Data-Driven Correction Methods for Numerical Solutions of the One-Dimensional Heat Equation

Sindre Stenen Blakseth

Department of Physics — NTNU 2021,
in collaboration with
Department of Mathematical Sciences &
Department of Engineering Cybernetics

Abstract

In this work, the possibility of improving numerical, physics-based simulation schemes using data-driven correction methods is explored. Two correction methods are investigated, both of which employ a neural network as part of the correction process. For the first method, labelled end-to-end learning, the neural network is trained to find a map from uncorrected numerical solutions to more accurate reference solutions. In other words, corrected solutions are given directly as the output of the neural network. For the second correction method, labelled hybrid modelling, the neural network is instead trained to map uncorrected numerical solutions to so-called correction source terms. The correction source terms are added to the discretized governing equation to form a modified system of equations. Solving this modified system of equations then yields the corrected solutions.

The two correction methods are investigated for one-dimensional heat conduction problems solved using the Implicit Euler finite-volume method simulation scheme. Both steady-state and unsteady problems are considered, providing insight into the correction methods' performance in different scenarios. Particular concern is given to their applicability in the context of digital twins. Both methods are found to improve the accuracy of the numerical solutions in some cases, but they fail to do so consistently. In conclusion, neither method is considered satisfactory for digital twin applications in their current form. However, hybrid modelling provides accurate corrections in most of the cases explored, and is considered superior to end-to-end learning. Possible improvements are suggested for both methods, but with emphasis on hybrid modelling.

Preface

The present work is a project report written as part of the Specialization Project in Physics (TFY4510) and submitted to the Department of Physics at the Norwegian University of Science and Technology (NTNU). It is a compulsory part of the Applied Physics profile of the Master of Science study program Applied Physics and Mathematics at NTNU.

For their helpful advice, feedback and suggestions, I extend my gratitude to my supervisors: associate professor Tor Nordam, Department of Physics, NTNU; professor Trond Kvamsdal, Department of Mathematical Sciences, NTNU; and professor Adil Rasheed, Department of Engineering Cybernetics, NTNU. I would like to thank Tor Nordam for his guidance regarding the formalities of the project. To Trond Kvamsdal, I am grateful for suggestions and advice regarding numerical simulations and for the feedback I have received on my writing. Adil Rasheed has also helped improve the quality of my writing, in addition to providing suggestions and advice regarding the machine learning aspect of the project, all of which I greatly appreciate.

Furthermore, I would like to thank Alexandra Metallinou Log for countless fruitful discussions, and for allowing me to adapt one of her figure designs for use in this report. Lastly, I am grateful to Jorid Stenen for assisting me in converting my pen-and-paper sketches into vector graphics.

Trondheim, January 2021
Sindre Stenen Blakseth

Contents

Nomenclature

Abbreviations

1D	One-Dimensional
BC	Boundary Condition
DNN	Deep Neural Network
FCNN	Fully Connected Neural Network
FLOPs	Floating Point Operations
FVM	Finite-Volume Method
IC	Initial Condition
ML	Machine Learning
MSE	Mean Squared Error
NN	Neural Network
ReLU	Rectified Linear Unit
TDMA	Tri-Diagonal Matrix Algorithm

Scalars and scalar fields

α	Thermal diffusivity,	m^2/s
Δt	Time step in numerical simulations,	s
Δx	Length of control volume/grid cell,	m
$\hat{q}$	Heat generation rate per unit volume,	J
μ_i	Chemical potential of substance i,	J
Ω	Control volume/grid cell,	m^3
ρ	Density,	kgm^{-3}
ζ	Grid size reduction factor,	1

A	Area,	m^2
c_i	Concentration of substance i,	$1/\mathrm{m}^3$
C_V	Heat capacity at constant volume,	J/K
c_V	Specific heat capacity at constant volume,	J/Kkg
E	Normalized error,	1
k	Conductivity,	W/Km
m	Mass,	kg
N_i	Mole number of substance i,	mol
N_j	Number of grid cells in numerical simulation,	1
N_t	Number of time levels in numerical simulation,	1
p_t	Temporal order of convergence,	1
p_x	Spatial order of convergence,	1
Q	Heat,	J
S	Entropy,	J/Kkg
s	Entropy per unit volume,	$\mathrm{J/Kkgm}^3$
T	Temperature,	K
t	Time,	s
U	Internal energy,	J
u	Internal energy per unit volume,	$\mathrm{J/m}^3$
V	Volume,	m^3
W	Work,	J
x	Position along 1st coordinate axis,	m
y	Position along 2nd coordinate axis,	m
z	Position along 3rd coordinate axis,	m

Vectors and matrices

$\mathbb{A}$	Coefficient matrix for linear system of equations,	1
$\hat{\boldsymbol{\sigma}}$	Correction source term,	$\mathrm{K\,s^{-1}}$
$\boldsymbol{\sigma}$	Discretized source term,	$\mathrm{K\,s^{-1}}$
$\boldsymbol{b}$	Right hand side of linear system of equations,	K
$\boldsymbol{n}$	Surface normal,	m
$\boldsymbol{q}$	Heat flux,	J
$\boldsymbol{R}$	Right hand side of ordinary differential equation,	$\mathrm{K\,s^{-1}}$
$\boldsymbol{r}$	Position,	m
$\boldsymbol{T}$	Discrete temperature profile,	K

Subscripts

a	Left domain boundary
b	Right domain boundary
e	Reference control volume's right boundary
j	Grid node index
$j+1/2$	Cell face index
w	Reference control volume's left boundary
c	Corrected
E	Center of reference control volume's right neighbour
exact	Exact, analytic expression
num	Numerical
P	Center of reference control volume
ref	Reference
u	Uncorrected
W	Center of reference control volume's left neighbour

Superscripts

∞	At steady state
n	Time level

Modifiers

$\bar{\ }$	Cell-averaged quantity

Sets

$\mathbb{R}$	The set of real numbers

Machine Learning Notation

α	Negative slope of LeakyReLU
η	Learning rate
$\hat{m}$	Bias-corrected estimator of gradient mean
$\hat{v}$	Bias-corrected estimator of gradient variance
$\hat{y}$	Target output
$\mathbb{W}$	Weight matrix
θ	Neural network parameter
$\boldsymbol{b}$	Bias vector
$\boldsymbol{w}$	Weight vector
$\boldsymbol{x}$	Input vector
$\boldsymbol{z}$	Output vector
a	Activated neuron output
b	Bias
C	Loss function
d	Dimensionality of neural network output
f	Activation function
M	Number of neurons in layer

N	Number of layers in network
y	Neural network output
z	Neuron output
E	Experience
P	Performance measure
T	Class of tasks

1. Introduction

1.1. Background and Motivation

Digital twins represent a paradigm shift within fields such as product development, verification and certification processes and management of assets [1, 2, 3]. Broadly defined, a digital twin refers to any digital representation of some real system. Such a system can be anything from a single object, e.g. an aeroplane component, to complex and dynamic biological systems [4]. A digital twin should incorporate as much information about its real-world counterpart as possible, and also evolve along with the real system. Furthermore, a digital twin should cover the *entire* life-cycle of the real system, not just its behaviour in a few selected situations. Lastly, it is not sufficient for a digital twin to merely *describe* the real system; it must also be able to *predict* the behaviour of the real system under different conditions [2, 5]. This last requirement makes digital twins useful in a wide variety of applications, including product lifetime management [3], vehicle design [4, 6] and industrial processes [7].

In the context of digital twins and related technologies, there is a need for simulation models which are

1. computationally efficient, such that they can operate in real time;
2. generalizable, such that they can handle changes in operating conditions (e.g. changes in the environment surrounding the real system);
3. self-evolving, in the sense that they are able to gather and learn from experience during continuous operation of the real system;
4. trustworthy, in the sense that their predictions are interpretable and satisfy relevant stability conditions.

One approach to the numerical simulation of a system is based on deriving a set of governing equation for the system using known physics and assumptions about the system's behaviour. These equations are then discretized, and lastly, the discretized equations are solved using a numerical solver. Such models, which we refer to as physics-based models, can be used to obtain trustworthy results due to the existence of well-developed error analysis methods. Examples of such analysis methods include Fourier analysis, normal mode analysis and the modified equation approach [8]. The two former methods can be used to derive stability conditions for a particular simulation scheme, while the latter is used to ensure that a scheme is consistent.

Physics-based simulation models are only as good as their underlying assumptions, and this can be both a strength and a weakness. It is a strength in the sense that, as long as its assumptions are valid, a physics-based model generalizes perfectly to changed operating conditions. However, if changes in the operating conditions cause some of its assumptions to be violated, the model is no longer valid. Thus, physics-based models do not generalize to situations where their underlying assumptions no longer hold.

Determining the validity of a physics-based simulation model's underlying assumptions can be difficult. For instance, it might be infeasible to accurately measure all relevant parameters due to inaccessibility of the real system. Another case where the validation of assumptions can pose problems is when the overall behaviour of the system depends sensitively on its initial conditions. This is the case for e.g. flow in pipelines with discontinuous cross-sections, where it can be difficult to predict the presence of so-called "resonant" waves *a priori*. For certain classes of numerical simulation schemes, the appropriate solution method depends on the number of waves present in the system [9], thereby making it difficult to *a priori* establish the validity of numerical simulations based on such schemes.

Another weakness of physics-based models is that they can be computationally costly. Consider for example systems experiencing turbulent flow, where the smallest and largest relevant spatial scales differ by more than six orders of magnitude [10]. So-called direct numerical simulations aim to fully resolve even the smallest spatial scales, and therefore require finely discretized grids, which in turn yields high computational complexity. This motivates the use of GPU clusters for such simulations [11]. However, in certain cases such as e.g. atmospheric simulations, not even modern supercomputers provide the computing power necessary to fully resolve all relevant physical processes [12].

In the present work, we examine the possibility of using data-driven machine learning (ML) techniques to overcome the weaknesses of purely physics-based models, as discussed above. This is inspired by recent achievements in applying data-driven models to physical problems. For example, physics-informed neural networks have been used to solve nonlinear partial differential equations [13] and perform atomistic modelling [14], while generative adversarial networks have been used to improve the resolution of turbulent flow simulations [15]. A recent survey has found that data-driven techniques are particularly useful in solving two classes of problems [12]: The first class is that of problems where current computational resources are insufficient for obtaining results of the desired accuracy, while the other includes problems where no complete set of governing equations is known. Coincidentally, these cases are exactly those where the weaknesses of purely physics-based models, as discussed above, are most prominent. Thus, the body of work described in the survey [12] suggests that it is possible to combine data-driven models and physics-based models such as to eliminate the weaknesses of the latter approach.

One paper which is closely related to the present work concerns the modelling of turbulence in large eddy simulations using a combination of physics-based and data-driven techniques [16]. More specifically, the paper authors use a neural network to generate a source term which is added to the turbulent system's governing equations. This source term is used to capture the effects of eddy currents that are too small to be modelled directly using their chosen domain discretization.

We believe that the approach of adding a neural network-generated source term to a governing equation is a direction of research with great potential. One benefit of generating source terms is that the source terms can be regularized independently of the final numerical solution. Such regularization can e.g. come in the form of choosing particular inputs for the neural networks, or from post-processing the source terms before inserting them into the governing equation, so as to avoid unphysical behaviour [16]. Another benefit is that the source terms can be interpreted in terms of relevant theory. One particularly interesting option is to study the ML-generated source term using symbolic regression – a technique which has proven effective in discovering analytic expressions for complex data [17, 18]. The regressed expressions of the corrective source terms can then be compared with known theory as a validation of the data-driven model. This level of interpretability and ability to enforce physically meaningful solutions through regularization are properties which are often missing from machine learning-based correction methods for physical simulations [12]. Yet, these qualities are highly desirable in e.g. the context of digital twins, as discussed above. This motivates the study of correction methods based on generating corrective source terms using data-driven techniques, which is an important part of the present work.

1.2. Research Objective and Contributions

The research objective of the work presented in this report is to answer the following research questions:

1. How can a physics-based simulation scheme and a data-driven correction method be combined into a unified model?

2. Can these models account for hidden physics? Here, "hidden physics" refers to physical processes that are either not included in the physics-based simulation, or processes that are not adequately resolved due to the discretization of the governing equations.

3. Do these models satisfy the requirements for use in digital twin applications, as described in the previous section?

In our investigation of the first question, we adopt two different approaches, labelled *end-to-end learning* and *hybrid modelling*. Both approaches use a neural network to perform the corrections. End-to-end learning is the simplest and most naive approach of the two, and serves as baseline to which we can compare the more involved hybrid modelling approach.

For the end-to-end approach, uncorrected solutions from physics-based, numerical simulations are given as input to the neural network. The neural network is then trained to find a mapping between these uncorrected solutions and more accurate reference solutions. The corrected solutions are then given directly as the output of the neural network. The term "end-to-end" in the name of the approach refers to the fact that the neural network must find a mapping directly from one solution to another without any *a priori* of the physics of the underlying process.

For the hybrid modelling approach, we adopt the idea of using a neural network to generate a corrective source term to account for unresolved physics, as described in [16]. The name “hybrid modelling” alludes to the intertwined relationship between the physics-based simulation and the data-driven correction method. Our hybrid modelling approach deviates slightly from the description in [16] in two ways. The first difference is that we define the corrective source term as a modification to the *discretized* governing equation, rather than the governing equation itself. In principle, this is only a change in perspective and does not fundamentally alter the applicability of the approach. However, the author believes this change makes it conceptually easier to see how the approach can be used to correct diffusion and dispersion errors caused by discretization. The second difference is that we perform a trivial generalization by not making any assumptions about the nature of the error modelled by the ML-generated source term.

Using the end-to-end learning and hybrid modelling approaches, we investigate the latter two research questions in the context of one-dimensional heat conduction problems. Consequently, a significant part of this work has been to implement and verify a simulation framework for solving such problems numerically. This report includes all derivations upon which the simulation framework has been built.

All code used to perform the numerical simulations and machine learning experiments described in this report was written by the author in Python 3.6. Appendix A provides an overview of both Python packages and hardware used in the present work. The author wishes to highlight the use of the scientific programming package NumPy [19] and the machine learning framework PyTorch [20].

1.3. Structure of the Report

This report contains eight chapters, of which the introduction you are currently reading is the first. An overview of relevant machine learning theory is presented in Chapter 2. Chapter 3 explains how one-dimensional heat conduction is modelled in the present work, both analytically as a partial differential equation and numerically using the Implicit Euler finite-volume method. Chapter 4 is devoted to explaining how numerical simulations can be corrected using the end-to-end learning approach and the hybrid modelling approach. To ensure that the Implicit Euler method described in Chapter 3 has been implemented correctly, a series of grid refinement studies have been performed, and these are described in Chapter 5. Chapter 6 covers a series of machine learning experiments where end-to-end learning and hybrid modelling were used to correct numerical simulation based on the Implicit Euler method implementation that was verified in Chapter 5. These machine learning experiments are discussed further in Chapter 7. Finally, the present work is concluded in Chapter 8, where suggestions for further work are also summarized.

2. Theory

In this chapter, relevant machine learning theory is presented. First, we define machine learning as a scientific discipline and cover some of its main subdivisions in Section 2.1. We then take closer look at one of these subdivisions – machine learning with neural networks – beginning with the structure of neural networks in Section 2.2. In Sections 2.3 and 2.4, we discuss the concepts of datasets and loss functions, which are prerequisites for the description of the training procedure of neural networks in Section 2.5. Finally, the concept of regularization is covered in Section 2.6.

Note that we use standard machine learning notation in this chapter. Unfortunately, this notation sometimes clashes with the standard physics notation used elsewhere in this report. To avoid confusion, the notation that is specific to this chapter is listed as a separate category called "Machine Learning Notation" in the Nomenclature. Furthermore, note that in an effort to make the present work self-contained and accessible to readers with an academic background comparable to that of the author, some explanations go into greater detail than is typical for work in the machine learning field.

2.1. Machine Learning

Machine learning (ML) is a branch of computer science concerned with the study of algorithms that are able to *learn* from experience. One widely used definition of *learning*, courtesy of Tom Mitchell [21], is the following:

Definition A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

The definition above can be illustrated with an example closely related to the problems studied in this report. Consider a computer program whose purpose is to predict the temperature of a metal bar one minute into the future given the bar's current temperature. T is then the class of tasks that involve making such predictions. The program's performance can be measured by waiting one minute until the true temperature can be measured and then calculating the difference between the true temperature and the program's prediction. If historical data about the metal bar's temperature has been recorded at one-minute intervals, the program can gain experience by making practice predictions of any particular measurement based on the previous measurement. Furthermore, as time passes, the program can make practice predictions on the newly recorded data as well. The temperature prediction problem can thus be summarized as

Task T: Predicting future temperature,

Performance measure P: Difference between predicted and true temperature,

Experience E: Making practice predictions based on historical data.

Mitchell's definition of learning is broad. Thus, with this definition, machine learning is also a broad field covering vastly different algorithms. It is therefore useful do divide the machine learning field into separate categories. One way to do this is to look at the program's degree of independence during learning. Many ML algorithms then fall into one of four categories [22], which are

1. Supervised learning,
2. Semi-supervised learning,
3. Unsupervised learning,
4. Reinforcement learning.

In *supervised learning*, the ML algorithm has access to a target output for each input data example it can use for training. In the context of the example above, this means that all the historical temperature profiles are labelled with timestamps. Thus, when the algorithm makes a practice prediction based on any one measurement, it can find the subsequent measurement and compare it with its prediction. In the present work, supervised learning has been used exclusively, but the other categories are also explained briefly for the purpose of completeness.

Supervised learning is contrasted by *unsupervised learning*, where no target output is known for any example input. If, in our example, none of the historical measurements had timestamps, then our algorithm would be solving an unsupervised learning problem. In this case, it would be part of the algorithm's learning process to figure out how the unlabelled measurements relate to each other and how to use them for evaluating its practice predictions.

As the name suggests, *semi-supervised learning* combines supervised and unsupervised learning by utilizing both labelled and unlabelled data. Our example problem is a semi-supervised learning problem if some historical measurements have timestamps while others do not.

Lastly, *reinforcement learning* is quite different from the others, as it is a process-oriented approach to learning. To illustrate, let us consider a different example where an ML algorithm is tasked with driving a car. Driving is a complex process, which requires the algorithm to take many different actions, like steering and applying the throttle or the brakes. If the algorithm is based on supervised learning, it receives feedback on each one of these actions. I.e., the algorithm receives feedback on the process of getting to the final state, not just on the final state itself as is the case with the other types of learning.

Another way of dividing the machine learning field into categories is to look at the data structures utilized by the algorithms. Within this categorization framework, one important category is that of (artificial) neural networks. These algorithms take inspiration from the way information is processed by the interconnected neurons found in animal brains. The term "artifical" is sometimes used to separate the neural networks used in ML from those found in nature. However, we will not make use of this term here, because the distinction should be clear from context; this report is only concerned with neural networks in the context of machine learning. Coincidentally, this report is also only concerned with machine learning algorithms utilizing neural networks. We therefore continue by examining the inner workings of neural networks in some detail.

2.2. Neural Networks

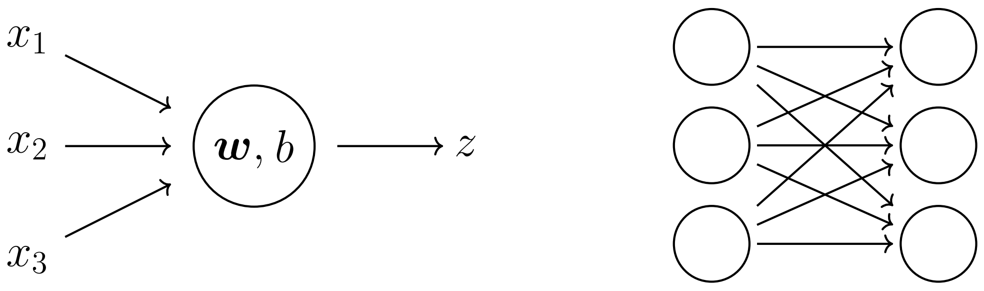
A neural network (NN) can be constructed in many different ways, but they typically consist of a selection of different *layers*, with each layer consisting of one or more *neurons*. A neuron is basically a simple processing unit which takes a vector of numbers as input, performs some computation using this input vector and outputs a single number. Each neuron is associated with certain parameters known as its *weights* and *bias*. These parameters play a crucial role in determining each neuron's output. When a network is learning, it does so by updating the values of each neuron's weights and bias such that the performance of the network as a whole is increased with respect to the performance measure used.

To explain the calculations carried out by the neurons, we must first introduce some notation. Let $\boldsymbol{x}$ denote the input vector of a neuron, and let M be the number of components in $\boldsymbol{x}$. Furthermore, let z be the neuron's single output, $\boldsymbol{w}$ be a vector containing the neuron's weights and b be the neuron's bias. The neuron's output is then given by the formula

$$z = \boldsymbol{w} \cdot \boldsymbol{x} + b. \tag{2.1}$$

For this formula to make sense, the neuron must have exactly M weights, such that $\boldsymbol{w}$ and $\boldsymbol{x}$ have the same length. From Equation (2.1), we see that the ith weight of the neuron, w_i, determines the contribution of the ith component of the input $\boldsymbol{x}$ to the the output z. Moreover, the bias b provides an off-set to the output z. The relations between $\boldsymbol{x}$, $\boldsymbol{w}$ and b are illustrated in Figure 2.1a for a neuron with input size $M = 3$.

On its own, a single neuron is not very powerful; it only performs a dot product calculation and an addition. The neurons' power comes in numbers – numbers which can be very large. The total number of parameters (weights and biases) of all the neurons in a network can, in certain cases, exceed 100 billion [23]. However, simply using the output of some neurons as the input of other neurons will not go very far, because all the operations involved are linear. In order for a network to learn non-linear relations, a non-linear activation function must be applied. In principle, each neuron could have its own unique activation function, but in practice, it is customary to use the same activation function for all neurons within the same layer. The activation function is then commonly referred to as the activation function *of the layer* rather than of the individual neuron. The role of the activation function, which we denote f, is to modify the neurons' outputs before these are used as input to other neurons. Thus, if a sending



(a) Neuron (b) Fully connected layers

Figure 2.1.: Left: An illustration of a neuron receiving a vector $\boldsymbol{x} = [x_1, x_2, x_3]$ and outputting a single number $z = w_1x_1 + w_2x_2 + w_3x_3 + b$. $\boldsymbol{w} = [x_1, w_2, w_3]$ are the neuron's weights, and b is its bias.
Right: An illustration of the connections between two layers of a fully connected neural network, with each layer consisting of three neurons.

neuron outputs the number z, the receiving neuron will take $f(z)$ as input, rather than just z.

There exists a wide variety of different activation functions; in principle, any non-linear function can be used as an activation function. However, some activation functions are more widely used than others. One popular choice is the Rectified Linear Unit (ReLU), which is defined as

$$\text{ReLU}(x) = \begin{cases} x & x \geq 0, \\ 0 & x < 0. \end{cases} \tag{2.2}$$

While being one of the simplest non-linear functions imaginable, ReLU has proven to perform equally well or better than other popular activation functions such as the hyperbolic tangent function [24]. Furthermore, ReLU is a non-saturating activation function, meaning that it does not converge to any fixed value in the limit of inputs with infinitely large absolute value. A benefit of non-saturating activation functions is that they do not suffer from the problem of vanishing/exploding gradients [25]. However, the fact that ReLU zeroes all negative inputs implies that neurons whose outputs are negative will not have any influence on the output of the network as a whole. These neurons will then not receive any feedback and are hence unable to update their weights and biases. Thus, if a neuron has weights and bias such that it gives negative output for a large spectrum of inputs, it is unlikely that the neuron will be able to update its parameters in a beneficial way. The LeakyReLU-function was designed to alleviate this issue. It is defined as

$$\text{LeakyReLU}(x) = \begin{cases} x & x \geq 0, \\ -\alpha x & x < 0, \end{cases} \tag{2.3}$$

where α is some constant between 0 and 1 whose value is chosen by the NN developer. The original incarnation of LeakyReLU used $\alpha = 0.01$ [26], but there are no hard and fast rules for determining the optimal value of α for a given problem.

Now, let us move on to further discuss the overall structure of neural networks. As mentioned at the start of this section, the neurons are typically structured in layers – one layer after another – with each layer containing a number of neurons. To further specify the structure of the neurons, it is necessary to know what types of layers are used in the network. In this work, we use a class of neural networks known as *fully connected neural networks* (FCNNs). The layers of a FCNN can generally be split into three categories, depending on their placement within the network. The *input layer* is the network's first layer, and it is responsible for reading the network's input. At the other end of the network, we find the *output layer* which is responsible for constructing the network's final output vector. All the layers (if any) in-between the input layer and the output layer are known as the *hidden layers*. If a FCNN has more than one hidden layer, we often refer to it as a *deep neural network* (DNN) [27]. (Note that other types of neural networks can also be classified as DNNs.) Machine learning using deep neural networks is often called *deep learning*.

In a FCNN, all neurons within the same layer are completely independent of each other. This means that they all have their own weights and biases, and that their outputs are independent. Furthermore, with the exception of the output layer, each neuron in a given layer sends its output to every neuron in the succeeding layer. To illustrate this, say that the output of the *jth* neuron in a given layer is denoted $z_j, j = 1, \ldots, M$. Then, all neurons in the subsequent layer will receive the same input vector, $[f(z_1), \ldots, f(z_M)]$, where f is an activation function. The connections between two layers in a FCNN are illustrated in Figure 2.1b for the case where both layers have 3 neurons each.

2.3. Datasets

In this work, we use neural networks to perform supervised learning. As explained earlier in this chapter, supervised learning requires our neural networks to have access to example inputs and corresponding target outputs. We will refer to the combination of one example input and its corresponding target output as one *data example*. Data examples are typically stored in data structures known as *datasets*. One way to construct a dataset is by making two lists, one containing example inputs and the other containing target outputs. Corresponding inputs and outputs are stored at the same index in their respective lists.

When developing neural networks, it is customary to use three different datasets: a *training set*, a *validation set* and a *test set*. All three datasets contain data examples of the same kind, but are used for different purposes. The training set is the only dataset which the neural network is allowed to use directly for learning, i.e. the network is only allowed to update its neurons' weights and biases based on its experience from the training set. The validation set is used by the model developer to make informed decisions when adjusting the network architecture or tuning other parameters of the neural network during development. Thus, the validation set also has some influence on the performance of the final model, albeit indirectly. This is not the case for the test set, which is used exclusively to evaluate the performance of the final, fully trained and tuned model.

2.4. Loss Functions

To gain experience, the neural network must have some way of receiving feedback on its performance during training. Such feedback is provided by a *loss function.* In the context of supervised learning, the loss function typically evaluates the "closeness" of the neural network's output with respect to the target output corresponding to the network's input. Thus, a small value of the loss function indicates good performance. During training, the neural network therefore updates its neurons' weights and biases in such a way as to minimize the loss function for all data examples in the training set. In the next section, we will discuss how these updates are performed, but first, a brief overview of the loss function used in this work is provided.

The mean squared error (MSE) loss function is a widely used loss function based on the l^2 norm of finite-dimensional vectors, $|| \cdot ||_{l^2}$. Consider two d-dimensional vectors $y = \{y_k\}_{k=1,\dots,d}$ and $\hat{y} = \{\hat{y}_k\}_{k=1,\dots,d}$ representing the output of a neural network and the corresponding target output respectively. The l^2 norm of $y - \hat{y}$ is then

$$\|y - \hat{y}\|_{l^2} = \left(\sum_{k=1}^{d} |y_k - \hat{y}_k|^2 \right)^{1/2} . \tag{2.4}$$

Using the l^2 norm, the MSE loss function is defined by

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{d} \|y - \hat{y}\|_{l^2}^2 = \frac{1}{d} \sum_{k=1}^{d} |y_k - \hat{y}_k|^2. \tag{2.5}$$

The MSE loss function has many desirable properties. Most importantly, it satisfies the metric axioms [28], so it is a well-defined measure of the difference between the outputs y and the targets $\hat{y}$. Furthermore, it is continuously differentiable for all y and $\hat{y}$, and the global minimum at $y = \hat{y}$ is its only local minimum. However, the MSE loss function also has some drawbacks. Most importantly, it makes performance on data examples with large absolute values more important than performance on data examples with small absolute values. Thus, a neural network trained to predict temperature profiles using the MSE loss function will be biased towards minimizing the error in high-temperature regions and care less about the error in low-temperature regions. For this reason, the predicted behaviour in low-temperature regions might be qualitatively wrong, even in cases where the MSE loss function indicates good performance. In cases where the predictions are part of a time series, and thereby depend on earlier predictions, such qualitative errors might lead to divergent behaviour.

2.5. Training a Neural Network

From the previous sections, we know how neural networks are constructed, how to structure data for supervised learning and how feedback can be given to the neural network during training. This section explains how the neural networks utilizes this feedback to gain experience, and thereby increase its performance. To learn, i.e. update the parameters (weights and biases) of its neurons, a neural network relies on an algorithm known as *backpropagation*. This algorithm is used to calculate the derivatives of the loss function with respect to the network's parameters. The parameters are then updated by the network's *optimizer*. We take a closer look at backpropagation first, and then explain the role of the optimizer later. Unless otherwise noted, this section is based on the discussions in [27] and [29].

Backpropagation is essentially just a clever way of applying the chain rule of derivatives, and can be conveniently summarized using the so-called *four equations of backpropagation*. These equations are usually written in vector form, which requires us to define some more notation. Let $\boldsymbol{z}^n$ be a vector containing the output of the neurons in layer n of a neural network with N layers, let $\boldsymbol{b}^n$ be a vector containing the biases of that layer and let $\mathbb{W}^n$ be a matrix where the entry w^n_{jk} describes the weight for the connection between the kth neuron of layer $n-1$ and the jth neuron of layer n. Furthermore, let C be the loss function used for training the network, let f' be the derivative of the activation function f with respect to its input, and, lastly, define $\boldsymbol{a}^n = f(\boldsymbol{z}^n)$. Our quantities of interest are then the partial derivatives $\frac{\partial C}{\partial b^n_j}$ and $\frac{\partial C}{\partial w^n_{jk}}$, and the four equations of backpropagation can be stated as follows:

$$\delta^N = \nabla C \odot f'(z^N) \tag{2.6}$$

$$\delta^n = \left(\left(w^{n+1}\right)^T \delta^{n+1}\right) \odot f'(z^n), \quad n = 1, 2, \dots, N-1 \tag{2.7}$$

$$\frac{\partial C}{\partial b^n_j} = \delta^n_j, \quad n = 1, 2, \dots, N \tag{2.8}$$

$$\frac{\partial C}{\partial w^n_{jk}} = a^{n-1}_k \delta^k_j, \quad n = 1, 2, \dots, N \tag{2.9}$$

Here, the δ's are intermediate vectors used to simplify the equations, superscript ^{T} indicates transposition and $\odot$ represents element-wise multiplication.

Now that we know how to compute the contribution of each network parameter to the final loss value, we need to establish how the parameters should be updated. The updates are performed by an *optimizer*, which is a data object that holds the current state of the network and applies some specified optimization algorithm to the parameters based on the gradients computed using the backpropagation algorithm. Different optimizers apply different optimization algorithms. One of the simplest optimization algorithms is called *gradient descent*, and uses the following update rules:

$$w^n_{jk} \leftarrow w^n_{jk} - \eta \frac{\partial C}{\partial w^n_{jk}}, \quad b^n_j \leftarrow b^n_j - \eta \frac{\partial C}{\partial b^n_j}. \tag{2.10}$$

Here, η is a parameter known as the learning rate. The value of η is chosen by the neural network developer, and determines the magnitude of the updates in the parameter values.

The most naive approach to gradient descent is to calculate $\frac{\partial C}{\partial b^n_j}$ and $\frac{\partial C}{\partial w^l_{jk}}$ for all data examples in the training set, average these gradients and then update the parameters.

This algorithm, known as *batch gradient descent*, takes a lot of time for large datasets, causing training to be impractically slow. This issue is addressed by the *stochastic* gradient descent algorithm. With this algorithm, the gradients for the full training set are approximated by gradients calculated using a small, randomly chosen subset of the training set called a *mini-batch*. In addition to requiring fewer gradient calculations, the inherent randomness of stochastic gradient descent also increases the probability that the optimization process will be able to escape poor local minima of the loss function C. (Note that some authors, including the author of [29], reserves the term stochastic gradient descent for the case when each mini-batch contains only on data example. The term *mini-batch* gradient descent is then used for the case of larger mini-batches.)

The Adam optimization algorithm, originally proposed in [30], is an extended variant of the stochastic gradient descent algorithm. Its update rule for an arbitrary parameter θ is

$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{\hat{v}} + \epsilon}\hat{m}. \tag{2.11}$$

Here, ϵ is a small number added to avoid potential division by zero, while $\hat{m}$ and $\hat{v}$ are bias-corrected estimators of the first moment (mean) and the second moment (variance) of the gradient $\frac{\partial C}{\partial \theta}$. These estimators are continuously updated during training using a decaying average update rule. Replacing $\frac{\partial C}{\partial \theta}$ with $\hat{m}$ in the update rule adds a kind of momentum to the updates, in the sense that larger updates are performed if the gradients have been pointing in roughly the same direction during consecutive iterations. This helps in traversing regions where the gradients of the loss functions are large in certain directions and small in others. Additionally, the term $\frac{\eta}{\sqrt{\hat{v}} + \epsilon}$ can be viewed as an effective learning rate that is unique to each parameter θ. Note that the effective learning rate is adaptive, since $\hat{v}$ is continuously updated during training. The use of parameter-specific, adaptive learning rates increases the robustness of the optimization algorithm, and is particularly helpful for sparse data.

2.6. Regularization

A common problem when training neural networks is a phenomenon known as *overfitting*. Overfitting is one of two possible manifestations of the variance-bias trade-off (the other being underfitting), and occurs when the network learns information that is specific to the data examples in the training set. An instructive analogy is the problem of performing polynomial regression on noisy data, which is illustrated in Figure 2.2. The black dots represent five noisy samplings of the function $y(x) = 1$ for different values of x in the range $[0, 2]$. The quartic interpolating polynomial (red) passes through all five data points, while the quadratic interpolating polynomial (blue) does not. However, the quadratic polynomial is generally closer to the target function $y(x) = 1$ than is the quartic polynomial. The difference in closeness is especially clear for x outside the range $[0, 2]$. In essence, performing interpolation with a larger-than-necessary representative power can reduce accuracy on new data, i.e, it can cause the interpolating polynomial to overfit. Similarly, neural networks might also overfit on their training data. To reduce

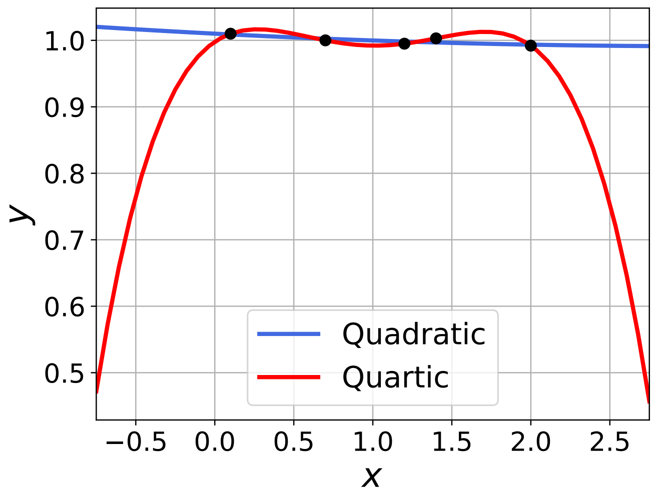


Figure 2.2.: An illustration of overfitting: Quadratic (blue) and quartic (red) interpolating polynomials for 5 noisy samplings (black dots) of the function $y(x) = 1 \ \forall x \in \mathbb{R}$.

overfitting, ML model developers can use different regularization techniques such as those presented in the (non-exhaustive) list below.

1. Early stopping: Stop training before overfitting occurs. In practice, this requires saving the model at regular time intervals during training. By studying the development of the training and validation losses, one can then choose to use a model that was saved at a time before signs of overfitting had emerged.

2. Parameter regularization: Impose penalties on large parameters by including a term in the loss function that is proportional to some norm of the network's parameters. In the context of the interpolation example above, parameter regularization can be compared to imposing penalties on using higher order monomials to define the interpolating polynomial. Alternatively, it can be compared to imposing penalties on the use of large polynomial coefficients.

3. Data augmentation: Use knowledge of the system at hand to generate more training data from the data that is available initially. For the problem of interpolating a function, this means to generate more points $(x, f(x))$ from those points that are already known. For example, if it is known that f is odd, then the known data points can be mirrored across the origin to generate new data points.

4. Dropout: Randomly remove neurons from the network with some fixed probability during training. The neural network must then learn to not depend heavily on the output of any single neuron. A close analogue to dropout in the context of interpolation is to randomly remove some monomial terms from the interpolating polynomial.

3. Physical Modelling

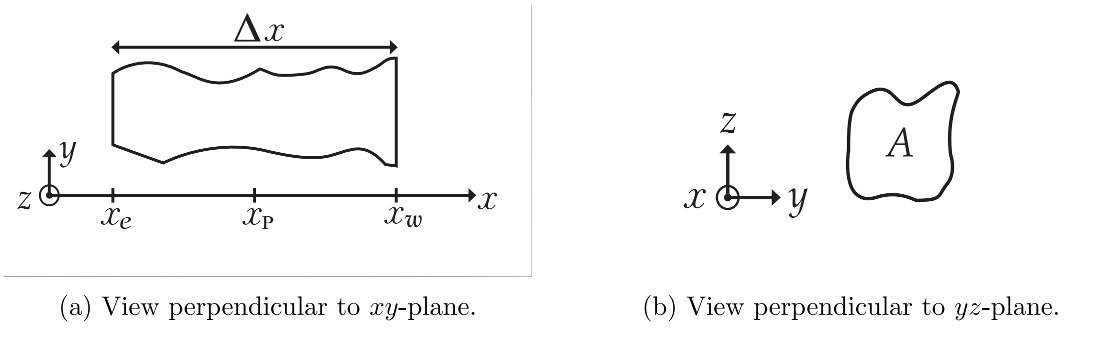
In order to accomplish our principal goal of studying data-driven correction methods, we need both a physical system to study and a numerical scheme to simulate the behaviour of that system. The purpose of this chapter is to take care of these needs. In Section 3.1, we derive a partial differential equation (PDE) governing one-dimensional (1D) heat conduction for a general system. We refer to this equation as the *1D heat equation.* The exact solutions of the 1D heat equation are derived for two specific cases in Section 3.2. Lastly, in Section 3.3, the 1D heat equation is discretized using the Implicit Euler finite-volume method (FVM). All numerical simulations described throughout the entirety of this report use this FVM scheme.

3.1. The 1D Heat Equation

In this derivation of the 1D heat equation, we will consider a control volume ΔV of length Δx in the x-direction with cross-sectional area A perpendicular to the x-axis. We allow the area A to change as a function of x, but not as a function of time, i.e., we assume the control volume to be stationary. An example control volume is illustrated in Figure 3.1. We assume that the center of the control volume is located at $x = x_\text{P}$, and that its left and right faces are located at $x = x_w$ and $x = x_e$ respectively.

We will assume that our control volume constitutes a closed system undergoing some process of finite duration Δt. For this process, the 1st law of thermodynamics takes the form

$$\Delta U = Q - W, \tag{3.1}$$



(a) View perpendicular to xy-plane.

(b) View perpendicular to yz-plane.

Figure 3.1.: A general control volume of length Δx and cross-sectional area A. Note that A is a function of x for the illustrated control volume.

where ΔU is the change in the control volume's internal energy during the process, Q is the heat added *to* the control volume, and W is the work done *on* the control volume *by* its surroundings.

We assume that no work is done on the control volume, such that $W = 0$. Equation (3.1) then reduces to

$$\Delta U = Q. \tag{3.2}$$

The heat Q added to the system has two possible contributions: the net heat flux across the control volume's boundary and the heat generation within the control volume itself. Mathematically, we then have

$$Q = \int_{t}^{t+\Delta t} \left(\int_{\partial V} \boldsymbol{q}(\boldsymbol{r}, t') \cdot (-\boldsymbol{n}) \; \mathrm{d}A + \int_{V} \hat{q}(\boldsymbol{r}, t') \; \mathrm{d}V \right) \mathrm{d}t', \tag{3.3}$$

where $\boldsymbol{r} = [x, y, z]$ is a position vector, $\boldsymbol{q} \cdot (-\boldsymbol{n})$ is the heat flux *into* the control volume across a surface with surface normal $\boldsymbol{n}$, and $\hat{q}$ is the rate at which heat is generated within an infinitesimal part $\mathrm{d}V$ of the control volume. The negative sign in front of $\boldsymbol{n}$ appears because is $\boldsymbol{n}$ is conventionally defined as pointing *out from* the control volume.

If we assume Δt to be sufficiently small, we can consider the inner integrals in Equation (3.3) to be approximately constant. We then get the approximation

$$Q \approx \Delta t \left(-\int_{\partial V} \boldsymbol{q}(\boldsymbol{r}) \cdot \boldsymbol{n} \; \mathrm{d}A + \int_{V} \hat{q}(\boldsymbol{r}) \; \mathrm{d}V \right) \tag{3.4}$$

Dividing Equation (3.2) with Δt and inserting the right-hand-side of Equation (3.4) for Q, we get

$$\frac{\Delta U}{\Delta t} = -\int_{\partial V} \boldsymbol{q}(\boldsymbol{r}) \cdot \boldsymbol{n} \; \mathrm{d}A + \int_{V} \hat{q}(\boldsymbol{r}) \; \mathrm{d}V. \tag{3.5}$$

In the limit of $\Delta t \to 0$, this yields

$$\frac{\mathrm{d}U}{\mathrm{d}t} = -\int_{\partial V} \boldsymbol{q}(\boldsymbol{r}) \cdot \boldsymbol{n} \; \mathrm{d}A + \int_{V} \hat{q}(\boldsymbol{r}) \; \mathrm{d}V \tag{3.6}$$

Since we are studying 1D heat conduction, we assume that heat is only transported in the x-direction. Hence $\boldsymbol{q}(\boldsymbol{r}) = q(\boldsymbol{r})\boldsymbol{x}$, where $\boldsymbol{x}$ is the unit vector in the x-direction. The face at $x = x_e$ has surface normal $\boldsymbol{n} = \boldsymbol{x}$ and the face at $x = x_w$ has surface normal $\boldsymbol{n} = -\boldsymbol{x}$, and it is therefore evident that the flux across these faces will contribute to the surface integral in Equation (3.6). We now assume they are in fact *the only* contributions to this surface integral, which allows us to rewrite the surface integral in Equation (3.6) as

$$\int_{\partial V} \boldsymbol{q}(\boldsymbol{r}) \cdot \boldsymbol{n} \; \mathrm{d}A = \int_{\partial V_e} q(\boldsymbol{r}) \; \mathrm{d}A - \int_{\partial V_w} q(\boldsymbol{r}) \; \mathrm{d}A, \tag{3.7}$$

where ∂V_e and ∂V_w denotes the control volume's right and left face respectively.

Since we are considering heat conduction in the x-direction only, the heat flux q must be independent of the y- and z-coordinates. Thus

$$\int_{\partial V_e} q(\boldsymbol{r})\ \mathrm{d}A = q(x = x_e)A(x = x_e) \quad \text{and} \quad \int_{\partial V_w} q(\boldsymbol{r})\ \mathrm{d}A = q(x = x_w)A(x = x_w). \tag{3.8}$$

For convenience, we use the following notation

$$q(x = x_e)A(x = x_e) = (qA)_e, \quad q(x = x_w)A(x = x_w) = (qA)_w. \tag{3.9}$$

With this notation, we combine Equations (3.7) and (3.8) and insert into Equation (3.6) to obtain

$$\frac{\mathrm{d}U}{\mathrm{d}t} = -(qA)_e + (qA)_w + \int_V \hat{q}(\boldsymbol{r})\ \mathrm{d}V \tag{3.10}$$

It is also possible to consider the control volume's internal energy change in a different way. Suppose that an infinitesimally small part $\mathrm{d}V$ of the control volume is in thermal equilibrium with its surroundings, such that its temperature is well-defined. Let S be the total entropy of the control volume, and let N_i be the mole number of substance i in the control volume. If we define the *local* internal energy, u, such that $\int_V u\ \mathrm{d}V = U$, the local entropy, s, such that $\int_V s\ \mathrm{d}V = S$ and the local concentration of substance i, c_i, such that $\int_V c_i\ \mathrm{d}V = N_i$, we can use the local form of Gibbs' equation [31]. It states that

$$\mathrm{d}u = T\mathrm{d}s + \sum_i \mu_i \mathrm{d}c_i \tag{3.11}$$

when the process is locally reversible. Here, μ_i is the chemical potential of substance i.

We have previously assumed that our control volume is closed, such that N_i is constant. This does not necessarily imply that $\mathrm{d}c_i = 0$, because we could e.g. have eddies within the control volume causing the local concentrations c_i to be non-constant. To make further progress, we therefore make the additional assumption that $\mathrm{d}c_i = 0$ for all i. From Equation (3.11), we then get

$$\mathrm{d}u = T\mathrm{d}s \quad \Longrightarrow \quad \frac{\mathrm{d}u}{\mathrm{d}s} = T. \tag{3.12}$$

Using the chain rule, we find that

$$\frac{\mathrm{d}u}{\mathrm{d}s} = \frac{\partial u}{\partial T}\frac{\partial T}{\partial s} = \frac{\partial u}{\partial t}\frac{\partial t}{\partial T}\frac{\partial T}{\partial s} = T. \tag{3.13}$$

For any physical system, $t(T)$ and $T(s)$ are both continuously differentiable with non-zero derivatives, so we can use the inverse function theorem to rearrange Equation (3.13) as

$$\frac{\partial u}{\partial t} = T\frac{\partial s}{\partial T}\frac{\partial T}{\partial t}. \tag{3.14}$$

.

To progress, we note that the definition of heat capacity at constant volume of a system with entropy S and temperature T is

$$C_V = \left(\frac{\partial Q}{\partial T}\right)_V. \tag{3.15}$$

Furthermore, for a reversible process, it is known that

$$\partial S = \frac{1}{T}\partial Q. \tag{3.16}$$

Combining Equations (3.15) and (3.16) yields

$$C_V = \left(T\frac{\partial S}{\partial T}\right)_V. \tag{3.17}$$

In our problem, we have already assumed V to be constant, so we drop the subscript denoting constant V in the following.

The heat capacity C_V, which is an extensive variable, can be expressed as the product of the systems mass m and the *specific* heat capacity c_V, which is an intensive variable. This means Equation (3.17) can be equivalently expressed as

$$mc_V = T\frac{\partial S}{\partial T}. \tag{3.18}$$

In this equation, we can replace all the extensive global variables with their corresponding local variables. We therefore make the replacements $m \to \rho$ and $S \to s$. This yields

$$\rho c_V = T\frac{\partial s}{\partial T} \tag{3.19}$$

We can now insert Equation (3.19) into Equation (3.14) and obtain

$$\frac{\partial u}{\partial t} = \rho c_V \frac{\partial T}{\partial t}, \tag{3.20}$$

which is valid for an infinitesimally small part of our control volume. To obtain an expression for the derivative of the control volume's total internal energy, we simply integrate both sides of Equation (3.20) across the control volume and get

$$\int\limits_V \rho c_V \frac{\partial T}{\partial t}\mathrm{d}V = \int\limits_V \frac{\partial u}{\partial t}\mathrm{d}V = \frac{\mathrm{d}}{\mathrm{dt}}\left(\int\limits_V u\mathrm{d}V\right) = \frac{\mathrm{d}U}{\mathrm{d}t}. \tag{3.21}$$

We now have two different equations for $\frac{\mathrm{d}U}{\mathrm{d}t}$, namely Equation (3.10) and Equation (3.21). Equating these yields

$$\int\limits_V \rho c_V \frac{\partial T}{\partial t}\mathrm{d}V = -(qA)_e + (qA)_w + \int\limits_V \hat{q}(\boldsymbol{r})\ \mathrm{d}V \tag{3.22}$$

All that remains is to express q as a function of T. This can be done using Fourier's law of heat conduction, which reads

$$q = -k\frac{\partial T}{\partial x} \tag{3.23}$$

in one dimension [32]. Here, k denotes thermal conductivity. Inserting Fourier's law (3.23) into Equation (3.22), we finally obtain the one dimensional heat equation:

$$\int\limits_V \rho c_V \frac{\partial T}{\partial t} \mathrm{d}V = \left(kA\frac{\partial T}{\partial x}\right)_e - \left(kA\frac{\partial T}{\partial x}\right)_w + \int\limits_V \hat{q}(\boldsymbol{r})\ \mathrm{d}V \tag{3.24}$$

More specifically, the above equation is often referred to as the 1D heat equation on *integral form* due to the integration over the control volume V. The 1D heat equation can also be written on *differential form* in certain cases, and this is something we will make use of when deriving exact solution of the 1D heat equation in the next section.

3.2. Exact Solutions of the 1D Heat Equation

Analytic expressions for the exact solution of the 1D heat equation on integral form (3.24) can be derived in several interesting cases. We will derive the exact solution in two cases of particular interest to us:

1. Steady-state heat conduction without source term for non-uniform physical parameters.

2. Steady-state heat conduction with constant source term for uniform physical parameters.

In both derivations, we will make use of the 1D heat equation on differential form, which can be derived from the integral form if the temperature T is *sufficiently smooth*. We therefore begin this section by deriving the differential form. Thereafter, some considerations regarding boundary conditions (BCs) are made before the exact solutions themselves are derived in Sections 3.2.1 and 3.2.2.

For our derivation of the 1D heat equation on differential form, we assume that the temperature T is twice continuously differentiable. The first term on the right hand side of Equation (3.24) can then be rewritten in the following way

$$\left(kA\frac{\partial T}{\partial x}\right)_e - \left(kA\frac{\partial T}{\partial x}\right)_w = \int\limits_{\partial V} kA\frac{\partial T}{\partial x}\boldsymbol{x}\cdot\boldsymbol{n}\ \mathrm{d}A \tag{3.25}$$

$$= \int\limits_{\partial V} kA\nabla T\cdot\boldsymbol{n}\ \mathrm{d}A \tag{3.26}$$

$$= \int\limits_{V} \nabla\cdot(kA\nabla T)\ \mathrm{d}V \tag{3.27}$$

$$= \int\limits_{V} \frac{\partial}{\partial x}\left(kA\frac{\partial T}{\partial x}\right)\ \mathrm{d}V. \tag{3.28}$$

In the above, the third equality follows from the divergence theorem (which is valid for smooth ∇T [33], i.e. for T that is twice continuously differentiable), while the last equality follows from the observation that $\frac{\partial T}{\partial y} = \frac{\partial T}{\partial z} = 0$ in our 1D system. The integral equation (3.24) as a whole can then be rewritten as

$$\int\limits_{V} \rho c_V\frac{\partial T}{\partial t}\mathrm{d}V = \int\limits_{V} \frac{\partial}{\partial x}\left(kA\frac{\partial T}{\partial x}\right)\ \mathrm{d}V + \int\limits_{V} \hat{q}(\boldsymbol{r})\ \mathrm{d}V \tag{3.29}$$

Since the control volume V is arbitrary, this can only hold if the integrands are equal. Thus,

$$\rho c_V\frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(kA\frac{\partial T}{\partial x}\right) + \hat{q}, \tag{3.30}$$

which is the 1D heat equation on differential form. It is valid for all $x \in (x_w, x_e)$.

The reason for deriving the differential form of the 1D heat equation is that it is often easier to obtain an analytic expression for T by integrating the differential form rather than manipulating the integral form. This convenience comes at the cost of the smoothness requirement. Note however that Equation (3.30) can be integrated to obtain a valid analytic expression for T even if T is not twice differentiable for *finitely many* values of x provided T is continuous everywhere. Let us denote the x-values where T is not twice differentiable by $x_1^*, x_2^*, \ldots, x_M^*$, where M is the number of such points, and suppose $x_w < x_1^* < x_2^* < \ldots x_M^* < x_e$. (Note that T might not even be once differentiable at these points.) Then, by construction, T is twice continuously differentiable on (x_w, x_1^*), so the differential form (3.30) is valid on this interval. The differential form can then be integrated to give a temperature function T that is valid for all $x \in (x_w, x_1^*)$. Under the assumption that T is continuous, we then also know that $T(x_1^*) = \lim_{x\to x_1^{*-}} T(x)$, where the minus sign in the superscript signifies that x is approaching x_1^* from below. Similar reasoning also holds for the intervals $(x_1^*, x_2^*), \ldots, (x_M^*, x_e)$. We may thereby conclude that integrating the differential form (3.30) to obtain a valid temperature profile is possible under the assumptions that T is continuous everywhere and twice differentiable everywhere except for at finitely many values of x.

The assumption that T be continuous everywhere is not a restrictive assumption. If T were to be discontinuous at some point $x = x^*$, there would be an infinitely large temperature flux across x^*, and this is clearly unphysical. In light of this, one might also question the physicality of a temperature that is non-differentiable at certain locations, and with good reason. If T is not differentiable at x^*, it is possible to construct an infinitesimally thin control volume centered on x^* which will have different heat fluxes across its left boundary and right boundary. Thus, there is a finite net heat flux into or out from the control volume. Since an infinitesimally thin control volume has an infinitesimally small mass, it also has an infinitesimally small heat capacity. The finite heat flux will then result in an infinitely large heating/cooling rate within the control volume, and this is also clearly unphysical. Thus, all physical temperature profiles are continuously differentiable. However, at a macroscopic level, temperature profiles can sometimes appear to have discontinuous derivatives. One example, which we will encounter later in this report, is the case when there are large and sudden jumps in the magnitude of the conductivity k. From a macroscopic perspective, it might then make sense to model k as a piecewise continuous function, in which case T becomes a continuous and piecewise differentiable function. For such models, the earlier argument about the validity of the differential form (3.30) is important.

The 1D heat conduction equation was derived for an arbitrary control volume with its left boundary at $x = x_w$ and its right boundary at $x = x_e$. If we want to derive the exact temperature T over some other domain $[x_a, x_b]$, there is no reason why we cannot choose to define our control volume such that $x_w = x_a$ and $x_e = x_b$. Therefore, we do exactly that. However, before we can solve the 1D heat conduction equation, we must prescribe appropriate boundary conditions (BCs). In our derivations, we will assume Dirichlet BCs, i.e. we fix the temperature at the boundaries such that $T(x_a, t) = T_a$ and $T(x_b, t) = T_b$ for all $t \geq 0$. Here, x_a and x_b are the x-coordinates of the domain's left and right boundary respectively, while T_a and T_b are the prescribed boundary temperatures. Having prescribed the BCs we are now ready to derive exact solutions. We will do this in the following two cases:

1. Steady-state heat conduction without source term for non-uniform physical parameters.
2. Steady-state heat conduction with constant source term for uniform physical parameters.

3.2.1. Exact Solution of Steady-State Problem without Source Term for Non-Uniform Physical Parameters

For a steady-state problem

$$\frac{\partial T}{\partial t} = 0. \tag{3.31}$$

Furthermore, we assume zero heat generation, so $\hat{q} = 0$. Equation (3.30) then becomes

$$\frac{\partial}{\partial x}\left(kA\frac{\partial T}{\partial x}\right) = 0 \tag{3.32}$$

Upon integration by x, the above becomes

$$kA\frac{\partial T}{\partial x} = C, \tag{3.33}$$

where C is some constant that is yet to be determined. Division by kA (which is possible since kA is always non-zero) and another integration by x now yields

$$\int_{x_a}^{x} \frac{\partial T}{\partial x}\mathrm{d}x = \int_{x_a}^{x} \frac{C}{kA}\mathrm{d}x \quad \Longrightarrow \quad T(x) - T(x_a) = \int_{x_a}^{x} \frac{C}{kA}\mathrm{d}x = C\int_{x_a}^{x} \frac{1}{kA}\mathrm{d}x \tag{3.34}$$

The value of C can now be determined by imposing the BC at $x = x_b$. Rearranging the equation above and inserting $x = x_b$ gives

$$T(x_b) = T(x_a) + C\int_{x_a}^{x_b} \frac{1}{kA}\mathrm{d}x, \tag{3.35}$$

such that $T(x_b) = T_b$ if $C = \left(\int_{x_a}^{x_b} \frac{1}{kA}\mathrm{d}x\right)^{-1}(T_b - T_a)$. The exact solution must then be

$$T(x) = T_a + \left(\int_{x_a}^{x_b} \frac{1}{kA}\mathrm{d}x\right)^{-1}(T_b - T_a)\int_{x_a}^{x} \frac{1}{kA}\mathrm{d}x. \tag{3.36}$$

3.2.2. Exact Solution of Steady-State Problem with Constant Source Term for Uniform Physical Parameters

For this case, we have $\frac{\partial T}{\partial t} = 0$ as in the previous case, but we do not assume $\hat{q} = 0$, so Equation (3.30) now reads

$$\frac{\partial}{\partial x}\left(kA\frac{\partial T}{\partial x}\right) + \hat{q} = 0 \tag{3.37}$$

Integrating this once yields

$$kA\frac{\partial T}{\partial x} = -\hat{q}x + C, \tag{3.38}$$

where C is some constant of integration. Division by kA (recall again that kA is non-zero) and another integration now gives

$$T(x) - T_a = -\frac{\hat{q}}{2kA}(x - x_a)^2 + \frac{C}{kA}(x - x_a). \tag{3.39}$$

Evaluating this for $x = x_b$ yields

$$T(x_b) = T_a - \frac{\hat{q}}{2kA}(x_b - x_a)^2 + \frac{C}{kA}(x_b - x_a), \tag{3.40}$$

from which it is evident that $T(x_b) = T_b$ if $C = \dfrac{kA}{x_b - x_a}(T_b - T_a) + \dfrac{\hat{q}}{2}(x_b - x_a)$. Using this choice of C, the exact solution is expressed as

$$T(x) = T_a - \frac{\hat{q}}{2kA}(x - x_a)^2 + \left(\frac{kA}{x_b - x_a}(T_b - T_a) + \frac{\hat{q}}{2}(x_b - x_a)\right)\frac{1}{kA}(x - x_a). \tag{3.41}$$

3.3. Finite-Volume Method for the 1D Heat Equation

As explained in Section 3.1, heat conduction problems in one dimension can be modelled using the 1D heat equation (3.24). We now want to solve this equation numerically over some physical domain $x \in [x_a, x_b]$ for $t \geq 0$. The solution method depends on the boundary conditions (BCs) that are prescribed. We assume Dirichlet BCs, i.e., we assume the boundary temperatures $T_a = T(x_a)$ and $T_b = T(x_b)$ to be fixed and given. A complete formulation of our problem is then the following:

Find $T(x, t)$ such that

$$\int_V \rho c_V \frac{\partial T}{\partial t} \mathrm{d}V = \left(kA\frac{\partial T}{\partial x}\right)_e - \left(kA\frac{\partial T}{\partial x}\right)_w + \int_V \hat{q}(\boldsymbol{r})\ \mathrm{d}V \quad \forall x \in [x_a, x_b],\ t \geq 0, \tag{3.42}$$

$$\text{and } T(x_a, t) = T_a,\ T(x_b, t) = T_b\ \forall\ t \geq 0 \tag{3.43}$$

As we have seen in Section 3.2, this problem can be solved analytically in certain cases. However, obtaining an analytic solution is not always feasible. In such cases, we can use a numerical method to obtain a numerical solution that approximates the true solution. Such numerical methods work by first discretizing the true equation and then solving the resulting system of equations. In this section, we will explore how the 1D heat equation can be discretized using a finite-volume method (FVM) that employs a central difference scheme for the spatial discretization and the Implicit Euler method for the temporal discretization.

Before we get into the discretization of the equation itself, we briefly explain how we discretize the physical domain over which we want to solve the equation. As in the derivation of the 1D heat equation in Section 3.1, we consider a control volume which we now denote Ω_P. With this control volume, which we will also refer to as a cell, we associate a *grid node* located at $x = x_\text{P}$ for some x_P contained within Ω_P. Furthermore, we denote the neighbouring control volumes by Ω_W (left neighbour) and Ω_E (right neighbour) and associate them with grid nodes at $x = x_\text{W}$ and $x = x_\text{E}$ respectively. The node locations x_P, x_W and x_E are chosen such that the face separating Ω_P and Ω_W is located midway between x_P and x_W, and such that the face separating Ω_P and Ω_E is located midway between x_P and x_E. The face locations are denoted x_w and x_e, respectively, and they can be expressed mathematically as $x_w = \frac{1}{2}(x_\text{P} + x_\text{W})$ and $x_e = \frac{1}{2}(x_\text{P} + x_\text{E})$. The control volumes Ω_P, Ω_W and Ω_E are illustrated in Figure 3.2, along with the locations of their respective grid nodes and cell faces.

For a physical domain divided into N_j cells using the discretization scheme above, we naturally get N_j grid nodes. All of these nodes lie in the interior of our physical

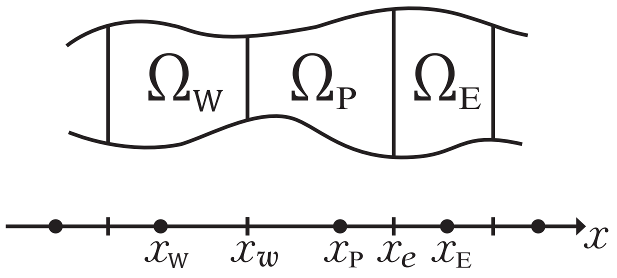


Figure 3.2.: A grid cell Ω_P along with its left neighbour Ω_W and right neighbour Ω_E. The cell faces separating the control volumes are located at x_w and x_e, midway between the grid nodes x_W and x_P, and x_P and x_E, respectively.

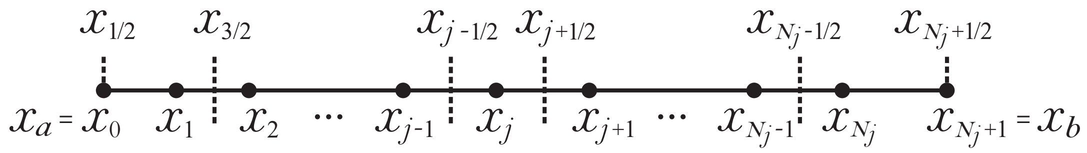


Figure 3.3.: Integral indices j are used to denote cell nodes, including the nodes at the domain boundaries. Half-integral indices $j + 1/2$ are used to denote cell faces. Note that the location of the left-most cell face coincides with the left-most grid node, such that $x_0 = x_{1/2} = x_a$. Similarly, we have $x_{N_j+1} = x_{N_j+1/2} = x_b$ at the right boundary.

domain, so they cannot be used to directly enforce the Dirichlet BCs. We circumvent this problem by adding two extra nodes – one at x_a and one at x_b. These extra nodes do not affect the locations of any cell faces. Consequently, the addition of the extra nodes means x_a and x_b are now locations of both a grid node and a cell face. Figure 3.3 provides a visualization how the physical domain is discretized in its interior and near its boundaries. This figure also shows how interior nodes are enumerated using the integral indices j and cell faces are enumerated with half-integral indices $j + 1/2$.

In addition to discretizing the spatial domain, we must also discretize the temporal domain. For this, we choose a fixed time step Δt, and consider discrete points in time separated by this fixed time step. We refer to these points in time as time levels, and use a superscript n to denote the nth time level. If the initial time is $t^0 = 0$, we have $t^n = n \cdot \Delta t$.

3.3.1. Finite-Volume Method for Unsteady Problems

We are now ready to derive a numerical simulation scheme for solving the 1D heat equation on integral form (3.24). First, we make some assumptions regarding the physical system at hand. The density ρ, the heat capacity c_V and the cross-sectional area A

are assumed constant, while the source term $\hat{q}$ is assumed constant in time (but not necessarily in space). Equation (3.24) can then be rewritten as

$$\rho c_V A \int_{x_w}^{x_e} \frac{\partial T}{\partial x} \mathrm{d}x = A \left(k \frac{\partial T}{\partial x} \right)_e - A \left(k \frac{\partial T}{\partial x} \right)_w + A \int_{x_w}^{x_e} \hat{q} \, \mathrm{d}x. \tag{3.44}$$

Defining $\Delta x_\mathrm{P} = x_e - x_w$ and dividing through by $\rho c_V A \Delta x_\mathrm{P}$ yields

$$\frac{1}{\Delta x_\mathrm{P}} \int_{x_w}^{x_e} \frac{\partial T}{\partial x} \mathrm{d}x = \frac{1}{\Delta x_\mathrm{P}} \left(\left(\alpha \frac{\partial T}{\partial x} \right)_e - \left(\alpha \frac{\partial T}{\partial x} \right)_w \right) + \frac{1}{\Delta x_\mathrm{P}} \int_{x_w}^{x_e} \sigma \, \mathrm{d}x. \tag{3.45}$$

Here, $\alpha = k/(\rho c_V)$ is the thermal diffusivity and $\sigma = \hat{q}/(\rho c_V)$ is a nameless quantity introduced for improved readability. We now introduce the *cell averages* $\bar{T}$ and $\bar{\sigma}$ defined by

$$\bar{T}_\mathrm{P} = \frac{1}{\Delta x_\mathrm{P}} \int_{x_w}^{x_e} T \, \mathrm{d}x, \quad \bar{\sigma}_\mathrm{P} = \frac{1}{\Delta x_\mathrm{P}} \int_{x_w}^{x_e} \sigma \, \mathrm{d}x. \tag{3.46}$$

Using these cell averages, Equation (3.45) can be rewritten as

$$\frac{\partial \bar{T}_\mathrm{P}}{\partial t} = \frac{1}{\Delta x_\mathrm{P}} \left(\left(\alpha \frac{\partial T}{\partial x} \right)_e - \left(\alpha \frac{\partial T}{\partial x} \right)_w \right) + \bar{\sigma}_\mathrm{P}. \tag{3.47}$$

Note that the equation above is still an exact rewritten version of the 1D heat equation (3.24). However, to make further progress, we must make some assumptions. The first approximation is one of the key ideas behind FVMs [8] – we approximate the cell-averaged temperature $\bar{T}_\mathrm{P}$ with the node value $T_\mathrm{P} = T(x_\mathrm{P})$, i.e., we assume that $\bar{T}_\mathrm{P} \approx T_\mathrm{P}$. In this derivation, we assume similarly that $\bar{\sigma}_\mathrm{P} \approx \sigma_\mathrm{P}$ as well. Inserting these approximations into Equation (3.47) yields

$$\frac{\partial T_\mathrm{P}}{\partial t} = \frac{1}{\Delta x_\mathrm{P}} \left(\left(\alpha \frac{\partial T}{\partial x} \right)_e - \left(\alpha \frac{\partial T}{\partial x} \right)_w \right) + \sigma_\mathrm{P}. \tag{3.48}$$

To discretize the right hand side of the equation above, we must approximate the face-evaluated quantities α and $\dfrac{\partial T}{\partial x}$ using node values (since we only have node values available). For the thermal diffusivity, we use harmonic averaging for reasons discussed in Appendix D.1. The harmonic average yields the following approximations:

$$\alpha_e \approx \frac{2\alpha_\mathrm{E}\alpha_\mathrm{P}}{\alpha_\mathrm{E} + \alpha_\mathrm{P}}, \quad \alpha_w \approx \frac{2\alpha_\mathrm{P}\alpha_\mathrm{W}}{\alpha_\mathrm{P} + \alpha_\mathrm{W}}. \tag{3.49}$$

For the spatial derivatives of the temperature, we use the central difference approximation given by

$$\left(\frac{\partial T}{\partial x} \right)_e \approx \frac{T_\mathrm{E} - T_\mathrm{P}}{x_\mathrm{E} - x_\mathrm{P}}, \quad \left(\frac{\partial T}{\partial x} \right)_w \approx \frac{T_\mathrm{P} - T_\mathrm{W}}{x_\mathrm{P} - x_\mathrm{W}}. \tag{3.50}$$

With these approximations, Equation (3.48) can be rewritten as

$$\frac{\partial T_{\mathrm{P}}}{\partial t}=\frac{1}{\Delta x_{\mathrm{P}}}\left(\frac{2\alpha_{\mathrm{E}}\alpha_{\mathrm{P}}}{\alpha_{\mathrm{E}}+\alpha_{\mathrm{P}}}\frac{T_{\mathrm{E}}-T_{\mathrm{P}}}{x_{\mathrm{E}}-x_{\mathrm{P}}}-\frac{2\alpha_{\mathrm{P}}\alpha_{\mathrm{W}}}{\alpha_{\mathrm{P}}+\alpha_{\mathrm{W}}}\frac{T_{\mathrm{P}}-T_{\mathrm{W}}}{x_{\mathrm{P}}-x_{\mathrm{W}}}\right)+\sigma_{\mathrm{P}}. \tag{3.51}$$

For a system consisting of N_j individual control volumes, we have N_j equations on the form of the one above. Using the integral and half-integral subscript notation defined earlier in the chapter, these equations can be rewritten as

$$\frac{\partial T_j}{\partial t}=\frac{1}{\Delta x_j}\left(\frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}+\alpha_j}\frac{T_{j+1}-T_j}{x_{j+1}-x_j}-\frac{2\alpha_j\alpha_{j-1}}{\alpha_j+\alpha_{j-1}}\frac{T_j-T_{j-1}}{x_j-x_{j-1}}\right)+\sigma_j \tag{3.52}$$

for $j=1,2,\ldots,N_j$. These equations constitute a system of N_j coupled equations which can be expressed in a compact way as

$$\frac{\partial \boldsymbol{T}(t)}{\partial t}=\boldsymbol{R}(\boldsymbol{T}(t)) \quad \text{with} \quad \boldsymbol{T}(t)=[T_1,\ldots,T_{N_j}]^T,\ \boldsymbol{R}(\boldsymbol{T}(t))=[R_1,\ldots,R_{N_j}]^T. \tag{3.53}$$

Here, the components of the vector $\boldsymbol{R}$ are given by

$$R_j=\frac{1}{\Delta x_j}\left(\frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}+\alpha_j}\frac{T_{j+1}-T_j}{x_{j+1}-x_j}-\frac{2\alpha_j\alpha_{j-1}}{\alpha_j+\alpha_{j-1}}\frac{T_j-T_{j-1}}{x_j-x_{j-1}}\right)+\sigma_j. \tag{3.54}$$

It now remains to discretize the temporal derivative $\frac{\partial T}{\partial t}$, for which use the Implicit Euler method. With this method, Equation (3.53) is discretized as

$$\frac{1}{\Delta t}\left(\boldsymbol{T}^{n+1}-\boldsymbol{T}^n\right)=\boldsymbol{R}\left(\boldsymbol{T}^{n+1}\right). \tag{3.55}$$

The jth equation of this system is

$$\begin{aligned}\frac{1}{\Delta t}\left(T_j^{n+1}-T_j^n\right)=&\\ &\frac{1}{\Delta x_j}\left(\frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}+\alpha_j}\frac{T_{j+1}^{n+1}-T_j^{n+1}}{x_{j+1}-x_j}-\frac{2\alpha_j\alpha_{j-1}}{\alpha_j+\alpha_{j-1}}\frac{T_j^{n+1}-T_{j-1}^{n+1}}{x_j-x_{j-1}}\right)+\sigma_j,\end{aligned} \tag{3.56}$$

which can be rewritten as

$$\begin{aligned}&-\frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}+\alpha_j}\frac{\Delta t}{\Delta x_j\Delta x_{j+1/2}}T_{j+1}^{n+1}+\\ &\quad\left(1+\frac{\Delta t}{\Delta x_j}\left(\frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}+\alpha_j}\frac{1}{\Delta x_{j+1/2}}+\frac{2\alpha_j\alpha_{j-1}}{\alpha_j+\alpha_{j-1}}\frac{1}{\Delta x_{j-1/2}}\right)\right)T_j^{n+1}+\\ &\qquad-\frac{2\alpha_j\alpha_{j-1}}{\alpha_j+\alpha_{j-1}}\frac{\Delta t}{\Delta x_j\Delta x_{j-1/2}}T_{j-1}^{n+1}=T_j^n+\sigma_j.\end{aligned} \tag{3.57}$$

From the equation above, it is clear that the system (3.53) can be written in matrix form

$$\mathbb{A}\boldsymbol{T}^{n+1}=\boldsymbol{b}(\boldsymbol{T}^n), \tag{3.58}$$

where $\mathbb{A}$ is a tridiagonal matrix. Its diagonal elements are

$$A_{j,j} = 1 + \frac{\Delta t}{\Delta x_j}\left(\frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}+\alpha_j}\frac{1}{\Delta x_{j+1/2}} + \frac{2\alpha_j\alpha_{j-1}}{\alpha_j+\alpha_{j-1}}\frac{1}{\Delta x_{j-1/2}}\right), \quad j = 1,\dots,N_j, \tag{3.59}$$

while its super- and subdiagonal elements are

$$A_{j,j+1} = -\frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}+\alpha_j}\frac{\Delta t}{\Delta x_j \Delta x_{j+1/2}}, \qquad j = 1,\dots,N_j-1, \tag{3.60}$$

$$A_{j-1,j} = -\frac{2\alpha_j\alpha_{j-1}}{\alpha_j+\alpha_{j-1}}\frac{\Delta t}{\Delta x_j \Delta x_{j-1/2}}, \qquad j = 2,\dots,N_j. \tag{3.61}$$

Finally, the components of the vector $\boldsymbol{b}$ on the right hand side of Equation (3.58) are

$$b_1 = T_1^n + \Delta t \sigma_1 + \frac{\alpha_a \Delta t}{\Delta x_1 \Delta x_{1/2}} T_a, \tag{3.62}$$

$$b_j = T_j^n + \Delta t \sigma_j, \qquad j = 2,\dots,N_j-1, \tag{3.63}$$

$$b_{N_j} = T_{N_j}^n + \Delta t \sigma_{N_j} + \frac{\alpha_b \Delta t}{\Delta x_{N_j} \Delta x_{N_j+1/2}} T_b. \tag{3.64}$$

3.3.2. Finite-Volume Method for Steady-State Problems

To obtain a set of discretized equations that is valid for steady-state problems, we simply let $\Delta t \to \infty$ in Equation (3.55). This is equivalent to letting $\frac{\partial T}{\partial t} \to 0$ in Equation (3.24), which is the characteristic of steady-state problems. In this limit, Equation (3.55) simplifies to

$$\boldsymbol{R}(\boldsymbol{T}^\infty) = \boldsymbol{0}, \tag{3.65}$$

where the jth equation reads

$$\frac{1}{\Delta x_j}\left(\frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}+\alpha_j}\frac{T_{j+1}^\infty - T_j^\infty}{x_{j+1}-x_j} - \frac{2\alpha_j\alpha_{j-1}}{\alpha_j+\alpha_{j-1}}\frac{T_j^\infty - T_{j-1}^\infty}{x_j - x_{j-1}}\right) + \sigma_j = 0. \tag{3.66}$$

These equations can be expressed as a tridiagonal system

$$\mathbb{A}\boldsymbol{T}^\infty = \boldsymbol{b}, \tag{3.67}$$

where the matrix $\mathbb{A}$ has elements

$$A_{j,j} = \frac{1}{\Delta x_j}\left(\frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}+\alpha_j}\frac{1}{\Delta x_{j+1/2}} + \frac{2\alpha_j\alpha_{j-1}}{\alpha_j+\alpha_{j-1}}\frac{1}{\Delta x_{j-1/2}}\right), \quad j = 1,\dots,N_j \tag{3.68}$$

$$A_{j,j+1} = -\frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}+\alpha_j}\frac{1}{\Delta x_j \Delta x_{j+1/2}}, \qquad j = 1,\dots,N_j-1 \tag{3.69}$$

$$A_{j-1,j} = -\frac{2\alpha_j\alpha_{j-1}}{\alpha_j+\alpha_{j-1}}\frac{1}{\Delta x_j \Delta x_{j-1/2}}, \qquad j = 2,\dots,N_j, \tag{3.70}$$

and the vector $\boldsymbol{b}$ has components

$$b_1 = \sigma_1 + \frac{\alpha_a}{\Delta x_1 \Delta x_{1/2}} T_a, \tag{3.71}$$

$$b_j = \sigma_j, \qquad j = 2, \dots, N_j - 1, \tag{3.72}$$

$$b_{N_j} = \sigma_{N_j} + \frac{\alpha_b}{\Delta x_{N_j} \Delta x_{N_j+1/2}} T_b. \tag{3.73}$$

3.3.3. Solving Tridiagonal Systems

To obtain numerical solutions using the Implicit Euler finite-volume method, we must solve the system (3.58) (for unsteady problems) or the system (3.67) (for steady-state problems). Both these systems are tri-diagonal, and can therefore be solved using the Tri-Diagonal Matrix Algorithm (TDMA) if the coefficient matrix $\mathbb{A}$ is diagonally dominant [8], i.e. if

$$|A_{j,j}| \geq \sum_{i \neq j} |A_{j,i}| \text{ for all } \qquad j = 1, \dots, N_j, \text{ and} \tag{3.74}$$

$$|A_{j,j}| > \sum_{i \neq j} |A_{j,i}| \text{ for at least one } j = 1, \dots, N_j. \tag{3.75}$$

In Appendix C, it is shown that the matrix $\mathbb{A}$, when defined as in Equation (3.58) or Equation (3.67), is diagonally dominant. We therefore use TDMA to solve the Equations (3.58) and (3.67).

4. Data-Driven Correction Methods

This chapter is mainly devoted to presenting the data-driven correction methods end-to-end learning and hybrid modelling. However, for the concept of corrections to make sense, it is imperative to have some way of measuring errors. The first section of this chapter, Section 4.1, therefore explains how the error of one temperature profile with respect to another temperature profile is measured in the present work. The presentations of end-to-end learning and hybrid modelling follow thereafter, in Sections 4.2 and 4.3, respectively.

4.1. Measuring Errors

One way of determining the quality of a numerical simulation scheme is to calculate the error of the numerical solutions it generates with respect to known reference solutions. In our context, this amounts to measuring the difference between numerical temperature profiles and known reference profiles. Throughout the present work, we encounter both discrete and continuous reference profiles, and finding a difference measure that is valid in both cases is not necessarily straight-forward. This section explains the difference measure used throughout the present work, which is based on the L_2 norm $\|\cdot\|_2$ for continuous functions

For a continuous, single-variable function $f : \mathbb{R} \rightarrow \mathbb{R}$, the L_2 norm on the interval $[x_a, x_b]$ is defined as

$$\|f\|_2 = \left(\int_{x_a}^{x_b} (f(x))^2 \,\mathrm{dx} \right)^{1/2}. \tag{4.1}$$

For any continuous temperature profile T, $\|T\|_2$ is then well-defined using the formula above. However, for a numerical temperature profile $\boldsymbol{T}_{\text{num}}$, which is discrete, the difference $\boldsymbol{T}_{\text{num}} - T$ is not well-defined, so $\|\boldsymbol{T}_{\text{num}} - T\|_2$ does not make sense. For this reason, we have to transform $\boldsymbol{T}_{\text{num}}$ into a continuous function T_{num} before we can compute the difference between it and any other temperature profile.

Consider a discrete, numerical temperature profile $\boldsymbol{T}_{\text{num}}$, and let $T_{\text{num},j}$ denote its jth component. With this notation, $T_{\text{num},j}$ describes the temperature at x_j, as calculated by the numerical scheme used to obtain $\boldsymbol{T}_{\text{num}}$. When transforming $\boldsymbol{T}_{\text{num}}$ into some continuous function T_{num}, it is then natural to require that $T_{\text{num}}(x_j) = T_{\text{num},j}$ for all $j = 1, \ldots, N_j$. Furthermore, T_{num} must respect the boundary conditions of the problem, so it is necessary that $T_{\text{num}}(x_a) = T_a$ and $T_{\text{num}}(x_b) = T_b$. For all other values of x, T_{num} can in principle be defined in infinitely many ways. We have chosen to define T_{num} such that it is linear between grid nodes, i.e., T_{num} is given by the expression

$$T_{\mathrm{num}} = \begin{cases} T_{\mathrm{num},j} & x \in \{x_j\}_{j=1,\ldots,N_j}, \\ T_a & x = x_a, \\ T_b & x = x_b, \\ \text{linear} & \text{otherwise.} \end{cases} \tag{4.2}$$

The L_2 norm $\|T_{\mathrm{num}} - T_{\mathrm{ref}}\|_2$ then provides a measure of the error of $\boldsymbol{T}_{\mathrm{num}}$ with respect to a *continuous* reference profile T_{ref}. Note that if the reference profile is instead a discrete profile $\boldsymbol{T}_{\mathrm{ref}}$, it must be linearized in the same way as $\boldsymbol{T}_{\mathrm{num}}$ before $\|T_{\mathrm{num}} - T_{\mathrm{ref}}\|_2$ can be calculated.

Our reason for choosing to define T_{num} as a linear function between grid nodes has to do with the origin of $\boldsymbol{T}_{\mathrm{num}}$. Throughout this work, all numerical temperature profiles are calculated using the Implicit Euler FVM scheme derived in Section 3.3. Furthermore, all simulations use equidistant grids, for which the Implicit Euler FVM has spatial order of accuracy $p_x = 2$ [34]. Consider also that the numerical temperature profile $\boldsymbol{T}_{\mathrm{num}}$ is defined to approximate some continuous reference profile T_{ref} at the grid nodes x_j. Thus, T_{num} should also be an approximation of T_{ref} and we want its order of accuracy to be the same as that of $\boldsymbol{T}_{\mathrm{num}}$. For this to occur, it can be shown that the order of accuracy of the interpolant used to obtain T_{num} from $\boldsymbol{T}_{\mathrm{num}}$ can be no less than $p_x - 1$. Since $p_x = 2$ in our case, the interpolant must be at least first order accurate, i.e. at least linear. Since we do not want to make our calculations more costly than necessary, we therefore choose to define T_{num} using linear interpolation.

We often want to compare the errors of numerical solutions of problems with different reference solutions. It is then necessary to *normalize* the error. The normalized error, which we denote E, is given by

$$E = \frac{\|T_{\mathrm{num}} - T_{\mathrm{ref}}\|_2}{\|T_{\mathrm{ref}}\|_2} = \left(\frac{\int\limits_{x_a}^{x_b} \left(T_{\mathrm{num}}(x) - T_{\mathrm{ref}}(x)\right)^2 \,\mathrm{dx}}{\int\limits_{x_a}^{x_b} \left(T_{\mathrm{ref}}(x)\right)^2 \,\mathrm{dx}} \right)^{1/2} \tag{4.3}$$

where T_{num} and T_{ref} are continuous functions (obtained through the linearization procedure described earlier in this section, if necessary).

The normalized error E, as given by Equation (4.3), can often by computed analytically. However, doing so is usually a cumbersome and time-consuming process vulnerable to human errors. Therefore, we choose instead to compute the integrals in Equation (4.3) using a numerical integration scheme. More specifically, we perform the numerical integration using the Gaussian quadrature approximation of order 5, as implemented in the Python library SciPy[1].

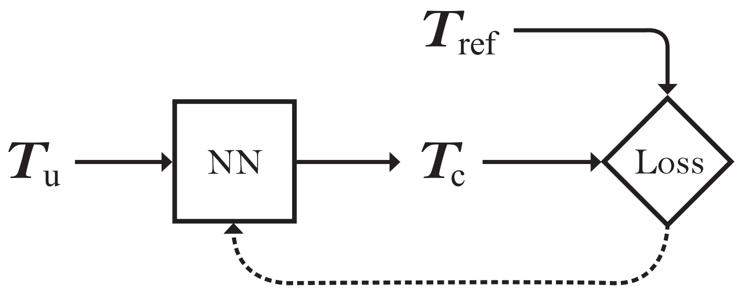

$\boldsymbol{T}_{\mathrm{ref}}$
$\boldsymbol{T}_{\mathrm{u}}$
NN
$\boldsymbol{T}_{\mathrm{c}}$
Loss

Figure 4.1.: End-to-end Learning: Information flow during training of neural network used in end-to-end learning. An uncorrected temperature profile $\boldsymbol{T}_{\mathrm{u}}$ is mapped to a corrected temperature profile $\boldsymbol{T}_{\mathrm{c}}$ by a neural network (NN). The corrected profile $\boldsymbol{T}_{\mathrm{c}}$ is compared to a reference profile $\boldsymbol{T}_{\mathrm{ref}}$ by some loss function. The dashed arrow represents the backpropagation algorithm used to update the parameters of the neural network.

4.2. End-to-end Learning

End-to-end learning is the most basic correction approach studied in the present work. For this approach, a neural network is used to define a direct mapping from uncorrected numerical solutions to corrected numerical solutions. The uncorrected solutions are given by a physics-based simulation scheme, and these solutions are approximations of corresponding reference solutions. In real-world applications, the reference solutions are typically measurements of the system being modelled. On the other hand, in academic work, the reference solutions might also be solutions from a more accurate physics-based simulation scheme. Ideally, we want the neural network to learn a mapping such that the corrected solutions are equal to the corresponding reference solutions.

In the context of one-dimensional heat conduction problems, the uncorrected numerical solutions are uncorrected temperature profiles which we label $\boldsymbol{T}_{\mathrm{u}}$. The corresponding corrected profiles and reference profiles are labelled $\boldsymbol{T}_{\mathrm{c}}$ and $\boldsymbol{T}_{\mathrm{ref}}$, respectively. When the heat conduction problem is defined using Dirichlet BCs, the boundary temperatures T_a and T_b represent information that is generally useful to the neural network. It is then sensible to include these temperatures in the input vector $\boldsymbol{T}_{\mathrm{u}}$ of the neural network. However, since these temperatures are known, there is no need to have them included in the neural network output $\boldsymbol{T}_{\mathrm{c}}$. The desired neural network mapping, denoted NN, for 1D heat conduction problems with Dirichlet BCs, is then given by

$$\begin{aligned} \mathrm{NN} : \mathbb{R}^{N_j+2} &\to \mathbb{R}^{N_j} \quad \text{such that} \quad \boldsymbol{T}_{\mathrm{c}} = \boldsymbol{T}_{\mathrm{ref}}. \\ \boldsymbol{T}_{\mathrm{u}} &\mapsto \boldsymbol{T}_{\mathrm{c}} \end{aligned} \tag{4.4}$$

[1]`https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.fixed_quad.html`

Here, N_j is the number of grid cells used to define the numerical solution, and $\mathbb{R}$ is the set of real numbers. The problem of training a neural network to find such a mapping can be solved as a supervised learning problem, for which the flow of information during training is shown in Figure 4.1. In the present work, we utilize the mean squared error loss function to measure the dissimilarity between the neural network outputs $\boldsymbol{T}_{\mathrm{c}}$ and the target outputs $\boldsymbol{T}_{\mathrm{ref}}$. However, other loss function can be used as well, as is discussed in Section 7.3.

From a human perspective, the task of the neural network in an end-to-end problem might seem deceptively easy, considering that the desired temperature corrections are likely small in comparison to the magnitude of the temperatures involved. However, the neural network does not know this; it has to learn on its own that it is generally a good idea to have an output that is somewhat similar to the input. Thus, end-to-end learning is not necessarily able to take full advantage of all the information that is included in the physics-based simulation scheme used to generate the uncorrected temperature profiles. An other issue is lack of interpretability due to the black box-like nature of neural networks. These shortcomings provide the motivation for exploring other correction approaches.

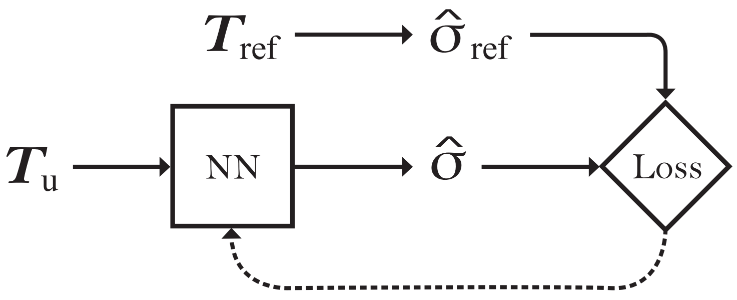
4.3. Hybrid Modelling

In this section, we present an alternative approach to correcting numerical temperature profiles. For this new approach, the physics-based simulations and the data-driven corrections are more intertwined, and we therefore label this approach *hybrid modelling.* Just like end-to-end learning, hybrid modelling also utilizes a neural network which takes an uncorrected numerical solution as input. The difference is that, for hybrid modelling, the neural network output is taken to be a so-called *correction source term*, denoted $\hat{\boldsymbol{\sigma}}$, rather than a corrected solution. The correction source term is then added to the system of equations that is solved by the simulation scheme, in order to obtain a modified system of equations. This modified system of equations is then solved (using the same solver as was used for solving the original system) to obtain the corrected solution. The ideal neural network output, which we refer to as the *reference* correction source term and denote $\hat{\boldsymbol{\sigma}}_{\mathrm{ref}}$, is the correction source term that results in the corrected solution being equal to the reference solution. Again, the reference solution can either be given by measurements, or by a more accurate physics-based simulation.

As in Section 4.2, we use the symbols $\boldsymbol{T}_{\mathrm{u}}$, $\boldsymbol{T}_{\mathrm{c}}$ and $\boldsymbol{T}_{\mathrm{ref}}$ to denote, respectively, the uncorrected solution, corrected solution and reference solution of a 1D heat conduction problem with Dirichlet BCs. Following the reasoning of Section 4.2, the mapping (again denoted NN) to be learned by the neural network, is then given by

$$\begin{aligned} \mathrm{NN} : \mathbb{R}^{N_j+2} &\to \mathbb{R}^{N_j} \quad \text{such that} \quad \boldsymbol{T}_{\mathrm{c}} = \boldsymbol{T}_{\mathrm{ref}}. \\ \boldsymbol{T}_{\mathrm{u}} &\mapsto \hat{\boldsymbol{\sigma}} \end{aligned} \tag{4.5}$$

Note that the dependence of $\boldsymbol{T}_{\mathrm{c}}$ on $\hat{\boldsymbol{\sigma}}$ is what constitutes the connection between the mapping and the condition it must fulfill.



$\boldsymbol{T}_{\mathrm{ref}}$
$\hat{\sigma}_{\mathrm{ref}}$
$\boldsymbol{T}_{\mathrm{u}}$
NN
$\hat{\sigma}$
Loss

Figure 4.2.: Hybrid modelling: Information flow during training of neural network used in hybrid modelling. An uncorrected temperature profile $\boldsymbol{T}_{\mathrm{u}}$ is mapped to a correction source term $\hat{\boldsymbol{\sigma}}$ by a neural network (NN), and a reference correction source term $\hat{\boldsymbol{\sigma}}_{\mathrm{ref}}$ is computed from a reference temperature profile $\boldsymbol{T}_{\mathrm{ref}}$. A loss function is used to compare the neural network output $\hat{\boldsymbol{\sigma}}$ and the target output $\hat{\boldsymbol{\sigma}}_{\mathrm{ref}}$. The dashed arrow represents the backpropagation algorithm used to update the parameters of the neural network.

As for end-to-end learning, the problem of training a neural network to learn the desired mapping can be formulated as a supervised learning problem. However, in the case of hybrid modelling, we cannot necessarily use a loss function that directly compares $\boldsymbol{T}_{\mathrm{c}}$ and $\boldsymbol{T}_{\mathrm{ref}}$. This is because calculating $\boldsymbol{T}_{\mathrm{c}}$ from $\hat{\boldsymbol{\sigma}}$ requires use of the physics-based simulation scheme. We want our correction approach to be compatible with pre-existing simulation scheme implementations, and, since these are generally not implemented in any machine learning framework, it is impossible to perform backpropagation through them. Without backpropagation, the neural network is unable to learn, and this prevents us from using loss functions that require $\boldsymbol{T}_c$ to be computed. To circumnavigate this issue, we observe that requiring $\boldsymbol{T}_{\mathrm{c}} = \boldsymbol{T}_{\mathrm{ref}}$ is, by construction, equivalent to requiring $\hat{\boldsymbol{\sigma}} = \hat{\boldsymbol{\sigma}}_{\mathrm{ref}}$. This allows us to reformulate the definition (4.5) of the ideal mapping in the following way:

$$\begin{aligned} \mathrm{NN} : \mathbb{R}^{N_j+2} &\to \mathbb{R}^{N_j} \quad \text{such that} \quad \hat{\boldsymbol{\sigma}} = \hat{\boldsymbol{\sigma}}_{\mathrm{ref}}. \\ \boldsymbol{T}_{\mathrm{u}} &\mapsto \hat{\boldsymbol{\sigma}} \end{aligned} \tag{4.6}$$

With this alternative formulation, the most natural loss functions are those comparing $\hat{\boldsymbol{\sigma}}$ and $\hat{\boldsymbol{\sigma}}_{\mathrm{ref}}$. For such loss functions, backpropagation is not an issue. However, they necessitate a method of calculating $\hat{\boldsymbol{\sigma}}_{\mathrm{ref}}$ from $\boldsymbol{T}_{\mathrm{ref}}$. Given that such a method exists, we have a well-posed supervised learning problem for which the flow of information during training is illustrated in Figure 4.2. For the heat conduction problems considered in this work, the reference correction source term can be defined as the residual of the discretized governing equation when the reference solution is inserted in the place of the uncorrected numerical solution. This is discussed further in Section 4.3.1 for steady-state problems and in Section 4.3.2 for unsteady problems.

It is clear that hybrid modelling is a more involved correction approach than is end-to-end learning. As compensation, hybrid modelling also has some benefits. One benefit is that the task of the neural network is in some sense simpler than it is for the end-to-end learning approach, even though this claim might seem counter-intuitive at first. A human typically has little intuition about how to construct a correction source term from a temperature profile, and so would probably prefer to construct a corrected temperature profile directly instead. However, a neural network does not have any intuition about either case, and therefore prefers whichever case has the simplest mapping between input and target output. In general, generating correction source terms is simpler, because correction source terms only depend on physics that are *not* accounted for by the physics-based simulation scheme. On the other hand, corrected temperature profiles must account for physics that is accounted for *and* physics that is not accounted for by the simulation scheme. Evidently, end-to-end learning then requires the neural network to learn more physics than is the case for hybrid modelling. This makes end-to-end learning generally more difficult than hybrid modelling. Another benefit of hybrid modelling is the fact that the correction sources terms can be analyzed using e.g. regression techniques for increased interpretability. Furthermore, imposing regularization constraints on the correction source term, either through the loss function or through post-processing, can possibly be used to increase the stability of corrected simulations.

4.3.1. Reference Correction Source Term for Steady-State Problems

For steady-state problems, the Implicit Euler FVM discretization of the heat equation is given by Equation (3.67). Suppose now that we have a reference solution $\boldsymbol{T}_{\mathrm{ref}}^{\infty}$ (e.g. from real-world measurements) for some steady-state problem. If we insert this reference temperature profile into Equation (3.67), we generally find that the right hand side of the equation does not equal the left hand side. That is, in mathematical terms, we find

$$\mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{\infty} - \boldsymbol{b} \neq 0, \tag{4.7}$$

with $\mathbb{A}$ and $\boldsymbol{b}$ defined as in Equation (3.67). We now define the reference correction source term as

$$\hat{\boldsymbol{\sigma}}_{\mathrm{ref}} = \mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{\infty} - \boldsymbol{b}. \tag{4.8}$$

Adding $\hat{\boldsymbol{\sigma}}_{\mathrm{ref}}$ to the right hand side of Equation (3.67) yields a new, modified system of equations and we define the corrected temperature profile $\boldsymbol{T}_{\mathrm{c}}^{\infty}$ as the solution of this system. Mathematically, this means that we have

$$\mathbb{A}\boldsymbol{T}_{\mathrm{c}}^{\infty} = \boldsymbol{b} + \hat{\boldsymbol{\sigma}}_{\mathrm{ref}}. \tag{4.9}$$

Inserting the definition of $\hat{\boldsymbol{\sigma}}$ into the system above yields

$$\mathbb{A}\boldsymbol{T}_{\mathrm{c}}^{\infty} = \boldsymbol{b} + \hat{\boldsymbol{\sigma}}_{\mathrm{ref}} = \boldsymbol{b} + (\mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{\infty} - \boldsymbol{b}) = \mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{\infty}. \tag{4.10}$$

Since $\mathbb{A}$ has full rank, $\mathbb{A}\boldsymbol{x} = \mathbb{A}\boldsymbol{y} \iff \boldsymbol{x} = \boldsymbol{y}$ for any vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. We can thereby conclude that solving the system (4.9) yields $\boldsymbol{T}_{\mathrm{c}}^{\infty} = \boldsymbol{T}_{\mathrm{ref}}^{\infty}$. Thus, Equation (4.8) defines a reference correction source term which guarantees that the corrected temperature profile is equal to the reference profile at all grid points for steady-state problems.

4.3.2. Reference Correction Source Term for Unsteady Problems

The derivation of the reference correction source term for unsteady problems largely follows the derivation for steady-state problems, though some details differ. Recall that for an unsteady problem, our Implicit Euler FVM discretization results in the system of equations given by Equation (3.58). Suppose now that we have two reference profiles, $\boldsymbol{T}_{\mathrm{ref}}^{n}$ and $\boldsymbol{T}_{\mathrm{ref}}^{n+1}$, corresponding to two successive time levels, t^n and t^{n+1}. Inserting these profiles into Equation (3.58), one finds that the right hand side and the left hand side of the equation are not equal in general; mathematically, we have

$$\mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{n+1} - \boldsymbol{b}(\boldsymbol{T}_{\mathrm{ref}}^{n}) \neq 0. \tag{4.11}$$

Motivated by the success in the steady-state case, we define the reference correction source term as

$$\hat{\boldsymbol{\sigma}}_{\mathrm{ref}} = \mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{n+1} - \boldsymbol{b}(\boldsymbol{T}_{\mathrm{ref}}^{n}), \tag{4.12}$$

which is analogous to Equation (4.8). If we now add this reference correction source term to the right hand side of Equation (3.58), and let $\boldsymbol{T}_{\mathrm{c}}$ denote the solution of the modified system, we get

$$\mathbb{A}\boldsymbol{T}_{\mathrm{c}}^{n+1} = \boldsymbol{b}(\boldsymbol{T}_{\mathrm{c}}^{n}) + \hat{\boldsymbol{\sigma}}_{\mathrm{ref}} = \boldsymbol{b}(\boldsymbol{T}_{\mathrm{c}}^{n}) + \left(\mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{n+1} - \boldsymbol{b}(\boldsymbol{T}_{\mathrm{ref}}^{n})\right). \tag{4.13}$$

We see that in the case of unsteady problems, the modified system of equations includes both $\boldsymbol{b}(\boldsymbol{T}_{\mathrm{c}}^{n})$ and $\boldsymbol{b}(\boldsymbol{T}_{\mathrm{ref}}^{n})$, and these are not equal in general. Thus, the correction source term defined by Equation (4.12) does not necessarily yield $\boldsymbol{T}_{\mathrm{c}}^{n+1} = \boldsymbol{T}_{\mathrm{ref}}^{n+1}$. However, it does yield $\boldsymbol{T}_{\mathrm{c}}^{n+1} = \boldsymbol{T}_{\mathrm{ref}}^{n+1}$ under the assumption that $\boldsymbol{T}_{\mathrm{c}}^{n} = \boldsymbol{T}_{\mathrm{ref}}^{n}$. This might seem like an unreasonable assumption at first sight, but $\boldsymbol{T}_{\mathrm{c}}^{n} = \boldsymbol{T}_{\mathrm{ref}}^{n}$ does actually hold in one important case – the case when the correction source term is included in the generation of $\boldsymbol{T}_{\mathrm{c}}$ from the start of the simulation process. This can be proven by induction as follows: Suppose that $\boldsymbol{T}_{\mathrm{ref}}^{0} = \boldsymbol{T}_{\mathrm{c}}^{0} = \boldsymbol{T}^{0}$ for some $\boldsymbol{T}^{0}$. Then,

$$\begin{aligned}\mathbb{A}\boldsymbol{T}_{\mathrm{c}}^{1} &= \boldsymbol{b}(\boldsymbol{T}_{\mathrm{c}}^{0}) + \hat{\boldsymbol{\sigma}}_{\mathrm{ref}}^{0} = \boldsymbol{b}(\boldsymbol{T}_{\mathrm{c}}^{0}) + \left(\mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{1} - \boldsymbol{b}(\boldsymbol{T}_{\mathrm{ref}}^{0})\right)\\ &= \boldsymbol{b}(\boldsymbol{T}^{0}) + \left(\mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{1} - \boldsymbol{b}(\boldsymbol{T}^{0})\right) = \mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{1},\end{aligned}$$

so we evidently have $\boldsymbol{T}_{\mathrm{c}}^{0} = \boldsymbol{T}_{\mathrm{ref}}^{0} \implies \boldsymbol{T}_{\mathrm{c}}^{1} = \boldsymbol{T}_{\mathrm{ref}}^{1}$ since $\mathbb{A}$ has full rank. Now suppose instead that $\boldsymbol{T}_{\mathrm{ref}}^{n} = \boldsymbol{T}_{\mathrm{c}}^{n} = \boldsymbol{T}^{n}$ for some $\boldsymbol{T}^{n}$. Analogously to the calculation above, we then have

$$\begin{aligned}\mathbb{A}\boldsymbol{T}_{\mathrm{c}}^{n+1} &= \boldsymbol{b}(\boldsymbol{T}_{\mathrm{c}}^{n}) + \hat{\boldsymbol{\sigma}}_{\mathrm{ref}}^{n} = \boldsymbol{b}(\boldsymbol{T}_{\mathrm{c}}^{n}) + \left(\mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{n+1} - \boldsymbol{b}(\boldsymbol{T}_{\mathrm{ref}}^{n})\right)\\ &= \boldsymbol{b}(\boldsymbol{T}^{n}) + \left(\mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{n+1} - \boldsymbol{b}(\boldsymbol{T}^{n})\right) = \mathbb{A}\boldsymbol{T}_{\mathrm{ref}}^{n+1},\end{aligned}$$

Consequently, $\boldsymbol{T}_{\mathrm{c}}^{n} = \boldsymbol{T}_{\mathrm{ref}}^{n} \implies \boldsymbol{T}_{\mathrm{c}}^{n+1} = \boldsymbol{T}_{\mathrm{ref}}^{n+1}$. By induction, we can then conclude $\boldsymbol{T}_{\mathrm{c}}^{n} = \boldsymbol{T}_{\mathrm{ref}}^{n}\ \forall n$ when $\boldsymbol{T}_{\mathrm{c}}^{0} = \boldsymbol{T}_{\mathrm{ref}}^{0}$ and $\boldsymbol{T}_{\mathrm{c}}^{n}$ is calculated using Equation (4.13) for all time levels $t^n,\ n = 1, 2, \ldots$. These assumptions do not impose any significant limitations on the kinds of problems we can solve using our hybrid modelling approach. We therefore use the reference correction source term as defined in Equation (4.12) when training neural networks for use with hybrid modelling on unsteady problems.

5. Grid Refinement Studies

Before the data-driven correction approaches presented in the previous chapter can be tested, it is imperative that we validate the implementation of our numerical simulation scheme. We perform this validation by conducting the grid refinement studies presented in this chapter. In each study, an empirical value of either the temporal convergence rate or the spatial convergence rate is calculated for a simple problem with known exact solution. If the numerical simulation scheme is implemented correctly, the observed empirical convergence rates should match the convergence rates found in the literature. In this chapter, four different grid refinement studies are presented, and they can be characterized as follows:

Study 1: Spatial grid refinement study for steady-state problem with constant parameters.

Study 2: Spatial grid refinement study for steady-state problem with spatially varying conductivity.

Study 3: Spatial grid refinement study for unsteady problem with constant parameters.

Study 4: Temporal grid refinement study for unsteady problem with constant parameters.

While the derivation of our numerical scheme (cf. Section 3.3) is valid for non-uniform spatial discretizations, we will only use equidistant grids in our machine learning experiments. For this reason, we also only perform grid refinement studies for equidistant grids. For an equidistant grid, it can be shown that the Implicit Euler finite-volume method is second order accurate in space and first order accurate in time [34]. This means that the leading terms of the (normalized) error E of the numerical solution with respect to the exact solution are proportional to Δx^2 and Δt. Mathematically, we write this as

$$E \propto \mathcal{O}(\Delta x^2, \Delta t). \tag{5.1}$$

As explained in Section 4.1, we use the L2-norm to calculate the normalized error E of the numerical solution (which is discrete) with respect to the exact solution (which is continuous).

All grid refinement studies in this work follow the same procedure. For a *spatial* grid refinement study, this procedure can be described using the list below.

1. Define the physical parameters x_a, x_b, k, A and $\hat{q}$. For unsteady problems, c_V and ρ must be defined as well.

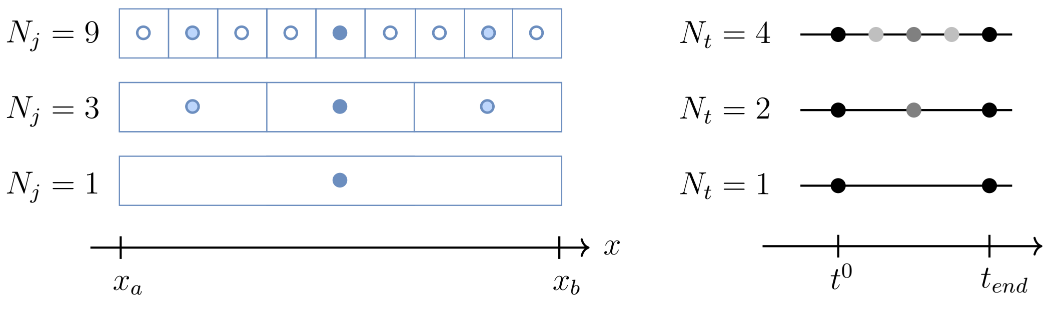
2. Choose initial conditions (ICs) and boundary conditions (BCs).
3. Find the exact solution of the 1D heat equation (3.24) for the chosen parameter values, ICs and BCs.
4. For unsteady problems, choose a fixed value of Δt such that the error of the temporal discretization is much smaller than the error of the spatial discretization. (This point is skipped for steady-state problems, where $\Delta t = \infty$ always.)
5. Choose an initial value of Δx, perform a simulation and calculate the error of the numerical solution with respect to the exact solution
6. Divide Δx by a constant factor ζ and perform another simulation using the new value of Δx. We refer to ζ as the *grid refinement factor*.
7. Calculate the error of the new numerical solution with respect to the exact solution, and find the empirical order of convergence.
8. Repeat points 6 and 7 until a clear trend emerges. (In our studies, we performed six repetitions, thus obtaining a total of seven error values for each study.)

The procedure for a *temporal* grid refinement study also follows the list above closely. The only difference is that the roles of Δx and Δt are reversed; one must choose a fixed value of Δx and gradually refine Δt.

When performing a spatial grid refinement study of a finite-volume method (FVM), it is important that all cell faces used in one grid remain cell faces in all finer grids. FVMs rely on approximating the cell face values in Equation (3.24), and it is entirely possible that the same approximation method will work better for certain placements of the cell faces than for others. If this is the case, simply moving the existing cell faces can reduce (or increase) the error of the numerical solution. Because of this effect, we might not obtain the expected convergence rate if cell faces are moved as the grid is refined, even if our implementation is correct. Due to similar considerations, the grid nodes used in one grid should also remain grid nodes in all finer grids. This is because FVMs rely on approximating cell-averaged values using grid node values, and certain grid node placements may then yield better approximations than others. To ensure that all cell faces remain cell faces and all grid nodes remain grid nodes as grids are refined, we use the grid refinement factor $\zeta = 3$ in all our spatial grid refinement studies. The relation between spatial grids refined using $\zeta = 3$ is shown in Figure 5.1a.

In the temporal discretization, the role of time levels is analogous to that of grid nodes in the spatial discretization. Therefore, it is important that time levels included in one simulation are also included in simulations of higher temporal resolution. To ensure this, we use $\zeta = 2$ in our temporal grid refinement study, thereby obtaining the temporal grid refinement illustrated in Figure 5.1b.

A central part of any grid refinement study is obtaining the empirical convergence rates. To obtain the spatial convergence rate of a numerical scheme empirically, we can compute the ratio of the errors E of two different numerical solutions; one calculated



(a) Spatial grid refinement. (b) Temporal grid refinement.

Figure 5.1.: Grid Refinement Studies: Spatial grid refinement with refinement factor $\zeta = 3$ (left) and temporal grid refinement with refinement factor $\zeta = 2$ (right). Note the alignment of cell faces, grid nodes and time levels. The design of Figure 5.1a is adapted from a design by Alexandra Metallinou Log.

using N_j grid cells and another calculated using N_j/ζ grid cells. Suppose the spatial convergence rate of our method is p_x. We then have

$$\frac{E(N_j/\zeta)}{E(N_j)} = \frac{E(\zeta\Delta x)}{E(\Delta x)} = \frac{(\zeta\Delta x)^{p_x}}{(\Delta x)^{p_x}}, \tag{5.2}$$

which implies that

$$\log_\zeta\left(\frac{E(N_j/\zeta)}{E(N_j)}\right) = \log_\zeta\left(\frac{(\zeta\Delta x)^{p_x}}{(\Delta x)^{p_x}}\right) = p_x \log_\zeta\left(\frac{\zeta\Delta x}{\Delta x}\right) = p_x \log_\zeta(\zeta) = p_x. \tag{5.3}$$

The temporal convergence rate can be obtained analogously. Let N_t be the number of time levels in our simulation; we then have

$$p_t = \log_\zeta\left(\frac{E(N_t/\zeta)}{E(N_t)}\right). \tag{5.4}$$

5.1. Grid Refinement Study #1

In the first grid refinement study, we investigate the spatial convergence rate of our Implicit Euler FVM implementation. In this study, we solve a steady-state problem with physical parameters that are constant throughout the domain.

5.1.1. Configuration and Exact Solution

Table 5.1.: Grid Refinement Study #1 (steady state, constant parameters): Configuration of discretization parameters, physical parameters, BCs and IC.

Parameter	Setting
Number of time levels	$N_t = 1$
Initial number of grid cells	$N_j = 5$
Grid refinement factor	$\zeta = 3$
Domain boundaries	$x_a = -1\,\mathrm{m},\ x_b = 1\,\mathrm{m}$
Conductivity	$k = 1\,\mathrm{W/Km}$
Cross-sectional area	$A = 1\,\mathrm{m}^2$
Heat generation rate	$\hat{q} = 2\,\mathrm{W/m}^3$
Boundary conditions	$T(x = x_a, t) = 0\,\mathrm{K},\ T(x = x_b, t) = 0\,\mathrm{K}$

Table 5.1 lists all parameter choices used in this experiment. Inserting these parameters into Equation (3.41) yields the exact solution

$$T_{\text{exact}}(x) = 1 - x^2. \tag{5.5}$$

For an illustration of this exact solution and its full derivation, see Appendix D.

5.1.2. Results and Discussion

Figure 5.2 shows the normalized error E of the numerical solution $\boldsymbol{T}_{\text{num}}$ with respect to the exact solution T_{exact} as a function of the number of grid cells N_j. The numerical solution was calculated using the Implicit Euler FVM, and the error E was calculated using Equation (4.3). Table 5.2 shows the value of E for each value of N_j. The table also includes the empirical spatial convergence rates p_x, as calculated using the formula (5.3) with $\zeta = 3$. We see that the empirically calculated convergence rate almost exactly matches the theoretical convergence rate of $p_x = 2$ for all grid refinements. This is a promising result, but we cannot conclude that our implementation has been verified on the basis of one result alone. More grid refinement studies must be performed before we can draw any conclusions.

Table 5.2.: Grid Refinement Study #1 (steady state, constant parameters): normalized L_2 errors E and corresponding empirical convergence rates p_x for the computed FVM-approximation of the steady-state temperature profile.

N_j	$E(N_j)$	p_x
5	3.0307e-2	-
15	3.3675e-3	2.0000
45	3.7417e-4	2.0000
135	4.1574e-5	2.0000
405	4.6193e-6	2.0000
1215	5.1326e-7	2.0000
3645	5.7100e-8	1.9988

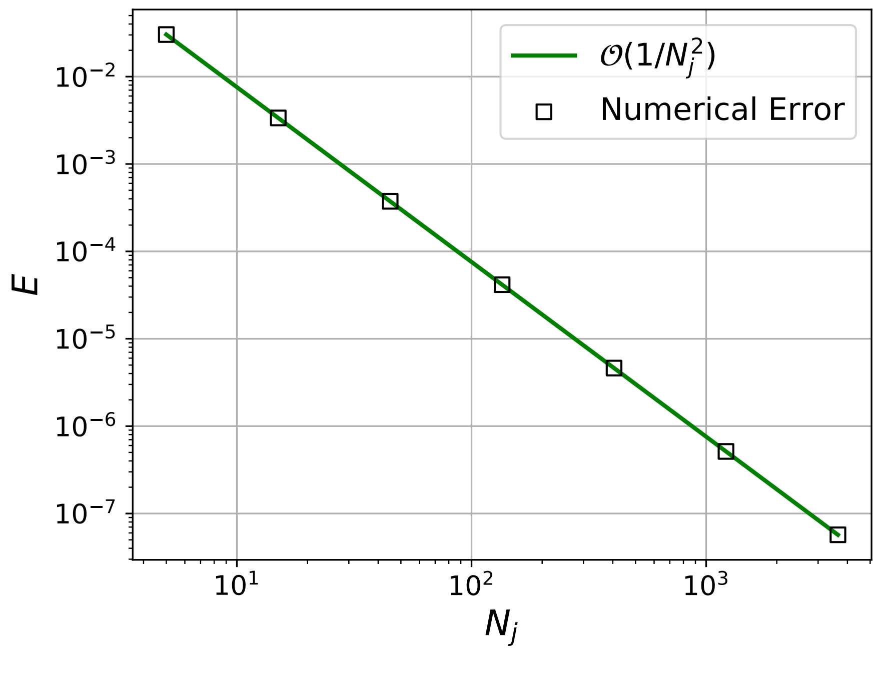


Figure 5.2.: Grid Refinement Study #1 (steady state, constant parameters): normalized L_2 error E of the computed FVM-approximation of the steady-state temperature profile.

5.2. Grid Refinement Study #2

The second grid refinement study also considers the spatial convergence rate of our FVM implementation on a steady-state problem. The difference with respect to Grid Refinement Study #1 is that we now introduce a spatially varying conductivity k.

5.2.1. Configuration and Exact Solution

Table 5.3.: Grid Refinement Study #2 (steady state, varying conductivity): Configuration of discretization parameters, physical parameters, BCs and IC.

Parameter	Setting
Number of time levels	$N_t = 1$
Initial number of grid cells	$N_j = 5$
Grid refinement factor	$\zeta = 3$
Domain boundaries	$x_a = 0\,\mathrm{m},\ x_b = 0.01\,\mathrm{m}$
Conductivity	$k = 5000x^2 - 55x + 0.25\,\mathrm{W/Km}$
Cross-sectional area	$A = 1\,\mathrm{m}^2$
Heat generation rate	$\hat{q} = 0\,\mathrm{W/m}^3$
Boundary conditions	$T(x = x_a, t) = 400\,\mathrm{K},\ T(x = x_b, t) = 600\,\mathrm{K}$
Initial condition	$T(x, t = 0) = 500\,\mathrm{K}\ \forall x \in (x_a, x_b)$

Table 5.3 lists all parameter choices used in this experiment. Inserting these parameters into Equation (3.36) yields the exact solution

$$T_{\text{exact}}(x) = 400 + 200\frac{\arctan\left(\frac{2000x-11}{\sqrt{79}}\right) - \arctan\left(\frac{-11}{\sqrt{79}}\right)}{\arctan\left(\frac{9}{\sqrt{79}}\right) - \arctan\left(\frac{-11}{\sqrt{79}}\right)}. \tag{5.6}$$

For an illustration of this exact solution and its full derivation, see Appendix D.

5.2.2. Results & Discussion

Figure 5.3 shows the normalized error E of the numerical solution $\boldsymbol{T}_{\text{num}}$ with respect to the exact solution T_{exact} as a function of the number of grid cells N_j. The numerical solution was calculated using the Implicit Euler FVM, and the error E was calculated using Equation (4.3). Table 5.4 shows the value of E for each value of N_j. The table also includes the empirical spatial convergence rates p_x, as calculated using the formula (5.3) with $\zeta = 3$. Again, we see that the result mostly match well with the expected convergence rate. The only significant outlier is the empirical convergence rate when N_j is increased from 5 to 15. It is known that the empirical convergence rate can be somewhat unreliable for small values of N_j, because each grid node is comparatively more important in this case. As the grid is refined, some of the new nodes might be placed at particularly advantageous or disadvantageous locations by chance, thus causing

Table 5.4.: Grid Refinement Study #2 (steady state, varying conductivity): normalized L_2 errors E and corresponding empirical convergence rates p_x for the computed FVM-approximation of the steady-state temperature profile.

N_j	$E(N_j)$	p_x
5	1.9255e-3	-
15	2.3815e-4	1.9024
45	2.7077e-5	1.9791
135	3.0272e-6	1.9944
405	3.3699e-7	1.9983
1215	3.7471e-8	1.9993
3645	4.1897e-9	1.9943

some variability in the observed empirical convergence rate. For larger values of N_j, these chance effects related to node placements tend to cancel out, leaving more reliable results. Accordingly, Table 5.4 shows that the empirical convergence rate becomes closer to 2 as N_j increases, except for in the final refinement where p_x decreases slightly. This decrease in p_x could be a sign that the error caused by the temporal discretization is beginning to influence the results, but the decrease is too small for any hard conclusions to be drawn.

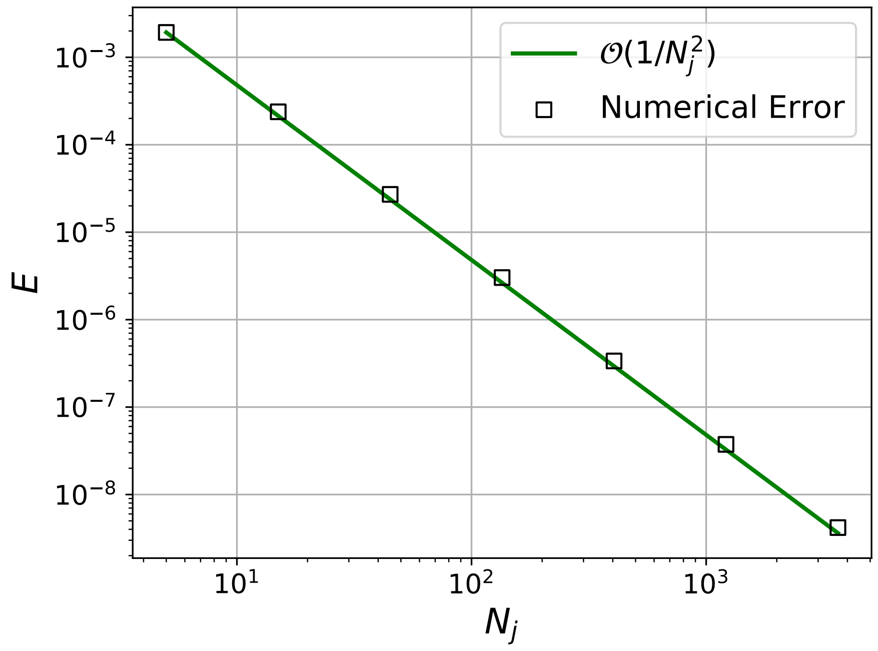


Figure 5.3.: Grid Refinement Study #2 (steady state, varying conductivity): normalized L_2 error E of the computed FVM-approximation of the steady-state temperature profile

5.3. Grid Refinement Study #3

In the third grid refinement study, we move on to an unsteady problem while still considering the spatial convergence rate of our implementation. As in Grid Refinement Study #1, we use constant physical parameters for this experiment. Unlike the previous grid refinement studies, this study encompasses two separate runs using different time steps. Two runs were performed because the first one used a time step which was not sufficiently small. We still include this run because it still provides useful insight into the behaviour of our implementation.

5.3.1. Configuration and Exact Solution

Table 5.5.: Grid Refinement Study #3 (spatial refinement, unsteady):
Configuration of discretization parameters, physical parameters, BCs and IC.

Parameter	Setting
Number of time levels (Run 1)	$N_t = 2 \cdot 10^4$
Number of time levels (Run 2)	$N_t = 2 \cdot 10^5$
End time of simulation	$t_{end} = 5\,\mathrm{s}$
Initial number of grid cells	$N_j = 5$
Refinement factor	$\zeta = 3$
Domain boundaries	$x_a = 0\,\mathrm{m},\ x_b = 1\,\mathrm{m}$
Conductivity	$k = 2500\,\mathrm{W/Km}$
Specific heat capacity	$c_V = 2000\,\mathrm{J/kgK}$
Density	$\rho = 1300\,\mathrm{kg/m^3}$
Cross-sectional area	$A = 1\,\mathrm{m^2}$
Heat generation rate	$\hat{q} = 0\,\mathrm{W/m^3}$
Boundary conditions	$T(x = x_a, t) = 250\,\mathrm{K},\ T(x = x_b, t) = 250\,\mathrm{K}$
Initial condition	$T(x, t = 0) = 250 + 50\sin(2\pi x)\,\mathrm{K}\ \forall x \in (x_a, x_b)$

Table 5.5 lists all parameter choices used in this experiment. Since k, c_V and ρ are all constants, we can use the second-to-last equation on page 70 of [35] to derive the exact solution

$$T_{\mathrm{exact}}(x, t = t_{end}) = 250 + 50\sin(2\pi x)\exp\{-\pi^2/52\}. \tag{5.7}$$

For an illustration of this exact solution and its full derivation, see Appendix D.

5.3.2. Results & Discussion

Figure 5.4 shows the normalized error E of the numerical solution $\boldsymbol{T}_{\mathrm{num}}$ with respect to the exact solution T_{exact} as a function of the number of grid cells N_j for both Run 1 (left) and Run 2 (right). The numerical solution was calculated using the Implicit Euler FVM, and the error E was calculated using Equation (4.3). Table 5.6 shows the value of E for each value of N_j in both runs. The table also includes the empirical spatial convergence rates p_x for both runs, as calculated using the formula (5.3) with $\zeta = 3$.

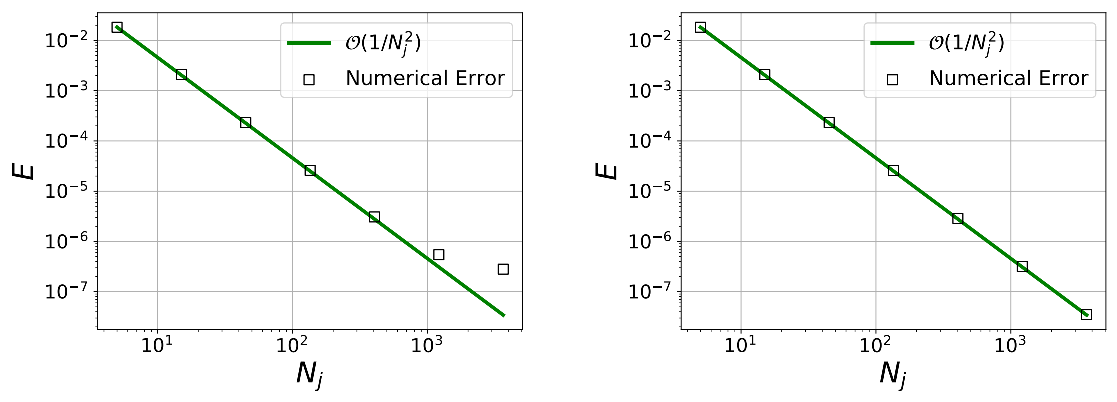
Considering Run 1 first, we see from Figure 5.4a that the error E reduces at the expected rate for small N_j, but levels off for large N_j. This effect is also clear in Table 5.6, where the empirical values of p_x are quite accurate for the first refinements, but drops significantly below its target value as N_j is increased further. This is a clear indication of insufficient temporal resolution; if the number of time levels N_t is too small, the error caused by the temporal discretization will be the dominating term in the total discretization error. When this occurs, further refining the spatial discretization has increasingly little impact on the total discretization error. Since the errors we are calculating are total discretization errors, we then observe convergence rates that are too low.

To test the hypothesis of Run 1 having insufficient temporal resolution, the number of time levels was increased by a factor 10 in Run 2. Figure 5.4b and Table 5.5 shows that the error E now consistently reduces at the expected rate also for large N_j. The largest deviation of p_x from its target value of 2 occurs when N_j is increased from $N_j = 5$ to $N_j = 15$. As discussed in Section 5.2.2, this is not of great concern as results can typically be somewhat unreliable for the smallest values of N_j. The fact that the same deviation can be seen in Run 1 strengthens the hypothesis that the deviation is caused by random effects related to the spatial discretization.

The empirical convergence rates observed in Run 2 yield increased confidence in the correctness of our FVM implementation. Another favorable observation can be made by comparing the convergence rates of Run 2 with those of Run 1. From Table 5.5, it is clear that increasing the number of time levels has had a positive impact on the accuracy of the results for *all* N_j – not just for large N_j. The empirical value of p_x for any given N_j is consistently closer to its target value in Run 2 than in Run 1. This is a clear indication that increasing N_t has indeed had the expected effect of reducing the contribution of the temporal discretization to the total discretization error. Thus, all results presented in this grid refinement study are consistent with a correct implementation.

Table 5.6.: Grid Refinement Study #3 (spatial refinement, unsteady): normalized L_2 errors E and corresponding empirical convergence rates p_x for the computed FVM-approximation of the steady-state temperature profile. Results for Run 1 in columns 2 and 3 and results for Run 2 in columns 4 and 5, counting from the left.

N_j	$E(N_j)$, Run 1	p_x, Run 1	$E(N_j)$, Run 2	p_x, Run 1
5	1.8281e-2	-	1.8281e-2	-
15	2.0839e-3	1.9767	2.0837e-3	1.9768
45	2.3238e-4	1.9967	2.3217e-4	1.9975
135	2.6014e-5	1.9932	2.5804e-5	1.9997
405	3.0790e-6	1.9425	2.8669e-6	2.0001
1215	5.4541e-7	1.5755	3.1819e-7	2.0010
3645	2.8131e-7	0.6027	3.4997e-8	2.0093



(a) Run 1

(b) Run 2

Figure 5.4.: Grid Refinement Study #3 (spatial refinement, unsteady): normalized L_2 error E of the computed FVM-approximation of the temperature profile at time $t = 5\,\text{s}$.

5.4. Grid Refinement Study #4

In this fourth and last grid refinement study, we investigate the temporal convergence rate of our implementation. For this, we solve the same physical problem as in grid refinement study #3 using our Implicit Euler FVM implementation, but perform temporal grid refinement instead of the spatial grid refinement performed earlier.

5.4.1. Configuration and Exact Solution

Table 5.7.: Grid Refinement Study #4 (spatial refinement, unsteady): Configuration of discretization parameters, physical parameters, BCs and IC.

Parameter	Setting
Initial number of time levels	$N_t = 10$
Refinement factor	$\zeta = 2$
End time of simulation	$t_{end} = 5\,\mathrm{s}$
Number of grid cells	$N_j = 3645$
Domain boundaries	$x_a = 0\,\mathrm{m},\ x_b = 1\,\mathrm{m}$
Conductivity	$k = 2500\,\mathrm{W/Km}$
Specific heat capacity	$c_V = 2000\,\mathrm{J/kgK}$
Density	$\rho = 1300\,\mathrm{kg/m^3}$
Cross-sectional area	$A = 1\,\mathrm{m^2}$
Heat generation rate	$\hat{q} = 0\,\mathrm{W/m^3}$
Boundary conditions	$T(x = x_a, t) = 250\,\mathrm{K},\ T(x = x_b, t) = 250\,\mathrm{K}$
Initial condition	$T(x, t = 0) = 250 + 50\sin(2\pi x)\,\mathrm{K}\ \forall x \in (x_a, x_b)$

Table 5.7 lists all parameter choices used in this experiment. Since we are solving the same physical problem as in Grid Refinement Study #3, all choices of the physical parameters, IC and BCs are the same as in Table 5.5. Furthermore, Equation (5.7) is the exact solution for this grid refinement study as well.

5.4.2. Results & Discussion

Figure 5.5 shows the normalized error E of the numerical solution $\boldsymbol{T}_{\text{num}}$ with respect to the exact solution T_{exact} as a function of the number of time levels N_t. The numerical solution was calculated using the Implicit Euler FVM, and the error E was calculated using Equation (4.3). Table 5.8 shows the value of E for each value of N_t. The table also includes the empirical temporal convergence rates p_t, as calculated using the formula (5.3) with $\zeta = 2$. Since the Implicit Euler FVM is first order accurate in time, we expect the empirical temporal convergence rate p_t to be close to 1. Table 5.8 and Figure 5.5 show that this is indeed what we find. All observed values of p_t are accurate to within 1%, which is a reasonable accuracy considering the low number of time levels used.

Table 5.8.: Grid Refinement Study #4 (temporal refinement, unsteady): normalized L_2 errors E and corresponding empirical convergence rates p_t for the computed FVM-approximation of the temperature profile at time $t = 5\,\mathrm{s}$.

N_t	$E(N_t)$	p_t
10	2.0684e-4	-
20	1.0401e-4	0.9917
40	5.2148e-5	0.9961
80	2.6102e-5	0.9985
160	1.3050e-5	1.0001
320	6.5170e-6	1.0018
640	3.2487e-6	1.0044

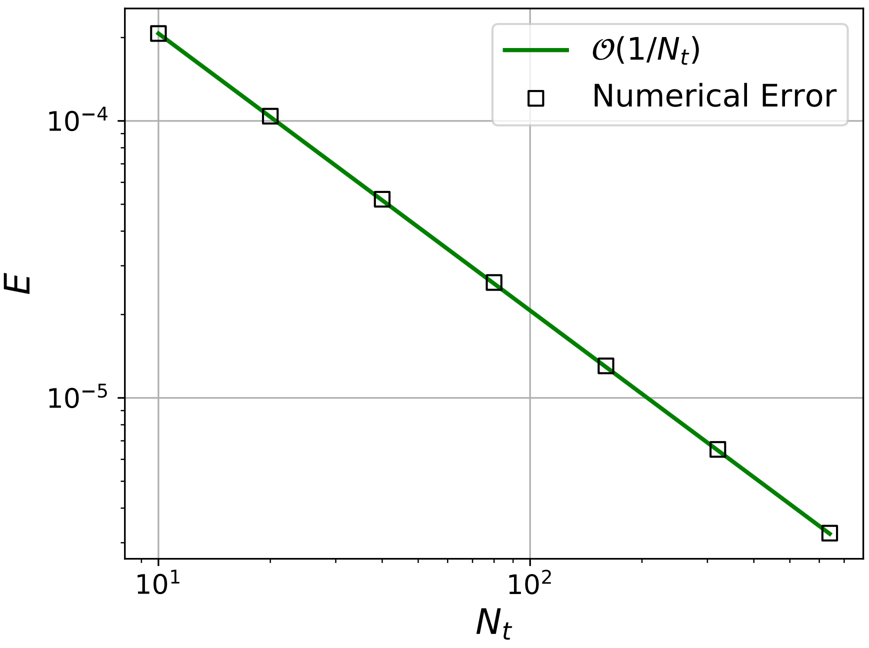


Figure 5.5.: Grid Refinement Study #4 (temporal refinement, unsteady): normalized L_2 error E of the computed FVM-approximation of the temperature profile at time $t = 5\,\mathrm{s}$.

5.5. Summary of Grid Refinement Studies

In this chapter, we have presented four grid refinement studies of our Implicit Euler FVM implementation. With the exception of Run 1 in Grid Refinement Study #3, the observed empirical convergence rates compare well with the theoretical values of $p_x = 2$ and $p_t = 1$ in all studies. Furthermore, the differences in the results of Run 1 and Run 2 in Grid Refinement Study #3 are as expected when the number of time levels is increased in a spatial grid refinement study. In conclusion, all results presented in this chapter are consistent with our numerical scheme being implemented correctly. We thereby consider our implementation verified, and move on to the chapter that is most central in answering our research questions – the chapter concerning our machine learning experiments.

6. Machine Learning Experiments

In this chapter, we compare the abilities of the end-to-end learning approach (Section 4.2) and the hybrid modelling approach (Section 4.3) to correct numerical temperature profiles. For this comparison, we conduct a total of seven machine learning experiments designed to test the accuracy of the corrected temperature profiles in different scenarios. In the first three experiments, described in Sections 6.2 through 6.4, we study steady-state problem. For these problems, we aim to train neural networks such that their learning generalizes to different boundary conditions. On the other hand, the latter four experiments, described in Sections 6.5 through 6.8, concern unsteady problems. We then train each neural network on data corresponding to a single choice of BCs and IC, and aim for networks whose learning generalizes well in time. Details of the experimental method are given in Section 6.1, while a summary of the experiments is provided in Section 6.9.

6.1. Experimental Method

To investigate the abilities of end-to-end learning and hybrid modelling to correct numerical temperature profiles, we need a numerical simulation scheme. For this, we use the Implicit Euler finite-volume method (FVM) implementation that was verified in Chapter 5. In all machine learning experiments described in this chapter, the Implicit Euler FVM is used to solve one-dimensional heat conduction problems for the same physical system. The relevant physical properties of this system are stated in Table 6.1. Note in particular that the conductivity of this system varies as a function of x, and is given by one of two functions, depending on the experiment. In most experiments, we let the

Table 6.1.: Machine Learning Experiments: The physical parameters of the system studied in all machine learning experiments. Note that all parameters are constant throughout the domain, except for the conductivity k.

Parameter	Setting
Domain boundaries	$x_a = 0\,\mathrm{m},\ x_b = 1\,\mathrm{m}$
Conductivity	k defined in Equation (6.1) or (6.2)
Specific heat capacity	$c_V = 200\,\mathrm{J/kgK}$
Density	$\rho = 200\,\mathrm{kg/m^3}$
Cross-sectional area	$A = 1\,\mathrm{m^2}$
Heat generation rate	$\hat{q} = 0\,\mathrm{W/m^3}$

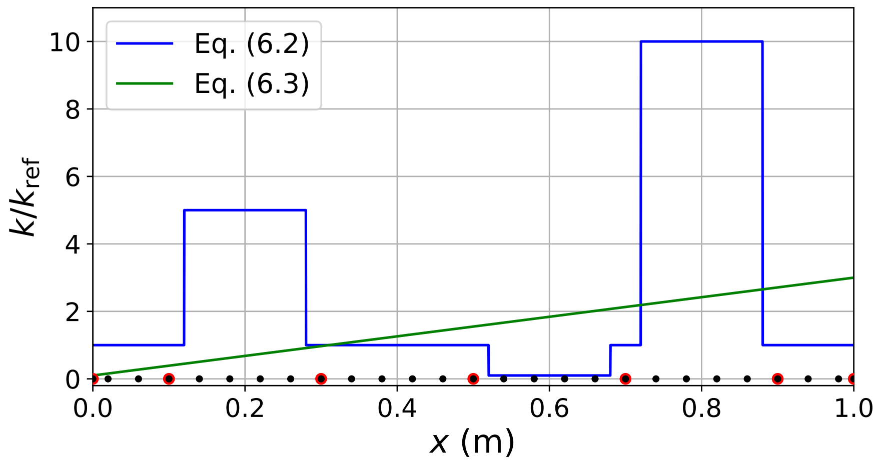


Figure 6.1.: Experimental Method: Conductivity profiles defined by Equation (6.1) and Equation (6.2). For reference, the grid nodes of spatial discretizations with $N_j = 5$ grid cells (red dots) and $N_j = 25$ grid cells (black dots) are also included.

conductivity be given by

$$k = \begin{cases} 5\,k_{\text{ref}} & x \in [0.12, 0.28], \\ 0.1\,k_{\text{ref}} & x \in [0.52, 0.68], \\ 10k_{\text{ref}} & x \in [0.72, 0.88], \\ k_{\text{ref}} & \text{otherwise}, \end{cases} \tag{6.1}$$

where $k_{\text{ref}} = 2500\,\text{W/Km}$ is a reference conductivity. However, in Steady-State Experiment #3 and Unsteady Experiment #4, we use a different conductivity profile instead. In these two experiments,

$$k = (0.1 + 2.9x)\,k_{\text{ref}}, \tag{6.2}$$

where $k_{\text{ref}} = 2500\,\text{W/Km}$ still. The conductivity profiles defined by Equations (6.1) and (6.2) are illustrated in Figure 6.1. This figure also illustrates the spatial discretizations used in the FVM simulations. For the experiments on steady-state problems, we use $N_j = 5$ grid cells, while $N_j = 25$ grid cells are used for the experiments on unsteady problems. The choice of boundary conditions and initial conditions vary from one experiment to another, and are therefore given in the description of each experiment.

Both end-to-end learning and hybrid modelling utilize neural networks. For all experiments covered in this chapter, we use the neural network architecture illustrated in Figure 6.2 for both approaches. This architecture was chosen for its simplicity, and we make no claim that it is an optimal choice. Other possible architectures are discussed in Section 7.3. The chosen architecture is a fully connected neural network with two hidden layers of 100 neurons each. The network's input layer has $N_j + 2$ neurons, where neurons 1 and $N_j + 1$ take the boundary temperatures T_a and T_b as input, while the

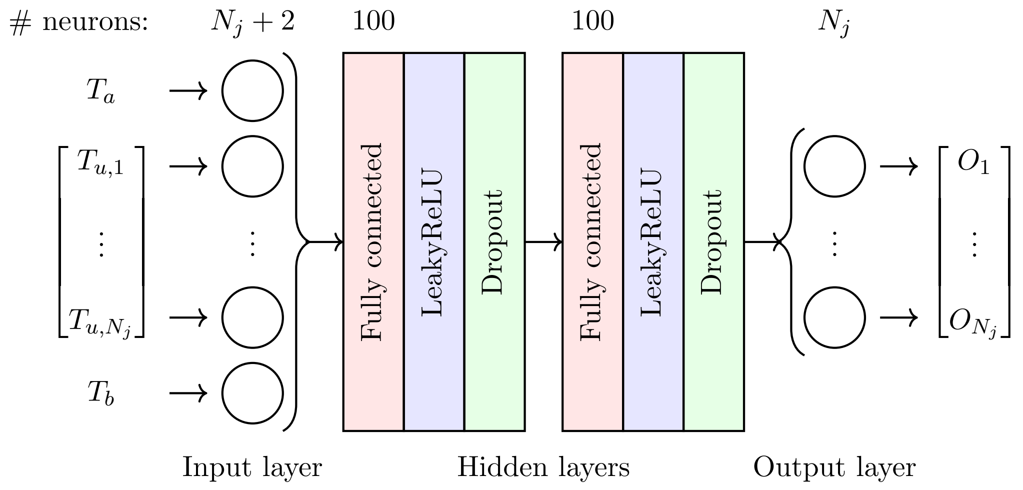


Figure 6.2.: Experimental Method: The neural network architecture used in this work. The network is a fully connected neural network consisting of an input layer, two fully connected hidden layers and an output layer. The input layer receives the boundary temperatures T_a and T_b, and the uncorrected temperature profile $\boldsymbol{T}_\mathrm{c}$, while the output layer outputs a vector $\boldsymbol{O}$ of N_j components. For end-to-end learning, $\boldsymbol{O} = \boldsymbol{T}_\mathrm{c}$, while $\boldsymbol{O} = \hat{\boldsymbol{\sigma}}$ for hybrid modelling. The number written above each layer indicates the number of neurons in that layer. A LeakyReLU activation function with negative slope 0.2 and dropout with probability p_d is applied to both hidden layers. No activation function is applied to the input layer and output layer. Note that the exact values of N_j and p_d are the only aspects of the network architecture that changes between experiments.

remaining neurons each take one the components of the uncorrected temperature profile $\boldsymbol{T}_\mathrm{u}$ as their input. Moreover, the output layer consist of N_j neurons – one neuron for each component of the desired output vector ($\boldsymbol{T}_\mathrm{c}$ for end-to-end learning; $\hat{\boldsymbol{\sigma}}$ for hybrid modelling). LeakyReLU activation and dropout is applied to both hidden layers. No activation function is used for the output layer.

Within any given experiment, the same parameter values are used for the neural networks of both end-to-end learning and hybrid modelling. These parameter values are listed in Table 6.2. Using the same parameter values for both approaches leaves a possibility that the chosen values might favour one approach over the other, which is undesirable. However, it *does* at least ensure a valid performance comparison for the particular parameter configuration that was chosen. An alternative experimental method would be to perform separate tuning processes for each approach and for each experiment. While this would likely improve the performance for both approaches, there is no guarantee we would find the optimal configuration for either approach. Thus, it could still be the case that the chosen configuration for one approach would be better

Table 6.2.: Experimental Method: An overview of the parameters governing the neural network training routine of each machine learning experiment.

Parameter	Steady-State Exps. #1, #2, #3	Unsteady Exps. #1, #2, #4	Unsteady Exp. #3
Optimizer	Adam	Adam	Adam
Learning rate	10^{-5}	10^{-4}	10^{-4}
Learning rate scheduling	No	No	No
Training iterations	20 000*	10 000	20 000
Mini-batch size	16	32	64
Dropout probability	0.0	0.1	0.1
LeakyReLU parameter	0.2	0.2	0.2
Data augmentation	No	No	Yes

* 7500 in Steady-State Experiment #3

than the configuration used for the other approach. This holds even if a systematic tuning method is used. For this reason, extensive parameter tuning, i.e. tuning of is not treated in this work.

We use supervised learning for training the neural networks used for both end-to-end learning and hybrid modelling. As discussed in Chapter 2, we then have to define datasets containing input data and corresponding target output data. For both approaches, the neural network's input will be an uncorrected temperature profile $\boldsymbol{T}_{\text{u}}$, which includes the boundary temperatures T_a and T_b. For end-to-end learning, the network's output vector is a corrected temperature profile $\boldsymbol{T}_{\text{c}}$, while for hybrid modelling, it is a correction source term vector $\hat{\boldsymbol{\sigma}}$. The output vectors do not include components for the boundary nodes x_a and x_b; since the temperatures there are fixed and known, it is meaningless to have the neural network attempt to correct them.

To obtain datasets like those described above for our experiments on *unsteady problems*, we run two parallel simulations of differing quality. For the low quality simulation, we assume that the conductivity k in the physical system is constant and given by $k = k_{\text{ref}}$. No such erroneous assumption is made for the high quality simulation, and this ensures that the low quality simulation contains some error which the high quality simulation does not. Moreover, we also use a finer temporal discretization for the high quality simulation than for the low quality one. This further increases the difference in accuracy between the two simulations. We run the simulations up to a total time $t_{\text{end}} = 3.1\,\text{s}$, using $N_t = 3.1 \cdot 10^3$ and $N_t = 3.1 \cdot 10^4$ time levels for the low quality and the high quality simulation respectively. For each coarse resolution time level $t^n = 0.001n$, $n = 1, 2, \ldots, 3100$, we define one data example where the input $\boldsymbol{T}_{\text{u}}$ is given by the low quality simulation and the reference profile $\boldsymbol{T}_{\text{ref}}$ is given by the high quality simulation. In the case of end-to-end learning, $\boldsymbol{T}_{\text{ref}}$ is used directly as the target output. For hybrid modelling, $\boldsymbol{T}_{\text{ref}}$ is instead inserted into Equation (4.12), along with the reference profile of the previous time level, to obtain $\hat{\boldsymbol{\sigma}}_{\text{ref}}$, which is then used as target output. The data examples corresponding to the first 2000 time levels are included in the training set, and the next 100 data examples are included in the validation set, while the final 1000 data examples constitute the test set. For completeness, we restate that both simulations use $N_j = 25$ grid cells.

In obtaining datasets for our *steady-state* experiments, we again use the erroneous assumption of $k = k_{\text{ref}}$ when generating the uncorrected temperature profiles $\boldsymbol{T}_{\text{u}}$. However, since each steady-state simulation yields only one temperature profile, it is only possible to define one data example per simulation. To obtain a sufficient number of data examples, we therefore perform simulations using different BCs and ICs. Another difference compared to the data generation process for unsteady problems is that, for steady problems, we do not perform parallel high quality simulations to generate $\boldsymbol{T}_{\text{ref}}$. The reason is that, for the system studied in these experiments, we can insert the parameters of Table 6.1 into Equation (3.36) and obtain exact steady-state solutions instead; one solution for any particular choice of BCs and IC. We therefore take the reference profiles $\boldsymbol{T}_{\text{ref}}$ to be exact steady-state solutions evaluated at the grid nodes $x_j,\ x = 1, \dots, N_j$. For end-to-end learning, $\boldsymbol{T}_{\text{ref}}$ is used as the target output directly, whereas $\hat{\boldsymbol{\sigma}}_{\text{ref}}$, as given by Equation (4.8) with $\boldsymbol{T}_{\text{ref}}$ inserted, is used as target output in the case of hybrid modelling. We restate that $N_j = 5$ grid nodes are used in our steady-state experiments.

Before using the datasets described above for training and evaluating our neural networks, we perform a pre-processing step where all data is re-scaled. Inputs and target outputs are re-scaled separately, using the following transformations

$$\boldsymbol{I} \to 2\frac{\boldsymbol{I} - I_{\min}}{I_{\max} - I_{\min}}, \quad \boldsymbol{O} \to 2\frac{\boldsymbol{O} - O_{\min}}{O_{\max} - O_{\min}}, \tag{6.3}$$

where $\boldsymbol{I}$ is some input vector (i.e. an uncorrected temperature profile) and $\boldsymbol{O}$ is some output vector (i.e. a corrected temperature profile or a correction source term vector). The scalars $I_{\min}$ and $I_{\max}$ are respectively the minimum and maximum value of all components of all input vectors in the *training set*. Similarly, $O_{\min}$ and $O_{\max}$ represent the minimum and maximum target output values found in the training set. The reason for only taking the minimum and maximum across the training set and not across all datasets is to avoid information leakage from the validation set or the test set into the training set.

With the scaling (6.3) applied, all data in the training set will lie in the interval $[-1, 1]$, which is desirable for several reasons. The first reason is that it ensures a fair comparison between the two correction approaches. Temperatures and corrections source terms typically differ in size by several orders of magnitude, and this difference could potentially influence the neural networks' ability to train successfully. The scaling above eliminates this possibility. Another benefit of scaling the data is that neural networks tend to learn faster and more successfully when the data in its training set has a mean close to zero [36].

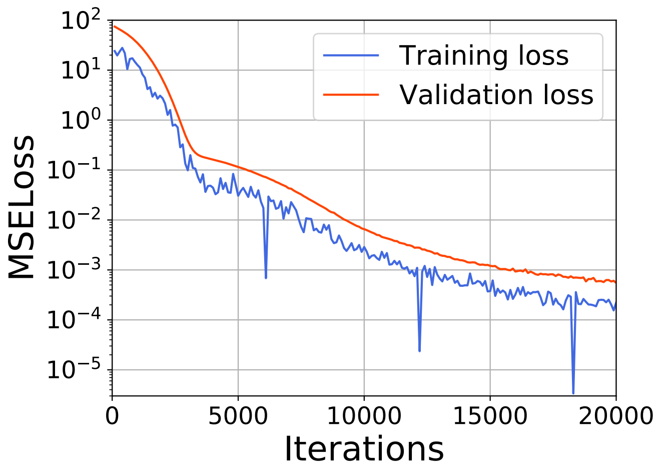
A final comment on the experimental method relates to our choice of optimization algorithm. As stated in Table 6.2, we use the Adam optimization algorithm in all experiments presented in this chapter. The Adam algorithm, which is based on stochastic gradient descent, is an inherently random algorithm (cf. Chapter 2). Consequently, different instances of the same model can perform differently when trained using the Adam algorithm. To account for this, we train eight different instances of the same neural network model for each approach in each experiment. By averaging results across the eight model instances, we aim to reduce the effects of the Adam algorithm's inherent randomness. Furthermore, in an effort to make our results reproducible, we seed the random number generators of both NumPy and PyTorch, and follow other advice[1] given by the developers of PyTorch. Lastly, note that detailed results for each model instance can be found in Appendix F.

6.2. Steady-State Experiment #1 – Interpolation

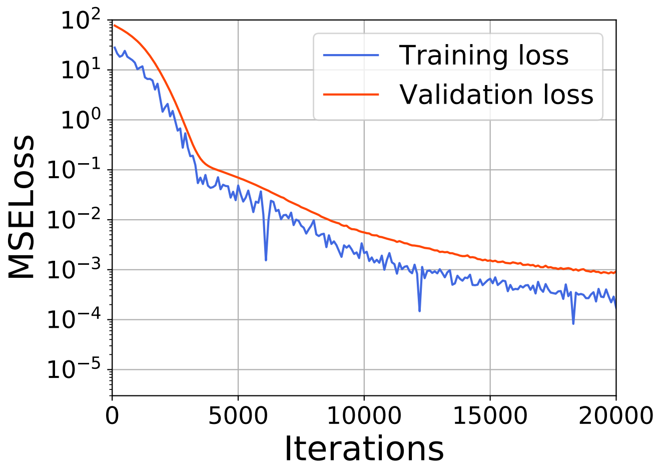
The purpose of this experiment is to investigate the performance of the presented correction approaches on problems where large corrections are desired, and where the three datasets possess similar statistical properties. By "large" corrections, we mean corrections that are of a similar order of magnitude as the temperature variation within any given temperature profile in the datasets. When large corrections are needed, it is possible for the corrections to improve the accuracy of the numerical temperature profiles even if the neural networks do not capture every nuance of the underlying physics. We therefore expect both correction methods to produce corrected temperature profiles $\boldsymbol{T}_{\mathrm{c}}$ that compare favorably with the uncorrected profiles $\boldsymbol{T}_{\mathrm{u}}$ in this experiment. Furthermore, the use of datasets with similar statistical properties means the current problem is comparable to interpolation, which is easier than extrapolation. In light of these considerations, it is clear that our correction approaches are used in a comparatively simple scenario for this experiment. The results from this experiment will therefore serve as a sanity check for our machine learning implementation and configuration. If a correction approach fails in this experiment, it can be sign of a faulty implementation or unreasonable architecture or parameters choices. Poor results for this experiment can also be indicative of flaws in the theoretical foundation of a poorly performing approach.

6.2.1. Configuration

For the training dataset, one data example was generated for each combination of BCs $(T_L, T_R) \in \mathcal{S} \times \mathcal{S}$ where $\mathcal{S} = \{250\,\mathrm{K}, 255\,\mathrm{K}, 260\,\mathrm{K}, \ldots, 395\,\mathrm{K}, 400\,\mathrm{K}\}$. The training dataset then contains a total of 961 training examples. For the validation dataset, we first generated 100 random two-decimal numbers within the interval $[250\,\mathrm{K}, 400\,\mathrm{K}]$. These were then split into 50 pairs, with each pair defining a pair of BCs (T_L, T_R). For each of these pairs, a validation example was generated. The testing dataset was created in the same way as the validation dataset, but with 100 different random numbers. All datasets were normalized in accordance with the discussion in Section 6.1.

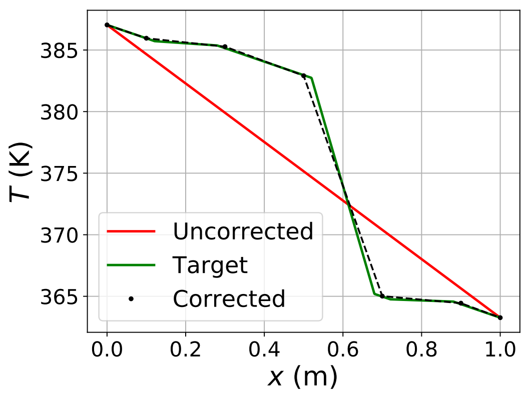


(a) End-to-end learning

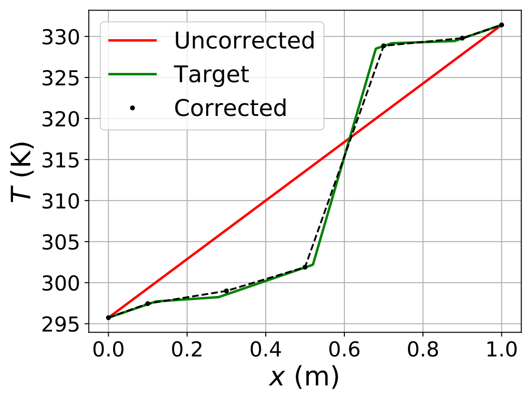


(b) Hybrid modelling

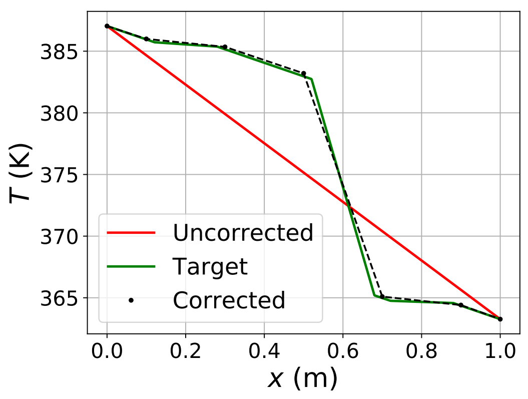
Figure 6.3.: Steady-State Experiment #1: Mean square error loss on training and validation set for two representative model instances – one instance for each correction approach.



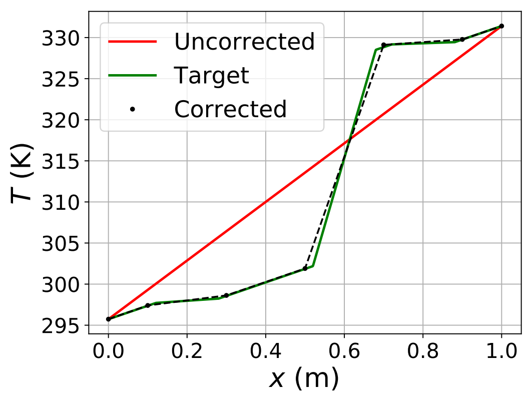
(a) End-to-end learning, example 1



(b) End-to-end learning, example 2



(c) Hybrid modelling, example 1



(d) Hybrid modelling, example 2

Figure 6.4.: Steady-State Experiment #1: Corrected temperature profiles for two randomly chosen data examples in the test set. The corrections were made by neural networks trained using the end-to-end approach (top row) and hybrid modelling approach (bottom row). The corresponding uncorrected profiles and reference profiles (labelled “Target”) are included for comparison.

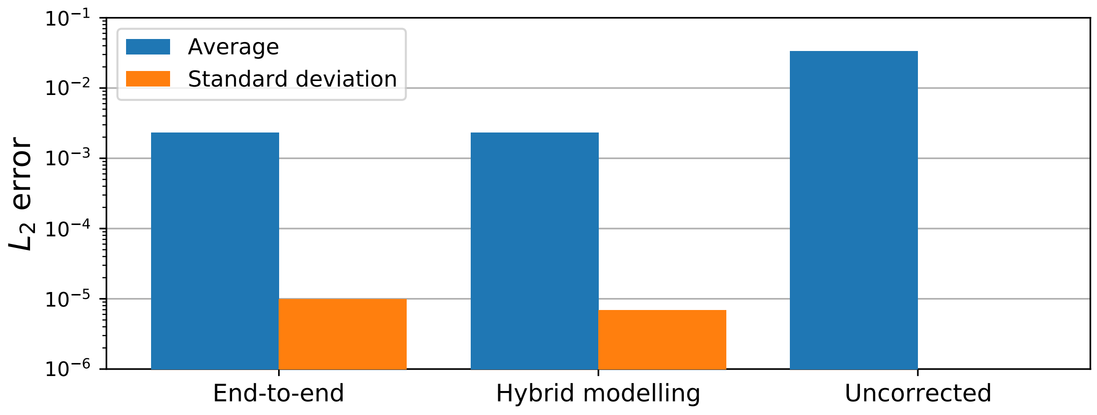


Figure 6.5.: Steady-State Experiment #1: Qualitative illustration of L_2 error statistics for temperature profiles generated using end-to-end learning, hybrid modelling and an uncorrected simulation. Note the logarithmic scaling of the vertical axis.

Table 6.3.: Steady-State Experiment #1: L_2 error statistics of predicted temperature profiles for end-to-end learning, hybrid modelling and an uncorrected simulation.

	End-to-end	Hybrid modelling	Uncorrected
Average	2.3209e-3	2.3260e-3	3.3765e-2
Std. dev.	9.9620e-6	6.9092e-6	0.0

6.2.2. Results

Statistics of the normalized L_2 errors $\|\boldsymbol{T}_\mathrm{c} - \boldsymbol{T}_\mathrm{ref}\|_2 / \|\boldsymbol{T}_\mathrm{ref}\|_2$ of corrected temperature profiles $\boldsymbol{T}_\mathrm{c}$ for end-to-end learning and hybrid modelling are given in Table 6.3, along with statistics of the normalized error $\|\boldsymbol{T}_\mathrm{u} - \boldsymbol{T}_\mathrm{ref}\|_2 / \|\boldsymbol{T}_\mathrm{ref}\|_2$ of uncorrected temperature profiles $\boldsymbol{T}_\mathrm{u}$. For each correction approach, the statistics (averages and standard deviations) are calculated based on the results of eight neural network model instances. Note that the uncorrected simulation is deterministic and therefore yields an empirical standard deviation of zero. Figure 6.5 provides a qualitative illustration of the data in Table 6.3. All statistics presented in the figure and the table are averaged across all data examples in the test set.

Figure 6.3a shows the training and validation losses of a representative end-to-end model instance. These are plotted as functions of the number of training iterations completed. Similarly, the training and validation losses of a representative hybrid modelling instance are shown in Figure 6.3b. Figure 6.4 shows corrected temperature profiles for two data examples from the test set, as computed by the representative end-to-end model instance (top row) and the representative hybrid modelling instance (bottom row).

[1]`https://pytorch.org/docs/stable/notes/randomness.html`

6.2.3. Discussion

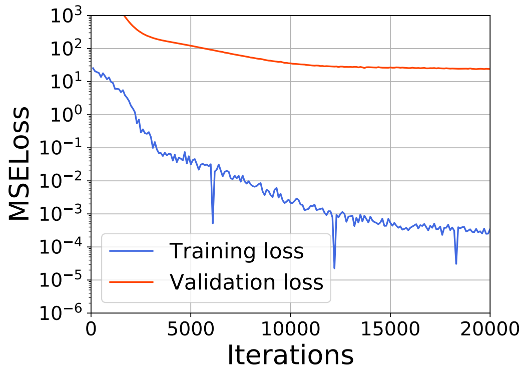
From Figure 6.3, we see that the number of training iterations was set at a level that is reasonable for both approaches. The general trend is that both the training losses and the validation losses are decreasing continuously throughout training, and level off towards the end of training. The graphs do not show any clear indications of overfitting, as the validation loss does not level off sooner or to a greater extent than the training loss. On the contrary, all losses seem to be decreasing slowly even at the end of training, so it is possible that training for an even larger number of iterations could have proven beneficial for the models' convergence. However, this was not investigated further because precise parameter tuning is beyond the scope of this work.

Both correction approaches yield reasonable corrected temperature profiles, as can be seen from Figure 6.4. Some deviations from the exact temperature profiles are clearly noticeable, but the corrected profiles of both approaches follow the general trends of the reference profiles. The same cannot be said for the uncorrected solutions, which turn out to be straight lines due to the assumption of constant conductivity for the numerical simulations. Thus, both approaches are capable of providing successful corrections for the two data examples illustrated. From the data presented in Table 6.3 and Figure 6.5, we see that this holds generally for the other data examples in the test set as well. The average value of the normalized error $\|\boldsymbol{T}_\text{c} - \boldsymbol{T}_\text{ref}\|_2/\|\boldsymbol{T}_\text{ref}\|_2$ is roughly 15 times smaller than the average of $\|\boldsymbol{T}_\text{u} - \boldsymbol{T}_\text{ref}\|_2/\|\boldsymbol{T}_\text{ref}\|_2$, regardless of which approach was used to generate $\boldsymbol{T}_\text{c}$. We take this as an indication that the correction approaches have been implemented correctly, and that the approaches themselves are not conceptually flawed.

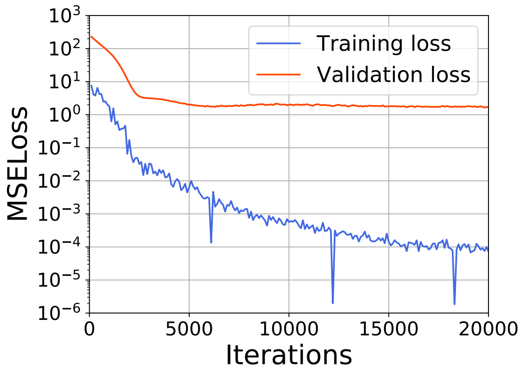
As for comparing the performance of hybrid modelling and end-to-end learning, the results of this experiment do not provide sufficient foundation for claiming one approach to be better than the other. The corrected temperature profiles in Figure 6.4 cannot be distinguished based on qualitative differences, and the differences in the L_2 error statistics in Table 6.3 and Figure 6.5 are not of practical significance.

6.3. Steady-State Experiment #2 – Extrapolation

In the second steady-state experiment, we continue to study the performance of our correction approaches on a problem where large corrections are desired. The difference with respect to the first steady-state experiment is that, this time, the statistical properties of the training set are different than those of the validation and test set. Thus, this experiment is a tougher test of the generalizability of the neural networks' learning, essentially requiring the networks to extrapolate. If one approach outperforms the other in this experiment, one might interpret this as an indication of the best-performing method being more robust than the other, i.e. that it is better suited for handling outlier data inputs. This is important for stability in real-world applications.



(a) End-to-end learning



(b) Hybrid modelling

Figure 6.6.: Steady-State Experiment #2: Mean square error loss on training and validation set for two representative model instances – one instance for each correction approach.

6.3.1. Configuration

The training set for this experiment was generated in exactly the same way as the training set for Steady-State Experiment #1; see Section 6.2.1 for the details. The validation set and the test set were also generated similarly, but for this experiment, the random numbers were drawn from a uniform distribution on the interval $[500\,\mathrm{K}, 1000\,\mathrm{K}]$. Thus, all temperatures in the training set lie within the interval $[250\,\mathrm{K}, 400\,\mathrm{K}]$, while all temperatures in the remaining two sets lie within the interval $[500\,\mathrm{K}, 1000\,\mathrm{K}]$. Again, all datasets were normalized before use, as described in Section 6.1. Furthermore, no machine learning parameters were changed from the previous experiment.

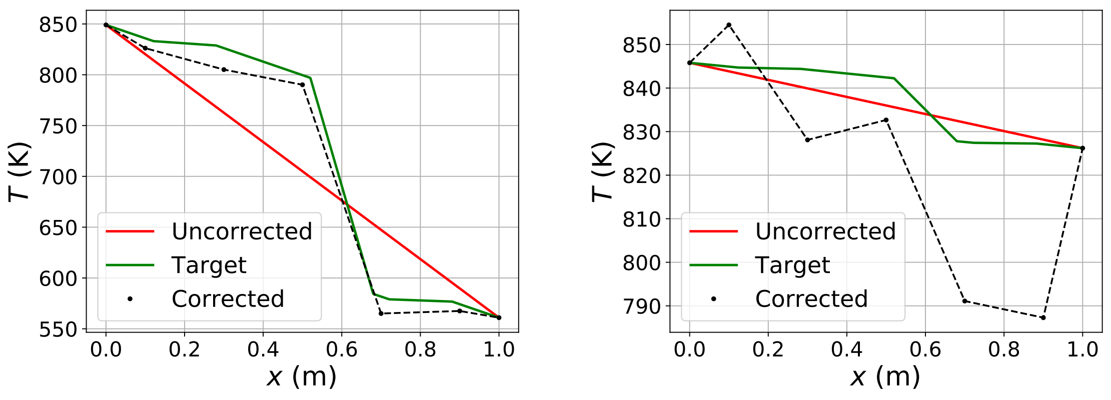
6.3.2. Results

Statistics of the normalized L_2 errors $\|\boldsymbol{T}_\mathrm{c} - \boldsymbol{T}_\mathrm{ref}\|_2 / \|\boldsymbol{T}_\mathrm{ref}\|_2$ and $\|\boldsymbol{T}_\mathrm{u} - \boldsymbol{T}_\mathrm{ref}\|_2 / \|\boldsymbol{T}_\mathrm{ref}\|_2$ are given in Table 6.4. These statistics are illustrated qualitatively in Figure 6.8. As in the previous experiment, the statistics are calculated based on the results of eight neural network model instances, and they are averaged across all data examples in the test set.

Training and validation losses of a representative end-to-end instance are shown in Figure 6.6a. The same information is shown for a representative hybrid modelling instance in Figure 6.6b. Figure 6.7 displays corrected temperature profiles for two data examples from the test set, as computed by the representative end-to-end model instance (top row) and the representative hybrid modelling instance (bottom row).

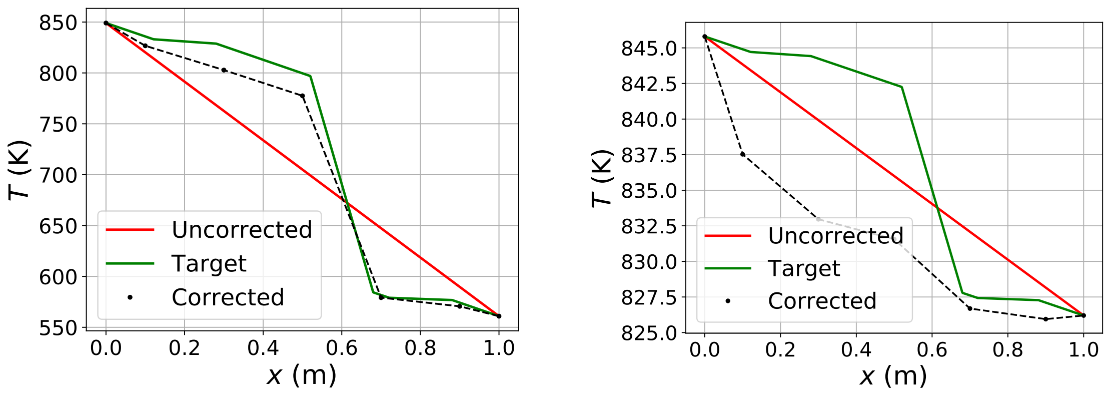
6.3.3. Discussion

From the L_2 error statistics in Figure 6.8, one can observe significant differences in accuracy for the different correction approaches and the uncorrected simulation. The hybrid



(a) End-to-end learning, example 1

(b) End-to-end learning, example 2



(c) Hybrid modelling, example 1

(d) Hybrid modelling, example 2

Figure 6.7.: Steady-State Experiment #2: Corrected temperature profiles for two randomly chosen data examples in the test set. The corrections were made by neural networks trained using the end-to-end approach (top row) and hybrid modelling approach (bottom row). The corresponding uncorrected profiles and reference profiles (labelled “Target”) are included for comparison.

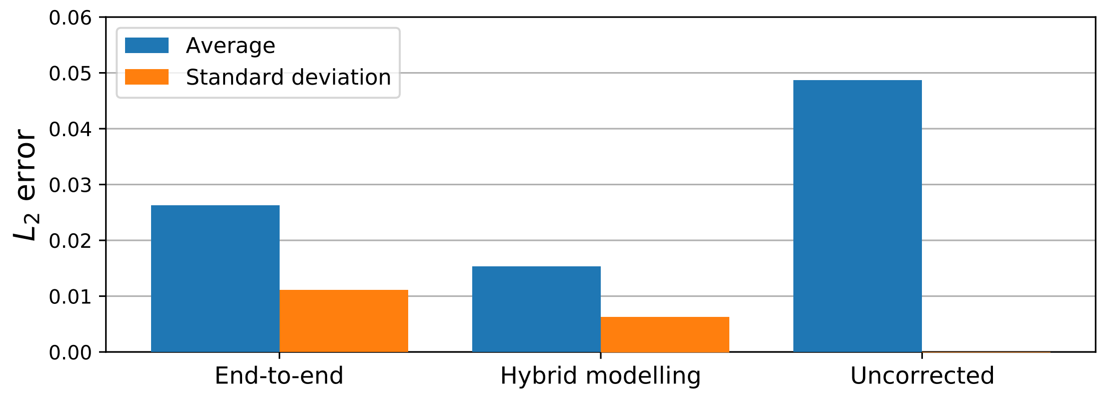


Figure 6.8.: Steady-State Experiment #2: Qualitative illustration of L_2 error statistics for temperature profiles generated using end-to-end learning, hybrid modelling and an uncorrected simulation.

Table 6.4.: Steady-State Experiment #2: L_2 error statistics of predicted temperature profiles for end-to-end learning, hybrid modelling and an uncorrected simulation.

	End-to-end	Hybrid modelling	Uncorrected
Average	2.6201e-2	1.5323e-2	4.8695e-2
Std. dev.	1.1079e-2	6.2838e-3	0.0

modelling corrections are significantly more accurate and less variable than those made using end-to-end learning. Meanwhile, the average error of the end-to-end corrected profiles are in turn significantly smaller than that of the uncorrected profiles. These results suggest that both correction methods are able to provide meaningful corrections to numerical simulations, even when presented with data that differs significantly from data seen during training. Furthermore, they also suggest that the hybrid modelling approach generalizes better than the end-to-end approach.

Looking at the corrected temperature profiles in Figure 6.7, the apparent success of the correction approaches becomes less obvious. We see that for data example 1 (left column), both correction approaches yield corrected profiles which are significantly more accurate than the uncorrected profiles. However, for data example 2, the corrected temperature profiles are more inaccurate than the uncorrected profile. In addition, both corrected profiles are clearly unphysical, since they are not monotonously increasing or decreasing, as is required for a steady-state temperature profile when no source term is present. It is therefore clear that neither correction approach is able to consistently provide trustworthy corrections for the current physical system, in spite of the low average errors observed.

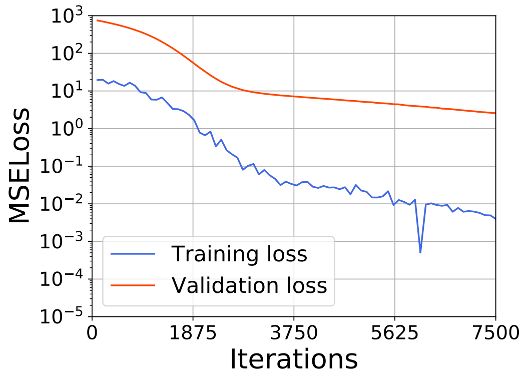
For hybrid modelling, the comparatively poor performance on example 2 can be explained as a consequence of the most striking difference between the two data examples

– the difference between the boundary temperatures is much smaller for example 2 than for example 1. In Appendix E, it is shown that the reference correction source term is proportional to the difference between the boundary temperatures. Thus, the neural network's target output is much larger for example 1 than for example 2. In accordance with the discussion of the MSE loss in Section 2.4, this means that, when minimizing the loss for the training set as a whole, it is more important for the neural network to increase its relative accuracy on example 1 than on example 2.

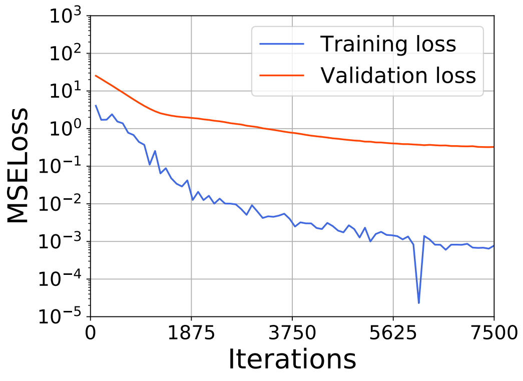
For end-to-end learning, the argument above cannot be used to explain the poor performance on example 2. Recall that end-to-end learning requires the neural network to output a temperature profile directly. Since $\|\boldsymbol{T}_{\text{ref}}\|_2$ is larger for example 2 than for example 1, the former example actually contributes more to the total training loss than the latter. (The normalization makes this statement less obvious than it might initially seem, but it is true regardless.) According to the argument above, end-to-end learning should then perform best on example 2, which does not appear to be the case. One potential explanation could be that the reference solution, which for this experiment is given by Equation (3.36), depends both on the temperature difference $T_b - T_a$ *and* the magnitude of T_a (or, equivalently, T_b). Since the temperatures of example 2 are further away from the temperature range covered by the training set, more extreme extrapolation is required for example 2 than for example 1, and this could explain the comparatively poor performance on example 2. Recall that the target output for hybrid modelling is independent of temperature magnitudes, so it sensible that the extrapolation has a greater impact on the performance of end-to-end learning than on the performance of hybrid modelling.

From a comparison of Figure 6.6 with Figure 6.3, we see that the validation losses are consistently larger in this experiment than in the previous one. However, this cannot be unambiguously interpreted because the MSE losses are not normalized. Since the temperatures in the validation set of this experiment are larger than those in the validation set of the previous experiment, such an increase in the validation loss would be observed even if the *relative* errors of the corrected temperature profiles were the same in both experiments. Furthermore, it is likely that the increased difficulty of the learning task has contributed to the increased validation loss as well. We thereby conclude that this observation need not raise any concerns.

A more interesting observation to be made from Figure 6.6 is that the validation losses seem to level off much sooner than the training losses. This is a serious sign of overfitting which casts some doubt on the validity of the discussion above, since it is likely that that overfitting has had some negative influence on the results obtained in this experiment. While the loss curves of Figure 6.6 suggest more severe overfitting for hybrid modelling, one cannot rule out that the performance of end-to-end learning may have been affected equally or even more severely. However, the mentioned signs of overfitting are not to be found in the loss curves of all model instances trained for this experiment. Despite this, the performance difference between the two approaches is fairly consistent across all instances, and the aforementioned unphysical behaviour is also persistent across all instances. For these reasons, we believe that overfitting alone



(a) End-to-end learning



(b) Hybrid modelling

Figure 6.9.: Steady-State Experiment #3: Mean square error loss on training and validation set for two representative model instances – one instance for each correction approach.

cannot explain the observations noted during the discussion of this experiment.

6.4. Steady-State Experiment #3 – Extrapolation II

In the third and final steady-state experiment, our goal is to investigate if we can replicate the results from Steady-State Experiment #2 for a different steady-state problem. To this end, we study the same physical system as before, but let the conductivity be defined by Equation (6.2) instead of Equation (6.1). We also use different boundary conditions to define the datasets, as discussed below.

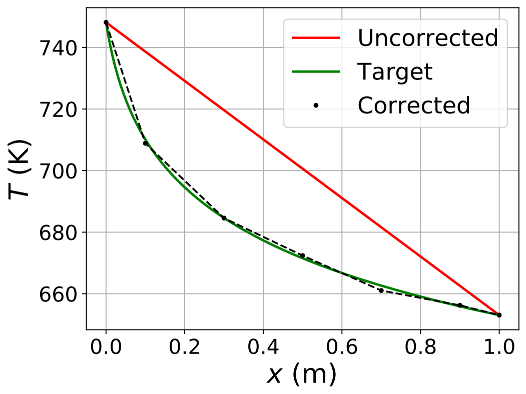
6.4.1. Configuration

The data generation procedure for this experiment is analogous to the one used in the previous two experiments. The only changes relate to the choice of BCs. For the training set, the BCs (T_L, T_R) are now taken as the elements of the Cartesian product $\mathcal{S} \times \mathcal{S}$ with $\mathcal{S} = \{600\,\mathrm{K}, 613.\bar{3}\,\mathrm{K}, 626.\bar{6}\,\mathrm{K}, \dots, 986.\bar{6}\,\mathrm{K}, 1000\,\mathrm{K}\}$. For the validation and testing sets, the randomly chosen BCs were now sampled from a uniform distribution on the interval $[200\,\mathrm{K}, 800\,\mathrm{K}]$.

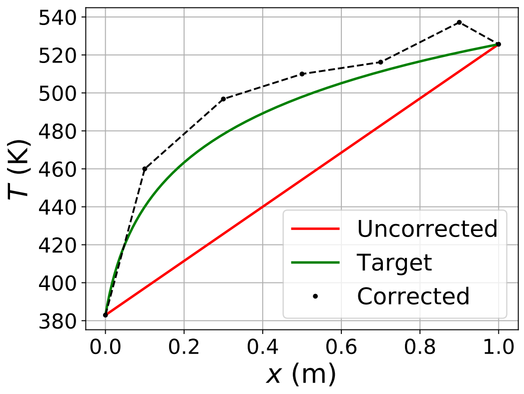
We continue to use the same parameters as in the earlier experiments with one exception; the number of training iterations is lowered from 20 000 to 7500. This change was motivated by the signs of overfitting observed in Steady-State Experiment #2, as well as similar signs observed during preliminary versions of the experiment presented here.

6.4.2. Results

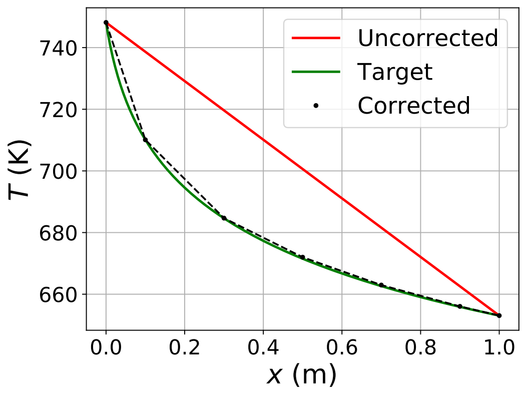
Statistics of the normalized L_2 error $\|\boldsymbol{T}_\mathrm{c} - \boldsymbol{T}_\mathrm{ref}\|_2/\|\boldsymbol{T}_\mathrm{ref}\|_2$ of corrected temperature profiles $\boldsymbol{T}_\mathrm{c}$ for end-to-end learning and hybrid modelling are given in Table 6.5, along with



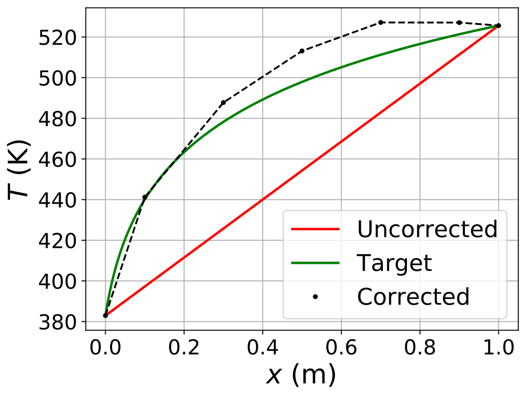
(a) End-to-end learning, example 1



(b) End-to-end learning, example 2



(c) Hybrid modelling, example 1



(d) Hybrid modelling, example 2

Figure 6.10.: Steady-State Experiment #3: Corrected temperature profiles for two randomly chosen data examples in the test set. The corrections were made by neural networks trained using the end-to-end approach (top row) and hybrid modelling approach (bottom row). The corresponding uncorrected profiles and reference profiles (labelled "Target") are included for comparison.

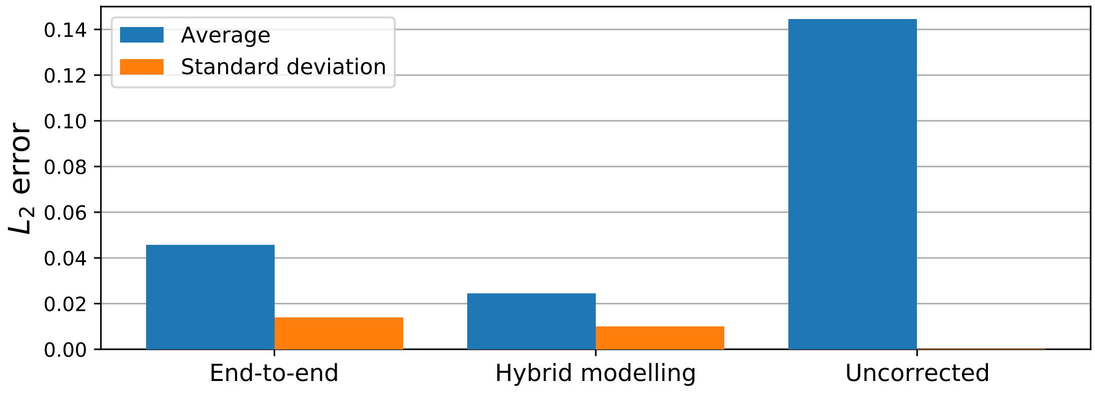


Figure 6.11.: Steady-State Experiment #3: Qualitative illustration of L_2 error statistics for temperature profiles generated using end-to-end learning, hybrid modelling and an uncorrected simulation.

Table 6.5.: Steady-State Experiment #3: L_2 error statistics of predicted temperature profiles for end-to-end learning, hybrid modelling and an uncorrected simulation.

	End-to-end	Hybrid modelling	Uncorrected
Average	4.5790e-2	2.4359e-2	1.4447e-1
Std. dev.	1.3876e-2	9.9153e-3	0.0

statistics of the normalized error $\|\boldsymbol{T}_\mathrm{u} - \boldsymbol{T}_\mathrm{ref}\|_2 / \|\boldsymbol{T}_\mathrm{ref}\|_2$ of uncorrected temperature profiles $\boldsymbol{T}_\mathrm{u}$. As before, these statistics are calculated based on the results of eight neural network model instances, and averaged across all data examples in the test set. The statistics are illustrated qualitatively in Figure 6.8.

The training and validation losses for a representative end-to-end model instance and a representative hybrid modelling instance are given in Figure 6.6a and 6.6b, respectively. Figure 6.7 shows corrected temperature profiles for two data examples from the test set, as computed by the representative end-to-end model instance (top row) and the representative hybrid modelling instance (bottom row).

6.4.3. Discussion

The training and validation losses plotted in Figure 6.9 do not exhibit any signs of overfitting, so the reduction in the number of training iterations was successful in its principal goal. From the losses shown here, one might get the impression that the number of iterations was actually set somewhat *too* low, as all losses (and especially those for end-to-end learning) were still decreasing at the end of the training period. However, from preliminary experiments it was observed that the losses consistently levelled off at roughly 7500 iterations, and a longer training period did not yield significant differences in performance compared to what is reported here.

Regarding the L_2 loss statistics in Table 6.5 and Figure 6.11, we observe much the same behaviour as in Steady-State Experiment #2; the hybrid modelling approach yields an average L_2 error which is roughly half that of the end-to-end approach. In addition, both approaches yield corrected temperature profiles $\boldsymbol{T}_\mathrm{c}$ which are far more accurate than the uncorrected profiles $\boldsymbol{T}_\mathrm{u}$. From Figure 6.10, we see that corrected temperature profiles appear more accurate now than in the previous experiment (cf. Figure 6.7). However, the corrected temperature profiles exhibit some unphysical behaviour in this experiment as well, as the profiles are not monotonously increasing for example 2. (Note that the naming of the data examples is specific to each experiment. The data examples illustrated for this experiment are not related to those illustrated for the previous experiments.) The performance difference between example 1 and example 2 is likely caused by their differing similarity to the training data. The temperatures of example 1 lie completely within the range covered by the training data, while the temperatures of example 2 lie completely outside this range. Thus, example 1 requires interpolation, while example 2 requires extrapolation, which is more prone to errors than interpolation. While the unphysical behaviour is less severe in this experiment than in the previous one, the unphysicality of the corrected solutions still diminishes the trustworthiness of the methods. The doubt concerning their eligibility for use in digital twin applications is therefore strengthened.

6.5. Unsteady Experiment #1 – Local Error

We now move on to unsteady problems, where the required corrections are typically much smaller than we have seen in the steady-state problems treated so far. In this first unsteady experiment, we consider the network's ability to make *individual* corrections. I.e., if $\boldsymbol{T}_\mathrm{c}^n = \boldsymbol{T}_\mathrm{ref}^n$ and $\boldsymbol{T}_\mathrm{u}^n = \boldsymbol{T}_\mathrm{ref}^n$ for some time level t^n, we investigate the accuracy of the corrected solutions $\boldsymbol{T}_\mathrm{c}^{n+1}$ and the uncorrected solutions $\boldsymbol{T}_\mathrm{u}^{n+1}$ with respect to the reference profiles $\boldsymbol{T}_\mathrm{ref}^{n+1}$. Using the jargon of numerical analysis, this is equivalent to investigating the *local* error of the corrected and the uncorrected numerical simulation scheme.

6.5.1. Configuration

When studying unsteady problems, the temperature profiles of any two consecutive time levels can be used to generate a unique data example. To generate a usable amount of data examples, it is therefore sufficient to study the temporal development of a system for a single combination of initial conditions and boundary conditions. In this experiment, we choose $T_a = 250\,\mathrm{K}$, $T_b = 400\,\mathrm{K}$ and $T^0(x) = 325\,\mathrm{K}$, $\forall x \in (x_a, x_b)$. We study the same physical system as in the Steady-State Experiments, and let the system's conductivity be given by Equation (6.1). To generate the datasets, we simulate the temporal development of this system using a low quality simulation and a high quality simulation, as explained in Section 6.1. The low quality simulation, which assumes constant conductivity and uses

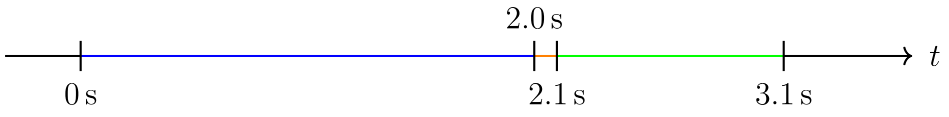


Figure 6.12.: Unsteady Experiment #1: Data from the time ranges coloured in blue, orange and green are used to define the training set, the validation set and the test set, respectively.

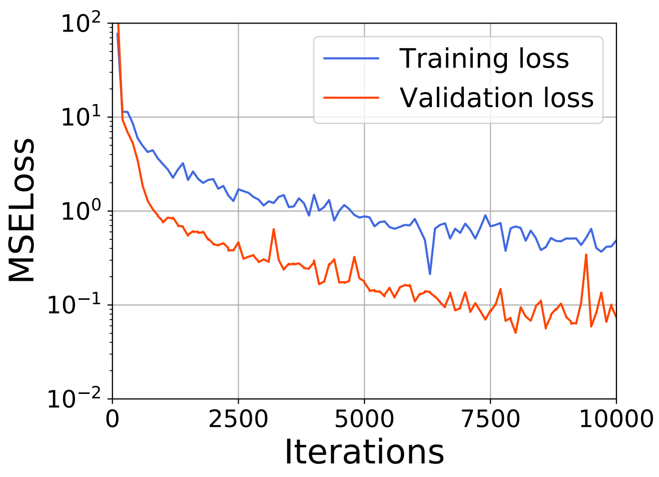
a larger time step, is used to generate the uncorrected profiles $\boldsymbol{T}_u^n$, while the high quality simulation generates $\boldsymbol{T}_{\text{ref}}^n$. Since we here are interested in *local* error corrections only, we use $\boldsymbol{T}_{\text{ref}}^n$ instead of $\boldsymbol{T}_u^n$ to generate $\boldsymbol{T}_u^{n+1}$. This way, we ensure that the uncorrected temperature profiles do not contain any accumulated errors from earlier time levels. The simulations end at the time $t_{\text{end}} = 3.1\,\text{s}$. With a coarse scale time step of $\Delta t = 0.001\,\text{s}$, this yields a total of $t_{\text{end}}/\Delta t = 3100$ data examples. These examples are distributed between the three datasets as illustrated in Figure 6.12; Examples for $t \in [0\,\text{s}, 2.0\,\text{s}]$, $t \in (2.0\,\text{s}, 2.1\,\text{s}]$ and $t \in (2.1\,\text{s}, 3.1\,\text{s}]$ constitute the training set, the validation set and the test, respectively.

During preliminary experiments, clear signs of overfitting were observed for both correction approaches. To alleviate this, the dropout probability p_d was increased to a non-zero value of $p_d = 0.1$, thereby adding some regularization to the neural networks used. It was also observed that continuing with the previously used learning rate of 10^{-5} yielded very slow convergence. The learning rate was therefore increased to 10^{-4}. For this new learning rate, training for 10 000 iterations with a mini-batch size of 32 appear to yield sufficient convergence.

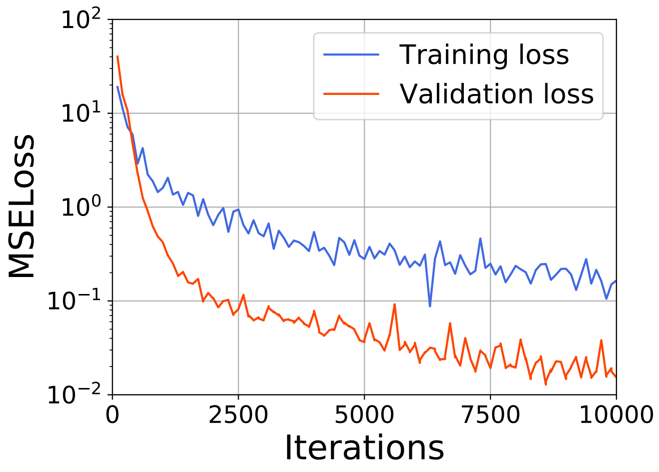
6.5.2. Results

Statistics of the normalized L_2 error $\|\boldsymbol{T}_{\text{c}}^n - \boldsymbol{T}_{\text{ref}}^n\|_2 / \|\boldsymbol{T}_{\text{ref}}^n\|_2$ of corrected temperature profiles $\boldsymbol{T}_{\text{c}}^n$ for end-to-end learning and hybrid modelling are given in Table 6.6, along with statistics of the normalized error $\|\boldsymbol{T}_{\text{u}}^n - \boldsymbol{T}_{\text{ref}}^n\|_2 / \|\boldsymbol{T}_{\text{ref}}^n\|_2$ of uncorrected temperature profiles $\boldsymbol{T}_{\text{u}}^n$. The statistics are presented for two time levels, $n = 2200$ and $n = 3100$. As in earlier experiments, these statistics are calculated based on the results of eight neural network model instances. However, since we now study the error at distinct time levels, no averaging across different test examples is performed. The statistics for time level $n = 3100$ are illustrated qualitatively in Figure 6.15.

The training and validation losses for a representative end-to-end model instance and a representative hybrid modelling instance are given in Figure 6.13a and 6.13b, respectively. Figure 6.14 shows the corrected temperature profiles at the final time level $t = 3.1\,\text{s}$, as computed using the representative end-to-end model instance (top) and the representative hybrid modelling instance (bottom).

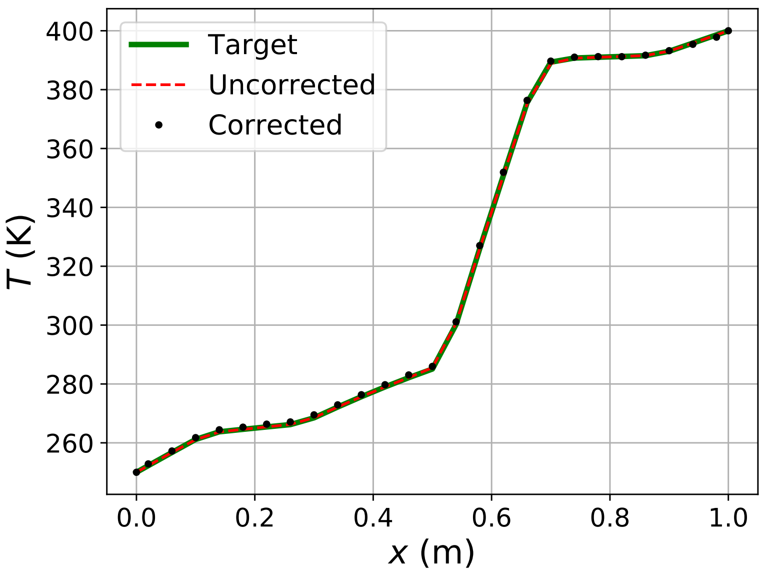


(a) End-to-end learning

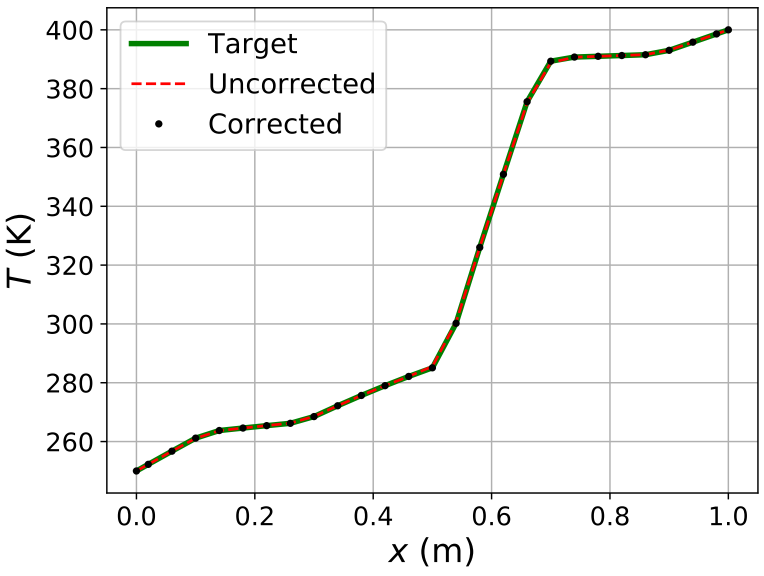


(b) Hybrid modelling

Figure 6.13.: Unsteady Experiment #1: Mean square error loss on training and validation set for two representative model instances – one instance for each correction approach.



(a) End-to-end learning



(b) Hybrid modelling

Figure 6.14.: Unsteady Experiment #1: Corrected temperature profiles at time $t = 3.1$ s. The corrections were made by neural networks trained using the end-to-end approach (top) and hybrid modelling approach (bottom). The corresponding uncorrected profiles and reference profiles (labelled "Target") are included for comparison.

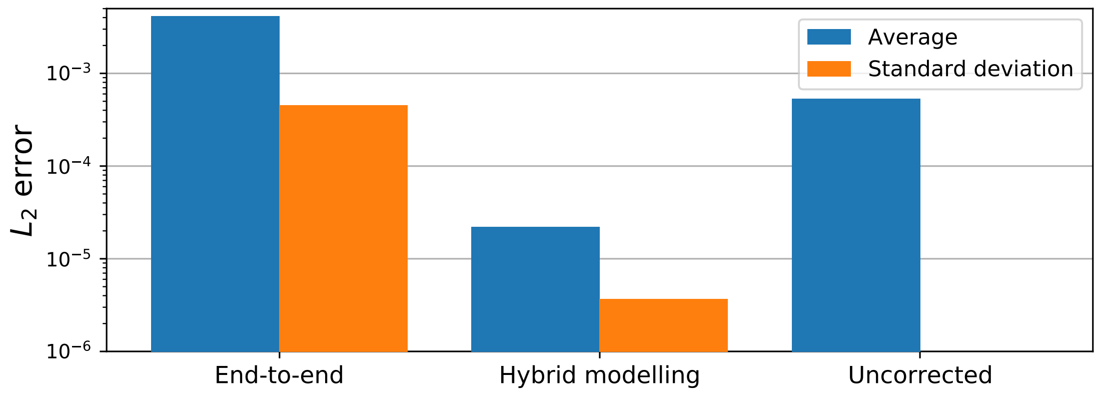


Figure 6.15.: Unsteady experiment #1: Qualitative illustration of L_2 error statistics for unsteady temperature profiles at time $t = 3.1\,\mathrm{s}$ generated using end-to-end learning, hybrid modelling and an uncorrected simulation. Note the logarithmic scaling of the vertical axis.

Table 6.6.: Unsteady Experiment #1: L_2 error statistics of predicted temperature profiles for end-to-end learning, hybrid modelling and an uncorrected simulation.

	End-to-end	Hybrid modelling	Uncorrected
Average	4.1664e-3	2.2069e-5	5.3244e-4
Std. dev.	4.5534e-4	3.7008e-6	0.0

6.5.3. Discussion

The loss curves in Figure 6.13 exhibit no clear signs of overfitting. They are also indicative of appropriately converged models, as the validation losses level off towards the end of training. This suggests that sensible parameter choices were made when setting up this experiment. However, the validation losses are significantly lower than the corresponding training losses, which is unusual. Through an auxiliary experiment, it was determined that the comparatively low validation loss was primarily due to dropout being used during training, but not during validation. For the auxiliary experiment, dropout was not used during neither training nor validation, and the training losses were then lower than the validation losses, as is usually the case. This behaviour can be explained as follows: Adding dropout to a network essentially limits its representative power by eliminating the use of some neurons. This causes a performance drop in comparison to the full network, and thereby an increased loss. Using dropout during training only may therefore yield a training loss that is higher than the validation loss.

At a glance, both corrected temperature profiles in Figure 6.14 appear to be of usable quality. However, looking closely at the end-to-end prediction in Figure 6.14a, one can observe that there is a significant overshoot for $x \in [0.1\,\mathrm{m}, 0.5\,\mathrm{m}]$ and a significant undershoot near the right boundary at $x_b = 1.0\,\mathrm{m}$. Overall, the corrected temperature profile appears to be of worse accuracy than the uncorrected profile. This impression has support in the L_2 error statistics presented in Table 6.6; the average error of the profiles corrected using end-to-end learning is approximately one order of magnitude *larger* than the average error of the uncorrected profiles.

For the hybrid modelling approach, the story is quite different. With this correction approach, the average error of the corrected profiles is roughly one order of magnitude *smaller* than that of the uncorrected profiles, which is a significant improvement. Since the error of the low quality simulation in this experiment is a combination of discretization error and errors due to flawed assumptions in the numerical simulation, one may interpret this result to suggest that hybrid modelling is well-suited to correcting both kinds of errors. However, the current experimental setup does not allow for distinguishing the approach's performance for one of the error types individually.

On the basis of this experiment, hybrid modelling appears to satisfy some requirements for digital twin applications. It appears to be generalizable, as it is able to perform successful corrections for time levels well beyond those used for training. The improved accuracy and physicality of the corrected solution further suggest that the approach is able to provide trustworthy corrections. On the other hand, this experiment has proven that end-to-end learning struggles when the error of the uncorrected numerical solution is comparatively small. It is clear that any correction methods which increases the error of is unsuited for use in any context, including that of digital twin applications.

Hybrid modelling likely outperforms end-to-end learning in this experiment because it has less strict accuracy requirements for the neural network. Since the errors to be corrected in this experiment are comparatively small, the target correction source terms are also small in the sense that the magnitude of the corrections they represent are small in comparison to the temperatures being corrected. Thus, if the correction source terms generated by the neural network have a relative error with respect to the corresponding target outputs of, say, 10%, the relative error of the corrected temperature profile will be significantly smaller than 10%. However, for end-to-end learning, the network output *is* the corrected temperature profile. So naturally, if the neural network is off by 10%, this will cause a very significant error in the corrected temperature profile of 10%. In essence, the error made by the neural network is less significant for the hybrid modelling approach because a lot of the underlying physics are contained in the discretized equations to which the correction source term is added, as discussed in Section 4.3. This stands in contrast to the end-to-end learning approach, where the neural network essentially has to learn *all* the underlying physics, including what is covered by the governing equation. End-to-end learning is therefore more difficult, which likely explains the observed difference in performance.

6.6. Unsteady Experiment #2 – Global Error

In Unsteady Experiment #2, we aim to investigate the stability of numerical simulations where the corrected solution after one time step is used as the initial condition for the next time step. More precisely, if $\boldsymbol{T}_{\mathrm{c}}^{n} = \boldsymbol{T}_{\mathrm{ref}}^{n}$ for some time level t^n and $m > 1$ is an integer, we are interest in the accuracy of the corrected predictions $\boldsymbol{T}_{\mathrm{c}}^{n+m}$ and the uncorrected predictions $\boldsymbol{T}_{\mathrm{u}}^{n+m}$ with respect to the reference solution $\boldsymbol{T}_{\mathrm{ref}}^{n+m}$. In other words, we want to investigate the *global* error of the corrected and uncorrected simulation scheme after m time steps.

6.6.1. Configuration

The only difference between this experiment and Unsteady Experiment #1 is the way we generate the test set. In this experiment, we use $\boldsymbol{T}_{u}^{n}$ rather than $\boldsymbol{T}_{\mathrm{ref}}^{n}$ to generate $\boldsymbol{T}_{u}^{n+1}$ for the *test set*. This allows for error accumulation in the low quality simulation from time $t = 2.1\,\mathrm{s}$ and onward. Note that in the training and validation sets, we still use $\boldsymbol{T}_{\mathrm{ref}}^{n}$ to generate $\boldsymbol{T}_{u}^{n+1}$. This choice was made to ensure that our correction methods remain interpretable. However, it means that our correction methods are still only trained to make local corrections. Thus, if the corrected profile at any time level contains some error, the correction methods are not trained to correct for this for any later time steps. We must therefore expect worse performance in this experiment than in Unsteady Experiment #1.

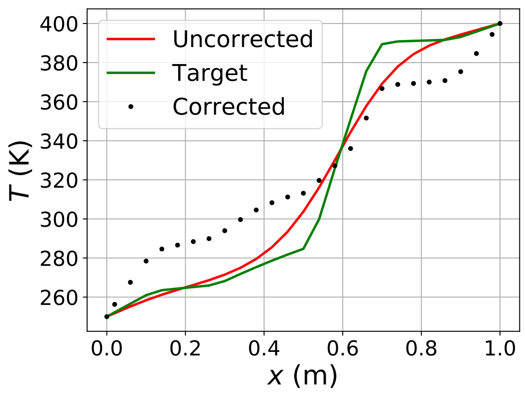
6.6.2. Results

Statistics of the normalized L_2 errors of corrected temperature profiles $\boldsymbol{T}_{\mathrm{c}}^{n}$ and uncorrected temperature profile $\boldsymbol{T}_{\mathrm{u}}^{n}$ are given in Table 6.7. The statistics are presented for two time levels, $n = 2200$ and $n = 3100$. As before, these statistics are calculated based on the results of eight neural network model instances. The statistics for time level $n = 2200$ (corresponding to $t = 2.2\,\mathrm{s}$) are illustrated qualitatively in Figure 6.17. Figure 6.16 shows the corrected temperature profiles at the intermediate time level $t = 2.2\,\mathrm{s}$ and the final time level $t = 3.1\,\mathrm{s}$, as computed by the representative end-to-end model instance (top row) and the representative hybrid modelling instance (bottom row). At these time levels, the global error has accumulated for $0.1\,\mathrm{s}$ and $1.0\,\mathrm{s}$, respectively.

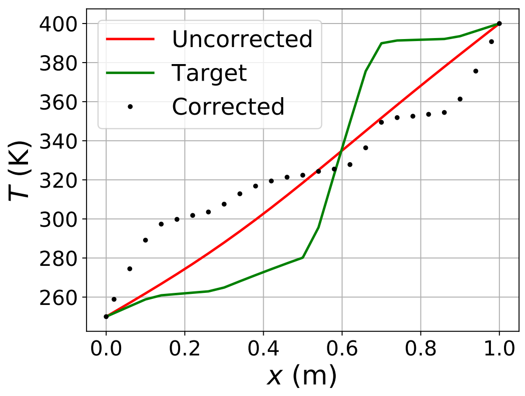
Note that since the training routine, training set and validation set are all the same as in Unsteady Experiment #1, the training and validation losses are also the same as for that experiment. These losses are illustrated in Figure 6.13.

6.6.3. Discussion

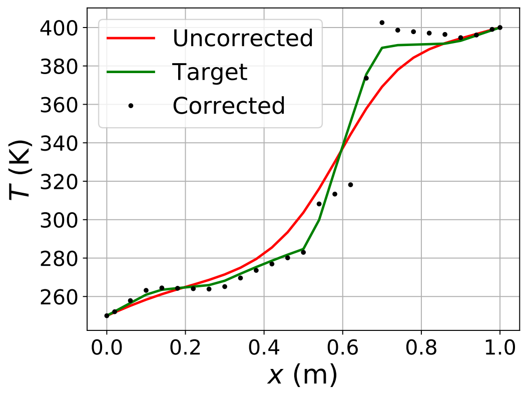
From the top row of Figure 6.16, it is clear that the end-to-end approach does not yield usable corrections for the present case. While the corrected profiles have, to some extent, captured the piecewise linear form of the reference solutions, they are otherwise highly inaccurate. For most grid points, the temperature is even increased where it should be decreased and vice versa, thereby making the corrected solutions more inaccurate than the uncorrected ones. From Table 6.7 and Figure 6.17, we find that this accuracy reduction is not unique to the two corrected solutions shown in Figure 6.16; the average L_2 error of the profiles corrected using the end-to-end approach is several times larger than the average L_2 error of the uncorrected profiles, both at $t = 2.2\,\mathrm{s}$ and $t = 3.1\,\mathrm{s}$.



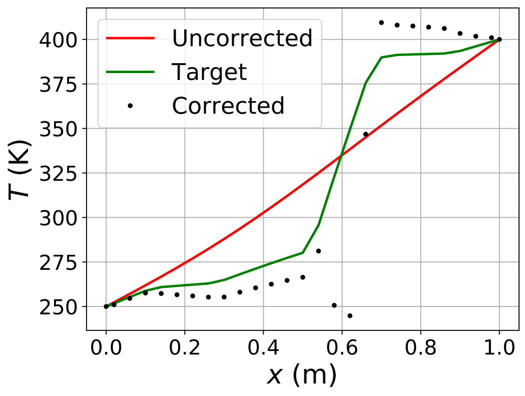
(a) End-to-end learning, $t = 2.2\,\mathrm{s}$



(b) End-to-end learning, $t = 3.1\,\mathrm{s}$



(c) Hybrid modelling, $t = 2.2\,\mathrm{s}$



(d) Hybrid modelling, $t = 3.1\,\mathrm{s}$

Figure 6.16.: Unsteady Experiment #2: Corrected temperature profiles for end-to-end learning (top row) and hybrid modelling (bottom row). The corresponding uncorrected profiles and reference profiles (labelled "Target") are included for comparison.

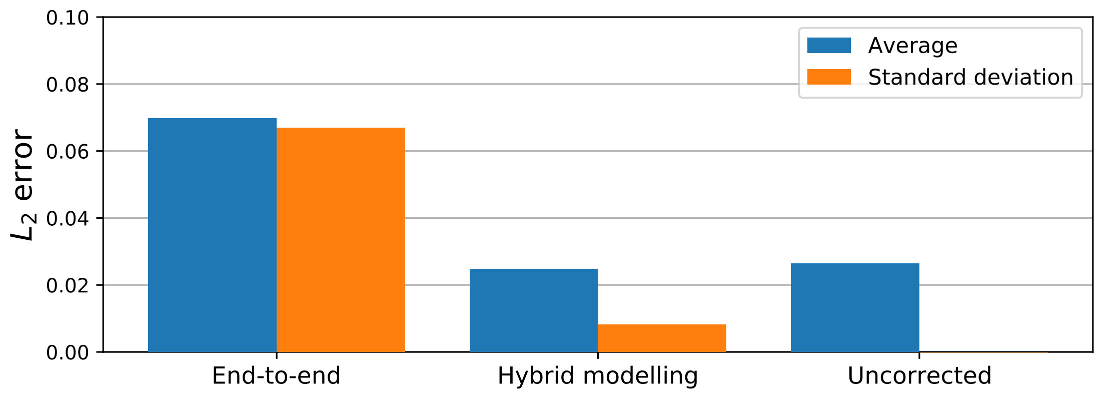


Figure 6.17.: Unsteady experiment #2: Qualitative illustration of L_2 error statistics for unsteady temperature profiles at time $t = 2.2\,\mathrm{s}$ generated using end-to-end learning, hybrid modelling and an uncorrected simulation. Note the logarithmic scaling of the vertical axis.

Table 6.7.: Unsteady Experiment #2: L_2 error statistics of predicted temperature profiles for end-to-end learning, hybrid modelling and an uncorrected simulation.

	Time	End-to-end	Hybrid modelling	Uncorrected
Average,	$t = 2.2\,\mathrm{s}$	6.9753e-2	2.4691e-2	2.6341e-2
Std. dev.,	$t = 2.2\,\mathrm{s}$	6.6843e-2	8.1959e-3	0.0
Average,	$t = 3.1\,\mathrm{s}$	1.6944e-1*	9.2467e-2	6.9085e-2
Std. dev.,	$t = 3.1\,\mathrm{s}$	2.4271e-1*	3.4330e-2	0.0

* Numerical overflow occurred for the L_2 error of one end-to-end learning model instance at $t = 3.1\,\mathrm{s}$. The marked results were calculated based on the remaining seven end-to-end instances.

The L_2 error of one end-to-end instance even diverged before the end of the simulation, yielding numerical overflow.

Looking at the temperature profiles corrected using hybrid modelling in the bottom row of Figure 6.16, one can observe that the corrected temperature profile after 100 test iterations ($t = 2.2\,\mathrm{s}$) is very similar to the reference profile for $x \in [0.0\,\mathrm{m}, 0.5\,\mathrm{m}]$ and $x \in [0.9\,\mathrm{m}, 1.0\,\mathrm{m}]$. However, for other values of x there are significant discrepancies. These are especially prominent for $x \approx 0.6\,\mathrm{m}$. After another 900 iterations, the error in the neighbourhood of $x = 0.6\,\mathrm{m}$ has increased significantly and resulted in a corrected temperature profile that is clearly unphysical. The simulations were not continued past $t = 3.1\,\mathrm{s}$, but the development from time $t = 2.2\,\mathrm{s}$ to $t = 3.1\,\mathrm{s}$ is reminiscent of divergent behaviour. Inspection of the datasets did not provide any insight into why the error was much greater for x around $0.6\,\mathrm{m}$. This suggests that the simulation corrected using hybrid modelling is unstable and unpredictable, and therefore not trustworthy.

From Figure 6.17 and Table 6.7, we see that the average L_2 error for the hybrid modelling approach at $t = 2.2\,\mathrm{s}$ is roughly the same as the average L_2 error of the uncorrected profiles. At $t = 3.1\,\mathrm{s}$, the uncorrected profiles are significantly more accurate, as can be seen from Table 6.7. Thus, we must conclude that neither correction approach is able to consistently provide accurate and reliable corrections over periods of time that encompass many time levels. However, the results for hybrid modelling suggest that improved accuracy can be obtained for shorter time periods. This observation is in line with the findings from the previous experiment. In any case, the discussion above has revealed that lack of accuracy is not the only issue of the corrected simulations.

6.7. Unsteady Experiment #3 – Data Augmentation

In Unsteady Experiment #3, we investigate the influence of data augmentation on the performance of the presented correction approaches. For this purpose, we repeat Unsteady Experiment #2, but this time we increase the amount of training data using two problem-specific data augmentation techniques.

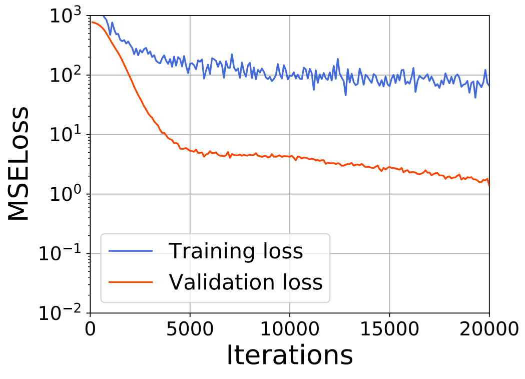
6.7.1. Configuration

The data generation process for this experiment is the same as the one used for Steady-State Experiment #2, with the exception that the training dataset was extended using two data augmentation techniques. These are both based on the physical properties of the system we are studying, and we refer to them as *shift augmentation* and *mirror augmentation.*

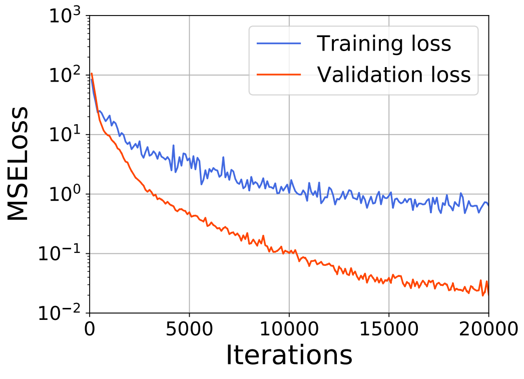
Shift augmentation is based on adding some constant C to all inputs $\boldsymbol{T}_{\mathrm{u}}$. For end-to-end learning, the same constant C must then be added to the targets $\boldsymbol{T}_{\mathrm{ref}}$. Because the correction source term only depends on temperature *gradients*, the targets $\hat{\boldsymbol{\sigma}}_{\mathrm{ref}}$ for hybrid modelling are unaffected by the addition of the constant C to the temperature profiles.

Mirror augmentation refers to the process of creating new data by mirroring the inputs $\boldsymbol{T}_{\mathrm{u}}$ around the center line of the physical domain, $x_{center} = \frac{1}{2}(x_b - x_a)$. For both end-to-end learning and hybrid modelling, the targets $\boldsymbol{T}_{\mathrm{ref}}$ or $\hat{\boldsymbol{\sigma}}_{\mathrm{ref}}$ must then be mirrored in the same way. In addition, since all temperature gradients switch signs when the temperature profiles are mirrored, the target $\hat{\boldsymbol{\sigma}}_{\mathrm{ref}}$ for hybrid modelling must also have their signs inverted.

In this experiment, we use shift augmentation five times with constants $C_l = l\,(T_b - T_a)$, $l = 1, 2, 3, 4, 5$. We then use mirror augmentation on both the original training set and the new training data obtained using shift augmentation. Through this procedure, we increase the amount of available training data by a factor 12. To account for the extra data, we increase the number of training iterations to 20 000 and increase the mini-batch



(a) End-to-end learning



(b) Hybrid modelling

Figure 6.18.: Unsteady Experiment #3: Mean square error loss on training and validation set for two representative model instances – one instance for each correction approach.

size to 64. All other machine learning-related parameters are the same as in Unsteady Experiments #1 and #2.

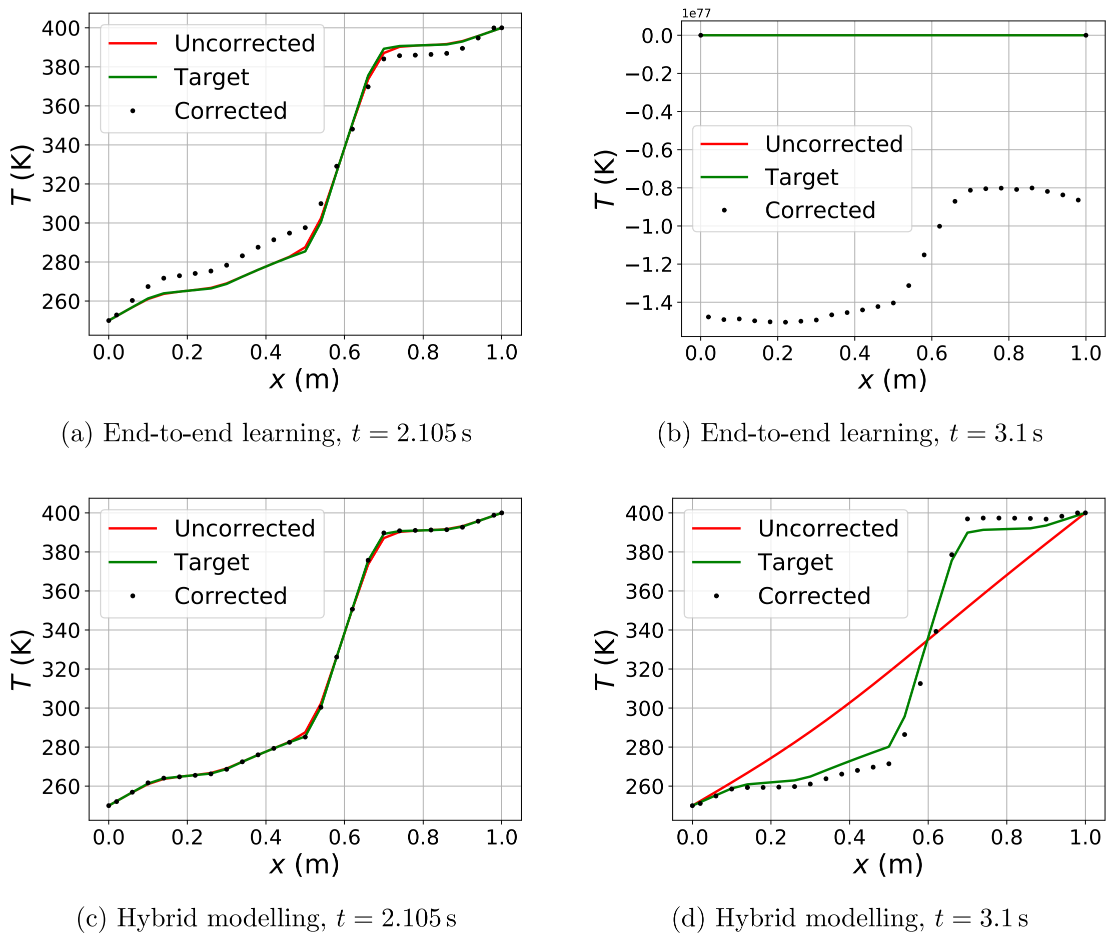
6.7.2. Results

Statistics of the normalized L_2 errors of corrected temperature profiles $\boldsymbol{T}_{\mathrm{c}}^n$ and uncorrected temperature profile $\boldsymbol{T}_{\mathrm{u}}^n$ are given in Table 6.9. The statistics are presented for two time levels, $n = 2120$ and $n = 3100$. As in earlier experiments, these statistics are calculated based on the results of eight neural network model instances. The statistics for time level $n = 2120$ are illustrated qualitatively in Figure 6.20.

The training and validation losses for an end-to-end model instance and a hybrid modelling instance are given in Figure 6.18a and 6.18b, respectively. Figure 6.19 shows the corrected temperature profiles at the intermediate time level $t = 2.120\,\mathrm{s}$ and the final time level $t = 3.1\,\mathrm{s}$, as computed by the same end-to-end model instance (top row) and hybrid modelling instance (bottom row). As in the earlier experiments, the hybrid modelling instance is a representative instance, in the sense that its performance is close to the average performance of all the hybrid modelling instances. However, for end-to-end learning, we have plotted the results for the best performing instances. This is because numerical overflow was encountered during testing of 6 out of the 8 end-to-end learning instances.

6.7.3. Discussion

The most striking result from this experiment is the divergent behaviour of end-to-end learning. In Figure 6.19a, we see that the error of the corrected solution is much greater than the error of the uncorrected solution after only 5 iterations. After the full test period of 1000 iterations, the corrected temperature profile is completely nonsensical,



(a) End-to-end learning, $t = 2.105\,\mathrm{s}$

(b) End-to-end learning, $t = 3.1\,\mathrm{s}$

(c) Hybrid modelling, $t = 2.105\,\mathrm{s}$

(d) Hybrid modelling, $t = 3.1\,\mathrm{s}$

Figure 6.19.: Unsteady Experiment #3: Corrected temperature profiles for end-to-end learning (top row) and hybrid modelling (bottom row). The corresponding uncorrected profiles and reference profiles (labelled "Target") are included for comparison.

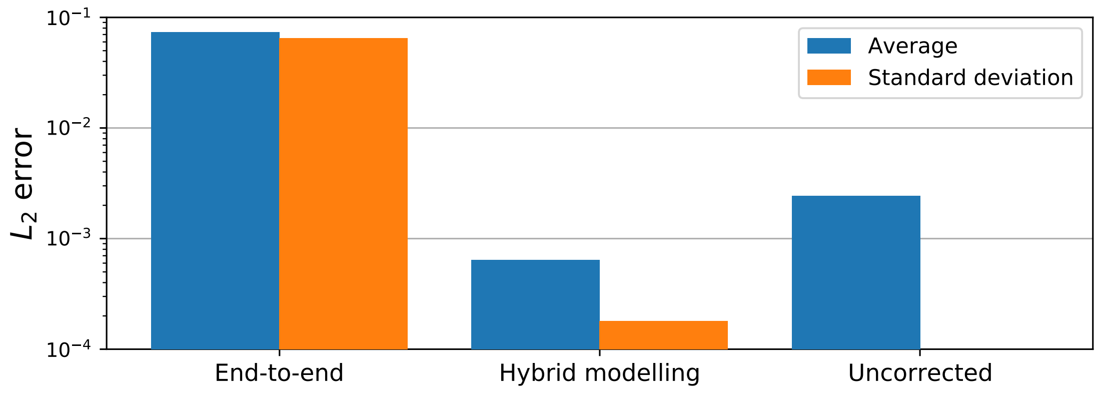


Figure 6.20.: Unsteady experiment #3: Qualitative illustration of L_2 error statistics for unsteady temperature profiles at time $t = 2.105\,\mathrm{s}$ generated using end-to-end learning, hybrid modelling and an uncorrected simulation. Note the logarithmic scaling of the vertical axis.

Table 6.8.: Unsteady Experiment #3: L_2 error statistics of predicted temperature profiles for end-to-end learning, hybrid modelling and an uncorrected simulation.

	Time	End-to-end	Hybrid modelling	Uncorrected
Average,	$t = 2.105\,\mathrm{s}$	7.3499e-2	6.4279e-4	2.4361e-3
Std. dev.,	$t = 2.105\,\mathrm{s}$	6.4957e-2	1.7977e-4	0.0
Average,	$t = 3.1\,\mathrm{s}$	6.3e105*	1.4929e-2	6.9085-2
Std. dev.,	$t = 3.1\,\mathrm{s}$	8.9e105*	5.0715e-3	0.0

with *negative* temperatures of magnitude on the order of 10^{77}. Considering also that 6 out of 8 end-to-end learning instances encountered numerical overflow during testing, it is clear that end-to-end learning is not a useful correction approach with the setup used in this experiment.

The explanation of the poor performance of end-to-end learning can possibly be found in Figure 6.18a. The loss curves shown are unusually flat during the last 15 000 training iterations, and actually starts to decrease more rapidly towards the end of the training. This suggests that the model had not yet converged before the training ended, probably as a result of the increased complexity caused by the increased amount of training data. Therefore, we hypothesize that extending the training period would yield significantly better results for end-to-end learning. This hypothesis could have been tested by performing another experiment with a longer training period. However, such an experiment has not been included in the present work due to time constraints. In any case, the experiment included here demonstrates that end-to-end learning is a vulnerable correction approach, as certain parameter choices can yield divergent behaviour.

The losses shown in Figure 6.18b do not indicate poor convergence for hybrid modelling. To the contrary, they are consistent with an appropriately converged model.

Since shift augmentation creates more variation in the training dataset for end-to-end learning than for hybrid modelling, it is not surprising that hybrid modelling is able to converge faster than end-to-end learning. The greater complexity of the learning task for end-to-end learning compared to that of hybrid modelling is likely also a contributing factor.

Comparing the results for hybrid modelling in this experiment with those from the previous experiment, it is clear that data augmentation has improved the performance of hybrid modelling. Hybrid modelling has reduced the error of the uncorrected temperature profile by a factor of roughly 3 at $t = 2.2\,\mathrm{s}$ and roughly 4.5 at $3.1\,\mathrm{s}$, as can be seen from Table 6.8. From Figure 6.19, we see that the unphysical behaviour observed at $x \approx 0.6\,\mathrm{m}$ in Figure 6.16 is no longer present. However, the corrected temperature profile at $t = 3.1\,\mathrm{s}$ is still unphysical as it is not monotonously increasing in the region $x \in [0.8, 1.0]$. Despite the observed accuracy improvement, hybrid modelling therefore fails to satisfy the trustworthiness-requirement of digital twin applications.

6.8. Unsteady Experiment #4 – Global Error II

In this final unsteady experiment, our objective is to investigate if the qualitative behaviour observed in Unsteady Experiment #2 carries over to a different problem. To this end, we study the same physical system as before, but let its conductivity k be given by Equation (6.2) rather than Equation (6.1).

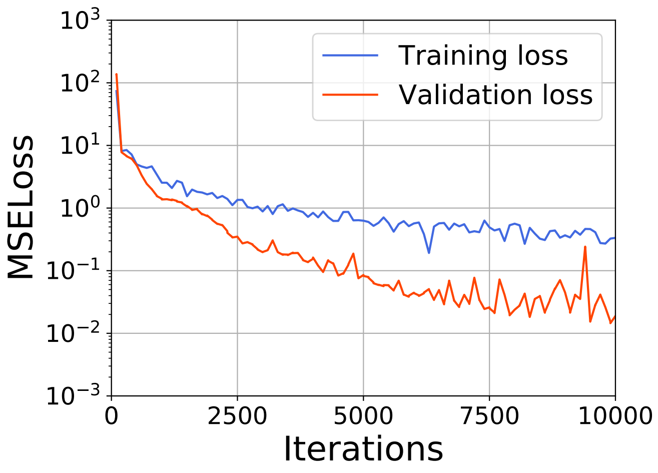
6.8.1. Configuration

Since this experiment is aimed at reproducing the results from Unsteady Experiment #2, we revert to using the machine learning parameters from that experiment. We also use the same data generation process as in Unsteady Experiment #2, except for the use of a different IC and different BCs. $T_a = 600\,\mathrm{K}$ and $T_b = 1000\,\mathrm{K}$ were used as BCs in this experiment, while the IC used was $T^0(x) = 800\,\mathrm{K}\ \forall x \in (x_a, x_b)$. No data augmentation was used in this experiment.

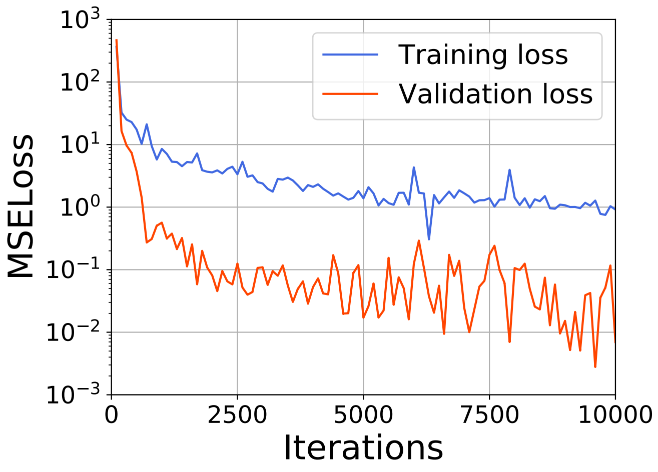
6.8.2. Results

Statistics of the L_2 errors of corrected temperature profiles $\boldsymbol{T}_{\mathrm{c}}^n$ and uncorrected temperature profiles $\boldsymbol{T}_{\mathrm{u}}^n$ are given in Table 6.9. The statistics are presented for two time levels, $n = 2105$ and $n = 3100$. As in earlier experiments, these statistics are calculated based on the results of eight neural network model instances. The statistics for time level $n = 2105$ are illustrated qualitatively in Figure 6.23.

The training and validation losses for a representative end-to-end model instance and a representative hybrid modelling instance are given in Figure 6.21a and 6.21b, respectively. Figure 6.22 shows the corrected temperature profiles at the intermediate time level $t = 2.105\,\mathrm{s}$ and the final time level $t = 3.1\,\mathrm{s}$, as computed by the representative end-to-end model instance (top row) and the representative hybrid modelling instance (bottom row).

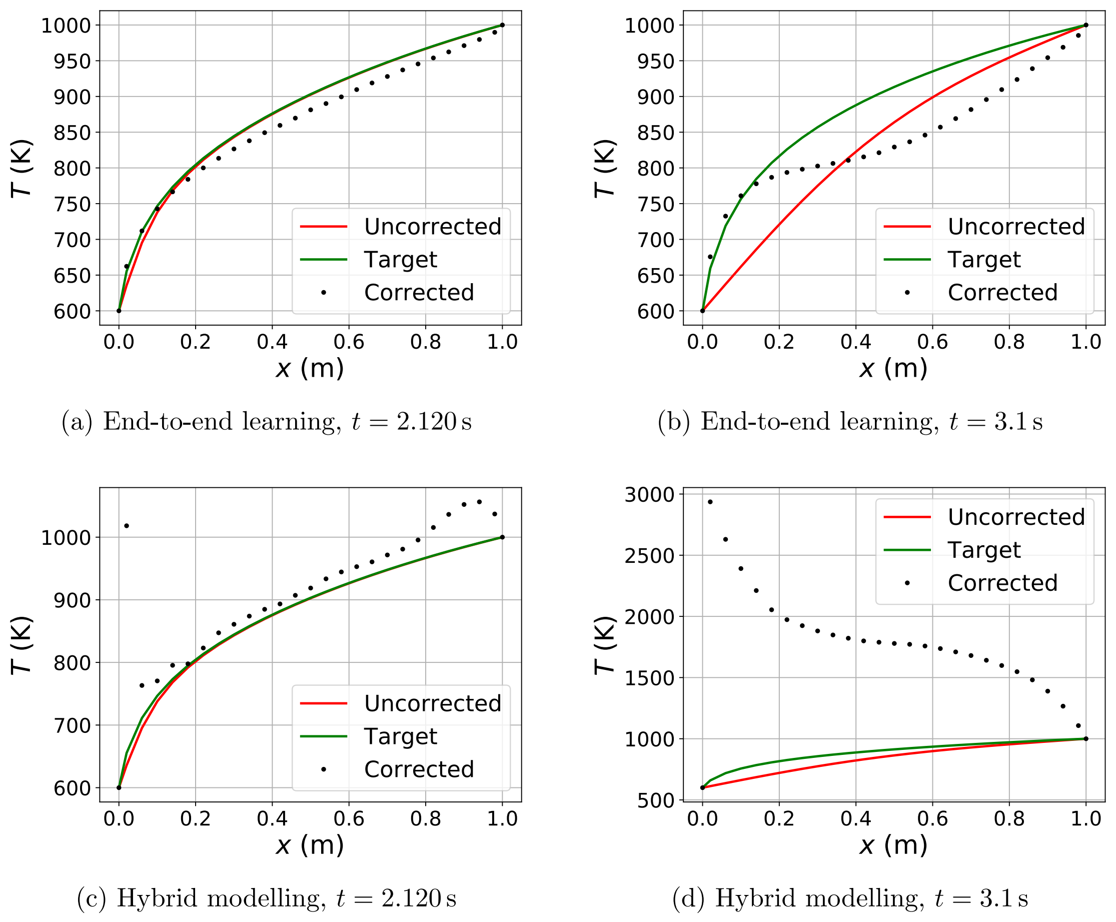


(a) End-to-end learning



(b) Hybrid modelling

Figure 6.21.: Unsteady Experiment #4: Mean square error loss on training and validation set for two representative model instances – one instance for each correction approach.



(a) End-to-end learning, $t = 2.120\,\mathrm{s}$

(b) End-to-end learning, $t = 3.1\,\mathrm{s}$

(c) Hybrid modelling, $t = 2.120\,\mathrm{s}$

(d) Hybrid modelling, $t = 3.1\,\mathrm{s}$

Figure 6.22.: Unsteady Experiment #4: Corrected temperature profiles for end-to-end learning (top row) and hybrid modelling (bottom row). The corresponding uncorrected profiles and reference profiles (labelled “Target”) are included for comparison.

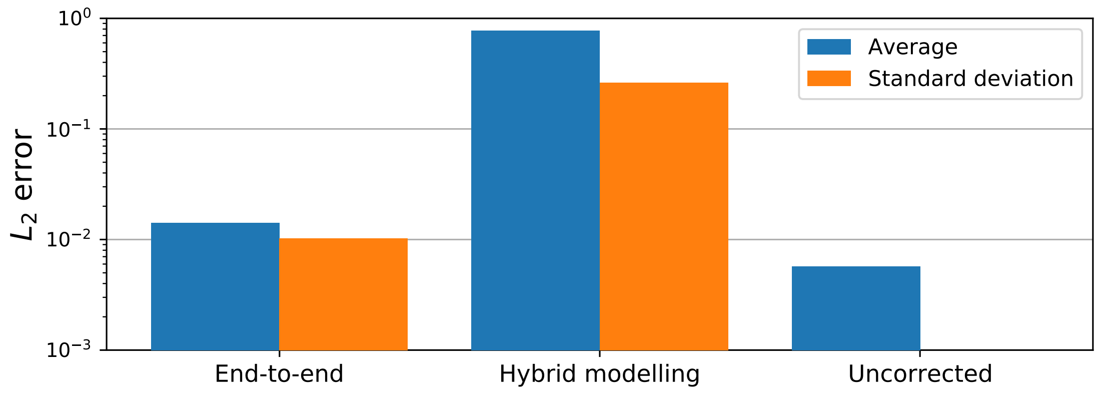


Figure 6.23.: Unsteady experiment #4: Qualitative illustration of L_2 error statistics for unsteady temperature profiles at time $t^{2020} = 2.020\,\mathrm{s}$ generated using end-to-end learning, hybrid modelling and an uncorrected simulation. Note the logarithmic scaling of the vertical axis.

Table 6.9.: Unsteady Experiment #4: L_2 error statistics of predicted temperature profiles for end-to-end learning, hybrid modelling and an uncorrected simulation.

	Time	End-to-end	Hybrid modelling	Uncorrected
Average,	$t = 2.120\,\mathrm{s}$	1.4188e-2	7.7131e-1	5.7066e-3
Std. dev.,	$t = 2.120\,\mathrm{s}$	1.0219e-2	2.6193e-1	0.0
Average,	$t = 3.1\,\mathrm{s}$	7.1423e-2	2.2069e0	6.9085-2
Std. dev.,	$t = 3.1\,\mathrm{s}$	3.8166e-2	3.7008e0	0.0

6.8.3. Discussion

For the end-to-end approach, the results from this experiment are qualitatively similar to those from Unsteady Experiment #2. The loss curves in Figure 6.21a indicate adequate model convergence during training, but the corrections are simply not accurate enough to be of any use in digital twin applications. From the Figure 6.23 and the top row of Figure 6.22, it is clear that the temperature profiles corrected using end-to-end learning are less accurate than the uncorrected profiles.

Surprisingly, hybrid modelling also performs poorly in this experiment; temperature profiles corrected using hybrid modelling are significantly less accurate than uncorrected profiles for the first and only time in the present work. In fact, the hybrid modelling-corrected simulation appears to diverge at the left side of the domain, as can be seen from the bottom row of Figure 6.22. The divergence is particularly strong for the grid point at $x = 0.02\,\mathrm{m}$. This result is interesting because it demonstrates that hybrid modelling, in its current form, is not suited for safety-critical simulations because stability cannot be guaranteed.

In this experiment (and the other unsteady experiments of this work), the neural networks are trained to only make individual corrections. They are therefore unable to correct for errors of earlier time steps, and this is what enables the divergent behaviour observed for hybrid modelling in this experiment. After the first test iteration, the corrected temperature at $x = 0.02\,\mathrm{m}$ was too high. By chance, the final neural network parameter values are such that when this too high temperature is fed to the neural network at the next iteration, the predicted correction source term is also too large. In turn, this causes the temperature at the next time level to increase further and become even more erroneous. Divergence occurs as this process repeats over many time levels.

The discussion above explains the mechanism behind the divergence of the hybrid modelling-corrected simulation. However, it does not explain why the error after the first iteration was large enough to cause such rapid divergence. The explanation probably lies in the data normalization. Due to the choice of the constant IC $T(x, t = 0) = 800\,\mathrm{K}$, the temperature gradients near the ends of the domain are very large at the earliest time levels ($t \lesssim 0.05\,\mathrm{s}$). This causes the corresponding reference correction source terms to be very large as well. Consequently, the vast majority of the reference correction source terms are concentrated close to the lower normalization bound of -1 after normalization. The exception is the correction source term at $x = 0.02\,\mathrm{m}$, which varies between 0 and -0.75 for most time levels. Since its absolute value is smaller than the others, predicting the correction source term at $x = 0.02\,\mathrm{m}$ accurately becomes comparatively unimportant for the task of minimizing the mean square error loss function. We hypothesize that, for this reason, the neural network's accuracy at $x = 0.02\,\mathrm{m}$ is worse than for other x. When the correction source terms are unnormalized, the correction source term at $x = 0.02\,\mathrm{m}$ is actually the largest one, and therefore also the one which has the largest impact on the corrected temperature profiles. The seemingly unimportant error in the correction source term at $x = 0.02\,\mathrm{m}$ is therefore magnified, causing a large error in the corrected temperature profile which eventually results in divergent behaviour. To verify that the normalization is in fact an underlying cause of the divergent behaviour, the present experiment was repeated with the initial condition $T(x, t = 0) = 600 + 200x\,\mathrm{K}$ which does not yield the same extreme normalization effects. In the modified experiment, the divergent behaviour at $x = 0.02\,\mathrm{m}$ disappeared, and the results for hybrid modelling were generally similar to those of Unsteady Experiment #2 (cf. Section 6.6).

Another curious result of this experiment is the erratic validation loss for the hybrid modelling approach, as shown in Figure 6.21b. Erratic loss curves can often be a sign that the learning rate is too high. However, when the learning rate is too high, the training loss is usually also quite erratic. Furthermore, it is not clear why a learning rate that has worked well in the previous three unsteady experiments should be considerably less suitable for this experiment. Lastly, the validation loss was not erratic in the auxiliary experiment discussed above where a different IC was used. We therefore believe that the erratic validation loss is not caused by a poorly chosen learning rate, but rather is an effect of the normalization-related issues discussed earlier.

6.9. Summary

In Steady-State Experiment #1 (cf. Section 6.2), we tested our data-driven correction approaches in an interpolation context. It was then observed that both end-to-end learning and hybrid modelling reduced the average L_2 error of the predicted steady-state temperature profiles by a factor 15 in comparison to the uncorrected predictions. We took this as an indication that the correction approaches were likely implemented correctly and could be used for further experiments. In the two subsequent steady-state experiments (cf. Section 6.3 and Section 6.4), we explored the correction method's ability to extrapolate to temperature profiles with temperatures sampled from outside the range covered by the training data. We then discovered that the accuracy of the corrections of both approaches varied significantly, and that both approaches yielded unphysical temperature profiles for certain test examples. However, the corrected temperature profiles were, on average, more accurate than the uncorrected temperature profiles. Moreover, hybrid modelling significantly outperformed end-to-end learning.

In Unsteady Experiment #1 (cf. Section 6.5), we studied the local error of the Implicit Euler FVM scheme with and without data-driven corrections. We then observed that end-to-end learning, our simplest correction method, was unable to provide useful corrections. On the other hand, the temperature profiles corrected using the hybrid modelling approach were significantly more accurate than the corresponding uncorrected temperature profiles. In Unsteady Experiment #2 (cf. Section 6.6), where we studied the global error of the FVM scheme with and without corrections, we found that the end-to-end learning approach continued to provide unusable corrections. We also found that the temperature profiles corrected using the hybrid modelling approach were no longer significantly more accurate than the uncorrected profiles, and that they were also becoming decidedly unphysical with apparent divergence at certain grid nodes. The results for the hybrid modelling approach were greatly improved through the use of data augmentation in Unsteady Experiment #3. On the other hand, the addition of data augmentation made the end-to-end approach perform worse than before, possibly due to insufficient training time. The last experiment, Unsteady Experiment #4 (cf. Section 6.8), was aimed at replicating the results from Unsteady Experiment #2 for a different physical system. For the end-to-end learning approach, Unsteady Experiment #4 gave qualitatively similar results as Unsteady Experiment #2, i.e. the corrected solutions were generally less accurate than the uncorrected solutions. However, for hybrid modelling, the results from the two experiments differed to a greater extent. While some signs of divergence could be seen in Unsteady Experiment #2, the divergence of the hybrid modelling-corrected simulation was much stronger in Unsteady Experiment #4. The divergent behaviour was likely enhanced by normalization-related effects which could have been avoided. Regardless, the fact that such divergent behaviour is possible significantly reduces the trustworthiness of hybrid modelling-corrected simulations.

7. Discussion

This chapter contains a holistic discussion of the results from the machine learning experiments of the previous chapter. First, in Section 7.1, the observations from the experiments are used to answer the research questions presented in Chapter 1. In Section 7.2, we then briefly discuss possible improvements in the design of the machine learning experiments. Lastly, suggestions on how to improve the data-driven correction methods are presented in Section 7.3.

7.1. Answering the Research Questions

How can a physics-based simulation scheme and a data-driven correction method be combined into a unified model? This question was treated extensively in Chapter 4, so the reader is referred to that chapter for the details. In Sections 4.2 and 4.3, we presented two approaches for creating unified models: end-to-end learning and hybrid modelling. While we have obtained results of varying quality in our machine learning experiments, the accuracy improvements observed in Steady-State Experiment #1 (cf. Section 6.2) are sufficient for concluding that both end-to-end learning and hybrid modelling are valid ways of combining a physics-based simulation scheme and a data-driven correction method into a unified model.

Can these models account for hidden physics? In all our machine learning experiments, the uncorrected simulations assume constant conductivity throughout the physical domain. The true conductivity profile used to generate the reference profiles is therefore an example of hidden physics for which we would like our methods correct. From Steady-State Experiment #1, we have learned that both correction approaches can improve the accuracy of the numerical solutions in some cases. This is sufficient for concluding that both end-to-end learning and hybrid modelling can account for hidden physics *to some extent*. However, throughout the earlier discussions in Chapter 6, the unphyiscality of the corrected solutions has been a recurring theme. It is therefore clear that the unified models developed in this work are unable to account for *all* hidden physics.

In the Unsteady Experiments (Section 6.5 through Section 6.8), some physics were also "hidden" by the temporal discretization error of the uncorrected simulations. Since end-to-end learning failed to provide corrected temperature profiles of acceptable accuracy in any of these experiments, we cannot conclude that end-to-end learning is suited for correcting discretization errors. However, as discussed previously, the poor accuracy of end-to-end learning in these experiments is most likely caused by the need for small corrections. We expect that end-to-end learning would be able to correct discretization error in circumstances where the errors to be corrected are larger in comparison to the magnitude of the uncorrected solutions. As for hybrid modelling, the results of Unsteady Experiments #1 and #3 indicate that hybrid modelling *is* able to correct discretization errors. However, performing an experiment where the discretization was the sole source of error would have allowed for a more certain conclusion.

It is also interesting to consider if the unified models are able to account for all physics that are included in the uncorrected physics-based simulations. We assert that this is not the case, since the uncorrected temperature profiles studied in this work are always physical, while unphysical traits have been observed in the corrected temperature profiles of both end-to-end learning and hybrid modelling. Thus, while the unified models contain information that the purely physics-based simulations do not, some information from the physics-based simulations is also lost. This information trade-off is what causes the unified models to be more accurate than the physics-based simulations in some cases, and less accurate in others. As previously noted, hybrid modelling makes more explicit use of the information encompassed by the physics-based simulation scheme, and therefore performs better than end-to-end learning in most of the experiments covered in Chapter 6. However, the fact that some information from the physics-based simulation is lost clearly shows that there is room for improving both correction approaches. Possible improvements are discussed in Section 7.3.

Do these models satisfy the requirements for use in digital twin applications, as described in Section 1.1? The requirements state that the models should be computationally efficient, generalizable, self-evolving and trustworthy. Data-driven correction methods are in principal always self-evolving, because there is no need to stop the training before the method is incorporated into the unified simulation model. New data can always be added to the training set such that the corrective part of the model can learn from this new data in the same way as it learned from the pre-existing data during the initial training. In Section 4.3, we have further argued that the corrections made using the hybrid modelling approach are themselves interpretable. One interesting possibility, which is discussed further in Section 7.3, is to perform symbolic regression on the neural network output. If the regression is successful, the regressed expressions can be incorporated into the physics-based simulation like ordinary source terms. In this way, hybrid modelling makes it possible for both the physics-based simulation *and* the data-driven corrections to be self-evolving.

From Steady-State Experiments #2 and #3, we see that both end-to-end learning and hybrid modelling can, in cases where large corrections are desired, improve the accuracy of numerical solutions which are significantly different from those seen during training. This demonstrates that both approaches yield unified models that generalize well for simple problems. From Unsteady Experiment #3, we see that hybrid modelling can also improve accuracy on tougher problems where small corrections are desired. Consequently, hybrid modelling can yield satisfactory generalizability on tougher problems as well. We cannot conclude similarly for end-to-end learning. When small corrections were desired, end-to-end learning did not yield improved accuracy for numerical solutions similar to those seen during training either. Thus, it seems that the general accuracy of end-to-end learning is the main issue. As a consequence, it is difficult to evaluate end-to-end learning-based models' ability to generalize.

At present, neither end-to-end learning nor hybrid modelling yield unified models that are sufficiently trustworthy. For both approaches, stability is the main concern. As both approaches yield unified models that can diverge under certain conditions, it is necessary to investigate the possibility of determining some kind of stability condition for the unified models. Without such a condition, neither approach can be applied to unsteady problems for simulations across multiple time levels. Using the methods for single-step predictions might be acceptable in certain cases, but certainly not in safety-critical applications. The risk of obtaining unphysical solutions like those illustrated on the right hand side of Figure 6.7 leaves much to be desired in terms of dependability and trustworthiness.

For end-to-end learning, interpretability is also an issue. While it might be theoretically possible to apply symbolic regression or similar techniques to the corrected temperatures profile, it is unlikely that this would be fruitful. First of all, it would only yield useful results in cases where there exists a (fairly simple) closed-form expression for the temperature profile of the system. Secondly, it would probably be better to simply apply the regression technique to the measured data directly. The other option for interpreting a unified model based on end-to-end learning would be to examine the inner workings of the neural network. This might provide some insight into what features the neural network focuses on, but these are not particularly interesting on their own. They are only interesting if they can be related to know physical processes, and, to the author's knowledge, it is not known how that would be done, or if it is even possible.

In Section 4.3, we claimed that one benefit of hybrid modelling is that the correction source term is interpretable, and that this is a strong point of hybrid modelling. Symbolic regression has been mentioned as one possible way of performing the interpretation. In addition, simpler analyses can also be conducted, as the correction source term can be directly related to the curvature of the numerical solution. For example, a positive correction source term in a 1D heat conduction problem implies that the uncorrected temperature profile is too convex. Looking at the physical system, it can be determined whether or not the change in curvature is sensible. However, the suggested methods of interpretation have not been tested as part of the present work. As will be discussed in Section 7.3, examining the applicability of the suggested methods empirically can form the basis of interesting future research.

Regarding computational efficiency, whether or not the unified models are sufficiently efficient must be investigated on a case-by-case basis. By simplifying the physics-based simulation and reducing the capacity of the neural network, it is, in principle, possible to construct unified models that satisfy any realistic efficiency constraint. However, drastically reducing model complexity will naturally affect the observed accuracy of the models. To the author's knowledge, it is not possible to determine *a priori* how much a model's complexity can be reduced before its accuracy drops beneath some predetermined threshold. Case-by-case investigations are therefore suggested. Since both end-to-end learning and hybrid modelling have failed to consistently provide satisfactory corrected temperature profiles (i.e. corrected temperature profiles that are both physical and more accurate than the corresponding uncorrected profiles), the efficiency of these particular implementation has not been of interest. Investigations of computational complexity has therefore not been part of this work. However, estimates of the number of floating point operations required for one iteration using both end-to-end learning and hybrid modelling are presented in Appendix G. These might prove helpful for efficiency studies in future work.

7.2. Lessons Learned

In any research endeavour, hindsight always reveals that certain parts of the investigation could or should have been carried out differently. The list below briefly summarizes some changes that the author would have made if the present work was to be repeated.

1. Use only one source of error in each unsteady experiment. This would allow for easier and more confident interpretations of the results from these experiments.

2. Extend the training periods and save the neural network models at regular intervals during training. This would permit utilization of the early stopping regularization technique. In addition to potentially improving performance, this would also make it easier to determine if the ML models have converged properly. Another benefit is that it would then be possible to compare models that have converged to a similar extent. This would allow for a fairer comparison in e.g. Unsteady Experiment #3 (cf. Section 6.7), where hybrid modelling seems to have achieved better convergence than end-to-end learning for the constant number of training iterations used for both approaches.

3. With the measure of error defined in Section 4.1, it is impossible for the uncorrected numerical solutions of the steady-state experiments to have zero error. The same also holds for the corrected solutions. It would therefore have been useful to know the minimum possible error, such that the observed errors could be compared to this baseline.

4. Normalize data such that the training data has a mean of 0 and a standard deviation of 1. This commonly used normalization method is better suited for handling data containing a few outliers, such as the data for Unsteady Experiment #4 (cf. Section 6.8). However, it should be noted that if we had been using this normalization method for the present work, we would not have obtained the highly interesting results of Unsteady Experiment #4. One might refer to our choice of normalization method as a happy accident, but it is something the author would have changed nonetheless.

7.3. Extending the Present Work

In Section 7.1, the lack of stability was identified as a major issue for the unified models presented and studied in this work. The lack of stability makes the simulations untrustworthy and also has a negative impact on accuracy. Improving stability would therefore make the unified models far more useful. Ideally, we would like to know *a priori* whether or not the predicted solutions of a unified model are going to be physical, and whether or not the error of the solutions will remain bounded. However, this is difficult because we do not know the exact mapping learned by the neural network involved – the neural network behaves much like a black box. One plausible way of coping with this, which has also been mentioned previously, is to use symbolic regression algorithms to learn a closed-form expression that fits the output of the neural network. Finding such closed-form expressions for end-to-end learning might prove difficult, if at all possible. However, since generating corrections source terms is simpler than generating temperature profiles, we believe that it is worth investigating the use of symbolic regression in conjunction with hybrid modelling. Symbolic regression has already been used to discover governing equations and Hamiltonians from neural network output [17]. It has also been used to discover unknown source terms from sparse observation data [18]. These results from the literature suggest that it should be possible to successfully perform symbolic regression on the output of a properly converged neural network used for hybrid modelling. The study in [17] found the regressed expressions to generalize better than the neural networks from which they were learned. In addition to making hybrid modelling more trustworthy, the use of symbolic regression might therefore also improve generalizability.

Accuracy has also been observed to be an issue for the unified models studied in this work. We propose three different ways of improving the accuracy of the models. The first suggestion is to use different loss functions in place of, or in addition to, the MSE loss function used in the present work. One possibility is to employ the framework of generative adversarial networks [37] to define an adversarial loss. This suggestion is inspired by the work in [15], where the adversarial framework is applied to fluid flow. Within this framework, the network generating corrected temperature profiles (for end-to-end learning) or correction source terms (for hybrid modelling) will play the role of the generator. In addition, a new discriminator network has to be implemented. Its role is to distinguish between output from the generator and either reference temperature profiles (for end-to-end learning) or reference correction source terms (for hybrid modelling). The resulting adversarial loss function for the generator is minimized when the generator is able to fool the discriminator.

Another possibility is to use physics-informed loss functions, as is done for the physics-guided neural networks in [38]. The exact form of such loss functions must be determined on a case-by-case basis. One example for 1D heat conduction problems with no heat generation is a loss function which penalizes corrected temperature profiles that are not monotonously increasing/decreasing. Another example can be given in the case of solving 1D heat conduction with von Neumann BCs[1] using the hybrid modelling approach. In

[1]For von Neumann BCs, heat fluxes – not temperatures – are specified at the boundaries.

this case, the integrated source term *including the correction source term* should equal the total heat entering or leaving the system, which is specified by the BCs.

The second proposal for improved accuracy applies to hybrid modelling, and involves better utilization of spatial information in the data-driven corrections. With our current neural network architecture, all spatial locations are treated equally with respect to each other. Considering that the 1D heat equation is a hyperbolic equation, this might appear reasonable. However, as the systems (3.58) and (3.67) are both tri-diagonal, component j of the optimal correction source term vector depends only on components $j-1$, j and $j+1$ of the reference temperature profile. In other words, the optimal correction source term depends on local information only. It is then not reasonable to use a neural network architecture which treats all locations equally. One possible fix is to adopt the approach of [16], where an individual neural network is used to generate each component of the source term vector. In our context, an ensemble of N_j networks would be required to calculate a full correction source term vector N_j components. In principle, network j should be able to generate component j of the correction source term vector using only data from the nodes $j-1$, j and $j+1$. However, it might be beneficial to include slightly more data, e.g. from nodes $j-2$ and $j+2$ as well. This is because we use the uncorrected temperature profiles as input to the neural network, and these contain less information than the reference profiles that define optimal correction source terms.

Another way to utilize spatial information is to incorporate convolutional layers [39] in the neural network. Depending on the dimensionality of the problem at hand, these can be either one-, two- or three-dimensional. Convolutional layers are commonly used in neural networks processing visual data due to their ability preserve spatial correlations. This trait makes convolutional layers a simpler alternative to the ensemble approach discussed in [16].

The final suggestion for improved accuracy is again inspired by the physics-guided neural networks described in [38]. In their work, the authors use both the input and the output of the physics-based simulation scheme as input to the neural network. In the present work, only the output of the simulation scheme has been used as neural network input. We hypothesize that including more information as input to the neural network will allow the network to learn about the temporal development of the system. This can be particularly useful for oscillatory systems where the same uncorrected temperature profile can appear during different phases of the system's period. At one time, this profile will need a certain set of corrections, while it might need completely different corrections at another time. Without any temporal information, the neural network will be unable to learn these subtleties. Empirical investigations must be carried out in order to determine what kind of neural network input is best. A good starting point might be to use $\boldsymbol{T}_{\mathrm{u}}^{n}$ and/or $\boldsymbol{T}_{\mathrm{ref}}^{n}$ as neural network input in addition to $\boldsymbol{T}_{\mathrm{u}}^{n+1}$.

8. Conclusions and Further Work

8.1. Conclusions

Two data-driven approaches for correcting physics-based numerical simulations, end-to-end learning and hybrid modelling, have been presented and tested in different scenarios on one-dimensional heat conduction problems. Hybrid modelling has been found to generally outperform end-to-end learning, providing superior accuracy in five out of seven experiments. The difference in accuracy is likely due to hybrid modelling making greater use of the physics-based numerical simulation scheme when generating the corrected solutions. Both approaches succeed in performing large corrections, but end-to-end learning fails to improve the accuracy of uncorrected simulation when only small corrections are required. On the other hand, hybrid modelling is still able to improve accuracy in such scenarios, especially when using data augmentation. In summary, both approaches are able to account for hidden physics in some cases, but they do not improve the accuracy of uncorrected simulations in all scenarios explored in this work.

Due to their ability to learn from experience during operation, data-driven correction methods are prime candidates for use in digital twin applications. However, both end-to-end learning and hybrid modelling have exhibited divergent behaviour when the corrected temperature at one time level is used to calculate the temperature at the subsequent time level. Both methods have also produced unphysical temperature profiles in several experiments. In their current form, both methods therefore fail to satisfy the requirement for use in digital twin applications.

8.2. Further Work

While neither correction approach is suitable for use in digital twin applications in their current form, the results for hybrid modelling are promising and justify further research. We believe that combining hybrid modelling with symbolic regression can increase both the trustworthiness and the generalizability of the unified model. Furthermore, we propose three ways of improving model accuracy. The first is to utilize different loss functions, such as physics-informed losses or adversarial losses. Secondly, we suggest utilizing spatial information through the use of convolutional layers in the neural network. Alternatively, spatial information can be utilized by creating an ensemble of neural networks, with each constituent network making corrections within a small part of the physical domain. Lastly, providing the neural network with more information, such as the input of the physics-based simulation scheme, might be beneficial.

In addition to pursuing the suggestions above, further research may also be based on the following open research questions:

- Is it possible to derive a stability condition for numerical schemes corrected using the hybrid modelling approach?
- Is it feasible *in practice* to extract useful and interpretable knowledge from correction source terms generated by a neural network, e.g. through the use of symbolic regression?

Bibliography

[1] E. Glaessgen and D. Stargel, "The digital twin paradigm for future NASA and US Air Force vehicles," in *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*, p. 1818, 2012.

[2] S. Boschert and R. Rosen, *Digital Twin—The Simulation Aspect*, pp. 59–74. Cham: Springer International Publishing, 2016.

[3] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, "Digital twin-driven product design, manufacturing and service with big data," *The International Journal of Advanced Manufacturing Technology*, vol. 94, pp. 3563–3576, 2018.

[4] R. M. C. Portela, C. Varsakelis, A. Richelle, N. Giannelos, J. Pence, S. Dessoy, and M. von Stosch, *When Is an In Silico Representation a Digital Twin? A Biopharmaceutical Industry Approach to the Digital Twin Concept*, pp. 1–21. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020.

[5] A. Rasheed, O. San, and T. Kvamsdal, "Digital twin: Values, challenges and enablers from a modeling perspective," *IEEE Access*, vol. 8, pp. 21980–22012, 2020.

[6] I. Barosan, A. A. Basmenj, S. G. R. Chouhan, and D. Manrique, "Development of a virtual simulation environment and a digital twin of an autonomous driving truck for a distribution center," in *Software Architecture* (H. Muccini, P. Avgeriou, B. Buhnova, J. Camar, M. Caporuscio, M. Franzago, A. Koziolek, P. Scandurr, C. Trubiani, D. Weyns, and U. Zdun, eds.), (Cham), pp. 542–557, Springer International Publishing, 2020.

[7] M. Stanko and M. Stommel, "Digital twin of the polyurethane rotational moulding process," in *Advances in Polymer Processing 2020* (C. Hopmann and R. Dahlmann, eds.), (Berlin, Heidelberg), pp. 324–335, Springer Berlin Heidelberg, 2020.

[8] B. Müller, "Introduction to computational fluid dynamics." Lecture notes for the course TEP4165 Computational Heat and Fluid Flow, 2018.

[9] A. M. Log, "Development and investigation of HLLC-type finite-volume methods for one and two-phase flow in pipes with varying cross-sectional area," Master's thesis, NTNU, 2020. Available online at `https://alexandramlog.wordpress.com/research/`, accessed 17.12.2020.

[10] W. K. George, "Lectures in turbulence for the 21st century," 2013. Available online at `http://www.turbulence-online.com/Publications/Lecture_Notes/Turbulence_Lille/TB_16January2013.pdf`, accessed 05.12.2020.

[11] A. Khajeh-Saeed and J. B. Perot, "Direct numerical simulation of turbulence using GPU accelerated supercomputers," *Journal of Computational Physics*, vol. 235, pp. 241–257, 2013.

[12] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, "Integrating physics-based modeling with machine learning: A survey," *arXiv preprint arXiv:2003.04919*, 2020.

[13] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving non-linear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[14] G. P. P. Pun, R. Batra, R. Ramprasad, and Y. Mishin, "Physically informed artificial neural networks for atomistic modeling of materials," *Nature communications*, vol. 10, no. 1, pp. 1–10, 2019.

[15] Y. Xie, E. Franz, M. Chu, and N. Thuerey, "tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–15, 2018.

[16] R. Maulik, O. San, A. Rasheed, and P. Vedula, "Subgrid modelling for two-dimensional turbulence using neural networks," *Journal of Fluid Mechanics*, vol. 858, p. 122–144, 2019.

[17] M. Cranmer, A. S. Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho, "Discovering symbolic models from deep learning with inductive biases," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[18] H. Vaddireddy, A. Rasheed, A. E. Staples, and O. San, "Feature engineering and symbolic regression methods for detecting hidden physics from sparse sensor observation data," *Physics of Fluids*, vol. 32, no. 1, p. 015113, 2020.

[19] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural*

Information Processing Systems 32 (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[21] T. M. Mitchell, *Machine Learning.* McGraw-Hill Science/Engineering/Math, 1997.

[22] I. Salian, "SuperVize me: What's the difference between supervised, unsupervised, semi-supervised and reinforcement learning?," 2018. Blog post, `https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/`, accessed: 17.11.2020, 17:24.

[23] A. Trask, D. Gilmore, and M. Russell, "Modeling order in neural word embeddings at scale," *arXiv preprint arXiv:1506.02338*, 2015.

[24] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, JMLR Workshop and Conference Proceedings, 11–13 Apr 2011.

[25] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[26] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30 No. 1, p. 3, 2013.

[27] M. A. Nielsen, *Neural Networks and Deep Learning.* Determination Press, 2015.

[28] C. Heil, *Metrics, Norms, Inner Products, and Operator Theory*, p. 32. Springer International Publishing AG, 2018.

[29] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] S. Kjelstrup, D. Bedeaux, E. Johannessen, and J. Gross, *Non-equilibrium Thermodynamics For Engineers*, p. 27. World Scientific Publishing Co, 2017.

[32] J. Taler and P. Duda, *Fourier Law*, pp. 3–6. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[33] R. A. Adams and C. Essex, *Calculus 2*, p. 925. Pearson Education Limited, 2013.

[34] J. C. Tannehill, D. A. Anderson, and R. H. Pletcher, *Computational Fluid Mechanics and Heat Transfer*, p. 130. Taylor & Francis, 1997.

[35] A. Abdulhaque, "Isogeometrisk analyse av Boussinesq ligningene," Master's thesis, NTNU, 2016.

[36] T. Stöttner, "Why data should be normalized before training a neural network," 2019. Available online at `https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d`, accessed 08.01.2020.

[37] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, pp. 2672–2680, 2014.

[38] A. Karpatne, W. Watkins, J. Read, and V. Kumar, "Physics-guided neural networks (PGNN): An application in lake temperature modeling," *arXiv preprint arXiv:1710.11431*, 2017.

[39] F. Lindseth, "DL-based CV: CNNs & ImClass." Lecture notes for the course TDT4265 Computer Vision and Deep Learning, 2020.

A. Hardware and Python Packages

All simulations and experiments described in this report can be run on most modern laptops. Throughout this work, the author has been working exclusively on an Acer Aspire E 15 laptop with 8 GB DDR4 memory and an Intel i7-7500U processor with a clock speed of 2.7 GHz. All code was written in Python 3.6.9 using the packages listed in Table A.1.

Name	Version
matplotlib	3.3.1
numpy	1.19.1
scipy	1.5.2
torch	1.4.0

Table A.1.: Python packages used during code development in conjunction with the present work.

B. Harmonic Average in Heat Conduction Problems

Suppose we have two adjoining control volumes with constant cross-sectional area A, each with their own constant conductivity, which we denote k_1 and k_2 respectively. Furthermore, let x_1 be a grid node in the left control volume, let $x_{3/2}$ be the location of the boundary between the control volumes, and let x_2 be a grid node in the right control volume. Moreover, let T_1, $T_{3/2}$ and T_2 denote the temperature at these locations. Since the conductivity of both control volumes is constant, we know that the steady-state temperature profile within each control volume must be linear. Applying Fourier's law to the control volumes individually then yields

$$\boldsymbol{q} = -k_1 \nabla T = -k_1 \frac{T_{3/2} - T_1}{x_{3/2} - x_1} \hat{\boldsymbol{x}} \tag{B.1}$$

for the left control volume and

$$\boldsymbol{q} = -k_2 \nabla T = -k_2 \frac{T_2 - T_{3/2}}{x_2 - x_{3/2}} \hat{\boldsymbol{x}} \tag{B.2}$$

for the right control volume. Under the assumption that no heat is generated precisely at the boundary, the heat flow $\boldsymbol{q}$ at the left and right side of the boundary must be equal. We can then solve Equation (B.2) for $T_{3/2}$ and insert into Equation (B.1) (or vice versa). Since all vectors are parallel to the x-axis, we drop the vector notation and obtain

$$T_{3/2} = T_2 + q \frac{x_2 - x_{3/2}}{k_2} \tag{B.3}$$

from Equation (B.2). Inserting into Equation (B.1) yields

$$q = -k_1 \frac{T_2 + q\frac{x_2 - x_{3/2}}{k_2} - T_1}{x_{3/2} - x_1} = -k_1 \frac{T_2 - T_1}{x_{3/2} - x_1} - q \frac{k_1}{k_2} \frac{x_2 - x_{3/2}}{x_{3/2} - x_1}, \tag{B.4}$$

$$\implies q \left(1 + \frac{k_1}{k_2} \frac{x_2 - x_{3/2}}{x_{3/2} - x_1}\right) = -k_1 \frac{T_2 - T_1}{x_{3/2} - x_1}, \tag{B.5}$$

$$\implies q \left(k_2 \left(x_{3/2} - x_1\right) + k_1 \left(x_2 - x_{3/2}\right)\right) = k_1 k_2 \left(T_2 - T_1\right). \tag{B.6}$$

The boundary must necessarily lie between the two control volume centers, so we can write

$$x_{3/2} = \gamma x_1 + (1 - \gamma) x_2 \tag{B.7}$$

for some $\gamma \in (0,1)$. Inserting this expression yields

$$q\left(k_2\left(\gamma x_1 + (1-\gamma)x_2 - x_1\right) + k_1\left(x_2 - \gamma x_1 - (1-\gamma)x_2\right)\right) = k_1 k_2 (T_2 - T_1), \tag{B.8}$$

$$\implies q\left(k_2(1-\gamma)(x_2 - x_1) + k_1\gamma(x_2 - x_1)\right) = k_1 k_2 (T_2 - T_1), \tag{B.9}$$

$$\implies q = \frac{k_1 k_2}{\gamma k_1 + (1-\gamma)k_2}\frac{T_2 - T_1}{x_2 - x_1}. \tag{B.10}$$

Comparing the expression above to Fourier's law, we see that if we want to calculate the heat flux across the boundary based on the temperatures T_1 and T_2, we must use an "effective" conductivity k_{eff} given by

$$k_{\text{eff}} = \frac{k_1 k_2}{\gamma k_1 + (1-\gamma)k_2}, \tag{B.11}$$

which is a weighted harmonic average of k_1 and k_2. If the boundary is midway between the grid nodes, which is the case for all discretizations treated in this work, $\gamma = 1/2$. The effective conductivity is then given by the regular harmonic mean

$$k_{\text{eff}} = \frac{2k_1 k_2}{k_1 + k_2}. \tag{B.12}$$

Thus, the use of the harmonic mean in our finite-volume method (FVM) ensures that our numerical solutions are physical in the sense that no heat is generated at the boundary between two control volumes of different (but constant) conductivities. Note that, for a general discretization, we have no guarantees that the conductivity within any one control volume will be constant. However, if we only know the conductivity at the grid points, it is a reasonable *a priori* approximation to assume constant conductivity within each control volume.

C. Diagonal Dominance

We begin by proving that the matrix $\mathbb{A}$ in the system (3.58) is diagonally dominant. First, note that $|A_{j,j}| = A_{j,j}$, $|A_{j-1,j}| = -A_{j-1,j}$ and $|A_{j,j+1}| = -A_{j,j+1}$ for all j because α, Δt and Δx (with appropriate subscripts when applicable) are all strictly positive. For $j = 1$, we have

$$\begin{aligned}\sum_{i \neq 1} |A_{1,i}| = |A_{1,2}| = -A_{1,2} \qquad & \qquad (\text{C.1})\\ = \frac{2\alpha_2\alpha_1}{\alpha_2 + \alpha_1} \frac{\Delta t}{\Delta x_1, \Delta x_{3/2}} \qquad & \qquad (\text{C.2})\\ < 1 + \frac{2\alpha_2\alpha_1}{\alpha_2 + \alpha_1} \frac{\Delta t}{\Delta x_1, \Delta x_{3/2}} + \frac{2\alpha_1\alpha_a}{\alpha_1 + \alpha_a} \frac{\Delta t}{\Delta x_1, \Delta x_{1/2}} \qquad & \qquad (\text{C.3})\\ = A_{1,1} = |A_{1,1}|. \qquad & \qquad (\text{C.4})\end{aligned}$$

Similarly, for $j = N_j$, we have

$$\begin{aligned}\sum_{i \neq N_j} |A_{N_j,i}| = |A_{N_j,N_j-1}| = -A_{N_j,N_j-1} \qquad & \qquad (\text{C.5})\\ = \frac{2\alpha_{N_j}\alpha_{N_j-1}}{\alpha_{N_j} + \alpha_{N_j-1}} \frac{\Delta t}{\Delta x_{N_j}, \Delta x_{N_j-1/2}} \qquad & \qquad (\text{C.6})\\ < 1 + \frac{2\alpha_{N_j}\alpha_{N_j-1}}{\alpha_{N_j} + \alpha_{N_j-1}} \frac{\Delta t}{\Delta x_{N_j}, \Delta x_{N_j-1/2}} + \frac{2\alpha_{N_j}\alpha_b}{\alpha_{N_j} + \alpha_b} \frac{\Delta t}{\Delta x_{N_j}, \Delta x_b} \qquad & \qquad (\text{C.7})\\ = A_{N_j,N_j} = |A_{N_j,N_j}|. \qquad & \qquad (\text{C.8})\end{aligned}$$

Lastly, for $j = 2, \ldots, N_j - 1$, we have

$$\begin{aligned}\sum_{i \neq j} |A_{j,i}| = |A_{j-1,j}| + |A_{j,j+1}| = -A_{j-1,j} - A_{j,j+1} \qquad & \qquad (\text{C.9})\\ = \frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}\alpha_j} \frac{\Delta t}{\Delta x_j \Delta x_{j+1/2}} + \frac{2\alpha_j\alpha_{j-1}}{\alpha_j\alpha_{j-1}} \frac{\Delta t}{\Delta x_j \Delta x_{j-1/2}} \qquad & \qquad (\text{C.10})\\ < 1 + \frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}\alpha_j} \frac{\Delta t}{\Delta x_j \Delta x_{j+1/2}} + \frac{2\alpha_j\alpha_{j-1}}{\alpha_j\alpha_{j-1}} \frac{\Delta t}{\Delta x_j \Delta x_{j-1/2}} \qquad & \qquad (\text{C.11})\\ = A_{j,j} = |A_{j,j}|. \qquad & \qquad (\text{C.12})\end{aligned}$$

It has now been shown that $\sum_{i \neq j} |A_{j,i}| < |A_{j,j}| \; \forall j = 1, \ldots, N_j$. Thus, $\mathbb{A}$ of Equation (3.58) satisfies the requirements (3.74) and (3.75) for being diagonally dominant.

The proof of the diagonal dominance of the matrix $\mathbb{A}$ in Equation (3.67) is analogous to the proof above. In particular, note that $|A_{j,j}| = A_{j,j}$, $|A_{j-1,j}| = -A_{j-1,j}$ and $|A_{j,j+1}| = -A_{j,j+1}$ still hold for all j. For $j = 1$, we have

$$\sum_{i\neq 1} |A_{1,i}| = |A_{1,2}| = -A_{1,2} \tag{C.13}$$

$$= \frac{2\alpha_2\alpha_1}{\alpha_2+\alpha_1}\frac{1}{\Delta x_1, \Delta x_{3/2}} \tag{C.14}$$

$$< \frac{2\alpha_2\alpha_1}{\alpha_2+\alpha_1}\frac{1}{\Delta x_1, \Delta x_{3/2}} + \frac{2\alpha_1\alpha_a}{\alpha_1+\alpha_a}\frac{1}{\Delta x_1, \Delta x_{1/2}} \tag{C.15}$$

$$= A_{1,1} = |A_{1,1}|. \tag{C.16}$$

Similarly, for $j = N_j$, we have

$$\sum_{i\neq N_j} |A_{N_j,i}| = |A_{N_j,N_j-1}| = -A_{N_j,N_j-1} \tag{C.17}$$

$$= \frac{2\alpha_{N_j}\alpha_{N_j-1}}{\alpha_{N_j}+\alpha_{N_j-1}}\frac{1}{\Delta x_{N_j}, \Delta x_{N_j-1/2}} \tag{C.18}$$

$$< \frac{2\alpha_{N_j}\alpha_{N_j-1}}{\alpha_{N_j}+\alpha_{N_j-1}}\frac{1}{\Delta x_{N_j}, \Delta x_{N_j-1/2}} + \frac{2\alpha_{N_j}\alpha_b}{\alpha_{N_j}+\alpha_b}\frac{1}{\Delta x_{N_j}, \Delta x_b} \tag{C.19}$$

$$= A_{N_j,N_j} = |A_{N_j,N_j}|. \tag{C.20}$$

Lastly, for $j = 2,\dots,N_j - 1$, we have

$$\sum_{i\neq j} |A_{j,i}| = |A_{j-1,j}| + |A_{j,j+1}| = -A_{j-1,j} - A_{j,j+1} \tag{C.21}$$

$$= \frac{2\alpha_{j+1}\alpha_j}{\alpha_{j+1}\alpha_j}\frac{\Delta t}{\Delta x_j \Delta x_{j+1/2}} + \frac{2\alpha_j\alpha_{j-1}}{\alpha_j\alpha_{j-1}}\frac{\Delta t}{\Delta x_j \Delta x_{j-1/2}} \tag{C.22}$$

$$= A_{j,j} = |A_{j,j}|. \tag{C.23}$$

It has now been shown that $\sum_{i\neq j} |A_{j,i}| = |A_{j,j}|\ \forall j = 2,\dots,N_j - 1$, and that $\sum_{i\neq j} |A_{j,i}| = |A_{j,j}|$ for $j = 1, N_j$. Thus, $\mathbb{A}$ of Equation (3.67) satisfies the requirements (3.74) and (3.75) for being diagonally dominant.

D. Exact Solutions

In this section, we derive the exact solutions used in the grid refinement studies of Chapter 5. Throughout these derivations, it is understood that the numerical values of all physical parameters are stated using the standard SI units. For increased readability, the units are not explicitly included in the calculations shown.

D.1. Exact Solution for Grid Refinement Study #1

For Grid Refinement Study #1, we have $k = 1$, $A = 1$, $\hat{q} = 2$ and $T_a = T_b = 0$ (cf. Table 5.2). Inserting into Equation (3.41) yields

$$T(x) = T_a - \frac{\hat{q}}{2kA}(x - x_a)^2 + \left(\frac{kA}{x_b - x_a}(T_b - T_a) + \frac{\hat{q}}{2}(x_b - x_a)\right)\frac{1}{kA}(x - x_a) \tag{D.1}$$

$$= 0 - \frac{2}{2}(x - (-1))^2 + \left(0 + \frac{2}{2}\cdot(1 - (-1))\right)\frac{1}{1}\cdot(x - (-1)) \tag{D.2}$$

$$= -(x+1)^2 + 2(x+1) = -x^2 - 2x - 1 + 2x + 1 = 1 - x^2 \quad \blacksquare \tag{D.3}$$

D.2. Exact Solution for Grid Refinement Study #2

For Grid Refinement Study #2, we have $A = 1$ and $k = 5000x^2 - 55x + 0.25$, $x_a = 0$, $x_b = 0.01$, $T_a = 400$ and $T_b = 600$ (cf. Table 5.4). Inserting these parameters, we obtain

$$T(x) = T_a + \left(\int_{x_a}^{x_b} \frac{1}{kA}\mathrm{d}x\right)^{-1} (T_b - T_a) \int_{x_a}^{x} \frac{1}{kA}\mathrm{d}x \tag{D.4}$$

$$= 400 + \left(\int_{0}^{0.01} \frac{1}{5000x^2 - 55x + 0.25}\mathrm{d}x\right)^{-1} (600 - 400) \int_{0}^{x} \frac{1}{5000x^2 - 55x + 0.25}\mathrm{d}x. \tag{D.5}$$

It can be verified, e.g. through the use of the online mathematics tool WolframAlpha[1], that

$$\int \frac{1}{5000x^2 - 55x + 0.25}\mathrm{d}x = \frac{2\arctan\left(\frac{2000x - 11}{\sqrt{79}}\right)}{5\sqrt{79}} + C, \tag{D.6}$$

[1] `https://www.wolframalpha.com/input/?i=int%281%2F%285000x%5E2+-+55x+%2B+0.25%29%29dx+from+0+to+0.01`

where C is a constant of integration. Evaluating this indefinite integral with $C = 0$ at $x = 0$ and $x = 0.01$, and inserting into the expression for the exact solution given above, one obtains

$$T(x) = 400 + 200\frac{\arctan\left(\frac{2000x-11}{\sqrt{79}}\right) - \arctan\left(\frac{-11}{\sqrt{79}}\right)}{\arctan\left(\frac{9}{\sqrt{79}}\right) - \arctan\left(\frac{-11}{\sqrt{79}}\right)} \quad \blacksquare \tag{D.7}$$

D.3. Exact Solution for Grid Refinement Studies #3 and #4

With the notation used in this report, the second-to-last equation on page 70 of [35] reads

$$T(x, y, t) = \sum_{l=1}^{\infty}\sum_{m=1}^{\infty} d_{lm} \sin\left(\frac{l\pi x}{L_x}\right) \sin\left(\frac{m\pi y}{L_y}\right) \exp\left(-\left(\left(\frac{l\pi}{L_x}\right)^2 + \left(\frac{l\pi}{L_y}\right)^2\right)\alpha t\right), \tag{D.8}$$

where $\alpha = k/(\rho c_V)$ is the thermal diffusivity, L_x and L_y define the length of the domain in the x- and y-directions, respectively, and d_{lm} represent Fourier coefficients. We consider only heat conduction in the x-direction, so the equation above simplifies to

$$T(x, t) = \sum_{l=1}^{\infty} d_l \sin\left(\frac{l\pi x}{L_x}\right) \exp\left(-\left(\frac{l\pi}{L_x}\right)^2 \alpha t\right). \tag{D.9}$$

In Grid Refinement Studies #3 and #4, we use $x_a = 0$ (as is required by the derivation of (6.8) in [35]), and $x_b = 1$, such that $L_x = x_b - x_a = 1$. We then get

$$T(x, t) = \sum_{l=1}^{\infty} d_l \sin(l\pi x) \exp\left(-(l\pi)^2 \alpha t\right). \tag{D.10}$$

The derivation in [35] assumes homogeneous Dirichlet BCs, while we use the non-homogeneous conditions $T_a = T_b = 250$. To account for this, the exact solution must be shifted by 250 degrees such that it becomes

$$T(x, t) = 250 + \sum_{l=1}^{\infty} d_l \sin(l\pi x) \exp\left(-(l\pi)^2 \alpha t\right). \tag{D.11}$$

For $t = 0$, the exact solution is then $T(x, t = 0) = 250 + \sum_{l=1}^{\infty} d_l \sin(l\pi x)^2$. Comparing this with the IC of Grid Refinement Studies #3 and #4, which is $T(x, t = 0) = 250 + 50\sin(2\pi x)$ (cf. Tables 5.6 and 5.8), it is clear the we must have $d_2 = 50$ and $d_l = 0\ \forall l \neq 2$. The exact solution can then be written as

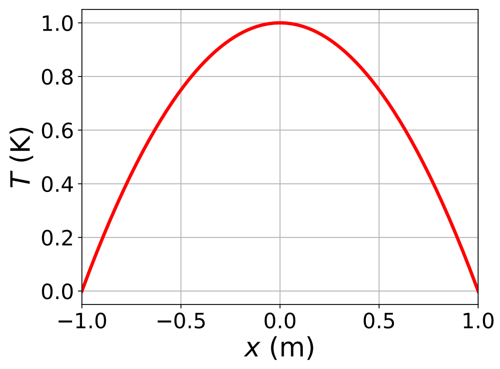
$$T(x, t) = 250 + 50\sin(2\pi x)\exp\left(-4\pi^2\alpha t\right) = 250 + 50\sin(2\pi x)\exp\left(-4\pi^2 kt/(\rho c_V)\right). \tag{D.12}$$

We have $t_{end} = 5$, $k = 2500$, $c_V = 2000$ and $\rho = 1300$. Inserting these parameter values into the equation above, we find that

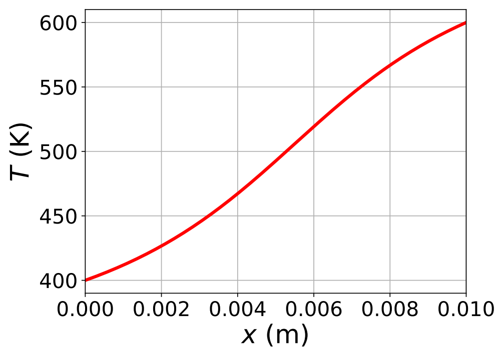
$$T(x, t = t_{end}) = 250 + 50\sin(2\pi x)\exp\{-\pi^2/52\} \quad \blacksquare \tag{D.13}$$

D.4. Illustrating the Exact Solutions

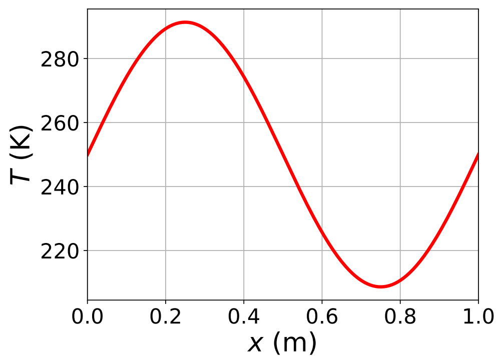
The exact solutions given by Equations (5.5), (5.6) and (5.7) (or, equivalently, by Equations (D.3), (D.7) and (D.13)) are illustrated in Figure D.1a, D.1b and D.1c, respectively.



(a) Equation (5.5)



(b) Equation (5.6)



(c) Equation (5.7)

Figure D.1.: An illustration of the exact solutions used in Grid Refinement Studies #1 (top row), #2 (middle row), and #3 and #4 (bottom row).

E. Reference Correction Source Term for Steady-State Problems

For the steady-state problem studied in Steady-State Experiments #1 and #2, the matrix $\mathbb{A}$ in Equation (4.8) is

$$\mathbb{A} = \frac{k_{\text{ref}}}{\rho c_V}\frac{1}{0.2}\begin{bmatrix} \frac{1}{0.2}+\frac{1}{0.1} & -\frac{1}{0.2} & 0 & 0 & 0 \\ -\frac{1}{0.2} & \frac{1}{0.2}+\frac{1}{0.2} & -\frac{1}{0.2} & 0 & 0 \\ 0 & -\frac{1}{0.2} & \frac{1}{0.2}+\frac{1}{0.2} & -\frac{1}{0.2} & 0 \\ 0 & 0 & -\frac{1}{0.2} & \frac{1}{0.2}+\frac{1}{0.2} & -\frac{1}{0.2} \\ 0 & 0 & 0 & -\frac{1}{0.2} & \frac{1}{0.1}+\frac{1}{0.2} \end{bmatrix} \tag{E.1}$$

$$= 5\frac{k_{\text{ref}}}{\rho c_V}\begin{bmatrix} 15 & -5 & 0 & 0 & 0 \\ -5 & 10 & -5 & 0 & 0 \\ 0 & -5 & 10 & -5 & 0 \\ 0 & 0 & -5 & 10 & -5 \\ 0 & 0 & 0 & -5 & 15 \end{bmatrix} \tag{E.2}$$

$$= 25\frac{k_{\text{ref}}}{\rho c_V}\begin{bmatrix} 3 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 3 \end{bmatrix}. \tag{E.3}$$

Furthermore, the vector $\boldsymbol{b}$ in the same equation is given by

$$\boldsymbol{b} = \frac{k_{\text{ref}}}{\rho c_V}\begin{bmatrix} \frac{1}{0.1\cdot 0.2}T_a \\ 0 \\ 0 \\ 0 \\ \frac{1}{0.1\cdot 0.2}T_b \end{bmatrix} = 25\frac{k_{\text{ref}}}{\rho c_V}\begin{bmatrix} 2T_a \\ 0 \\ 0 \\ 0 \\ 2T_b \end{bmatrix}. \tag{E.4}$$

Inserting into Equation (4.8) then yields

$$\begin{aligned} \hat{\boldsymbol{\sigma}}_{\text{ref}} &= \mathbb{A}\boldsymbol{T}^{\infty}_{\text{ref}} - \boldsymbol{b} \\ &= 25\frac{k_{\text{ref}}}{\rho c_V}\left(\begin{bmatrix} 3 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 3 \end{bmatrix}\begin{bmatrix} T^{\infty}_{\text{ref},1} \\ T^{\infty}_{\text{ref},2} \\ T^{\infty}_{\text{ref},3} \\ T^{\infty}_{\text{ref},4} \\ T^{\infty}_{\text{ref},5} \end{bmatrix} - \begin{bmatrix} 2T_a \\ 0 \\ 0 \\ 0 \\ 2T_b \end{bmatrix}\right) \end{aligned} \tag{E.5}$$

This means that the components of $\hat{\boldsymbol{\sigma}}$ are given by

$$\begin{aligned}
\hat{\sigma}_{\text{ref},1} &= 25\frac{k_{\text{ref}}}{\rho c_V}\left(3T^{\infty}_{\text{ref},1} - T^{\infty}_{\text{ref},2} - 2T_a\right)\\
&= 25\frac{k_{\text{ref}}}{\rho c_V}\Bigg(3\left(T_a + \left(\int_{x_a}^{x_b}\frac{1}{kA}\mathrm{d}x\right)^{-1}(T_b - T_a)\int_{x_a}^{0.1}\frac{1}{kA}\mathrm{d}x\right)\\
&- \left(T_a + \left(\int_{x_a}^{x_b}\frac{1}{kA}\mathrm{d}x\right)^{-1}(T_b - T_a)\int_{x_a}^{0.3}\frac{1}{kA}\mathrm{d}x\right) - 2T_a\Bigg)\\
&= 25\frac{k_{\text{ref}}}{\rho c_V}(T_b - T_a)\left(\int_{x_a}^{x_b}\frac{1}{kA}\mathrm{d}x\right)^{-1}\left(3\int_{x_a}^{0.1}\frac{1}{kA}\mathrm{d}x - \int_{x_a}^{0.3}\frac{1}{kA}\mathrm{d}x\right),
\end{aligned}$$

$$\begin{aligned}
\hat{\sigma}_{\text{ref},2} &= 25\frac{k_{\text{ref}}}{\rho c_V}\left(-T^{\infty}_{\text{ref},1} + 2T^{\infty}_{\text{ref},2} - T^{\infty}_{\text{ref},3}\right)\\
&= 25\frac{k_{\text{ref}}}{\rho c_V}\Bigg(-\left(T_a + \left(\int_{x_a}^{x_b}\frac{1}{kA}\mathrm{d}x\right)^{-1}(T_b - T_a)\int_{x_a}^{0.1}\frac{1}{kA}\mathrm{d}x\right)\\
&+ 2\left(T_a + \left(\int_{x_a}^{x_b}\frac{1}{kA}\mathrm{d}x\right)^{-1}(T_b - T_a)\int_{x_a}^{0.3}\frac{1}{kA}\mathrm{d}x\right)\\
&- \left(T_a + \left(\int_{x_a}^{x_b}\frac{1}{kA}\mathrm{d}x\right)^{-1}(T_b - T_a)\int_{x_a}^{0.5}\frac{1}{kA}\mathrm{d}x\right)\Bigg)\\
&= 25\frac{k_{\text{ref}}}{\rho c_V}(T_b - T_a)\left(\int_{x_a}^{x_b}\frac{1}{kA}\mathrm{d}x\right)^{-1}\left(-\int_{x_a}^{0.1}\frac{1}{kA}\mathrm{d}x + 2\int_{x_a}^{0.3}\frac{1}{kA}\mathrm{d}x - \int_{x_a}^{0.5}\frac{1}{kA}\mathrm{d}x\right),
\end{aligned}$$

and so on. Thus $\hat{\boldsymbol{\sigma}}_{\text{ref}} = \boldsymbol{C}(T_b - T_a)$, where $\boldsymbol{C}$ is a vector given by

$$\boldsymbol{C} = 25\frac{k_{\text{ref}}}{\rho c_V}\left(\int_{x_a}^{x_b}\frac{1}{kA}\mathrm{d}x\right)^{-1}\begin{bmatrix} 3\int_{x_a}^{0.1}\frac{1}{kA}\mathrm{d}x - \int_{x_a}^{0.3}\frac{1}{kA}\mathrm{d}x \\ -\int_{x_a}^{0.1}\frac{1}{kA}\mathrm{d}x + 2\int_{x_a}^{0.3}\frac{1}{kA}\mathrm{d}x - \int_{x_a}^{0.5}\frac{1}{kA}\mathrm{d}x \\ -\int_{x_a}^{0.3}\frac{1}{kA}\mathrm{d}x + 2\int_{x_a}^{0.5}\frac{1}{kA}\mathrm{d}x - \int_{x_a}^{0.7}\frac{1}{kA}\mathrm{d}x \\ -\int_{x_a}^{0.5}\frac{1}{kA}\mathrm{d}x + 2\int_{x_a}^{0.7}\frac{1}{kA}\mathrm{d}x - \int_{x_a}^{0.9}\frac{1}{kA}\mathrm{d}x \\ -\int_{x_a}^{0.7}\frac{1}{kA}\mathrm{d}x + 3\int_{x_a}^{0.9}\frac{1}{kA}\mathrm{d}x \end{bmatrix}. \tag{E.6}$$

F. Raw Data from Machine Learning Experiments

Bold highlighting indicates data of the model instances used as so-called representative instances in the main text.

F.1. Raw Data from Steady-State Experiment #1

Table F.1.: Steady-State Experiment #1: Test results for end-to-end learning approach.

Instance #	Average test loss	Average L_2 error
1	6.6609e-4	2.3109e-3
2	6.3190e-4	2.3191e-3
3	5.7479e-4	2.3311e-3
4	6.5412e-4	2.3165e-3
5	6.8386e-4	2.3045e-3
6	8.1686e-4	2.3303e-3
7	7.6999e-4	2.3309e-3
8	**6.0953e-4**	**2.3240e-3**
Average	6.7589e-4	2.3209e-3
Std dev	8.0995e-5	9.9620e-6

Table F.2.: Steady-State Experiment #1: Test results for hybrid modelling approach.

Instance #	Average test loss	Average L_2 error
1	2.6476e-4	2.3237e-3
2	2.7519e-4	2.3251e-3
3	1.8874e-4	2.3227e-3
4	2.7332e-4	2.3190e-3
5	2.1783e-4	2.3240e-3
6	**2.4982e-4**	**2.3418e-3**
7	2.2257e-4	2.3286e-3
8	2.1462e-4	2.3233e-3
Average	2.3836e-4	2.3260e-3
Std dev	3.1857e-5	6.9092e-6

F.2. Raw Data from Steady-State Experiment #2

Table F.3.: Steady-State Experiment #2: Test results for end-to-end learning approach.

Instance #	Average test loss	Average L_2 error
1	3.3477e1	3.0606e-2
2	6.5703e0	1.2822e-2
3	5.1562e1	3.9575e-2
4	1.1418e1	1.6624e-2
5	6.3790e1	4.3941e-2
6	2.1844e1	2.3136e-2
7	**2.2235e1**	**2.4692e-2**
8	1.2203e1	1.8214e-2
Average	2.7887e1	2.6201e-2
Std dev	2.0427e1	1.1079e-2

Table F.4.: Steady-State Experiment #2: Test results for hybrid modelling approach.

Instance #	Average test loss	Average L_2 error
1	**2.8587e0**	**1.3771e-2**
2	6.3446e0	8.1494e-3
3	4.1847e0	8.3321e-3
4	8.6277e0	9.2793e-3
5	2.0703e0	2.2446e-2
6	9.4935e0	1.8870e-2
7	1.1355e1	1.8384e-2
8	4.7783e0	2.3351e-2
Average	6.2141e0	1.5323e-2
Std dev	3.3304e0	6.2838e-3

F.3. Raw Data from Steady-State Experiment #3

Table F.5.: Steady-State Experiment #3: Test results for end-to-end learning approach.

Instance #	Average test loss	Average L_2 error
1	3.3422e0	4.9748e-2
2	2.5483e0	4.4585e-2
3	6.3123e0	6.7982e-2
4	9.9489e-1	2.9747e-2
5	8.7906e-1	2.4248e-2
6	**2.5415e0**	**4.4706e-2**
7	3.0787e0	4.9385e-2
8	3.9983e0	5.5922e-2
Average	2.9619e0	4.5790e-2
Std dev	1.7326e0	1.3876e-2

Table F.6.: Steady-State Experiment #3: Test results for hybrid modelling approach.

Instance #	Average test loss	Average L_2 error
1	**2.8710e-1**	**2.4958e-2**
2	3.7288e-1	1.9463e-2
3	1.5248e-1	1.2341e-2
4	1.5123e-1	1.3569e-2
5	3.4274e-1	2.7055e-2
6	2.2302e-1	2.5299e-2
7	4.2051e-1	2.8370e-2
8	5.6000e-1	4.3821e-2
Average	3.1374e-1	2.4359e-2
Std dev	1.4027e-1	9.9153e-3

F.4. Raw Data from Unsteady Experiment #1

Table F.7.: Unsteady Experiment #1: Test results for end-to-end learning approach.

Instance #	Average test loss	L_2 error, t=1000
1	7.4148e0	3.8458e-3
2	1.0518e1	4.5905e-3
3	5.6971e0	3.3540e-3
4	1.0667e1	4.6826e-3
5	**8.7722e0**	**4.1659e-3**
6	9.9020e0	4.4572e-3
7	7.4032e0	3.8410e-3
8	9.6767e0	4.3942e-3
Average	8.7564e0	4.1664e-3
Std dev	1.7689e0	4.5534e-4

Table F.8.: Unsteady Experiment #1: Test results for hybrid modelling approach.

Instance #	Average test loss	Average L_2 error, t=1000
1	1.3122e1	2.9485e-5
2	9.3979e0	2.4085e-5
3	6.8353e0	2.0903e-5
4	8.1848e0	2.3645e-5
5	5.5078e0	1.9005e-5
6	**7.4677e0**	**2.1814e-5**
7	6.3153e0	1.9867e-5
8	4.8741e0	1.7750e-5
Average	7.7131e0	2.2069e-5
Std dev	2.6193e0	3.7008e-6

F.5. Raw Data from Unsteady Experiment #2

Table F.9.: Unsteady Experiment #2: Test results for end-to-end learning approach.

Instance #	Average L_2 error, t=100	Average L_2 error, t=1000
1	**6.5972e-2**	**3.8458e-1**
2	3.1357e-2	4.5905e-2
3	1.8771e-1	3.3540e-1
4	1.3384e-1	4.6826e-1
5	8.2811e-3	4.1659e-2
6	3.5954e-3	4.4572e-2
7	1.0569e-1	Overflow
8	2.1583e-2	9.7359e-2
Average	6.9753e-2	1.6944e-1
Std dev	6.6843e-2	2.4271e-1

Table F.10.: Unsteady Experiment #2: Test results for hybrid modelling approach.

Instance #	Average L_2 error, t=100	Average L_2 error, t=1000
1	1.8205e-2	5.9986e-2
2	2.1928e-2	9.0844e-2
3	2.2840e-2	8.3398e-2
4	**2.4855e-2**	**9.5602e-2**
5	1.5343e-2	6.7136e-2
6	3.1088e-2	9.9313e-2
7	2.1829e-2	7.3255e-2
8	4.1441e-2	1.7020e-1
Average	2.4691e-2	9.2467e-2
Std dev	8.1959e-3	3.4330e-2

F.6. Raw Data from Unsteady Experiment #3

Table F.11.: Unsteady Experiment #3: Test results for end-to-end learning approach.

Instance #	Average L_2 error, t=5	Average L_2 error, t=1000
1	2.2067e-2	Overflow
2	**2.7416e-2**	**3.8363e74**
3	2.1510e-1	Overflow
4	1.1077e-2	Overflow
5	9.2262e-2	Overflow
6	6.1146e-2	Overflow
7	6.8400e-2	Overflow
8	9.0528e-2	1.2615e106
Average	7.3499e-2	6.3077e105
Std dev	6.4957e-2	8.9204e105

Table F.12.: Unsteady Experiment #3: Test results for hybrid modelling approach.

Instance #	Average L_2 error, t=5	Average L_2 error, t=1000
1	**5.9487e-4**	**1.6841e-2**
2	6.9593e-4	1.5170e-2
3	8.9133e-4	1.7658e-2
4	5.1759e-4	1.3579e-2
5	9.1190e-4	2.4772e-2
6	4.0407e-4	8.5557e-3
7	5.9173e-4	9.8432e-3
8	5.3488e-4	1.3011e-2
Average	6.4279e-4	1.4929e-2
Std dev	1.7977e-4	5.0715e-3

F.7. Raw Data from Unsteady Experiment #4

Table F.13.: Unsteady Experiment #4: Test results for end-to-end learning approach.

Instance #	Average L_2 error, t=20	Average L_2 error, t=1000
1	2.5145e-2	5.5089e-2
2	3.8753e-3	3.4060e-2
3	3.0285e-3	1.3939e-1
4	3.1000e-2	1.0481e-1
5	**1.9134e-2**	**6.3999e-2**
6	1.4010e-2	9.6175e-2
7	1.1071e-2	3.5794e-2
8	6.2395e-3	4.2063e-2
Average	1.4188e-2	7.1423e-2
Std dev	1.0219e-2	3.8166e-2

Table F.14.: Unsteady Experiment #4: Test results for hybrid modelling approach.

Instance #	Average L_2 error, t=20	Average L_2 error, t=1000
1	1.3122e-1	2.9485e0
2	**9.3979e-2**	**2.4085e0**
3	6.8353e-1	2.0903e0
4	8.1848e-1	2.3645e0
5	5.5078e-2	1.9005e0
6	7.4677e-1	2.1814e1
7	6.3153e-1	1.9867e0
8	4.8741e-1	1.7750e0
Average	7.7131e-1	2.2069e0
Std dev	2.6193e-1	3.7008e0

G. Computational Complexity of Unified Models

The Tri-Diagonal Matrix Algorithm uses roughly $8N_j$ floating point operations (FLOPs) to solve the system (3.58) or the system (3.67) [8]. Suppose now that the end-to-end learning approach is used to correct the numerical solution given by one of the two systems of equations. Suppose further that a generalized version of the architecture illustrated in Figure 6.2 is used. This generalized architecture has N_{hidden} hidden layers (followed by LeakyReLU activation and dropout), each consisting of N_{neurons} neurons. We assume that the network has finished training, such that dropout is disabled.

Calculating the output of the first hidden layer of the network requires $N_{\text{neurons}}(N_j+2)$ multiplications (for multiplying input with weights), $N_{\text{neurons}}(N_j+1)$ additions (for adding the weight-input-products) and another N_{neurons} additions (for adding the biases). In total, this amounts to $2N_{\text{neurons}}(N_j+2)$ FLOPs. The LeakyReLU activation requires a further $4N_{\text{neurons}}$ FLOPs[1]. This brings the total number of FLOPs so far to $N_{\text{neurons}}(2N_j+8)$. Similarly, the number of FLOPs for the output layer is $N_{\text{neurons}}(2N_j)$, where the reduction in the number of FLOPs is caused by the reduced number of neurons in the output layer compared to the input layer and the lack of activation function for the output layer. Lastly, each hidden layer requires a total of $N_{\text{neurons}}(2N_{\text{neurons}}+4)$ FLOPs. The entire forward pass through the network therefore then requires

$$\begin{aligned} &N_{\text{neurons}}(2N_j+8) + N_{\text{neurons}}(2N_j) + N_{\text{hidden}} \cdot N_{\text{neurons}}(2N_{\text{neurons}}+4) \\ &= N_{\text{neurons}}(4N_j + 8 + N_{\text{hidden}}(2N_{\text{neurons}}+4)) \end{aligned}$$

Thus, considering also the FLOPs required to solve the discretized governing equations, one iteration of a unified model using end-to-end learning requires

$$\text{FLOPs}_{\text{end-to-end}} = 8N_j + N_{\text{neurons}}(4N_j + 8 + N_{\text{hidden}}(2N_{\text{neurons}}+4)) \tag{G.1}$$

floating point operations.

For hybrid modelling, the output of the neural network must be added to the vector on the right hand side of the system of equations. This requires N_j additions. Furthermore, the system must be solved a second time, which requires another $8N_j$ FLOPs. The total number of FLOPs for one iteration of a unified model based on hybrid modelling therefore requires the following number of FLOPs:

$$\begin{aligned} \text{FLOPs}_{\text{hybrid}} &= \text{FLOPs}_{\text{end-to-end}} + N_j + 8N_j \\ &= 17N_j + N_{\text{neurons}}(4N_j + 8 + N_{\text{hidden}}(2N_{\text{neurons}}+4)). \end{aligned} \tag{G.2}$$

[1]This can be seen from the top equation on this page: `https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html`