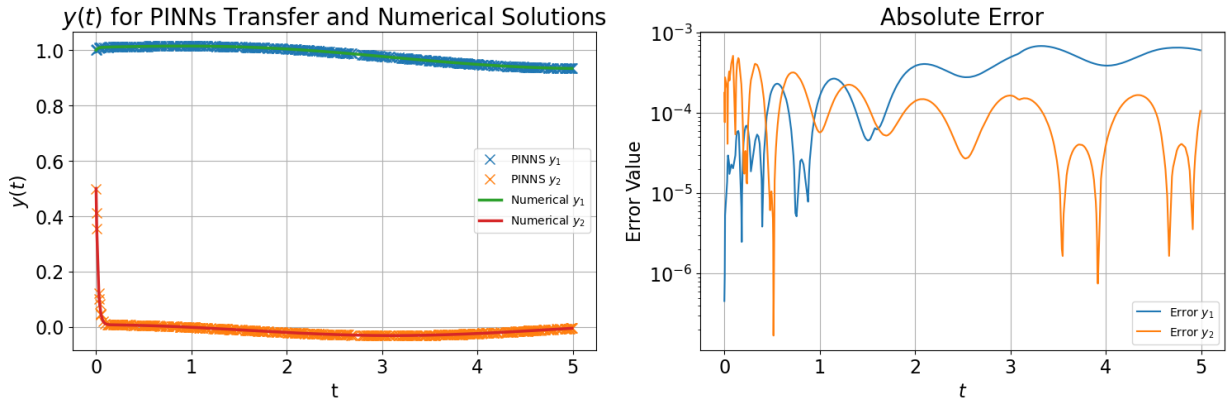

Figure 36: Transfer learning solution with perturbation for Equation 29 with $\alpha = 50$, $\beta = 0.5$ and $p = 10$. The left subplot shows the transfer learned solutions (in blue/in orange) versus the true solutions (in green/in red) and the right subplot shows the absolute error.

The obtained solution aligns well with the numerical solution, and the initial conditions are satisfied. The error plot reflects the absolute error between PINNs and numerical solutions, which is approximately between 10^{-3} and 10^{-5}.

- $\alpha = 100$, $\beta = 0.5$, $p = 10$ and $SR \cong 10000$

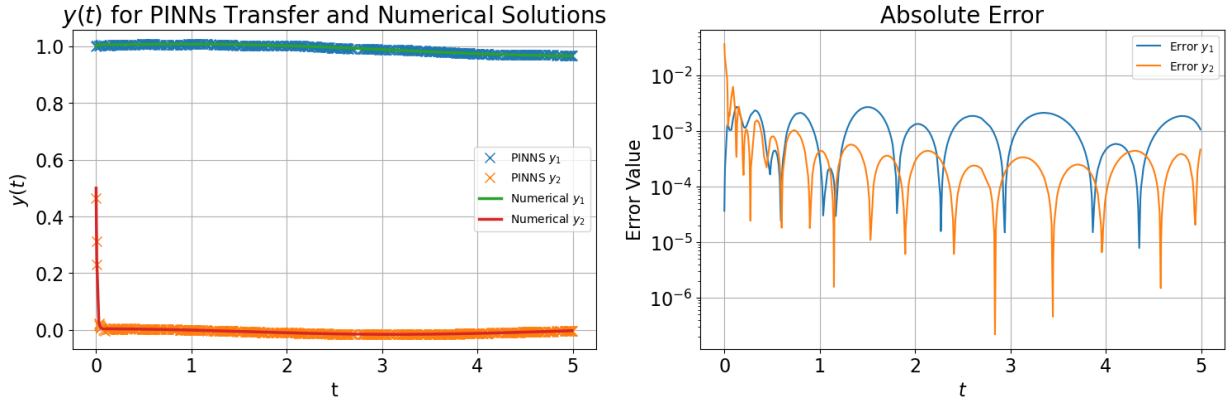

Figure 37: Transfer learning solution with perturbation for Equation 29 with $\alpha = 100$, $\beta = 0.5$ and $p = 10$. The left subplot shows the transfer learned solutions (in blue/in orange) versus the true solutions (in green/in red) and the right subplot shows the absolute error.

The obtained solution aligns well with the numerical solution but the initial conditions are not completely satisfied. The network struggles a bit in the time scale of the transient phase of y_2. The

error plot reflects the absolute error between PINNs and numerical solutions, which is approximately between 10^{-2} and 10^{-4}.

- $\alpha = 100$, $\beta \in [0, 1]$, $p \in [0, 40]$ and $SR \cong 10000$

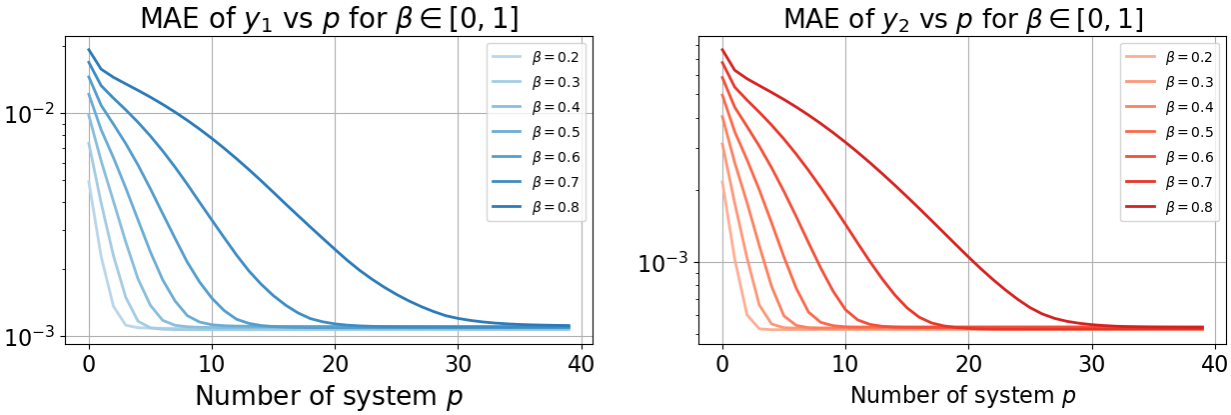

Figure 38: Mean Absolute Error comparison between PINNs and numerical solutions as $p \in [0, 40]$ increases, across different values of $\beta \in [0, 1]$

Analyzing the results depicted in Figure 38, the MAE comparison between PINNs and numerical solutions is presented. The variation in the number of systems to solve, denoted by p, is explored across different values of β and a fixed $\alpha = 100$. It is observed that as the non-linearity coefficient β increases, a higher number of systems is required. This phenomenon can be attributed to the heightened non-linearity associated with larger β values. Additionally, the coefficient is also used for perturbation, leading to larger powers of β when β is greater. In all cases, the MAE stabilizes around 10^{-3}, but the range of p varies from 3 to 30 for β values between 0.2 and 0.8.

- $\beta = 0.5$, $\alpha \in [10, 100]$, $p \in [0, 20]$

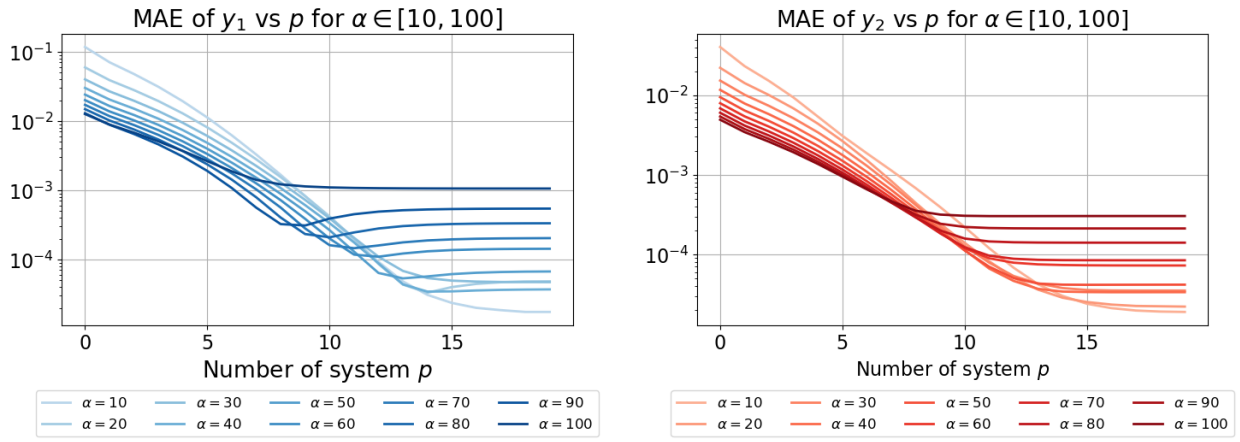

Figure 39: Mean Absolute Error comparison between PINNs and numerical solutions as $p \in [0, 20]$ increases, across different stiffness values of $\alpha \in [10, 100]$

The Figure 39 follows a similar concept to the previous one, but with a fixed $\beta = 0.5$. In this case, the number of systems to solve, denoted by p, increases for different stiffness levels $\alpha \in [10, 100]$. It is observed that as α increases, the error limit is reached faster. This can be explained by the dominance of the component with α over the non-linearity component. Additionally, the error limit becomes progressively smaller as α increases. It goes from 10^{-5} to 10^{-3} when α increases from 10 to 100. This phenomenon is attributed to the increased difficulty in solving the equation for the model as it becomes stiffer.

- $\beta = 0.5$ p fix, $\alpha \in [10, 150]$

Figure 40: Mean Absolute Error and Maximum Absolute Error solving Equation 29 using transfer learning by perturbation as α increases from 10 to 150

We can see on Figure 40 that from a general perspective, the error tends to increase with the growth in stiffness. The MAE converges within the range of 10^{-2} to 10^{-3} respectively for y_1

and y_2. It is important to acknowledge the challenge of encoding the transient phase within the time scale of the initial condition of y_2, as indicated by the convergence of $MaxAE$ to 0.5. When $\alpha < 100$, the $MaxAE$ of y_2 consistently remains below 10^{-2} so we can transfer without a noteworthy compromise in accuracy. This method demonstrates superior performance in comparison to the alternative method, notably exhibiting the capability to compute solutions within a stiffer regime than the one employed during training.

- Computational Time comparative studies with numerical methods: *increasing α stiffness parameter*

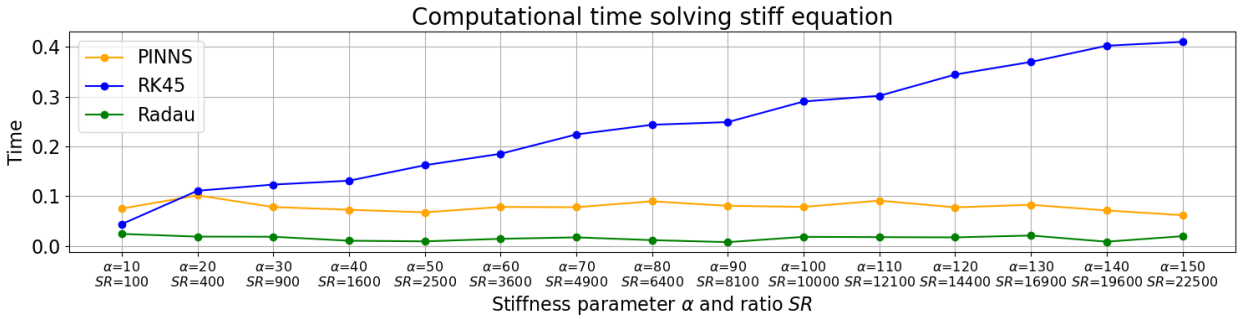

Figure 41: Comparison of computational time solving Equation 29 with PINNs ($p = 10$), RK45 and Radau with increasing stiffness

One can observe from Figure 41 that the computation time of RK45 increases significantly as the stiffness of the equation rises. This can be explained by its explicit nature. The PINNs ($p = 10$) is around 10^{-1} second (most of the computational time come from the matrix inversion). The Radau method outperformed the others. PINNs and Radau methods demonstrate independent computation times from the stiffness of the equation.

- Computational Time comparative studies with numerical methods: *change initial y_0 condition within a stiff domain*

Same as before, we trained a multi-head model under non-stiff regimes, each characterized by distinct initial conditions.

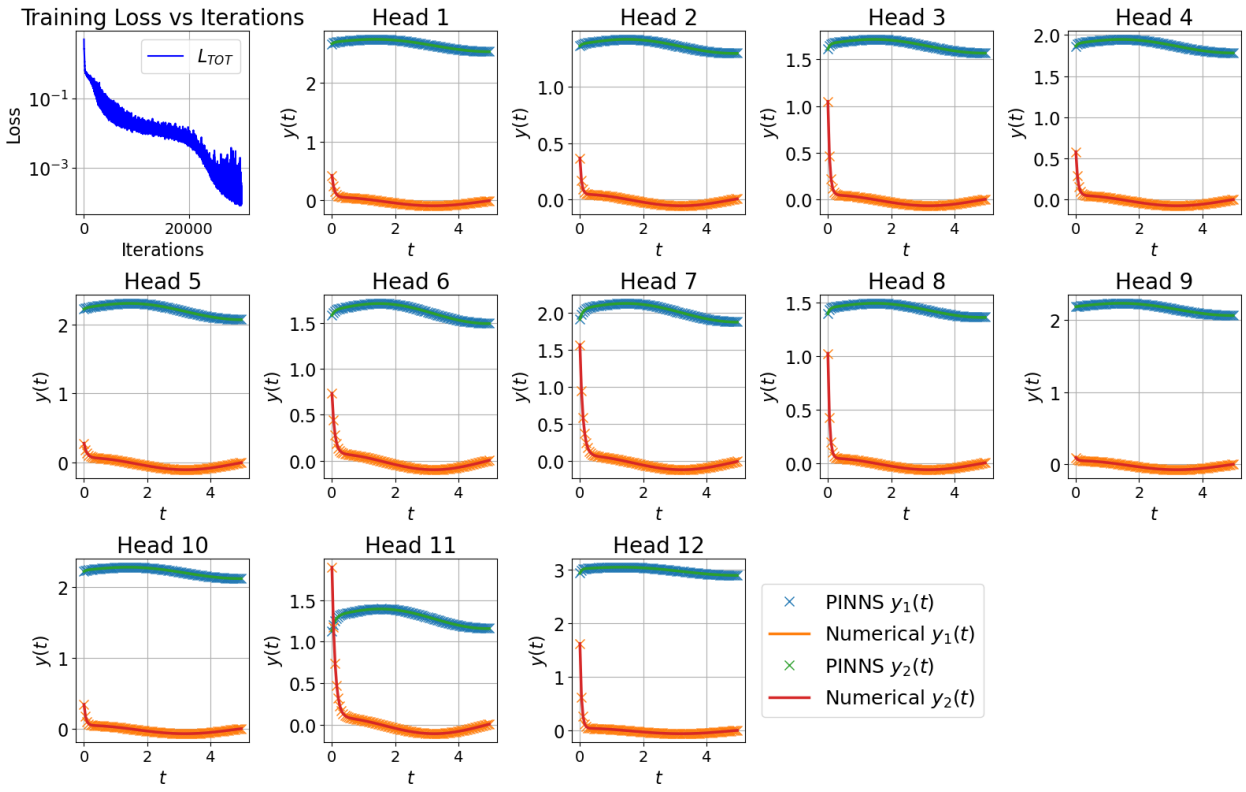

Figure 42: Training results for Equation 29 after 30000 iterations with $l_r = 10^{-4}$, using 12 heads, $\alpha \in [0, 20]$, $y_1(0) \in [1, 3]$ and $y_2(0) \in [0, 2]$

The figure 42 shows the training outcome of the multihead with varying initial conditions. The network training is successful, since the loss reach approximately 10^{-4}. The learned solutions closely match the numerical solutions across all 12 heads.

Now, we can extract the H_{fix} latent representation of this training session and compute M in a stiff regime according to equation 18. Then, we will compute several solutions with various initial conditions different that the training one. To solve the p systems using PINNs, there is no need to recalculate the M matrix. In Equation 25, only the force function F_i varies in each systems, and the M matrix remains independent of the force function (see Equation 18).

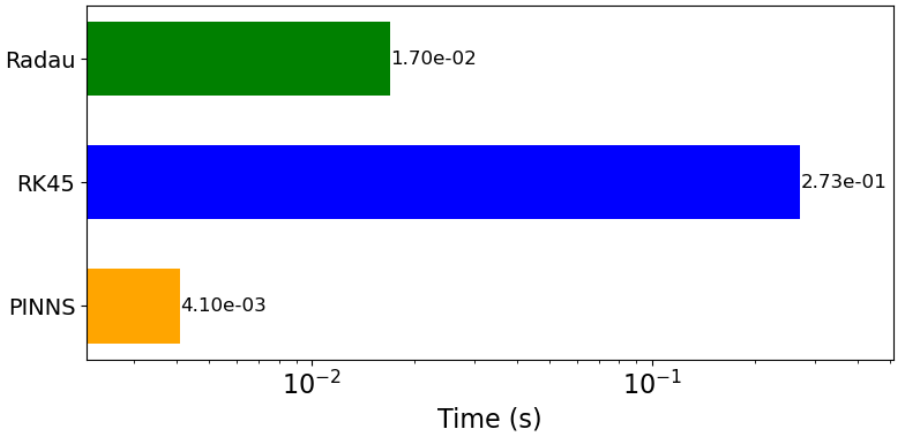

Figure 43: Average computational time solving 1000 Equations 29 in a stiff regime with $\alpha = 90$ ($p = 10$) for PINNs) by changing initial conditions at each iteration

The figure 43 shows that the transfer learning PINNs ($p = 10$) method was about 4 times faster than the Radau Method and about 70 times faster that RK45.

Discussion

The aim of this thesis was to develop a methodology that adapts the approaches outlined by Desai and al. to address stiff problems [15]. The original work demonstrated the capability to solve various differential equations and subsequently transfer learned solutions. This research extends these methods to address stiffness by using the stiffness parameter α. This value can be shifted to a non-stiff regime during training and then shifted to a stiffer regime when the transfer learning operates.

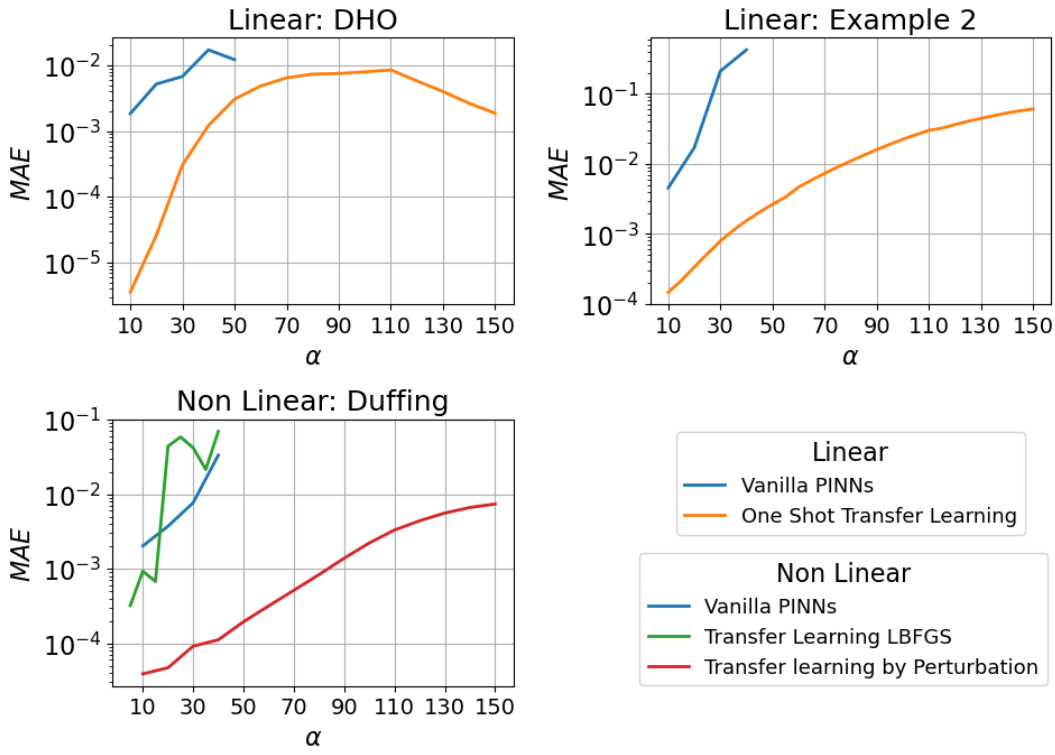

Figure 44: Comparison of each method's error with increasing stiffness parameter α. The error is represented by the mean of the Mean Absolute Error of y_1 and y_2 with the numerical solution. The first plot illustrates the result of the linear Equation 20, the second plot the one of the linear Equation 21 and the last one the result of the non-linear Equation 29

The results presented in Figure 44 are summarized in terms of MAE, indicating that our approach surpasses both vanilla PINNs and other optimization-based transfer learning methods.

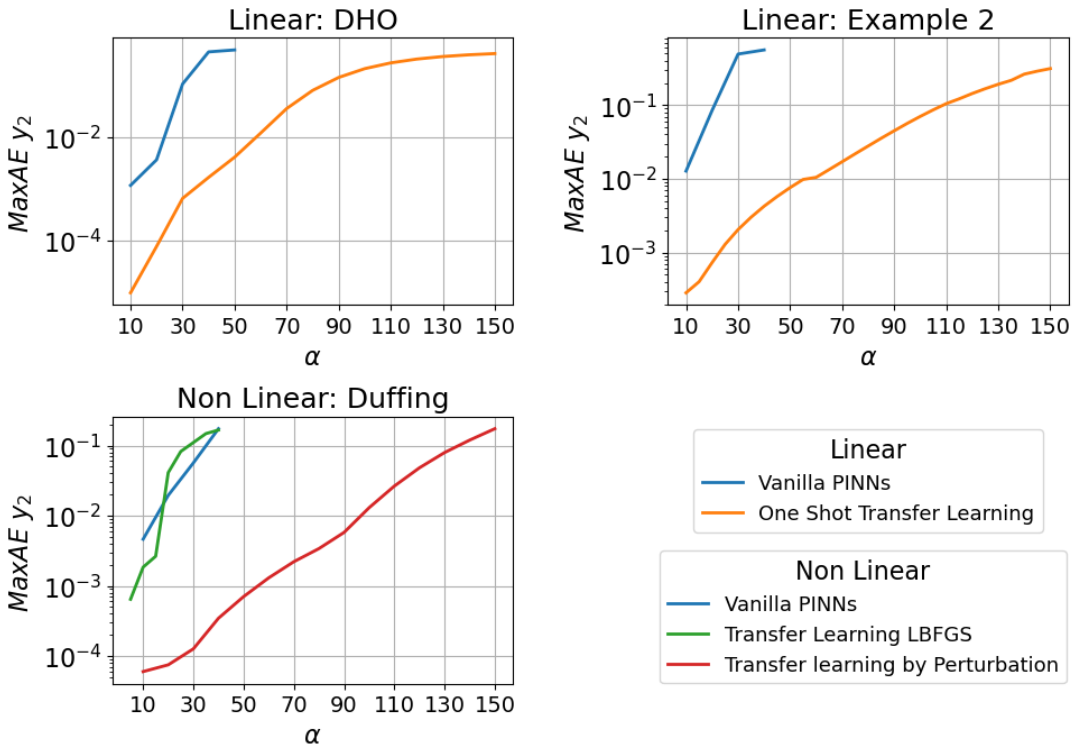

Figure 45: Comparison of each method's error with increasing stiffness parameter α. The error is represented by the Maximum Absolute Error of y_2. This metric catches the error in vicinity of the transient phase created by the stiffness. The first plot illustrates the result of the linear Equation 20, the second plot the one of the linear Equation 21 and the last one the result of the non-linear Equation 29

One significant challenge in addressing stiffness is the ability to encode the solution in the vicinity of the transient phase. Figure 45 illustrates that our method outperformed other PINNs-based approaches for encoding this behavior with a $MaxAE$ that is lower in stiff regimes. For this examples it is possible to transfer up to $\alpha = 60, 100, 110$ for training regimes of $\alpha \in [0, 10]$.
Comparative analysis with numerical methods, as depicted in Figures 15, 25, and 41, reveals that our solution's computational time remains independent of stiffness, offering competitive performance, but still a bit less than the Radau method. Furthermore, Figures 17, 27, and 43 highlight the superior computational efficiency of our methodology when changing initial conditions or force functions within a stiff domain.

The findings of this study indicated that the solution to a stiff problem can be computed by training the model in a less stiff or even a non-stiff regime. This observation holds significant implications, especially considering the challenges highlighted in prior research about training in a stiff domain [12].

Previous investigations [14, 13] have primarily focused on addressing stiff problems by retraining PINNs each time the stiffness regime, initial conditions, or force function change. Our results demonstrated the feasibility of obtaining the solution without retraining the model. Such an approach holds considerable value in terms of computational efficiency, making it competitive when compared to traditional numerical methods. Moreover, the computational time to get the solution is independent of the stiffness of the equation.
Furthermore, our results highlighted the potential for expedited computations, when changing initial conditions and force functions within a stiff domain. The efficiency gained through our approach surpasses that of conventional numerical methods. To illustrate this concept, we can consider a practical scenario with chemical kinetics, involving a stiff equation, such as the Robertson equation. The stiffness arises because of an important variation in the rate of change among the chemical components. Our approach could facilitate a rapid computation of the solution through different initial conditions within the stiff domain. This method could also be effective in initial condition inferring experiments.

A primary constraint of this work is that we cannot transfer to an infinite stiffness regime without sacrificing accuracy in the solution. Even if the jump in stiffness can be significant, it depends on the stiffness regime of the training process. However, the more we train in a stiff regime, the futher in the stiffness we will be able to transfer later.
Another notable limitation pertains to the applicability of these methods in the context of non-linear equations. Specifically, the perturbation expansion transfer learning method assumes the existence of a perturbation solution for the non-linear system. However, in many stiff problems such as the Van der Pol and Robertson equations, it becomes evident after experiments that no finite perturbation solution exists, necessitating an infinite degree of expansion (i.e., $p = \infty$).

There is potential for advancing the methodology by enhancing its capacity to train in stiffer domains. This could be achieved through the allocation of additional computational resources during the training process or by employing larger models. By increasing the stiffness limits during model training, it would be possible to transfer to even stiffer regimes. Investigating these avenues would provide valuable insights into the scalability of the proposed approach, in the realm of solving stiff problems.

Conclusion

This thesis examines the resolution of stiff ODEs using PINNs. The results confirm that the inherent characteristics of stiffness, particularly during the transient phase, pose a significant challenge for PINNs in accurately learning and encoding the solution. The study found that using a multi-head architecture to train PINNs in a low-stiffness regime allows the model to encode a general representation of the equation and stiffness-related behavior. This enables the model to calculate solutions in stiffer regimes by transfer learning without requiring re-training. This method can be extended to non-linear equations that have a perturbation solution. This approach's computational time is independent of the stiffness, but it is not as effective as numerical methods such as Radau for calculating solutions in regimes with varying stiffness, for the examples discussed in this work. However, our approach is more efficient than numerical methods when modifying initial conditions and force functions in a stiff domain.

Although our results include transfer-learned solutions that are far from the training stiffness regime, a more exhaustive training process would be beneficial. In other words, if computational resources permit, training many more heads in a stiffer training regime with a low learning rate could ensure that the latent space H captures an even more general representation of the stiffness problem's form. This would enhance scalability in transfer learning, allowing for the computation of solutions even in stiffer regimes.

It is worth noting that this is a relatively new area of research that does not have a well-established theory neither a vast number of contributions, that numerical methods provide, such as error bound or robustness. Namely, it is known that neural networks do not provide a guarantee in terms of training stability or convergence. Hence, for most real-world, critical applications where accuracy of solutions is paramount, our network-based solution should be used in conjunction with numerical approaches.

Based on these conclusions, future research could investigate stiff Partial Differential Equations, following the transfer learning approach of Wang et al. [12]. This extension could include stiff linear or non-linear Partial Differential Equations, such as the one-dimensional advection-reaction system [31], which is a stiff problem in the area of global tropospheric chemistry transport.

Emilien Seiler

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2016, pp. 779–788. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.91

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[4] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[5] X. Jin, S. Cai, H. Li, and G. E. Karniadakis, "Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations," *Journal of Computational Physics*, vol. 426, p. 109951, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999120307257

[6] K. Shah, P. Stiller, N. Hoffmann, and A. Cangi, "Physics-informed neural networks as solvers for the time-dependent schrödinger equation," 2022.

[7] T. Zhang, Y. Zhang, W. E, and Y. Ju, *DLODE: a deep learning-based ODE solver for chemistry kinetics*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2021-1139

[8] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*, 01 1996, vol. 14.

[9] Robertson, *The Solution of a Set of Reaction Rate Equations*, 1967.

[10] M. Girotti, "The van der pol oscillator," 2022. [Online]. Available: https://mathemanu.github.io/teach.html

[11] G. Dahlquist, *3 Years of Numerical Instability, Part I*, 1985.

[12] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient pathologies in physics-informed neural networks," 2020. [Online]. Available: https://arxiv.org/pdf/2001.04536.pdf

[13] W. Ji, W. Qiu, Z. Shi, S. Pan, and S. Deng, "Stiff-pinn: Physics-informed neural network for stiff chemical kinetics," *The Journal of Physical Chemistry A*, vol. 125, no. 36, p. 8098–8106, Aug. 2021. [Online]. Available: http://dx.doi.org/10.1021/acs.jpca.1c05102

[14] H. Baty, "Solving stiff ordinary differential equations using physics informed neural networks (pinns): simple recipes to improve training of vanilla-pinns," 2023.

[15] S. Desai, M. Mattheakis, H. Joy, P. Protopapas, and S. Roberts, "One-shot transfer learning of physics-informed neural networks," 2022.

[16] W. Lei, P. Protopapas, and J. Parikh, "One-shot transfer learning for nonlinear odes," 2023.

[17] C. F. . Curtiss and J. O. Hirschfelder, "Integration of stiff equations." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 38 3, pp. 235–43, 1952. [Online]. Available: https://api.semanticscholar.org/CorpusID:1577381

[18] *7. Ordinary Differential Equations*, pp. 185–234. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898717952.ch7

[19] L. F. Shampine and S. Thompson, "Stiff systems," *Scholarpedia*, vol. 2, no. 3, p. 2855, 2007, revision #139228.

[20] Lepik and H. Hein, *Stiff Equations*, 01 2014, pp. 45–57.

[21] J. D. Lambert, *Numerical Methods for Ordinary Differential Systems*, 1992.

[22] J. Butcher, "A history of runge-kutta methods," *Applied Numerical Mathematics*, vol. 20, no. 3, pp. 247–260, 1996. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0168927495001085

[23] E. Hairer and G. Wanner, "Stiff differential equations solved by radau methods," *Journal of Computational and Applied Mathematics*, vol. 111, no. 1, pp. 93–111, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037704279900134X

[24] D. J. Higham and L. N. Trefethen, "Stiffness of odes," *BIT Numerical Mathematics*, vol. 33, no. 2, pp. 285–303, Jun 1993. [Online]. Available: https://doi.org/10.1007/BF01989751

[25] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Dec 1989. [Online]. Available: https://doi.org/10.1007/BF02551274

[26] P. L. Lagaris, L. Tsoukalas, S. Safarkhani, and I. Lagaris, "Systematic construction of neural forms for solving partial differential equations inside rectangular domains, subject to initial, boundary and interface conditions," *International Journal on Artificial Intelligence Tools*, vol. 29, 05 2020.

[27] M. Pratama and A. Gunawan, "Exploring physics-informed neural networks for solving boundary layer problems," *Journal of Fundamental Mathematics and Applications (JFMA)*, vol. 6, no. 2, pp. 101–116, 2023.

[28] S. Roweis, "Matrix identities," 1999. [Online]. Available: https://cs.nyu.edu/~roweis/notes/matrixid.pdf

[29] M. G. Anne Kværnø, "Stiff differential equations," 11 2020. [Online]. Available: https://wiki.math.ntnu.no/_media/tma4130/2020h/stiffode.pdf

[30] A. S. Berahas, J. Nocedal, and M. Takac, "A multi-batch l-bfgs method for machine learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/8ebda540cbcc4d7336496819a46a1b68-Paper.pdf

[31] M. Santillana, L. Zhang, and R. Yantosca, "Estimating numerical errors due to operator splitting in global atmospheric chemistry models: Transport and chemistry,"

Journal of Computational Physics, vol. 305, pp. 372–386, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999115007299

Appendix

Derivation of the Stiffness Ratio of Equation 20 Damped Harmonic Oscillator

$$J = \begin{bmatrix} 0 & 1 \\ -1 & -\alpha \end{bmatrix}$$

$$p_J(\lambda) = det(A - \lambda I) = \lambda^2 + \alpha\lambda + 1$$

$$\lambda_{1,2} = \frac{-\alpha \pm \sqrt{\alpha^2 - 4}}{2}, \quad Re(\lambda_{1,2}) = \lambda_{1,2}$$

$$|\lambda_1| = \frac{\alpha - \sqrt{\alpha^2 - 4}}{2}, \quad |\lambda_2| = \frac{\alpha + \sqrt{\alpha^2 - 4}}{2}, \quad |\lambda_1| < |\lambda_2|$$

$$SR = \frac{|\lambda_2|}{|\lambda_1|} = \frac{\alpha + \sqrt{\alpha^2 - 4}}{\alpha - \sqrt{\alpha^2 - 4}} = \frac{(\alpha + \sqrt{\alpha^2 - 4})^2}{4} \propto \alpha^2$$

Derivation of the Stiffness Ratio of Equation 21

$$J = \begin{bmatrix} -2 & 1 \\ (\alpha - 1) & -\alpha \end{bmatrix}$$

$$p_J(\lambda) = \det(J - \lambda I) = \lambda^2 + (2 + \alpha)\lambda + \alpha + 1$$

$$\lambda_{1,2} = \frac{-(2 + \alpha) \pm \sqrt{(2 + \alpha)^2 - 4(\alpha + 1)}}{2} = \frac{-(2 + \alpha) \pm \sqrt{\alpha^2}}{2},$$

$$\mathrm{Re}(\lambda_{1,2}) = \lambda_{1,2} \, |\lambda_1| = 1, \quad |\lambda_2| = \alpha + 1, \quad |\lambda_1| < |\lambda_2|$$

$$\mathrm{SR} = \frac{|\lambda_2|}{|\lambda_1|} = \alpha + 1 \propto \alpha$$

Derivation of the Stiffness Ratio of Equation 29 Duffing

$$J = \begin{bmatrix} 0 & 1 \\ -0.1 - 3\beta y_1^2 & -\alpha \end{bmatrix}$$

$$p_J(\lambda) = det(A - \lambda I) = \lambda^2 + \alpha\lambda + 0.1 + 3\beta y_1^2$$

$$\Delta = \alpha^2 - 4/10 - 12\beta y_1^2, \quad \beta < 1,$$

$$\text{In stiff regime: } y_1(t) \simeq y_1(0) = 1$$

$$\Delta \simeq \alpha^2 - \frac{4}{10} - 12\beta$$

$$\lambda_{1,2} = \frac{-\alpha \pm \sqrt{\alpha^2 - \frac{4}{10} - 12\beta}}{2}, \quad Re(\lambda_{1,2}) = \lambda_{1,2}$$

$$|\lambda_1| = \frac{\alpha - \sqrt{\alpha^2 - \frac{4}{10} - 12\beta}}{2}, \quad |\lambda_2| = \frac{\alpha + \sqrt{\alpha^2 - \frac{4}{10} - 12\beta}}{2}, \quad |\lambda_1| < |\lambda_2|$$

$$SR = \frac{|\lambda_2|}{|\lambda_1|} = \frac{\alpha + \sqrt{\alpha^2 - \frac{4}{10} - 12\beta}}{\alpha - \sqrt{\alpha^2 - \frac{4}{10} - 12\beta}} = \frac{(\alpha + \sqrt{\alpha^2 - \frac{4}{10} - 12\beta})^2}{\frac{4}{10} + 12\beta} \propto \alpha^2$$

Derivation of the perturbation form of Duffing Equation 29

$$A = \begin{bmatrix} 0 & -1 \\ 0.1 & \alpha \end{bmatrix}, \quad g = \begin{bmatrix} 0 \\ -y_1^3 \end{bmatrix}, \quad f = \begin{bmatrix} 0 \\ cos(t) \end{bmatrix}$$

$$(\sum_{i=0}^{p} [\beta^i Y_i])^3 = \sum_{a=0}^{p} \sum_{b=0}^{p} \sum_{c=0}^{p} \beta^{a+b+c} Y_a Y_b Y_c$$

$$\beta^n \Rightarrow a + b + c = n$$

$$a = b = c \Rightarrow 1 \text{ options}$$

$$a = b \text{ or } a = c \text{ or } b = c \Rightarrow 3 \text{ options}$$

$$a \neq b \neq c \Rightarrow A_3^3 = 6 \text{ options of permutations}$$

$$\text{Power of } \beta^n : \sum_{a+b+c=n}^{0 \leq a,b,c \leq n} \phi(a,b,c) X_a X_b X_c \text{ with } \phi(a,b,c) = \begin{cases} 6 \text{ if } a \neq b \neq c \\ 1 \text{ if } a = b = c \\ 3 \text{ otherwise} \end{cases}$$

Then after $\forall i$:

$$F_i = \begin{cases} \begin{bmatrix} 0 \\ cos(t) \end{bmatrix} & \text{if } i = 0 \\ \begin{bmatrix} 0 \\ -\sum_{a+b+c=i-1}^{0\leq a,b,c\leq i-1} \phi(a,b,c) X_a X_b X_c \end{bmatrix} \text{ with } \phi(a,b,c) = \begin{cases} 6 \text{ if } a \neq b \neq c \\ 1 \text{ if } a = b = c \\ 3 \text{ otherwise} \end{cases} & \text{otherwise} \end{cases}$$