Solutions of inverse problems of the one-dimensional heat conduction equation using Physics Informed Neural Networks

Dr. Andreas H.J. Tewes

March 28, 2024

Contents

List of Figures

1 Introduction

Heat transport can take place in three fundamentally different ways [7]. By radiation, conduction or flow, also known as convection. In the case of radiation, electromagnetic radiation is emitted so that this radiation transport can also take place in a vacuum. Heat flow takes place in fluid media, i.e. gases and liquids, in which mass transport can take place. With their movement, the substances transport heat from one place to another. Heat conduction,finally, only occurs in matter and is not associated with macroscopic movement. Heat conduction takes place through interaction between atoms or molecules. However, this requires local energy and therefore temperature differences. Conversely, local temperature differences can be compensated for by the associated heat transport.
In the context of this work, only heat conduction will be dealt with, although the basic methodology of Physics Informed Neural Networks (PINNs) is in no way limited to this particular case. Furthermore, we will limit ourselves to the consideration of the one-dimensional heat conduction equation. The possible extension to two and three dimensions will be dealt with later in a separate chapter.
In mathematics and physics, we speak of inverse problems when a cause is to be derived from its effect. The opposite, forward or direct problems, on the other hand, involve deducing the effect from the cause. The Earth's gravitational pull or gravity causes the falling (acceleration) of particles with mass in the gravitational field. If the force of gravity (cause) is known, the trajectory of a falling body (effect) can be calculated for given initial conditions. This is a forward problem. A classic example of an inverse problem is reconstructive imaging in the context of computed tomography. The cause, namely the explicitly location-dependent attenuation of the X-rays as they pass through a body, can be derived from their overall attenuation after passing through the entire body (effect). Inverse problems occur frequently in practice and are significantly more difficult to solve mathematically than their direct relatives.
This report is structured as follows. The second chapter first discusses the fundamentals of heat conduction in the form of the heat conduction equation and its mathematical structure. Chapter three introduces the theory of Physics Informed Neural Networks and lays the foundations for their subsequent application to specific questions in the context of this report. Chapter four deals with data acquisition, which was carried out in two different ways in the context of this work. Both simulated and explicitly measured data were used. The focus of this work is on the determination of thermal diffusivity. Using the different data sets, the relevant procedure for the case of constant thermal diffusivity is discussed in Chapter 4 and that of variable thermal diffusivity in Chapter 5. Chapter 6 discusses the possible extension of the methods presented for the two- and three-dimensional heat conduction equation. The report concludes with a summary and an outlook in chapter 7.

2 The one-dimensional heat conduction equation

In the one-dimensional heat conduction equation, we assume a rod of length l whose diameter is negligible as shown in figure 1. The temperature $T(x,t)$ will generally change as a function of both the position x and the time t. This temperature satisfies the following partial differential equation (PDE):

$$C_p\,\rho\,\frac{\partial T}{\partial t}(x,t) = \frac{\partial}{\partial x}\left(\lambda(x)\,\frac{\partial T}{\partial x}(x,t)\right) + q(x,t) \tag{1}$$

Here, $\lambda(x)$ corresponds to the generally location-dependent thermal conductivity and $q(x,t)$ to a thermal power density dependent on location and time. The sign of q determines whether it is a heat source (positive sign) or a heat sink (negative sign). C_p and ρ represents the rod's heat capacity and density respectively. In case of a constant thermal conductivity equation (1) can be further simplified to:

$$C_p\,\rho\,\frac{\partial T}{\partial t}(x,t) = \lambda\,\frac{\partial^2 T}{\partial x^2}(x,t) + q(x,t) \tag{2}$$

Both types of equations are going to be considered in the following chapters. From a mathematical point of view equation (1) represents a parabolic partial differential equation. A detailed derivation of this equation can be found in Appendix A. In this work q can always be understood as an external heat source or, from the point of view of systems theory, as a temporary external pertubation of the overall system. Its relaxation towards a stationary equilibrium is precisely what enables us to obtain information about the quantity we are interested in. Equivalents of equation (1) and (2) in two or three dimensions are considered in chapter 7.

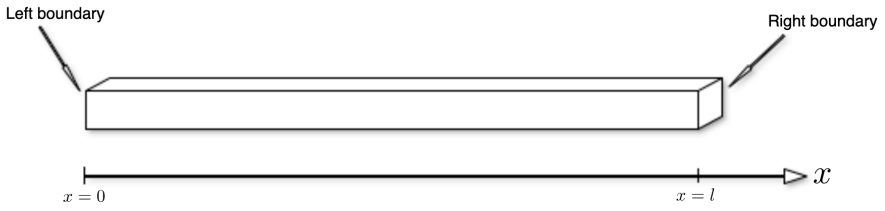


Figure 1: A rod being described by location x. The rod's diameter is supposed to be negligible

3 Introduction to Physics Informed Neural Networks

Physics-informed neural networks, a term introduced by Raissi et al. [20] refers to the use of neural networks, usually a form of multilayer perceptron (MLP), to solve ordinary or partial differential equations. The reference to physics probably results from the examples discussed, most of which originate from the field of physics, although the method itself is by no means restricted to physical problems. Neural networks in general and the multilayer perceptron in particular are used for the functional approximation of solutions of (partial) differential equations (P)DEs. The use of neural networks to solve partial differential equations dates back to the 90s of the last century. Based on the idea of Hopfield networks, the first methods for using neural networks to solve (P)DEs were developed. The (P)DEs themselves were already taken into account here in a type of loss function, still referred to here as energy functionals, which had to be minimised [10,12]. After almost two decades in which, according to figure 2, no significant further developments have taken place, the use of neural networks to solve (P)DEs has become much more important and has attracted more attention again since 2017. This is presumably also due to the fact that powerful frameworks such as Tensorflow or PyTorch [1,17] and powerful GPUs have only been available since around 2015. But why are neural networks particularly suitable function approximators for solving (P)DEs?
This question was also already discussed at the end of the 1990s [9].

The mathematical justification for using MLPs as approximators for solutions of (P)DEs lies in the fact that continuous functions can be approximated with arbitrary accuracy by a multilayer perceptron. More precisely, with:

$$\mathcal{M}(\sigma) := span\left\{\sigma(<\underline{w}|\underline{x}> -\theta) \,|\, \theta \in \mathbb{R},\, \underline{w} \in \mathbb{R}^n\right\} \tag{3}$$

the following theorem [18] holds:

Theorem 1 *Let $\sigma \in \mathcal{C}(\mathbb{R})$. Then $\mathcal{M}(\sigma)$ is dense in $\mathcal{C}(\mathbb{R}^n)$, in the topology of uniform convergence on compacta, if and only if σ is not a polynomial.*

σ is what is known as an *activation function* when it comes to neural network terminology. This is usually a non-polynomial function and this applies also in the context of this work.

MLPs are known to be differentiable in terms of their weights $\underline{w}$ and biases θ. This is utilised in supervised learning in the form of backpropagation. This is a possibly further modified form of gradient descent. The output of the MLP is compared with the target output using a loss function and the latter is then differentiated in relation to the degrees of freedom of the MLP (weights and biases). However, the loss function can be differentiated not only according to the internal degrees of freedom, but also in relation to the input values or neurons. This can also be done up to an arbitrary order. In this way, an appropriately configured MLP can be inserted into a given (P)DE and the deviation from zero

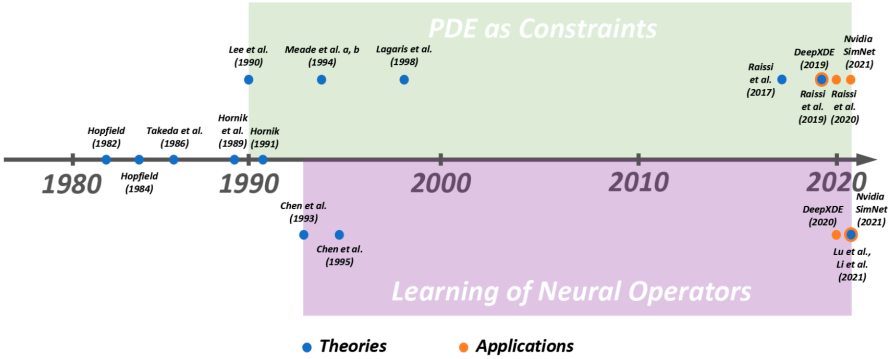


Figure 2: Milestones in the development of theories and applications for solving PDEs using neural networks, taken from [10]

can be interpreted as a loss function if necessary. In this way, the network can be trained to approximately solve a given (P)DE. This will be explained in more detail using an example taken from [11].

Given is Burgers' equation:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2} \tag{4}$$

with a suitable initial condition and Dirichlet boundary conditions. ν is a given constant. Figure 3 shows the representation via MLP.

The loss function is defined as follows:

$$\mathcal{L} = w_{ini}\,\mathcal{L}_{ini} + w_{bnd}\,\mathcal{L}_{bnd} + w_{pde}\,\mathcal{L}_{pde} \tag{5}$$

$$\mathcal{L}_{ini} = \frac{1}{N_{ini}} \sum_{j=1}^{N_{ini}} \left(u(x_j, t = t_{ini}) - u_j\right)^2 \tag{6}$$

$$\mathcal{L}_{bnd} = \frac{1}{N_{bnd}} \sum_{j=1}^{N_{bnd}} \left(u(x_j, t_j) - u_j\right)^2 \quad x_j \in \text{boundary} \tag{7}$$

$$\mathcal{L}_{pde} = \frac{1}{N_{pde}} \sum_{j=1}^{N_{pde}} \left(\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} - \nu \frac{\partial^2 u}{\partial x^2}\right)^2 |_{(x_j, t_j)} \tag{8}$$

It consists of individual loss functions, each of which quantifies deviations between the approximation using MLP and the specified values at the initial time or on the boundary. Furthermore, errors in the fulfilment of Burgers' equation itself are taken into account, by sampling arbitrary values from the entire domain. The influence of the different loss functions can be taken into account using weights. Up to this point, PINNs are merely another form of numerical

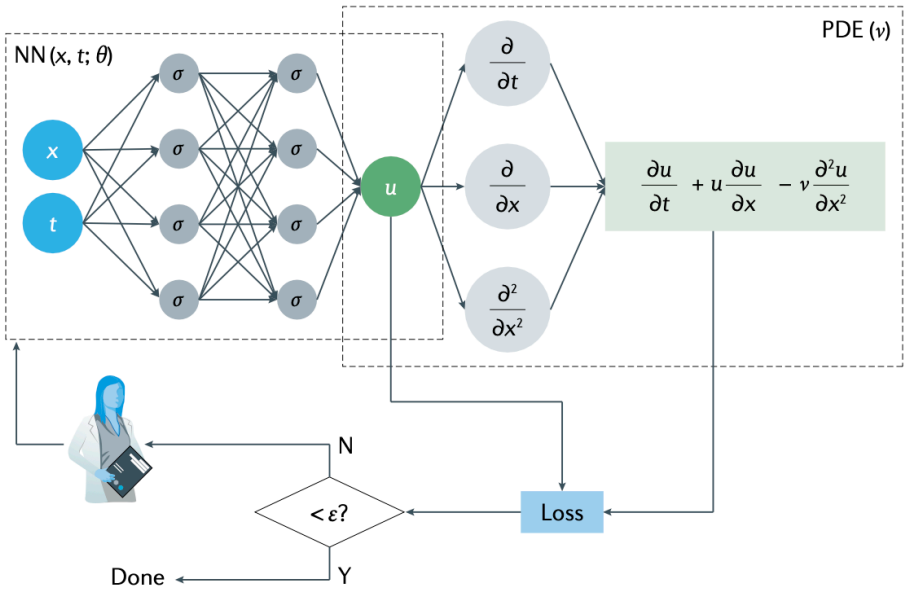


Figure 3: PINN for solving Burgers' equation, taken from [11]

solution of (P)DEs. Their special feature and, in particular, their unique selling point compared to other established methods, such as the Finite Difference Method (FDM) or the Finite Element Method (FEM), is that, analogous to the initial and boundary values, other values from within the entire domain can also be taken into account in the form of an additional loss function. However, this opens up the possibility of incorporating measured values directly into the numerical solution process. In the author's opinion, this is the deeper reason for the term "physics informed" because in this way the theoretical model can be directly coupled to the experiment. For a number of illustrative and impressive examples, please refer to [11, 20–22]. This unique feature of PINNs can also be illustrated using figure 4. PINNs occupy a kind of middle position between problems where machine learning is usually being used and those which can be handled theoretically. In the event that a comprehensive theory is available, e.g. in the form of a (partial) differential equation, and initial and boundary conditions are given, established numerical methods can be used. This corresponds to the left-hand side of the figure. Problems for which there is no theory but a large amount of data are predestined for the application of machine learning methods such as neural networks. Historically, a typical example of this was the development of systems for detecting faces in natural images. These problems are located in the right part of the figure. However, problems for which a theoretical description is available, e.g. in the form of (partial) differential equations, but which still have parameters or even functions that are not known, can

be located here in the centre, provided that measured values/example data are available. We can then use a PINN to solve the (partial) differential equation and determine the unknown parameters or functions in parallel. PINNs therefore offer the possibility of considering theory and data in a joint model.

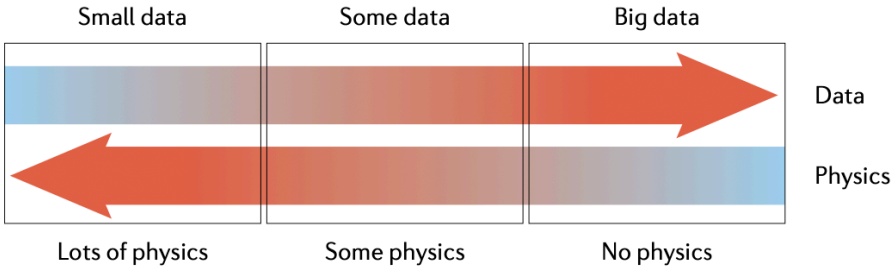


Figure 4: Data versus theory, taken from [11]

According to [11], theoretical knowledge can be taken into account in three different ways. The first is what they call the **observational bias**. Here, physical correlations are implicitly introduced into the learning process via the data used for training. Symmetries of the underlying problem are inevitably reflected in the data. The second way is what they call **inductive bias**. Here, physical relationships and constraints are implemented directly in the structure of the neural network. This form is the strictest but also the most difficult to realise. An example of this can for instance be found in [4]. An analogy in the field of machine learning is, for example, the realisation of so-called convolutional networks, which are designed to be translation-invariant in terms of their architecture. The authors refer to the third and final method with the term **learning bias**. Here, as already shown above using the example, physics is introduced into the training process in the form of loss functions. In this way, a variety of problems can be addressed. These also called soft penalty constraints, however, can only be approximately satisfied. The general idea of **learning bias**, however, was already proposed in the late 1980s as mentioned above.
In this thesis, solutions to inverse problems of the one-dimensional heat conduction equation are considered. Specifically, it deals with the determination of the thermal diffusivity, a material property, based on pure temperature measurements. To what extent are PINNs therefore particularly suitable for solving inverse problems? Solving inverse problems in the context of physics often involves determining the properties of materials or media in relation to other physical quantities. For example, the thermal diffusivity is to be determined as part of this work. Depending on the task, the quantity to be determined can be a scalar, vector or tensor quantity. Furthermore, it can be a constant quantity or a quantity dependent on location, time and possibly other observables. The aim is therefore to determine unknown parameters or functions. The use

of neural networks that already have parameters to be learned (weights and biases) suggests that the additional parameters to be determined, be they constants or functions, should simply be treated as additional weights in general or as an additional part of the neural network itself, which are optimised during the learning process in the very same way as the network's own parameters. The large number of results that have been achieved using PINNs for inverse problems provide further a posteriori justification for their use [5, 6, 11, 20].

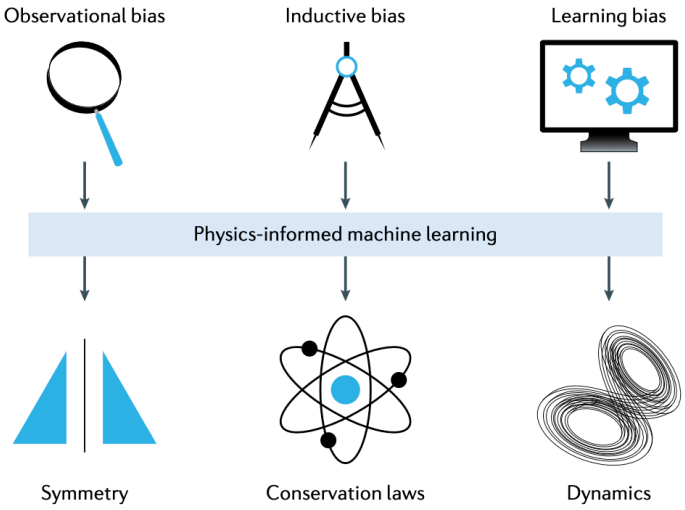


Figure 5: How to embed physics in machine learning, taken from [11]

4 Data acquisition

The data required for training the neural networks was generated in two different ways. On the one hand, simulations were used for this purpose. The simulations were created using the Finite Difference Method. Secondly, real measurements were carried out on a test specimen with the support of the Bundesanstalt für Materialforschung und -prüfung in Berlin. Details of the simulations and the real measurements are described in the following two subsections. Neumann boundary conditions were used throughout this work. This assumption is justified as long as heat dissipation through the boundary is negligible. Furthermore, heat losses due to radiation or convection (see also chapter 2) were also neglected. However, it should be emphasised that both effects could be treated with the same methodology by adding the necessary terms to the corresponding partial differential equation.

4.1 Data simulation

4.1.1 Constant thermal diffusivity

In the case of constant thermal diffusivity, the Crank-Nicolson method [14] was used. In this method, the PDE is discretised as follows: [1]

$$\frac{\partial T}{\partial t}(x,t) = a\,\frac{\partial^2 T}{\partial x^2}(x,t) + q(x,t) \quad \text{with } a = \frac{\lambda}{C_p\,\rho} \tag{9}$$

$$\frac{\partial T}{\partial t}(x_i,t_n) \approx \frac{T_i^{n+1} - T_i^n}{\Delta t} \quad \text{with } T_i^n = T(x_i,t_n) = T(i\,\Delta x, n\,\Delta t)$$

$$\frac{\partial^2 T}{\partial x^2}(x_i,t_n) \approx \frac{T_{i+1}^{n+1} - 2\,T_i^{n+1} + T_{i-1}^{n+1}}{\Delta x^2} \tag{10}$$

$$q_i^n = q(x_i,t_n) = q(i\,\Delta x, n\,\Delta t) \tag{11}$$

By substituting the two discretisations into equation 9, we obtain the following difference equation:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \frac{a}{2\,\Delta x^2}\left(T_{i+1}^{n+1} - 2\,T_i^{n+1} + T_{i-1}^{n+1} + T_{i+1}^{n} - 2\,T_i^{n} + T_{i-1}^{n}\right)$$
$$\ldots \quad + q_i^{n+\frac{1}{2}} \quad \text{with } q_i^{n+\frac{1}{2}} := \frac{1}{2}\left(q_i^n + q_i^{n+1}\right)$$

Here, the second derivative by location was replaced by the arithmetic mean of the corresponding difference equations at time t_n and t_{n+1}. This averaging process makes the corresponding equation unconditionally stable [14]. This means that the step size Δt can be selected independently of the step size Δx without the method becoming numerically unstable. By summarising the constants, we finally obtain

$$-r\,T_{i+1}^{n+1} + (2\,r+2)\,T_i^{n+1} - r\,T_{i-1}^{n+1} = r\,T_{i+1}^{n} + (2-2\,r)\,T_i^{n} + r\,T_{i-1}^{n} + 2\,\Delta t\,q_i^{n+\frac{1}{2}} \tag{12}$$

[1] $q(x,t)$ must not be confused with the expression of the same name in appendix A. This is the $q(x,t)$ there divided by the product of $C_p\,\rho$.

where

$$r = \frac{a\,\Delta t}{\Delta x^2} \tag{13}$$

However, this difference equation has so far only taken the PDE into account. In order to fulfil the boundary conditions, further discretisation is required. The Neumann boundary conditions required in this work are therefore

$$\frac{\partial T}{\partial x}(0,t) = \frac{\partial T}{\partial x}(l,t) = 0 \quad \forall t \geq 0 \tag{14}$$

where l is the length of the rod. We are now using the following Ansatz:

$$\begin{aligned} \alpha\, f(x+h) &= \alpha \left(f(x) + f'(x)\, h + \frac{1}{2} f''(x)\, h^2 \right) + \mathcal{O}(h^3) \\ \beta\, f(x+2\,h) &= \beta \left(f(x) + f'(x)\, 2\,h + \frac{1}{2} f''(x)\, 4\,h^2 \right) + \mathcal{O}(h^3) \end{aligned}$$

which simply corresponds to a Taylor expansion of the function $f(x)$. By addition we get

$$\alpha\, f(x+h) + \beta\, f(x+2\,h) \approx (\alpha+\beta)\, f(x) + (\alpha+2\,\beta)\, f'(x) + (\alpha+4\,\beta)\, \frac{h^2}{2} f''(x)$$

By choosing $\alpha = 2$ and $\beta = -\frac{1}{2}$ and a simple conversion, we finally obtain

$$f'(x) \approx \frac{-3\, f(x) + 4\, f(x+h) - f(x+2\,h)}{2\,h}$$

Applied to the derivatives at the boundary, we obtain the discretisation shown below

$$\begin{aligned} \frac{\partial T}{\partial x}(0,t_n) &\approx \frac{-3\,T(0,t_n) + 4\,T(1\,\Delta x, t_n) - T(2\,\Delta x, t_n)}{2\,\Delta x} \\ \frac{\partial T}{\partial x}(l,t_n) &\approx \frac{3\,T(l,t_n) - 4\,T((I-1)\,\Delta x, t_n) + T((I-2)\,\Delta x, t_n)}{2\,\Delta x} \end{aligned}$$

Here $I\,\Delta x = l$ applies. By setting the equations to zero and solving for the temperatures on the boundary, we finally obtain

$$T_0^n = \frac{4}{3}\, T_1^n - \frac{1}{3} T_2^n \tag{15}$$

$$T_I^n = \frac{4}{3}\, T_{I-1}^n - \frac{1}{3} T_{I-2}^n \tag{16}$$

We have now set up a linear system of equations that allows us to calculate the temperatures along the rod at time t_{n+1} from those at time t_n. This can be

illustrated more clearly using matrices

$$\begin{pmatrix} \frac{2}{3}r+2 & -\frac{2}{3}r & 0 & 0 & \cdots & 0 \\ -r & 2\,r+2 & -r & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & -\frac{2}{3}\,r & \frac{2}{3}r+2 \end{pmatrix} \begin{pmatrix} T_1^{n+1} \\ \vdots \\ T_{I-1}^{n+1} \end{pmatrix} =$$

$$\begin{pmatrix} 2-2\,r & r & 0 & 0 & \cdots & 0 \\ r & 2-2\,r & r & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & r & 2-2\,r \end{pmatrix} \begin{pmatrix} T_1^{n} \\ \vdots \\ T_{I-1}^{n} \end{pmatrix} + \begin{pmatrix} r\,T_0^n + 2\,q_1^{n+\frac{1}{2}}\,\Delta t \\ 2\,q_2^{n+\frac{1}{2}}\,\Delta t \\ \vdots \\ 2\,q_{I-2}^{n+\frac{1}{2}}\,\Delta t \\ r\,T_I^n + 2\,q_{I-1}^{n+\frac{1}{2}}\,\Delta t \end{pmatrix}$$

and

$$\begin{aligned} T_0^{n+1} &= \frac{4}{3}\,T_1^{n+1} - \frac{1}{3}\,T_2^{n+1} \\ T_I^{n+1} &= \frac{4}{3}\,T_{I-1}^{n+1} - \frac{1}{3}\,T_{I-2}^{n+1} \end{aligned}$$

Since the temperature values at time $t = 0$, i.e. T_i^0 for $i = 0 \ldots I$ are given due to the initial condition, the temperatures at each subsequent time step can be calculated. This system of equations can be solved very efficiently with the Thomas algorithm [15] due to the special structure of the coefficient matrix (left side), which has a tridiagonal form. Appendix B shows some snapshots of test cases being used for testing the implementation in python.

4.1.2 Non-uniform thermal diffusivity

In the case of non-uniform thermal diffusivity, a direct method is used in contrast to the Crank-Nicolson method. It should be noted that this method is not unconditionally stable. However, this will be considered later. The heat conduction equation in the case of non-uniform thermal diffusivity can be represented as follows (see also Appendix A):

$$C_p\,\rho\,\frac{\partial T}{\partial t}(x,t) = \frac{\partial}{\partial x}\left(\lambda(x)\,\frac{\partial T}{\partial x}(x,t)\right) + q(x,t) \tag{17}$$

For the sake of simplicity, the simulation was based on $C_p\,\rho = 1$. The PDE is now discretised as follows:

$$\begin{aligned} \frac{\partial T}{\partial t}(x_i,t_n) &\approx \frac{T_i^{n+1}-T_i^n}{\Delta t} \quad \text{with } T_i^n = T(x_i,t_n) \\ &= T(i\,\Delta x, n\,\Delta t) \end{aligned} \tag{18}$$

$$\frac{\partial}{\partial x}\left(\lambda(x_i)\,\frac{\partial T}{\partial x}(x_i,t_n)\right) + q(x_i,t_n) \approx \frac{\lambda_{i+\frac{1}{2}}\,\frac{T_{i+1}^n - T_i^n}{\Delta x} - \lambda_{i-\frac{1}{2}}\,\frac{T_i^n - T_{i-1}^n}{\Delta x}}{\Delta x} + q_i^{n+\frac{1}{2}}$$

assuming

$$\begin{aligned} \lambda_{i\pm\frac{1}{2}} &:= \frac{\lambda(x_i) + \lambda(x_{i\pm1})}{2} = \frac{\lambda_i + \lambda_{i\pm1}}{2} \\ q_i^{n+\frac{1}{2}} &:= \frac{q(x_i, t_n) + q(x_i, t_{n+1})}{2} = \frac{q_i^n + q_i^{n+1}}{2} \end{aligned}$$

Equation (18) can now be solved for the temperature in the next step:

$$T_i^{n+1} = \frac{\Delta t}{\Delta x^2}\left(\lambda_{i+\frac{1}{2}}(T_{i+1}^n - T_i^n) + \lambda_{i-\frac{1}{2}}(T_{i-1}^n - T_i^n)\right) + \Delta t\, q_i^{n+\frac{1}{2}} + T_i^n \tag{19}$$

The boundary values are calculated analogue to the previous case of constant thermal diffusivity. The Neumann stability analysis now results in the following constraint for the choice of Δt depending on Δx and $\lambda_{i+\frac{1}{2}}$:

$$\Delta t \leq \min_i \left(\frac{\Delta x^2}{2\,\lambda_{i+\frac{1}{2}}}\right) \quad \text{see also [19]} \tag{20}$$

Compliance with equation (20) is checked during implementation. In the event of a violation, a corresponding warning is issued. As this is a direct method, the computational effort is lower compared to the Crank-Nicolson method. However, this comes at the cost of limited stability. Appendix C shows some snapshots of test cases being used for testing the implementation in python.

4.2 Measurements

An experimental setup was also used to validate the method presented in this thesis for estimating the thermal diffusivity based on different temperature measurements. This was done in co-operation and with the active support of the Bundesanstalt für Materialforschung und -prüfung in Berlin [2]. The experimental setup is shown in figures 6 and 7. A specimen is thermally heated on one side using a laser. An infrared camera alternately creates a thermal image of the side of the sample facing towards or away from the excitation. Two measurements with a sufficient cooling phase in between are carried out for this purpose. As the laser and camera are synchronised with each other, precise reproducibility is guaranteed. The recordings were made at a recording frequency of 100 Hz. The pixel values are output directly as temperatures. The specimen was excited by a pulse with a pulse length of either 100, 250 or 500 ms. Although various samples were measured for the analysis, only data from one specimen will be analysed in this thesis. The reasons for this decision are discussed in more detail in the corresponding chapter.

[2] My special thanks go to Mr Lecompagnon for making this measurement project possible and for his active support.

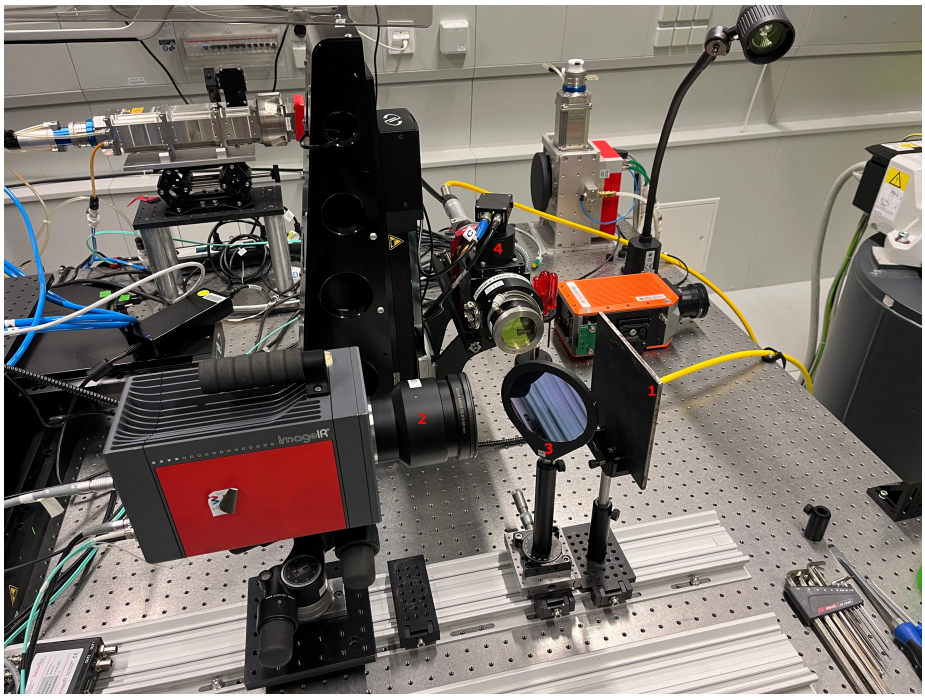


Figure 6: The experimental setup used to measure the temperatures at both sides of the specimen. In this case, the side facing the excitation is analysed. The components used are marked with the numbers 1-4. The test specimen, in this case a steel sheet, is labelled (1). (2) describes the infrared camera with which the surface temperature of the sample can be measured. (3) describes a dichroic mirror that deflects the laser beam (laser is labelled (4)) onto the sample. The mirror is permeable to the thermal radiation recorded by the infrared camera.

5 Estimating constant thermal diffusivity

This chapter presents and discusses the results of both the simulation and the experimental measurements. Details of the simulation and the measurements were presented in the previous chapter.

5.1 Simulated data

The data used in this chapter was generated using the Crank-Nicolson method. Details can be found in chapter 4.1.1. Two different types of excitation were analysed. In the first case, excitation took place at a randomly selected point on the sample. In the second case, the specimen was excited on the right boundary

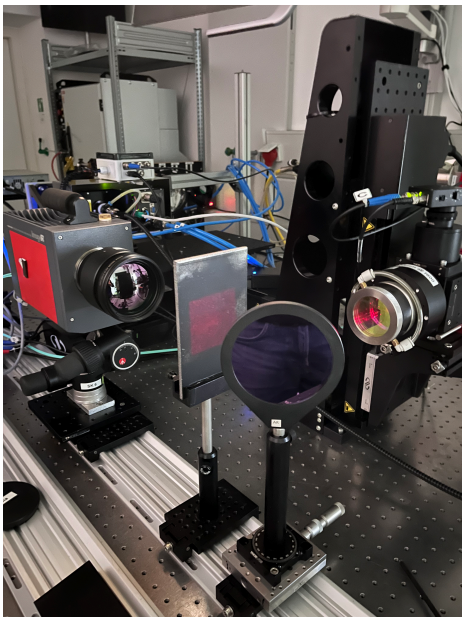


Figure 7: This figure shows the analysis of the side facing away from the excitation. Furthermore, the area being thermally excited by the laser, which has been widened accordingly, can be recognised as a red rectangle on the specimen.

only. The excitation consisted of the product of two Gaussian functions:

$$q(x,t) := A\, e^{-\frac{(x-x_0)^2}{\sigma_x^2}}\; e^{-\frac{(t-t_0)^2}{\sigma_t^2}} \tag{21}$$

The parameters were selected in such a way that only a short-term excitation took place and the amount of $q(x,t)$ was negligible for the further process. Figure 8 illustrates this form of excitation once again.

5.2 Estimating thermal diffusivity using PINN for simulated data

A PINN was used to determine the constant thermal diffusivity. This PINN takes into account the known PDE, including the excitation, except for the unknown thermal diffusivity, which is used as an additional parameter to be "learnt" by the network. Example values for the temperature at different locations and at different times, generated based on the simulation, are used as additional input for the network. The networks topology is shown in figure 9. This is an ordinary multilayer perceptron with three hidden layers and 32 neurons per layer. The hyperbolic tangent was used as an activation function for all layers except the input and output layer. The input layer consists of

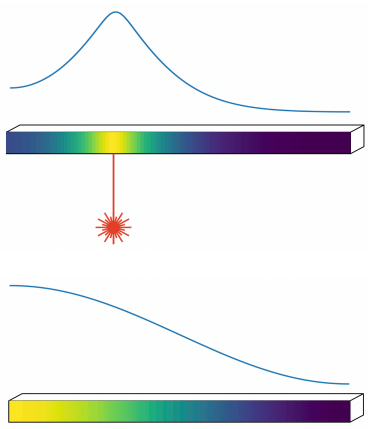


Figure 8: This illustration clearly shows the excitation of the specimen (top) and the state after the excitation has subsided (bottom).

two neurons for the location x and the time t and the output represents the calculated temperature $T(x,t)$. The python framework DeepXDE [13] was used to create and calculate the PINNs. The results for various selected example values for thermal diffusivity are presented in figure 10 and discussed below. The exemplary sample code can be found in Appendix D.

As can be seen in the diagrams in Figure 10, the value for the estimated thermal diffusivity converges always towards the real value actually used in the simulation. The rate of convergence decreases as the value of the thermal diffusivity increases. Furthermore, it can be seen that the error in the evaluation of the PINN is significantly lower for initial and boundary values than for the fulfilment of the PDE or compliance with the measured values. It should also be emphasised that the loss values were not evenly taken into account in the total loss. While a weighting factor of 1.0 was used for the PDE loss, boundary loss and initial loss, the sample loss was strongly weighted with a factor of 25.0.

In order to enable a comparison between the results based on simulated and real measured values, tests were also carried out in which only values on the boundary ($x = 0$ or $x = l$) of the specimen were taken into account during training (see also section 4.2).

The results in Figure 11 show that although only measured values at the boundary were used here, the thermal diffusivity can still be determined using the PINN in good agreement with the true value. In addition, fewer than 30 measured values were taken into account here. This differs from the measurements in Figure 10, where 200 measured values along the entire specimen were taken into account for the training. However, this fact is reflected in the significantly more volatile loss curves. The reduced convergence speed in the third example is striking. This and also the very high volatility in the loss curves

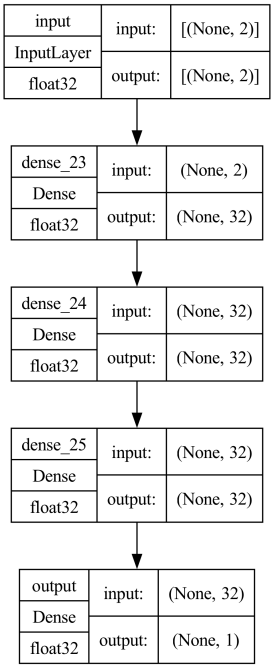
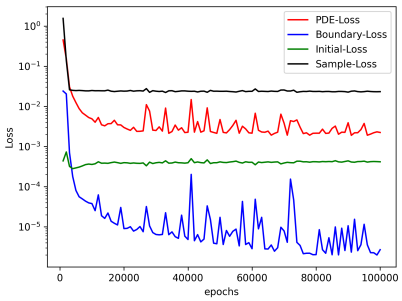


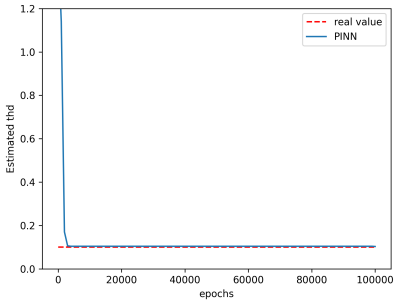
Figure 9: Network topology of PINNs being used for estimating the constant thermal diffusivity. The input of the network consists of the location x and the time t. The output corresponds to the associated temperature T.

is due to the fact that the evaluation was only carried out after the excitation had subsided and the homogeneous heat conduction equation was used for estimating thermal diffusivity. The weighting of the loss values was the same in all three cases. The PDE loss weight was set to 1.0, the boundary loss weight to 5.0 and the sample loss was again weighted more heavily and set to 25.0. The initial values were not taken into account for all 3 test cases.

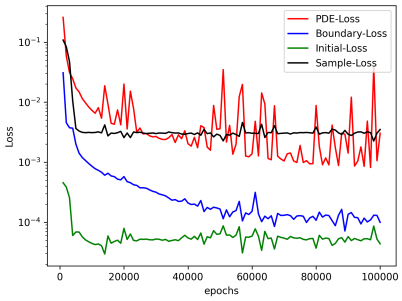
Although the focus of this work is on the determination of the thermal diffusivity, the PINN not only determines this constant but also approximates the whole solution of the underlying PDE. Therefore, the quality of this approximation will also be considered in the following. For this purpose, an excitation function was used that allowed an analytical solution of the heat conduction equation. Furthermore, the PDE was solved numerically using the Crank-Nicolson method to produce values being used for training the PINN. This allowed the analytical solution to be directly compared with the numerical solution of the PINN. The temperature $T(x,t)$ and its partial derivatives $\frac{\partial^2 T}{\partial x^2}(x,t)$ and $\frac{\partial T}{\partial t}(x,t)$ were compared at different points in time. It should be emphasised that the



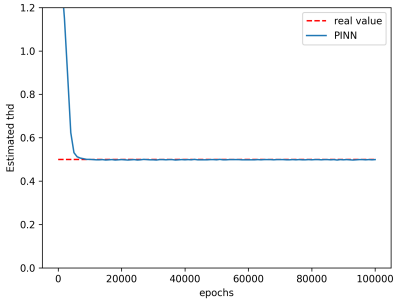
(a) Losses vs. epochs



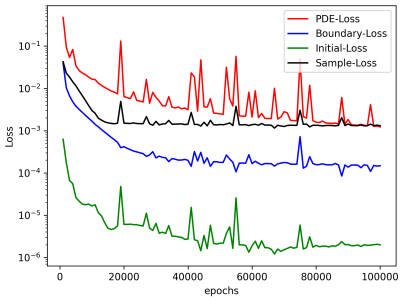
(b) Estimated thd vs. epochs



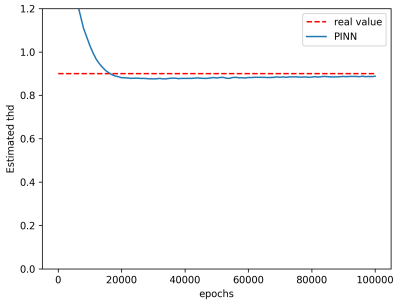
(c) Losses vs. epochs



(d) Estimated thd vs. epochs

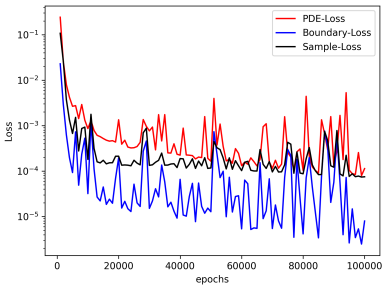


(e) Losses vs. epochs

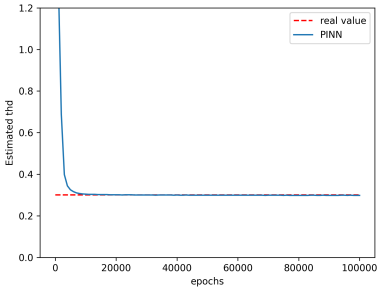


(f) Estimated thd vs. epochs

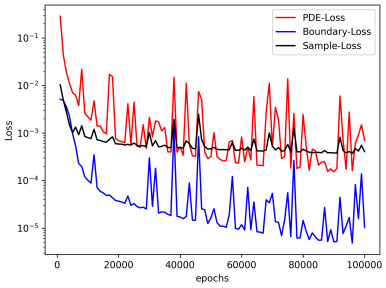
Figure 10: Results for three different values of thermal diffusvity (thd). Losses are shown left, separated by PDE-Loss, loss for boundary values, loss for initial values and loss for sample values. The right-hand side shows the training of thermal diffusivity. The true values of thd are shown as a red dashed line. In all cases, a total of 200 measured values (based on the simulation) were used for training. Specifically, 50 equidistantly selected x-positions at 4 different times were used for training. In all cases, the excitation took place at $x = 0.3$ and and was explicitly taken into account



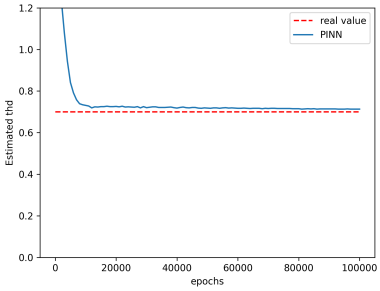
(a) Losses vs. epochs



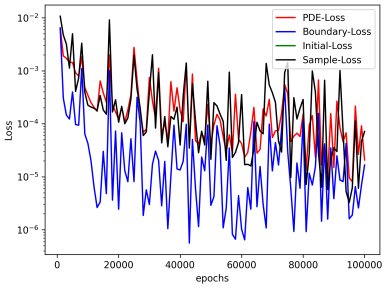
(b) Estimated thd vs. epochs



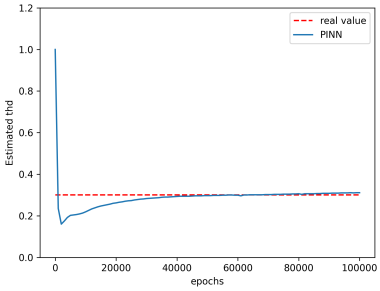
(c) Losses vs. epochs



(d) Estimated thd vs. epochs



(e) Losses vs. epochs



(f) Estimated thd vs. epochs

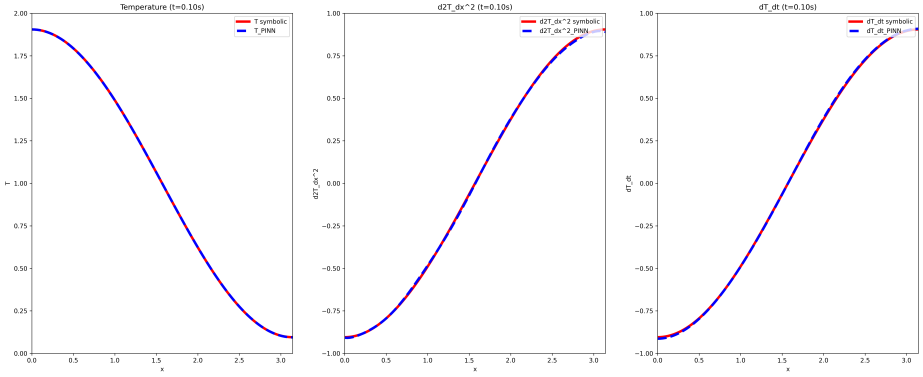
Figure 11: Results for different values of thermal diffusivity (thd). Losses are shown left, separated by PDE-Loss, loss for boundary values and loss for sample values. The right-hand side shows the training of thermal diffusivity. The true values of thd are shown as a red dashed line. In all cases measured values (based on simulation) at the left and right boundary were used. The results in the top row were generated based on 8 different points in time. The excitation took place at $x = 0.3$ and was explicitly taken into account. The results in the centre row were generated based on 6 different points in time. Here, too, the excitation took place at $x = 0.3$ and was explicitly considered. Eventually, the results in the bottom row were generated using 14 different points in time. Here, the excitation took place at the right boundary ($x = 1.0$) and the evaluation started after the excitation decayed

derivatives of the numerical solution were calculated directly using automatic differentiation [3].

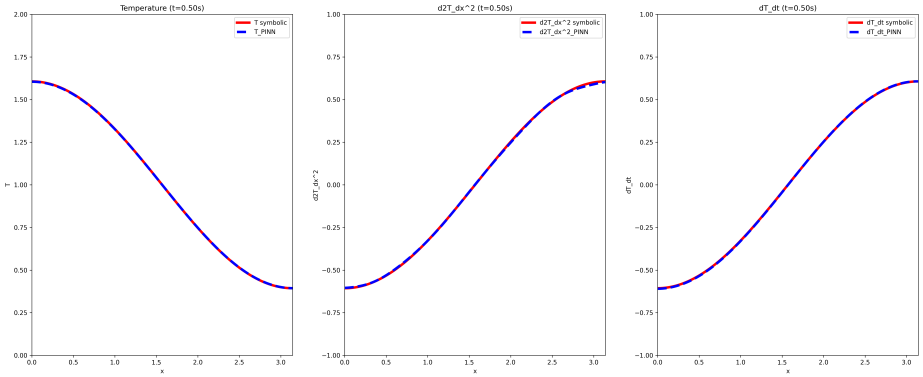
As we can see from Figure 12, there is good agreement between the values calculated using PINN and the true analytical solution. This applies both to the calculated temperature $T(x,t)$ and to the partial derivatives $\frac{\partial^2 T}{\partial x^2}$ and $\frac{\partial T}{\partial t}$. It should be emphasised once again that the derivatives were not calculated numerically but by means of automatic differentiation using the PINN. Here too, only measured boundary values (based on the simulation) for 9 different points in time, i.e. a total of 18 measured values, were used to train the PINN. In the chosen example, the second derivative with respect to x and the derivative with respect to t coincide in the case of the analytical solution.
For the sake of completeness, Figure 13 shows the loss curves and the estimated thermal diffusivity as a function of the training epochs for the comparison between PINN and analytical solution. Once again, the good agreement between the estimated and actual thermal diffusivity is evident.

5.3 Estimating thermal diffusivity using PINN for measured data

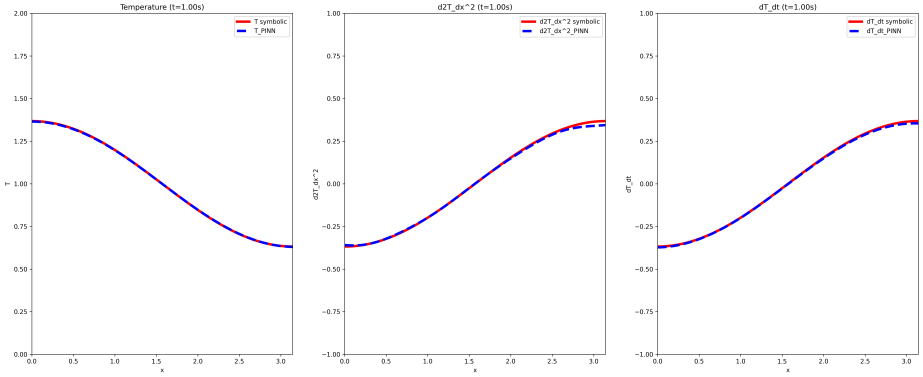
In addition to the training data generated using simulation, real measurements were also carried out in cooperation with the Bundesanstalt für Materialforschung und -prüfung in Berlin. Although the specimens are extended bodies, the heat propagation in the centre of the body can be approximated by a thin rod due to its homogeneity. The experimental setup is described in chapter 4.2. The image from the infrared camera of the area captured by the laser is shown in Figure 14 as an example. The pixel values corresponds to the measured temperature. To reduce noise, a 23x23 window is placed around the centre pixel and the temperature values minus a reference value (image taken shortly before the start of excitation) are arithmetically averaged over this window.
Based on these measurements, a PINN was created analogous to the procedure described in the previous section and the thermal diffusivity was determined. A laser was used to excite one side of the specimen as it is shown again schematically in Figure 15. Several test specimens consisting of different materials were examined. However, the evaluation of the measurement results showed that the assumption of pure thermal conduction is no longer sufficiently justified for the materials with higher thermal diffusivity (see also figure 16). A temperature loss due to convection presumably takes place here, which in principle can also be taken into account in the corresponding PDE even though this was not done in the context of this work.
Figure 17 shows the results using PINN and the estimation using the Laser Flash Method [16] for structural steel. The Laser Flash Method represents a standard for this type of measurement. The PINNs network topology remained the same as that shown in figure 9. The measured values were only recorded after the excitation was switched off. Therefore the homogeneous heat conduction equation



(a) $T(x, 0.1)$, $\frac{\partial^2 T}{\partial x^2}(x, 0.1)$ and $\frac{\partial T}{\partial t}(x, 0.1)$

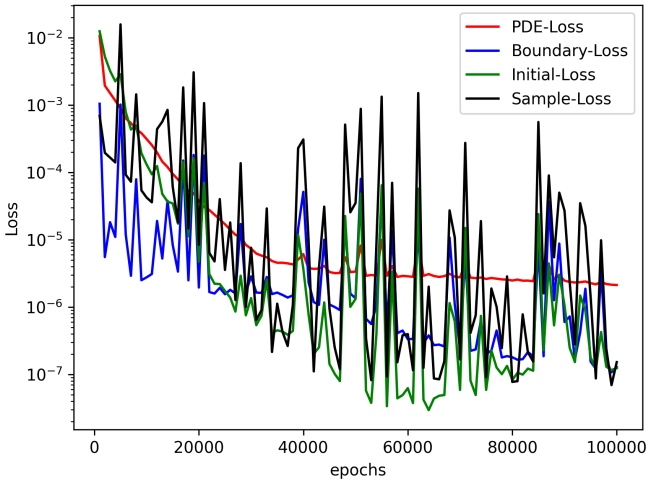


(b) $T(x, 0.5)$, $\frac{\partial^2 T}{\partial x^2}(x, 0.5)$ and $\frac{\partial T}{\partial t}(x, 0.5)$

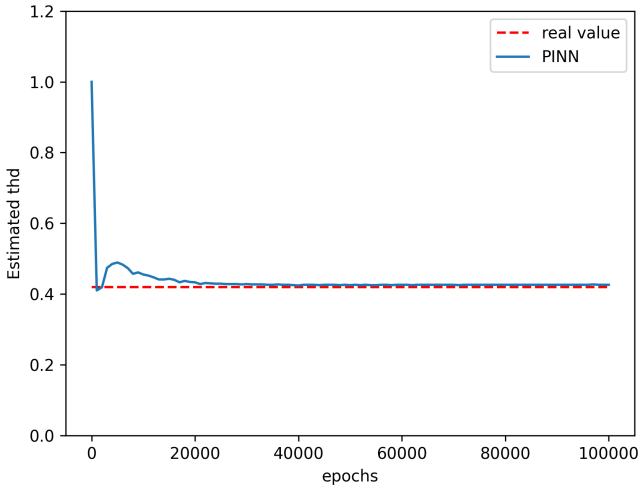


(c) $T(x, 1.0)$, $\frac{\partial^2 T}{\partial x^2}(x, 1.0)$ and $\frac{\partial T}{\partial t}(x, 1.0)$

Figure 12: Comparison between values calculated using PINN for $T(x, t)$, $\frac{\partial^2 T}{\partial x^2}(x, t)$ and $\frac{\partial T}{\partial t}(x, t)$ and the real values based on the analytical solution at different points in time



(a) Losses for PDE (weight=1.0), boundary (weight=5.0), initial (weight=1.0) and sample values (weight=25.0)



(b) Estimated thd vs. epochs

Figure 13: Loss-rates and estimated thermal diffusivity for comparison between PINN and symbolic solution

was used. For this reason, the initial condition could not be taken into account as well. The individual loss functions were weighted and added to a global loss function. PDE loss (2.0), boundary loss (2.0) and sample loss (15.0). In total, temperature measurements were taken into account at 7 different points in time (0s-1.5s) at the boundary after switching off the excitation. This means a total of 14 measuring points.
As the convergence behaviour of the PINN is dependent on the magnitude of the thermal diffusivity, as already observed in chapter 5.2, the measured values were scaled in time so that the expected value for the thermal diffusivity was in the interval $[0, 1]$. This can be justified as follows. Assume that the function $T(x, t)$ solves the heat conduction equation with the constant thermal diffusivity a:

$$\frac{\partial T}{\partial t}(x,t) = a\,\frac{\partial^2 T}{\partial x^2}(x,t) \tag{22}$$

Then the following applies to the function $\tilde{T}(x, t) := T(x, s \cdot t)$:

$$\frac{\partial \tilde{T}}{\partial t}(x,t) = a\,s\,\frac{\partial^2 \tilde{T}}{\partial x^2}(x,t) \tag{23}$$

Therefore, a value $s < 1$, i.e. a stretching of time, results in a reduction of the thermal diffusivity.

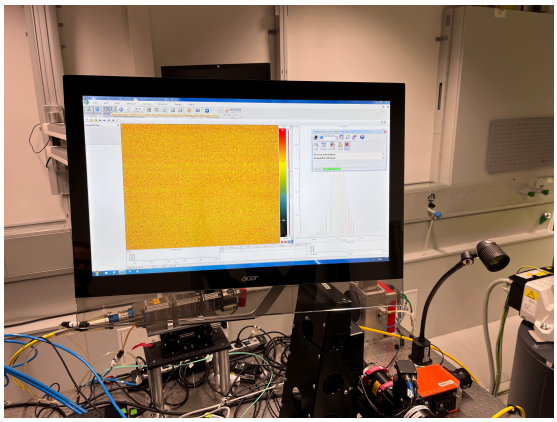


Figure 14: Imaging of the excited area using the infrared camera

The results in Figure 17 show that the PINN is basically suitable for estimating the thermal diffusivity based on comparatively few measured values. This applies at least as long as the assumption of the adiabatic change of state is at least approximately fulfilled. For materials with low thermal diffusivity, this is the case with short measurement times at the temperature range used here. However, as the convergence when training the network decreases with increasing thermal diffusivity, it is advisable, as described above, to adjust the value range of the thermal diffusivity appropriately by scaling the measurement

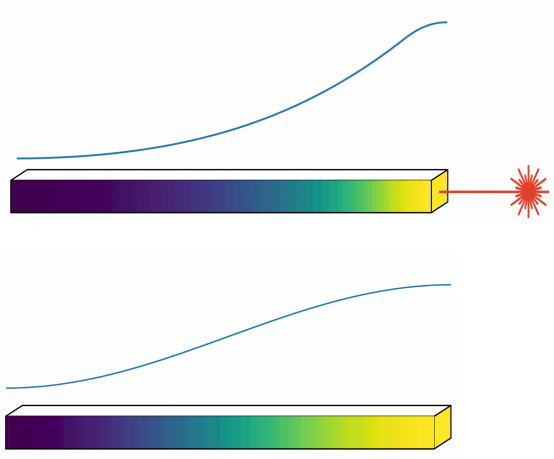


Figure 15: Schematic illustration of the one-sided excitation of the specimen by a laser (top) and after the end of the excitation (bottom).

time. However, as the calculated thermal diffusivity has to be multiplied by the reciprocal of the scaling factor during the back calculation, the accuracy deteriorates as a result, especially for high values. If the PINN is also to be used for higher conductivities, furthermore, deviations from the pure thermal conductivity, such as those caused by convection or radiation, should be explicitly included in the model (PDE).

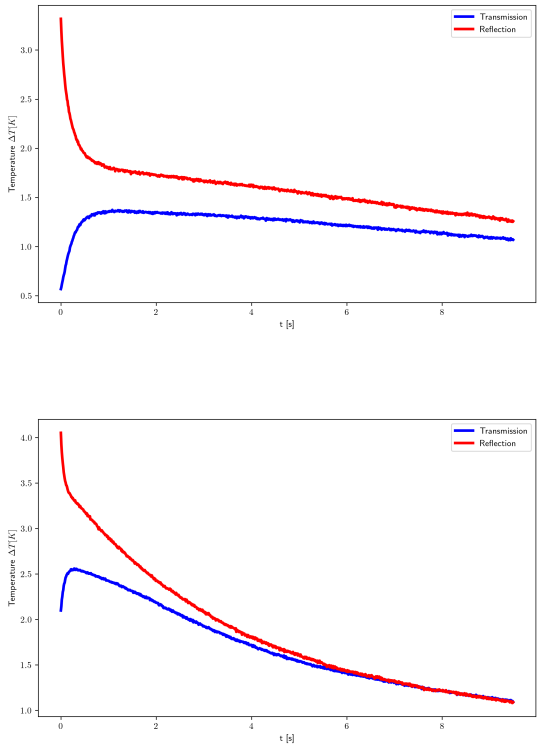
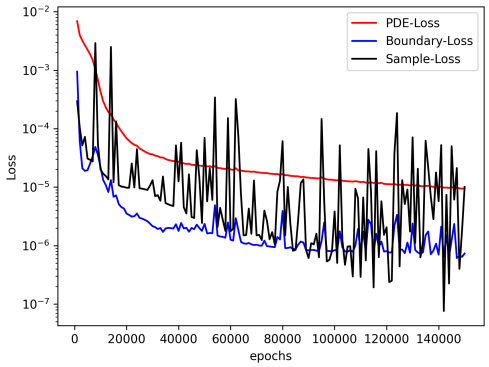


Figure 16: Boundary temperatures as a function of time for the excited side (red) and the side facing away from the excitation (blue). The upper figure shows the results for structural steel, the lower figure those for aluminium. As can be clearly seen, the assumption of an adiabatic change of state is not fulfilled neither for the first nor for the second measurement. For structural steel, however, this assumption can be made at least approximately for the first 1.5 seconds. It should be noted here that the thermal diffusivity of aluminium is almost an order of magnitude higher than that of steel

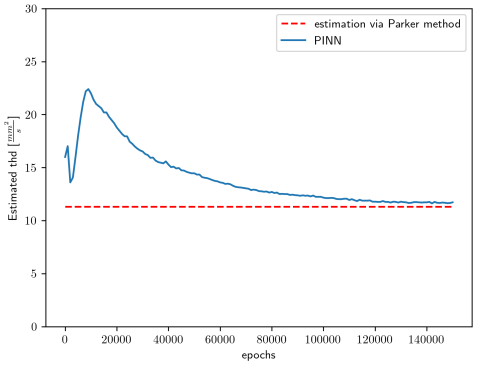
6 Estimating non-uniform thermal diffusivity

Whereas in the last chapter the case of a constant thermal diffusivity was considered, in this chapter an inhomogeneous body will be assumed with regard to the diffusivity. The thermal diffusivity should therefore be an explicit function of the location [3]. In the following, we will therefore consider the more general

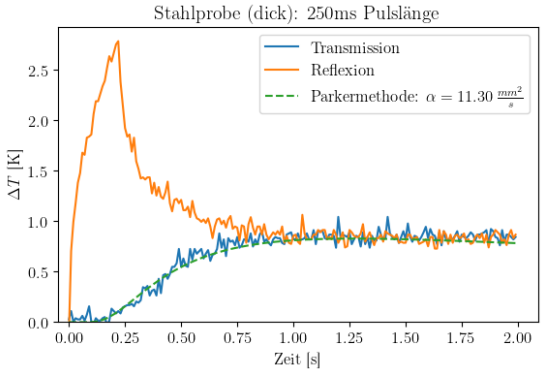
[3] In general, the thermal diffusivity will also depend on the temperature itself. However, due to the rather low excitation, this is not considered here.



(a) Losses vs. epochs



(b) Thermal diffusivity estimated by PINN



(c) Thermal diffusivity (called α here) calculated via Parker Method

Figure 17: Estimation of the thermal diffusivity for structural steel using the PINN and the laser flash method

form of the heat conduction equation:

$$C_p\,\rho\,\frac{\partial T}{\partial t}(x,t) = \frac{\partial}{\partial x}\left(\lambda(x)\,\frac{\partial T}{\partial x}(x,t)\right) + q(x,t) \tag{24}$$

As only a theoretical consideration is to be made in this chapter, the PDE is simplified by setting $C_p\,\rho$ equal to 1. In a real problem, however, this product would also have to be regarded as an unknown function to be "learnt" by the PINN.
A simplified PDE will therefore be assumed in the following analysis:

$$\frac{\partial T}{\partial t}(x,t) = \frac{\partial}{\partial x}\left(\lambda(x)\,\frac{\partial T}{\partial x}(x,t)\right) + q(x,t) \tag{25}$$

In addition to solving the PDE, i.e. estimating the function $T(x,t)$, the PINN must now also be able to describe the function $\lambda(x)$, which will continue to be referred to as thermal diffusivity in the following. Therefore, the network used in the last chapter (see also Figure 9) must be extended. The modified network topology is shown in figure 18. This network now consists of two branches. The left-hand branch corresponds to the structure used so far and is intended to approximate the function $T(x,t)$. This branch therefore has two input neurons for x and t and one output neuron for T. The right-hand branch shares the input neuron for x and has one output neuron, which should correspond to λ. From now on, we will refer to the λ branch or T branch, depending on which part of the PINN is to be considered. First, the topology, i.e. the required number of layers and neurons per layer, of the λ branch was analysed using an example. An example for the function $\lambda(x)$, as used later also in the simulation of the non-uniform heat conduction equation, was used for this purpose. Figure 19 shows the quality of approximation of the λ branch in dependence of the chosen topology. Based on this, the number of hidden layers for the λ branch was set to 2, but the number of neurons was increased to 10. This is intended to increase the safety potential of the approximation.
The PINN training was divided into two stages. In the first step, the T branch of the PINN was trained based on the measured values along the entire body. As before, the measurement data was generated using simulation, but now the finite difference method described in section 4.1.2 was used. In all cases, the excitation took place on the right side for a limited time. In this first step, the weights of the λ branch were set to "non-trainable" and therefore not changed. Only the boundary loss and the sample loss were used for training. The measured values were only recorded after the end of the excitation. The aim is to ensure that the T branch can approximate the temperature as a function of time and location as well as possible beyond the time of the excitation. The λ branch is not taken into account here, as it can only influence the total loss via the PDE loss.
Only when the T-branch can approximate the function $T(x,t)$ with $t > t_{eoe}$ is the λ branch trained in a second step. t_{eoe} is intended to denote the time at which the excitation is switched off. To do this, derivatives of T must be

calculated in relation to t and x. As in the last section, this is also done here using automatic differentiation.
However, in order to avoid calculating higher derivatives, as these would lead to a greater error, the heat conduction equation (see also formula (25)) was first integrated with respect to x. It should be noted that only the period after excitation has been switched off is being considered.

$$\begin{aligned}
\frac{\partial T}{\partial t}(x,t) &= \frac{\partial}{\partial x}\left(\lambda(x)\,\frac{\partial T}{\partial x}(x,t)\right) \\
\Rightarrow \int_0^x \frac{\partial T}{\partial t}(\tilde{x},t)\,d\tilde{x} &= \int_0^x \frac{\partial}{\partial x}\left(\lambda(\tilde{x})\,\frac{\partial T}{\partial x}(\tilde{x},t)\right) d\tilde{x} \\
\Leftrightarrow \int_0^x \frac{\partial T}{\partial t}(\tilde{x},t)\,d\tilde{x} &= \lambda(x)\,\frac{\partial T}{\partial x}(x,t) - \lambda(0)\,\frac{\partial T}{\partial x}(0,t) \\
\Leftrightarrow \int_0^x \frac{\partial T}{\partial t}(\tilde{x},t)\,d\tilde{x} &= \lambda(x)\,\frac{\partial T}{\partial x}(x,t)
\end{aligned}$$

The last transformation results from the Neumann boundary condition. PDE loss is therefore defined as follows:

$$\text{PDE-Loss}(x,t) = \left(\lambda(x)\,\frac{\partial T}{\partial x}(x,t) - \int_0^x \frac{\partial T}{\partial t}(\tilde{x},t)\,d\tilde{x}\right)^2 \tag{26}$$

While the derivatives to x and t are calculated using automatic differentiation as already mentioned, the integral is approximated using the composite trapezoidal rule. Only this loss function is used for training the λ branch.
The Tensorflow framework [2] in version 2.14.0 was used to create and calculate the PINN. The results for various selected example functions for thermal diffusivity are presented in figures 20-25. The exemplary sample code can be found in Appendix E.
In all cases, 1500 (out of $1.17 \cdot 10^6$ totally simulated values) different measured values were used for training the T branch of the PINN. These were randomly drawn from the $[0,1]\times[0.33,1.5]$ x-t-domain. As can be seen from the examples, the PINN is basically able to approximate the course of the thermal diffusivity as a function of the location using the measured temperature values. The deviations in relation to the actual course of the thermal diffusivity are particularly evident in the boundary areas. It should also be noted here that the estimate was not made in the full range of the x-values, i.e. in the interval $[0,1]$, but in the range $[0.1,0.9]$. This increasing deviation can be explained by the Neumann boundary conditions. $\frac{\partial T}{\partial x}$ approaches zero as x does get close to the boundary (see also figure 24). This, however, leads to an decrease in accuracy (see also equation (26)).

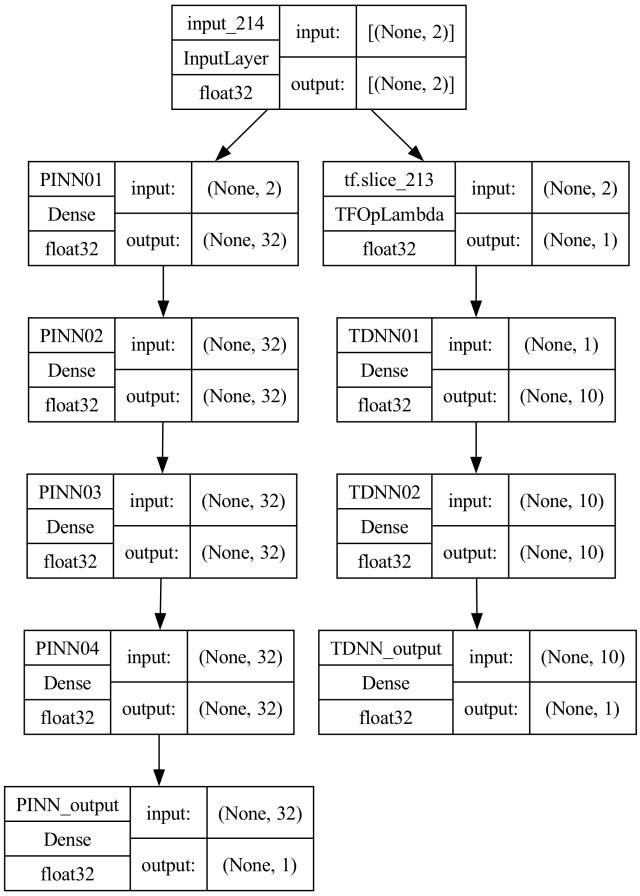


Figure 18: Network topology being used to describe the PINN in case of the non-uniform heat equation (25)

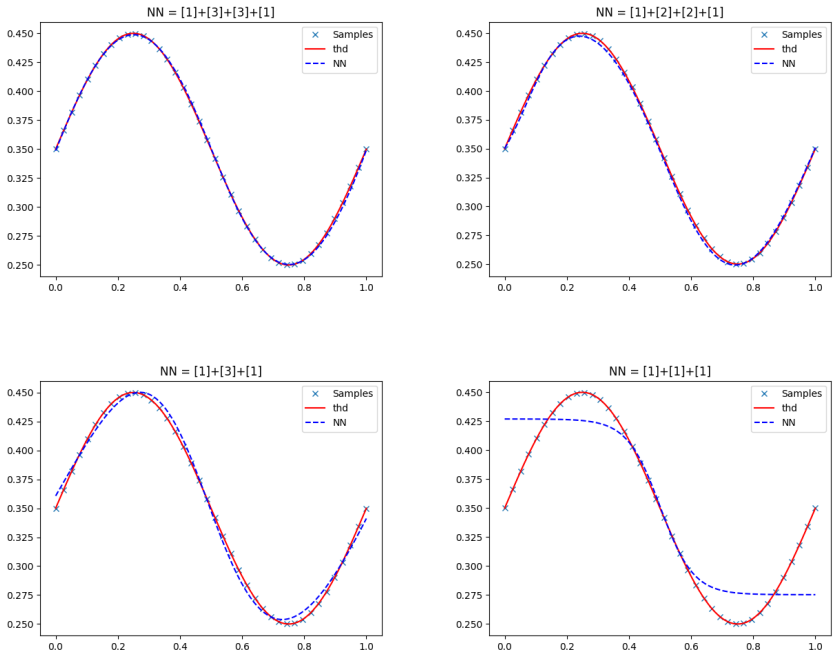
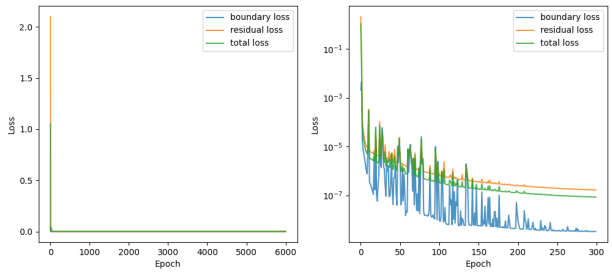
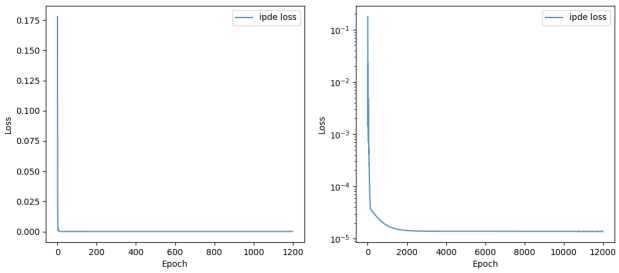


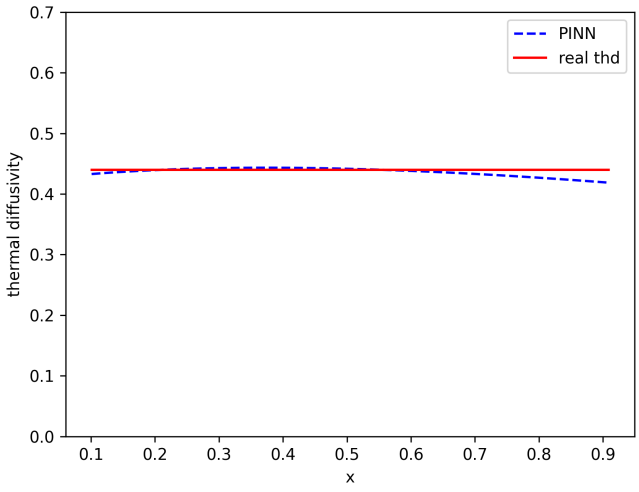
Figure 19: Approximation of a given example function for the thermal diffusivity $\lambda(x)$ depending on the selected topology of the λ branch of the PINN. The number of layers used and neurons per layer are indicated in square brackets. The expression [1] + [3] + [3] + [1] is therefore intended to express that the network has one input neuron, one output neuron and two hidden layers with 3 neurons each.



(a) Sample-Loss (weight=1.0),here also called residual loss, boundary loss (weight=1.4) and total loss for training T branch of PINN for constant (flat) thermal diffusivity. The total loss is a weighted sum of both losses

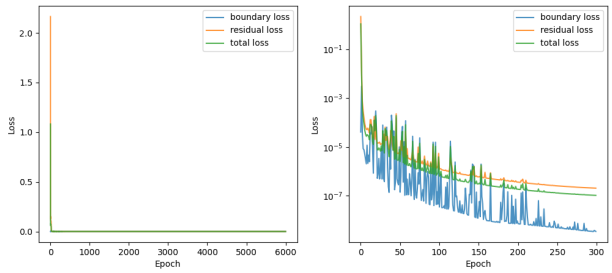


(b) PDE-Loss for the λ branch of the PINN for the constant (flat) thermal diffusivity

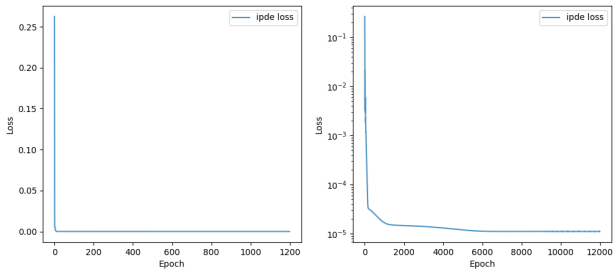


(c) Location-dependent estimate for the constant (flat) thermal diffusivity

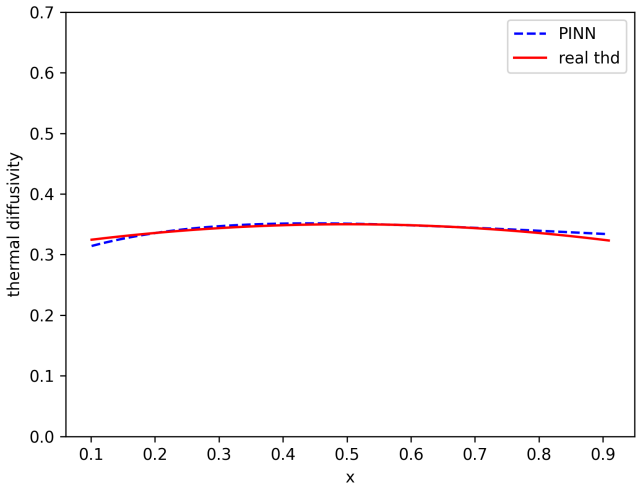
Figure 20: Training results and estimated thermal diffusivity for the case of constant (flat) thermal diffusivity



(a) Sample-Loss (weight=1.0),here also called residual loss, boundary loss (weight=1.4) and total loss for training T branch of PINN for quadratic thermal diffusivity. The total loss is a weighted sum of both losses

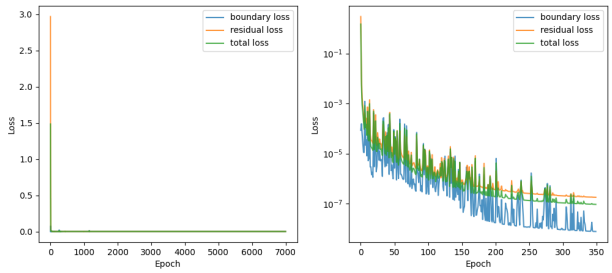


(b) PDE-Loss for the λ branch of the PINN for quadratic thermal diffusivity

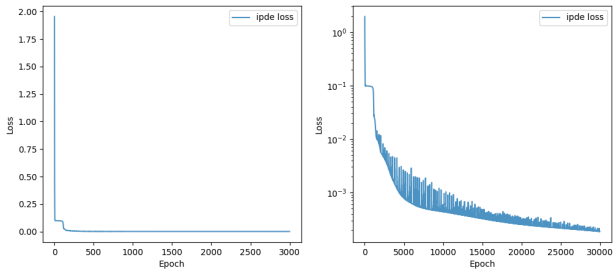


(c) Location-dependent estimate for quadratic thermal diffusivity

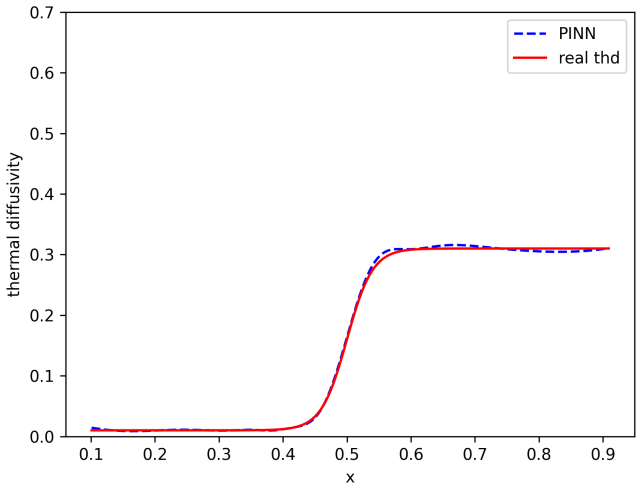
Figure 21: Training results and estimated thermal diffusivity for the case of quadratic thermal diffusivity



(a) Sample-Loss (weight=1.0),here also called residual loss, boundary loss (weight=1.4) and total loss for training T branch of PINN for sigmoid thermal diffusivity. The total loss is a weighted sum of both losses

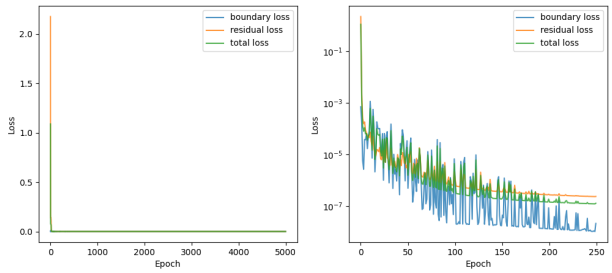


(b) PDE-Loss for the λ branch of the PINN for sigmoid thermal diffusivity

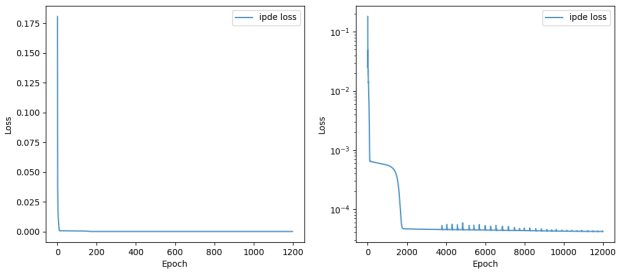


(c) Location-dependent estimate for sigmoid thermal diffusivity

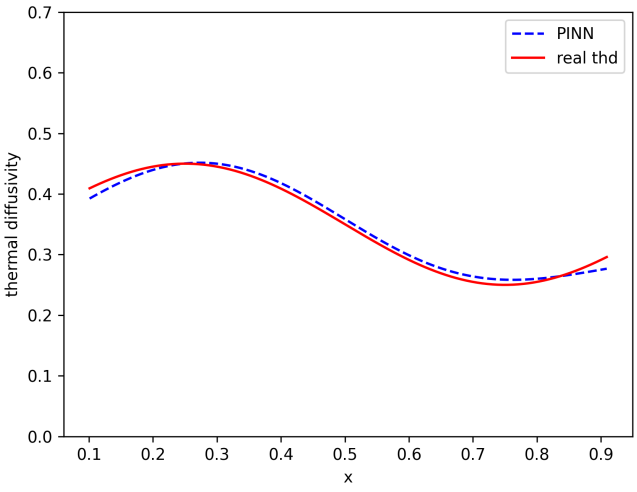
Figure 22: Training results and estimated thermal diffusivity for the case of sigmoid thermal diffusivity



(a) Sample-Loss (weight=1.0),here also called residual loss, boundary loss (weight=1.4) and total loss for training T branch of PINN for sine thermal diffusivity. The total loss is a weighted sum of both losses

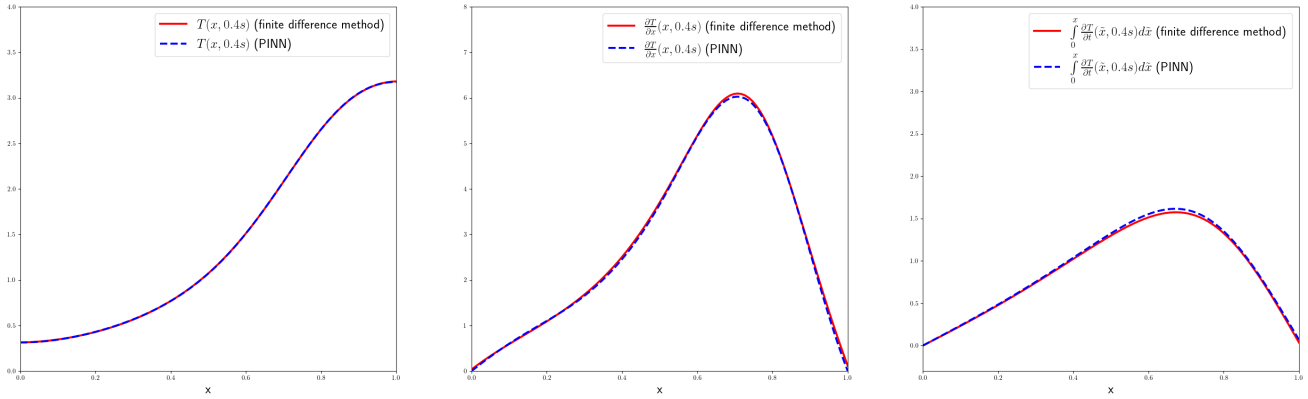


(b) PDE-Loss for the λ branch of the PINN for sine thermal diffusivity

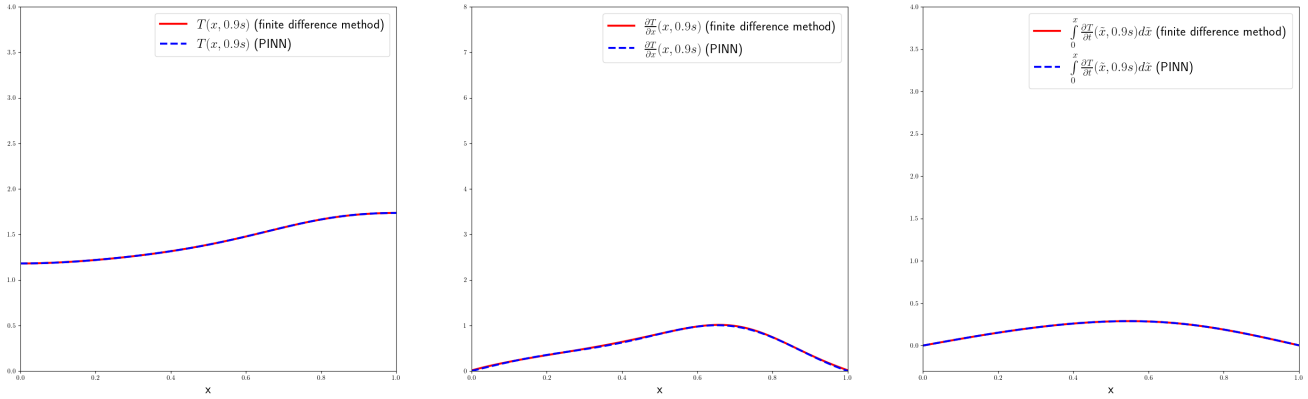


(c) Location-dependent estimate for sine thermal diffusivity

Figure 23: Training results and estimated thermal diffusivity for the case of sine thermal diffusivity



(a) Comparison between reference values by Finite Difference Method and PINN for $t = 0.4s$ and sine thermal diffusivity



(b) Comparison between reference values by Finite Difference Method and PINN for $t = 0.9s$ and sine thermal diffusivity

Figure 24: Comparison for $T(x,t)$, $\frac{\partial T}{\partial x}(x,t)$ and $\int_0^x \frac{\partial T}{\partial t}(\tilde{x},t)d\tilde{x}$ for two different time steps

7 Expansions into higher dimensions

In this chapter, possible extensions of the approaches presented so far to the two- or three-dimensional heat conduction equation will be discussed.
The approach from chapter 5 does not explicitly take the dimension into account. An extension to higher dimensions therefore seems to be obvious. For practical use, however, it would again be necessary to only consider temperature measurements on the boundary (i.e. the edge of the body in two dimensions or the surface in three dimensions). The extent to which the results presented in this work can also be confirmed qualitatively and quantitatively in higher dimensions naturally remains subject to further theoretical and practical analysis. However, the procedure for the non-uniform heat conduction equation, as discussed in chapter 6, is explicitly dimension-dependent. In the following, a possible extension to the two-dimensional problem will be discussed. Although an extension to three dimensions would be possible in principle according to an analogue scheme, it makes little sense from a practical point of view, as temperature measurements inside the body cannot be realised.
A possible extension of the approach presented in chapter 6 will now be outlined for the case of a rectangular plate.

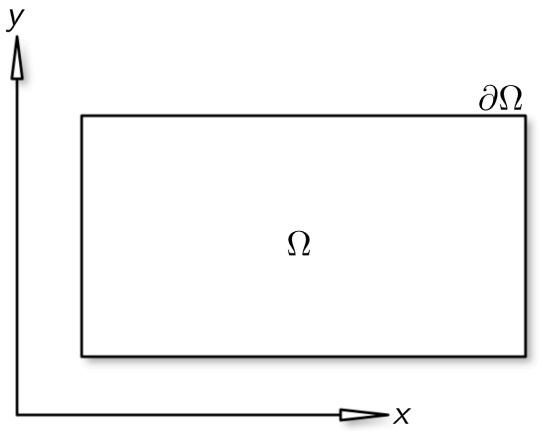


Figure 25: Plate with inner area Ω and boundary $\partial\Omega$.

The extension of the heat conduction equation to three dimensions is basically analogous to the derivation in Appendix A. We obtain:

$$C_p\,\rho\,\frac{\partial T}{\partial t}(x,y,z,t) = \nabla\cdot\Big(\lambda(x,y,z)\,\nabla T(x,y,z,t)\Big) + q(x,y,z,t) \qquad (27)$$

In the two-dimensional case, this can also be represented as follows:

$$C_p\,\rho\,\frac{\partial T}{\partial t}(x,y,t) = \frac{\partial}{\partial x}\left(\lambda(x,y)\,\frac{\partial T}{\partial x}(x,y,t)\right) + \frac{\partial}{\partial y}\left(\lambda(x,y)\,\frac{\partial T}{\partial y}(x,y,t)\right) + q(x,y,t) \tag{28}$$

As described in chapter 6, it should also be assumed here that the measurement only takes place immediately after switching off the excitation. This eliminates the term $q(x,y,t)$ in equation (28). Integration of the equation on both sides with respect to $\tilde{x}$ from 0 to x results:

$$\begin{aligned}\int_0^x C_p\,\rho\,\frac{\partial T}{\partial t}(\tilde{x},y,t)\,d\tilde{x} &= \int_0^x \frac{\partial}{\partial \tilde{x}}\left(\lambda(\tilde{x},y)\,\frac{\partial T}{\partial \tilde{x}}(\tilde{x},y,t)\right)\,d\tilde{x} \\ &\quad + \int_0^x \frac{\partial}{\partial y}\left(\lambda(\tilde{x},y)\,\frac{\partial T}{\partial y}(\tilde{x},y,t)\right)\,d\tilde{x} \\ &= \lambda(x,y)\,\frac{\partial T}{\partial x}(x,y,t) - \lambda(0,y)\,\frac{\partial T}{\partial x}(0,y,t) \\ &\quad + \int_0^x \frac{\partial}{\partial y}\left(\lambda(\tilde{x},y)\,\frac{\partial T}{\partial y}(\tilde{x},y,t)\right)\,d\tilde{x} \\ &= \lambda(x,y)\,\frac{\partial T}{\partial x}(x,y,t) + \frac{\partial}{\partial y}\int_0^x \lambda(\tilde{x},y)\,\frac{\partial T}{\partial y}(\tilde{x},y,t)\,d\tilde{x}\end{aligned} \tag{29}$$

In the last step, the Neumann boundary condition $\frac{\partial T}{\partial \underline{n}}(x,y,t) =< \nabla T|\underline{n} >= 0$ for $(x,y) \in \partial\Omega$ was taken into account. Here, $\underline{n}$ denotes the normal vector on the boundary $\partial\Omega$. Furthermore, the derivative with respect to y was swapped with the integration with respect to $\tilde{x}$ (The mathematical prerequisites necessary for this can be considered to be fulfilled from a physical point of view).
A further integration according to $\tilde{y}$ results:

$$\begin{aligned}\int_0^y \int_0^x C_p\,\rho\,\frac{\partial T}{\partial t}(\tilde{x},\tilde{y},t)\,d\tilde{x}\,d\tilde{y} &= \int_0^y \lambda(x,\tilde{y})\,\frac{\partial T}{\partial x}(x,\tilde{y},t)\,d\tilde{y} \\ &\quad + \int_0^x \lambda(\tilde{x},y)\frac{\partial T}{\partial y}(\tilde{x},y,t)\,d\tilde{x}\end{aligned} \tag{30}$$

The Neumann boundary condition was again used here. A procedure analogous to that in chapter 6 would now first determine the function $T(x,y,t)$ based on the measured values and then use the equation 30 as a loss function to determine $\lambda(x,y)$. The integrals would again be approximated numerically using suitable

quadrature methods. The product $C_p\,\rho$ would also have to be understood as an additional unknown function and "learnt" in parallel. Since both unknowns, $\lambda(x, y)$ and the product $C_p\,\rho$ can only be determined up to a common scaling factor, the value of one of the two functions would have to be fixed at at least one point using an additional loss function.

8 Summary and outlook

In this work, theoretical and practical analyses were carried out to determine the thermal diffusivity using Physics Informed Neural Networks. The constant thermal diffusivity could be reliably determined based on simulated data using measured values along the body but also using only boundary values. This was also confirmed in the real measurement on a test specimen of not too high thermal diffusivity. If this condition was not met, there were clear deviations with regard to an adiabatic change of state, so that the model did not lead to any meaningful results here. Even if this was explicitly not part of this work, it should be noted that these deviations, e.g. due to convection, could in principle also be captured by an extension of the model. Furthermore, the heat conduction equation was also analysed theoretically for the case of non-uniform thermal diffusivity. Under the simplifying assumption that the product of specific heat capacity and density is constant, it was shown that an MLP is also able to describe the functional relationship between thermal diffusivity and location both qualitatively and quantitatively. However, this required measured values along the body.

Based on these results, the next possible investigations are outlined below. Firstly, the estimation of the constant thermal diffusivity can be transferred directly to extended bodies. For this purpose, only the three-dimensional heat conduction equation and measurements of the temperature on the surface have to be used. The actual algorithm would remain unchanged except for the adjustment of the PDE. Further experiments would be required to show that the measured values on the surface are sufficient for an extended body and that the results of the one-dimensional case can therefore be generalised.

The determination of a non-uniform thermal diffusivity or thermal conductivity has already been outlined in this work for the two-dimensional case of a rectangular plate. For this purpose, the one-dimensional approach, the plausibility of which was demonstrated by means of simulated data in this work, was extended to questions of the two-dimensional heat conduction equation. The scenario of a rectangular plate can basically be simulated in an experiment. Here too, however, it is advisable to verify the plausibility of the algorithm in the two-dimensional case by first using simulated data and only then by using measured data.

Finally, the author would like to thank the Bundesanstalt für Materialforschung und -prüfung in Berlin once again for their very collegial support with regard to the experimental measurements.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey, 2015. cite arxiv:1502.05767Comment: 43 pages, 5 figures.

[4] Tom Beucler, Michael Pritchard, Stephan Rasp, Jordan Ott, Pierre Baldi, and Pierre Gentine. Enforcing analytic constraints in neural networks emulating physical systems. *Physical Review Letters*, 126(9), March 2021.

[5] Yuyao Chen, Lu Lu, George Em Karniadakis, and Luca Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Opt. Express*, 28(8):11618–11633, Apr 2020.

[6] Jeremias Garay, Jocelyn Dunstan, Sergio Uribe, and Francisco Sahli Costabal. Physics-informed neural networks for blood flow inverse problems, 2023.

[7] Christian Gerthsen, Hans O. Kneser, and Helmut Vogel. *Physik*, page 214. Springer, Berlin, 1989.

[8] Christian Gerthsen, Hans O. Kneser, and Helmut Vogel. *Physik*, page 197. Springer, Berlin, 1989.

[9] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[10] Shudong Huang, Wentao Feng, Chenwei Tang, and Jiancheng Lv. Partial differential equations meet deep neural networks: A survey. *ArXiv*, abs/2211.05567, 2022.

[11] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6), 5 2021.

[12] Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91:110–131, 11 1990.

[13] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[14] N. Ozisik. *Finite Difference Methods in Heat Transfer*, pages 118–121. Heat Transfer. Taylor & Francis, 1994.

[15] N. Ozisik. *Finite Difference Methods in Heat Transfer*, pages 50–52. Heat Transfer. Taylor & Francis, 1994.

[16] W. J. Parker, R. J. Jenkins, C. P. Butler, and G. L. Abbott. Flash Method of Determining Thermal Diffusivity, Heat Capacity, and Thermal Conductivity. *Journal of Applied Physics*, 32(9):1679–1684, 09 1961.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[18] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[19] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C, Second Edition*, pages 850–851. Cambridge University Press, Cambridge, USA, second edition, 1992.

[20] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[21] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[22] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10566*, 2017.

9 Appendices

9.1 Appendix A

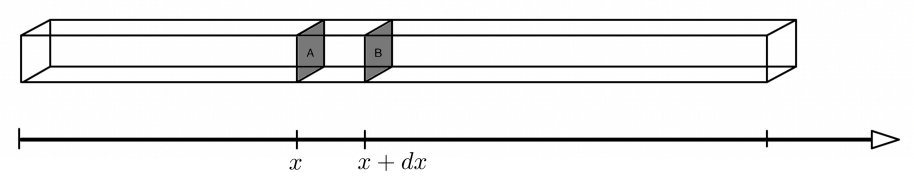


Figure 26: Coordinate system along the rod

Figure 26 schematically shows a rod including the associated coordinate system. A small area of the bar is also marked. For the amount of heat dQ within the "volume" between x and $x + dx$, the following applies:

$$dQ = C_p \, dm \, dT \tag{31}$$

where dQ denotes the change in the amount of heat in the labelled volume, C_p the specific heat capacity, dm the mass and dT the temperature change [8]. The change in the amount of heat within the volume can also be described by the heat flux through the boundary surfaces A and B with the according heat flux densities j_A and j_B at the left and right boundaries respectively [7] and by the generation of heat within the volume q as follows:

$$\begin{aligned}
j_A(x) &= -\lambda(x) \frac{\partial T}{\partial x}(x,t) && (32)\\
j_B(x+dx) &= -\lambda(x+dx) \frac{\partial T}{\partial x}(x+dx,t) && (33)\\
dQ &= j_A(x,t)\, d\sigma \, dt - j_B(x+dx,t)\, d\sigma \, dt + q(x,t)\, dx \, d\sigma \, dt && (34)
\end{aligned}$$

where $d\sigma$ describes the rod's cross-section and dt the time interval. By inserting the expressions for j_A and j_B we then obtain:

$$\begin{aligned}
dQ &= -\lambda(x) \frac{\partial T}{\partial x}(x,t)\, d\sigma \, dt + \lambda(x+dx) \frac{\partial T}{\partial x}(x+dx,t)\, d\sigma \, dt \\
&\quad + q(x,t)\, dx \, d\sigma \, dt \\
&= \frac{\partial T}{\partial x}(x,t) \frac{\lambda(x+dx) - \lambda(x)}{dx} \, dx \, d\sigma \, dt \\
&\quad + \lambda(x+dx) \frac{\frac{\partial T}{\partial x}(x+dx,t) - \frac{\partial T}{\partial x}(x,t)}{dx} \, dx \, d\sigma \, dt \\
&\quad + q(x,t)\, dx \, d\sigma \, dt && (35)
\end{aligned}$$

where the last transformation has been realised by factoring out $\frac{\partial T}{\partial x}$ and $\lambda(x+dx)$ respectively. By equating the expressions from (31) and (35) and taking into account the identity $dV = dx\, d\sigma$, we finally obtain:

$$\begin{aligned} C_p \frac{dm}{dV}\frac{dT}{dt} &= \frac{\partial T}{\partial x}(x,t)\,\frac{\lambda(x+dx)-\lambda(x)}{dx} \\ &\quad +\lambda(x+dx)\,\frac{\frac{\partial T}{\partial x}(x+dx,t)-\frac{\partial T}{\partial x}(x,t)}{dx} + q(x,t) \end{aligned}$$

By now performing the limit $dx \to 0$ and $dt \to 0$ on both sides, and taking into account $\frac{dm}{dV}$ is just the density ρ, we finally obtain:

$$C_p\,\rho\,\frac{\partial T}{\partial t}(x,t) = \frac{\partial T}{\partial x}(x,t)\,\frac{d\lambda}{dx}(x) + \lambda(x)\frac{\partial^2 T}{\partial x^2}(x,t) + q(x,t) \tag{36}$$

or equivalently

$$C_p\,\rho\,\frac{\partial T}{\partial t}(x,t) = \frac{\partial}{\partial x}\left(\lambda(x)\,\frac{\partial T}{\partial x}(x,t)\right) + q(x,t) \tag{37}$$

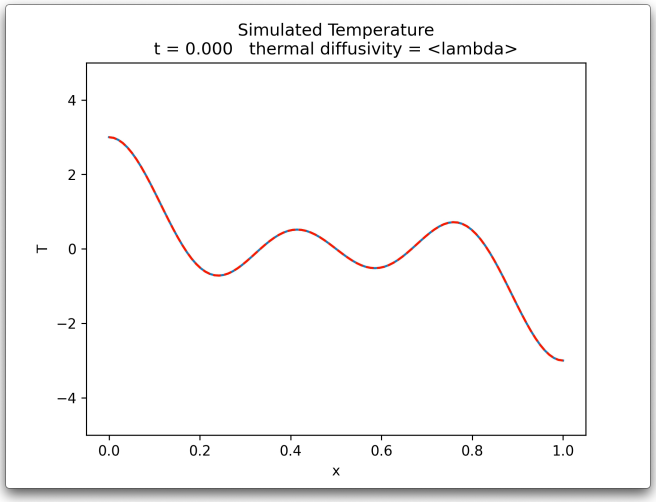
Finally, the last equation can be further simplified to the case of constant thermal conductivity:

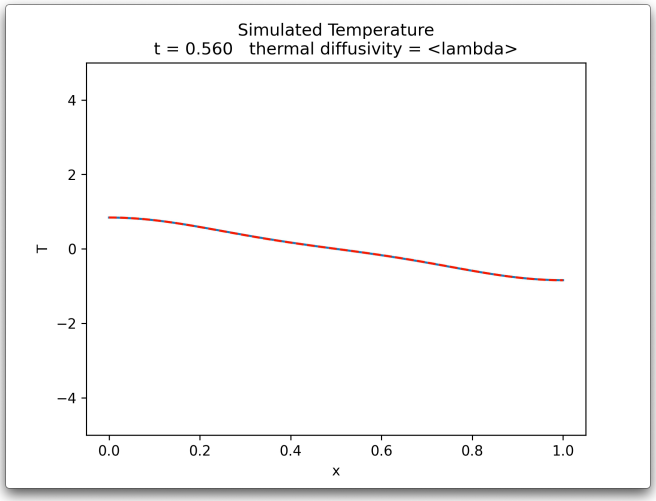
$$C_p\,\rho\,\frac{\partial T}{\partial t}(x,t) = \lambda\,\frac{\partial^2 T}{\partial x^2}(x,t) + q(x,t) \tag{38}$$

9.2 Appendix B

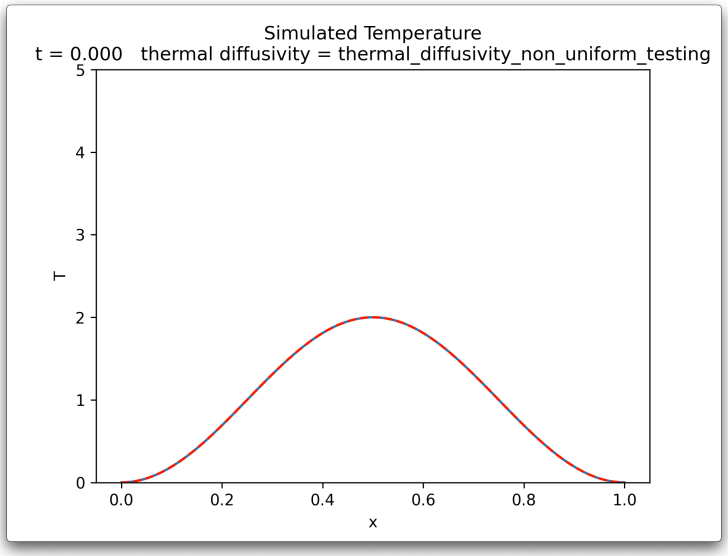
This appendix contains some snapshots from three different test cases of the Python implementation of the Crank-Nicolson method. The blue and dashed red lines show the analytical and numerical solutions respectively. The solutions are shown at different times.

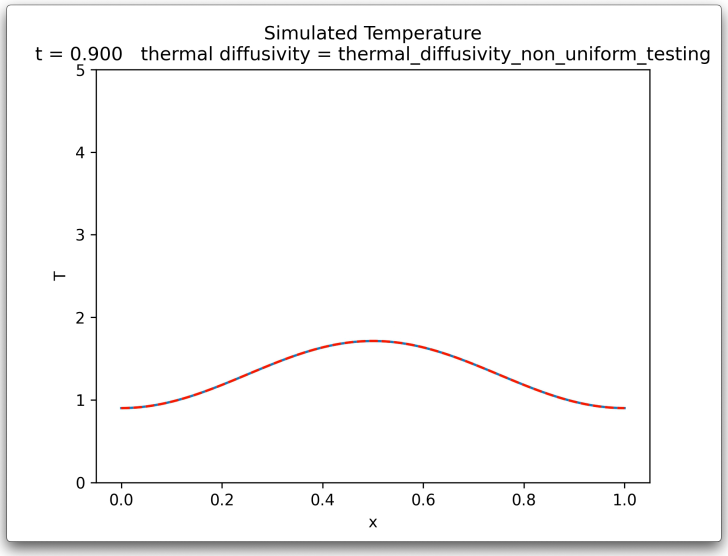
9.2.1 Test case A



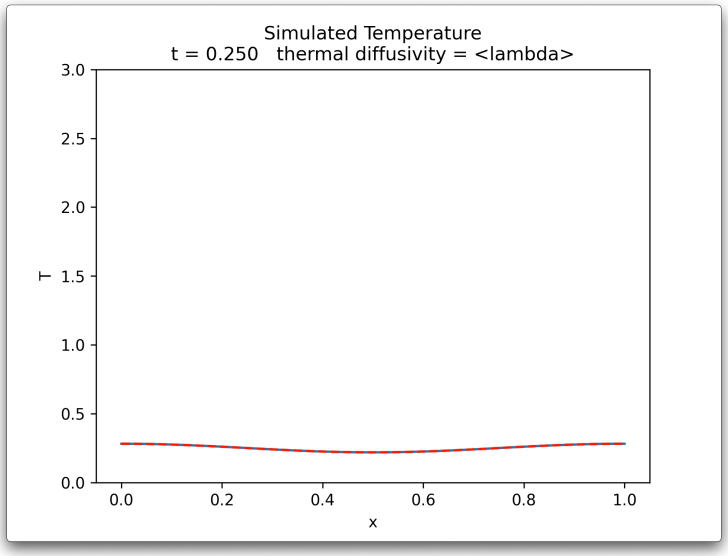


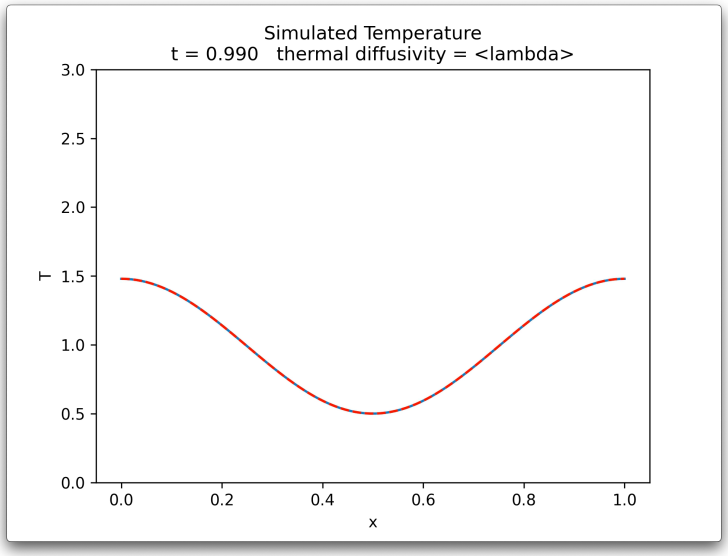
9.2.2 Test case B





9.2.3 Test case C

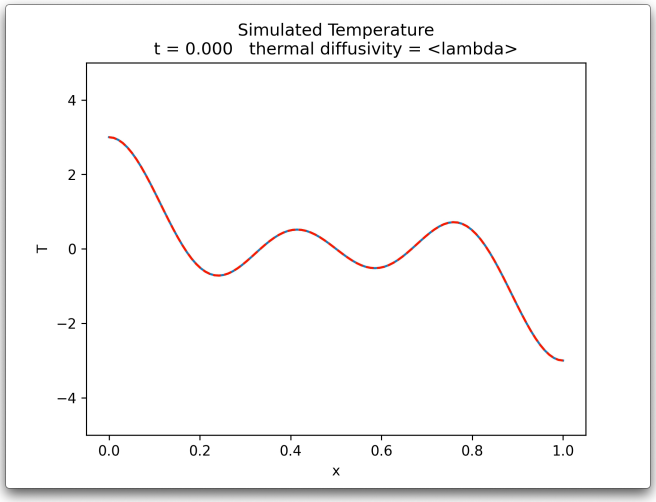


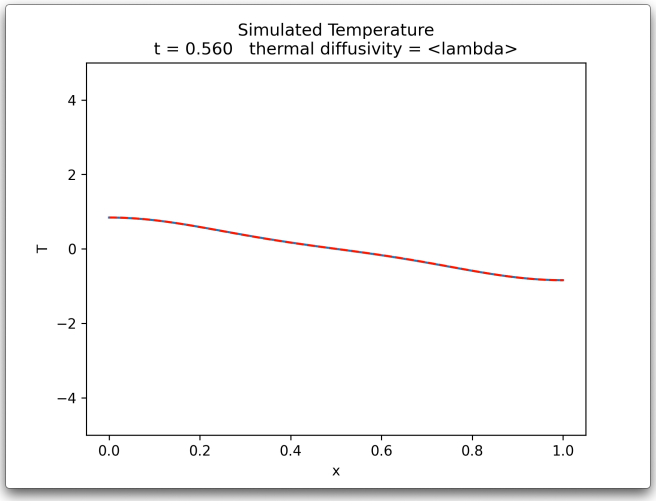


9.3 Appendix C

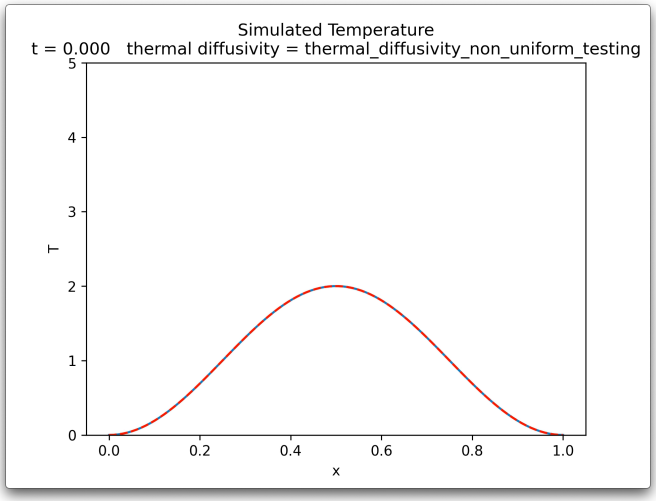
This appendix contains some snapshots from two different test cases of the Python implementation for the non-uniform thermal diffusivity. The blue and dashed red lines show the analytical and numerical solutions respectively. The solutions are shown at different times.

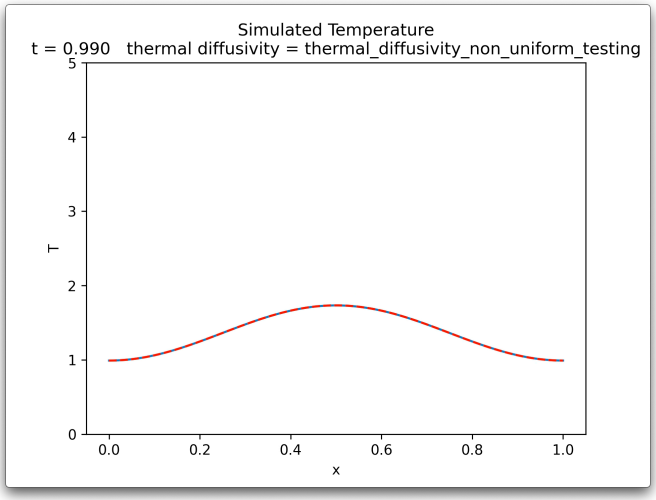
9.3.1 Test case A





9.3.2 Test case B





9.4 Appendix D

Python code for training a PINN to determine the constant thermal diffusivity using the DeepXDE framework:

```
import deepxde as dde
import numpy as np
from deepxde.backend import tf
import matplotlib.pyplot as plt
import pandas

C = dde.Variable(1.5)
# ground truth: C=0.3
length=1.0

def pde(x, y):
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)

    return (
        dy_t
        - C * dy_xx
        - excitation(x[:,0:1],x[:,1:])
    )

def excitation(x,t):
    # pulse strength
    s=100
    ret = s*tf.exp(-(x-0.3)**2/0.001)*tf.exp(-(t-0.15)**2/0.001)
    return ret

def func(x):
    return 0.0*np.cos(x[:,0:1])*np.exp(-x[:,1:])

def boundary_lr(x, on_boundary):
    return ((on_boundary and np.isclose(x[0],0.0) or (on_boundary and
        np.isclose(x[0],1.0))))


geom = dde.geometry.Interval(0, length)
timedomain = dde.geometry.TimeDomain(0, 0.3)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)

bc = dde.icbc.NeumannBC(geom, lambda x: 0.0*x, boundary_lr)
ic = dde.icbc.IC(geomtime, func, lambda _, on_initial: on_initial)

# load measurements
samples = np.load('Values/values.npz')
step_size = 2
```

```
observe_x = np.vstack((samples['x'][0:-1:step_size], \
          np.full(len(samples['x'][0:-1:step_size]),0.10))).T
observe_y = samples['T'][0:-1:step_size,1000].reshape \
          ((len(samples['x'][0:-1:step_size]),1))

observe_x2 = np.vstack((samples['x'][0:-1:step_size], \
           np.full(len(samples['x'][0:-1:step_size]),0.15))).T
observe_y2 = samples['T'][0:-1:step_size,1500].reshape \
           ((len(samples['x'][0:-1:step_size]),1))

observe_x = np.concatenate((observe_x, observe_x2),axis=0)
observe_y = np.concatenate((observe_y, observe_y2),axis=0)

observe_x3 = np.vstack((samples['x'][0:-1:step_size], \
           np.full(len(samples['x'][0:-1:step_size]),0.20))).T
observe_y3 = samples['T'][0:-1:step_size,2000].reshape \
           ((len(samples['x'][0:-1:step_size]),1))

observe_x = np.concatenate((observe_x, observe_x3),axis=0)
observe_y = np.concatenate((observe_y, observe_y3),axis=0)

observe_x4 = np.vstack((samples['x'][0:-1:step_size], \
           np.full(len(samples['x'][0:-1:step_size]),0.25))).T
observe_y4 = samples['T'][0:-1:step_size,2500].reshape \
           ((len(samples['x'][0:-1:step_size]),1))

observe_x = np.concatenate((observe_x, observe_x3),axis=0)
observe_y = np.concatenate((observe_y, observe_y3),axis=0)

observe_y = dde.icbc.PointSetBC(observe_x, observe_y, component=0)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc, ic, observe_y],
    num_domain=400,
    num_boundary=20,
    num_initial=10,
    anchors=observe_x,
    num_test=10000,
)

layer_size = [2] + [32] * 3 + [1]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.nn.FNN(layer_size, activation, initializer)


model = dde.Model(data, net)
```

```
model.compile("adam", lr=0.001, external_trainable_variables=C, \
              loss_weights=[1.0,1.0,1.0,25.0])


variable = dde.callbacks.VariableValue(C, period=1000,
    filename='thd.dat')
variable2 = dde.callbacks.VariableValue(C, period=1000)

losshistory, train_state = model.train(iterations=100000,
    callbacks=[variable,variable2])

dde.saveplot(losshistory, train_state, issave=True, isplot=True)

# finally plot thd as function of epochs
df = pandas.read_csv('thd.dat',header=None, sep=' ')
x = df[0]
y = df[1].str.strip('[]').astype(float)
y_gt = np.full(x.shape[0],0.1)
plt.figure()
plt.ylim(0.0,1.2)
plt.plot(x,y_gt,color='red', linestyle='dashed',label='real value')
plt.plot(x,y,label='PINN')
plt.xlabel('epochs')
plt.ylabel('Estimated thd')
plt.legend(loc="upper right")
plt.savefig('thd.png', format='png', dpi=300)
plt.show()
```

9.5 Appendix E

Python code for training a PINN to determine the non-uniform thermal diffusivity using the Tensorflow framework:

```
from typing import Tuple, List, Union, Callable
import numpy as np
import tensorflow as tf
import tensorflow_probability as tfp
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
from ipywidgets import interact
import ipywidgets as widgets
import IPython
import os
import scipy.optimize

LOSS_TOTAL        = 'loss_total'
LOSS_BOUNDARY     = 'loss_boundary'
LOSS_RESIDUAL     = 'loss_residual'
LOSS_IPDE         = 'loss_ipde'


def create_history_dictionary() -> dict:
    return {
        LOSS_TOTAL: [],
        LOSS_BOUNDARY: [],
        LOSS_RESIDUAL: [],
        LOSS_IPDE: []
    }

def  create_dense_model(num_inputs=2, layers=[[12,10,10,12], [10,10]],
     activation='gelu'):
    # input layer
    xt = tf.keras.layers.Input(shape=(num_inputs,), dtype=tf.float32) #
         input for PINN
    x  = tf.slice(xt, [0,0],[-1,1])                                  # input
         for TDNN

    # hidden layer for PINN (representing T(x,t))
    inp_pinn = xt

    index = 1
    for layer in layers[0]:
        layer_name = 'PINN'+ str(index).zfill(2)
        inp_pinn = tf.keras.layers.Dense(layer, activation=activation,
            kernel_initializer='glorot_normal',name=layer_name)(inp_pinn)
        index+=1

    # output layer for PINN
```

```
    output_pinn = tf.keras.layers.Dense(1, activation='linear',
        name='PINN_output', kernel_initializer='glorot_normal',
        use_bias=False)(inp_pinn)

    # hidden layer for TDNN (representing thermal diffusivity lambda(x))
    inp_tdnn = x

    index=1
    for layer in layers[1]:
        layer_name = 'TDNN' + str(index).zfill(2)
        inp_tdnn = tf.keras.layers.Dense(layer, activation=activation,
            kernel_initializer='glorot_normal',
            name=layer_name)(inp_tdnn)
        index+=1

    # output layer for TDNN
    output_tdnn = tf.keras.layers.Dense(1, activation='linear',
        name='TDNN_output', kernel_initializer='glorot_normal',
        use_bias=False)(inp_tdnn)

    return tf.keras.models.Model(inputs=xt, outputs=[output_pinn,
        output_tdnn])

# set PINN and/or TDNN trainable or non-trainable
def set_PINN_trainable(model, trainable):
    for layer in model.layers:
        if ((layer.name).startswith('PINN')):
            layer.trainable=trainable

def set_TDNN_trainable(model, trainable):
    for layer in model.layers:
        if ((layer.name).startswith('TDNN')):
            layer.trainable=trainable

class HeatModel(tf.keras.Model):
    def __init__(self, backbone, loss_residual_weight=1.0,
        loss_boundary_weight=1.0, loss_ipde_weight=0.0,
        thd_computation=False, **kwargs):
        super(HeatModel, self).__init__( **kwargs)

        self.backbone           = backbone
        self._loss_residual_weight = 1.0
        self._loss_boundary_weight = 1.0
        self._loss_ipde_weight  = 0.0
        self.res_loss           = tf.keras.losses.MeanSquaredError()
        self.bnd_loss           = tf.keras.losses.MeanSquaredError()
        self.ipde_loss          = tf.keras.losses.MeanSquaredError()
        self.thd_computation    = thd_computation
```

```
@tf.function
def call(self, inputs, training=False):

    if (self.thd_computation):
        ############# collect data for estimating thermal diffusivity
            ###############
        xt_ipde = inputs[0]
        xt_int = inputs[1]
        with tf.GradientTape(persistent=True) as tape:
            tape.watch(xt_ipde)
            tape.watch(xt_int)

            u, thd        = self.backbone(xt_ipde, training=training)
            u_int, dummy  = self.backbone(xt_int, training=training)

        grad_domain = tape.batch_jacobian(u, xt_ipde)
        grad_int   = tape.batch_jacobian(u_int, xt_int)

        dT_dx      =
            tf.reshape(grad_domain[...,0],(grad_domain.shape[0],1))
        dT_dt_int  = tf.reshape(grad_int[...,1],(xt_ipde.shape[0],
            num_support_points))


        # estimate integral
        int_dT_dt=tfp.math.trapz(dT_dt_int,
            tf.reshape(xt_int[:,0],(xt_ipde.shape[0],num_support_points)))
        int_dT_dt  = tf.reshape(int_dT_dt, (-1,1))
        int_dT_dt = tf.cast(int_dT_dt,tf.float32)

        ipde = dT_dx * thd - int_dT_dt

        return ipde

    else:

        xt_samples = inputs[0]
        xt_bnd    = inputs[1]

        # determine boundary value via backbone
        with tf.GradientTape(watch_accessed_variables=False) as tape:
            tape.watch(xt_bnd)
            u_bnd , dummy = self.backbone(xt_bnd, training=training)


        first_order = tape.batch_jacobian(u_bnd, xt_bnd)
        du_dx = first_order[...,0]

        # determine sample values via backbone
        u_samples, dummy = self.backbone(xt_samples,
```

```
            training=training)

        return u_samples, du_dx

@tf.function
def train_step(self, data):
    inputs, outputs = data
    u_samples_exact, u_bnd_exact = outputs

    with tf.GradientTape() as tape:
        u_samples, u_bnd = self(inputs, training=True)

        loss_residual = self.res_loss(u_samples_exact, u_samples)
        loss_boundary = self.bnd_loss(u_bnd_exact, u_bnd)

        loss_total =
            1.0/2.0*(self._loss_residual_weight*loss_residual +
            self._loss_boundary_weight*loss_boundary)

    gradients = tape.gradient(loss_total, self.trainable_variables)
    self.optimizer.apply_gradients(zip(gradients,
        self.trainable_variables))

    # update metrics
    return loss_total, loss_boundary, loss_residual

@tf.function
def train_step_thd(self, inputs):

    with tf.GradientTape() as tape:
        ipde = self(inputs, training=True)
        loss_ipde    = tf.math.reduce_mean(tf.math.square(ipde),
            axis=0)[0]


    gradients = tape.gradient(loss_ipde, self.trainable_variables)
    self.optimizer.apply_gradients(zip(gradients,
        self.trainable_variables))

    # update metrics
    return loss_ipde

def fit_custom(self, inputs: List['tf.Tensor'], outputs:
    List['tf.Tensor'], epochs: int, print_every: int = 1000):
    history = create_history_dictionary()

    if (self.thd_computation == False):
        set_TDNN_trainable(backbone, False)
        set_PINN_trainable(backbone, True)
    else:
```

```
            set_TDNN_trainable(backbone, True)
            set_PINN_trainable(backbone, False)

        for epoch in range(epochs):
            if (self.thd_computation == False):
                loss_total, loss_boundary, loss_residual =
                    self.train_step([inputs, outputs])

                history[LOSS_TOTAL].append(loss_total)
                history[LOSS_BOUNDARY].append(loss_boundary)
                history[LOSS_RESIDUAL].append(loss_residual)

                if epoch % print_every == 0:
                    tf.print(f"Epoch {epoch}, Loss Residual:
                        {loss_residual:.2E}, Loss Boundary:
                        {loss_boundary:.2E}, Total Loss:
                        {loss_total:.2E}")

            else:
                loss_ipde = self.train_step_thd(inputs)
                history[LOSS_IPDE].append(loss_ipde)

                if epoch % print_every == 0:
                    tf.print(f"Epoch {epoch}, Loss IPDE: {loss_ipde:.2E}")

        return history

def plot_training_loss_linlog(history, save_path=None, show=True):
    """
        plot training loss with both linear and log y scales

        Args:
            history: The history object returned by the model.fit()
                method.
            save_path: The path to save the plot to.
    """
    plt.figure(figsize=(12, 5), dpi = 100)
    plt.subplot(1, 2, 1)
    plt.xscale("linear")
    plt.yscale("linear")

    if LOSS_BOUNDARY in history:
        if len(history[LOSS_BOUNDARY]) > 0:
            plt.plot(history[LOSS_BOUNDARY][0:-1:10], label='boundary
                loss', alpha=0.8)
    if LOSS_RESIDUAL in history:
        if len(history[LOSS_RESIDUAL]) > 0:
            plt.plot(history[LOSS_RESIDUAL][0:-1:10], label='residual
                loss', alpha=0.8)
    if LOSS_TOTAL in history:
```

```
        if len(history[LOSS_TOTAL]) > 0:
            plt.plot(history[LOSS_TOTAL][0:-1:10], label='total loss',
                alpha = 0.8)
    if LOSS_IPDE in history:
        if len(history[LOSS_IPDE]) > 0:
            plt.plot(history[LOSS_IPDE][0:-1:10], label='ipde loss',
                alpha = 0.8)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.xscale("linear")
    plt.yscale("log")

    if LOSS_BOUNDARY in history:
        if len(history[LOSS_BOUNDARY]) > 0:
            plt.plot(history[LOSS_BOUNDARY][0:-1:200], label='boundary
                loss', alpha=0.8)
    if LOSS_RESIDUAL in history:
        if len(history[LOSS_RESIDUAL]) > 0:
            plt.plot(history[LOSS_RESIDUAL][0:-1:200], label='residual
                loss', alpha=0.8)
    if LOSS_TOTAL in history:
        if len(history[LOSS_TOTAL]) > 0:
            plt.plot(history[LOSS_TOTAL][0:-1:200], label='total loss',
                alpha = 0.8)
    if LOSS_IPDE in history:
        if len(history[LOSS_IPDE][0:-1:200]) > 0:
            plt.plot(history[LOSS_IPDE], label='ipde loss', alpha = 0.8)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    if save_path is not None:
        plt.savefig(save_path)
    if show:
        plt.show()

def thermal_diffusivity_quadratic(x):
    return (-0.16*(x-0.5)**2+0.35)

def thermal_diffusivity_flat(x,td=0.44):
    num = len(x)
    ret = np.full(num, td,dtype=np.float32) # m^2/s
    return ret

def thermal_diffusivity_sine(x):
    return (0.1*np.sin(2*np.pi*x)+0.35)


def plot_thd(x0, x1):
```

```
    x  = np.linspace(x0,x1, 200, dtype=np.float32)
    xx = tf.reshape(x, (-1,1))
    tt = tf.zeros((x.shape[0],1))
    xt = tf.concat((xx,tt), axis=1)

    dummy, thd = backbone(xt)
    plt.figure()
    plt.ylim(0,0.7)
    plt.plot(x,thd,linestyle='--',color='blue',label='PINN')
    plt.plot(x, thermal_diffusivity_quadratic(x),color='red',label='real
        thd')
    plt.legend(loc="upper right")
    plt.xlabel('x')
    plt.ylabel('thermal diffusivity')
    plt.savefig('thd.png', format='png', dpi=300)
    plt.show()


t_start = 4000 #(start index for considering only relaxation back to
    equilibrium beyond excitation)
t_end = 15000
x_begin = 0
x_end = 100
dx = 1e-2     #(step size in x)
dt = 1e-4     #(step size in t)
learn_model = True
learn_thd  = True

# load and create data
samples = np.load('Values/values.npz')
num_sample_points_domain = 1500
num_sample_points_boundary = 200
num_sample_points_ipde   = 50
num_support_points       = 30

idx_x = tf.random.uniform([num_sample_points_domain], minval=x_begin,
    maxval=x_end,dtype=tf.int32)
idx_t = tf.random.uniform([num_sample_points_domain], minval=t_start,
    maxval=t_end,dtype=tf.int32) # (>duration of excitation)

x_domain = tf.reshape(samples['x'][idx_x],[num_sample_points_domain,1])
t_domain = tf.reshape(samples['t'][idx_t],[num_sample_points_domain,1])
T_domain = tf.reshape(samples['T'][idx_x,
    idx_t],[num_sample_points_domain, 1])
xt_domain = tf.concat((x_domain,t_domain),axis=1)

t_boundary = tf.random.uniform((num_sample_points_boundary,1),
    minval=t_start*dt, maxval=t_end*dt, dtype=tf.float32, seed=666)
x_boundary = tf.ones((num_sample_points_boundary//2,1),dtype=tf.float32)
x_boundary = tf.concat([x_boundary,
```

```
    tf.zeros((num_sample_points_boundary//2,1))], axis=0)
x_boundary = tf.random.shuffle(x_boundary, seed=666)
xt_boundary = tf.concat([x_boundary, t_boundary], axis=1)
y_boundary = tf.zeros_like(xt_boundary[:, 1:])


# estimating thd starts at position idx_x0 and ends at idx_x1
idx_x0 = 10
idx_x1 = 90

inputs = [xt_domain, xt_boundary]
outputs = [T_domain, y_boundary]

backbone = create_dense_model(activation='gelu',layers=[[32,32,32,32],
    [10,10]])

##################
# Learn mapping #
##################
if learn_model:
    pinn = HeatModel(backbone, loss_residual_weight=1.0,
        loss_boundary_weight=1.4, loss_ipde_weight=0.0,
        thd_computation=False)

    scheduler = tf.keras.optimizers.schedules.ExponentialDecay(1e-2,
        decay_steps=2000, decay_rate=0.80)
    optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=scheduler)

    pinn.compile(optimizer=optimizer, metrics=[LOSS_TOTAL,
        LOSS_BOUNDARY, LOSS_RESIDUAL])
    history = pinn.fit_custom(inputs, outputs, epochs=60000)
    plot_training_loss_linlog(history, save_path='Losses_network.png')

    backbone.save('model/backbone.tf')

##############
# Learn thd #
##############
if learn_thd:
    backbone = tf.keras.models.load_model('model/backbone.tf')
    scheduler = tf.keras.optimizers.schedules.ExponentialDecay(1e-2,
        decay_steps=1000, decay_rate=0.8)
    optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=scheduler)

    pinn = HeatModel(backbone, loss_residual_weight=0.0,
        loss_boundary_weight=0.0, loss_ipde_weight=1.0,
        thd_computation=True)
    pinn.compile(optimizer=optimizer, metrics=[LOSS_IPDE])

    # time steps for determining thermal diffusivity
```

```
time_steps_idx = np.linspace(t_start,t_end-1,8,dtype=np.int32)

# all positions with a value greater than threshold_factor*maximum
    are considered
threshold_factor = 0.3

# list of training xt-values
list_xt = []

################################################################################
# loop over all time steps and determine interval [x_0,x_1] to train
    network #
################################################################################
for j in time_steps_idx:
    idx_x = np.arange(idx_x0,idx_x1,dtype=int)
    idx_t = np.full(idx_x.shape[0],j)
    x  = tf.reshape(samples['x'][idx_x],[idx_x.shape[0],1])
    t  = tf.reshape(samples['t'][idx_t],[idx_t.shape[0],1])
    xt = tf.concat((x,t),axis=1)

    # sample dT_dx at time step time_steps_idx[j]
    with tf.GradientTape() as tape:
        tape.watch(xt)
        u, _ = backbone(xt, training=False)

    grad_domain = tape.batch_jacobian(u, xt)
    dT_dx      =
        tf.reshape(grad_domain[...,0],(grad_domain.shape[0],1))

    # store xt-values being used finally for training thd
    idx_max = np.argmax(dT_dx)
    i = 1
    l = 1

    while (idx_max-i>0 and
        dT_dx[idx_max-i,0]>threshold_factor*dT_dx[idx_max,0]):
        i+=1

    while(idx_max+l<dT_dx.shape[0] and
        dT_dx[idx_max+l,0]>threshold_factor*dT_dx[idx_max,0]):
        l+=1

    idx_x_lower = idx_max - i
    idx_x_upper = idx_max + l

    idx_xx =
        np.arange(np.max((idx_x0,idx_x_lower)),np.min((idx_x_upper,idx_x1)),dtype=int)
    idx_tt = np.full(idx_xx.shape[0],j)
    xx = tf.reshape(samples['x'][idx_xx],[idx_xx.shape[0],1])
    tt = tf.reshape(samples['t'][idx_tt],[idx_tt.shape[0],1])
```

```
    xt_thd = tf.concat((xx,tt),axis=1)
    list_xt.append(xt_thd)

# update input
xt_ipde = list_xt[0]
for i in range(1,len(list_xt)):
    xt_ipde = tf.concat((xt_ipde,list_xt[i]),axis=0)

num_sample_points_ipde = xt_ipde.shape[0]
xt_ipde = tf.random.shuffle(xt_ipde)

# create support points for integration
xt_int = np.zeros((num_sample_points_ipde, num_support_points, 2))
xt_int[:,:,1] =
    np.reshape(np.repeat(xt_ipde[:,1],num_support_points),(num_sample_points_ipde,
    num_support_points))
xt_int[:,:,0] = np.linspace(np.zeros(num_sample_points_ipde),
    xt_ipde[:,0], num_support_points).T
xt_int = tf.convert_to_tensor(xt_int)
xt_int = tf.reshape(xt_int,
    (num_support_points*num_sample_points_ipde,2))

inputs = [xt_ipde, xt_int]

history = pinn.fit_custom(inputs, [], epochs=12000)
plot_training_loss_linlog(history, save_path='Losses_PINN.png')
plot_thd(samples['x'][idx_x0], samples['x'][idx_x1])
```