Friedrich-Alexander-Universität Erlangen-Nürnberg

Research Center for Mathematics of Data (FAU MoD)

and Chair of Dynamics, Numerics and Control (DCN)

Approximating the Wave Equation via Physics Informed Neural Networks: Various Forward and Inverse Problems

September 13, 2022

Internship Report

Dania Sana
(sanadania5@gmail.com)

Supervised by:
Dr. Yue Wang
(yue.wang@fau.de, FAU MoD & DCN)

Abstract

This report summarizes the research activities and results obtained during the internship in the Research Center for Mathematics of Data in Erlangen. We present our progress on the application of Physics-Informed Neural Networks (PINNs) to solve various forward and inverse problems in PDEs, where we take the well-understood 1 dimensional wave equation as an example for numerical experiment and error analysis.

For forward problems (computing numerical solutions of given systems), we first establish a general framework of PINNs- that are trained using the loss functions to respect the governing physic laws and match the initial and boundary conditions. This model is able to solve the forward initial boundary valued problem (IBVP), and has shown promising numerical results in different cases even in the extreme case of no data. The requirement of accurate and fast prediction of numerical solutions is studied in two approaches: 1) Analyzing and evaluating the performance of the PINNs model by comparing its output to the analysis solution and examining the training and validation error; 2) in parallel, we review various structure and size of NN (number of nodes and layers) and investigate the required computational time, in order to achieve a satisfying performance of our model. More complicated cases, e.g. mixed boundary conditions, or degenerating wave equations, can be solved in the same framework.

For inverse problems, we generalize the PINNs algorithm to calculate the boundary control to realize the null controllability of the system in a given finite time. Additionally, we combine the model-driven and data-driven approach in solving the parameter identification problem.

This proposed PINNs methodology is an elegant and flexible way to include physical knowledge in machine-learning algorithms and accelerate the approximation and accuracy when real data are used. The results and numerical experiments presented in this report should lead to not only a deep theoretical understanding (e.g. convergency and accuracy study) in the future, but also to a practical usage of PINNs framework to solve both forward and inverse physical problems from real world applications.

Key Words

physics-informed neural networks, machine learning, wave equation, boundary controllability, parameter identification

Contents

1 Introduction and Main Results

Accurate and fast predictions of numerical solutions are of significant interest in many areas of science and industry. On one hand, most theoretical methods used in the industry are the result of deriving differential equations that are based on conservation laws, physical principles, and/or phenomenological behaviors for a particular process. On the other hand, real-time capable methods and algorithms are required in modern industrial applications. In the view of this, very recently, Physics-Informed Neural Networks (PINNs) architectures, which combine the real data and the partial differential models (PDEs) together with a respective set of boundary and initial conditions, have started to arise frequently in many areas of science and engineering (see [17], [4], [2] and [9]) and emerged as a powerful tool for developing flexible PDE solvers (see NVIDIA [12] and DeepXDE toolbox [5]).

While the focus of other methods employing neural networks for solving PDEs is on mitigating the curse of dimensionality in high dimensions, the strength of PINNs lies in their flexibility in that they can be applied to a great variety of challenging PDEs, whereas classical numerical approximations typically require tailoring to the specifics of a particular PDE. e.g., strong nonlinearities, convection dominance or shocks.

The intrinsic Machine-Learning (ML) nature of PINNs enables huge speed-ups for simulations at inference time (i.e., after training) while the introduced physics ensures the conservation of the physical laws behind the problem. For the sake of simplicity, we take the wave equation as a basic model, the behavior of solution is well understood in theoretically and practically. Moreover, we mainly focus on revisiting the classical PINNs framework for solving forward and inverse problems, and on generating a solver in Python (see in our open-source on github: **github.com/DCN-FAU-AvH/PINNs_wave_equation**.)

The objectives and main results of this research internship is threefold.

- **First**, we revisit the classical PINNs framework for solving forward initial boundary valued problems, and build a solver in Python.
  - Establish the PINNs approach for solving the forward 1d wave equation with various boundary conditions
  - Evaluation on the performance of PINNs: Error analysis and Numerical experiment
  - Closer look into the computational time and accuracy by changing the size or structure of NNs.
  - Approximating the degenerating 1d wave equation with mixed boundary conditions via PINNs
- **Secondly**, we consider inverse problem addressed on the wave equation:
  - Following the work done in [6], we consider the null controllability of the wave equation via the PINNs and plot the desired boundary control numerically.
  - Furthermore, we solve the parameter identification problem for one unknown parameter (the speed of wave propagation) by a modified loss function, which measures the error on the real data set. In this way, we combine the physics-driven loss and the data-driven loss in this toy example.
- **Thirdly**, the numerical tools developed here are upload to the DCN-github as an open source for further research in theoretical and practical way. See in **github.com/DCN-FAU-AvH/PINNs_wave_equation**.

The main work is realized in Python programming language. To solve the forward problems we compiled the machine learning model of Physics-informed neural networks on two optimization algorithms "adam"

and "L-BFGS-B" respectively. We compare the performance of the ML models compiled using only "adam" algorithm and combined optimization algorithms ("adam"+"L-BFGS-B") respectively, by analyzing the resulting outputs of each of these models, on the training set and comparing them to the true outputs of this set. We investigate the performance of this PINNs framework on the training and validation set using the loss function of our setting on the training set and the summation of the square loss between the predicted and true output of a disjoint set from the training set respectively [16]. In order to solve the inverse problem we employ "adam" optimization algorithm only.

The layout of the report follows as such:

Section 2-3 are devoted to the detailed work and results obtained on each of the above objectives. In section 4, we describe the extension and perspective of the PINNs In section 5, we summarize the activities during the internship. And an overview of numerical results is listed in the Appendix.

2 Forward problems of 1d Wave Equation: Approximating solution via Physics-Informed Neural Networks

In this section, we introduce and revisit the classical PINNs framework for solving forward initial boundary valued problems of one dimensional wave equation:

$$y_{tt}(x,t) - a(x)y_{xx}(x,t) = 0, \quad t > 0, 0 \le x \le 1$$

with given initial data $(y, y_t)(0,x) = (y^0(x), y^1(x)), 0 \le x \le 1$. To be specific, in section 2.1 we take $a(x)$ as a positive constant function, which implies a classical linear wave equation with well-known analytic solution. While in section 2.4 we extend our work to the model with static degenerating, i.e. $a(x) = |x - x_0|^\alpha$ with positive constant α and fixed damage point $x_0 \in (0,1)$.

In the machine learning language, PINNs is composed of the following steps:

(1) design an artificial neural network $y(t,x;\theta)$ as a surrogate of the true solution $y(t,x)$.

(2) build a training set that is used to train the neural network

(3) define an appropriate loss function which accounts for residuals of the PDE, initial, boundary, and final conditions, and

(4) train the network by minimizing the cost function defined in the previous step.

The above algorithm for solving forward problems with various boundary conditions (Dirichlet or Neumann type) and its realization will be shown in Sections 2.1 and 2.3. Once the surrogate model of the original PDE system (i.e. PINN framework) is established, we evaluate the performance of PINNs by testing the error and computational time, and try to improve the quality of PINN by changing the architecture of the NN (for example: keep the size of NN (N nodes), then make a shallow net (N nodes on one layer) or deep net (several layers)) and also change the numbers of nodes in the NNs. The results are listed in the section 2.2.

2.1 PINN Algorithm for solving forward problems and its Realization.

2.1.1 Model: 1D wave equation with Dirichlet BC

Consider the following wave equation with the Dirichlet boundary conditions:

$$\begin{cases} y_{tt}(x,t) = 4y_{xx}(x,t), \quad 0 \le x \le 1,\ 0 < t < 2 \\ y(0,t) = 0, \quad 0 \le t \le 2 \\ y(1,t) = 0, \quad 0 \le t \le 2 \\ y(x,0) = sin(\pi x), \quad 0 < x < 1 \\ y_t(x,0) = 0, \quad 0 < x < 1 \end{cases} \tag{1}$$

The exact solution for $y(x,t)$ is given by $sin(\pi x)cos(2\pi t)$ [15]. In the following, we will introduce the PINN framework to compute the numerical solution to the system (1).

2.1.2 PINN algorithm and loss function

Neural Network. To obtain $y(x,t)$ that solves (1) through PINN, we chose the structure of the neural network to have two nodes in the input layer $(x,t) = (x_1, x_2)$ and one node in the output layer which contains the prediction for the value of $y(x,t) = y(x_1, x_2)$. There were four hidden layers included in the structure, where each of them had 50 nodes.

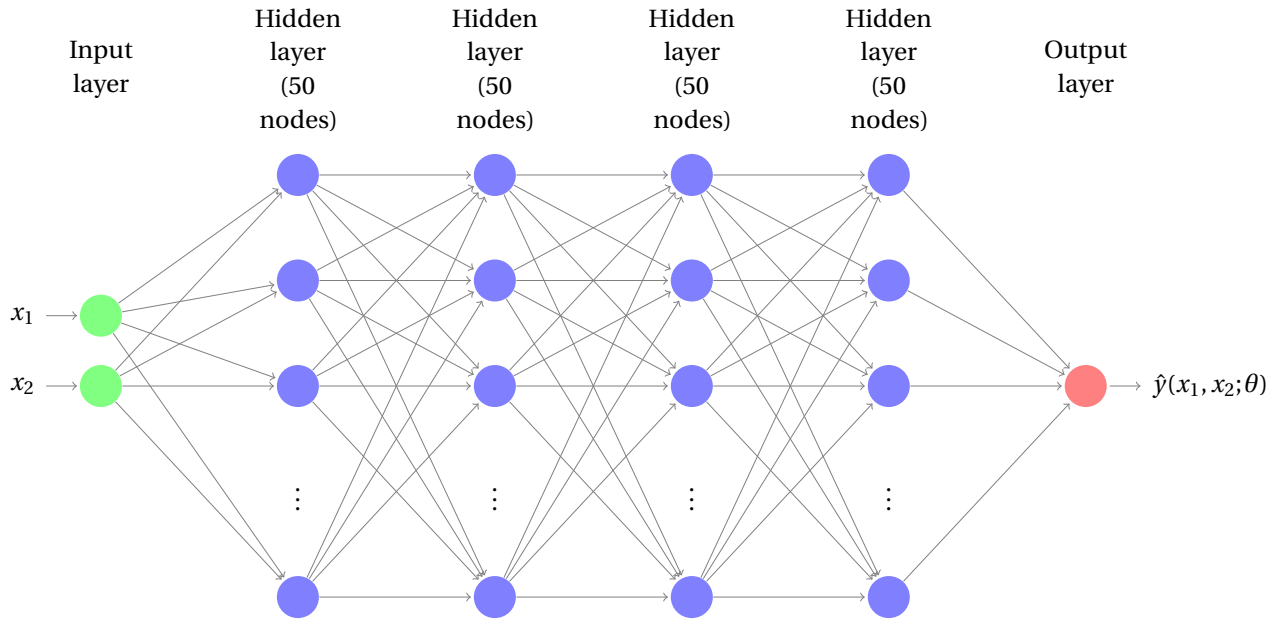


Figure 1: Neural Network structure

Here, we consider a deep feedforward neural network (DeepPDE) whose main goal is to approximate some function, in our case $y(x,t)$ for any input (x,t). This model is called feedforward because information flows through the function being evaluated from (x,t), through the intermediate computations used to define $y(x,t)$, and finally to the output $\hat{y}(x,t) \approx y(x,t)$. There are no feedback connections in which outputs of the model are fed back into itself [7].

The general structure of this neural network consists of $d+1$ input channels $\mathbf{x} = (x,t) \in \mathbb{R}^{d+1}$ and a scalar output $\hat{y}(x,t;\theta)$. In our case, $d = 1$.

To be specific, the solution $\hat{y}(x,t;\theta)$, which corresponds to the output of the neural network, is constructed as described in [6], mainly:

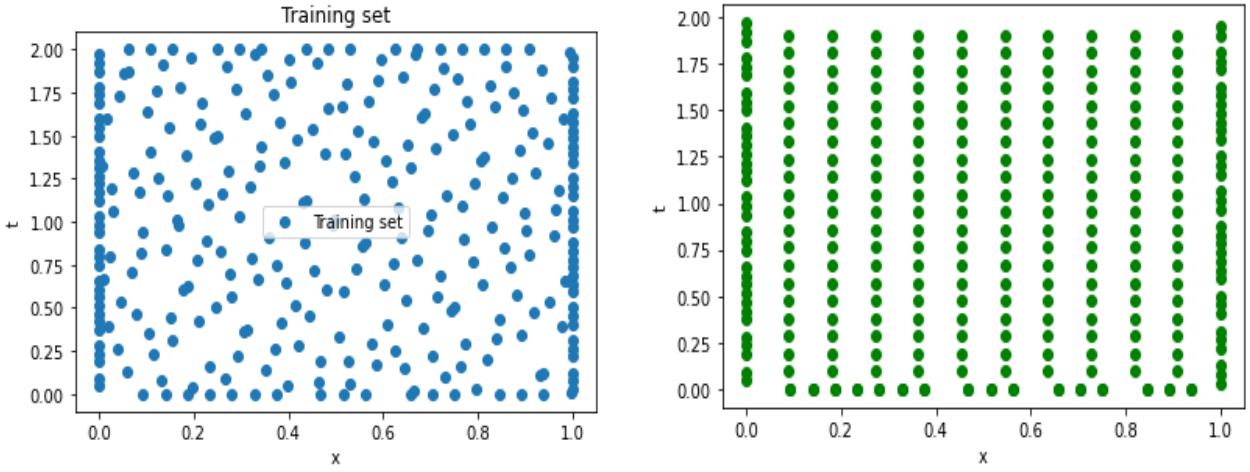
$$\begin{cases} \text{input layer:} \quad \mathcal{N}^0(\mathbf{x}) = \mathbf{x} = (x,t) \in \mathbb{R}^{d+1}, \\ \text{hidden layers:} \quad \mathcal{N}^\ell(\mathbf{x}) = \sigma(\boldsymbol{W}^\ell \mathcal{N}^{\ell-1}(\boldsymbol{x}) + \boldsymbol{b}^\ell) \in \mathbb{R}^{N_\ell}, \quad \ell = 1, ..., L-1, \text{ with } L = 4, \text{ and} \\ \text{output layer:} \quad \hat{y}(\boldsymbol{x};\theta) = \mathcal{N}^L(\boldsymbol{x}) = \boldsymbol{W}^L \mathcal{N}^{L-1}(\boldsymbol{x}) + \boldsymbol{b}^L \in \mathbb{R} \end{cases} \tag{2}$$

where:

- $\mathcal{N}^\ell$: $\mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ is the ℓ layer with N_ℓ nodes,
- $\boldsymbol{W}^\ell \in \mathbb{R}^{N_\ell \times \mathbb{N}_{\ell-1}}$ and $\boldsymbol{b}^\ell \in \mathbb{R}^{N_\ell}$ are the weights and the biases and $\boldsymbol{\theta} = \{\boldsymbol{W}^\ell, \boldsymbol{b}^\ell\}_{\ell=1,...,L=4}$ are the parameters of our neural network, and
- σ is an activation function which acts component-wise. In our implementation, we consider $\sigma(s) = tanh(s)$ for $s \in \mathbb{R}$.

Training dataset. The general training set τ of this model is selected in the interior domain $\tau_{int} \subset (0,1) \times (0,2)$ and on the boundaries $\tau_{x=0} \subset \{0\} \times [0,2]$, $\tau_{x=1} \subset \{1\} \times [0,2]$, $\tau_{t=0} \subset (0,1) \times \{0\}$. Thus $\tau = \tau_{int} \cup \tau_{x=0} \cup \tau_{x=1} \cup \tau_{t=0}$. The training set we used consisted of 300 samples $\{(x_i,t_i); y(x_i,t_i)\}_{i=1}^{300}$, where $y(x_k,t_k)$ is the solution of (1) at (x_k,t_k). 200 training samples were chosen from $(0,1) \times (0,2)$ and the rest was taken from the boundary of the domain. The following plot shows the training samples (x,t):

Loss function. The loss function we needed to minimize during the training process is expressed as the summation of the square of the difference pertaining to each of the equations in (1) that the prediction



(a) Training set

(b) Test set

Figure 2: Training dataset and Test dataset

should satisfy. This condition we pose on this model, enables us to find the optimal parameters of the neural network. In other words, given the input (x_i, t_i), let $y(x_i, t_i)$ and $\hat{y}(x_i, t_i)$ be the true solution of (1) and the output of the neural network respectively. The loss function used for training the NN with parameter θ is given by

$$L(\theta;\tau) = L_{int}(\theta;\tau_{int}) + L_{x=0}(\theta;\tau_{x=0}) + L_{x=1}(\theta;\tau_{x=1}) + L_{t=0}(\theta;\tau_{t=0}) + L^{par}_{t=0}(\theta;\tau_{t=0}), \tag{3}$$

where

$$L_{int}(\theta;\tau_{int}) = \frac{1}{|\tau_{int}|}\sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}_{tt}(x_i,t_i) - 4\hat{y}_{xx}(x_i,t_i))|^2, \tag{4}$$

$$L_{x=0}(\theta;\tau_{x=0}) = \frac{1}{|\tau_{x=0}|}\sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{y}(x_i,t_i)|^2, \tag{5}$$

$$L_{x=1}(\theta;\tau_{x=1}) = \frac{1}{|\tau_{x=1}|}\sum_{(x_i,t_i)\in\tau_{x=1}} |\hat{y}(x_i,t_i)|^2, \tag{6}$$

$$L_{t=0}(\theta;\tau_{t=0}) = \frac{1}{|\tau_{t=0}|}\sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}(x_i,t_i) - sin(\pi x_i)|^2, \tag{7}$$

$$L^{par}_{t=0}(\theta;\tau_{t=0}) = \frac{1}{|\tau_{t=0}|}\sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}_t(x_i,t_i)|^2. \tag{8}$$

Thus, the optimal parameters $\boldsymbol{\theta}^*$ of the neural network satisfy

$$\boldsymbol{\theta}^* = \underset{\theta}{\operatorname{argmin}}\ L(\theta;\tau). \tag{9}$$

2.1.3 Numerical realization and solution plotting

We apply the above loss function (3) on the training samples (part of the domain and boundary, see 2a), and we get the blue line in Figure 3, which implies the train loss function decreases w.r.t the training time.

At the same time, we calculate the loss function

$$\begin{aligned} L(\theta;\tau_{test}) = L_{test,int}(\theta;\tau_{test,int}) + L_{test,x=0}(\theta;\tau_{test,x=0}) + L_{test,x=1}(\theta;\tau_{test,x=1}) + \\ + L_{test,t=0}(\theta;\tau_{test,t=0}) + L^{par}_{test,t=0}(\theta;\tau_{test,t=0}) \end{aligned} \tag{10}$$

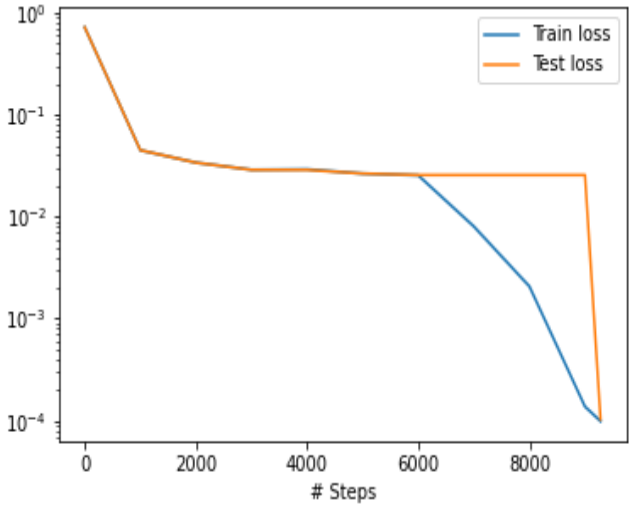


Figure 3: Train and test loss (by *adam* and *L-BFGS-B* optimization algorithms)

on the test samples. The testing set is denoted as Let τ_{test}=$\{(\hat{x}_k, \hat{t}_k); \hat{y}_k\}_{k=1}^{300} = \tau_{test,int} \cup \tau_{test,x=0} \cup \tau_{test,x=1} \cup \tau_{test,t=0}$, where $\tau_{test,int}$ and the other sets denote the test samples inside the domain and on the boundary respectively (see Figure 2b). The value of (10) is plotted in orange line shown in Figure 3.

The combined *adam* and *L-BFGS-B* optimization algorithms. The number of steps in the Figure 3 (known also as number of epochs) indicates the number of iterations used to train the model, thus the number of times the network weights are updated. We can clearly see in Figure 3 that the train loss decreases when the number of such iterations increases. Interestingly, we observe that the plot for the test loss does not follow this pattern. The test loss starts to decrease when increasing the number of steps from 0 to around 1000, then it increases slowly until it sharply drops. This drop happens because I combine two optimization algorithms *adam* [10] and *L-BFGS-B* (Limited-memory- Broyden–Fletcher–Goldfarb–Shanno algorithm) [18] respectively. The second one simulates fast convergence. **The combined *adam* and *L-BFGS-B* optimization algorithms are employed as the optimization algorithm to enhance both global search and local tuning.** The whole training process consists initially of 8000 iterations of the *adam* optimizer with 0.001 learning rate (LR) and iterations of *L-BFGS-B* optimizer until the loss converges to a small tolerance [14]. For smooth and regular solutions of a PDE system, the *L-BFGS* optimizer can find a better solution with a small number of iterations compared to the *adam* optimizer, due to second-order accuracy as opposed to *adam*, which is first-order accurate but in general more robust [8]. When we used only *adam* optimizer to compile the model, I did not obtain this stiff drop but a more graduate one. However, the performance of this model was worse. Figure 4 shows the behaviour of the train/test loss of the model when using only *adam* optimizer. Figure 5a shows the best trained result of the training samples (obtained in the step when the train and test loss have the smallest value) when using only *adam* optimizer. We can clearly see that this plot is far from predicting the true value of the training samples. Figure 5b however, shows the best prediction of the training samples when using both algorithms. This prediction seems much more reasonable and similar to the true solution in Figure 6.

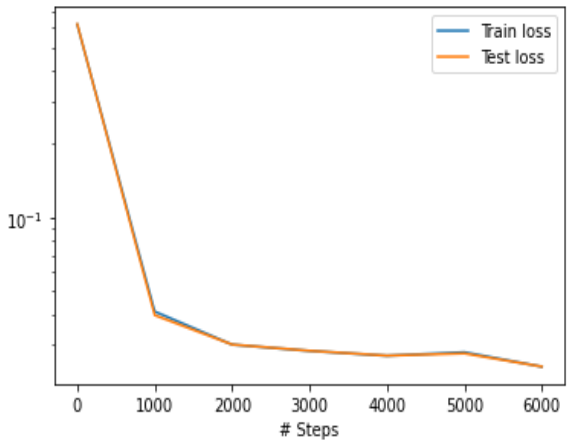
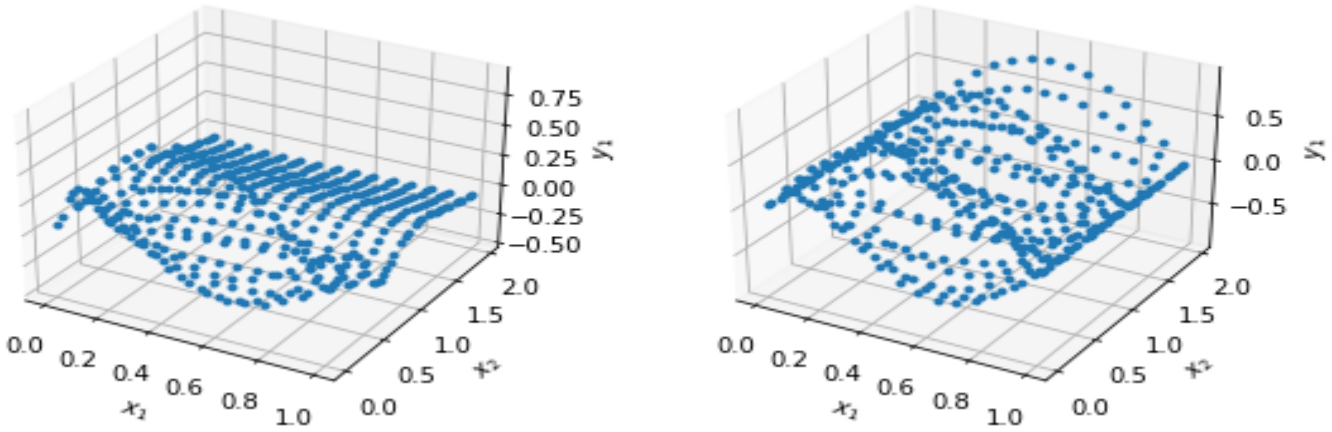


Figure 4: Train and test loss(adam optimizer only)



(a) Best trained result; adam optimizer only

(b) Best trained result; adam and L-BFGS-B optimizers

Figure 5: Best trained result by two kinds of algorithms

The solution plotting. The following plots show the exact solution and the PINN resulting solution of this wave equation:

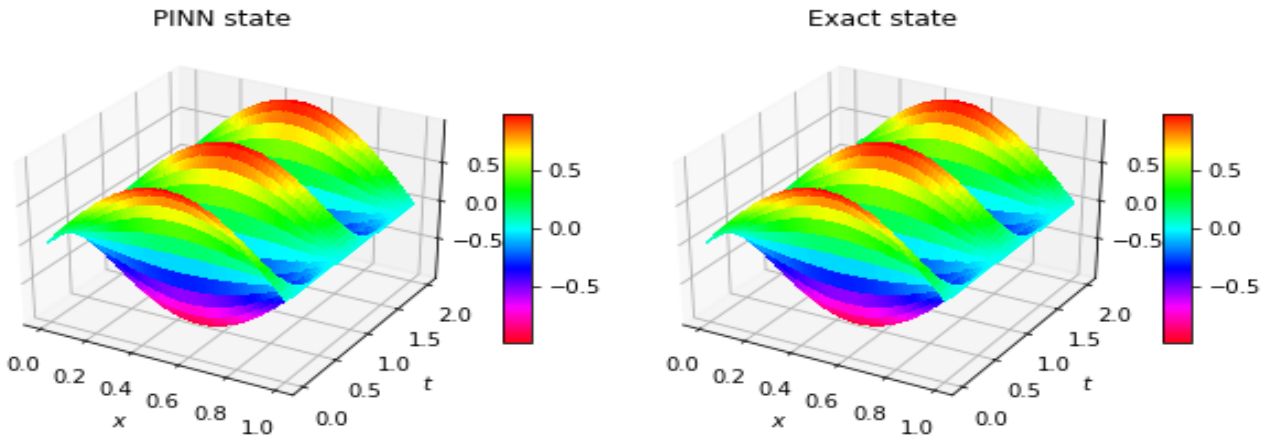


Figure 6: PINN and exact solution

Figure 7 displays the difference between the exact and PINN state. We see that the difference of these states equals mostly zero, which suggests a reasonable match between these two states.

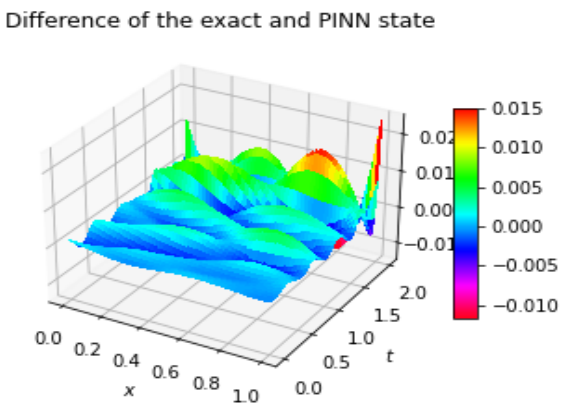


Figure 7: Difference between the PINN-predicted and exact solution

2.2 Evaluation on the performance of PINNs

2.2.1 Error analysis and computational time

The **training error** provides insight of how well the predicted outputs of the training inputs fit the training outputs, i.e., how the model performs in the training set. The training error is given by:

$$TE = L(\boldsymbol{\theta}^*; \tau),$$

where $L(\boldsymbol{\theta}^*; \tau)$ is defined in (3) with $\boldsymbol{\theta}^*$ as in (9). Figure 8 shows how the training error changes when increasing the number of the training samples. It shows that increasing the number of training samples increases the training error. This result is not a surprise because the more training samples, the more difficult is for the model to suit them all, which yields a greater training error.

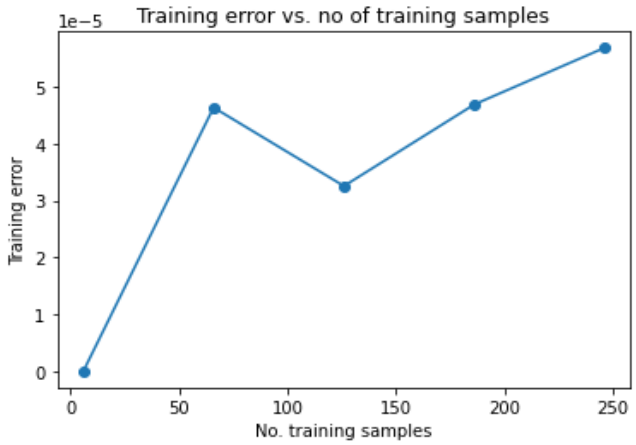


Figure 8: Training error

Error on a validation set, however, enables us to judge the performance of the machine learning model in a set of data other than the training set. Even if the model works very well on training data (the training error is negligible), this error is necessary to determine whether this model can be applied on any input data and still yield valid results. To find this error of our machine learning model (PINN), we used the validation set approach [16]. If $T = \{(x_i, t_i; y_i)\}$ denotes the set of all available data, this approach suggests that T is randomly split into two disjoint sets T_1 (training set) and T_2 (validation set). The error on the validation set is then given by:

$$EGE = \frac{1}{|T_2|} \sum_{(x_i,t_i;y_i)\in T_2} L(y_i, \hat{y}(x_i, t_i)) \tag{11}$$

$\hat{y}(x_i, t_i)$ is the value predicted by PINN for (x_i, t_i) and L is the square loss. Let us now see how this quantity changes when increasing the number of training samples:

Figure 9: Error on a validation set

Figure 9 shows that this error decreases initially when the number of training samples increases from 6 to 66. This should not be a surprise since in this way, the model learns more from the training samples and it generalizes better. However, we observe that the error is nearly 0 even for 66 training samples, which is a relatively small set. The error continues to be constantly negligible when the size of the training set increases.

Computational time. Increasing the number of training samples comes with computational costs. The code execution was extremely slow when considering a lot of training samples (it even exceeded one hour!). The following plot demonstrates this fact (time is given in seconds). We consider 8 different training samples with size 6, 12, 24, 48, 96, 192, 384, and 768 respectively. Our goal was to see how the computational time changes when increasing the size of the training set by a factor of two. We can see that the larger the difference between the size of training sets, the more compilation time is needed. One can see that the compilation time fails to increase by a factor of two. Moreover, the larger the difference between the size of the training sets, the more compilation time is needed.

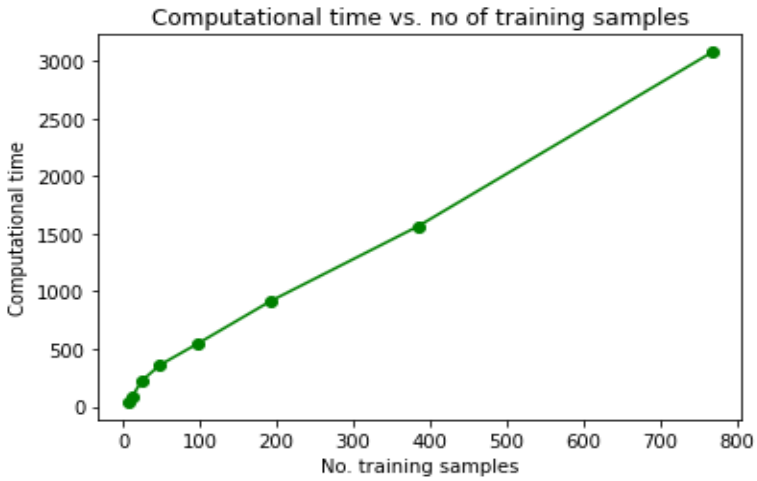


Figure 10: Computational time

Test error vs. computational time. The plot which shows the dependence of the test error from the computational time needed, enables us to analyze in more detail the performance of our machine learning

model. We aim to develop a model which can be executed in a reasonable time and performs well (test loss is small). We used 300 training samples, 250 testing samples (inside the domain) and 10000 epochs to obtain the Figure 11.

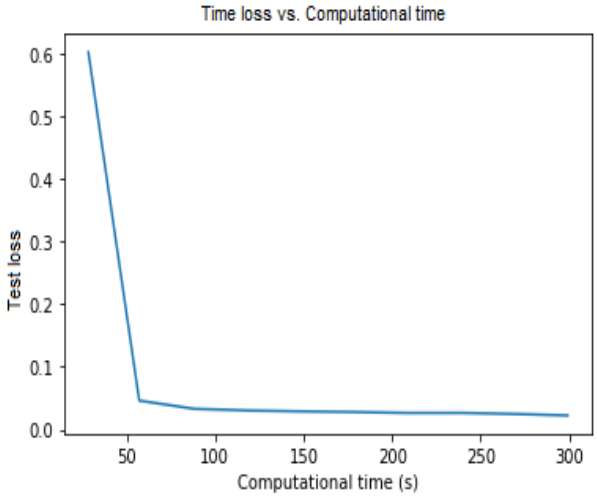


Figure 11: Test loss vs. computational time

Figure 11 shows that initially the decrease of the test loss is accompanied by an increase in computational time. However, we notice that this pattern disrupts and even though the model requires more time to be executed, the test loss appears to be nearly constant.

2.2.2 Discussion on the best structure of NNs.

We change the structure of the neural network to see how this influences the resulting performance of our model. The following results are obtained when the structure of the neural network is $4 \times [50]$ (four hidden layers with 50 nodes each), $2 \times [100]$ (two hidden layers with 100 nodes each), $10 \times [20]$ (10 hidden layers with 20 nodes each), and $4 \times [100]$ (4 hidden layers with 100 nodes each).

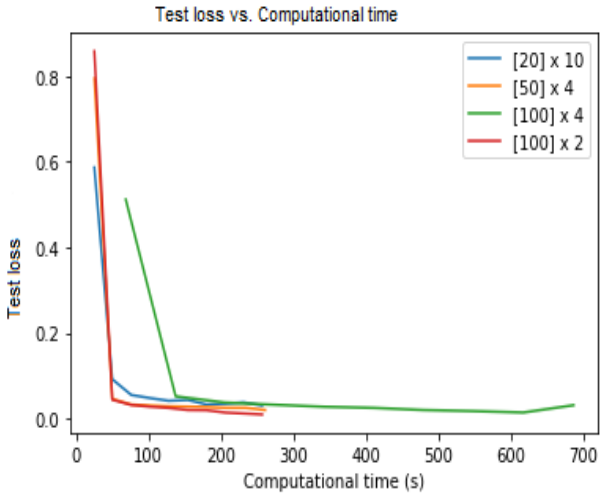


Figure 12: Test loss vs. computational loss

We can see from Figure 12 that the neural networks with the same number of nodes ($[20] \times 10$, $[100] \times 4$, $[50] \times 4$) follow almost the same pattern: the test loss sharply decreases when more computations are needed, until it reaches a constant negligible test loss even when the computational time increases. However, for

the neural network with 400 nodes ([100] × 4), the initial test loss is the smallest, but more time is required to obtain it. This should be intuitively clear since the neural network structure is more complex and thus more computations are required to acquire its optimal parameters. Nevertheless, we can see that the test loss reaches a constant low value which is the same as the one reached by the other neural network structures, One may claim that increasing the complexity of the structure of the neural network needs more computational time, however it does not necessarily improve the predictive performance of the model.

2.2.3 Discussion on the number of nodes in the neural network.

In order to determine how the number of nodes in the neural network influences the performance of our model, we consider eight different NN structures with 10, 30, 50, 80, 100, 150, 200, and 300 nodes respectively to compile the model and obtain the test loss. Figure 13 shows how the test error changes when increasing the number of iterations (i.e., the computational time) in these eight different NN settings.

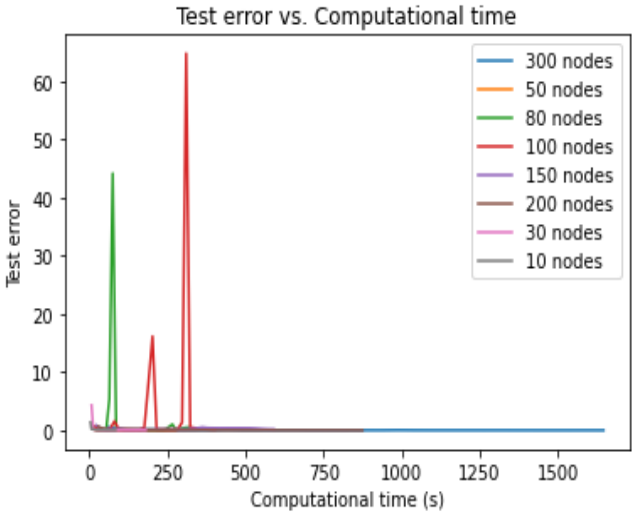
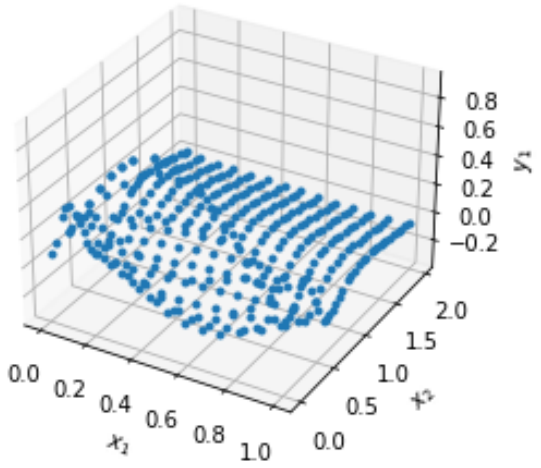
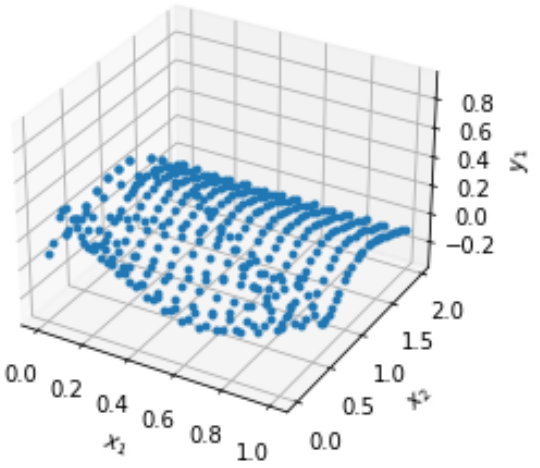


Figure 13: Test loss vs. computational time: differnt number of nodes

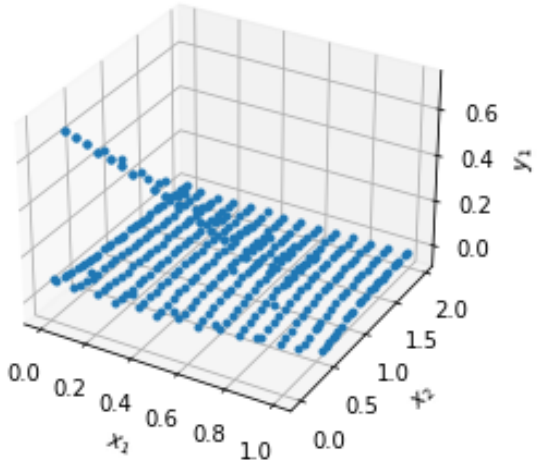
Our model performed very badly when the number of nodes was 80 or 100 because the test error does not converge to 0 but oscillates. We notice a better performance of the model, when we increase the number of nodes to be 200 and 300, because the test error of the model, after 30000 epochs reaches a constant negligible value. However, this improved performance, comes with computational costs. We clearly see that the NNs with 200 and 300 nodes require more time to be trained and compiled compared to the other chosen NNs structures. We can see how these NNs structures act on the training set through the following plots. Based on our previous observations, we would expect that NNs with number of nodes 200 and 300, yield better predictions of this set, letting the plot of the best trained results be more similar to the true outcome of the training set shown in 6.



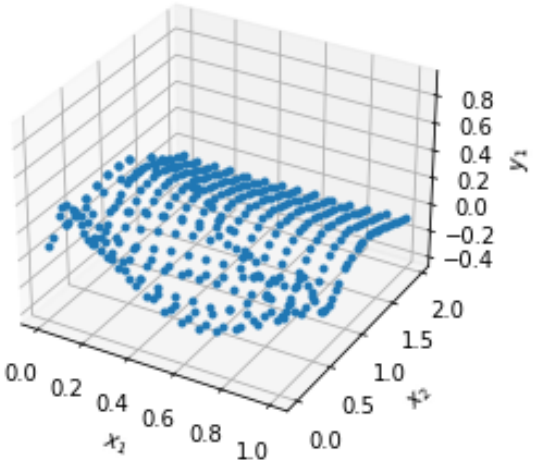
(a) Neural network structure: 10 nodes, 2 layers



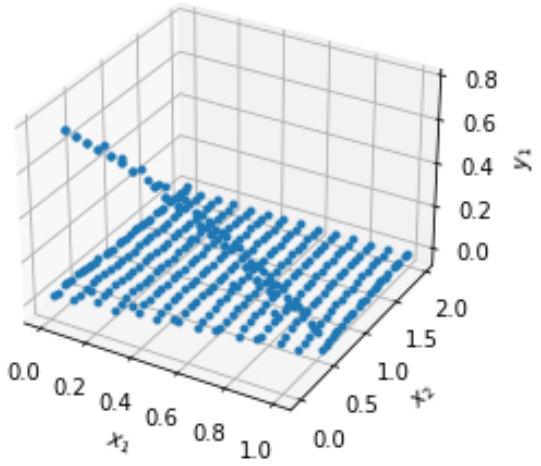
(b) Neural network structure: 30 nodes, 6 layers



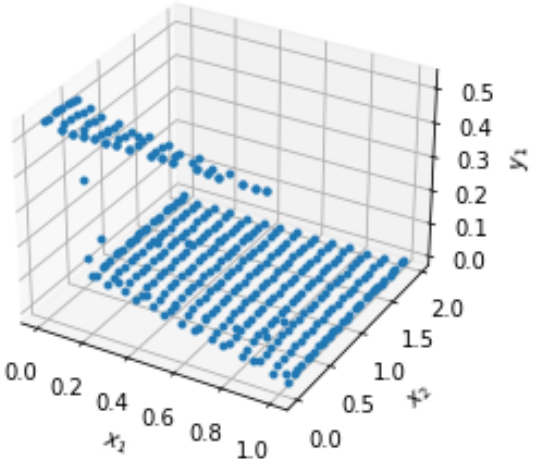
(c) Neural network structure: 50 nodes, 10 layers



(d) Neural network structure: 80 nodes, 10 layers

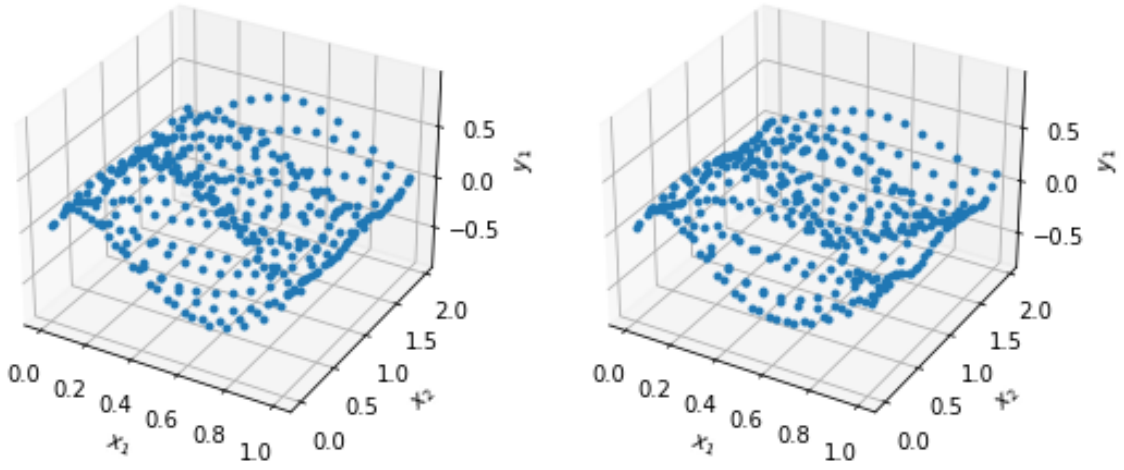


(e) Neural network structure: 100 nodes, 10 layers



(f) Neural network structure: 150 nodes, 15 layers

Figure 14: Best trained result: NN with 10, 30, 50, 80, 100, and 150 nodes



(a) Neural network structure: 200 nodes, 4 layers

(b) Neural network structure: 300 nodes, 6 layers

Figure 15: Best trained result: NN with 200 and 300 nodes

2.3 Extension: Approximating 1d wave equation with Neumann BC

Let us now consider the following wave equation with Neumann boundary conditions:

$$\begin{cases} y_{tt}(x,t) = 4y_{xx}(x,t), \quad 0 < x < 1,\ 0 < t < 2 \\ y_x(0,t) = 0, \quad 0 \le t \le 2 \\ y_x(1,t) = 0, \quad 0 \le t \le 2 \\ y(x,0) = sin(\pi x), \quad 0 < x < 1 \\ y_t(x,0) = 0, \quad 0 < x < 1 \end{cases} \tag{12}$$

For the given input (x,t), $y(x,t)$ that solves (12) is the output of the neural network equipped with two nodes in the input layer (x,t), one node in the output layer (value of $y(x,t)$) and four hidden layers where each of them has 50 nodes. We let the number of epochs of this model be 30000.

Training dataset. The general training set τ_1 of this model is selected in the interior domain $\tau_{int_1} \subset (0,1) \times (0,2)$ and on the boundaries $\tau_{x=0_1} \subset \{0\} \times [0,2]$, $\tau_{x=1_1} \subset \{1\} \times [0,2]$, $\tau_{t=0_1} \subset (0,1) \times \{0\}$. Thus $\tau = \tau_{int_1} \cup \tau_{x=0_1} \cup \tau_{x=1_1} \cup \tau_{t=0_1}$. The training set we used consisted of 800 samples $\{(x_i,t_i); y(x_i,t_i)\}_{i=1}^{800}$ where $y(x_k,t_k)$ is the solution of (1) at (x_k,t_k) found by a PDE solver `Python` offers. 500 training samples were chosen from $(0,1) \times (0,2)$ and the rest was taken from the boundary of the domain. The plot 16 shows the training samples $((x,t); y(x,t))$ we used.

Loss function. Similarly to the above example, the loss function is expressed as the summation of the square of the difference corresponding to each of the equations in (12). Namely, given input (x_i,t_i) let $\hat{y}(x_i,t_i)$ and θ be the prediction (output) and parameters of the neural network respectively. We acquire the following

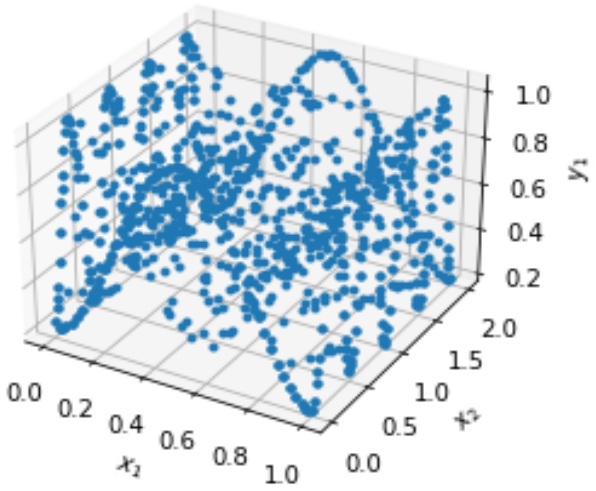


Figure 16: Best trained result

loss functions that we need to minimize:

$$L_{int_1}(\theta;\tau_{int_1}) = \frac{1}{|\tau_{int_1}|}\sum_{(x_i,t_i)\in\tau_{int_1}} |\hat{y}_{tt}(x_i,t_i) - 4\hat{y}_{xx}(x_i,t_i))|^2, \tag{13}$$

$$L_{x=0}(\theta;\tau_{x=0_1}) = \frac{1}{|\tau_{x=0_1}|}\sum_{(x_i,t_i)\in\tau_{x=0_1}} |\hat{y}_x(x_i,t_i)|^2, \tag{14}$$

$$L_{x=1_1}(\theta;\tau_{x=1_1}) = \frac{1}{|\tau_{x=1_1}|}\sum_{(x_i,t_i)\in\tau_{x=1_1}} |\hat{y}_x(x_i,t_i)|^2, \tag{15}$$

$$L_{t=0_1}(\theta;\tau_{t=0_1}) = \frac{1}{|\tau_{t=0_1}|}\sum_{(x_i,t_i)\in\tau_{t=0_1}} |\hat{y}(x_i,t_i) - sin(\pi x_i)|^2, \tag{16}$$

$$L^{par}_{t=0_1}(\theta;\tau_{t=0_1}) = \frac{1}{|\tau_{t=0_1}|}\sum_{(x_i,t_i)\in\tau_{t=0_1}} |\hat{y}_t(x_i,t_i)|^2. \tag{17}$$

The loss function used for training is:

$$L(\theta;\tau_1) = L_{int_1}(\theta;\tau_{int_1}) + L_{x=0_1}(\theta;\tau_{x=0_1}) + L_{x=1_1}(\theta;\tau_{x=1_1}) + L_{t=0_1}(\theta;\tau_{t=0_1}) + L^{par}_{t=0_1}(\theta;\tau_{t=0_1}) \tag{18}$$

The optimal parameters $\boldsymbol{\theta}^*$ of the neural network satisfy $\boldsymbol{\theta}^* = \arg\min_\theta L(\theta;\tau_1)$. The train and test loss of this model are shown in the figure 17, where we notice that after a certain number of steps, the value of the test loss drops sharply. As analyzed previously, this happens because we use *adam* and *L-BFGS-B* optimization algorithms where the second one simulates fast convergence.

The PINN predicted solution $y(x,t)$ is plotted in Figure 18.

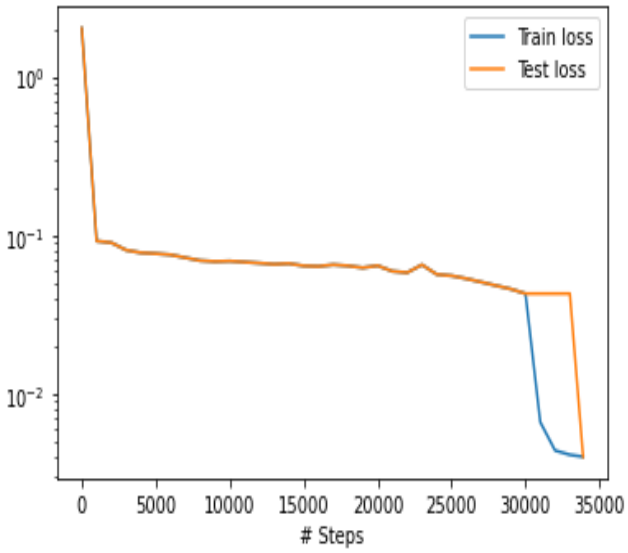


Figure 17: Train and test loss

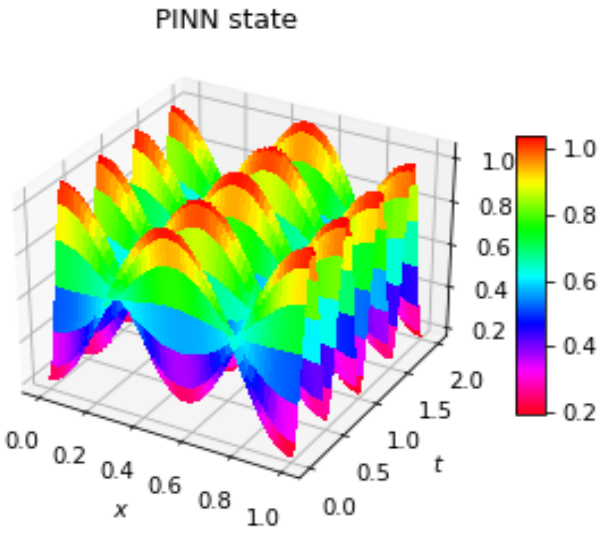


Figure 18: PINN-predicted solution to wave equation with Neumann boundary conditions

2.4 Approximating degenerating 1d wave equation $y_{tt} - a(x)y_{xx} = 0$ via PINN and FDM

In this section, we investigate the behavior of solution to the degenerating wave equation with mixed boundary conditions (Dirichlet and Neumann):

$$\begin{cases} y_{tt}(x,t) = a(x)y_{xx}(x,t), \quad 0 \le x \le 1,\ 0 < t < 2 \\ y(0,t) = 0, \quad 0 \le t \le 2 \\ y_x(1,t) = 0, \quad 0 \le t \le 2 \\ y(x,0) = \sin(\frac{\pi x}{2}), \quad 0 \le x \le 1 \\ y_t(x,0) = 0, \quad 0 \le x \le 1, \end{cases} \tag{19}$$

where the $a(x)$ contains a degeneration at fixed point $x = 0.5$. In this linear case, the static degeneration is provided by the a special type of coefficient $a(x) = |x - x_0|^\alpha, \alpha \ge 0$, where $\alpha = 0$ refers to no degeneration, while $\alpha = 1, \alpha = 2$ mark weak and strong degeneration, respectively (see [1],[11] and [3]). These three cases are investigate in the following 3 subsections, and we observe that in the strong degeneration case (i.e. $\alpha = 2$),

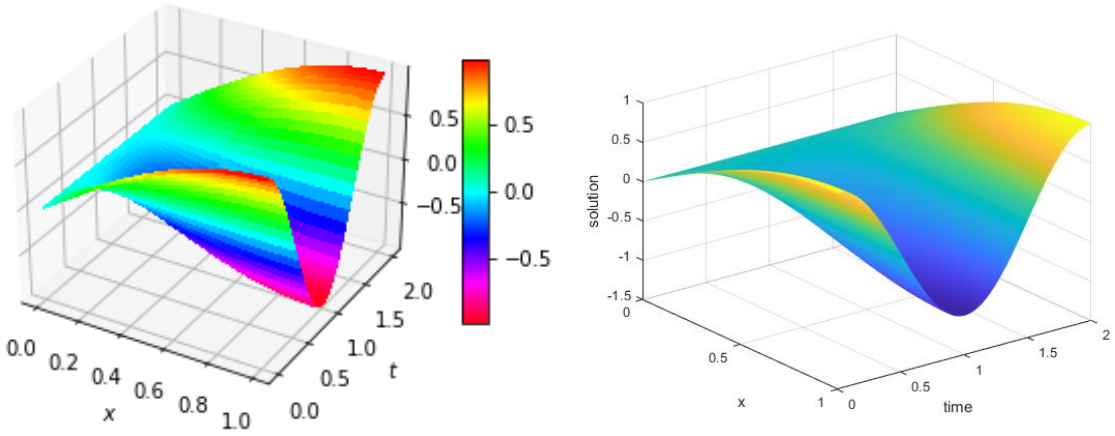
the solution is not wave-like crossing the degeneration point $x = 0.5$.

In the experiment, we choose the training set ($\tau = \tau_{int} \cup \tau_{x=0} \cup \tau_{t=0} \cup \tau_{t=2}$) to include the interior of the domain ($\tau_{int} \subset (0,1) \times (0,2)$) and the boundary of it ($\tau_{x=0} \subset \{0\} \times (0,2)$, $\tau_{t=0} \subset (0,1) \times \{0\}$). The loss function we need to minimize to be able to find the optimal parameters of the neural network is given as:

$$L = \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}_{tt}(x_i,t_i) - a(x_i)\hat{y}_{xx}(x_i,t_i))|^2 + \frac{1}{|\tau_{x=0}|} \sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{y}(x_i,t_i)|^2 + \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}_t(x_i,t_i)|^2 +$$
$$+ \frac{1}{|\tau_{x=1}|} \sum_{(x_i,t_i)\in\tau_{x=1}} |\hat{y}_x(x_i,t_i)|^2 + \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}(x_i,t_i) - sin(\pi x_i/2)|^2 + \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}(x_i,t_i) - y(x_i,t_i)|^2.$$

2.4.1 No degeneration: $a(x) \equiv 4$

We used 1400 uniformly distributed training samples (800 inside the domain and 600 on the boundary); we let the structure of NN be as in 1. Initially we use *adam* optimization algorithm (LR=0.001) for 7000 epochs and then L-BFGS-B until the train/test losses converge to a small tolerance. The output $\hat{y}(x,t)$ of the neural network which approximates the solution $y(x,t)$ of 19 is shown in Figure 19 (a). Figure 19 (b) shows the numerical solution $y(x,t)$ of 19 solved by FDM in Matlab. We can see that the PINNs output for $y(x,t)$ follows the same pattern as the compared numerical solution of $y(x,t)$, which means that our PINNs model is successful in solving (19).



(a) PINNs solution **(b)** FDM solution

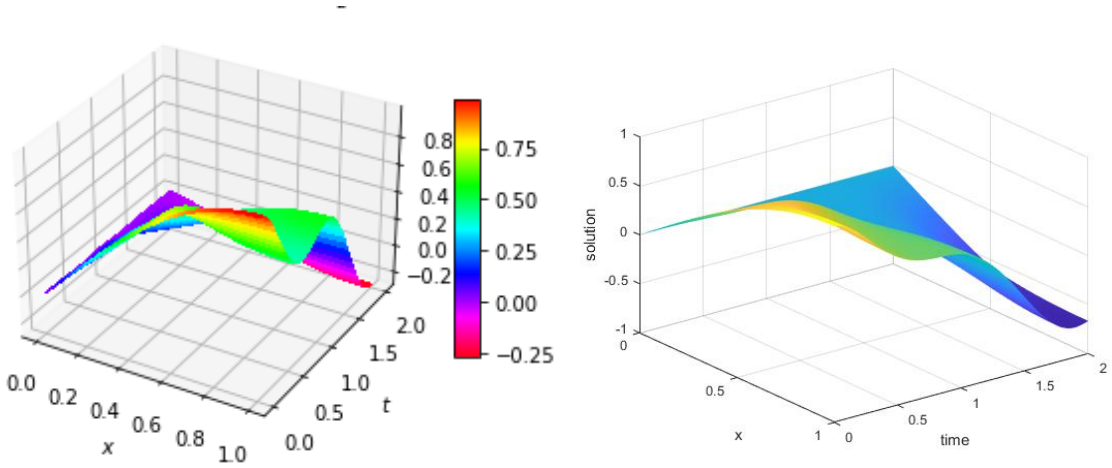
Figure 19: Solution of the degenerating wave equation: $a(x) \equiv 4$

2.4.2 Weak degeneration: $a(x) = 8|x - \frac{1}{2}|$

For this case, we used the same algorithms and NN structure but a higher number of training samples: 3000 uniformly distributed training samples respectively (2000 inside the domain and 1000 in the boundary of the domain). The PINNs solution $y(x,t)$ of 19 is displayed in Figure 20 (a). We can see that the pattern of this solution is very similar to the true value of $y(x,t)$, shown in Figure 20 (b).

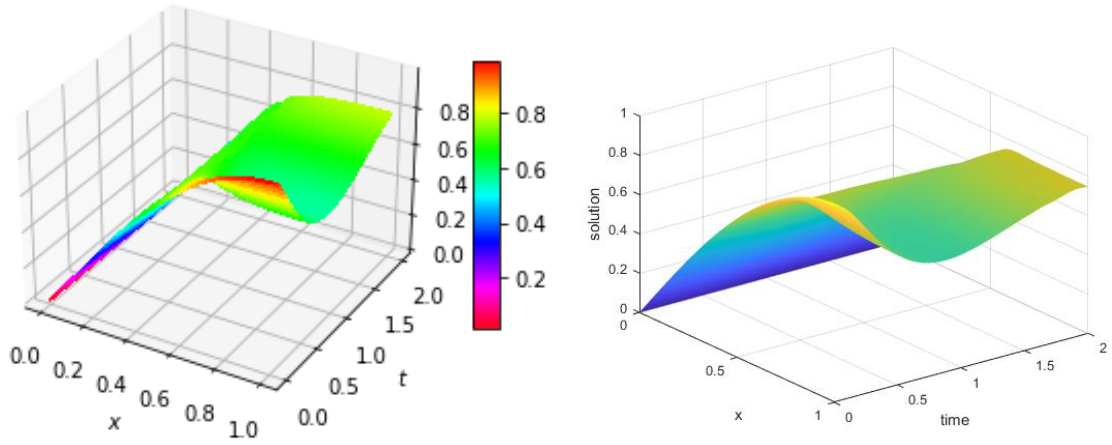
2.4.3 Strong degeneration: $a(x) = 16|x - \frac{1}{2}|^2$

We used 3000 uniformly distributed training samples (2000 inside the domain and 1000 in the boundary of the domain) and the same NN structure and optimization algorithms as in the previous example. The PINNs output that approximates the solution $y(x,t)$ of 19 is plotted in Figure 21 (a). We see that this solution follows almost the same pattern as the true solution $y(x,t)$ of 19, displayed in 21 (b).



(a) PINNs solution **(b)** FDM solution

Figure 20: Solution of the degenerating wave equation: $a(x) \equiv 8|x - \frac{1}{2}|$



(a) PINNs solution **(b)** FDM solution

Figure 21: Solution of the degenerating wave equation: $a(x) = 16|x - \frac{1}{2}|^2$

3 Inverse problems of 1d wave equations solved by PINNs

In this section, we invest the first step to solve inverse problems arising in PDEs. In section 3.1, we follow the work done in [6], and consider the null controllability of the wave equation via the PINNs and plot the desired boundary control numerically. The work done here can be easily extended to other types of PDEs. In section 3.2, we solve the parameter identification problem for one unknown parameter (the speed of wave propagation) by a modified loss function, which measures the error on the real data set. In this way, we combine the **physics-driven loss** and **the data-driven loss** in this toy example.

3.1 The null boundary controllability for wave equation

In this part, we ask: can we find a boundary control u to drive the solution from given initial data to final data (taken as 0)? Mathematically, you are facing the following null controllability problem (which is over-posed, you can not solve by classical numerical solver).

$$\begin{cases} y_{tt}(x,t) = 4y_{xx}(x,t), \quad 0<x<1,\ 0<t<4 \\ y(0,t) = 0, \quad 0 \le t \le 4 \\ y(1,t) = u(t), \quad 0 \le t \le 4 \\ y(x,0) = \sin(\pi x), \quad 0<x<1 \\ y_t(x,0) = 0, \quad 0<x<1 \\ y(x,T) = 0, \quad 0<x<1 \\ y_t(x,T) = 0, \quad 0<x<1 \end{cases} \tag{20}$$

3.1.1 The PINNs algorithm for solving controllability problem

We select the training set ($\tau = \tau_{int} \cup \tau_{x=0} \cup \tau_{t=0} \cup \tau_{t=4}$) to include the interior of the domain ($\tau_{int} \subset (0,1)\times(0,4)$) and the boundary of it ($\tau_{x=0} \subset \{0\}\times(0,4)$, $\tau_{t=0} \subset (0,1)\times\{0\}$, $\tau_{t=4} \subset (0,1)\times\{4\}$).

For the given controllability problem as (20), the loss function we need to minimize is given as:

$$L = \frac{1}{|\tau_{int}|}\sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}_{tt}(x_i,t_i) - 4\hat{y}_{xx}(x_i,t_i))|^2 + \frac{1}{|\tau_{x=0}|}\sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{y}(x_i,t_i)|^2 + \frac{1}{|\tau_{t=0}|}\sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}(x_i,t_i) - sin(\pi x_i)|^2 +$$
$$+ \frac{1}{|\tau_{t=0}|}\sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}_t(x_i,t_i)|^2 + \frac{1}{|\tau_{t=4}|}\sum_{(x_i,t_i)\in\tau_{t=4}} |\hat{y}(x_i,t_i)|^2 + \frac{1}{|\tau_{t=4}|}\sum_{(x_i,t_i)\in\tau_{t=4}} |\hat{y}_t(x_i,t_i)|^2.$$

To obtain PINNs output that approximates the solution $y(x,t)$ of the system 20, we used 1700 training samples (1000 in the interior of the domain and 700 in the boundary of it). We compiled the model under two algorithms *adam* (10000 epochs) and *L-BFGS-B*. The structure of the NN we utilized is displayed in 1.

3.1.2 Dirichlet control function u(t) traced by PINNs

We know that $u(t) = y(1,t)$ from 20. In order to plot $u(t)$, we select 40000 equidistant t values as t_i that lie in the segment $[0,4]$ and obtain the PINNs output for the inputs $(1,t_i)_{i=1}^{40000}$. We implement these mathematical steps in `Python` and obtain the Figure 23 representing the desired control function $u(t)$ to realize the null final state of the system. By this contrstuctive method, we can similarly design the Neumann or Robin controls.

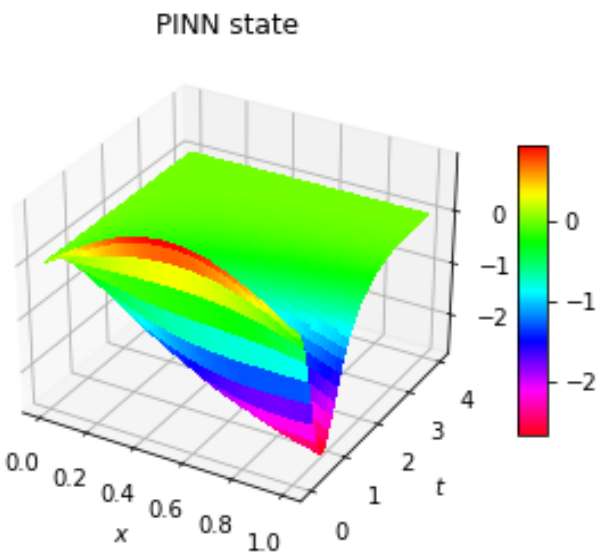


Figure 22: The PINNs solution $\hat{y}(x,t)$

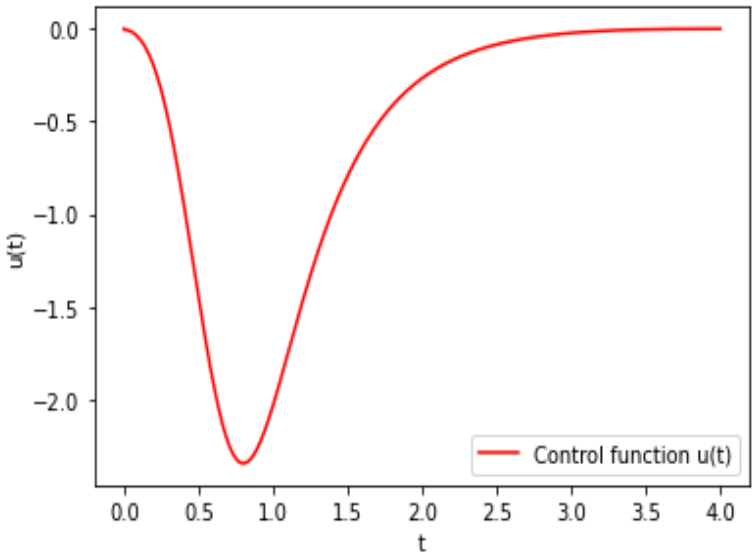


Figure 23: Control function $u(t) = y(1,t)$

3.2 Parameter identification problem

Physics-informed neural networks offer an effective approach to solve the inverse PDE problems, such as parameter Identification problem in different types of PDEs. Navier-Stokes equations are of particular importance in applied mathematics, since they they may be used to model the weather, ocean currents, water flow in a pipe and air flow around a wing. It has been shown that PINNs framework can correctly identify the unknown parameters of 2D-Navier-Stokes equation, even when the training data was corrupted with noise [13]. This result motivates us to use PINNs to discover λ that satisfies (1):

$$\begin{cases} y_{tt}(x,t) = \lambda y_{xx}(x,t), \quad 0 < x < 1, \, 0 < t < 2 \\ y(0,t) = 0, \quad 0 \le t \le 2 \\ y(1,t) = 0, \quad 0 \le t \le 2 \\ y(x,0) = sin(\pi x), \quad 0 < x < 1 \\ y_t(x,0) = 0, \quad 0 < x < 1 \end{cases} \tag{21}$$

3.2.1 Discovering parameter in wave equation via PINN: the physics-driven loss and the data-driven loss

We selected 1800 training points 1000 of which were uniformly distributed inside the domain $(0,1) \times (0,2)$. We chose 200 equally distanced points at each of the boundaries x=0, x=1, t=0, and t=2 of the domain. Let τ_{int} and $\tau_{x=0} \cup \tau_{x=1} \cup \tau_{t=0} \cup \tau_{t=2}$ be the training sets inside the domain and in the boundary of the domain respectively. We set the initial value of λ to be 1. The neural network, builds upon this fact and in our implementation, we consider 6000 epochs (known as the number of times the network weights are updated). Let $\hat{y}(x_i, t_i)$ be the output of the neural network with the given input (x_i, t_i). We choose the structure of our neural network be as in 1 in all the following implementations. The *adam optimization* algorithm finds the optimal parameters of the neural network with the available value of λ. We consider λ to change at every epoch and the model finds the optimal value of it, with the available NN parameters, that minimizes the loss function given below:

$$\mathrm{L}=\frac{1}{|\tau_{int}|}\sum_{(x_i,t_i)\in\tau_{int}}|\hat{y}_{tt}(x_i,t_i)-\lambda\hat{y}_{xx}(x_i,t_i))|^2+\frac{1}{|\tau_{x=0}|}\sum_{(x_i,t_i)\in\tau_{x=0}}|\hat{y}(x_i,t_i)|^2+\frac{1}{|\tau_{t=0}|}\sum_{(x_i,t_i)\in\tau_{t=0}}|\hat{y}_t(x_i,t_i)|^2+$$
$$+\frac{1}{|\tau_{x=1}|}\sum_{(x_i,t_i)\in\tau_{x=1}}|\hat{y}(x_i,t_i)|^2+\frac{1}{|\tau_{t=0}|}\sum_{(x_i,t_i)\in\tau_{t=0}}|\hat{y}(x_i,t_i)-sin(\pi x_i)|^2+\frac{1}{|\tau_{int}|}\sum_{(x_i,t_i)\in\tau_{int}}|\hat{y}(x_i,t_i)-y(x_i,t_i)|^2.$$

This initial value of λ even though small, enables that the λ predictions from the model nearly reach its true value after 3000 epochs.

Even though we chose a very small initial value of λ, the model was successful in predicting the true value of λ after exactly 3000 epochs illustrated in Figure 24.

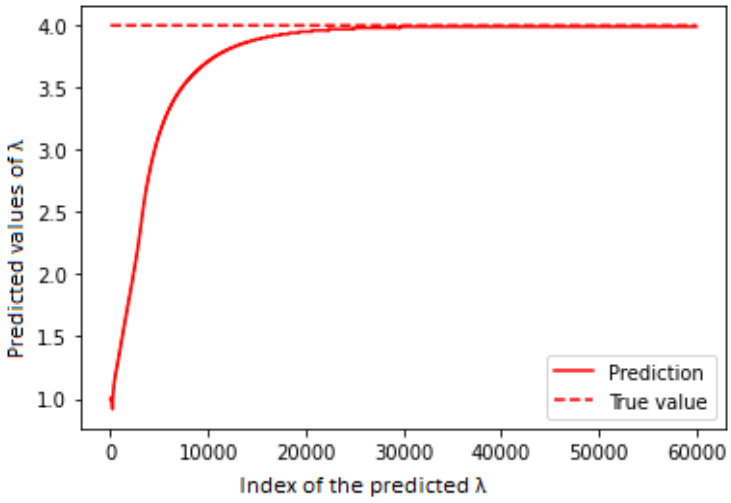


Figure 24: Predicted vs. true values of λ ($\lambda_0 = 1$; weights of the loss=[1, 1, 1, 1, 1, 1])

The solution of the inverse problem generates the best found value of λ to be 3.99. We aim to see how this new model with this λ predicts the input data and compare it to the exact solution of our wave equation in 1.

To better interpret the accuracy of the model let us observe the difference between these two states:

Figure 26 suggests that the performance of our model is acceptable, by showing that the difference between the two states in Figure 25 is included in the interval [-0.02, 0.02] units. We clearly see from Figure 26 that only small regions of the difference amount to 0.02 or -0.02. We conclude that our developed machine learning model performs well with a negligible difference between the exact and PINNs predicted states.
It is important to see how our model performs on the training input and compare it to the true result. Figure

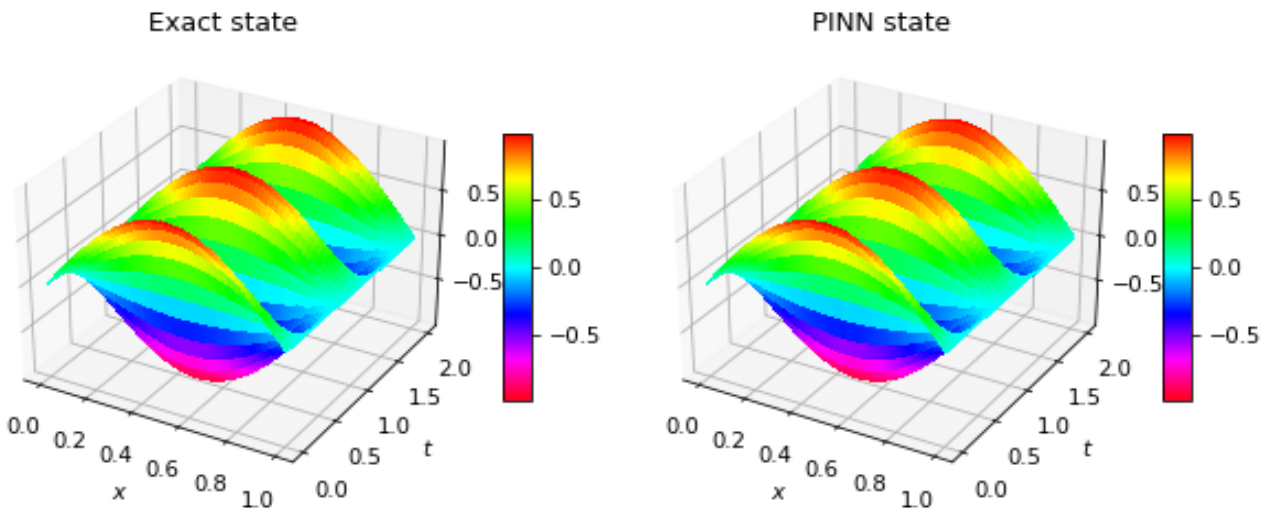


Figure 25: PINN and exact solution ($\lambda_0 = 1$; weights of the losses=[1, 1, 1, 1, 1, 1])

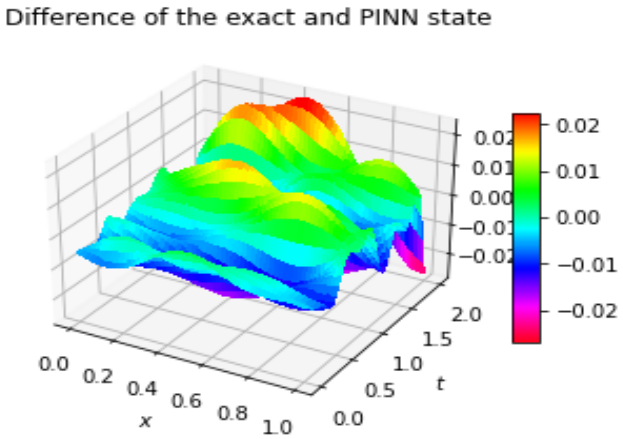


Figure 26: Difference between two states ($\lambda_0 = 1$; weights of the losses=[1, 1, 1, 1, 1, 1])

27 shows that it can indeed work well in this set. We see that the outputs in the two below pictures follow the same pattern.

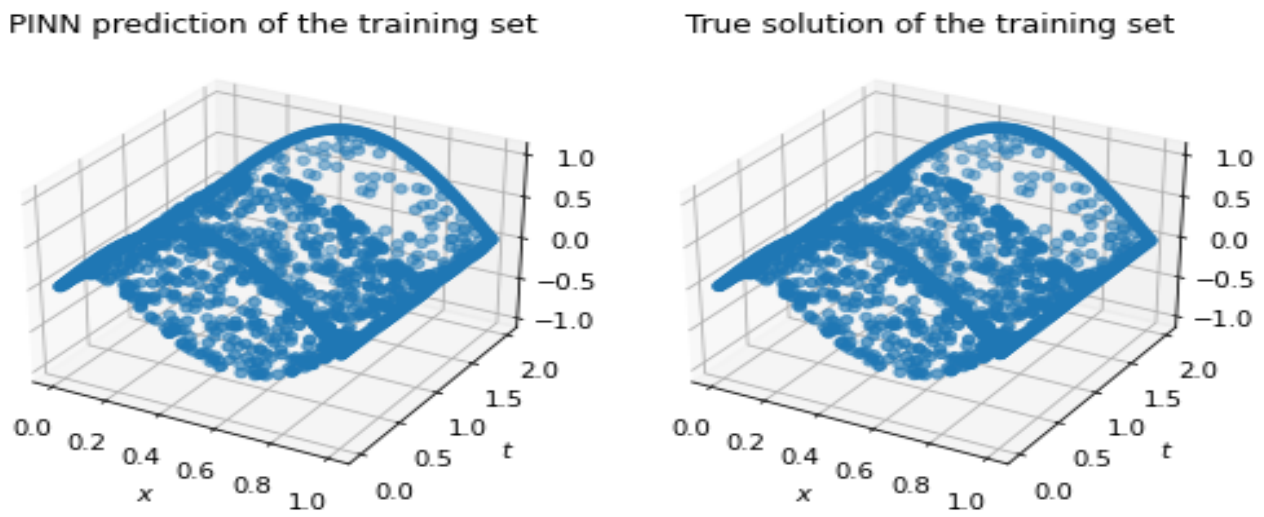


Figure 27: PINN and exact solution of the training set ($\lambda_0 = 1$; weights of the losses = [1, 1, 1, 1, 1, 1])

We can even see the difference between the true and PINNs predicted outcomes in the following plot:

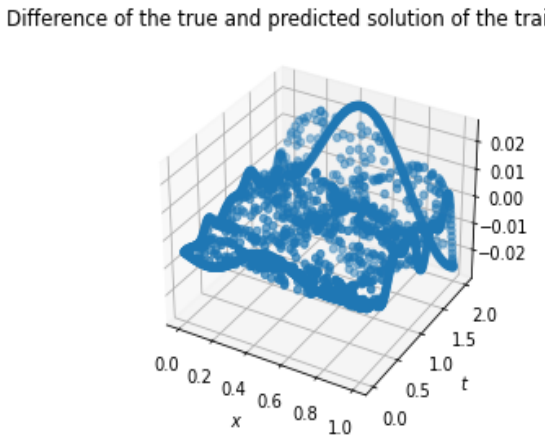


Figure 28: Difference of the results of the training set

Similar to Figure 26, the difference between the exact and PINNs outputs on the training set lies in the interval [-0.02, 0.02], where the majority of these differences is negligible. We can therefore say, that the model has a satisfying performance on the training data.

We let λ update in every epoch with $\lambda_0 = 1$. We obtain therefrom the separate losses for our model at the end of each epoch with the updated value of $\lambda = \lambda_k$ for $k = 0, 1, ..., \#\text{epochs} = 60000$ as below:

$$L_{PDE} = \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}_{tt}(x_i,t_i) - \lambda_k \hat{y}_{xx}(x_i,t_i))|^2, \tag{22}$$

$$L_{x=0} = \frac{1}{|\tau_{x=0}|} \sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{y}(x_i,t_i)|^2, \tag{23}$$

$$L_{t=0_1} = \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}_t(x_i,t_i)|^2, \tag{24}$$

$$L_{x=1} = \frac{1}{|\tau_{x=1}|} \sum_{(x_i,t_i)\in\tau_{x=1}} |\hat{y}(x_i,t_i)|^2, \tag{25}$$

$$L_{t=0_2} = \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}(x_i,t_i) - sin(\pi x_i)|^2, \tag{26}$$

$$L_{\text{train}} = \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}(x_i,t_i) - y(x_i,t_i)|^2. \tag{27}$$

Figure 29 shows the value of each of these individual losses at a particular iteration step used to train the model (epoch):
We can see that each of these losses converges to 0 after a certain number of epochs. This can provide justification on why our model is successful in solving the inverse problem, as well as in approximating almost the exact output on any given data as shown Figure 24 and 25 respectively.

3.2.2 Adaptive Loss-Weighting

In this section, we aim to improve the performance of our model by increasing the weight on the last component of the loss function in 22.

The weights of the loss function: [1, 1, 1, 1, 1, 6]; $\lambda_0 = 1$

Let us consider another example, where λ changes every one epoch and we let its initial value to be 1. We changed the weights in the loss function components and obtained therefrom the new loss function that we needed to minimize, putting a greater weight on the last component which is the summation over the training set of the square difference between the true output value of y and its prediction from our neural

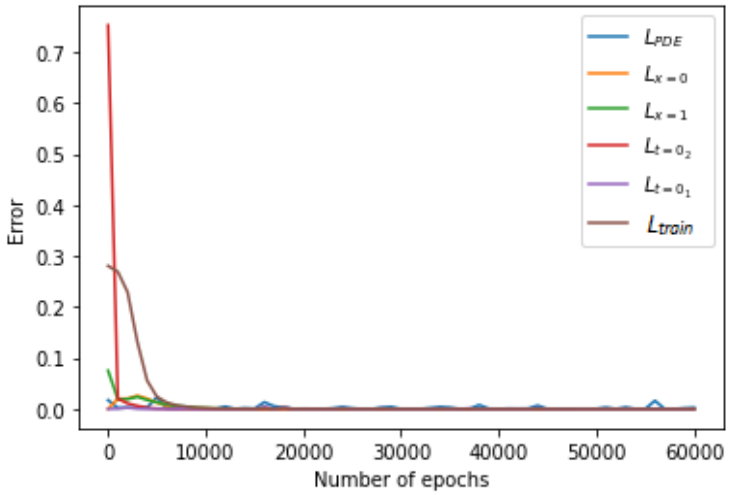


Figure 29: Predicted vs. true values of λ ($\lambda_0 = 1$)

network $\hat{y}$:

$$L = 1 \times \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}_{tt}(x_i,t_i) - \lambda \hat{y}_{xx}(x_i,t_i))|^2 + 1 \times \frac{1}{|\tau_{x=0}|} \sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{y}(x_i,t_i)|^2 + 1 \times \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}_t(x_i,t_i)|^2 +$$
$$+1 \times \frac{1}{|\tau_{x=1}|} \sum_{(x_i,t_i)\in\tau_{x=1}} |\hat{y}(x_i,t_i)|^2 + 1 \times \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}(x_i,t_i) - sin(\pi x_i)|^2 + 6 \times \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}(x_i,t_i) - y(x_i,t_i)|^2$$

The following plot illustrates the predicted values of λ:

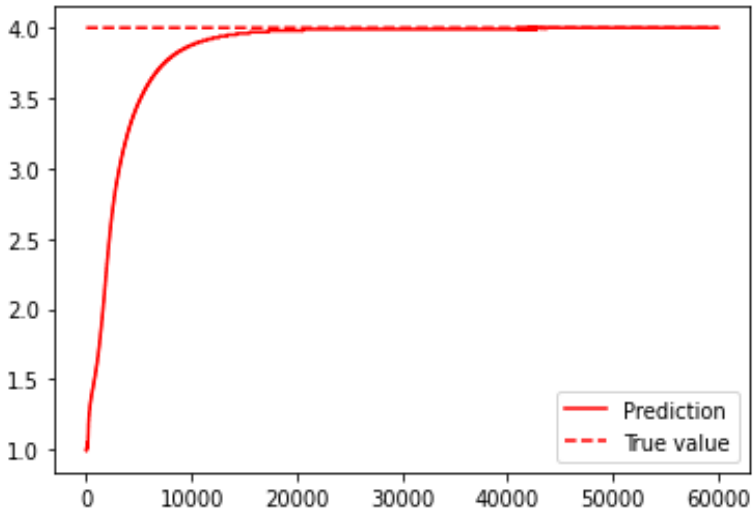


Figure 30: Predicted vs. true values of λ ($\lambda_0 = 1$), weights=[1, 1, 1, 1, 1, 6]

Our PINNs model finds the best value of λ to be 4.0. Figure 30 shows that the predicted values of λ nearly reach its true value after 2000 epochs. Figure 24 shows however that this is achieved after approximately 2700 epochs. Moreover, when using equal weights in the loss function as in the first example, we obtained the best predicted value of λ to be 3.99, whereas this model can reach a perfect prediction of λ equal to its true value. Let us know analyze the performance of this developed model by comparing its output of the input data (x, t) to the true value of $y(x, t)$ that satisfies 12.

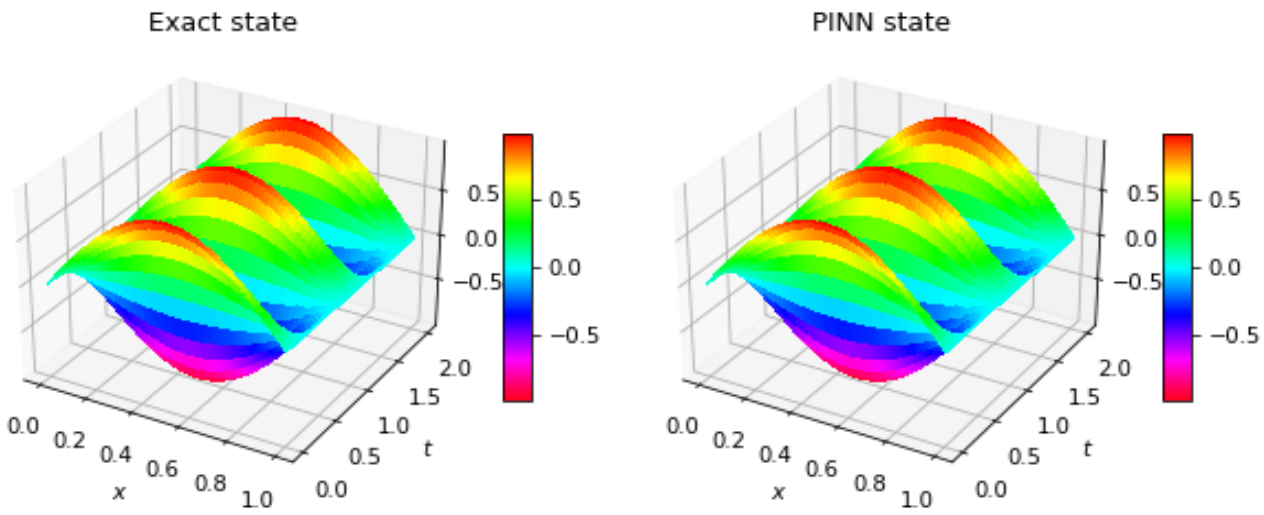


Figure 31: PINN and exact solution; loss weights=[1, 1, 1, 1, 1, 6]

The following plot which shows the difference between these two states enables us to better reason about the choice of this model:

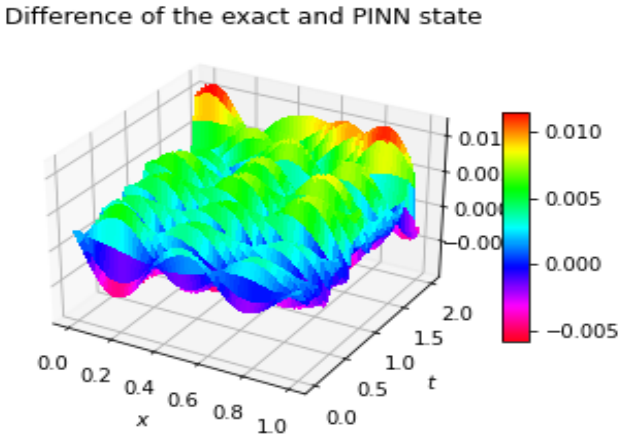


Figure 32: Difference between two states; loss weights=[1, 1, 1, 1, 1, 6]

Figure 32 shows that the difference between the exact result and PINNs predicted output lies in the range [-0.005, 0.010] where only negligible regions of the difference plot touch the boundary of this interval. Moreover, we can see that this difference equals 0 in the majority of its surface. Figure pertaining to the same selection of weights, shows however that this difference lies in the segment [-0.02, 0.02], where a smaller surface show a difference of 0. From these arguments, we conclude that this weights' selection improved the predictive performance of our model. Let us see how this model predicts the output of the training input data:

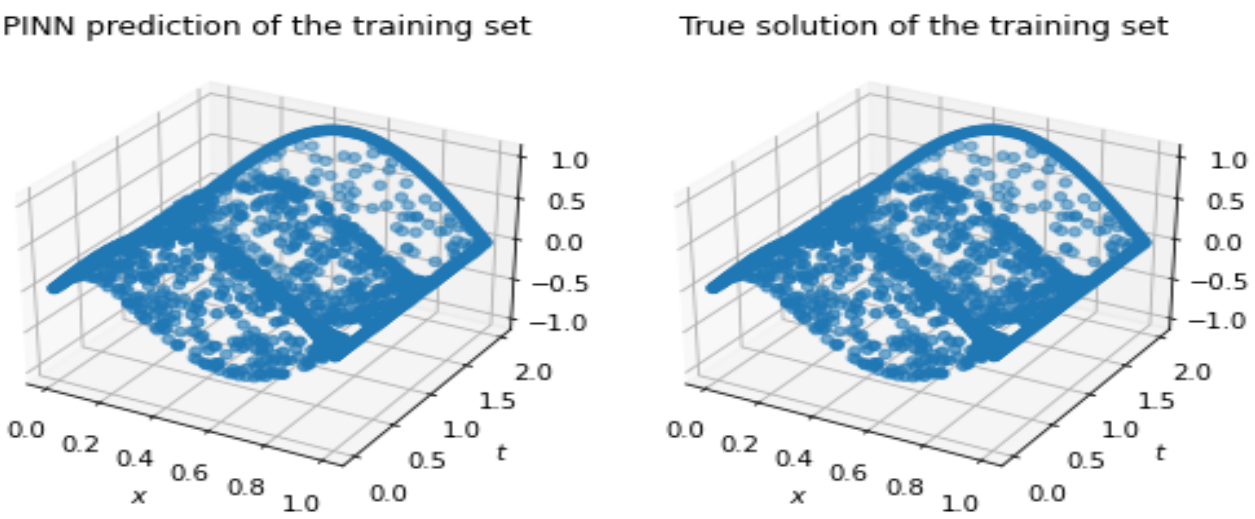


Figure 33: PINN predicted and exact solution of the training set; loss weights=[1, 1, 1, 1, 1, 6]

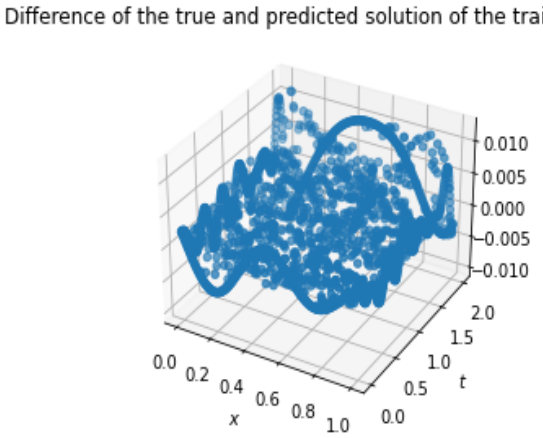


Figure 34: Difference of the results of the training set; loss weights=[1, 1, 1, 1, 1, 6]

Figure 34 shows that the model performs well on the training sample, indicating that the exact results and PINNs predicted outputs on the majority of the training samples coincide. We can see that only a few points touch the boundaries of the interval [-0.01, 0.01].

We conclude this section by summarizing in a plot the individual losses of this model outlined in 22.

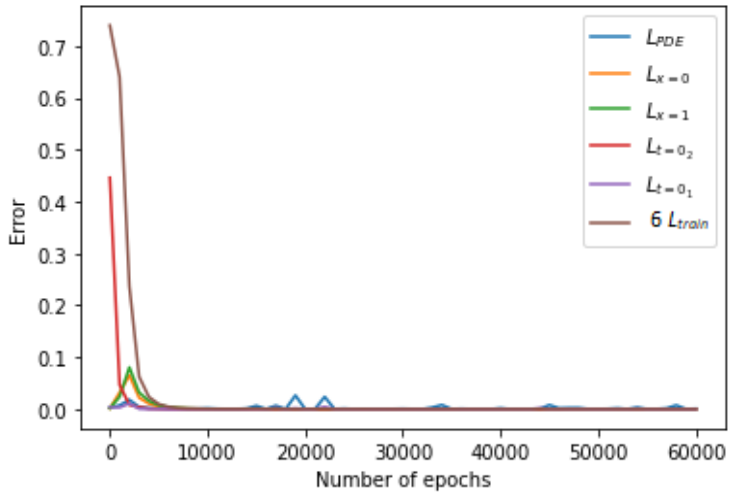


Figure 35: Separate loss functions; loss weights=[1, 1, 1, 1, 1, 6]

Figure 35 shows that each of these losses converge to 0 after some number of epochs. We can even see that this pattern is very similar to the Figure 29 which means that both models work well and fulfill the goal of PINNs.

The weights of the loss function: [0.1, 0.1, 0.1, 0.1, 0.1, 3]; $\lambda_0 = 1$ We now consider another setting for our model with $\lambda_0 = 1$ and the following loss function:

$$L = 0.1 \times \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}_{tt}(x_i,t_i) - \lambda \hat{y}_{xx}(x_i,t_i))|^2 + 0.1 \times \frac{1}{|\tau_{x=0}|} \sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{y}(x_i,t_i)|^2 + 0.1 \times \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}_t(x_i,t_i)|^2 +$$
$$+0.1 \times \frac{1}{|\tau_{x=1}|} \sum_{(x_i,t_i)\in\tau_{x=1}} |\hat{y}(x_i,t_i)|^2 + 0.1 \times \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}(x_i,t_i) - sin(\pi x_i)|^2 + 3 \times \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}(x_i,t_i) - y(x_i,t_i)|^2$$

Figure 36 shows the predicted values of λ:

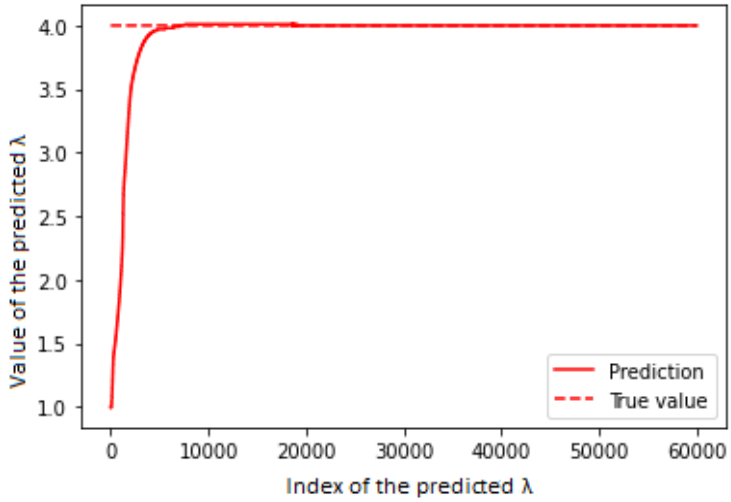


Figure 36: Predicted vs. true values of λ ($\lambda_0 = 1$), weights=[0.1, 0.1, 0.1, 0.1, 0.1, 3]

We can clearly from Figure 36 that our model finds a very good λ approximation (3.99) of its true value (4.0) after less than 10000 epochs (around 7000 to be precise). The first example we analyzed (with equal loss weights), however, shows that this result can be reached after 3000 epochs (Figure 24), which requires more compiling time. Moreover, we can see that the λ predictions from this model coincide with its true value after exactly 20000 epochs. The second example we examined (with the loss weights [1, 1, 1, 1, 1, 6]) shows nevertheless that the predicted λ value equals its true value after approximately 45000 epochs which is computationally more expensive.

The following plots show the exact solution, PINNs predicted solution and their difference to enable a better analysis of the model performance:

Figure 38 shows that the difference between the PINNs predicted and exact state is almost included in the interval [-0.005, 0.005] units which suggests that the two resulting outputs differ here less than in our two previous cases (see Figures 26 and 32)

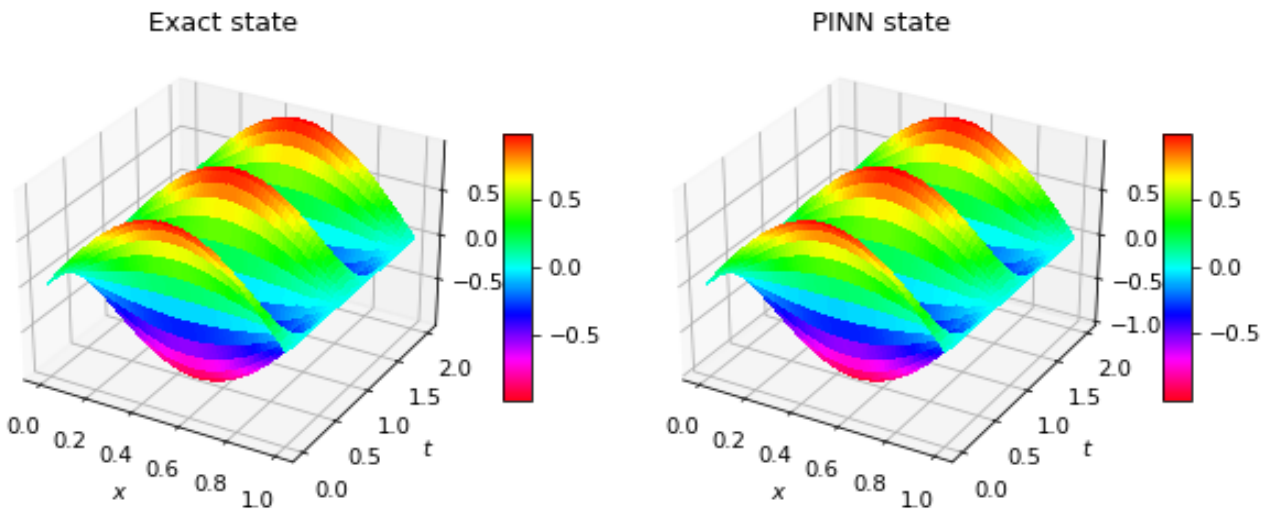


Figure 37: PINN and exact solution; loss weights=[0.1, 0.1, 0.1, 0.1, 0.1, 3]

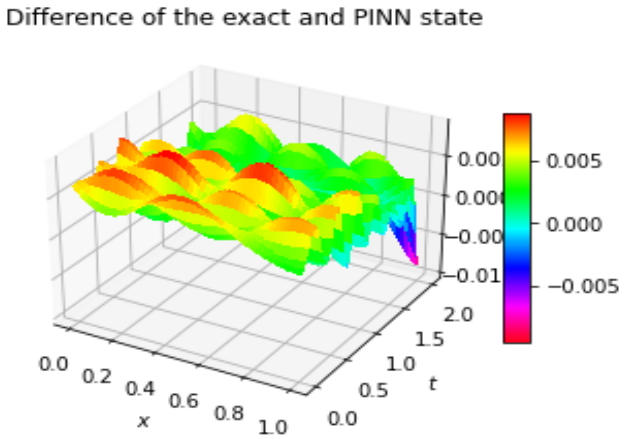


Figure 38: Difference between two states; loss weights=[0.1, 0.1, 0.1, 0.1, 0.1, 3]

The PINNs and the true output of the training samples input as well as their difference are plotted below:

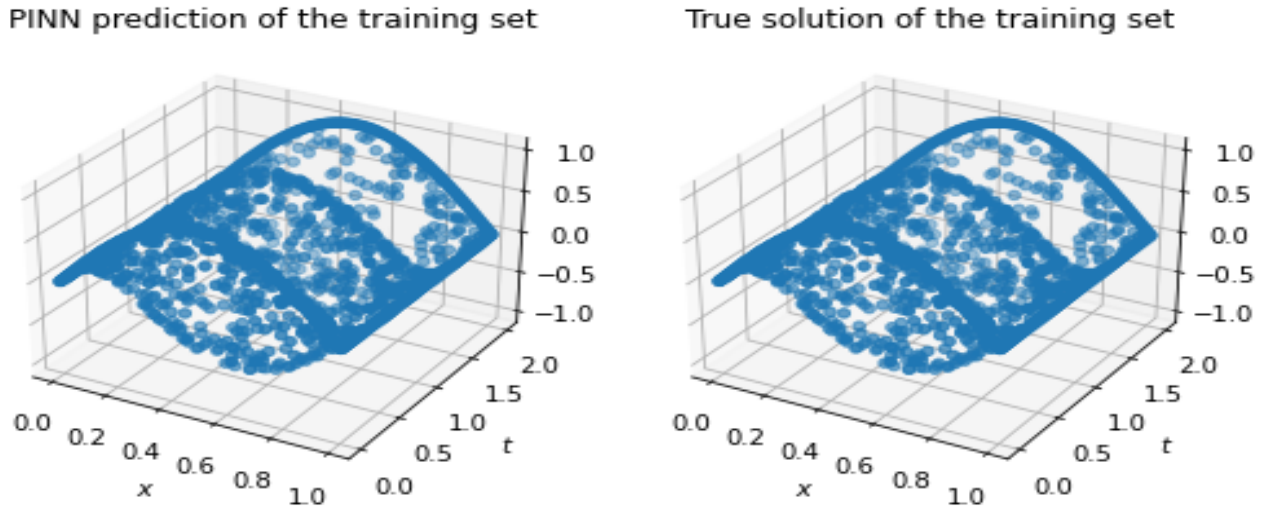


Figure 39: PINN predicted and exact solution of the training set; loss weights=[0.1, 0.1, 0.1, 0.1, 0.1, 3]

Similarly to our previous two examples (see Figures 27 and 33), the predictive performance of this model

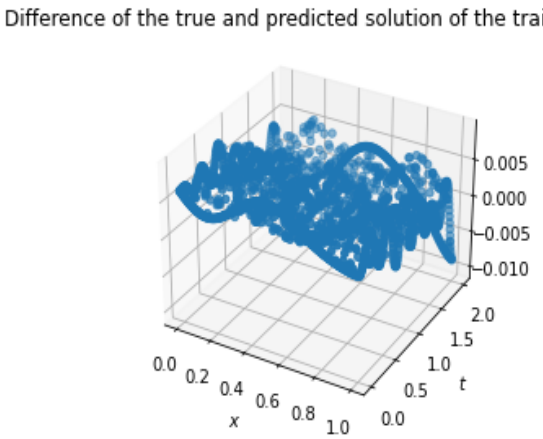


Figure 40: Difference of the results of the training set; loss weights=[0.1, 0.1, 0.1, 0.1, 0.1, 3]

on the training sample is satisfying, since the difference of the exact and PINNs predicted outputs on this sample, differ negligibly.

The individual loss components of this model are shown below:

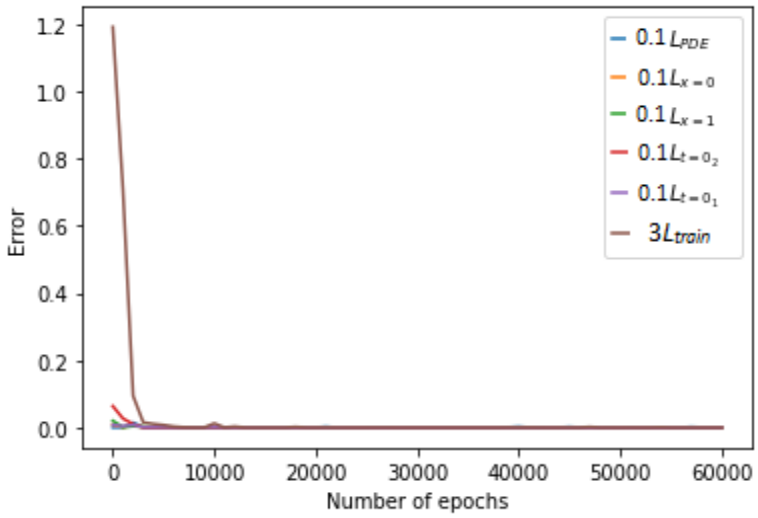


Figure 41: Separate loss functions; loss weights=[0.1, 0.1, 0.1, 0.1, 0.1, 3]

Figure 41 shows that each of the loss components of our loss function converges to 0 after some epochs. However, it is interesting to notice that this plot does not show any initial oscillations of any of the components, like the one of L_{PDE} in Figures 29 and 35 of our previous models. We can conclude that this model performs well, even when we use less epochs, which saves compiling time.

The weights of the loss function: [1, 1, 1, 1, 1, 1]; $\lambda_0 = 3.8$ We select the initial value of λ to be 3.8, which is quite near its true value. We set the number of epochs to be 60000. We let the loss function we need to minimize be:

$$L = \frac{1}{|\tau_{int}|}\sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}_{tt}(x_i,t_i) - \lambda\hat{y}_{xx}(x_i,t_i))|^2 + \frac{1}{|\tau_{x=0}|}\sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{y}(x_i,t_i)|^2 + \frac{1}{|\tau_{t=0}|}\sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}_t(x_i,t_i)|^2 +$$
$$+\frac{1}{|\tau_{x=1}|}\sum_{(x_i,t_i)\in\tau_{x=1}} |\hat{y}(x_i,t_i)|^2 + \frac{1}{|\tau_{t=0}|}\sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}(x_i,t_i) - sin(\pi x_i)|^2 + \frac{1}{|\tau_{int}|}\sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}(x_i,t_i) - y(x_i,t_i)|^2.$$

Figure 42 displays the values of λ predicted by our model at the end of each epoch.

The best value of the predicted λ is found to be 4.0, exactly as its true value after approximately 52000 epochs. Figure 43 shows the exact and the PINNs predicted output of our PDE, under the conditions of this

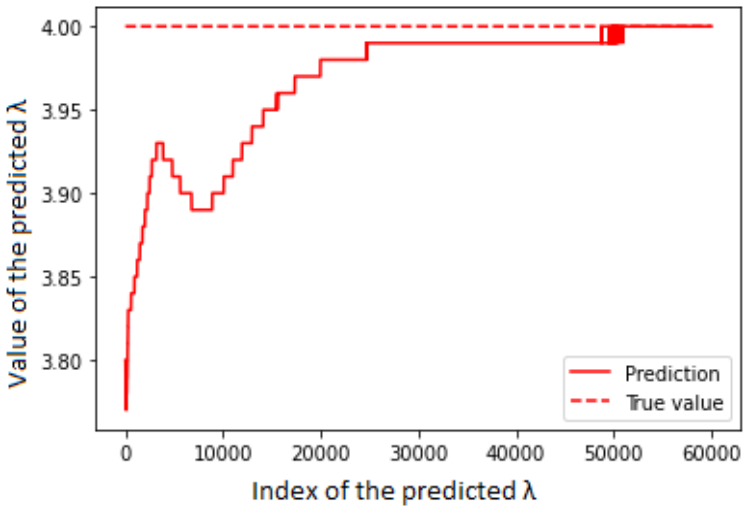


Figure 42: Predicted vs. true values of λ ($\lambda_0 = 3.8$); loss weights=[1, 1, 1, 1, 1, 1]

developed model.

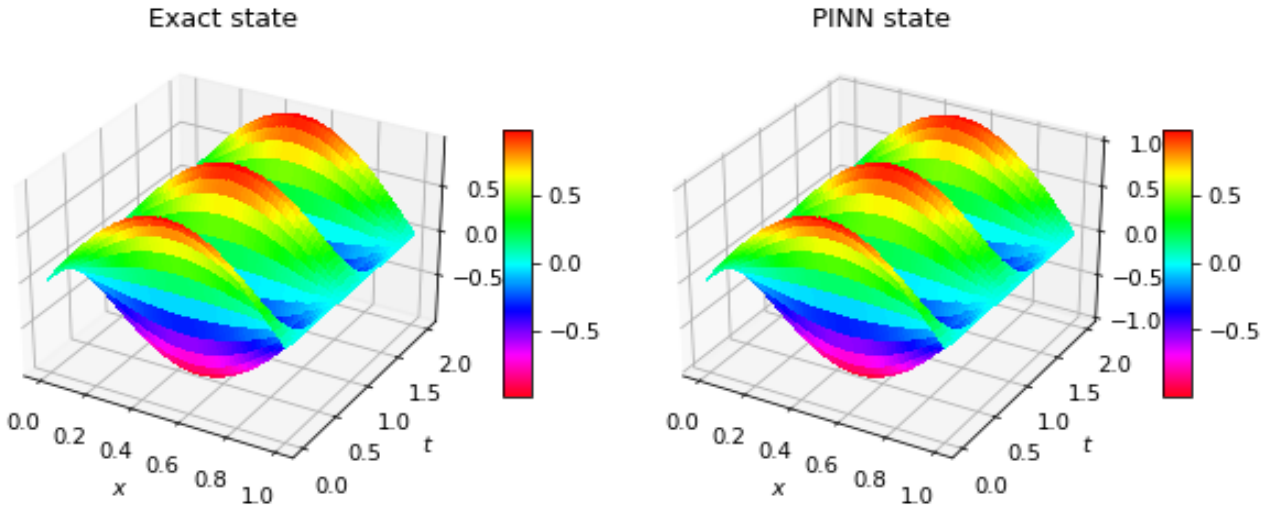


Figure 43: PINN and exact solution

Roughly speaking, the PINN state of Figure 43 nearly coincide with the Exact state. However, similar to or previous approaches, we plot also the difference between these two states as shown in Figure 44.

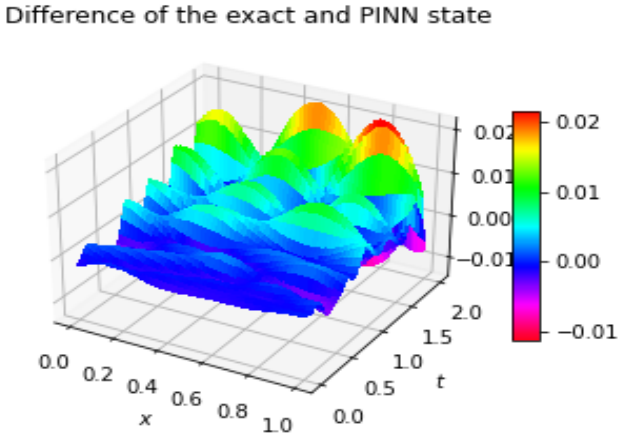


Figure 44: Difference between two states

We see from Figure 44 that the difference between two states is generally negligible, included in the interval [-0.01, 0.02] indicating a satisfying predictive performance of our model. Moreover, we notice that the greatest part of the surface indicates that the difference between two states is 0, i.e., the two states mostly coincide. Next, we can judge the predictive performance of our model on the training set through the following plots which display the exact and PINNs predicted output as well as their difference (see Figures 45 and 46 respectively).

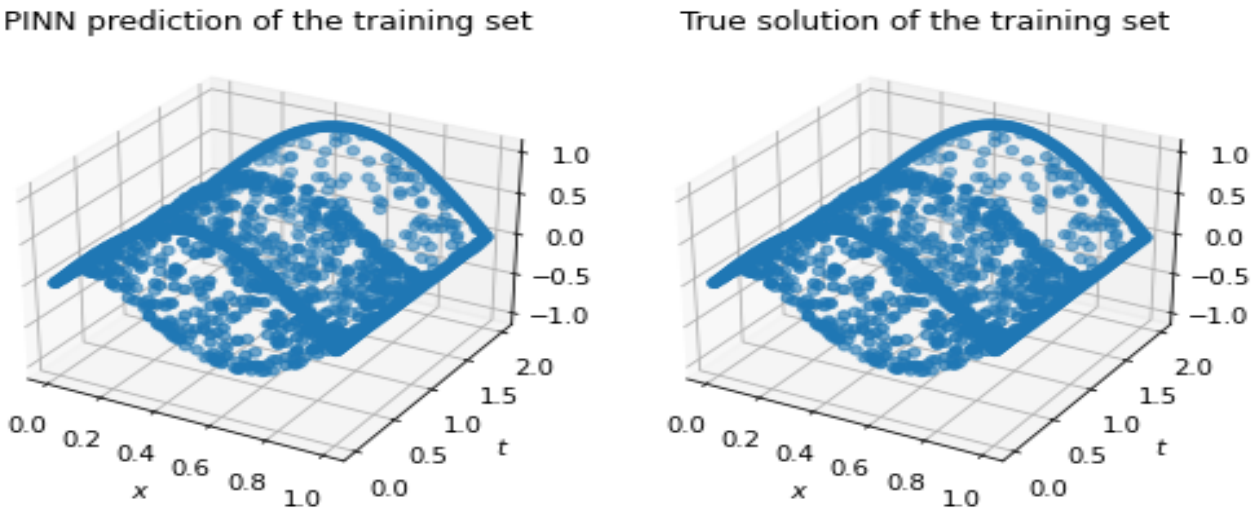


Figure 45: PINN predicted and exact solution of the training set; loss weights=[1, 1, 1, 1, 1, 1]

We can infer from Figure 46 that the difference between these the exact and PINNs predicted states on the training set is negligible; mostly around 0 and reaches its highest value of 0.02.

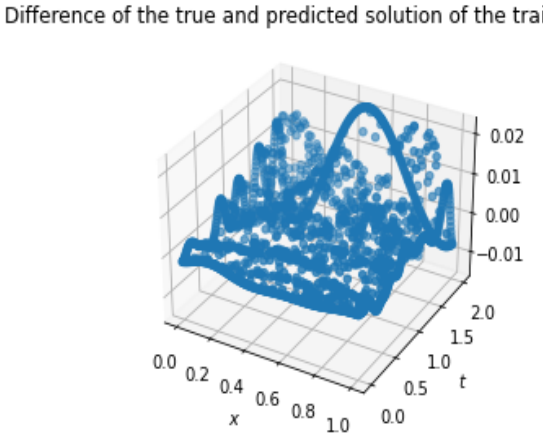


Figure 46: Difference of the results of the training set; loss weights=[1, 1, 1, 1, 1, 1]

The individual loss components of the loss function we used, are shown in Figure 47:

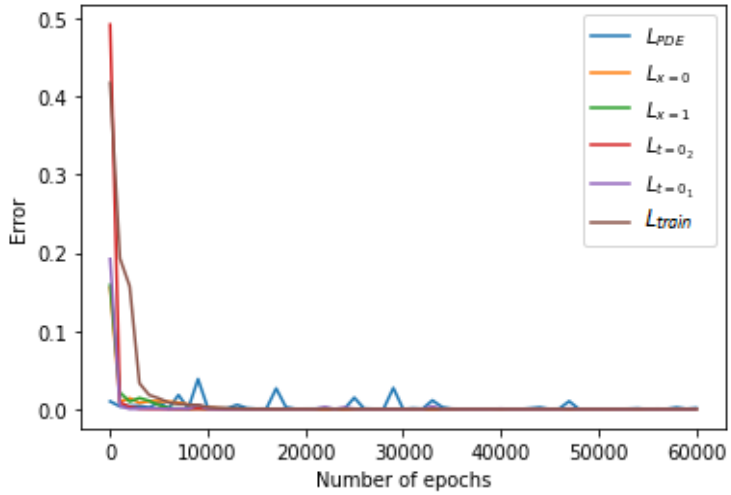


Figure 47: Separate loss functions; loss weights=[1, 1, 1, 1, 1, 1]

Each of the loss components converges to 0 after 50000 epochs. Moreover, exactly after 52000 epochs our model could predict the true value of λ (see Figure 42). We arrive in the conclusion that this model has a satisfying performance which is reached when a high number of epochs is used (50000). This fact makes it in particular computationally costly.

The weights of the loss function: [1, 1, 1, 1, 1, 6]; $\lambda_0 = 3.8$ Motivated from the first three examples, we now analyze whether increasing the weight on the last loss component can lower the computational time of our model and still yield meaningful results. We consider $\lambda_0 = 3.8$ and let the loss function be:

$$L = 1 \times \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}_{tt}(x_i,t_i) - \lambda \hat{y}_{xx}(x_i,t_i))|^2 + 1 \times \frac{1}{|\tau_{x=0}|} \sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{y}(x_i,t_i)|^2 + 1 \times \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}_t(x_i,t_i)|^2 +$$
$$+1 \times \frac{1}{|\tau_{x=1}|} \sum_{(x_i,t_i)\in\tau_{x=1}} |\hat{y}(x_i,t_i)|^2 + 1 \times \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}(x_i,t_i) - sin(\pi x_i)|^2 + 6 \times \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}(x_i,t_i) - y(x_i,t_i)|^2$$

Figure 48 shows the predicted values of λ at the end of every epoch. We can clearly observe that the exact value of λ (4.0) is predicted after exactly 36000 epochs, which is already an improvement compared to the higher number of epochs required in the previous example (around 52000; see Figure 42).

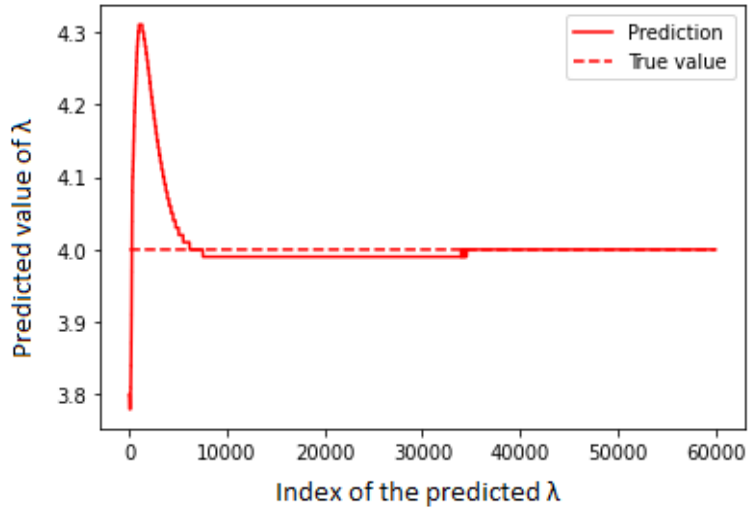


Figure 48: Predicted vs. true values of λ ($\lambda_0 = 3.8$); loss weights=[1, 1, 1, 1, 1, 6]

Figure 49 shows the PINNs predicted and exact output for this model. Figure 50, which displays the difference between the exact and PINNs state, suggests that this model works very well, since the greatest part of this difference is green (indicating a value of 0; i.e., te two states mostly coincide). A good performance of the model is noticed also on the training set, where the difference between the exact and PINNs predicted outputs amounts generally to 0 (see Figure 52).

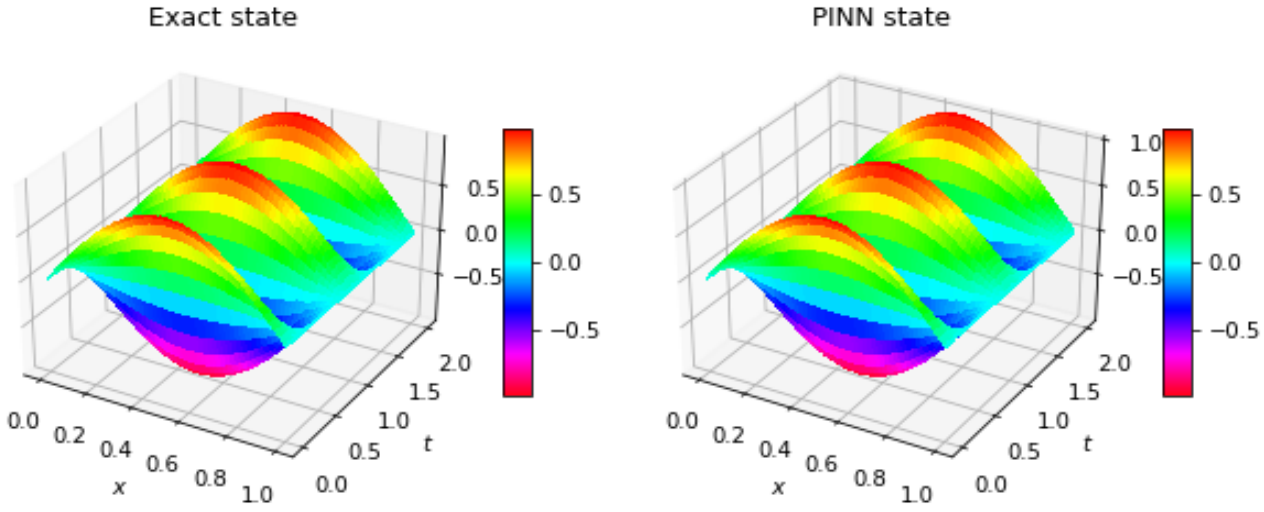


Figure 49: PINN and exact solution

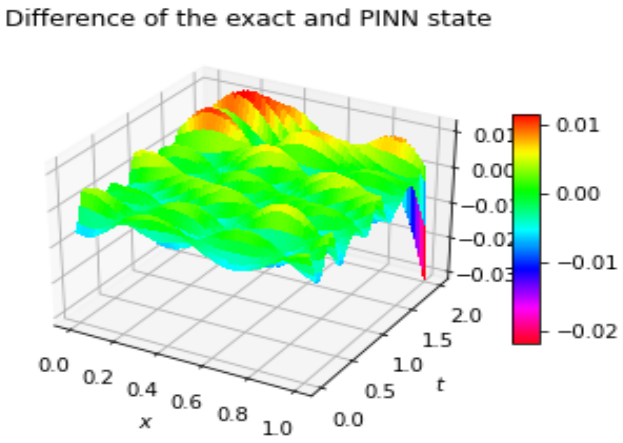


Figure 50: Difference between two states

The PINNs and the true output of the training samples input as well as their difference are plotted below:

Figure 53 shows the individual loss components we used in the loss function of this model: We can see that the loss components converge to 0 (and stay in that range constant) after 30000 epochs. The same performance could be achieved by the previous model after 50000 epochs, requiring therefore more time. We can conclude that this model (with the selection of weights as indicated in the top) is computationally cheaper and outputs valid results; thus recommended.

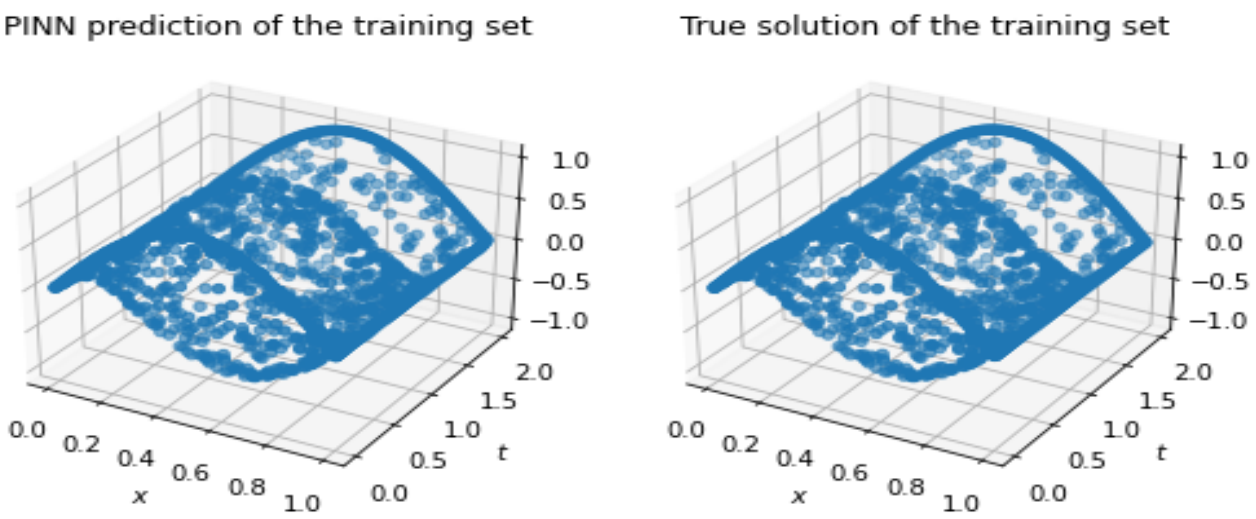


Figure 51: PINN predicted and exact solution of the training set; loss weights=[1, 1, 1, 1, 1, 6]

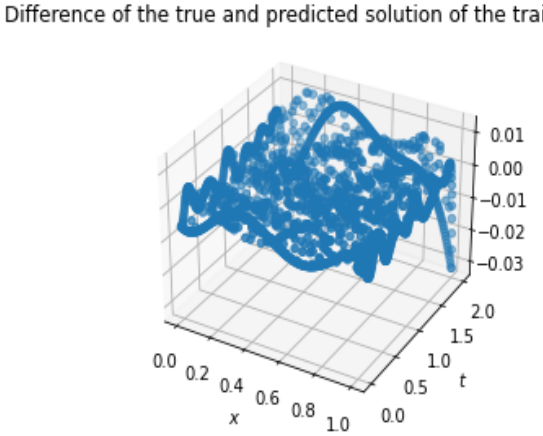


Figure 52: Difference of the results of the training set; loss weights=[1, 1, 1, 1, 1, 6]

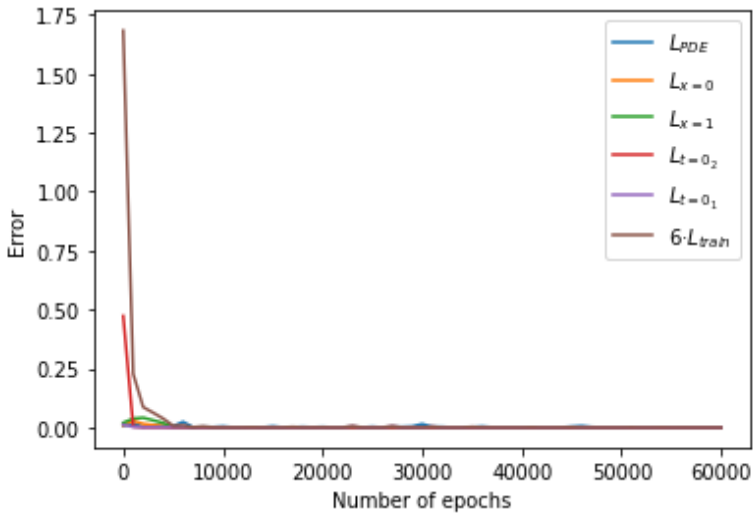


Figure 53: Separate loss functions; loss weights=[1, 1, 1, 1, 1, 6]

The weights of the loss function: [1, 1, 1, 1, 1, 1]; $\lambda_0 = 0.05$ We finally aim that our Machine Learning model performs well on solving the inverse problem, even though the initial value of λ is far from its true

value, 0.05 respectively. We integrate the following loss function in our model:

$$L = \frac{1}{|\tau_{int}|}\sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}_{tt}(x_i,t_i) - \lambda\hat{y}_{xx}(x_i,t_i))|^2 + \frac{1}{|\tau_{x=0}|}\sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{y}(x_i,t_i)|^2 + \frac{1}{|\tau_{t=0}|}\sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}_t(x_i,t_i)|^2 +$$
$$+\frac{1}{|\tau_{x=1}|}\sum_{(x_i,t_i)\in\tau_{x=1}} |\hat{y}(x_i,t_i)|^2 + \frac{1}{|\tau_{t=0}|}\sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}(x_i,t_i) - sin(\pi x_i)|^2 + \frac{1}{|\tau_{int}|}\sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}(x_i,t_i) - y(x_i,t_i)|^2.$$

Figure 54 shows the predicted values of λ for each epoch. We ca see that the model fails to predict the exact value of λ, even though we used a high number of epochs (60000). However, the resulting predictions seem to very slowly approach the true value.

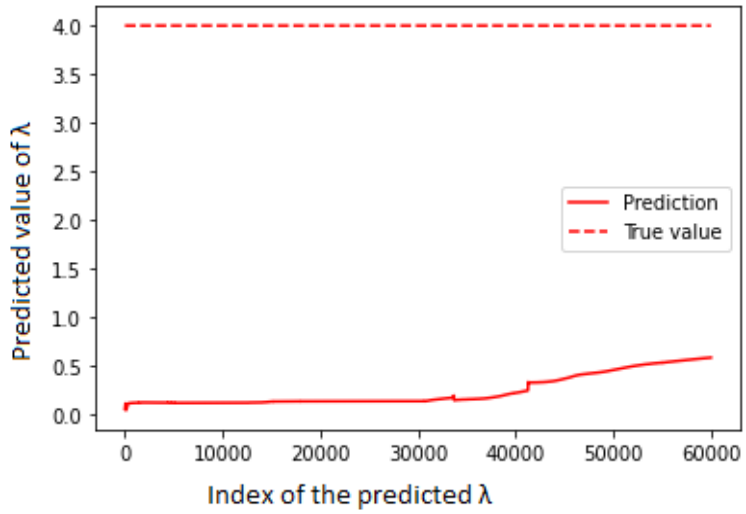


Figure 54: Predicted vs. true values of λ ($\lambda_0 = 0.05$), weights=[1, 1, 1, 1, 1, 1]

Figure 55 shows that the developed model with the best value of λ it could find, can not infer properly the solution of 1. We ca clearly see that PINN state is far from a meaningful and valid result. Moreover, 56 shows that the difference between the PINNs predicted and exact output is actually quite big compared to our previous cases, resulting in a worse prediction performance.

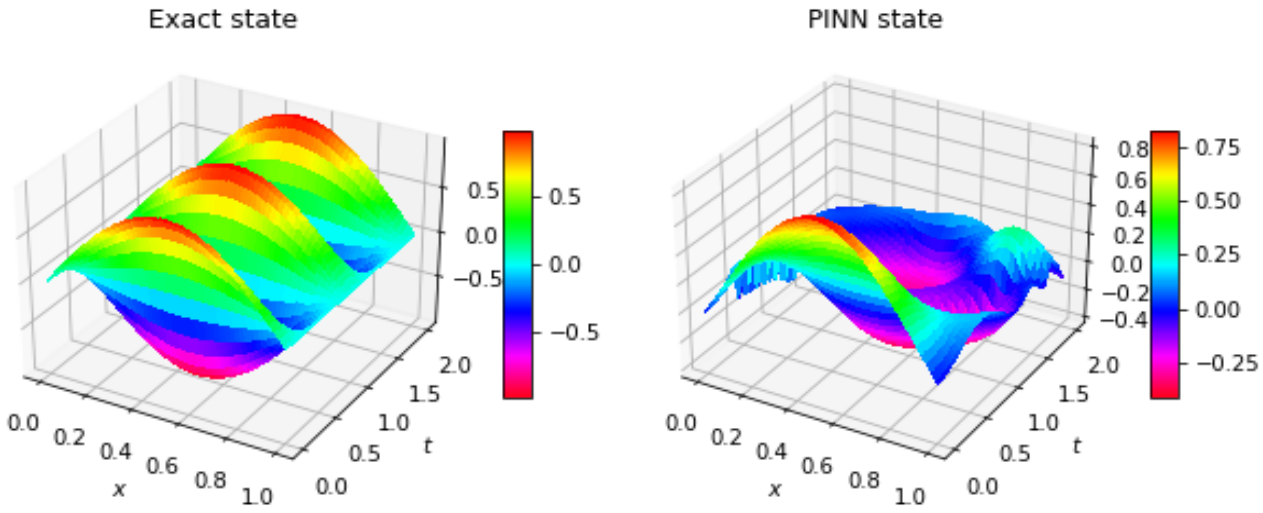


Figure 55: PINN and exact solution; $\lambda_0 = 0.05$, loss weights=[1, 1, 1, 1, 1, 1]

We can now analyze the performance of this model on the training set we used. Figure 57 shows that the pattern the PINNs outputs on the training set is quite different from the one of the exact outputs on this set.

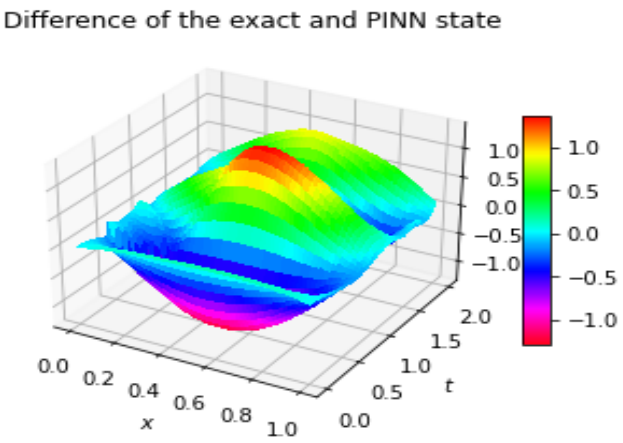


Figure 56: Difference between two states

Figure 58 shows that the difference between these two states on the training set is quite big compared to our previous models. We can therefore conclude that this model performs quite badly on the training set as well.

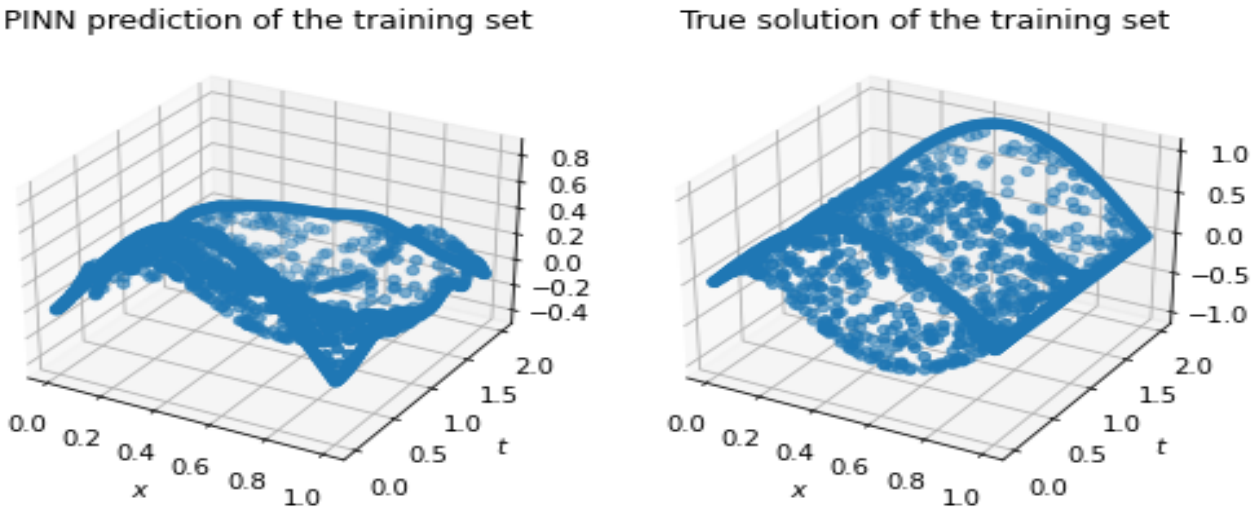


Figure 57: PINN predicted and exact solution of the training set;$\lambda_0 = 0.05$, loss weights=[1, 1, 1, 1, 1, 1]

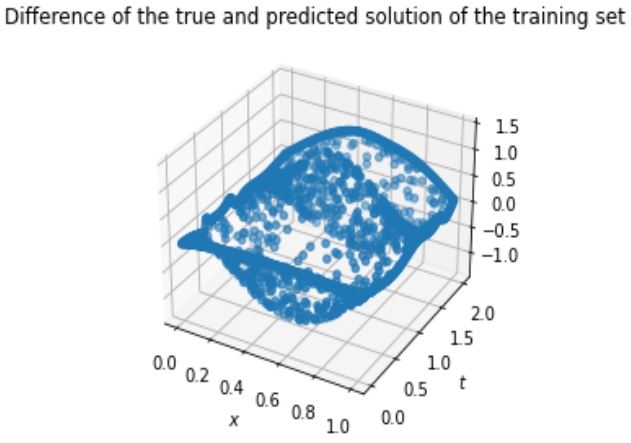


Figure 58: Difference of the results of the training set;$\lambda_0 = 0.05$, loss weights=[1, 1, 1, 1, 1, 1]

Since our model does not perform well, we would expect that the loss components of it, do not converge to 0. We need therefore, to analyze each of these components, so that we can adapt the weights in the loss function properly and enable better predictions. Figure 59 shows the individual losses as a function of the epochs.

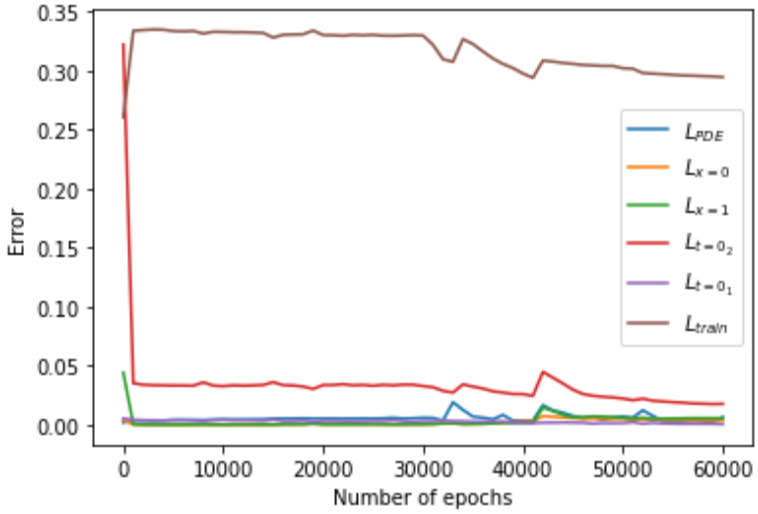


Figure 59: Separate loss functions; loss weights=[1, 1, 1, 1, 1, 1]

We can see that L_{train} is far from converging to 0. In the next section, we increase the weight on this component so that the model focuses on minimizing in particular this loss. We obtain therefrom a model with a better performance.

The weights of the loss function: [1, 1, 1, 1, 1, 6]; $\lambda_0 = 0.05$ We let $\lambda_0 = 0.05$ and increase the weight of the last loss component such that we get the following loss function that we need to minimize:

$$L = 1 \times \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}_{tt}(x_i,t_i) - \lambda \hat{y}_{xx}(x_i,t_i))|^2 + 1 \times \frac{1}{|\tau_{x=0}|} \sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{y}(x_i,t_i)|^2 + 1 \times \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}_t(x_i,t_i)|^2 +$$
$$+1 \times \frac{1}{|\tau_{x=1}|} \sum_{(x_i,t_i)\in\tau_{x=1}} |\hat{y}(x_i,t_i)|^2 + 1 \times \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{y}(x_i,t_i) - sin(\pi x_i)|^2 + 6 \times \frac{1}{|\tau_{int}|} \sum_{(x_i,t_i)\in\tau_{int}} |\hat{y}(x_i,t_i) - y(x_i,t_i)|^2$$

Below, we can see the predicted value of λ resulting at the end of each epoch:

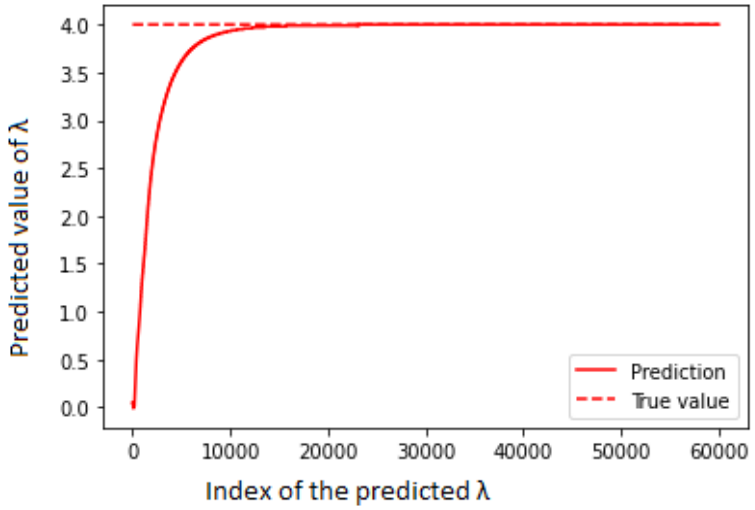


Figure 60: Predicted vs. true values of λ ($\lambda_0 = 0.05$), loss weights=[1, 1, 1, 1, 1, 6]

The model achieves to predict the exact value of λ after 25000 epochs. In this regard, this model performs

better than the previous one, where the true value of λ could not be predicted even after 60000 epochs (see Figure 54), Figure 61 shows the PI ad exact state of the points in the domain. We can see that their patterns are quite similar. Moreover, Figure 62 suggests that the difference between these states is negligible (included i the segment [-0.015, 0.005]) compared to the previous result (see Figure 56)

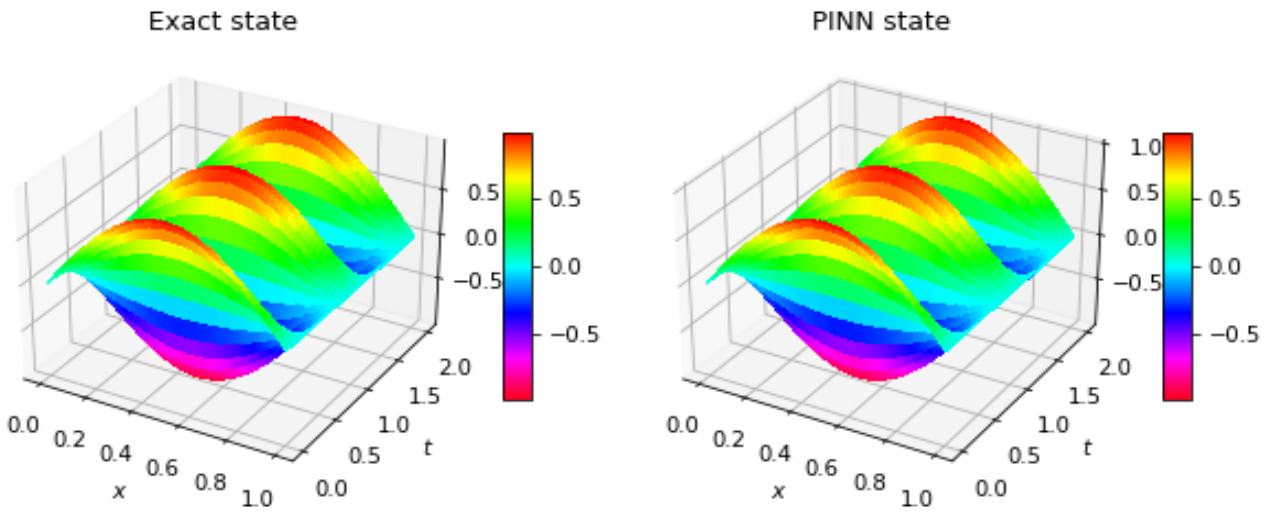


Figure 61: PINN and exact solution

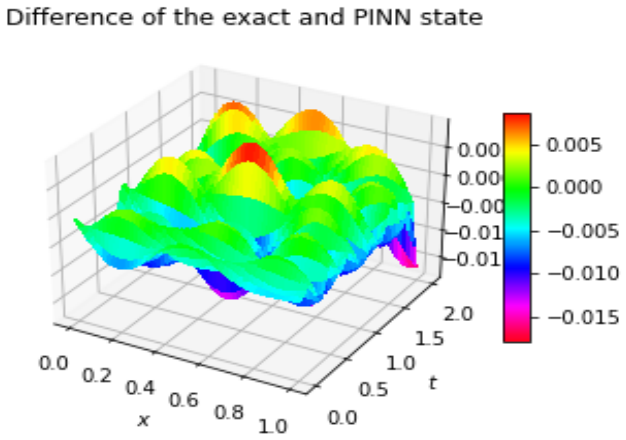


Figure 62: Difference between two states

Figures 63 and 64 enable us to analyze the performance of this model on the training set. We can see that it performs quite well: the PINN predicted and exact outputs on the training sample differ negligibly as shown in Figure 64 (included in the segment [-0.015, 0.005]).

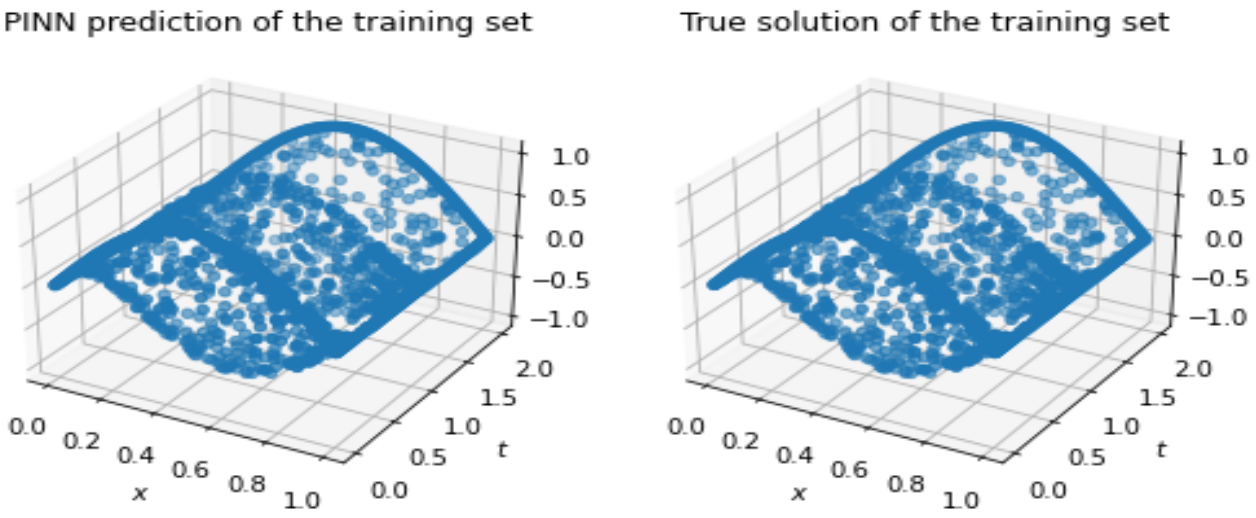


Figure 63: PINN predicted and exact solution of the training set; loss weights=[1, 1, 1, 1, 1, 6]

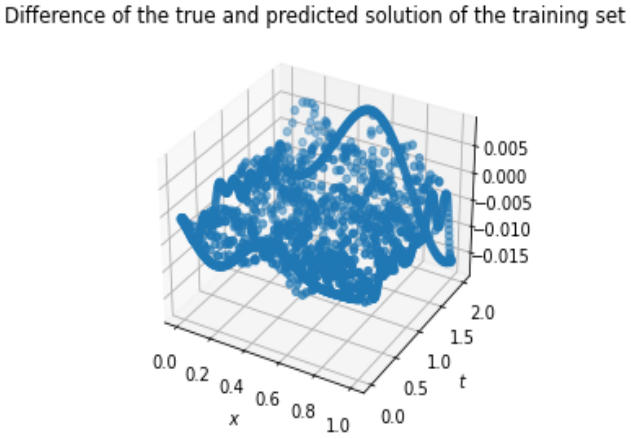


Figure 64: Difference of the results of the training set; loss weights=[1, 1, 1, 1, 1, 6]

We expect that increasing the weight on L_{train} enables L_{train} converge to 0. Te following plot supports our claim: we can see that all the individual losses converge to 0, which suggests that our model works very well. We could infer this fact also from our above arguments: the true value of λ was found after 25000 epochs; the PINNs and exact state nearly coincide, the model predictions of the outputs on the training set differed negligibly from the exact outputs.

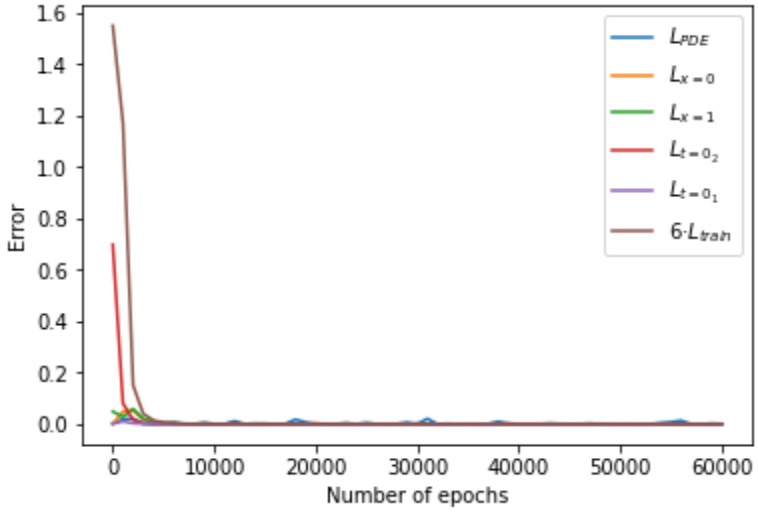


Figure 65: Separate loss functions; loss weights=[1, 1, 1, 1, 1, 6]

4 Internship Activities

This internship was organized by FAU Research Center for Mathematics of Data and lasted for three months (15.06.2022-14.09.2022). I worked remotely for the first and last two weeks and arrived in Erlangen on 1st of July where I stayed for the whole months of July and August. My first day at FAU was really wonderful! I had the chance to meet many Math Phds, Postdocs and also my supervisor Dr. Yue Wang. I felt integrated into their community! I was also very satisfied to have my own office, which helped me work focused and productive.
My learning experience on this topic started with the reading of the article [6] which offered a proper introduction and fundamental concepts in the field of PINNs. I was based on the codes presented in this article, to conduct my own realizations and obtain the above-described analysis and results. An important role in my work, played the open source of the *deepxde* library in Python that I use in all my realizations [5] respectively. I started applying the framework of PINNs to an easy example, that of solving the wave PDE system of equations shown in 1 whose true solution is known. I also applied this framework to solve the degenerating wave equation in 19 and the controllability problem in 20. I was assigned another task that could be solved using PINNs such as finding the solution of the inverse problem in 21.
I had the opportunity to meet the staff of Fraunhofer Institute for Integrated Circuits IIS in Erlangen who was working with PINNs to solve their applied math problems. We had weekly meetings to deliver relevant results in this field and help each other conceive better accuracy and performance of our models. I presented my findings and my progress made on this topic within each week.

5 Conclusions

We obtained some important results, while analyzing the simple 1d wave equation regarding the performance of our PINNs framework, the role of the structure of the neural network, the computational time needed to execute the model when varying some of its components. We can summarize our findings in the following points:

- The ML model performed better (in solving the simple 1d wave equation 1), when we compiled it using two optimization algorithms ("adam"+"L-BFGS-B") than when we only used "adam" algorithm. (See Figures 5, 3, and 4)
- The training error of ML model (used to solve the 1d wave equation in 1) mostly increased when the size of the training set increases. (See Figure 8)
- The summation of the square loss between the PINNs output and true output of $y(x,t)$ in 1 on the validation set as in 11 decreases when the size of the training set increases, however it stays constantly negligible when more than 70 training sample were used. (See Figure 9)
- The more training samples used to train the ML model, the more time is needed to compile/execute the developed model. (See Figure 10)
- We used many NN structures (to solve the forward 1d wave equation) where each of them had different number of nodes (we let the number of nodes range from 10 to 300). The NN with 300 nodes resulted in the best model performance (test loss converged to 0) and in the most computationally expensive one (see Figures 13 and 15).
- When solving the inverse problem in 21 to obtain the value of λ, we noticed that increasing the weight of the last component on the loss function of 21 which expresses the summation of the square loss

between the true and PINNs predicted output on the training set, resulted in a better and faster predictive performance of the model (see Figures 24, 30, and 36).

6 Acknowledgements

I would like to thank Prof. Dr. DhC. Enrique Zuazua and Dr. Michael Schuster for enabling this research summer internship, which was a unique experience for me. I am very grateful to my supervisor Dr. Yue Wang who assisted me in every difficulty I encountered during my learning. Finally I want to thank Dr. Daniel Veldman whose advices and suggestions helped me achieve significant results and improve the accuracy of my developed machine learning models.

Appendix

Here is an overview of numerical results related to this report, and is public on the

https://github.com/DCN-FAU-AvH/PINNs_wave_equation,

where you can donwnload the code of
Physics-informed neural network (PINN)

- solving forward problems
  - **Wave_equation.py** solves the 1d wave equation with Dirichlet Boundary conditions.
  - **Wave_equation_otherBC** solves the 1d wave equation with Neumann boundary conditions.
  - **train_error_val_error_time.py** displays the train error/validation error/computational time-size of training set dependencies
  - **test_loss_time.py** shows the test error-computational time dependency for a specific structure of neural network
  - **all_together_loss_time.py** shows the test error-computational time dependency for different structures of neural networks.
  - **changing_nodes_test_loss.py** shows the test error-computational time dependency for neural network structures with different numbers of nodes
  - **first_case_no_damage.py** solves the degenerating 1d wave equation when $a(x) \equiv 4$
  - **second_case_damage.py** solves the degenerating 1d wave equation when $a(x) = 8|x - 0.5|$
  - **third_case_double_damage.py** solves the degenerating 1d wave equation when $a(x) = 16|x-0.5|^2$
- solving inverse problems
  - **control.py** solves the null controllability problem of the 1d wave equation
  - **inverse_problem.py** solves the parameter identification problem of the 1d wave equation

References

[1] Fatiha Alabau-Boussouira, Piermarco Cannarsa, and Günter Leugering. “Control and stabilization of degenerate wave equations”. In: *SIAM Journal on Control and Optimization* 55.3 (2017), pp. 2052–2087.

[2] Muhammad M Almajid and Moataz O Abu-Al-Saud. “Prediction of porous media fluid flow using physics informed neural networks”. In: *Journal of Petroleum Science and Engineering* 208 (2022), p. 109205.

[3] Vladimir L Borsch, Peter I Kogut, and Günter Leugering. “On an initial boundary-value problem for 1D hyperbolic equation with interior degeneracy: series solutions with the continuously differentiable fluxes”. In: *April 2020Journal of Optimization Differential Equations and their Applications* 28.1 (2020), pp. 1–42.

[4] Shengze Cai et al. “Physics-informed neural networks for heat transfer problems”. In: *Journal of Heat Transfer* 143.6 (2021).

[5] *DeepXDE*. 2019. URL: https://deepxde.readthedocs.io/en/latest/.

[6] Carlos J Garcia-Cervera, Mathieu Kessler, and Francisco Periago. “A first step towards controllability of partial differential equations via physics-informed neural networks”. In: ().

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016, p. 164.

[8] Ameya Jagtap et al. “Physics-informed neural networks for inverse problems in supersonic flows”. In: (Feb. 2022).

[9] Weiqi Ji et al. “Stiff-pinn: Physics-informed neural network for stiff chemical kinetics”. In: *The Journal of Physical Chemistry A* 125.36 (2021), pp. 8098–8106.

[10] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).

[11] Peter I Kogut, Olha P Kupenko, and Günter Leugering. “On Boundary Exact Controllability of One-Dimensional Wave Equations with Weak and Strong Interior Degeneration”. In: *arXiv preprint arXiv:2103.08260* (2021).

[12] *NVIDIA Modulus*. URL: https://developer.nvidia.com/modulus.

[13] M. Raissi, P. Perdikaris, and G.E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[14] Chengping Rao, Hao Sun, and Yang Liu. “Physics informed deep learning for computational elastodynamics without labeled data”. In: *arXiv preprint arXiv:2006.08472* (2020).

[15] *The wave equation*. Accessed: 23.06.2022. URL: https://www.uni-muenster.de/imperia/md/content/physik_tp/lectures/ws2016-2017/num_methods_i/wave.pdf.

[16] Peter Zaspel. *Machine Learning Lecture Notes*. Feb. 2022.

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

[18] Ciyou Zhu et al. “Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization”. In: *ACM Transactions on mathematical software (TOMS)* 23.4 (1997), pp. 550–560.