TECHNISCHE
UNIVERSITÄT
DARMSTADT

Simulation and Optimization of Gas Transport Problems using Physics-Informed Neural Networks

**Vom Fachbereich Mathematik
der Technischen Universität Darmstadt
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)
genehmigte Dissertation**

von

Erik Laurin Strelow, M. Sc.

aus Flörsheim am Main

Referent:	Prof. Dr. Jens Lang
Korreferent:	Prof. Dr. Jan Giesselmann
Tag der Einreichung:	18. Oktober 2023
Tag der mündlichen Prüfung:	4. Dezember 2023

Darmstadt 2023
D17

Simulation and Optimization of Gas Transport Problems using Physics-Informed Neural Networks

Accepted doctoral thesis by Erik Laurin Strelow, M.Sc.

Darmstadt, Technische Universität Darmstadt

Date of thesis defense: December 4, 2023
Tag der mündlichen Prüfung: 4. Dezember 2023

Year of publication of the doctoral thesis on tuprints: 2024
Jahr der Veröffentlichung der Dissertation auf tuprints: 2024

Please cite this document as / Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-267694
URL: https://tuprints.ulb.tu-darmstadt.de/26769

This document is provided by tuprints, e-publishing service of TU Darmstadt / Dieses Dokument wird bereitgestellt von tuprints, E-Publishing-Service der TU Darmstadt
http://tuprints.ulb.tu-darmstadt.de
tuprints@ulb.tu-darmstadt.de

Zusammenfassung

Diese Arbeit befasst sich tiefgehend mit physikalisch informierten (*physics-informed*) neuronalen Netzwerken. Die Methode baut auf Deep Learning Techniken auf, wird im Allgemeinen zur Lösung von Differentialgleichungen verwendet und in dieser Arbeit auf Gastransportprobleme angewendet. Die vorliegende Bearbeitung dieses bislang wenig erforschten Themenfeldes leistet einen Beitrag zu einem besseren Verständnis der Methode und richtet seinen Fokus auf drei Schwerpunkte:

Zunächst beschäftigen wir uns mit den grundlegenden Eigenschaften. Hier beweisen wir einerseits Fehlerabschätzungen für ein lineares System von Transportgleichungen, die den Fehler der Approximation durch den Wert der Verlustfunktion beschränken. Die Abschätzungen zeigen allerdings auch, dass die Methode Probleme mit langen Zeitintervallen und hohen charakteristischen Geschwindigkeiten hat. Zum anderen wird aufgezeigt, wie die Methode effizient implementiert werden kann.

Neben der Standardmethode wurden zahlreiche Variationen von physikalisch informierten neuronalen Netzwerken entwickelt. Diese vergleichen wir miteinander und befassen uns mit dem zweiten Schwerpunkt, der Suche nach der effektivsten Trainingsstrategie. Hier führen wir umfangreiche numerische Tests durch, die verschiedene Architekturen von neuronalen Netzwerken, Optimierungsverfahren, Integrationsverfahren und Verfahren zur Wahl von Gewichten in der Verlustfunktion beinhalten.

In den Ergebnissen zeigt sich, dass für jedes Problem eine andere neuronale Netzwerk Architektur am besten geeignet ist. Als Optimierungsverfahren ist das Adam Verfahren mit der richtigen Lernrate und genügend Iterationen den anderen Verfahren überlegen. Bei den verschiedenen Integrationsverfahren lassen sich kaum Unterschiede erkennen und die verschiedenen Konvergenzraten der Verfahren übertragen sich nicht auf die Konvergenzrate von physikalisch informierten neuronalen Netzwerken. Dies zeigt ein kompliziertes Verhältnis zwischen Optimierungs- und Quadraturfehler und, dass der Fehler nicht beliebig reduziert werden kann.

Bei den Verfahren zur Wahl von Gewichten in der Verlustfunktion kann nur die aufwendige Zufallssuche die Genauigkeit verbessern. Abschließend betrachten und erweitern wir eine andere Formulierung der Verlustfunktion. Durch die Erweiterung lässt sich deren Genauigkeit steigern, aber die der ursprünglichen Verlustfunktion nicht übertreffen.

Als Drittes befassen wir uns mit der Lösung von optimalen Steuerungsproblemen durch physikalisch informierte neuronale Netzwerke. Hier betrachten wir einen direkten Ansatz und entwickeln einen neuen, indirekten Ansatz. Wir zeigen, dass der direkte Ansatz das optimale Steuerungsproblem nicht ausreichend widerspiegelt und unzulässige Lösungen berechnet. Der indirekte und auf der Adjungierten aufbauende Ansatz berechnet wiederum zulässige Lösungen, die auch die Zielfunktion minimieren. Wir verdeutlichen dies durch numerische Ergebnisse von zwei Testproblemen.

Abstract

This thesis investigates the application of physics-informed neural networks to solve gas transport problems. Physics-informed neural networks are a new numerical method that applies deep learning techniques to solve problems involving differential equations. Much knowledge is still to be developed, and this thesis is a contribution to the understanding of the method, focusing on three main areas.

First, we contribute to the fundamental knowledge of physics-informed neural networks. Here, in a theoretical investigation, we prove error estimates for a linear system of transport equations that bound the generalization error of the method by the values of the loss function. The estimates thus validate the method. However, they also show that the method has problems with long time intervals and high characteristic speeds. Furthermore, a practical investigation shows that the standard implementation can be improved with a more efficient approach.

Second, we focus on the most effective training strategy to obtain physics-informed neural networks. Besides the standard method, many variants of physics-informed neural networks have been proposed. We survey these variants and perform extensive numerical tests involving different neural architectures, optimization methods, sampling strategies, and loss balancing methods.

The results show that a specific neural network architecture is best suited for each problem. The Adam optimization method outperforms the other optimization methods with an appropriate learning rate and sufficient iterations. Higher-order sampling strategies have no significant advantages over the commonly used Latin hypercube sampling, and thus the convergence rates of sampling strategies do not affect the convergence rate of physics-informed neural networks. This demonstrates a complex interaction between the optimization and quadrature error, and also shows that the generalization error cannot be arbitrarily reduced in practical applications.

For the loss balancing methods, only an expensive random search can improve the accuracy. Finally, we review and extend another loss function formulation. While the extension increases the accuracy, it does not exceed the accuracy of the original loss function.

Third, we address optimal control problems using physics-informed neural networks. Here, we consider a direct approach and develop a new indirect approach. We show that the direct approach does not adequately reflect the optimal control problem and computes infeasible solutions. In contrast, the indirect adjoint-based approach, computes feasible solutions that also minimize the objective function. We illustrate this with numerical tests from two test problems.

Acknowledgements

First of all, I would like to thank my supervisor, Professor Jens Lang, for giving me the opportunity to explore this new and exciting topic. I am very grateful for his patient guidance and experienced support over the years. I would also like to thank Professor Marc Pfetsch for being my co-supervisor.

Next, I would like to thank the co-referee Professor Jan Giesselmann and the members of the examination board Professor Yann Disser, Professor Oliver Weeger.

I especially thank Alf Gerisch for his technical support and proofreading the thesis. In addition, I also thank Selina and Kenan for proofreading the thesis.

I would like to thank my family and friends for their consistent support. Also, a special thanks to the Numerical Analysis and Scientific Computing Research Group for their helpfulness and the very friendly atmosphere.

This work was supported by the German Research Foundation within the collaborative research center Transregio 154 as well as the Graduate School CE within the Centre for Computational Engineering at TU Darmstadt. I gratefully acknowledge the computing time provided on the high-performance computer Lichtenberg at the NHR Centers NHR4CES at the TU Darmstadt.

Contents

1. Introduction **1**

2. Two exemplary problems **5**
2.1. Linear problem . . . 5
2.1.1. Exact solution . . . 6
2.2. Nonlinear problem . . . 7

3. Fundamentals of deep learning **11**
3.1. Designing neural networks . . . 11
3.1.1. Fully connected deep neural networks . . . 12
3.1.2. Hamiltonian-inspired neural networks. . . . 13
3.2. Training neural networks . . . 15
3.2.1. Adam Optimizer . . . 16
3.2.2. L-BFGS . . . 18
3.2.3. Initialization . . . 18
3.3. Automatic differentiation . . . 20
3.4. Implementation . . . 22
3.5. Numerical tests . . . 23

4. Physics-informed simulations **28**
4.1. Loss based on differential form . . . 29
4.1.1. Physics-informed neural networks . . . 29
4.1.2. Sampling strategies . . . 50
4.1.3. Loss balancing weighting . . . 58
4.2. Loss based on integral form . . . 64
4.2.1. Numerical Results . . . 67
4.3. Conclusion . . . 68

5. Physics-informed optimization **71**
5.1. Optimal control problems . . . 71
5.2. Direct approach . . . 73
5.2.1. Numerical tests . . . 74
5.3. Indirect approach . . . 76
5.3.1. Adjoint-based optimality conditions . . . 76
5.3.2. Physics-informed approach . . . 81
5.3.3. Numerical results . . . 83
5.4. Conclusion . . . 89

6. Conclusion **90**

Bibliography **92**

1. Introduction

Just in the last three years, the world has experienced multiple crises: a global pandemic, a war in Ukraine, an energy crisis, inflation, and a worsening climate crisis. These crises are not independent events, but rather a polycrisis [57]: An interplay between different crises that interact and amplify each other. To overcome these ever-growing challenges, scientific research is important on which informed mitigation strategies can be developed.

In this work, we want to contribute to the aspect of methods to reduce energy consumption, relieve the energy scarcity, which helps to mitigate both the energy and climate crises. We will address this task with another emerging technology, machine learning. Machine learning comprises algorithmic approaches to learn and improve from data without being explicitly programmed for specific tasks. These approaches have made enormous advancements in the last decade, suggesting that they are capable of solving a wide variety of problems and eventually reaching human intelligence.

In the following, we will explore both aspects in more detail: the energy crisis Germany is facing and the impact of artificial intelligence. Subsequently, we will describe how we combine both topics and outline the contributions of this work.

Germany's energy crisis

Final act. We begin on the morning of September 26, 2022. The news shows images of natural gas rising from the Baltic Sea, coming from the damaged Nord Stream gas pipelines that supply Germany with natural gas. What might appear to be a major environmental disaster and a waste of natural gas that Germany desperately needed at the time, was the final act of the much proclaimed German natural gas (or, more broadly, energy) strategy.

The damaged Nord Stream pipelines resolved Germany's ethical dilemma of supporting a war through gas purchases on the one hand, and a feared economic and social decline on the other hand. However, this situation has been decided by a yet unknown party by sabotaging the Nord Stream pipeline and jeopardizing Germany's future.

The past strategy. But what happened first? Compared to other carbon-based energy sources, natural gas emits significantly less carbon dioxide for the same amount of energy. As a result, replacing old coal-fired power plants or oil-fired heaters with natural gas-powered alternatives reduces carbon emissions. In addition, gas-fired power plants complement renewable energy sources by generating electricity during periods when wind or solar power is not available [56]. From this perspective, natural gas was viewed as a bridge enabling a smoother energy transition.

Crucially, there are also other sectors that benefit from cheap natural gas. For example, the chemical industry and energy-intensive sectors such as the steel industry have

maintained their global competitiveness by relying on cheap natural gas. Thus, the Nord Stream pipelines connected two countries with aligned interests: one that has abundant natural resources and another that wants them to fuel economic growth. Importantly, the gas was largely purchased at fixed prices, independent of the world market. In this arrangement, however, Germany sacrificed its energy sovereignty by becoming overly reliant on a single nation.

A new reality. Without the Nord Stream pipelines, Germany needs to purchase natural gas from neighboring countries and from the global liquefied natural gas (LNG) market. Germany now has to follow market prices and has a huge and sudden demand. This led to a cascade of unfortunate consequences: reduced gas availability in the LNG market, causing shortages in less developed countries, or rising electricity costs as the latter are linked to gas prices through gas-fired power plants.

As a result, Germany has to save natural gas because there is no longer enough supply. However, saving has consequences too. For instance, it harms industrial production and pushes Germany into a minor technical recession. So far, major damage has been averted. But the crisis is not over yet. Coal-fired power plants also burned more to compensate for the lack of gas, exacerbating carbon emissions. Lastly, greenhouses, which typically use gas to heat during the winter, saved gas and produced fewer vegetables. This has led to a reduction in supply and an increase in the cost of living.

Challenges ahead. Germany's energy strategy is facing major challenges and needs a new direction. Renewable energy is the key to an energy-independent and climate-neutral future. There are also plans to replace natural gas consumption with hydrogen [55]. However, the concrete details remain unclear, especially considering the inefficiencies in its production process and the associated demand for green energy.

While the immediate and distant futures are different, they share the same fundamental challenges. Currently, renewable energy resources are not enough to satisfy the energy demand, therefore carbon energy resources will be required in the coming years. However, natural gas is not as cheap and plentiful as it used to be. In a future without carbon energy sources, energy will be produced by renewable energy sources, which are not as plentiful as the carbon resources of the past. Thus, efficient use of energy is important now and will be in the future.

The efficient (or mathematically optimal) use of energy is a major concern of this work. Mathematical methods can be used to simulate and optimize energy systems, and thus help in both time frames. While our primary focus revolves around natural gas and hydrogen, the underlying principles are broadly applicable. These principles are based on machine learning techniques.

Artificial intelligence

Omnipresence and Perception. It has long been a goal to develop artificial intelligence (AI) methods that can match the human capabilities [16, p. 1]. Over the past decade, advances in hardware, machine learning software, and methods have enabled

impressive progress: Image recognition methods are available on every smartphone, superhuman performance at the board game Go [51] or the prediction of protein structure with AlphaFold [24]. Furthermore, the broader public has recently become more engaged with machine learning tools such as the StableDiffusion image generation and conversational models such as ChatGPT. These are leading examples of research that transitioned into widely recognized and used technologies.

Since these tools are very accessible, the public's perception of what counts as artificial intelligence is actually the deciding factor. In fact, platforms like ChatGPT are very much tailored to maximize this perception. Recent successes make it difficult to distinguish between human and machine intelligence, suggesting that the overall search for artificial intelligence is reaching its final goal. However, in 1950, Alan Turing developed the Turing test to assess exactly this question. So far, no algorithm has successfully pretended to be a human and passed the test.

An AI Revolution? It is very difficult to predict what machine learning will eventually be capable of. Building on recent successes, reaching human intelligence seems possible to some, and tougher challenges like self-driving cars or humanoid robots in factories and schools are on the horizon. This might replace many jobs with AI, potentially sparking a crisis or another technological revolution [4]. However, such a shift would favor those with vast computing resources and data, making it very unequal.

From another perspective, machine learning is just a complementary technology that enables algorithmic solutions to some previously unsolvable problems. This debate can only be resolved by testing machine learning approaches on various tasks and assessing their performance. This leads to the main goal of this thesis: We aim to identify the advantages and disadvantages of machine learning for our specific scenario.

Computational view

By approximating underlying physical systems, numerical methods allow the simulation of future states and optimization for energy efficiency. These physical systems are typically modeled by differential equations. In this work, we consider one specific class: Hyperbolic balance laws in a one-dimensional pipe $[x_{\mathrm{L}}, x_{\mathrm{R}}]$ over time $[t_{\mathrm{I}}, t_{\mathrm{E}}]$ that are formalized by

$$\partial_t u + \partial_x (F \circ u) = g(u) \quad \text{in} \quad (x_{\mathrm{L}}, x_{\mathrm{R}}) \times (t_{\mathrm{I}}, t_{\mathrm{E}}) \,, \tag{1.1}$$

for a state vector $u(x,t)$, a flux function $F(u)$ and a source term $g(u)$. Hyperbolic balance laws are used to model gas flow of natural gas and hydrogen, and equation (1.1) is the building block for more complex systems such as gas networks.

Classical numerical methods. Much research has focused towards the simulation and optimization of real-world gas networks. The key to successfully solve optimization problems is the simulation problem, which provides the solution and sensitivity information to the optimization method. However, the underlying transport structure of the solution requires specialized methods. In particular, simulation methods have been developed that are adaptive in space, time, and the applied gas transport model [44]. Using simplified models whenever possible can significantly reduce the simulation time.

Reduced methods, which are tailored to specific problems, can further reduce computational costs. However, as a prominent example, reduced basis methods, which are often successfully applied to different types of differential equations, have fundamental problems with differential equations that model (gas) transport. See this overview of the current state [5]. Further improvements are needed to handle large gas networks and more complex scenarios such as optimization.

Machine learning based methods. Machine learning methods that have proven effective in other tasks have recently been applied to solve differential equations. The methods are grouped under the term *physics-informed* machine learning, as highlighted in [25], and incorporate physical laws into the training process. The first method are *physics-informed neural networks* (PINNs), which were introduced in [36] and quickly gained popularity. The main advantage is the flexibility of the technique, which allows specific tuning to a variety of differential equations. This has led to many proposed variants and extensions of the original approach.

However, a major limitation is the lack of knowledge about the method and its extensions. Knowledge from classical numerical methods cannot be transferred to PINNs. Despite the promising outlook, it remains unclear what performance can be expected when applied to gas transport problems. This is due to the lack of comparisons between the variants. A deeper understanding of the method, its variants, and the underlying mechanisms is critical. In addition, it must be evaluated whether an extension to solve optimization problems is possible.

Contribution. The overall goal of this thesis is twofold: First, we will give an in-depth overview of physics-informed neural networks and their extensions. We will also detail an efficient implementation and derive theoretical properties. Second, we will assess their performance when applied to gas transport problems with comprehensive numerical tests. Here, we will start with simulation problems. Then, we will transfer the knowledge to solve optimal control problems. This thesis builds on the results in [54] and expands them in new directions.

In summary, we aim to determine how physics-informed neural networks can be effectively applied to gas transport problems, evaluating their advantages and disadvantages for simulation and optimization tasks to close the existing knowledge gap.

Outline. We start by introducing two example problems modeled by hyperbolic balance laws in Chapter 2. Both problems will guide us through this thesis. Next, in Chapter 3, we will introduce the fundamentals of deep learning, the foundation of the remaining chapters. Physics-informed neural networks and their variants are introduced, and analyzed in theory and by numerical tests in Chapter 4. This knowledge is then applied to solve optimization problems based on our example problems in Chapter 5. Finally, in Chapter 6, we summarize and present our conclusions.

2. Two exemplary problems

Throughout this work, we will consider two distinct problems modeled by different hyperbolic balance laws: a linear problem and a nonlinear problem. While the linear problem holds primarily academic significance, the nonlinear problem resembles real-world scenarios more closely by using the nonlinear isentropic Euler equations and a friction term. Both problems will serve different roles in our numerical tests, thus enabling a broad view of the methods presented in this work.

For the linear problem, the solution structure is well-known and can be described explicitly. The nonlinear problem introduces additional complexities, making the solution more challenging to comprehend. Consequently, numerical simulation is needed to obtain a reference solution.

Both problems consider a one-dimensional pipe $[x_\mathrm{L}, x_\mathrm{R}]$ where the information of two state variables is transported from the left to the right boundary and vice versa over the time $[t_\mathrm{I}, t_\mathrm{E}]$. The boundary conditions are specified in accordance with the direction of the characteristics of each respective solution.

The abstract problem formulation is as follows: We seek a function $u(x,t)\colon [x_\mathrm{L}, x_\mathrm{R}] \times [t_\mathrm{I}, t_\mathrm{E}] \to \mathbb{R}^2$ that satisfies the balance law

$$\partial_t u(x,t) + \partial_x (F \circ u)(x,t) = g\big(u(x,t)\big) \quad \text{for } (x,t) \in D_\mathrm{EQ} \coloneqq (x_\mathrm{L}, x_\mathrm{R}) \times (t_\mathrm{I}, t_\mathrm{E}), \tag{2.1a}$$

with a flux $F\colon \mathbb{R}^2 \to \mathbb{R}^2$ and a source $g\colon \mathbb{R}^2 \to \mathbb{R}^2$; the initial condition

$$u(x, t_\mathrm{I}) = b_\mathrm{I}(x) \qquad \text{for} \quad x \in D_\mathrm{I} \coloneqq [x_\mathrm{L}, x_\mathrm{R}] \tag{2.1b}$$

with initial data $b_\mathrm{I}\colon D_\mathrm{I} \to \mathbb{R}^2$; the left and right boundary condition

$$u_1(x_\mathrm{L}, t) = b_\mathrm{L}(t) \qquad \text{for} \quad t \in D_\mathrm{L} \coloneqq [t_\mathrm{I}, t_\mathrm{E}], \tag{2.1c}$$

$$u_2(x_\mathrm{R}, t) = b_\mathrm{R}(t) \qquad \text{for} \quad t \in D_\mathrm{R} \coloneqq [t_\mathrm{I}, t_\mathrm{E}], \tag{2.1d}$$

with left boundary data $b_\mathrm{L}\colon D_\mathrm{L} \to \mathbb{R}$ and right boundary data $b_\mathrm{R}\colon D_\mathrm{R} \to \mathbb{R}$.

In the following, we introduce the two instances of the abstract problem (2.1).

2.1. Linear problem

For this problem, we consider the spatial interval $[x_\mathrm{L}, x_\mathrm{R}] = [0, 1]$ and the time interval $[t_\mathrm{I}, t_\mathrm{E}] = [0, 4]$. The state vector u should fulfill the abstract problem with a linear flux function and a zero source term. Specifically, we consider

$$u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}, \qquad F(u) = \begin{pmatrix} u_2 \\ u_1 \end{pmatrix} \qquad \text{and} \qquad g(u) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Continuous piecewise linear interpolation. Before we complete the problem with boundary and initial data, we define a continuous piecewise linear interpolation function ι, which interpolates input-output pairs (α_i, β_i) for $i = 1, ..., n$ according to

$$\iota(x; \alpha_1, ..., \alpha_n; \beta_1, ..., \beta_n) = \iota(x; \alpha; \beta) = \beta_i + \frac{\beta_{i+1} - \beta_i}{\alpha_{i+1} - \alpha_i}(x - \alpha_i) \text{ for } x \in [\alpha_i, \alpha_{i+1}]\,. \quad (2.2)$$

To complete this problem, we consider the initial data

$$(b_{\mathrm{I}})_1(x) = \iota(x; 0, 1; 1, -1) \quad \text{and} \quad (b_{\mathrm{I}})_2(x) = \iota(x; 0, 1; -1, 0)$$

and the boundary data on the left respectively right side

$$\begin{aligned} b_{\mathrm{L}}(t) = \iota(t; 0.0,\ & 0.5,\ 1.0, 1.5, 2.0,\ 2.5, 3.0, 4.0; \\ & 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 0.0, 0.0)\,, \\ b_{\mathrm{R}}(t) = \iota(t; 0.0, 1.0,\ & 1.5,\ 2.0, 2.5, 3.0, 3.5, 4.0; \\ & 0.0, 0.0, -1.0, -1.0, 1.0, 1.0, 0.0, 0.0)\,. \end{aligned}$$

See Figure 2.1 for a visualization of the initial, left, and right boundary data.

2.1.1. Exact solution

For this problem, we can construct the solution that will be used as a reference later. First, we observe that the system matrix is diagonalizable. That is, we have

$$F(u) = Au \qquad \text{with} \qquad A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = R\Psi R^{-1}$$
$$\text{and} \quad R = \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}, \quad \Psi = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}, \quad R^{-1} = \begin{pmatrix} -1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}.$$

Now we multiply (2.1a) from the left by R^{-1} and introduce $w = R^{-1}u$ to observe

$$\begin{aligned} \partial_t w_1 - \partial_x w_1 &= 0 \qquad \text{in } D_{\mathrm{EQ}}\,, \\ \partial_t w_2 + \partial_x w_2 &= 0 \qquad \text{in } D_{\mathrm{EQ}}\,, \end{aligned}$$

a system of linear transport equations with constant speed [34, p. 47]. The boundary conditions are transformed into

$$\begin{aligned} u_1(t) &= \begin{pmatrix} 1 & 0 \end{pmatrix} Rw(x_{\mathrm{L}}, t) = -w_1(x_{\mathrm{L}}, t) + w_2(x_{\mathrm{L}}, t) = b_{\mathrm{L}}(t)\,, \\ u_2(t) &= \begin{pmatrix} 0 & 1 \end{pmatrix} Rw(x_{\mathrm{R}}, t) = \ \ w_1(x_{\mathrm{R}}, t) + w_2(x_{\mathrm{R}}, t) = b_{\mathrm{R}}(t)\,. \end{aligned} \quad (2.3)$$

Hence, this system is only coupled at the boundary.

Now, for any point $(x, t) \in D_{\mathrm{EQ}}$ we can trace back the characteristics to calculate the function values $w_1(x, t)$ and $w_2(x, t)$ [34, p. 59]. For example, for $w_1(x, t)$ the characteristics have the direction $\binom{-1}{1}$ because the first eigenvalue is negative. Either, the characteristic intersects the $t = t_{\mathrm{I}}$ line at $x_* = x + t$ with $x_* \leq x_{\mathrm{R}}$. Then, the value of $w_1(x, t)$ is determined by the initial condition $w(x, t_{\mathrm{I}}) = R^{-1}b_{\mathrm{I}}(x)$. That is, we have $w_1(x, t) = w_1(x_*, t_{\mathrm{I}})$. Or, the characteristic intersects the right boundary at $t_* =$

$t+x-x_{\mathrm{R}}$, then we can insert the right boundary condition, $w_1(x,t) = b_{\mathrm{R}}(t_*) - w_2(x_{\mathrm{R}}, t_*)$, and recursively calculate $w_2(x_{\mathrm{R}}, t_*)$.

A similar calculation can be carried out for $w_2(x,t)$. This leads to the solution

$$w_1(x,t) = \begin{cases} w_1(x+t, t_{\mathrm{I}}) & \text{if } x+t \leq x_{\mathrm{R}}\,, \\ b_{\mathrm{R}}(t+x-x_{\mathrm{R}}) - w_2(x_{\mathrm{R}}, t+x-x_{\mathrm{R}}) & \text{otherwise}\,, \end{cases}$$

$$w_2(x,t) = \begin{cases} w_2(x-t, t_{\mathrm{I}}) & \text{if } x-t \geq x_{\mathrm{L}}\,, \\ b_{\mathrm{L}}(t-x+x_{\mathrm{L}}) + w_1(x_{\mathrm{L}}, t-x+x_{\mathrm{L}}) & \text{otherwise}\,. \end{cases}$$

As noted in [13, p.19], this solution concept extends beyond solutions that are differentiable with a continuous derivative. We will follow this notion here, and define $u = Rw$ as the solution of this problem. This is important because the initial and boundary data of the linear problem are not differentiable everywhere, and thus the solution is not differentiable everywhere.

2.2. Nonlinear problem

In contrast to the previous problem, this problem is nonlinear and describes the gas flow in a pipe with $[x_{\mathrm{L}}, x_{\mathrm{R}}] = [0, 2]$ and $[t_{\mathrm{I}}, t_{\mathrm{E}}] = [0, 6]$. It is similar to the problem described in [11, p. 71].

We start by introducing the variables ρ for the gas density, p for the pressure, v for the velocity, and the product ρv for the momentum. The state variables are the density ρ and the momentum ρv. To model the gas flow, we use the isentropic Euler equations, which are an appropriate choice for this particular task, see [34, p. 296]. We augment the conservation law with a nonlinear friction term. This model is in line with the hierarchy of models in [12]. In summary, we have

$$u = \begin{pmatrix} \rho \\ \rho v \end{pmatrix}, \quad F(u) = \begin{pmatrix} \rho v \\ \rho v^2 + p \end{pmatrix}, \quad g = \begin{pmatrix} 0 \\ -\frac{\lambda}{2D}\rho v|v| \end{pmatrix}, \quad \text{with} \quad p = \kappa \rho^\gamma\,.$$

We set the adiabatic exponent to $\gamma = 1.41$, the value for hydrogen.

In a realistic scenario, κ is between 10^3 and 10^4 and depends on the initial entropy of the gas. However, for reasons we will work out in this thesis, we choose a small κ, a less realistic scenario, and set $\kappa = 1$. To obtain a coherent problem, the sizes of ρ and ρv are also not realistic. This requires us to avoid calculating the friction term with the Colebrook-White equation and we set $\lambda = 1$. We also have $D = 1$.

The initial state $b_{\mathrm{I}}(x)$ is the steady state with respect to $\rho = 2$ at the left boundary and $\rho v = 0.5$ at the right boundary. Therefore, in this problem, the gas flows from the left boundary to the right. See Figure 2.2 for a visualization of the initial data.

Now consider a scenario where the gas enters at the left boundary and is extracted by the consumers at the right boundary. The gas demand from the consumers varies, causing ρv to increase or decrease over time. This momentarily causes the gas to expand or contract, which is reflected in changes in the density ρ. These density fluctuations are operationally undesirable and require efforts to mitigate them. This is achieved by adjusting the density at the left boundary. For example, by using a compressor to modify

the density before the gas enters the pipe. Therefore, later in Chapter 5, we will consider $b_{\mathrm{L}}(t)$ as a control that allows us to adjust the density at the left boundary.

In our concrete scenario, we consider the momentum at the right boundary

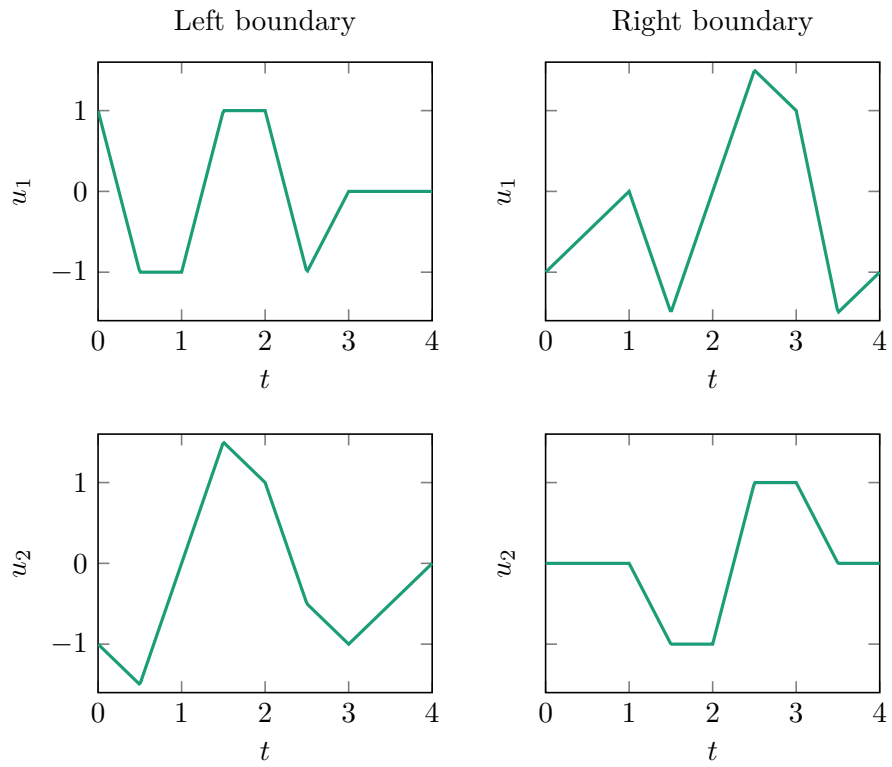
$$b_{\mathrm{R}}(t) = \begin{cases} 0.5 & 0.00 \leq t < 0.25\,, \\ -0.064(t-0.25)^3 + 0.24(t-0.25)^2 + 0.5 & 0.25 \leq t < 2.75\,, \\ 1.0 & 2.75 \leq t < 4.50\,, \\ 0.5(t-4.50)^3 - 0.75(t-4.50)^2 + 1.0 & 4.50 \leq t < 5.50\,, \\ 0.75 & 5.50 \leq t \leq 6.00\,. \end{cases}$$

Leaving the density at the left boundary unchanged, i.e. $b_{\mathrm{L}}(t) = 2$, results in a significant density drop at the right boundary. See Figure 2.2 for the corresponding solution. For our simulation problem we consider the following control for the density at the left boundary

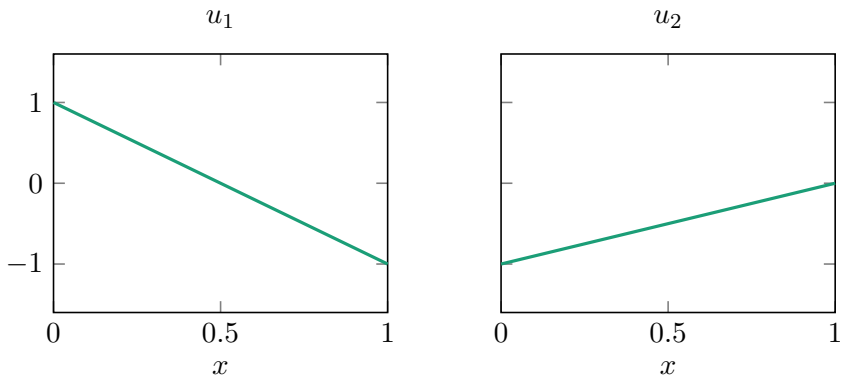
$$\begin{aligned} b_{\mathrm{L}}(t) = \iota(t; &0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0; \\ &2.0, 2.3, 2.3, 2.3, 2.2, 2.1, 2.0)\,. \end{aligned} \tag{2.4}$$

This control is able to maintain an almost constant density at the right boundary. In Chapter 5, our objective is to automate the process of determining an optimal control while also taking the associated compressor costs into account.

We compute the reference solution numerically with the implicit box method presented in [39]. We use $128 \cdot 2$ space points and $512 \cdot 6$ time points for the discrete solution. The resulting nonlinear system of equations is then solved using the MATLAB function `fsolve`. The steady state is obtained similarly, but with a much longer time interval.

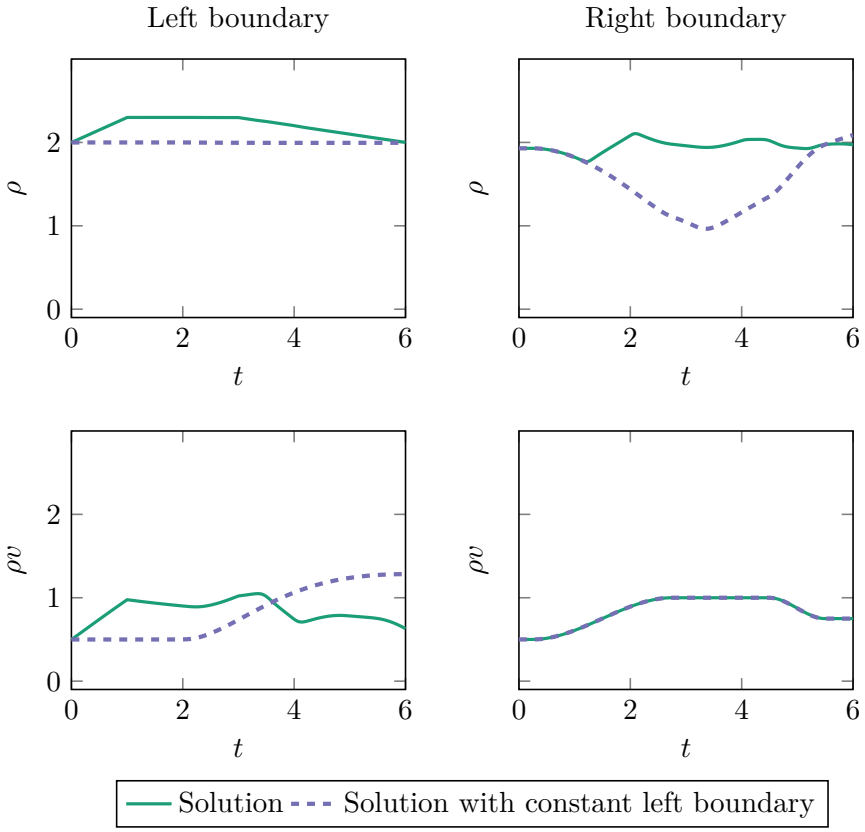


(a) Exact solution of the linear problem at the left and right boundary.

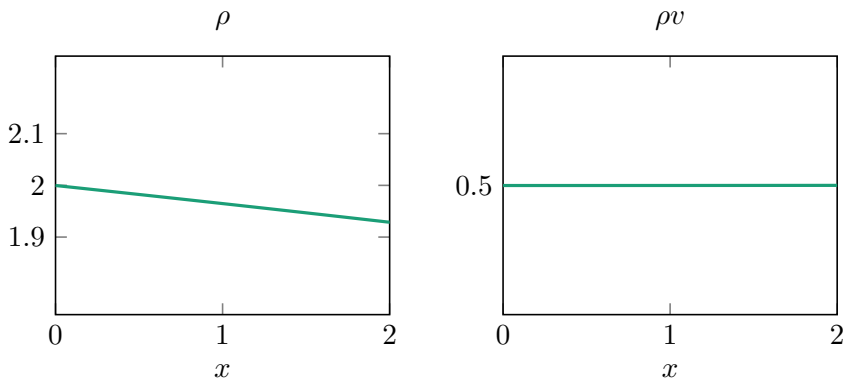


(b) Initial data of the linear problem.

Figure 2.1.: Left and right boundary as well as the initial data of the linear problem.



(a) Exact solution of the isentropic problem at the left and right boundary. The solution uses the left boundary data defined by (2.4). The solution with the constant left boundary data uses $b_{\mathrm{L}}(t) = 2$.



(b) Initial data of the isentropic problem.

Figure 2.2.: Left and right boundary as well as the initial data of the isentropic problem.

3. Fundamentals of deep learning

Machine learning is the automatic learning of complex patterns and representations. This can include tasks such as image recognition, text generation, and more. In this chapter, we focus on a specific machine learning technique called deep learning.

At its core, deep learning revolves around parameters θ and a function $h(\,\cdot\,;\theta)$, the neural network, that can be adjusted by θ. The goal is to find parameters θ such that the neural network $h(\,\cdot\,;\theta)$ approximates a target function, and is achieved by an interplay of the following building blocks.

- Specific constructions, the architecture, of $h(\,\cdot\,;\theta)$ allow for the approximation of a large class of functions. We present two different designs in Section 3.1.
- The parameters θ are obtained by nonlinear optimization methods, the training process, explained in Section 3.2.
- Nonlinear optimization requires derivatives with respect to the parameters θ which are obtained by automatic differentiation. This is described in Section 3.3.
- Finally, we describe how these techniques are efficiently implemented in soft- and hardware in Section 3.4.

Later, we will apply the deep learning techniques in two key areas. First, we will use them to approximate the solutions of the abstract simulation problem (2.1) in Chapter 4. Second, we will solve optimization problems that build on the abstract simulation problem in Chapter 5.

3.1. Designing neural networks

There are many possible constructions of the neural network $h(\,\cdot\,;\theta)$. These are commonly referred to as neural network architectures. Some of these architectures are capable of accurately approximating a wide range of functions by appropriately choosing the parameters θ. This property is known as the universal approximation property. While in theory a single neural network architecture with the universal approximation property should suffice, in practice many different architectures have been developed for specific use cases.

A neural network is typically a composition of multiple layers that transform the input into multiple intermediate representations and finally into the output. Historically, fully connected layers have been the basic architecture of neural networks. Specialized layers such as residual layers and convolutional layers have been designed for image recognition [21]. Transformer networks, such as ChatGPT, have a so-called attention mechanism and demonstrate impressive performance in natural language processing tasks [62].

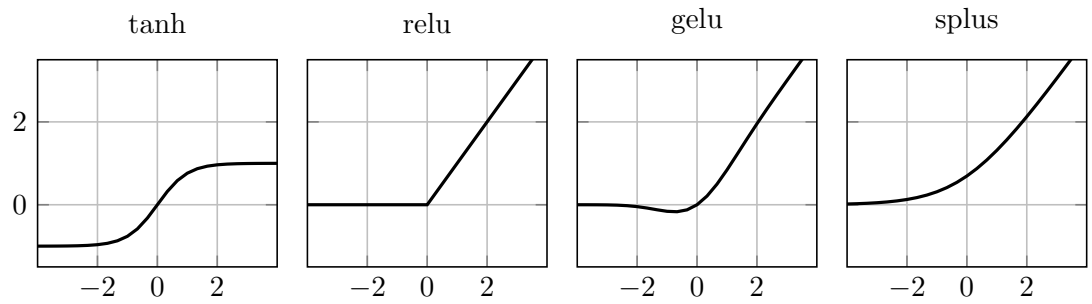


Figure 3.1.: The tanh, relu, gelu and splus activation function.

In this work, we will consider two architectures: fully connected neural networks and Hamiltonian-inspired neural networks. While the former is commonly used, the latter may offer potential advantages for our use case.

3.1.1. Fully connected deep neural networks

Universal approximation. We start by considering the task of approximating an affine linear function $f\colon \mathbb{R}^n \to \mathbb{R}^k$. To accomplish this, we use an affine model

$$h(y;\theta) = Ay + b\,,$$

with $A \in \mathbb{R}^{k\times n}$, $b \in \mathbb{R}^k$ and $\theta = (A, b)$. Since every affine function can be expressed by h, there exist parameters θ such that $h(\,\cdot\,;\theta)$ is equal to f. In this sense, h is able to exactly represent the class of affine functions.

To extend this approach to the approximation of nonlinear functions, we introduce a nonlinearity into our model. We define $\sigma\colon \mathbb{R} \to \mathbb{R}$ as a nonlinear function, which is called an activation function in this context. Popular choices for the activation function include

$$\sigma(y) = \tanh(y) \quad \text{or} \quad \sigma(y) = \mathrm{relu}(y) \coloneqq \max(0, y)\,.$$

See Figure 3.1 for a visualization of both functions and also others. The activation function σ is extended to vectors by applying it componentwise.

We now modify the model by applying an affine transformation, then a nonlinearity, and then another affine transformation. This leads to

$$\begin{aligned} h(y;\theta) &= (h_2 \circ \sigma \circ h_1)(y)\,, \quad \text{with} \\ h_1(y_1;\theta) &= A_1 y_1 + b_1 \quad \text{and} \quad h_2(y_2;\theta) = A_2 y_2 + b_2\,, \end{aligned} \tag{3.1}$$

for parameters $\theta = (A_1, b_1, A_2, b_2)$, $A_1 \in \mathbb{R}^{n_2\times n_1}$, $b_1 \in \mathbb{R}^{n_2}$, $A_2 \in \mathbb{R}^{n_3\times n_2}$ and $b_2 \in \mathbb{R}^{n_3}$. Here, $n_1 = n$ and $n_3 = k$ are based on the input and output dimensions, and n_2 can be chosen. Despite the simple construction, this is sufficient to approximate the large class of continuous functions arbitrarily well.

Theorem 1 (Universal approximation theorem [45, Proposition 3.7])**.** Let $K \subseteq \mathbb{R}^k$ be compact, σ be continuous and not a polynomial. Furthermore, let $\|\cdot\|$ be a vector norm.

Then, for every continuous function $f\colon K \to \mathbb{R}^n$ and every $\varepsilon > 0$, there exist parameters $\theta = (A_1, b_1, A_2, b_2)$ such that

$$\max_{y \in K} \|f(y) - h(y;\theta)\| < \varepsilon \,,$$

where h is the neural network defined by (3.1).

The number n_2 controls the number of parameters of the neural network. The number of parameters of a neural network is the sum of the entries of the matrices and vectors in θ. It is important to note that the Universal approximation theorem makes no assertion about the actual value of n_2 which can be arbitrarily high. However, for specific activation functions, there are estimates of the required number of parameters and bounds on the rate of convergence. For further details, we refer to [10] for the tanh activation function and [40] for the relu activation function. Finally, there are many different variants of Theorem 1, each with its own assumptions, including those on σ.

Deep neural networks. The neural network h in (3.1) can be decomposed into multiple layers. The first layer $\sigma \circ h_1$, converts the input y_1 into an intermediate representation y_2. This layer is commonly referred to as the hidden layer, and the individual elements of y_2 are known as neurons. The width of this layer is determined by the number of its neurons. The final layer h_2 serves as the output layer and typically does not have an activation function. This layer transforms the last intermediate representation into the final output.

One can make the neural network arbitrarily deep by adding more hidden layers. This leads to the deep fully connected neural network

$$h(x;\theta) = \big(h_{N+1} \circ (\sigma \circ h_N) \circ ... \circ (\sigma \circ h_2) \circ (\sigma \circ h_1)\big)(y)\,, \tag{3.2}$$

with $N+1$ layers, N hidden layers, and the corresponding affine transformations $h_i(y_i) = A_i y_i + b_i$.The different layers are separated by the additional parentheses in (3.2). The deep neural network h is parameterized by the weights A_i and biases b_i contained in $\theta = (A_1, b_1, A_2, b_2, ..., A_{N+1}, b_{N+1})$. In the following, we always assume that $A_2, ..., A_N \in \mathbb{R}^{K\times K}$, $b_1, ..., b_N \in \mathbb{R}^{K\times K}$ and denote K as the number of neurons. The size of A_1 depends on the input size, and A_{N+1}, b_{N+1} depend on the output size.

The activation function has a significant impact on the approximation capabilities of the neural network. While relu neural networks are very capable of approximating continuous piecewise linear functions [40], tanh neural networks are smooth and thus approximate smooth functions most effectively. Therefore, the activation function should be chosen carefully. In the next subsection, we explore an approach to embed more structural information into the neural network.

3.1.2. Hamiltonian-inspired neural networks.

A fully connected deep neural network is a very general architecture for approximating functions. In this subsection, we introduce a more specific neural network architecture that is derived from physical processes. This architecture is based on a connection between neural networks and ordinary differential equations that is established by a numerical time integrator. This connection allows an analysis using numerical analysis tools.

Residual layer. Fully connected layers can be modified into residual layers [21] by introducing a skip connection into the architecture. Formally, each hidden layer $(\sigma \circ h_i)$ is replaced with a residual layer

$$g_i(y_i) = y_i + \sigma(A_i y_i + b_i)\,.$$

This allows the intermediate values y_i to be passed unchanged to the next layer and ensures that no information is dropped by the hidden layer. While this approach simplifies the approximation of the identity function, it makes it more difficult to approximate constant functions.

We can rewrite the calculation of the intermediate values into

$$y_{i+1} = y_i + \sigma(A_i y_i + b_i) \tag{3.3}$$

and observe that the discrete values y_1, y_2, ... are approximations of the solution of the system of ordinary differential equations

$$y'(t) = \sigma\big(A(t)y(t) + b(t)\big) \tag{3.4}$$

obtained by the explicit Euler time integrator with step size one and suitable functions $A(t)\colon \mathbb{R} \to \mathbb{R}^{K\times K}$, $b(t)\colon \mathbb{R} \to \mathbb{R}^K$ with K as the number of neurons.

This connection allows the analysis of both the continuous system (3.4) and the discrete system (3.3) using numerical analysis tools. This has been conducted in [19] with a focus on the stability of each system. From their point of view, the continuous system (3.4) is stable if $A(t)$ changes only slowly and the real parts of the eigenvalues of the Jacobian $J_A(t)$ are non-positive. In addition, the stability of the time integrator and thus of the discrete system (3.3) has been considered. They, show that (3.3) is not unconditionally stable.

Hamiltonian systems. One of the remedies suggested by the authors is to consider the symmetric Hamiltonian system

$$\begin{aligned} z'(t) &= -\sigma\big(A(t)^\top y(t) + b(t)\big)\,,\\ y'(t) &= \quad \sigma\big(A(t) \;\; z(t) + b(t)\big)\,, \end{aligned}$$

which is unconditionally stable. The authors in [19] discretize the system using the symplectic Störmer-Verlet scheme to obtain a stable discretization. Using again step size one, they derive with $z_{-1/2} = 0$

$$\begin{aligned} z_{i+1/2} &= z_{i-1/2} - \sigma\big(A_i^\top y_i \qquad + b_i\big)\,,\\ y_{i+1} &= y_i \qquad + \sigma\big(A_i \; z_{i+1/2} + b_i\big)\,. \end{aligned}$$

This discrete system forms the basis of the Hamiltonian-inspired neural network.

Hamiltonian systems are used to model conserved energy in a dynamical system. This is similar to our use case, and thus Hamiltonian-inspired neural networks may have some potential to approximate solutions of the balance law (1.1). In our numerical tests, we consider both architectures, fully connected deep neural networks and Hamiltonian-inspired neural networks, and evaluate which one is best suited.

Outlook. The connection between deep neural networks and ordinary differential equations has been studied in other works. For example, reversible neural networks have been proposed, which offer a computational advantage by reducing the memory required to compute the gradient with respect to the parameter [7]. In addition, continuous-in-depth neural networks have been proposed with parameters that are continuous in time [46]. There is room for further research to establish a link between the ordinary differential equation of the architecture and the partial differential equation of the physical system we want to solve.

From a broader perspective, embedding structural information in neural network architectures is an elegant way to naturally ensure certain properties in a very strong sense. This has been done, for example, in [28]. A review of structure-preserving deep learning has been carried out in [6]. However, this approach requires a specialized architecture for each property, and it is not possible to embed any desired property.

In the next section, we introduce a second and our main way to obtain neural networks with certain properties. This approach is based on nonlinear optimization and enforces properties only in a weaker sense. Both approaches are conceptually different ways of achieving the same goal.

3.2. Training neural networks

We have already introduced neural networks. In this subsection, we introduce their training process. Training is the process of determining the parameter θ such that $h(\,\cdot\,;\theta)$ closely satisfies the desired properties, based on the nonlinear optimization of a loss function.

Loss function. The loss function $L(\theta)$ is defined as follows: The value of the loss function decreases as the accuracy of the approximation $h(\,\cdot\,;\theta)$ increases.

For example, we define the mean squared error loss. As a goal, the neural network $h(x;\theta)$ should approximate a function $f(x)\colon \mathbb{R}^n \to \mathbb{R}$. To proceed, we consider input-output pairs $(x_i, y_i) = (x_i, f(x_i))$ for $i = 1, ..., d$ and define the pointwise deviation $\ell(x, y; \theta) = y - h(x;\theta)$. The mean squared error loss is now defined by

$$L(\theta) = \frac{1}{d}\sum_{i=1}^{d}\big(\ell(x_i, y_i;\theta)\big)^2\,. \tag{3.5}$$

Minimizing $L(\theta)$ implies that $h(x;\theta)$ satisfies the input-output pairs in a least squares sense.

The structure of the mean squared error loss function (3.5), where each point is evaluated individually by ℓ, squared, and then summed up, is a very common construction. In this context, the input-output pairs (x_i, y_i) are often referred to as training data, and d is the size of the data set. However, the exact definition of ℓ may vary from one application to another. In the following, we assume that L has the form (3.5).

Nonlinear optimization. In the training process, one wants to find parameters θ such that the loss function L is minimal and thus the approximation is as good as possible.

Consequently, the optimization problem

$$\min_{\theta} \; L(\theta) \tag{3.6}$$

is considered, which is also called the training problem.

In the remainder of this section, we introduce two very important nonlinear optimization methods that are commonly used to solve the training problem (3.6). The first is a gradient descent method, that directly minimizes the loss values, and the second is a Newton-like method that solves for a stationary point, $\nabla L(\theta) = 0$. We also introduce initialization strategies that compute a starting point for the nonlinear optimization methods.

3.2.1. Adam Optimizer

A straightforward approach to solve (3.6) is the gradient descent method. Here, the parameters θ are updated by moving in the direction of the negative gradient $\nabla_\theta L$. Specifically, for a step size $\eta > 0$ the parameters θ are updated according to

$$\theta \leftarrow \theta - \eta \nabla_\theta L \, .$$

In the context of deep learning, the step size is often referred to as the learning rate. According to the optimization theory, η is usually obtained by a line search algorithm. Then, performing gradient descent steps iteratively guarantees a monotonically decreasing sequence of loss values and that every accumulation point of this sequence is a stationary point [60, p. 22].

However, for practical machine learning tasks, this approach is insufficient. This can be attributed to the following factors:

- In practice, the size of the data set d is large and thus the computation of $L(\theta)$ is expensive. Hence, a line search with many function evaluations is very expensive. Usually, this computational budget is spent more effectively by calculating more gradients and updating more often.

- The computational cost of evaluating the gradient of $L(\theta)$, as defined by (3.5), increases with the size of the data set d. Instead of computing the gradient with respect to the entire data set, it is common to compute the gradient with respect to a smaller subset. This gradient can be viewed as an approximation of the full gradient. Alternatively, this approach can be embedded in a stochastic setting where samples are drawn from a random distribution at each iteration. In the context of machine learning, this is known as mini-batching.

- Gradient descent methods are known for not taking the shortest path to a stationary point. This depends on the optimization landscape and can lengthen the optimization time. This behavior is also known as zigzagging and mini-batching exacerbates this problem, see [16, p. 293] or [60, p.42].

These considerations have led to the development of new variants of the gradient descent method. One of the most popular variations is the Adam (Adaptive Moment Estimation) method [26], which we introduce now. The method is based on two ideas: momentum and magnitude.

Momentum. This is also known as the heavy ball method, which provides a neat visualization of the idea: Imagine a heavy ball rolling down a hill. The direction of the ball is determined by the gradient of the hill. However, because the ball is heavy and moving, it will continue to move even if it reaches a saddle point or a local minimum. This prevents the ball from getting stuck on the way to the valley and thus reaches a lower point. Now we incorporate this idea into the gradient descent method.

Formally, the momentum is a weighted average of all gradients, with the most recent gradients having a much higher weight. We denote the momentum with s. The momentum is initially zero and is updated at each optimization step according to

$$s \leftarrow \beta_1 s + (1 - \beta_1) g \,,$$

where $g = \nabla_\theta L$ is the gradient or the gradient with respect to a subset of the training data. Also, $\beta_1 = 0.9$ is the exponential decay rate for the momentum. The update rule of s is commonly referred to as an exponential moving average.

Crucially, the momentum computes a main direction that suppresses outliers. Updating the parameters with the momentum, rather than the gradient, prevents premature stagnation and reduces zigzagging in the optimization process. This is particularly effective for managing rapidly changing or noisy gradients, which are common in stochastic settings.

The Adam method computes at the k-th step a correction of the momentum

$$\hat{s} \leftarrow \frac{1}{1 - \beta_1^k} s \,.$$

This correction ensures that the initial value of s is weighted less at the beginning of the optimization process.

Magnitude. While the momentum determines the direction, the magnitude determines the speed at which each parameter is updated. The idea is as follows: The update for a parameter should be slower in a steep landscape and faster in a flat landscape. The magnitude is a measurement of the steepness and is obtained by a moving average of the componentwise squared gradient. We denote the magnitude with r, initially set $r = 0$, and update r according to

$$r \leftarrow \beta_2 r + (1 - \beta_2)\, g * g \,,$$

where $\beta_2 = 0.99$ is the exponential decay rate for the magnitude and $*$ is the componentwise multiplication.

Now, calculating $\sqrt{r}$ componentwise measures the steepness for each parameter and can be understood as a weighted Euclidean norm measuring the magnitude of the (weighted) gradients for each parameter. Thus, if $\sqrt{r}$ is large, we want to slow down, and if $\sqrt{r}$ is small, we want to speed up. This procedure is based on the Root Mean Square Propagation (RMSProp) method [16, p. 303].

Again at step k, a correction reduces the influence of the initialization of r,

$$\hat{r} \leftarrow \frac{1}{1 - \beta_2^k} r \,.$$

Update step. The parameters θ are updated with respect to the momentum $\hat{s}$ scaled by the learning rate η and by the inverse of the of square root of the magnitude $\hat{r}$. In summary, we have the following update step

$$\theta \leftarrow \theta - \eta \frac{\hat{s}}{\sqrt{\hat{r}} + \delta} ,$$

where $\delta = 10^{-8}$ is needed to ensure a strictly positive denominator and the calculation on the right hand side is performed componentwise. The authors of the Adam method proved a convergence result in their publication. Crucially, this result requires a decaying learning rate. Thus, in this work we use an exponentially decaying learning rate η_k, that is defined at the k-th step with

$$\eta_k = \eta_{\text{INIT}} \cdot \eta_{\text{RATE}}^{\frac{k}{\eta_{\text{STEPS}}}} , \tag{3.7}$$

where η_{INIT} is the initial learning rate, η_{RATE} is the decay rate and η_{STEPS} is the decay steps. Hence, the learning rate is decayed every η_{STEPS} steps by η_{RATE}.

3.2.2. L-BFGS

In addition to the Adam method, we now introduce a second-order optimization method. Such methods solve for a stationary point, $\nabla_\theta L(\theta) = 0$, by utilizing the Hessian matrix H of $L(\theta)$ at the position θ. They solve the stationary point equations with the Newton method. This leads to the update rule

$$\theta \leftarrow \theta - \eta H^{-1} \nabla_\theta L(\theta) ,$$

for a step size η. Since neural networks have many parameters, computing, storing, and inverting the Hessian H of L is computationally intractable in most cases.

Here, the limited-memory variant of the Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method allows computing with the inverse Hessian by combining two ideas: First, the Hessian is not computed exactly, but an approximation of the inverse is computed, which is refined with newly computed gradients as the optimization progresses. Second, the inverse of the Hessian is never constructed explicitly. The method stores a low-rank approximation of the Hessian consisting of a limited number of last gradients. Then, the inverse Hessian-vector product $H^{-1}\nabla_\theta L(\theta)$ is carried out based on these gradients. Furthermore, a line search is used to calculate the step size η. In summary, this method enables the optimization of large neural networks with a second-order method.

3.2.3. Initialization

The presented nonlinear optimization methods share an iterative refinement strategy. Crucially, they require a starting point, or initial parameters. The success of deep learning methods depends strongly on the initial parameters, so they must be chosen properly. Several initialization schemes have been developed for specific activation functions. Hence, both cannot be selected separately. In the following, we want to review the main construction ideas of the initialization schemes, see also [16, Chapter 8.4].

Randomness. We begin with the following observation: If two rows of a weight matrix in a hidden layer are linearly dependent, then the gradient with respect to these rows is the same up to a constant. Thus, such a hidden layer provides one less optimization direction, and makes one row in such a weight matrix redundant. To provide a maximum of possible optimization directions, the rows should be different transformations, ideally linearly independent, to break any symmetries [16, p. 297].

In practice, this is established by random initialization schemes. This implies that we can initialize arbitrarily many parameters, but also that any result is inherently based on randomness. But again, one must be careful: Drawing weights that are too large or too small result in entries that have a disproportionately large or small effect on the gradient. Thus, the weights are drawn from a random distribution characterized by a density function with compact support.

Here, two options are usually considered: a uniform distribution $U(a, b)$, where every point in $[a, b]$ has the same probability, or a truncated normal distribution $\mathcal{N}_T(\sigma)$ with a mean of zero and a standard deviation of σ. The density function of a truncated normal distribution is equal to a scaled density function of a suitable normal distribution on $[-2\sigma, 2\sigma]$ and is zero otherwise. The suitable scaled normal distribution is chosen such that the density function of a truncated normal distribution is indeed a probability density function.

Forward and backward pass. But how to select these random distributions? In [15] and [22] simplified calculations were performed with fully connected deep neural networks without a nonlinear activation function. They studied two properties: how, for a given input distribution, the weights affect the output of the neural network, and how the weights affect the gradient of the loss function with respect to the weights. Hence, they consider the forward and the backward pass. The terminology is explained in more detail in Section 3.3.

We start with the forward pass. The variance of the input distribution should be conserved when the input is transformed through multiple layers of the neural network. Hence, the output has the same variance as the input, thus the data is not amplified or reduced. Most initialization methods aim to preserve a distribution with a mean of zero and a variance of one. Therefore, in this thesis, we will always adjust the input data to have an empirical mean of zero and an empirical variance of one. This is similar to the approach described in [27].

We continue with the backward pass. Here, the derivatives of the loss function are examined to avoid two common issues. It is crucial to prevent one entry of the weight matrix from having a negligible (*diminishing gradient*) or excessive (*exploding gradient*) effect on the gradient. These issues prevent the optimization algorithm from effectively reducing the loss function. Therefore, one tries to keep the mean of the gradient entries of the loss close to zero and the variance of the gradient entries of the loss close to one.

Common initialization schemes. Both, the forward and backward, considerations lead to (partly conflicting) conditions on the distribution of the weights. The initialization schemes balance between them and suggest distributions to draw from. Table 3.1 shows common realizations of such initialization schemes based on different assumptions.

Initialization	Uniform Distribution	Truncated Normal Distribution
Xavier [15]	$U\left(-\sqrt{\frac{6}{k_1+k_2}}, \sqrt{\frac{6}{k_1+k_2}}\right)$	$\mathcal{N}_T\left(\frac{2}{k_1+k_2}\right)$
He [22]	$U\left(-\sqrt{\frac{6}{k_1}}, \sqrt{\frac{6}{k_1}}\right)$	$\mathcal{N}_T\left(\frac{2}{k_1}\right)$

Table 3.1.: Initialization schemes for uniform and truncated normal distributions for weight $A \in \mathbb{R}^{k_2 \times k_1}$.

The Xavier initialization is best suited for sigmoid-like functions, and therefore we use it in combination with the tanh activation. The He initialization is best suited for relu-like activation functions, and we use it for these.

3.3. Automatic differentiation

In the previous sections, we introduced neural networks as universal tools for approximating functions and discussed that these networks are typically obtained by minimizing a loss function. To perform nonlinear optimization techniques, it is crucial to have access to the gradient of the loss function with respect to the parameter θ of the neural network.

One way to obtain these gradients is through analytical derivation. However, this approach is error-prone and becomes cumbersome when the model or loss function changes. Another option is to approximate the derivatives using the finite difference method. Unfortunately, this approximation is by definition inaccurate and not feasible for calculating derivatives for functions with many variables, as in our case.

Therefore, we turn our attention to automatic differentiation, a widely used algorithm for deep learning applications. It accurately calculates derivatives up to machine precision and can be applied to virtually any function composed of functions with known derivatives. See [17] for a reference.

In the following presentation, we consider a function $f\colon \mathbb{R}^{n_1} \to \mathbb{R}^{n_k}$ that is composed of elementary differentiable functions (sin, tanh, exp, ...) and operations ($+$, $-$, $\cdot$, ...). Thus we assume, there are differentiable functions $f_i\colon \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i+1}}$ with

$$f = f_k \circ f_{k-1} \circ ... \circ f_2 \circ f_1 \,. \tag{3.8}$$

For a given input y, we can calculate the output $f(y) = y_{k+1}$ iteratively with $y_{i+1} = f_i(y_i)$ and $y_1 = y$. Now the Jacobian of f, J_f, can be decomposed by the chain rule into

$$J_f(y) = J_{f_k}(y_k) \cdot J_{f_{k-1}}(y_{k-1}) \cdot ... \cdot J_{f_2}(y_2) \cdot J_{f_1}(y_1) \,. \tag{3.9}$$

The Jacobians J_{f_i} are assumed to be known, but performing Matrix-Matrix products becomes impractical for many parameters. As a result, the approach shifts towards computing Matrix-Vector products. Specifically, the Jacobian-Vector product $J_f(y) \cdot v$ and the Vector-Jacobian product $w^\top \cdot J_f(y)$ are calculated. Each product corresponds to a distinct automatic differentiation mode with unique characteristics.

Jacobian-Vector product (Forward mode). One can compute the Jacobian-Vector product $J_f(y) \cdot v = v_{k+1}$ iteratively as $v_{i+1} = J_{f_i}(y_i) \cdot v_i$ with the starting point $v_1 = v$.

Crucially, y_i and v_i can be computed at the same time while traversing from f_1 to f_k. This approach is therefore known as forward mode. At the i-th step, only the information y_i and v_i is needed to compute y_{i+1} and v_{i+1}. As a result, this mode has a small memory footprint.

If e_m represents the m-th unit vector, one forward pass calculates the m-th column of the Jacobian J_f

$$J_f(y) \cdot e_m = \begin{pmatrix} \partial_{y_1} f(y) & \dots & \partial_{y_{n_1}} f(y) \end{pmatrix} e_m = \partial_{y_m} f(y) \; . \tag{3.10}$$

Therefore, to obtain the complete Jacobian, one must perform as many forward passes as there are columns in the Jacobian.

Vector-Jacobian product (Reverse mode). Similarly, the Vector-Jacobian product $w^\top \cdot J_f(y) = w_1^\top$ is computed iteratively with $w_{k+1} = w$ and $w_i^\top = w_{i+1}^\top \cdot J_f(y_i)$.

Now note that computing w_i requires y_i. In the extreme case, obtaining w_k necessitates y_k. This implies two things: First, we need to traverse from f_1 to f_k to obtain y_1 to y_k, and second, we need to store them to use them in reverse as we traverse from J_k to J_1 to calculate w_k to w_1. This approach is thus known as the reverse mode and requires significantly more memory compared to the forward mode.

If e_m represents the m-th unit vector, a single backward pass computes the m-th row of the Jacobian J_f

$$e_m^\top \cdot J_f(y) = e_m^\top \begin{pmatrix} J_{f_1}(y) \\ \vdots \\ J_{f_{n_k}}(y) \end{pmatrix} = J_{f_m}(y) \; . \tag{3.11}$$

Consequently, obtaining the complete Jacobian requires as many backward passes as the number of rows.

Computational considerations. Both forward and reverse modes can compute the entire Jacobian $J_f(y) \in \mathbb{R}^{n_k \times n_1}$ at a given point y. To compute the entire Jacobian, the forward mode requires n_1 forward passes, while the backward mode requires n_k passes. Depending on the specific values of n_1 and n_k, either the forward or the backward mode is more efficient and should be preferred.

As a consequence, the reverse mode is particularly useful for optimizing neural networks. As explained in the previous sections, a loss function $L(\theta)$ that depends on many parameters θ, n_1 being large, with only one output, $n_k = 1$, is optimized. Therefore, only one backward pass is needed to compute the gradient $\nabla_\theta L$. Despite the disadvantages of the reverse mode compared to the forward mode, this is a significant advantage and enables training of very large neural networks. This special case of automatic differentiation is usually called backpropagation.

The considered functions f in (3.8) may seem to limit the number of functions that can be automatically differentiated with the algorithm described above, since functions such as $f = g(h(x), k(l(x))$ are not directly covered. But every general composition of functions and operations, known as computational graphs, can be reformulated to fit into

the presented scheme. However, the scheme can also be extended to capture functions of many inputs and outputs of vectors, matrices, and so on. Implementations usually choose this route and provide a very flexible interface to the user.

The computation of the Jacobian-Vector product and the Vector-Jacobian product is another computational graph. This allows for automatic differentiation of these graphs again to obtain higher order derivatives.

In this work, we will leverage automatic differentiation in two key areas. First, to compute the gradient of the loss function, and second, as we will introduce in the next chapter in Section 4.1, to calculate the derivatives in the balance law (1.1). For the second point, we will provide a comparison between forward and reverse mode to compute the derivatives in Subsection 4.1.1.d. In summary, we will utilize both the forward mode and the reverse mode in our implementation.

3.4. Implementation

Software. Currently, there are several advanced deep learning frameworks available, developed by companies such as Google and Facebook, among others. These frameworks are provided free of charge to the end user and offer sophisticated implementations of the algorithms discussed above. For our purposes, we will use Jax [3] as it closely aligns with the mathematical formulation presented in this work and provides key benefits that we will discuss in Subsection 4.1.1.d with multiple code examples. Other popular choices are TensorFlow and PyTorch.

Hardware. When designing a processor, a choice must be made between generalization, which allows for many different tasks to be performed decently, and specificity, which allows specific tasks to be performed highly efficiently. CPUs are excellent at flexibility and generalization. However, they are less effective at massively parallel computation.

In contrast, Graphics Processing Units (GPUs) are better suited for parallel computing. Originally designed for graphics, GPUs have evolved into General-Purpose Computing on Graphics Processing Units (GPGPU), making them highly suitable for deep learning and our intended applications [53]. For a more detailed explanation, see [16, p. 439]. As a result, GPUs achieve significantly higher floating point operations per second (FLOP/s) rates compared to CPUs. See also a the runtime comparison between a CPU and a GPU implementation in the next section, Section 3.5.

Broader view. Fundamentally, computer hardware performs calculations using discrete approximations of real numbers. However, many numerical challenges, such as solving differential equations, involve continuous problems with functions. This necessitates to formulate discrete problems to bridge the gap between hardware (discrete) and the continuous problems, but this often introduces additional complexity that overshadows the underlying continuous problem.

Deep learning frameworks offer a different and really compelling approach that allows working closely with the continuous problem and hides many discretization details. They offer a realization of “Computing numerically with functions instead of numbers”

[58], using neural networks, optimization methods, and automatic differentiation as the foundation. The approach described in [58] is based on piecewise Chebyshev polynomials, discrete cosine transform, and a recurrence relation to obtain the derivatives. While neural networks scale easily to many dimensions, Chebyshev polynomials do not.

3.5. Numerical tests

Before moving on to the next chapter, we want to identify neural network architectures that are very effective at approximating the solutions of our example problems. We seek either highly accurate architectures or efficient architectures, the latter defined by a better ratio of computational cost to accuracy.

Setup

State prediction. In this test we consider neural networks h with two outputs, $(h_1, h_2) = h(x, t; \theta)$. These outputs are used to construct an approximation of the state vector $u(x, t)$, denoted by $\overline{u}(h_1, h_2)$. For the linear problem we set $\overline{u}(h_1, h_2) = (h_1, h_2)^\top$.

For the isentropic problem, this is not feasible: For reasons explained in detail in Chapter 4, it is very important for the isentropic problem that the predicted density is positive for all possible parameters θ. Therefore, we cannot proceed as in the linear case. Instead, we consider different realizations of $\overline{u} = \overline{u}(h_1, h_2)$ and define an approximation of ρ and ρv, denoted by $\overline{\rho}$ and $\overline{\rho v}$, respectively.

To ensure a positive density, we consider a bijective function $\tau \colon \mathbb{R} \to (0, \infty)$ and set $\overline{\rho} = \tau(h_1)$. In addition, we interpret the second prediction h_2 as either $\overline{\rho v}$ or $\overline{v}$. In summary, we have either

$$\overline{u}(h_1, h_2) = \begin{pmatrix} \overline{\rho} \\ \overline{\rho v} \end{pmatrix} \qquad \text{and} \qquad \overline{\rho} = \tau(h_1)\,, \qquad \overline{\rho v} = h_2\,, \tag{3.12}$$

$$\text{or} \qquad \overline{u}(h_1, h_2) = \begin{pmatrix} \overline{\rho} \\ \overline{\rho} \cdot \overline{v} \end{pmatrix} \qquad \text{and} \qquad \overline{\rho} = \tau(h_1)\,, \qquad \overline{v} = h_2\,. \tag{3.13}$$

Relative error. Throughout this thesis, we measure the distance between an approximation and a reference solution with the relative 2-norm error. To define this error, we consider a solution f and a function $\overline{f}(h_1, h_2)$ that depends on the output of the neural network $(h_1, h_2) = h(x, t; \theta)$. We measure the difference between f and the approximation $\overline{f} \circ h$. Then, for a finite set $\overline{D} \subset D_{\text{EQ}}$ and neural network parameters θ, the error is defined by

$$\operatorname{error}\big[f; \theta\big] = \frac{\sqrt{\sum_{(x,t)\in\overline{D}} \Big(f(x,t) - \overline{f}\big(h(x,t;\theta)\big)\Big)^2}}{\sqrt{\sum_{(x,t)\in\overline{D}} \big(f(x,t)\big)^2}}\,. \tag{3.14}$$

We write $\operatorname{error}[f]$ when the parameters θ are unambiguous from the context.

In this numerical test, we train the neural networks by minimizing the loss function

$$L(\theta) = \operatorname{error}[u_1; \theta] + \operatorname{error}[u_2; \theta]\,,$$

where the error is computed with respect to a training set of 30 000 randomly sampled points. Note that this loss function requires the solution of the simulation problem and therefore has no practical use. In the next chapter, we will consider other loss functions that do not require the solution.

We validate the result by computing the error with respect to a validation set containing 50 000 randomly sampled points. In this thesis, we will refer to this error as the generalization error, and may only write error when it is clear from the context. The training and validation sets are sampled independently, and for the isentropic test case we sample from the grid points of the discrete reference solution, which are about 800 000 points.

Hyperparameter tuning. We evaluate various hyperparameters to determine the optimal architecture. Our choices include fully connected neural networks and Hamiltonian-inspired neural networks. We further consider the following hyperparameter

$$N \in \{2, 3, 4, 5, 6\}, \quad K \in \{20, 30, 40, 50, 60\}, \quad \sigma \in \{\tanh, \mathrm{relu}, \mathrm{splus}, \mathrm{gelu}\}.$$

In the case of the isentropic problem, we consider $\tau \in \{\exp, \mathrm{splus}\}$. The gelu, Gaussian error linear units [23], and splus, Softplus [66], activation functions are defined by

$$\mathrm{gelu}(x) = \frac{x}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right), \qquad \mathrm{splus}(x) = \log\big(1 + \exp(x)\big),$$

where erf is the Gauss error function. Both activation functions are smooth approximations of the relu activation.

We train each neural network with the Adam method and the L-BFGS method. For the Adam method, we perform 100 000 iterations and consider the following learning rate parameter

$$\eta_{\textsc{Steps}} \in \{1000, 5000, 10000\}, \quad \eta_{\textsc{Rate}} = 0.9, \quad \eta_{\textsc{Init}} \in \{0.1, 0.01, 0.001\}.$$

From these nine approximations, we consider the approximation with the lowest loss value after training. To limit the influence of randomness, whether from random initialization or random sampling, we conduct each test three times and report the average error.

Results

We start by measuring the time required for a CPU and a GPU to evaluate the gradient of the loss function. See Figure 3.2 for the runtime measurements. We observe that the GPU requires almost two orders of magnitude less time to compute the gradient of the loss function. Therefore, in this thesis, we use a GPU exclusively to obtain all results.

The results of the linear problem are listed in the Tables 3.2 and 3.3, and for the isentropic problem in the Tables 3.4 and 3.5. Each table lists the most accurate and most efficient neural network architectures for each activation function. The listed error is the generalization error, which is calculated with respect to the validation set. Efficiency for the parameters θ is measured by the ratio

$$\frac{\mathrm{error}\big[u_1; \theta_*\big] + \mathrm{error}\big[u_2; \theta_*\big]}{\mathrm{error}\big[u_1; \theta\big] + \mathrm{error}\big[u_2; \theta\big]} \cdot \frac{\#\theta_*}{\#\theta},$$

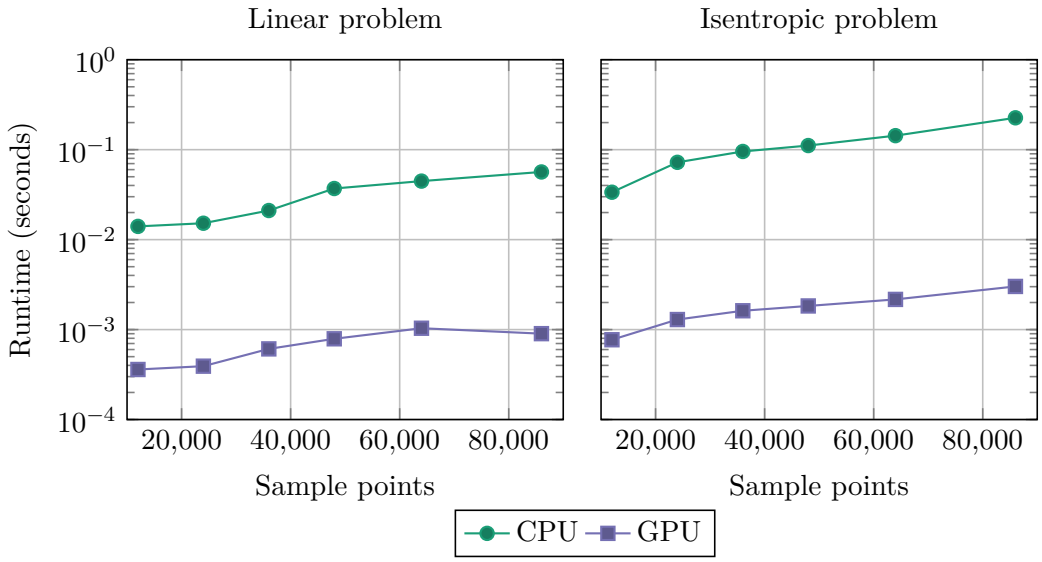


Figure 3.2.: Time to evaluate one gradient $\nabla_\theta L$ measured on a CPU and a GPU for a different number of sample points. For the isentropic problem we use a neural network with $N = 5$, $K = 50$, $\sigma = \tanh$ and for the linear problem $N = 2$, $K = 30$, $\sigma = \text{relu}$. The CPU measurements were taken on a Intel Xeon Gold 6354 CPU and the GPU measurements were taken on a Nvidia RTX A6000.

where θ_* are the accurate reference parameters. Also, $\#\theta_*$ and $\#\theta$ denote the number of parameters of θ_* and θ, respectively. Architectures with a high efficiency offer a better balance between accuracy and number of parameters.

For the isentropic problem, we observe that a higher number of parameters leads to lower generalization errors. However, it is always possible to save computational cost, by reducing the number of parameters, without sacrificing too much accuracy. The gelu activation function provides the most accurate results, followed by the tanh activation function. Furthermore, the state prediction (3.12) combined with the splus function almost always gives the best result. The best results were obtained with the Adam method and a fully connected deep neural network. The results of the Hamiltonian-inspired neural networks are not as good.

For the linear problem, we see that the relu activation function provides the best results. For the fully connected deep neural networks, the best results require only two hidden layers and are obtained by the Adam method. The best results have been computed with the L-BFGS method and the Hamiltonian-inspired neural network.

The tests show that the best neural network architecture and activation function depends on the problem and should be chosen carefully. Building on the foundation of this chapter, the next chapter introduces loss functions that directly incorporate the balance law. There, we will train neural networks to approximate the solution, without relying on the solution itself. The neural network architectures identified in this test will play an important role in this approach.

	Most accurate					Most efficient				
σ	N	K	$\#\theta$	error$[u_1]$	error$[u_2]$	N	K	$\#\theta$	error$[u_1]$	error$[u_2]$
tanh	6	50	13002	1.45e−3	1.74e−3	3	20	942	6.44e−3	8.06e−3
relu	2	20	**522**	**4.51e−4**	**5.02e−4**	2	30	1082	**6.75e−4**	**7.07e−4**
splus	6	50	13002	1.34e−3	1.66e−3	4	20	1362	4.07e−3	4.91e−3
gelu	6	50	13002	1.34e−3	1.44e−3	2	20	**522**	5.02e−3	6.24e−3

(a) Results obtained with the Adam optimization.

	Most accurate					Most efficient				
σ	N	K	$\#\theta$	error$[u_1]$	error$[u_2]$	N	K	$\#\theta$	error$[u_1]$	error$[u_2]$
tanh	6	50	13002	6.36e−4	7.60e−4	4	20	1362	1.86e−3	2.28e−3
relu	2	30	**1082**	**2.00e−4**	**2.22e−4**	2	20	**522**	**2.02e−4**	**2.54e−4**
splus	6	50	13002	1.09e−2	1.33e−2	2	20	522	2.30e−2	2.85e−2
gelu	6	30	4802	1.53e−3	1.85e−3	3	20	942	3.81e−3	4.71e−3

(b) Results obtained with the L-BFGS optimization.

Table 3.2.: Results for the linear problem approximated by fully connected neural networks. For each activation function, the most accurate and efficient result is shown.

	Most accurate					Most efficient				
σ	N	K	$\#\theta$	error$[u_1]$	error$[u_2]$	N	K	$\#\theta$	error$[u_1]$	error$[u_2]$
tanh	5	60	14942	1.80e−3	2.21e−3	3	20	**942**	5.15e−3	6.45e−3
relu	6	50	**13002**	**2.65e−5**	**3.39e−5**	4	20	1362	**5.36e−5**	**6.38e−5**
splus	6	50	13002	1.67e−3	2.06e−3	4	20	1362	3.88e−3	4.76e−3
gelu	6	50	13002	1.34e−3	1.63e−3	3	20	942	4.14e−3	5.11e−3

(a) Results obtained with the Adam optimization.

	Most accurate					Most efficient				
σ	N	K	$\#\theta$	error$[u_1]$	error$[u_2]$	N	K	$\#\theta$	error$[u_1]$	error$[u_2]$
tanh	6	50	13002	8.32e−4	1.02e−3	4	20	1362	2.32e−3	2.88e−3
relu	3	30	2012	**2.17e−4**	**2.56e−4**	3	20	**942**	**2.57e−4**	**2.65e−4**
splus	4	20	**1362**	6.32e−3	7.63e−3	3	20	942	1.19e−2	1.47e−2
gelu	5	30	3872	2.51e−3	3.06e−3	4	20	1362	3.94e−3	4.85e−3

(b) Results obtained with the L-BFGS optimization.

Table 3.3.: Results for the linear problem approximated by Hamiltonian-inspired neural networks. For each activation function, the most accurate and efficient result is shown.

	Most accurate							Most efficient						
σ	h	τ	N	K	$\#\theta$	error$[\rho]$	error$[\rho v]$	h	τ	N	K	$\#\theta$	error$[\rho]$	error$[\rho v]$
tanh	(3.12)	splus	6	50	13002	7.42e−5	1.80e−4	(3.12)	splus	3	20	**942**	4.57e−4	1.04e−3
relu	(3.12)	splus	6	50	13002	2.38e−4	4.05e−4	(3.12)	exp	2	30	1082	5.37e−4	1.12e−3
splus	(3.12)	splus	6	50	13002	1.01e−4	2.30e−4	(3.12)	splus	6	20	2202	2.31e−4	4.78e−4
gelu	(3.12)	splus	5	50	**10452**	**3.77e−5**	**7.68e−5**	(3.12)	splus	5	20	1782	**1.45e−4**	**2.56e−4**

(a) Results obtained with the Adam optimization.

	Most accurate							Most efficient						
σ	h	τ	N	K	$\#\theta$	error$[\rho]$	error$[\rho v]$	h	τ	N	K	$\#\theta$	error$[\rho]$	error$[\rho v]$
tanh	(3.12)	splus	6	50	13002	1.25e−4	3.55e−4	(3.12)	splus	4	20	1362	**2.90e−4**	**7.70e−4**
relu	(3.12)	splus	6	50	13002	2.67e−4	4.95e−4	(3.12)	splus	2	20	**522**	1.12e−3	2.54e−3
splus	(3.12)	splus	6	30	**4802**	1.32e−3	3.32e−3	(3.12)	exp	2	20	522	4.16e−3	1.10e−2
gelu	(3.12)	exp	6	30	4802	**1.05e−4**	**2.80e−4**	(3.12)	exp	2	20	522	8.76e−4	2.77e−3

(b) Results obtained with the L-BFGS optimization.

Table 3.4.: Results for the isentropic problem approximated by fully connected neural networks. For each activation function, the most accurate and efficient result is shown.

	Most accurate							Most efficient						
σ	h	τ	N	K	$\#\theta$	error$[\rho]$	error$[\rho v]$	h	τ	N	K	$\#\theta$	error$[\rho]$	error$[\rho v]$
tanh	(3.12)	splus	6	50	13002	1.34e−4	3.17e−4	(3.12)	exp	3	20	**942**	4.36e−4	1.02e−3
relu	(3.12)	splus	6	40	**8402**	1.89e−4	5.03e−4	(3.12)	splus	4	20	1362	5.74e−4	1.39e−3
splus	(3.12)	splus	5	50	10452	1.39e−4	3.71e−4	(3.12)	splus	4	20	1362	2.95e−4	7.35e−4
gelu	(3.12)	splus	6	50	13002	**9.91e−5**	**2.65e−4**	(3.12)	splus	5	20	1782	**1.35e−4**	**3.51e−4**

(a) Results obtained with the Adam optimization.

	Most accurate							Most efficient						
σ	h	τ	N	K	$\#\theta$	error$[\rho]$	error$[\rho v]$	h	τ	N	K	$\#\theta$	error$[\rho]$	error$[\rho v]$
tanh	(3.12)	splus	6	30	4802	**2.78e−4**	**6.75e−4**	(3.12)	splus	5	20	1782	**4.12e−4**	**9.68e−4**
relu	(3.12)	splus	5	30	3872	6.53e−4	1.25e−3	(3.12)	splus	4	20	1362	8.01e−4	1.67e−3
splus	(3.12)	splus	3	30	**2012**	1.38e−3	3.70e−3	(3.12)	splus	3	20	**942**	1.80e−3	5.56e−3
gelu	(3.12)	splus	5	30	3872	8.30e−4	2.49e−3	(3.12)	splus	3	20	942	8.58e−4	2.46e−3

(b) Results obtained with the L-BFGS optimization.

Table 3.5.: Results for the isentropic problem approximated by Hamiltonian-inspired neural networks. For each activation function, the most accurate and efficient result is shown.

4. Physics-informed simulations

The last chapter introduced the fundamentals of deep learning, including neural networks $h(\cdot;\theta)$, the concept of a loss function $L(\theta)$, and the associated training process. The loss function L encodes the properties the neural network $h(\cdot;\theta)$ should fulfill. To approximate the solution of the abstract simulation problem (2.1) with a neural network $h(x,t;\theta)$, we have already seen a realization of L that embeds the solution itself. But this approach is not practical.

In this chapter, we introduce two different realizations of L. Crucially, both encode the abstract simulation problem (2.1) without incorporating any unknown information about the solution. One realization is based on the differential formulation of the balance law (2.1a) and is presented in Section 4.1. The other realization is based on the integral formulation and is presented in Section 4.2. Before we continue, we need to address an issue that appears when integrating the nonlinear isentropic flux function into the loss function.

Arbitrary neural network predictions

We mentioned this issue briefly in Chapter 3, and now we will examine it more closely. Ideally, the approximation of the state vector u, denoted by $\overline{u}$, is predicted by the neural network. That is, for $(h_1, h_2) = h(x,t;\theta)$ we set

$$\overline{u}(h_1, h_2) = (h_1, h_2)^\top . \tag{4.1}$$

However, this poses some problems for the procedure presented in this chapter.

For now, we assume that (4.1) holds. During the training process, the parameters θ can be arbitrary, and therefore, the predictions of the state vector $\overline{u}$ can also be arbitrary. In this chapter, we want to integrate the balance law (1.1) into the training process. Hence, it is necessary to evaluate the flux $F(\overline{u})$ faithfully for any predicted state vector $\overline{u}$.

In the case of the nonlinear isentropic flux, the equation of state, $p = \kappa\rho^\gamma$, cannot be computed for negative densities. Furthermore, a division by zero is possible when calculating $v = \rho v/\rho$. Although this scenario seems unlikely, it does happen.

In Chapter 3, we introduced the altered state vector $\overline{u}(h_1, h_2)$, which can be used to alter the output of a neural network in a beneficial way. Specifically, we considered realizations of $\overline{u}(h_1, h_2)$ that ensure a positive predicted density $\overline{\rho}$. A positive density resolves the aforementioned issues, and as shown in the tests of the previous chapter

$$\overline{u}(h_1, h_2) = \begin{pmatrix} \overline{\rho} \\ \overline{\rho v} \end{pmatrix}, \quad \text{with} \quad \overline{\rho} = \text{splus}(h_1) \quad \text{and} \quad \overline{\rho v} = h_2$$

provides the most accurate predictions. Analogous to the altered state vector, we also define an altered flux $\overline{F}(h_1, h_2)$ and an altered source $\bar{g}(h_1, h_2)$ according to

$$\overline{F}(h_1, h_2) = \begin{pmatrix} \overline{\rho v} \\ \bar{\rho}\bar{v}^2 + \bar{p} \end{pmatrix}, \quad \bar{g}(h_1, h_2) = \begin{pmatrix} 0 \\ -\frac{\lambda}{2D}\overline{\rho v}|\bar{v}| \end{pmatrix},$$
$$\text{with} \quad \bar{p} = \kappa\bar{\rho}^\gamma \quad \text{and} \quad \bar{v} = \frac{\overline{\rho v}}{\bar{\rho}}.$$

In the linear case, we can evaluate the flux function for any state vector, so we do not have the same problems as in the isentropic case. Hence, we set $\bar{u}(h_1, h_2) = (h_1, h_2)^\top$, $\overline{F}(h_1, h_2) = F(\bar{u}(h_1, h_2))$ and $\bar{g}(h_1, h_2) = 0$. We continue for both problems with $\bar{u}$, $\overline{F}$ and $\bar{g}$ in the following.

4.1. Loss based on differential form

In this section, we introduce the loss function that encodes the abstract simulation problem (2.1), which is based on the differential form of the balance law. We refer to this loss function as physics-informed loss function, which is used to train physics-informed neural networks.

Both, the physics-informed loss function and neural networks, are presented in Subsection 4.1.1. In the remaining subsections of this section, we will study two important extensions of the original approach: Sampling strategies in Subsection 4.1.2 and Loss balancing weighting in Subsection 4.1.3.

4.1.1. Physics-informed neural networks

Residuals. As a first step to construct the physics-informed loss function, we define pointwise residuals for each equation in the abstract simulation problem (2.1). Each residual measures how well the approximation $\bar{u}(h(x, t; \theta))$ of $u(x, t)$ with the parameters θ satisfies the respective equation.

Specifically, for (x, t) we define the residual of the balance law (2.1a)

$$\ell_{\text{EQ}}(x, t; \theta) = \partial_t\big(\bar{u} \circ h\big)(x, t; \theta) + \partial_x\big(\overline{F} \circ h\big)(x, t; \theta) - (\bar{g} \circ h)(x, t; \theta). \tag{4.2a}$$

The derivatives contained in ℓ_{EQ} are obtained by automatic differentiation. The implementation is detailed in Subsection 4.1.1.d. The residual of the initial condition (2.1b) is defined by

$$\ell_{\text{I}}(x; \theta) = \bar{u}\big(h(x, t_{\text{I}}; \theta)\big) - b_{\text{I}}(x), \tag{4.2b}$$

the left boundary condition (2.1c) translates to

$$\ell_{\text{L}}(t; \theta) = \bar{u}_1\big(h(x_{\text{L}}, t; \theta)\big) - b_{\text{L}}(t), \tag{4.2c}$$

and the right boundary condition (2.1d) translates to

$$\ell_{\text{R}}(t; \theta) = \bar{u}_2\big(h(x_{\text{R}}, t; \theta)\big) - b_{\text{R}}(t). \tag{4.2d}$$

Physics-informed loss function. The residuals are used as an indicator of the distance between the approximation and the exact solution, and should hence be minimized. Therefore, for each residual, the mean squared error for a finite set of randomly selected sample points is considered in the physics-informed loss function.

For an exact definition, we consider the finite sets $\overline{D}_{\mathrm{EQ}} \subset D_{\mathrm{EQ}}$, $\overline{D}_{\mathrm{I}} \subset D_{\mathrm{I}}$, $\overline{D}_{\mathrm{L}} \subset D_{\mathrm{L}}$, and $\overline{D}_{\mathrm{R}} \subset D_{\mathrm{R}}$ of randomly sampled points and denote with d_{EQ}, d_{I}, d_{L}, and d_{R} the number of (sample) points in each respective set. The standard procedure draws the samples using the Latin hypercube strategy. A description of this strategy is given in Subsection 4.1.2. Now, the mean squared error for each residual is defined by

$$\begin{aligned} \overline{L}_{\mathrm{EQ}}(\theta) &\coloneqq \frac{1}{d_{\mathrm{EQ}}} \sum_{(x,t)\in\overline{D}_{\mathrm{EQ}}} \left\|\ell_{\mathrm{EQ}}(x,t;\theta)\right\|_2^2, \quad \overline{L}_{\mathrm{I}}(\theta) \coloneqq \frac{1}{d_{\mathrm{I}}} \sum_{x\in\overline{D}_{\mathrm{I}}} \left\|\ell_{\mathrm{I}}(x;\theta)\right\|_2^2, \\ \overline{L}_{\mathrm{L}}(\theta) &\coloneqq \frac{1}{d_{\mathrm{L}}} \sum_{t\in\overline{D}_{\mathrm{L}}} \left(\ell_{\mathrm{L}}(t;\theta)\right)^2, \qquad\qquad \overline{L}_{\mathrm{R}}(\theta) \coloneqq \frac{1}{d_{\mathrm{R}}} \sum_{t\in\overline{D}_{\mathrm{R}}} \left(\ell_{\mathrm{R}}(t;\theta)\right)^2. \end{aligned} \tag{4.3}$$

Finally, we aim to minimize the sum of each mean squared error. This leads to the training problem

$$\min_{\theta} L_{\mathrm{DIF}} \qquad \text{with} \qquad L_{\mathrm{DIF}} = \overline{L}_{\mathrm{EQ}}(\theta) + \overline{L}_{\mathrm{I}}(\theta) + \overline{L}_{\mathrm{L}}(\theta) + \overline{L}_{\mathrm{R}}(\theta), \tag{4.4}$$

the physics-informed loss function. This approach draws a lot of inspiration from other deep learning techniques by using the mean squared error loss in combination with random sampling.

Outline. Physics-informed neural networks are neural networks trained by the physics-informed loss function and are a new numerical method to approximate solutions of differential equations. This thesis is centered around this approach that was popularized by [36]. Therefore, we provide an in-depth presentation in the following subsubsections.

We begin by characterizing physics-informed neural networks in Subsubsection 4.1.1.a and compare them to other numerical methods. Then, we derive the method again from a mathematical perspective in Subsubsection 4.1.1.b. This perspective enables us to derive error estimates for the linear problem in Subsubsection 4.1.1.c. After that, we look at the implementation of ℓ_{EQ} in Subsubsection 4.1.1.d. Finally, in Subsubsection 4.1.1.e, we complete the presentation by performing numerical tests.

4.1.1.a. Characteristics of physics-informed neural networks

Classification. Physics-informed neural networks are comparable to other methods for solving differential equations, such as finite element, finite volume, and spectral methods. However, they have unique properties compared to these classical methods that we would like to highlight.

First, the neural network parameters, at least in the architectures used in this work, affect the solution on the entire domain. Therefore, physics-informed neural networks are a global method, similar to spectral methods, and in contrast to finite element methods with local elements, for example. Second, to evaluate the loss function, we need the approximation of the solution. Hence, physics-informed neural networks use an implicit

approach, like the implicit box method. Third, in this approach, time is treated like any other dimension and is not discretized using the method of lines, which is commonly applied to initial value problems.

Degrees of freedom and conditions. Traditional numerical methods for solving partial differential equations feature a carefully constructed one-to-one relationship between conditions and degrees of freedom. In contrast, physics-informed neural networks decouple conditions, residuals that should be zero, and degrees of freedom, parameters of the neural network, allowing both to be chosen independently. Consequently, the solvability of the conditions by the parameters, which is ensured by traditional solvers, is not guaranteed. Instead, the goal is to minimize the residuals and meet the conditions as closely as possible. The implications of this will be discussed later.

This flexibility enables the method to be easily adapted to a wide variety of differential equations, see for example [65, 48, 50, 64, 27, 2, 54]. In Chapter 5, we take advantage of this flexibility to solve optimization problems. In addition, the number of sample points is typically greater than the number of parameters. Thus, the least squares collocation technique [18] is the most similar numerical method to physics-informed neural networks.

We now take another look at the derivation of the physics-informed loss L_{DIF}.

4.1.1.b. Mathematical perspective

We want to reconsider the derivation of $\overline{L}_{\text{EQ}}$, $\overline{L}_{\text{L}}$, $\overline{L}_{\text{R}}$, and $\overline{L}_{\text{I}}$. Although presenting them as the mean squared error loss of the residuals is concise, it neglects important properties of the method. But they are essential for the next steps.

Continuous setting. The residuals ℓ_{EQ}, ℓ_{I}, ℓ_{L}, and ℓ_{R} should be minimal over the entire domain. Therefore, we measure them using a (continuous) norm. Considering the definition of $\overline{L}_{\text{EQ}}$, $\overline{L}_{\text{L}}$, $\overline{L}_{\text{R}}$, and $\overline{L}_{\text{I}}$, the $\mathscr{L}_2$-norm fits exactly. For a function $g\colon D \to \mathbb{R}^n$ and $D \subset \mathbb{R}^k$ the $\mathscr{L}_2$-norm is defined by

$$\|g\|_{\mathscr{L}_2(D)} = \sqrt{\int_D \|g(x)\|_2^2 \,\mathrm{d}x}\,,$$

where $\|\cdot\|_2$ denotes the Euclidean-norm and the integral is a Lebesgue integral.

Consequently, we consider the following squared $\mathscr{L}_2$-norms of the residuals

$$L_{\text{EQ}} = \|\ell_{\text{EQ}}\|^2_{\mathscr{L}_2(D_{\text{EQ}})} = \int_{D_{\text{EQ}}} \left\|\ell_{\text{EQ}}\right\|_2^2 \mathrm{d}(x,t)\,, \qquad L_{\text{I}} = \|\ell_{\text{I}}\|^2_{\mathscr{L}_2(D_{\text{I}})} = \int_{D_{\text{I}}} \left\|\ell_{\text{I}}\right\|_2^2 \mathrm{d}x\,,$$

$$L_{\text{L}} = \|\ell_{\text{L}}\|^2_{\mathscr{L}_2(D_{\text{L}})} = \int_{D_{\text{L}}} \left(\ell_{\text{L}}\right)^2 \mathrm{d}t\,, \qquad L_{\text{R}} = \|\ell_{\text{R}}\|^2_{\mathscr{L}_2(D_{\text{R}})} = \int_{D_{\text{R}}} \left(\ell_{\text{R}}\right)^2 \mathrm{d}t\,.$$

For hyperbolic problems, where shocks or contact discontinuities occur, the $\mathscr{L}_2$-norm is not suitable for measuring residuals. This requires other approaches such as those presented in [8]. However, we do not expect discontinuities to occur in our nonlinear problem, or more generally in gas networks, and therefore do not consider this further.

The next step is to discretize the integrals using a quadrature method. By applying the Monte Carlo quadrature and neglecting the volume of the integration domain, one

obtains the original definition of the discrete objective $\overline{L}_{\text{EQ}}$, $\overline{L}_{\text{L}}$, $\overline{L}_{\text{R}}$, and $\overline{L}_{\text{I}}$. Specifically, for $k = \text{EQ}, \text{L}, \text{R}, \text{I}$ the Monte Carlo approximation of L_k is defined by

$$\frac{1}{\operatorname{vol}(D_k)} L_k(\theta) \approx \frac{1}{d_k} \sum_{z \in \overline{D}_k} \ell_k(z;\theta)^2 = \overline{L}_k(\theta)\,. \tag{4.5}$$

The derivation emphasizes the chosen quadrature rule, but also shows that we can substitute other quadrature rules. We will discuss the Monte Carlo quadrature and other choices further in Subsection 4.1.2.

Note, that the relative 2-norm error, defined by (3.14), is also a Monte Carlo approximation of the relative $\mathscr{L}_2$-error, since we have

$$\begin{aligned}\Big(\operatorname{error}\big[f;\theta\big]\Big)^2 &= \frac{\sum_{(x,t)\in\overline{D}} \big(f(x,t) - \overline{f}(x,t;\theta)\big)^2}{\sum_{(x,t)\in\overline{D}} \big(f(x,t)\big)^2} \\ &\approx \frac{\int_{D_{\text{EQ}}} \big(f(x,t) - \overline{f}(x,t;\theta)\big)^2 \,\mathrm{d}(x,t)}{\int_{D_{\text{EQ}}} \big(f(x,t)\big)^2 \,\mathrm{d}(x,t)}\,,\end{aligned}$$

for functions $f(x,t)$, $\overline{f}(h_1,h_2)$, a neural network $h(x,t;\theta)$ and a finite set $\overline{D} \subset D_{\text{EQ}}$.

Multi-objective setting. We have just derived discrete approximations of the norms that measure each residual. Formally, we want to minimize the loss functions $\overline{L}_{\text{EQ}}$, $\overline{L}_{\text{L}}$, $\overline{L}_{\text{R}}$, and $\overline{L}_{\text{I}}$ simultaneously. Consequently, we consider the multi-objective optimization problem

$$\min_{\theta} \Big(\overline{L}_{\text{EQ}}(\theta), \overline{L}_{\text{I}}(\theta), \overline{L}_{\text{L}}(\theta), \overline{L}_{\text{R}}(\theta)\Big)\,. \tag{4.6}$$

In the original formulation, this problem is converted into a single-objective optimization problem by forming the sum of all objectives. This results in the definition of L_{DIF} and the physics-informed training problem (4.4).

However, this conversion weights every objective by the multiplicative inverse of the volume of the respective domain, see (4.5). In a more general setting, the objectives are weighted differently, which may have a positive impact on the resulting approximation. This leads to loss-balancing strategies, which will be introduced in Subsection 4.1.3.

Before we proceed with this topic, we expand on the mathematical perspective we just introduced and derive error estimates for the linear problem.

4.1.1.c. Error estimates in the linear case

In this subsubsection, we develop further knowledge about the underlying mechanisms of physics-informed neural networks by deriving error estimates for the linear problem. These estimates bound the generalization error by the loss values L_{I}, L_{L}, L_{R}, L_{EQ} and thus show that minimizing the losses decreases the generalization error. This procedure is based on the work in [49].

In the following, we consider a problem with the linear flux

$$F(u) = Au \qquad \text{with} \qquad A = \begin{pmatrix} 0 & 1 \\ c^2 & 0 \end{pmatrix} \quad \text{and} \quad c > 0\,.$$

This flux function can be cast as a generalization of the linear flux, or as a flux that is derived from the nonlinear Euler equations equipped with the isothermal equation of state, $p = c^2\rho$, when the nonlinear term ρv^2 is neglected [12, p. 12]. In this setting, c corresponds to the speed of sound of the considered gas.

Absolute continuity. Our linear problem has continuous piecewise linear initial and boundary data, and thus the solution u is only differentiable almost everywhere on D_{EQ}. The same holds for the approximation u^*, if u^* is defined by a neural network with the relu activation function. To capture a general class of solutions and approximations, we will use the notion of absolute continuous functions. We summarize the following important properties:

- A function $f\colon [a,b] \to \mathbb{R}$ is absolute continuous if the derivative f' exists almost everywhere, the derivative f' is Lebesgue-integrable and

$$f(t) - f(a) = \int_a^t f'(s)\,\mathrm{d}s \qquad \text{for all } t \in [a,b]\,.$$

- The sum of two absolute continuous functions is absolute continuous.

- The product of two absolute continuous functions defined on a bounded domain is also absolute continuous.

Setup. We denote with $u(x,t)\colon D_{\text{EQ}} \to \mathbb{R}^2$ the solution of the problem and with $u^*(x,t)\colon D_{\text{EQ}} \to \mathbb{R}^2$ the approximation. Note, that $u^*(x,t;\theta)$ differs from $\overline{u}$ since it directly depends on x and t. One could set $u^*(x,t) = (\overline{u} \circ h)(x,t;\theta)$ for a neural network h with parameters θ. We assume that $u_i(\cdot,t)$, $u_i^*(\cdot,t)$, $u_i(x,\cdot)$, $u_i^*(x,\cdot)$ are absolute continuous functions for $i = 1,2$, $x \in [x_{\text{L}}, x_{\text{R}}]$ and $t \in [t_{\text{I}}, t_{\text{E}}]$. Furthermore, u should satisfy the following linear problem

$$\begin{cases} \partial_t u + A\partial_x u = 0 & \text{a.e. in } D_{\text{EQ}}\,, \\ u(x,t_{\text{I}}) = b_{\text{I}}(x) & \text{on } D_{\text{I}}\,, \\ u_1(x_{\text{L}},t) = b_{\text{L}}(t) & \text{on } D_{\text{L}}\,, \\ u_2(x_{\text{R}},t) = b_{\text{R}}(t) & \text{on } D_{\text{R}}\,. \end{cases}$$

The approximation u^* should fulfill the following perturbed linear problem

$$\begin{cases} \partial_t u^* + A\partial_x u^* = \ell_{\text{EQ}}(x,t) & \text{a.e. in } D_{\text{EQ}}\,, \\ u^*(x,t_{\text{I}}) = b_{\text{I}}(x) + \ell_{\text{I}}(x) & \text{on } D_{\text{I}}\,, \\ u_1^*(x_{\text{L}},t) = b_{\text{L}}(t) + \ell_{\text{L}}(t) & \text{on } D_{\text{L}}\,, \\ u_2^*(x_{\text{R}},t) = b_{\text{R}}(t) + \ell_{\text{R}}(t) & \text{on } D_{\text{R}}\,, \end{cases}$$

for the pointwise perturbations $\ell_{\text{EQ}}(x,t)$, $\ell_{\text{I}}(x)$, $\ell_{\text{L}}(t)$, $\ell_{\text{R}}(t)$ which could be defined analogous to (4.2a), (4.2b), (4.2c), (4.2d). The difference between the exact solution and the

solution of the pertubated system $\mu = u - u^*$ thus satisfies

$$\begin{cases} \partial_t \mu + A\partial_x \mu = -\ell_{\mathrm{EQ}}(x,t) & \text{a.e. in } D_{\mathrm{EQ}}\,, \\ \mu(x,t_{\mathrm{I}}) = -\ell_{\mathrm{I}}(x) & \text{on } D_{\mathrm{I}}\,, \\ \mu_1(x_{\mathrm{L}},t) = -\ell_{\mathrm{L}}(t) & \text{on } D_{\mathrm{L}}\,, \\ \mu_2(x_{\mathrm{R}},t) = -\ell_{\mathrm{R}}(t) & \text{on } D_{\mathrm{R}}\,. \end{cases}$$

Note that u^* and therefore ℓ_{I}, ℓ_{L}, ℓ_{R}, ℓ_{EQ} as well as L_{I}, L_{L}, L_{R}, L_{EQ} may depend on the parameters θ. In the following, we assume that the parameters θ are constant and therefore do not explicitly state the dependency.

We begin with the following

Observation. The matrix A is diagonalizable with

$$A = R\Psi R^{-1} \quad \text{and} \quad R = \begin{pmatrix} -1/c & 1/c \\ 1 & 1 \end{pmatrix}, \quad \Psi = \begin{pmatrix} -c & 0 \\ 0 & c \end{pmatrix}, \quad R^{-1} = \begin{pmatrix} -c/2 & 1/2 \\ c/2 & 1/2 \end{pmatrix}.$$

We set $\omega = R^{-1}\mu$, then ω satisfies the system

$$\begin{cases} \partial_t \omega + \Psi\partial_x \omega = \beta(x,t) & \text{a.e. in } D_{\mathrm{EQ}}\,, \qquad (4.7) \\ \omega(x,t_{\mathrm{I}}) = \alpha(x) & \text{on } D_{\mathrm{I}}\,, \\ \omega_2(x_{\mathrm{L}},t) = \omega_1(x_{\mathrm{L}},t) - c\,\ell_{\mathrm{L}}(t) & \text{on } D_{\mathrm{L}}\,, \qquad (4.8) \\ \omega_1(x_{\mathrm{R}},t) = -\ell_{\mathrm{R}}(t) - \omega_2(x_{\mathrm{R}},t) & \text{on } D_{\mathrm{R}}\,, \qquad (4.9) \end{cases}$$

with $\alpha(x) = -R^{-1}\ell_{\mathrm{I}}(x), \beta(x,t) = -R^{-1}\ell_{\mathrm{EQ}}(x,t)$.

Using the same approach as in Subsection 2.1.1, we can construct an explicit description of ω. Crucially, we need to account for the source term on the right-hand side. Using the solution formula stated in [13, p. 19], we have

$$\omega_1(x,t) = \begin{cases} \alpha_1(x+ct) + \displaystyle\int_0^t \beta_1\big(x + c(t-s), s\big)\,\mathrm{d}s & \text{if } x + ct \le x_{\mathrm{R}}, \\ \omega_1\left(x_{\mathrm{R}}, t + \dfrac{x - x_{\mathrm{R}}}{c}\right) + \displaystyle\int_{t+\frac{x-x_{\mathrm{R}}}{c}}^t \beta_1\big(x + c(t-s), s\big)\,\mathrm{d}s & \text{otherwise,} \end{cases}$$

$$\omega_2(x,t) = \begin{cases} \alpha_2(x-ct) + \displaystyle\int_0^t \beta_2\big(x - c(t-s), s\big)\,\mathrm{d}s & \text{if } x_{\mathrm{L}} \le x - ct, \\ \omega_2\left(x_{\mathrm{L}}, t + \dfrac{x_{\mathrm{L}} - x}{c}\right) + \displaystyle\int_{t+\frac{x_{\mathrm{L}}-x}{c}}^t \beta_2\big(x - c(t-s), s\big)\,\mathrm{d}s & \text{otherwise.} \end{cases}$$

Note, in contrast to the construction in Chapter 2, we have not incorporated the boundary conditions.

For convenience, we define the length of the spatial domain with $\delta = x_{\mathrm{R}} - x_{\mathrm{L}}$. For the diagonalized system, we can derive the following estimates.

Lemma 1. 1. For every T with $t_{\mathrm{I}} \le T \le \delta/c$, the following inequality holds

$$\begin{aligned} \left\| \begin{pmatrix} \omega_1(x_{\mathrm{R}},\,\cdot\,) \\ \omega_2(x_{\mathrm{L}},\,\cdot\,) \end{pmatrix} \right\|_{\mathscr{L}_2([t_{\mathrm{I}},T])} &\le \|\ell_{\mathrm{R}}\|_{\mathscr{L}_2([t_{\mathrm{I}},T])} + c\,\|\ell_{\mathrm{L}}\|_{\mathscr{L}_2([t_{\mathrm{I}},T])} \\ &\quad + c^{-\frac{1}{2}}\,\|\alpha\|_{\mathscr{L}_2([x_{\mathrm{L}},x_{\mathrm{R}}])} + \delta^{\frac{1}{2}} c^{-1}\,\|\beta\|_{\mathscr{L}_2([x_{\mathrm{L}},x_{\mathrm{R}}]\times[t_{\mathrm{I}},\delta/c])}\,. \end{aligned} \qquad (4.10)$$

2. For every T with $\delta/c \le T \le t_{\mathrm{E}}$, the following inequality holds

$$\begin{aligned}\left\| \begin{pmatrix} \omega_1(x_{\mathrm{R}},\,\cdot\,) \\ \omega_2(x_{\mathrm{L}},\,\cdot\,) \end{pmatrix} \right\|_{\mathscr{L}_2([t_{\mathrm{I}},T])} &\le \left\| \begin{pmatrix} \omega_1(x_{\mathrm{R}},\,\cdot\,) \\ \omega_2(x_{\mathrm{L}},\,\cdot\,) \end{pmatrix} \right\|_{\mathscr{L}_2([t_{\mathrm{I}},T-\delta/c])} \\ &\quad + \|\ell_{\mathrm{R}}\|_{\mathscr{L}_2([t_{\mathrm{I}},T])} + c\,\|\ell_{\mathrm{L}}\|_{\mathscr{L}_2([t_{\mathrm{I}},T])} \\ &\quad + c^{-\frac{1}{2}}\,\|\alpha\|_{\mathscr{L}_2([x_{\mathrm{L}},x_{\mathrm{R}}])} + \delta^{\frac{1}{2}} c^{-1}\,\|\beta\|_{\mathscr{L}_2([x_{\mathrm{L}},x_{\mathrm{R}}]\times[t_{\mathrm{I}},T])}\,. \end{aligned} \tag{4.11}$$

3. The estimate

$$\|\omega\|_{\mathscr{L}_2([x_{\mathrm{L}},x_{\mathrm{R}}]\times[t_{\mathrm{I}},t_{\mathrm{E}}])} \le C\big(\exp(t_{\mathrm{E}}-t_{\mathrm{I}})-1\big) \tag{4.12}$$

holds for $M = \left\lceil \frac{c(t_{\mathrm{E}}-t_{\mathrm{I}})}{\delta} \right\rceil$ and

$$\begin{aligned} C &= c^{\frac{1}{2}} M\,\|\ell_{\mathrm{R}}\|_{\mathscr{L}_2([t_{\mathrm{I}},t_{\mathrm{E}}])} + c^{\frac{3}{2}} M\,\|\ell_{\mathrm{L}}\|_{\mathscr{L}_2([t_{\mathrm{I}},t_{\mathrm{E}}])} \\ &\quad + (M+1)\,\|\alpha\|_{\mathscr{L}_2([x_{\mathrm{L}},x_{\mathrm{R}}])} + \Big(M\delta^{\frac{1}{2}} c^{-\frac{1}{2}} + 1\Big)\,\|\beta\|_{\mathscr{L}_2([x_{\mathrm{L}},x_{\mathrm{R}}]\times[t_{\mathrm{I}},t_{\mathrm{E}}])}\,. \end{aligned}$$

Proof. We start by showing the second assertion. Without loss of generality we assume $t_{\mathrm{I}} = 0$. We insert the boundary conditions (4.8), (4.9) and use the triangle inequality to obtain

$$\left\| \begin{pmatrix} \omega_1(x_{\mathrm{R}},\,\cdot\,) \\ \omega_2(x_{\mathrm{L}},\,\cdot\,) \end{pmatrix} \right\|_{\mathscr{L}_2([0,T])} \le \left\| \begin{pmatrix} \omega_2(x_{\mathrm{R}},\,\cdot\,) \\ \omega_1(x_{\mathrm{L}},\,\cdot\,) \end{pmatrix} \right\|_{\mathscr{L}_2([0,T])} + \left\| \begin{pmatrix} \ell_{\mathrm{R}} \\ c\ell_{\mathrm{L}} \end{pmatrix} \right\|_{\mathscr{L}_2([0,T])}\,. \tag{4.13}$$

According to the exact definition of ω, we see that $\omega_2(x_{\mathrm{R}},\,\cdot\,)$ respectively $\omega_1(x_{\mathrm{L}},\,\cdot\,)$ are defined piecewise on $[0,\delta/c]$ and $[\delta/c,T]$. Consequently, we define

$$\mathcal{A}(t) = \begin{pmatrix} \alpha_2(x_{\mathrm{R}}-ct) \\ \alpha_1(x_{\mathrm{L}}+ct) \end{pmatrix} \quad \text{and} \quad \mathcal{B}^{(1)}(t) = \begin{pmatrix} \int_0^t \beta_2\big(x_{\mathrm{R}}-c(t-s),s\big)\,\mathrm{d}s \\ \int_0^t \beta_1\big(x_{\mathrm{L}}+c(t-s),s\big)\,\mathrm{d}s \end{pmatrix} \text{ on } [0,\delta/c]\,;$$

$$\mathcal{W}(t) = \begin{pmatrix} \omega_2(x_{\mathrm{L}},t-\delta/c) \\ \omega_1(x_{\mathrm{R}},t-\delta/c) \end{pmatrix} \quad \text{and} \quad \mathcal{B}^{(2)}(t) = \begin{pmatrix} \int_{t-\delta/c}^t \beta_2\big(x_{\mathrm{R}}-c(t-s),s\big)\,\mathrm{d}s \\ \int_{t-\delta/c}^t \beta_1\big(x_{\mathrm{L}}+c(t-s),s\big)\,\mathrm{d}s \end{pmatrix} \text{ on } [\delta/c,T]\,.$$

Let $\mathcal{A}(t)$ be zero outside of $[0,\delta/c]$, $\mathcal{W}(t)$ be zero outside of $[\delta/c,T]$ and

$$\mathcal{B}(t) = \begin{cases} \mathcal{B}^{(1)}(t) & \text{for } t \in [0,\delta/c]\,, \\ \mathcal{B}^{(2)}(t) & \text{for } t \in (\delta/c,T]\,, \end{cases}$$

then we have

$$\begin{pmatrix} \omega_2(x_{\mathrm{R}},t) \\ \omega_1(x_{\mathrm{L}},t) \end{pmatrix} = \mathcal{A}(t) + \mathcal{W}(t) + \mathcal{B}(t)\,.$$

Therefore, we conclude

$$\left\| \begin{pmatrix} \omega_2(x_{\mathrm{R}},\,\cdot\,) \\ \omega_1(x_{\mathrm{L}},\,\cdot\,) \end{pmatrix} \right\|_{\mathscr{L}_2([0,T])} \le \|\mathcal{A}\|_{\mathscr{L}_2([0,\delta/c])} + \|\mathcal{W}\|_{\mathscr{L}_2([\delta/c,T])} + \|\mathcal{B}\|_{\mathscr{L}_2([0,T])}\,. \tag{4.14}$$

We estimate the norms in (4.14) separately. We start with the norm of $\mathcal{A}$ and observe

$$\|\mathcal{A}\|^2_{\mathscr{L}_2([0,\delta/c])} = \int_0^{\delta/c} \mathcal{A}_1(t)^2 + \mathcal{A}_2(t)^2\,\mathrm{d}t = \frac{1}{c}\int_{x_{\mathrm{L}}}^{x_{\mathrm{R}}} \alpha_2(x)^2 + \alpha_1(x)^2\,\mathrm{d}x = \frac{1}{c}\,\|\alpha\|^2_{\mathscr{L}_2([x_{\mathrm{L}},x_{\mathrm{R}}])}\,. \tag{4.15}$$

The change of variables is performed by two different substitutions, specifically

$$\int_0^{\delta/c} \alpha_2(x_{\mathrm{R}} - ct)^2 \,\mathrm{d}t = \frac{1}{c}\int_{x_{\mathrm{L}}}^{x_{\mathrm{R}}} \alpha_2(x)^2 \,\mathrm{d}x\,, \qquad \int_0^{\delta/c} \alpha_1(x_{\mathrm{L}} + ct)^2 \,\mathrm{d}t = \frac{1}{c}\int_{x_{\mathrm{L}}}^{x_{\mathrm{R}}} \alpha_1(x)^2 \,\mathrm{d}x\,.$$

Also by substitution we derive for the norm of $\mathcal{W}$

$$\begin{aligned}\|\mathcal{W}\|^2_{\mathscr{L}_2([\delta/c,T])} &= \int_{\delta/c}^{T} \omega_2(x_{\mathrm{L}}, t-\delta/c)^2 + \omega_1(x_{\mathrm{R}}, t-\delta/c)^2 \,\mathrm{d}t \\ &= \int_0^{T-\delta/c} \omega_2(x_{\mathrm{L}}, t)^2 + \omega_1(x_{\mathrm{R}}, t)^2 \,\mathrm{d}t = \left\| \begin{pmatrix} \omega_1(x_{\mathrm{R}},\,\cdot\,) \\ \omega_2(x_{\mathrm{L}},\,\cdot\,) \end{pmatrix} \right\|^2_{\mathscr{L}_2([0,T-\delta/c])}\,.\end{aligned} \tag{4.16}$$

It remains to bound the norm of $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)^\top$. We start by splitting the squared $\mathscr{L}_2$-norm

$$\|\mathcal{B}\|^2_{\mathscr{L}_2([0,T])} = \|\mathcal{B}_1\|^2_{\mathscr{L}_2([0,T])} + \|\mathcal{B}_2\|^2_{\mathscr{L}_2([0,T])}\,. \tag{4.17}$$

We present only the estimate for the norm of $\mathcal{B}_1$. However, the estimate of $\mathcal{B}_2$ works with the same arguments. First, we observe by Hölder's inequality for $t \in [0, \delta/c]$

$$\begin{aligned}\Big(\mathcal{B}_1^{(1)}(t)\Big)^2 &= \left(\int_0^t \beta_2\big(x_{\mathrm{R}} - c(t-s), s\big)\,\mathrm{d}s\right)^2 \\ &\le \left(\int_0^t \big|\beta_2\big(x_{\mathrm{R}} - c(t-s), s\big)\big|\,\mathrm{d}s\right)^2 \le \frac{\delta}{c}\int_0^t \beta_2\big(x_{\mathrm{R}} - c(t-s), s\big)^2\,\mathrm{d}s\,,\end{aligned}$$

and for $t \in [\delta/c, T]$

$$\Big(\mathcal{B}_1^{(2)}(t)\Big)^2 \le \left(\int_{t-\delta/c}^t \big|\beta_2\big(x_{\mathrm{R}} - c(t-s), s\big)\big|\,\mathrm{d}s\right)^2 \le \frac{\delta}{c}\int_{t-\delta/c}^t \beta_2\big(x_{\mathrm{R}} - c(t-s), s\big)^2\,\mathrm{d}s\,.$$

Combing both yields

$$\begin{aligned}&\|\mathcal{B}_1\|^2_{\mathscr{L}_2([0,T])} \\ &\quad= \int_0^{\delta/c} \Big(\mathcal{B}_1^{(1)}\Big)^2\,\mathrm{d}t + \int_{\delta/c}^T \Big(\mathcal{B}_1^{(2)}\Big)^2\,\mathrm{d}t \\ &\quad\le \frac{\delta}{c}\int_0^{\delta/c}\int_0^t \beta_2\big(x_{\mathrm{R}} - c(t-s), s\big)^2\,\mathrm{d}s\,\mathrm{d}t + \frac{\delta}{c}\int_{\delta/c}^T\int_{t-\delta/c}^t \beta_2\big(x_{\mathrm{R}} - c(t-s), s\big)^2\,\mathrm{d}s\,\mathrm{d}t \\ &\quad= \frac{\delta}{c}\int_{B_1} \beta_2\big(x_{\mathrm{R}} - c(t-s), s\big)^2\,\mathrm{d}(s,t) + \frac{\delta}{c}\int_{B_2} \beta_2\big(x_{\mathrm{R}} - c(t-s), s\big)^2\,\mathrm{d}(s,t)\,.\end{aligned}$$

In the last step we used the domains

$$B_1 = \{(t,s)\colon t \in [0, \delta/c]\,,\, 0 \le s \le t\}\,, \quad B_2 = \{(t,s)\colon t \in [\delta/c, T]\,,\, t - \delta/c \le s \le t\}\,.$$

We transform B_1 and B_2 by

$$\varphi(t,s) = \big(x_{\mathrm{R}} - ct + cs, s\big) \quad \text{with} \quad J_\varphi = \begin{pmatrix} -c & c \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad |\det(J_\varphi)| = c\,.$$

The domain B_1 is a triangle and the corners are mapped as follows

$$\varphi(0,0) = (x_{\mathrm{R}}, 0)\,, \quad \varphi(\delta/c, 0) = (x_{\mathrm{L}}, 0)\,, \quad \varphi(\delta/c, \delta/c) = (x_{\mathrm{R}}, \delta/c)\,.$$

Hence B_1 is transformed into

$$\varphi(B_1) = \left\{ (x,\tau)\colon x \in [x_{\mathrm{L}}, x_{\mathrm{R}}]\,, \tau \in \left[0, \frac{x - x_{\mathrm{L}}}{c}\right] \right\}.$$

The domain B_2 is a parallelogram and its corners are mapped according to

$$\begin{aligned} \varphi(\delta/c, 0) &= (x_{\mathrm{L}}, 0)\,, & \varphi(\delta/c, \delta/c) &= (x_{\mathrm{R}}, \delta/c)\,, \\ \varphi(T, T - \delta/c) &= (x_{\mathrm{L}}, T - \delta/c)\,, & \varphi(T,T) &= (x_{\mathrm{R}}, T)\,. \end{aligned}$$

Thus, we have

$$\varphi(B_2) = \left\{ (x,\tau)\colon x \in [x_{\mathrm{L}}, x_{\mathrm{R}}]\,, \tau \in \left[\frac{x - x_{\mathrm{L}}}{c}, T - \frac{x_{\mathrm{R}} - x}{c}\right] \right\}.$$

In summary, we conclude

$$\begin{aligned} &\int_0^{\delta/c} \left(\mathcal{B}_1^{(1)}\right)^2 \mathrm{d}t + \int_{\delta/c}^{T} \left(\mathcal{B}_1^{(2)}\right)^2 \mathrm{d}t \\ &\quad \le \frac{\delta}{c} \int_{B_1} \beta_2\big(x_{\mathrm{R}} - c(t-s), s\big)^2 \,\mathrm{d}(s,t) + \frac{\delta}{c} \int_{B_2} \beta_2\big(x_{\mathrm{R}} - c(t-s), s\big)^2 \,\mathrm{d}(s,t) \\ &\quad = \frac{\delta}{c^2} \int_{\varphi(B_1)} \beta_2(x,\tau)^2 \,\mathrm{d}(x,\tau) + \frac{\delta}{c^2} \int_{\varphi(B_2)} \beta_2(x,\tau)^2 \,\mathrm{d}(x,\tau) \\ &\quad = \frac{\delta}{c^2} \int_{x_{\mathrm{L}}}^{x_{\mathrm{R}}} \int_0^{\frac{x - x_{\mathrm{L}}}{c}} \beta_2(x,\tau)^2 \,\mathrm{d}\tau \,\mathrm{d}x + \frac{\delta}{c^2} \int_{x_{\mathrm{L}}}^{x_{\mathrm{R}}} \int_{\frac{x - x_{\mathrm{L}}}{c}}^{T - \frac{x_{\mathrm{R}} - x}{c}} \beta_2(x,\tau)^2 \,\mathrm{d}\tau \,\mathrm{d}x \\ &\quad = \frac{\delta}{c^2} \int_{x_{\mathrm{L}}}^{x_{\mathrm{R}}} \int_0^{T - \frac{x_{\mathrm{R}} - x}{c}} \beta_2(x,\tau)^2 \,\mathrm{d}\tau \,\mathrm{d}x \\ &\quad \le \frac{\delta}{c^2} \|\beta_2\|^2_{\mathscr{L}_2([x_{\mathrm{L}}, x_{\mathrm{R}}] \times [0,T])}\,. \end{aligned} \tag{4.18}$$

Performing the same arguments one can show

$$\|\mathcal{B}_2\|^2_{\mathscr{L}_2([0,T])} = \int_0^{\delta/c} \left(\mathcal{B}_2^{(1)}\right)^2 \mathrm{d}t + \int_{\delta/c}^{T} \left(\mathcal{B}_2^{(2)}\right)^2 \mathrm{d}t \le \frac{\delta}{c^2} \|\beta_1\|^2_{\mathscr{L}_2([x_{\mathrm{L}}, x_{\mathrm{R}}] \times [0,T])}\,.$$

This yields with (4.17) and (4.18) and

$$\|\mathcal{B}\|^2_{\mathscr{L}_2([0,T])} = \|\mathcal{B}_1\|^2_{\mathscr{L}_2([0,T])} + \|\mathcal{B}_2\|^2_{\mathscr{L}_2([0,T])} \le \frac{\delta}{c^2} \|\beta\|^2_{\mathscr{L}_2([x_{\mathrm{L}}, x_{\mathrm{R}}] \times [0,T])}\,. \tag{4.19}$$

By combining (4.13), (4.14), (4.15), (4.16) and (4.19) we obtain the inequality (4.11).

Next, we show the first assertion. Without loss of generality we assume $t_{\mathrm{I}} = 0$. Note that (4.13) also holds in this case. Together with the previous definitions, we have

$$\left\| \begin{pmatrix} \omega_2(x_{\mathrm{R}}, \cdot\,) \\ \omega_1(x_{\mathrm{L}}, \cdot\,) \end{pmatrix} \right\|_{\mathscr{L}_2([0,T])} \le \|\mathcal{A}\|_{\mathscr{L}_2([0,\delta/c])} + \left\| \mathcal{B}^{(1)} \right\|_{\mathscr{L}_2([0,\delta/c])}\,.$$

The norm of $\mathcal{A}$ has been estimated in (4.15) and we have proven in (4.18) that

$$\int_0^{\delta/c} \left(\mathcal{B}_1^{(1)}\right)^2 \mathrm{d}t \leq \frac{\delta}{c^2} \int_{x_\mathrm{L}}^{x_\mathrm{R}} \int_0^{\frac{x-x_\mathrm{L}}{c}} \beta_2(x,\tau)^2 \,\mathrm{d}\tau \,\mathrm{d}x \leq \frac{\delta}{c^2} \left\|\beta_2\right\|^2_{\mathcal{L}_2([x_\mathrm{L},x_\mathrm{R}]\times[0,\delta/c])} .$$

With the same arguments one can show

$$\int_0^{\delta/c} \left(\mathcal{B}_2^{(1)}\right)^2 \mathrm{d}t \leq \frac{\delta}{c^2} \left\|\beta_1\right\|^2_{\mathcal{L}_2([x_\mathrm{L},x_\mathrm{R}]\times[0,\delta/c])} .$$

With these results and a similar argument as for assertion two, we obtain (4.10).

Finally, we show the third assertion. We build the inner product of ω and (4.7) to observe

$$\begin{aligned}\partial_t(\omega_1^2+\omega_2^2) + \partial_x(c\omega_2^2 - c\omega_1^2) &= 2\omega_1\beta_1 + 2\omega_2\beta_2 \\ &\leq \omega_1^2 + \beta_1^2 + \omega_2^2 + \beta_2^2 \quad \text{a.e. in } [x_\mathrm{L},x_\mathrm{R}]\times[t_\mathrm{I},t_\mathrm{E}]\,, \end{aligned} \tag{4.20}$$

where we have used the identities $\omega_i\partial_t(\omega_i) = \frac{1}{2}\partial_t(\omega_i^2)$, $\omega_i\partial_x(\omega_i) = \frac{1}{2}\partial_x(\omega_i^2)$ and the inequality $2\omega_i\beta_i \leq \omega_i^2 + \beta_i^2$.

Now, we consider the integral of (4.20) on the domain $[x_\mathrm{L},x_\mathrm{R}]\times[t_\mathrm{I},T]$ for $T\in[t_\mathrm{I},t_\mathrm{E}]$ and obtain

$$\begin{aligned}&\int_{x_\mathrm{L}}^{x_\mathrm{R}} \int_{t_\mathrm{I}}^{T} \partial_t(\omega_1^2+\omega_2^2) + \partial_x(c\omega_2^2 - c\omega_1^2)\,\mathrm{d}t\,\mathrm{d}x \\ &\quad= \int_{x_\mathrm{L}}^{x_\mathrm{R}} \omega_1(x,T)^2 + \omega_2(x,T)^2\,\mathrm{d}x - \int_{x_\mathrm{L}}^{x_\mathrm{R}} \omega_1(x,t_\mathrm{I})^2 + \omega_2(x,t_\mathrm{I})^2\,\mathrm{d}x \\ &\qquad + c\int_{t_\mathrm{I}}^{T} \omega_1(x_\mathrm{L},t)^2 - \omega_1(x_\mathrm{R},t)^2 + \omega_2(x_\mathrm{R},t)^2 - \omega_2(x_\mathrm{L},t)^2\,\mathrm{d}t \\ &\quad\leq \int_{t_\mathrm{I}}^{T} \int_{x_\mathrm{L}}^{x_\mathrm{R}} \omega_1^2 + \beta_1^2 + \omega_2^2 + \beta_2^2\,\mathrm{d}x\,\mathrm{d}t\,.\end{aligned}$$

Rearranging yields

$$\begin{aligned}&\int_{x_\mathrm{L}}^{x_\mathrm{R}} \omega_1(x,T)^2 + \omega_2(x,T)^2\,\mathrm{d}x \\ &\quad\leq \int_{t_\mathrm{I}}^{T} \int_{x_\mathrm{L}}^{x_\mathrm{R}} \omega_1^2 + \omega_2^2\,\mathrm{d}x\,\mathrm{d}t \\ &\qquad + c\int_{t_\mathrm{I}}^{T} \omega_1(x_\mathrm{R},t)^2 - \omega_1(x_\mathrm{L},t)^2 + \omega_2(x_\mathrm{L},t)^2 - \omega_2(x_\mathrm{R},t)^2\,\mathrm{d}t \\ &\qquad + \int_{x_\mathrm{L}}^{x_\mathrm{R}} \omega_1(x,t_\mathrm{I})^2 + \omega_2(x,t_\mathrm{I})^2\,\mathrm{d}x + \int_{t_\mathrm{I}}^{T} \int_{x_\mathrm{L}}^{x_\mathrm{R}} \beta_1^2 + \beta_2^2\,\mathrm{d}x\,\mathrm{d}t \\ &\quad\leq \int_{t_\mathrm{I}}^{T} \int_{x_\mathrm{L}}^{x_\mathrm{R}} \omega_1^2 + \omega_2^2\,\mathrm{d}x\,\mathrm{d}t \\ &\qquad + c\left\| \begin{pmatrix} \omega_1(x_\mathrm{R},\cdot) \\ \omega_2(x_\mathrm{L},\cdot) \end{pmatrix} \right\|^2_{\mathcal{L}_2([t_\mathrm{I},t_\mathrm{E}])} + \left\|\alpha\right\|^2_{\mathcal{L}_2([x_\mathrm{L},x_\mathrm{R}])} + \left\|\beta\right\|^2_{\mathcal{L}_2([x_\mathrm{L},x_\mathrm{R}]\times[t_\mathrm{I},t_\mathrm{E}])} .\end{aligned} \tag{4.21}$$

We continue by estimating the norm of ω at the boundary in (4.21). By iteratively applying the estimate (4.11) $\left\lfloor \frac{c(t_\mathrm{E}-t_\mathrm{I})}{\delta} \right\rfloor$ times and then the estimate (4.10) we deduce

$$\begin{aligned}
&\left\| \begin{pmatrix} \omega_1(x_\mathrm{R},\cdot) \\ \omega_2(x_\mathrm{L},\cdot) \end{pmatrix} \right\|_{\mathscr{L}_2([t_\mathrm{I},t_\mathrm{E}])} \\
&\quad \leq M \Big(\|\ell_\mathrm{R}\|_{\mathscr{L}_2([t_\mathrm{I},t_\mathrm{E}])} + c\,\|\ell_\mathrm{L}\|_{\mathscr{L}_2([t_\mathrm{I},t_\mathrm{E}])} + c^{-\frac{1}{2}}\,\|\alpha\|_{\mathscr{L}_2([x_\mathrm{L},x_\mathrm{R}])} + \delta^{\frac{1}{2}} c^{-1}\,\|\beta\|_{\mathscr{L}_2([x_\mathrm{L},x_\mathrm{R}]\times[t_\mathrm{I},t_\mathrm{E}])} \Big) \\
&\quad =: C_1\,,
\end{aligned}$$

for $M = \left\lceil \frac{c(t_\mathrm{E}-t_\mathrm{I})}{\delta} \right\rceil$. This yields

$$\int_{x_\mathrm{L}}^{x_\mathrm{R}} \omega_1(x,T)^2 + \omega_2(x,T)^2 \,\mathrm{d}x \leq C_2 + \int_{t_\mathrm{I}}^{T} \int_{x_\mathrm{L}}^{x_\mathrm{R}} \omega_1^2 + \omega_2^2 \,\mathrm{d}x\,\mathrm{d}t$$

with $C_2 = cC_1^2 + \|\alpha\|^2_{\mathscr{L}_2([x_\mathrm{L},x_\mathrm{R}])} + \|\beta\|^2_{\mathscr{L}_2([x_\mathrm{L},x_\mathrm{R}]\times[t_\mathrm{I},t_\mathrm{E}])}$. We apply Grönwall's inequality for integrals and obtain

$$\int_{x_\mathrm{L}}^{x_\mathrm{R}} \omega_1(x,T)^2 + \omega_2(x,T)^2 \,\mathrm{d}x \leq C_2 \exp(T - t_\mathrm{I})\,.$$

Finally, we can estimate the squared $\mathscr{L}_2$-norm of ω

$$\|\omega\|^2_{\mathscr{L}_2([x_\mathrm{L},x_\mathrm{R}]\times[t_\mathrm{I},t_\mathrm{E}])} = \int_{t_\mathrm{I}}^{t_\mathrm{E}} \int_{x_\mathrm{L}}^{x_\mathrm{R}} \omega_1(x,T)^2 + \omega_2(x,T)^2 \,\mathrm{d}x\,\mathrm{d}T \leq C_2 \left(\exp(t_\mathrm{E} - t_\mathrm{I}) - 1\right)\,. \tag{4.22}$$

The estimate (4.12) is now obtained by taking the root of (4.22) and with $\sqrt{C_2} \leq C$. □

Remark 1. The constant M measures the maximum number of times an information travels from one spatial boundary to the other. Any information that leaves the domain is reintroduced and continues to travel. The method does not guarantee that any condition is exactly satisfied. Therefore, errors are introduced at each boundary and as the information travels through the domain. Thus, the generalization error increases in proportion to the distance traveled. The dependence on M in the error estimate describes exactly this relationship.

This property is unique to systems with coupled boundary conditions. For example, this is not the case for a scalar transport equation, where the information leaves the domain and is not reintroduced.

To derive an estimate for $\mu = u - u^*$, we apply the third assertion of the Lemma.

Theorem 2. Let $c \geq 1$ and $M = \left\lceil \frac{c(t_\mathrm{E}-t_\mathrm{I})}{\delta} \right\rceil$. Then the following estimate holds

$$\begin{aligned}
\|u - u^*\|_{\mathscr{L}_2(D_\mathrm{EQ})} \leq \Bigg(&\sqrt{2} M c^{\frac{1}{2}} \left(\sqrt{L_\mathrm{R}} + c\sqrt{L_\mathrm{L}} \right) + c(M+1)\sqrt{L_\mathrm{I}} \\
&+ \left(M\sqrt{\delta c} + c \right) \sqrt{L_\mathrm{EQ}} \Bigg) \left(\exp(t_\mathrm{E} - t_\mathrm{I}) - 1 \right).
\end{aligned}$$

Proof. By construction we have

$$\|u - u^*\|_{\mathscr{L}_2(D_{\mathrm{EQ}})} = \|\mu\|_{\mathscr{L}_2(D_{\mathrm{EQ}})} = \|R\omega\|_{\mathscr{L}_2(D_{\mathrm{EQ}})} \leq \|R\|_2 \, \|\omega\|_{\mathscr{L}_2(D_{\mathrm{EQ}})} \,,$$

where we denote by $\|B\|_2$ for a matrix B the matrix norm induced by the $\|\cdot\|_2$ vector norm.

For $c \geq 1$ we calculate $\|R\|_2 = \sqrt{2}$ and $\left\|R^{-1}\right\|_2 = c/\sqrt{2}$. Hence, we have

$$\|\beta\|_{\mathscr{L}_2(D_{\mathrm{EQ}})} = \left\|-R^{-1}\ell_{\mathrm{EQ}}\right\|_{\mathscr{L}_2(D_{\mathrm{EQ}})} \leq \left\|R^{-1}\right\|_2 \|\ell_{\mathrm{EQ}}\|_{\mathscr{L}_2(D_{\mathrm{EQ}})} = \frac{c}{\sqrt{2}}\sqrt{L_{\mathrm{EQ}}}\,,$$
$$\|\alpha\|_{\mathscr{L}_2(D_{\mathrm{I}})} = \left\|-R^{-1}\ell_{\mathrm{I}}\right\|_{\mathscr{L}_2(D_{\mathrm{I}})} \leq \left\|R^{-1}\right\|_2 \|\ell_{\mathrm{I}}\|_{\mathscr{L}_2(D_{\mathrm{I}})} = \frac{c}{\sqrt{2}}\sqrt{L_{\mathrm{I}}}\,.$$

Now we insert (4.12) and obtain

$$\sqrt{2}\,\|\omega\|_{\mathscr{L}_2(D_{\mathrm{EQ}})} \leq \left(\sqrt{2}Mc^{\frac{1}{2}}\left(\sqrt{L_{\mathrm{R}}} + c\sqrt{L_{\mathrm{L}}}\right) + c(M+1)\sqrt{L_{\mathrm{I}}} \right.$$
$$\left. + \left(M\sqrt{\delta c} + c\right)\sqrt{L_{\mathrm{EQ}}}\right)\left(\exp(t_{\mathrm{E}} - t_{\mathrm{I}}) - 1\right). \qquad \square$$

The theorem shows a clear relationship between the losses L_{I}, L_{R}, L_{L}, L_{EQ} and the generalization error $\|u - u^*\|_{\mathscr{L}_2(D_{\mathrm{EQ}})}$. Minimizing the loss functions also minimizes the generalization error. In this sense, it validates the deviation of the physics-informed loss L_{DIF}.

We formulate some further remarks.

Remark 2 (Simulating real gas pipelines)**.** The error estimate becomes weaker for larger c or when the temporal domain is much longer than the spatial domain. For example, the linear problem has $c = 1$ and $M = 4$.

However, a more realistic scenario with a 50km gas pipeline simulated over a time interval of 24h with a speed of sound of 340m/s has $M \approx 588$. According to the theorem, $\sqrt{L_{\mathrm{EQ}}}$ must be minimized by additional $\log_{10}(M\sqrt{\delta c} + c) \approx 6.3$ orders to have the same influence on the result as in our linear problem.

This behavior has also been described multiple times in the literature for scalar linear transport equations, see for example [30, 37, 9]. The authors identified the characteristic speed (in our case the speed of sound) of the solution as the main factor for the accuracy of the results. They showed practically that for characteristic speeds of ten or higher, the generalization error increases. Therefore, we have chosen our examples deliberately to avoid this problem and we do not investigate this further.

Finally, there are publications that have successfully used physics-informed neural networks to simulate real-world scenarios, but they have not considered gas transport problems. See for example [27, 2].

Remark 3 (Loss balancing weights)**.** The losses in the estimate are not equally weighted, and their influence changes depending on c and M. This suggests that some losses have more influence and are therefore more important than others for the generalization error. This suggests a complicated interaction between the different losses. We will investigate this further in Subsection 4.1.3.

So far we have derived an estimate that is based on the values of the norms L_{EQ}, L_{I}, L_{L}, and L_{R}. However, in our implementation, we do not minimize the continuous integrals, but discrete approximations. Therefore, we turn our attention to this setting and denote by L_k^* the approximation of L_k for $k = \text{EQ}, \text{I}, \text{L}, \text{R}$. Note, L_k^* and $\overline{L}_k$ differ by the volume factor of the respective domain that is neglected in the derivation of L_{DIF}.

Corollary 1. Let M be defined as before. Assume $c \geq 1$,

$$\left|L_{\text{EQ}} - L_{\text{EQ}}^*\right| < \varepsilon_{\text{EQ}}\,, \quad |L_{\text{I}} - L_{\text{I}}^*| < \varepsilon_{\text{I}}\,, \quad |L_{\text{L}} - L_{\text{L}}^*| < \varepsilon_{\text{L}} \quad \text{and} \quad |L_{\text{R}} - L_{\text{R}}^*| < \varepsilon_{\text{R}}\,,$$

then it holds

$$\begin{aligned}
&\|u - u^*\|_{\mathcal{L}_2(D_{\text{EQ}})} \\
&\quad \leq \Bigg(\sqrt{2} M c^{\frac{1}{2}} \left(\sqrt{\varepsilon_{\text{R}} + |L_{\text{R}}^*|} + c\sqrt{\varepsilon_{\text{L}} + |L_{\text{L}}^*|} \right) \\
&\qquad + c(M+1)\sqrt{\varepsilon_{\text{I}} + |L_{\text{I}}^*|} + \left(M\sqrt{\delta c} + c \right) \sqrt{\varepsilon_{\text{EQ}} + \left|L_{\text{EQ}}^*\right|} \Bigg) \left(\exp(t_{\text{E}} - t_{\text{I}}) - 1 \right).
\end{aligned}$$

Proof. For example, by the triangle inequality we obtain

$$L_{\text{EQ}} = |L_{\text{EQ}}| = \left|L_{\text{EQ}} - L_{\text{EQ}}^* + L_{\text{EQ}}^*\right| \leq \varepsilon_{\text{EQ}} + \left|L_{\text{EQ}}^*\right|\,.$$

We proceed similarly for the other losses and then obtain the assertion by applying Theorem 2. □

Remark 4 (Error decomposition)**.** The corollary above decomposes the generalization error into two contributions: the quadrature errors (ε_k) and the training errors (L_k^*). The quadrature error is easily controlled by a converging quadrature rule and an increasing number of sample (or, quadrature) points. In the next section, 4.1.2, we investigate the quadrature error.

Controlling the training error is much harder. The training error can be decomposed into the optimization error, the error that occurs because the optimizer does not find the global minimum, and the approximation error, the error between the best approximation of the neural network with respect to the chosen architecture and the exact solution. Especially, Theorem 1 guarantees for any continuous function the existence of a sequence of neural network parameters, such that the corresponding neural networks converge to the function. In summary, the corollary shows that the convergence of physics-informed neural networks is possible, if we manage to minimize all errors arbitrarily. We will see later whether convergence can be observed in practice.

Additionally, traditional machine learning techniques suggest that one should not minimize the training error as much as possible, as that can increase the risk of overfitting and cause the approximation to decline eventually. Even the founding article of physics-informed neural networks mentions concerns about overfitting [36]. However, our results clearly show that lower training and quadrature errors result in lower generalization errors. This is in line with other findings in [49] and points to a deviation between traditional machine learning techniques and physics-informed machine learning.

4.1.1.d. Implementation

The loss of the balance law L_{EQ} sets the physics-informed neural networks apart from the solution-based loss function used in Chapter 3. As a consequence, L_{EQ} is the main building block enabling physics-informed neural networks. To evaluate L_{EQ}, we need to compute $\ell_{\text{EQ}}(x, t; \theta)$ for many points, and thus the efficient computation of ℓ_{EQ} is critical.

Usually, reverse mode automatic differentiation is used to obtain the derivatives in ℓ_{EQ}. For instance, the authors in [36] demonstrate a reverse mode implementation. This preference can be attributed to the fact that reverse mode differentiation is the primary approach of the widely used deep learning frameworks TensorFlow or PyTorch.

In this subsubsection, we explore the implementation of ℓ_{EQ} from a broader perspective. We examine three different implementations: two based on forward mode differentiation and one based on reverse mode differentiation. Finally, we compare the efficiency of the implementations.

For this presentation, we assume that the parameters θ are constant and thus $h(x, t) = h(x, t; \theta)$.

Forward mode differentiation. To compute ℓ_{eq} we need the time derivative of $\overline{u} \circ h$ and the spatial derivative of $\overline{F} \circ h$. By applying (3.10) these can be obtained by the Jacobian-Vector products

$$\partial_t(\overline{u} \circ h)(x, t) = J_{\overline{u} \circ h}(x, t) \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \text{and} \quad \partial_x(\overline{F} \circ h)(x, t) = J_{\overline{F} \circ h}(x, t) \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{4.23}$$

This leads us to the first implementation. The corresponding code is displayed in Listing 4.1a. This and the following implementations are based on the Jax [3] framework.

Fast forward mode differentiation. The previous implementation is not ideal and can be optimized by exploiting the structure of the derivatives in ℓ_{eq}. To perform similar computations only once, we split both Jacobian-Vector products into

$$\begin{aligned} J_{\overline{u} \circ h}(x, t) \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} &= J_{\overline{u}}(h_1, h_2) \cdot J_h(x, t) \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \text{and} \\ J_{\overline{F} \circ h}(x, t) \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} &= J_{\overline{F}}(h_1, h_2) \cdot J_h(x, t) \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \end{aligned}$$

with $(h_1, h_2) = h(x, t)$. Then, we compute $J_h(x, t) \cdot \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ using a vectorized version of the Jacobian-Vector product $v \mapsto J_h v$. This result is subsequently used to compute the remaining Jacobian-Vector products. See Listing 4.1b for the implementation.

Aside: Automatic code vectorization. The main difference between the implementations in Listing 4.1b and 4.1a is the vectorization of the Jacobian-Vector product. Automatic vectorization is a very powerful approach to simplify the generation of fast implementations. The automatic vectorization is not a loop (which would lead to the same implementation as in Listing 4.1a) and not a parallel execution, but a transformation of the function that replaces each operation with a vectorized one.

As a result, Listing 4.1b requires only one evaluation of h, while Listing 4.1a requires three evaluations. The derivatives of h are computed using the same operations, and thus, we traverse the computational graph of h only once.

In abstract terms, vectorization adds a new dimension to each input parameter and to the output of a function. For instance, the vector-matrix product

$$f\colon \mathbb{R}^k \to \mathbb{R}^k\,, \quad b \mapsto bA\,,$$

with $A \in \mathbb{R}^{k\times k}$ is vectorized into matrix-matrix product

$$\hat{f}\colon \mathbb{R}^{n\times k} \to \mathbb{R}^{n\times k}\,, \quad B \mapsto BA\,.$$

Usually, vectorization is performed manually. However, frameworks like Jax [3] provide automatic code vectorization, to simplify the process. In the second next paragraph, we will compare the efficiency of the non-vectorized code in Listing 4.1a with the vectorized code in Listing 4.1b.

Reverse mode differentiation. By reverse mode automatic differentiation, we obtain one row of the Jacobian, see (3.11). In order to compute $\partial_t(\overline{u}\circ h)$ and $\partial_x(\overline{F}\circ h)$, we need to compute the complete Jacobians $J_{\overline{u}\circ h}$, $J_{\overline{F}\circ h}$. Again, we split $J_{\overline{u}\circ h} = J_{\overline{u}}\cdot J_h$, $J_{\overline{F}\circ h} = J_{\overline{F}}\cdot J_h$ and vectorize the Vector-Jacobian products. Thus we compute

$$W_{\overline{u}} = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\cdot J_{\overline{u}}(x,t)\,, \qquad W_{\overline{F}} = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\cdot J_{\overline{F}}(x,t)\,,$$
$$\begin{pmatrix}\partial_x(\overline{F}\circ h) & \partial_t(\overline{F}\circ h)\\ \partial_x(\overline{u}\circ h) & \partial_t(\overline{u}\circ h)\end{pmatrix} = \begin{pmatrix}W_{\overline{F}}\\ W_{\overline{u}}\end{pmatrix}\cdot J_h(h_1,h_2)\,,$$

with $(h_1, h_2) = h(x,t)$. Importantly, only half of the computed derivatives are actually needed. See Listing 4.1c for the corresponding implementation.

Comparison. In terms of the number of Jacobian-Vector or Vector-Jacobian products that are required by the three implementations, the forward mode implementations are more efficient. To further quantify this comparison, we can measure the number of floating-point operations (FLOP) required to evaluate the gradient of $\ell_{eq}(x,t)$ with respect to the parameter θ, which is the most expensive operation in our application.

For this measurement, and all such measurements in this work, we rely on the data provided by the Jax framework. Note, that the estimated number of operations depends on the device and the applied algorithms. Additionally, the number of operations does not linearly map to the runtime. That is, increasing the number of operations by a factor of 10 does not imply that the runtime increases by a factor of 10. This is rooted in the underlying massively parallel computing architecture of GPUs. Therefore, the FLOP numbers presented should be interpreted as an approximation, and as a reference to compare the computational cost between different methods and/or implementations.

For a fully connected neural network with 5 hidden layers and 50 neurons, along with 48 000 sample points and the isentropic balance law, we obtain the following results. Implementation 4.1a requires 5.48×10^8 FLOP, implementation 4.1b requires 4.13×10^8 FLOP, and implementation 4.1c requires 7.39×10^8 FLOP on a Nvidia RTX A6000

```
def fwd_slow(self, x, t, h):
    def h_then_state(x_, t_):
        return self.state(*h(x_, t_))

    # Compute J_{ū∘h}(x,t) · (0 1)
    (_, _), (rho_t, rho_v_t) = jax.jvp(h_then_state, (x, t), (jnp.zeros_like(x),
                                                              jnp.ones_like(t)))

    def h_then_flux(x_, t_):
        return self.flux(*h(x_, t_))

    # Compute J_{F̄∘h}(x,t) · (1 0)
    (_, _), (f_1_x, f_2_x) = jax.jvp(h_then_flux, (x, t), (jnp.ones_like(x),
                                                           jnp.zeros_like(t)))

    g = self.rhs(*h(x, t))

    return jnp.square(rho_t + f_1_x) + jnp.square(rho_v_t + f_2_x - g)
```

(a) This function computes $\ell_{eq}(x, t; \theta)$ using the forward mode automatic differentiation.

```
def fwd_fast(self, x, t, h):
    v_x = jnp.stack((jnp.ones_like(x), jnp.zeros_like(t)), axis=0)
    v_t = jnp.stack((jnp.zeros_like(t), jnp.ones_like(t)), axis=0)

    def h_jvp(v_1, v_2):
        return jax.jvp(h, (x, t), (v_1, v_2))

    # Compute J_h(x,t) · (1 0 / 0 1)
    (((h_1, _), (h_2, _)), ((h_1_x, h_1_t),
                            (h_2_x, h_2_t))) = jax.vmap(h_jvp)(v_x, v_t)

    # Compute J_ū(h_1,h_2) · (∂_t h_1 / ∂_t h_2) and J_F̄(h_1,h_2) · (∂_x h_1 / ∂_x h_2)
    (_, _), (rho_t, rho_v_t) = jax.jvp(self.state, (h_1, h_2), (h_1_t, h_2_t))
    (_, _), (f_1_x, f_2_x) = jax.jvp(self.flux, (h_1, h_2), (h_1_x, h_2_x))
    g = self.rhs(h_1, h_2)

    return jnp.square(rho_t + f_1_x) + jnp.square(rho_v_t + f_2_x - g)
```

(b) Implementation of $\ell_{eq}(x, t; \theta)$ using a vectorized forward mode automatic differentiation to avoiding redundant calculations and group together similar operations.

Listing 4.1.: Three different implementations of $\ell_{eq}(x, t; \theta)$. For all implementations, `self.state` is the realization of $\overline{u}$ and `self.flux` the realization of $\overline{F}$. Furthermore, the function `jax.jvp` computes the Jacobian-Vector product, `jax.vjp` computes the Vector-Jacobian product and `jax.vmap` vectorizes functions. Continued on the next page.

```
def rev(self, x, t, h):
    # jax.vjp calculates h(x,t) and returns a function
    # that computes h_vjp: w ↦ w^⊤ J_h(x,t).
    (h_1, h_2), h_vjp = jax.vjp(h, x, t)

    # state_vjp: w ↦ w^⊤ J_ū(x,t)
    _, state_vjp = jax.vjp(self.state, h_1, h_2)
    # flux_vjp: w ↦ w^⊤ J_F̄(x,t)
    _, flux_vjp = jax.vjp(self.flux, h_1, h_2)

    w_1 = jnp.stack((jnp.ones_like(x),
                     jnp.zeros_like(x)), axis=0)
    w_2 = jnp.stack((jnp.zeros_like(x),
                     jnp.ones_like(x)), axis=0)

    # Compute (1 0; 0 1) · J_ū(h_1,h_2)
    w_state_1, w_state_2 = jax.vmap(state_vjp)((w_1, w_2))
    # Compute (1 0; 0 1) · J_F̄(h_1,h_2)
    w_flux_1, w_flux_2 = jax.vmap(flux_vjp)((w_1, w_2))

    w_h_1 = jnp.concatenate((w_state_1,
                             w_flux_1), axis=0)
    w_h_2 = jnp.concatenate((w_state_2,
                             w_flux_2), axis=0)

    # Compute (W_F̄; W_ū) · J_h(x,t)
    ((rho_x, rho_v_x, f_1_x, f_2_x),
     (rho_t, rho_v_t, f_1_t, f_2_t)) = jax.vmap(h_vjp)((w_h_1, w_h_2))

    g = self.rhs(h_1, h_2)

    return jnp.square(rho_t + f_1_x) + jnp.square(rho_v_t + f_2_x - g)
```

(c) Implementation of $\ell_{eq}(x, t; \theta)$ using reverse mode automatic differentiation.

Listing 4.1.: Three different implementations of $\ell_{eq}(x, t; \theta)$.

GPU. Hence, the second implementation uses about 56% and the first implementation uses about 75% of the FLOP required by the third implementation.

In summary, the second implementation 4.1b is the most efficient and provides a significant improvement over the reverse mode implementation that is usually employed. Therefore, we will only use this implementation.

4.1.1.e. Numerical Results

In this subsubsection, we perform numerical tests of the physics-informed neural networks introduced in this subsection. Our goal is to evaluate the performance of the two test problems described in Chapter 2. In addition, the results of these tests serve as a baseline for a comparison with the variants of physics-informed neural networks introduced later in this chapter.

Here, we use the neural network architectures that gave either the most accurate or the most efficient results in Chapter 3. They are listed in Table 4.1. Furthermore, the following optimization schemes are considered:

- Adam method. Here, we perform 100 000 iterations and use the exponentially decaying learning rate defined by (3.7). We test each of the following learning rate parameter combinations

  $$\eta_{\text{STEPS}} \in \{1000, 5000, 10000\}\,, \quad \eta_{\text{RATE}} = 0.9\,, \quad \eta_{\text{INIT}} \in \{0.1, 0.01, 0.001\}\,.$$

  From every obtained approximation, we select the approximation with the lowest loss value after training.

- L-BFGS method. Here we use the implementation from the SciPy [63] package, and we set the stop criterion to a gradient tolerance of 10^{-7}.

- A hybrid of the Adam and the L-BFGS method. Here, we first perform 10 000 Adam iterations and then run the L-BFGS method. This procedure was first introduced in [36] and is based on the idea to use a gradient-based method if the parameters are far away from the (local) optima and to use a higher-order method if the parameters are near. In this sense, the procedure is similar to the Levenberg-Marquardt method. For the Adam method, we test the same learning rates as described above. From every obtained approximation, we select the approximation with the lowest loss value after training.

To simplify the test setup, we use the same number of sample points for the initial data as well as the left and right boundary data. That is, we have

$$d_{\text{L}} = d_{\text{R}} = d_{\text{I}} =: d_{\text{BD}}\,.$$

The following combinations of the number of boundary sample points and the number of balance law sample points are included

$$d_{\text{BD}} \in \{2000, 4000, 8000\}\,, \quad d_{\text{EQ}} \in \{8000, 16000, 32000, 48000\}\,.$$

Linear			Isentropic		
σ	N	K	σ	N	K
relu	2	20	tanh	4	20
relu	2	30	tanh	6	30
relu	2	50	tanh	6	50
tanh	5	30	gelu	4	50
tanh	5	50	gelu	5	50
gelu	3	20	gelu	6	30
relu*	4	20	gelu*	5	20
relu*	6	50			

Table 4.1.: Neural network architectures for the linear and isentropic problem. An asterisk indicates a Hamiltonian-inspired neural network.

We conduct each test three times, each time with different randomly chosen sample points and initial parameters to mitigate the influence of randomness. Then, we calculate the average errors and report them.

Again, we measure the accuracy and efficiency of the results. Efficiency is measured by the ratio of the generalization error to the floating point operations (FLOP) required to obtain the result. Specifically, the efficiency for the parameter θ is measured by

$$\frac{\operatorname{error}\big[u_1;\theta_*\big] + \operatorname{error}\big[u_2;\theta_*\big]}{\operatorname{error}\big[u_1;\theta\big] + \operatorname{error}\big[u_2;\theta\big]} \cdot \frac{\mathrm{FLOP}_*}{\mathrm{FLOP}}, \tag{4.24}$$

where θ_* are the accurate reference parameters, FLOP are the number of floating point operations required to obtain θ and FLOP_* are the number of floating point operations required to obtain θ_*. As before, the floating point operations are measured by the Jax framework. Since we are using the L-BFGS method from the SciPy package, we can only give a very rough estimate of the FLOP required to obtain these results.

The following tables show the most accurate and the most efficient results for each neural network architecture and optimization method. See Table 4.2 for the results of the linear problem and Table 4.3 for the results of the isentropic problem.

For the linear problem, neural networks using the relu activation function do not have the necessary regularity to be trained with the L-BFGS method, so these tests were not performed. However, the results show that neural networks with a relu activation function, especially fully connected neural networks with two hidden layers, provide the best results. These results have a similar accuracy as those obtained in Chapter 3, where we trained with the exact solution. The other activation functions require significantly more parameters and sample points to produce worse results. In addition, we can save about half the computational cost for results that are an order of magnitude worse.

For the isentropic problem, the gelu activation function gives the most accurate results. However, the results are up to two orders of magnitude worse than the results of the linear problem and also worse than the results obtained in Chapter 3. The neural networks with the tanh activation function provide a cost-effective solution with only a small loss in accuracy. Here, we can save up to an order of magnitude in computational cost.

			Most accurate					Most efficient				
σ	N	K	d_{EQ}	d_{BD}	FLOP	error$[u_1]$	error$[u_2]$	d_{EQ}	d_{BD}	FLOP	error$[u_1]$	error$[u_2]$
relu	2	20	48000	2000	**1.6e12**	**1.93e−5**	**2.41e−5**	16000	4000	**8.7e11**	**2.39e−5**	**3.08e−5**
relu	2	30	16000	8000	**1.8e12**	**2.98e−5**	**2.97e−5**	16000	2000	**9.7e11**	**3.28e−5**	**4.14e−5**
relu	2	50	48000	2000	3.9e12	2.73e−4	3.77e−4	16000	2000	**1.6e12**	4.84e−4	4.26e−4
tanh	5	30	16000	8000	1.1e13	5.35e−3	6.66e−3	8000	2000	4.8e12	5.93e−3	7.40e−3
tanh	5	50	48000	2000	4.3e13	3.38e−3	4.17e−3	8000	4000	8.8e12	4.14e−3	5.16e−3
gelu	3	20	32000	2000	2.7e13	1.04e−2	1.31e−2	8000	2000	7.8e12	1.05e−2	1.31e−2
relu*	4	20	8000	2000	**1.9e12**	1.76e−4	2.22e−4	16000	2000	2.8e12	5.62e−4	6.29e−4
relu*	6	50	48000	8000	4.9e13	**8.86e−5**	**1.26e−4**	8000	2000	9.8e12	**1.35e−4**	**1.59e−4**

(a) Results obtained by the Adam method. The three lowest values per column are highlighted.

			Most accurate					Most efficient				
σ	N	K	d_{EQ}	d_{BD}	FLOP	error$[u_1]$	error$[u_2]$	d_{EQ}	d_{BD}	FLOP	error$[u_1]$	error$[u_2]$
tanh	5	30	48000	4000	5.7e13	1.28e−2	1.59e−2	8000	2000	4.0e12	2.14e−2	2.68e−2
tanh	5	50	16000	2000	**2.7e13**	**1.13e−2**	**1.41e−2**	8000	4000	**1.5e11**	2.01e−1	2.54e−1
gelu	3	20	32000	8000	5.9e13	1.14e−2	1.43e−2	8000	2000	8.7e12	**2.07e−2**	**2.59e−2**

(b) Results obtained by the L-BFGS method. The lowest value per column is highlighted.

			Most accurate					Most efficient				
σ	N	K	d_{EQ}	d_{BD}	FLOP	error$[u_1]$	error$[u_2]$	d_{EQ}	d_{BD}	FLOP	error$[u_1]$	error$[u_2]$
tanh	5	30	16000	8000	2.3e13	3.57e−3	4.47e−3	8000	2000	9.0e12	**4.68e−3**	**5.85e−3**
tanh	5	50	32000	8000	6.5e13	**2.75e−3**	**3.44e−3**	8000	4000	**9.3e11**	1.76e−2	2.19e−2
gelu	3	20	8000	8000	**2.1e13**	7.08e−3	8.80e−3	8000	2000	1.6e13	8.29e−3	1.03e−2

(c) Results obtained by the hybrid method. The lowest value per column is highlighted.

Table 4.2.: Results for the linear problem. An asterisk indicates a Hamiltonian-inspired neural network.

			Most accurate					Most efficient				
σ	N	K	d_{EQ}	d_{BD}	FLOP	error[ρ]	error[ρv]	d_{EQ}	d_{BD}	FLOP	error[ρ]	error[ρv]
tanh	4	20	16000	8000	**6.1e12**	1.45e−3	4.60e−3	8000	2000	**2.7e12**	1.51e−3	4.79e−3
tanh	6	30	16000	8000	**1.3e13**	1.10e−3	3.50e−3	8000	4000	**6.6e12**	1.34e−3	4.28e−3
tanh	6	50	32000	8000	3.9e13	1.06e−3	3.41e−3	8000	2000	**9.7e12**	1.28e−3	4.12e−3
gelu	4	50	16000	2000	4.8e13	**8.94e−4**	**2.84e−3**	8000	2000	2.6e13	**9.66e−4**	**3.02e−3**
gelu	5	50	8000	8000	4.7e13	**8.58e−4**	**2.69e−3**	8000	2000	3.3e13	**1.03e−3**	**3.24e−3**
gelu	6	30	32000	2000	8.3e13	**9.83e−4**	**3.12e−3**	8000	2000	2.4e13	**1.11e−3**	**3.51e−3**
gelu*	5	20	8000	4000	**2.3e13**	6.67e9	6.67e9	8000	2000	2.0e13	6.67e9	6.67e9

(a) Results obtained by the Adam method.

			Most accurate					Most efficient				
σ	N	K	d_{EQ}	d_{BD}	FLOP	error[ρ]	error[ρv]	d_{EQ}	d_{BD}	FLOP	error[ρ]	error[ρv]
tanh	4	20	48000	8000	3.3e13	**1.38e−3**	**4.41e−3**	8000	4000	5.4e12	**1.71e−3**	**5.52e−3**
tanh	6	30	16000	8000	**2.2e13**	**1.13e−3**	**3.61e−3**	8000	2000	7.9e12	**1.54e−3**	**4.96e−3**
tanh	6	50	8000	8000	**1.9e13**	**1.09e−3**	**3.57e−3**	8000	4000	**1.1e11**	1.97e−2	6.04e−2
gelu	4	50	48000	4000	1.4e14	1.86e−3	5.89e−3	8000	4000	**4.7e11**	1.85e−2	5.88e−2
gelu	5	50	32000	4000	1.3e14	1.53e−3	4.82e−3	8000	4000	**4.3e11**	2.06e−2	6.31e−2
gelu	6	30	16000	8000	5.4e13	1.68e−3	5.40e−3	8000	2000	1.4e13	**2.96e−3**	**9.23e−3**
gelu*	5	20	16000	2000	**4.5e12**	4.29e0	3.68e0	8000	2000	2.5e12	4.29e0	3.70e0

(b) Results obtained by the L-BFGS method.

			Most accurate					Most efficient				
σ	N	K	d_{EQ}	d_{BD}	FLOP	error[ρ]	error[ρv]	d_{EQ}	d_{BD}	FLOP	error[ρ]	error[ρv]
tanh	4	20	32000	8000	**2.1e13**	**1.10e−3**	**3.57e−3**	8000	2000	**3.5e12**	**1.44e−3**	**4.64e−3**
tanh	6	30	32000	8000	**3.4e13**	**1.02e−3**	**3.27e−3**	8000	2000	7.3e12	**1.11e−3**	**3.57e−3**
tanh	6	50	32000	8000	4.7e13	**9.95e−4**	**3.18e−3**	8000	4000	**1.1e12**	6.71e−3	2.08e−2
gelu	4	50	48000	2000	1.4e14	1.37e−3	4.29e−3	8000	4000	**3.0e12**	7.37e−3	2.26e−2
gelu	5	50	32000	4000	1.2e14	1.16e−3	3.70e−3	8000	8000	3.5e13	**1.32e−3**	**4.17e−3**
gelu	6	30	32000	8000	1.2e14	1.57e−3	4.96e−3	8000	4000	2.0e13	1.85e−3	5.85e−3
gelu*	5	20	8000	2000	**7.3e12**	5.45e0	5.22e0	16000	4000	1.4e13	6.99e0	4.04e0

(c) Results obtained by the hybrid method.

Table 4.3.: Results for the isentropic problem. An asterisk indicates a Hamiltonian-inspired neural network. The three lowest values per column are highlighted.

For both problems, neither the L-BFGS nor the hybrid method provides an accuracy or efficiency advantage over the Adam method. Therefore, only the Adam method is considered in the following numerical tests. However, we must emphasize that we repeated each Adam test nine times to find the best learning rate. This cost was not considered in the evaluation because it depends on the number of learning rates tested and can most likely be reduced by more sophisticated techniques.

The results of this numerical test are the baseline, and in the remainder of this chapter we introduce variants of the original physics-informed neural networks and test whether we can improve accuracy or efficiency. Here, we consider different sampling strategies in Subsection 4.1.2, loss balancing strategies in Subsection 4.1.3, and a loss function based on the integral form of the balance law in Section 4.2.

4.1.2. Sampling strategies

Random sampling strategies are widely used in machine learning. They prevent the model from learning only on specific training data and thus from overfitting. This procedure is enabled by stochastic gradient descent methods. Sometimes training data is augmented and thus can be extended indefinitely. For example, images can be cropped, scaled, rotated, mirrored, and so on.

In this perspective, the sets D_{EQ}, D_{I}, D_{L}, D_{R} are infinite sampling spaces and therefore also the possible training data. Following the machine learning path, physics-informed neural networks utilize random sampling strategies to effectively learn solutions of differential equations.

From a different perspective, we derived error estimates in the last section that showed the important role of quadrature errors. The quadrature errors are introduced by approximating the $\mathscr{L}_2$-norms, thus by considering $\overline{L}_{\text{EQ}}$, $\overline{L}_{\text{I}}$, $\overline{L}_{\text{L}}$,$\overline{L}_{\text{R}}$ instead of L_{EQ}, L_{I}, L_{L}, L_{R}. In this subsection, we take a closer look at quadrature methods that are based on random sampling and evaluate their performance in our use case. In particular, we will introduce the Latin hypercube sampling, the sampling strategy proposed in [36]. For an overview, see [42].

Specifically, for a square integrable function $f\colon [0,1]^d \to \mathbb{R}$ we consider approximations of the integral

$$F = \int_{[0,1]^d} f(x)\,\mathrm{d}x \approx \overline{F}\,.$$

These approximations are based on the following observation: Let X be a random variable distributed as $U([0,1]^d)$, denoted by $X \sim U([0,1]^d)$. Since the probability density function of X is equal to one, the expected value of $f(X)$ is given by

$$\mathbb{E}\big(f(X)\big) = \int_{[0,1]^d} f(x)\,\mathrm{d}x = F\,.$$

The quadrature rules, that we are about to present, construct estimates of the expected value $\mathbb{E}(f(X))$ and thus approximations of F. We begin with the Monte Carlo method.

4.1.2.a. Monte Carlo method

The Monte Carlo method draws random samples uniformly and takes the average of the corresponding function values. Consequently, let $X_1, ..., X_n \sim U([0,1]^d)$ be independent

random variables. Then, the Monte Carlo estimate of $\mathbb{E}(f(X))$ is defined by

$$\overline{F}_{\mathrm{MC}} = \frac{1}{n}\sum_{i=1}^{n} f(X_i)\,.$$

This estimate is unbiased because $\mathbb{E}(\overline{F}_{\mathrm{MC}}) = F$. Furthermore, since f is square integrable and $X \sim U([0,1]^d)$, we deduce by the identity of Bienaymé

$$\mathbb{V}\big(\overline{F}_{\mathrm{MC}}\big) = \frac{\mathbb{V}\big(f(X)\big)}{n} = \frac{\sigma^2}{n}\,,$$

for $\sigma^2 = \mathbb{V}(f(X))$. Since the Monte Carlo estimate is based on randomness, we investigate the expected error. Specifically, we consider the root mean square error and observe

$$\sqrt{\mathbb{E}\Big(\big(\overline{F}_{\mathrm{MC}} - F\big)^2\Big)} = \sqrt{\mathbb{V}\big(\overline{F}_{\mathrm{MC}}\big)} = \frac{\sigma}{\sqrt{n}}\,.$$

This error bound indicates a fundamentally different error composition in comparison to deterministic methods. Crucially, this bound is independent of the dimension d. This makes the Monte Carlo method the only suitable choice for large d, since for deterministic quadrature rules the number of quadrature points usually grows exponentially with the dimension. However, the error bound only decays at a rate of $\sqrt{n}$. This is slower than the usually considered deterministic quadrature rules. Therefore, the independence of the dimension is achieved by a very slow rate of convergence. The bound further depends on the variance σ^2 of $f(X)$. The error decreases for smaller variances σ^2.

Next, we introduce variations of the Monte Carlo estimate. These methods have in common that they trade randomness for determinism to achieve a faster error decay rate. We start with the Latin hypercube sampling.

4.1.2.b. Latin hypercube sampling

The Latin hypercube sampling (LHS) strategy divides every dimension into n equidistant intervals and draws a sample in each interval. A sample of the domain is obtained by randomly pairing samples of each dimension. Hence, we consider independent random variables $Y_{i,j} \sim U([0,1])$ for $i = 1,...,n$ and $j = 1,...,d$. Furthermore, let $\pi_1,...,\pi_d \colon \{0,...,n-1\} \to \{0,...,n-1\}$ be random permutations. Then, the i-th random sample of dimension j is defined by

$$\hat{Y}_{i,j} = \frac{\pi(i-1) + Y_{i,j}}{n} \quad \text{for} \quad i = 1,...,n \quad \text{and} \quad j = 1,...,d\,.$$

Consequently, the i-th random sample of $[0,1]^d$ is

$$X_i = \begin{pmatrix} \hat{Y}_{i,1} & \dots & \hat{Y}_{i,d} \end{pmatrix} \quad \text{for} \quad i = 1,...,n\,.$$

The Latin hypercube estimate is now defined by

$$\overline{F}_{\mathrm{LHS}} = \frac{1}{n}\sum_{i=1}^{n} f(X_i)\,.$$

As shown in [52], this estimate is again unbiased, $\mathbb{E}(\overline{F}_{\mathrm{LHS}}) = F$.

Estimating the variance of $\overline{F}_{\mathrm{LHS}}$ is more delicate. In [52] it is shown that for nearly additive functions, Latin hypercube sampling outperforms the classical Monte Carlo method. In particular, for $f_0 \in \mathbb{R}$ and specifically defined univariate $f_j(x)\colon \mathbb{R} \to \mathbb{R}$ the function

$$f_{\mathrm{ADD}}(x) = f_0 + \sum_{j=1}^{d} f_j(x_j)$$

is considered. Then, it is shown that

$$\mathbb{V}\Big(\overline{F}_{\mathrm{LHS}}\Big) \leq \frac{1}{n} \int_{[0,1]^d} (f - f_{\mathrm{ADD}})^2 \,\mathrm{d}x + o\left(\frac{1}{n}\right),$$

where $o(\,\cdot\,)$ is the little–o notation. This implies, if f is nearly additive, which is the case when $(f - f_{\mathrm{ADD}})^2$ is small, then the convergence rate improves. Finally, in [41] it was shown that in the worst case the variance of $\overline{F}_{\mathrm{LHS}}$ is only slightly larger than the variance of $\overline{F}_{\mathrm{MC}}$, since

$$\mathbb{V}\Big(\overline{F}_{\mathrm{LHS}}\Big) \leq \frac{\sigma^2}{n-1}\,.$$

In summary, LHS can improve the rate of convergence for some functions, while not being much worse than the Monte Carlo procedure. The method is also easy to implement and scales to very high dimensions. Therefore, the Latin hypercube sampling is a good default choice.

However, the scheme does not provide a generalization to other hypercubes than $[0,1]^d$. Since every dimension is divided into the same number of intervals, the best choice is to scale the samples from $[0,1]^d$ to the desired hypercube. This implies that for hypercubes with highly unequal dimension sizes, the longer dimensions are severely underrepresented. This is in contrast to traditional numerical methods. For example, to solve the nonlinear equation system that arises when simulating the nonlinear problem with the implicit box method, we need twelve times more time steps per unit than space points per unit. Therefore, we turn our attention to the stratified sampling strategy, which allows for a more even distribution of points, even for different side lengths.

4.1.2.c. Stratified sampling

The stratified sampling strategy divides the hypercube $[0,1]^d$ into m^d cubes of side length $1/m$. Then, one sample in each cube is drawn uniformly. We denote these samples by X_i for $i = 1, ..., m^d$. The stratified estimate is defined by

$$\overline{F}_{\mathrm{STR}} = \frac{1}{n} \sum_{i=1}^{n} f(X_i) \quad \text{for} \quad n = m^d\,.$$

Again, the estimate is unbiased, $\mathbb{E}(\overline{F}_{\mathrm{STR}}) = F$. It has been further shown in [32, 20] that

$$\mathbb{V}(\overline{F}_{\mathrm{STR}}) \leq \mathbb{V}(\overline{F}_{\mathrm{MC}})\,.$$

Consequently, the stratified sampling strategy is always as good as the standard Monte Carlo method. However, under the assumption that f is continuously differentiable, it has been shown with a Taylor expansion in [20] that

$$\mathbb{V}\left(\overline{F}_{\text{STR}}\right) \in \mathcal{O}\left(n^{-1-2/d}\right) ,$$

where $\mathcal{O}(\,\cdot\,)$ is the big–O notation. Unlike the other random strategies, the rate depends on the dimension d. Furthermore, the stratified strategy provides an improvement over the Monte Carlo procedure for small d. For example, the rate for the root mean square error is $1/n$ for $d = 2$.

This strategy has a generalization for hypercubes with arbitrary side lengths, since it allows different numbers of intervals per dimension. However, for large d, the advantage becomes smaller as the convergence rate increases and the implementation becomes harder. Finally, Latin hypercube and stratified sampling are identical for $d = 1$.

4.1.2.d. Sobol sequences

The last method we consider is Sobol sequences. A Sobol sequence is a sequence of deterministic points $x_1, ..., x_n \in [0, 1]^d$ with $n = 2^m$ such that the difference

$$F - \overline{F}_{\text{SOBOL}} = F - \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

is minimal, the points fill the space evenly and have no apparent pattern. Such methods are called quasi-Monte Carlo methods since they trade randomness for maximum speed of convergence. An error convergence rate of $\mathcal{O}(n^{-1}\log(n)^{d-1})$ has been proven for functions with a certain kind of bounded variation, see [43]. This convergence rate is better than the rate of the Monte Carlo method, but the rate increases for larger d.

Randomness can be reintroduced by scrambling the Sobol sequence, which is a random perturbation of the sequence while preserving its space-filling properties. It can be shown that for functions with sufficient regularity, the root mean square error decays as $\mathcal{O}(n^{-3/2}\log(n)^{(d-1)/2})$ [43]. This is the fastest convergence rate of the considered sampling strategies. However, again, the rate slows down for larger d.

Sobol sequences are only defined for hypercubes $[0, 1]^d$. Therefore, we need to scale the points to obtain a scheme for a general hypercube. Also, the number of samples can only be a power of two. As outlined in [43], dropping any points of the sequence eliminates the advantages of Sobol points. Sobol sequences are not trivial to implement, thus we use the implementation provided by SciPy [63].

For an example of the samples each strategy draws, see Figure 4.1. Note, that the Latin hypercube strategy draws one sample for each row and column, and the stratified strategy draws one sample per cell.

4.1.2.e. Convergence comparison

In this subsubsection, we practically compare the previously presented random-based quadrature rules with deterministic composite Gauss-Legendre rules by measuring the quadrature error of L_{EQ}. We consider two different instances of L_{EQ}: the linear balance

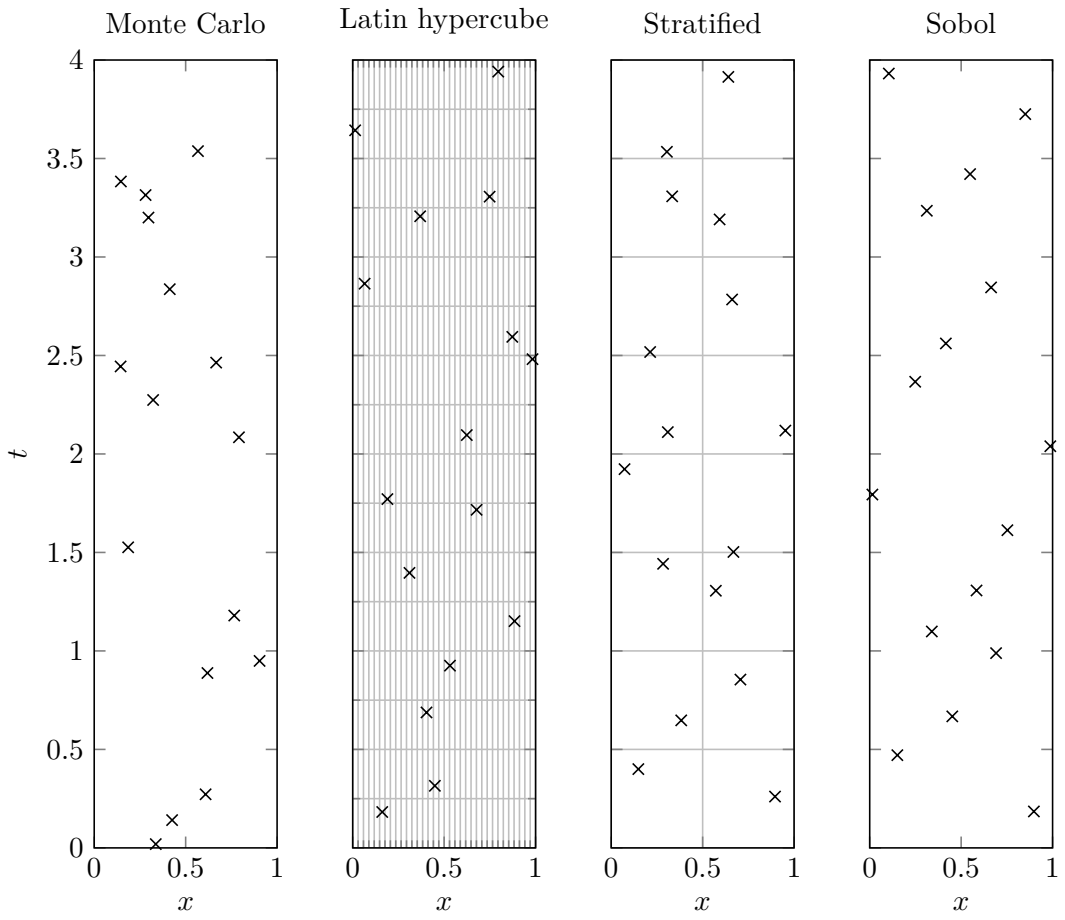


Figure 4.1.: Comparison of different sampling strategies when sampling 16 points in the domain $[0, 1] \times [0, 4]$.

law with a fully connected relu neural network, $N = 2$ and $K = 30$, and the isentropic balance law with a fully connected gelu neural network, $N = 5$ and $K = 50$.

This test uses the initial parameters of the neural networks. The error of the random quadrature rules is the root mean square error of 20 calculations. The error is computed with respect to a reference value that is calculated with 4 096 000 points. The results are computed using 32-bit floating point numbers and displayed in Figure 4.2.

We observe that the higher-order methods converge faster for the isentropic problem, and we can distinguish the different rates of convergence: The Monte Carlo and Latin hypercube sampling strategies have the slowest convergence rate. The stratified and Sobol strategies have roughly the same order of convergence as the Gauss-Legendre quadrature of order one. The higher-order Gauss-Legendre quadrature rules have the highest order of convergence. This can be attributed to the smoothness of the gelu activation function, which is transferred to L_{EQ}.

This is in contrast to the linear problem with the relu activation function. Here, the higher-order methods cannot show their advantage and are as fast as the Sobol or stratified sampling strategies.

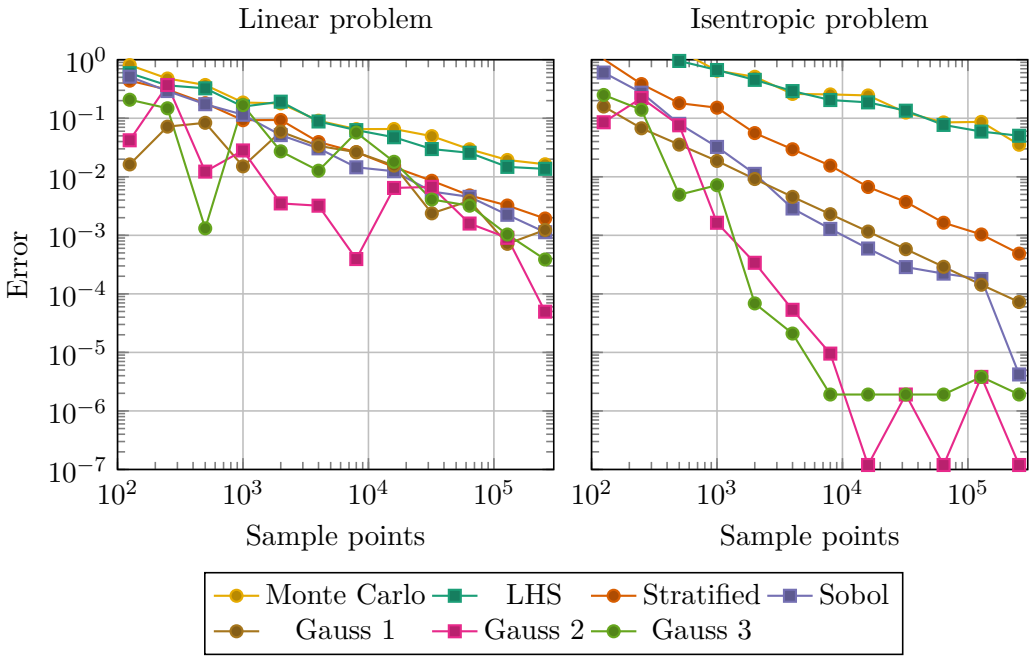


Figure 4.2.: Convergence analysis of different methods approximating $L_{\mathrm{EQ}}(\theta)$.

4.1.2.f. Numerical Results

In this subsubsection, we test the different sampling strategies presented in this subsection when used by physics-informed neural networks to solve the two test problems. In addition to the random-based quadrature rules, we also consider deterministic composite Gauss-Legendre rules. We use the Gauss-Legendre method with two quadrature points for the linear problem, and three quadrature points for the isentropic problem.

For the random-based quadrature rules, we consider the following two modes:

- Static. We draw the points in advance and do not change them during the training process. This is in line with the previous procedure and the original method [36].
- Dynamic [9]. We draw new points at each iteration. This is consistent with stochastic optimization, where samples are drawn from an infinite distribution.

Furthermore, we use the neural networks that performed well in the previous test. These are listed in Table 4.4. Again, we use the same number of sample points d_{BD} for the initial, the left and the right boundary data. For the Sobol points, we take the nearest power of two for the number of sample points. Similarly, for the composite Gauss-Legendre rules and the stratified sampling strategy, the number of cells is chosen so that the cells are approximately a square and the number of sample points is close to the target value.

We consider all combinations of the following parameters

$$d_{\mathrm{BD}} \in \{1000, 2000, 4000\}\,, \quad d_{\mathrm{EQ}} \in \{2000, 4000, 8000, 16000\}\,.$$

Linear			Isentropic		
σ	N	K	σ	N	K
relu	2	20	tanh	4	20
relu	2	30	tanh	6	30
tanh	5	30	gelu	5	50
relu*	6	50	gelu	6	30

Table 4.4.: Neural network architectures for the linear and isentropic problem. An asterisk indicates a Hamiltonian neural network.

The neural networks are trained for 100 000 Adam iterations with the following learning rate parameters

$$\eta_{\text{STEPS}} \in \{1000, 5000, 10000\}\,, \quad \eta_{\text{RATE}} = 0.9\,, \quad \eta_{\text{INIT}} \in \{0.1, 0.01, 0.001\}\,.$$

From these nine approximations, we select the approximation with the lowest loss value L_{DIF} after training. We run each test three times with different random values and only report the mean error.

See Table 4.5, which lists the most accurate and most efficient results for both test problems and each quadrature rule. As before, the efficiency is the ratio of the generalization error to the computational cost, defined by (4.24).

For the linear test, we obtain more accurate results at a lower computational cost when using Sobol points instead of Latin hypercube sampling. However, this difference is not noticeable in the most efficient results. Also, the difference between the most accurate and efficient results is rather small: we can double the error for half the computational cost. In addition, there is no advantage to dynamic sampling over static sampling.

The isentropic test requires fewer sample points than the linear test. Again, for this test, we see a slight improvement in the errors and sample points required when using an integration strategy other than Latin hypercube sampling. The most efficient results show that only a small number of sample points are required to achieve relatively accurate results. Again, we do not see a significant advantage of dynamic sampling.

4.1.2.g. Convergence comparison revisited

As a final step, we extend the convergence test performed in Subsection 4.1.2.e to physics-informed neural networks. We measure the generalization errors of the obtained physics-informed neural networks for the different sampling strategies and number of sample points for both test problems. In all tests, we set $d_{\text{BD}} = 4\,000$ and train for 100 000 Adam iterations. We try the same learning rates as in Subsubsection 4.1.2.f and only consider the result with the lowest loss value L_{DIF}. The reported error is the average error of three runs. The linear problem uses a relu neural network with $N = 2$ and $K = 30$. The isentropic problem uses a gelu neural network with $N = 5$ and $K = 50$.

The results are shown in Figure 4.3. While in Subsubsection 4.1.2.e we could distinguish the different quadrature rules by their convergence order, this is no longer possible. For the isentropic problem, all strategies perform almost identically, and only the number of sample points determines the generalization error. For the linear problem, we can

	Quadrature	σ	N	K	d_{EQ}	d_{BD}	FLOP	error[u_1]	error[u_2]
static	LHS	relu	2	20	16000	4000	8.7e11	2.39e−5	3.08e−5
static	Stratified	relu	2	20	16000	2000	**6.7e11**	4.23e−5	5.24e−5
static	Sobol	relu	2	30	4000	4000	**7.8e11**	**1.88e−5**	**2.36e−5**
static	Gauss 2	relu	2	20	16000	4000	9.1e11	**1.89e−5**	**2.41e−5**
dynamic	LHS	relu	2	30	2000	4000	**6.8e11**	3.33e−5	2.43e−5
dynamic	Stratified	relu	2	30	4000	4000	7.9e11	2.91e−5	3.54e−5
dynamic	Sobol	relu	2	20	16000	4000	9.2e11	**1.58e−5**	**1.97e−5**

(a) Most accurate results of the linear problem.

	Quadrature	σ	N	K	d_{EQ}	d_{BD}	FLOP	error[u_1]	error[u_2]
static	LHS	relu	2	20	4000	2000	**3.5e11**	5.32e−5	6.18e−5
static	Stratified	relu	2	20	4000	1000	**2.5e11**	1.03e−4	1.28e−4
static	Sobol	relu	2	20	4000	2000	**3.6e11**	**4.65e−5**	**5.78e−5**
static	Gauss 2	relu	2	20	4000	2000	3.7e11	5.79e−5	7.28e−5
dynamic	LHS	relu	2	30	4000	2000	5.1e11	**3.78e−5**	**4.69e−5**
dynamic	Stratified	relu	2	20	4000	2000	3.6e11	5.42e−5	7.01e−5
dynamic	Sobol	relu	2	20	8000	2000	5.3e11	**2.82e−5**	**3.48e−5**

(b) Most efficient results of the linear problem.

	Quadrature	σ	N	K	d_{EQ}	d_{BD}	FLOP	error[ρ]	error[ρv]
static	LHS	gelu	5	50	16000	4000	6.4e13	8.93e−4	2.81e−3
static	Stratified	gelu	5	50	2000	1000	**9.4e12**	8.75e−4	**2.66e−3**
static	Sobol	gelu	5	50	2000	1000	**9.6e12**	**8.29e−4**	**2.61e−3**
static	Gauss 3	gelu	5	50	16000	1000	5.6e13	8.69e−4	2.72e−3
dynamic	LHS	gelu	5	50	8000	2000	3.1e13	8.69e−4	2.74e−3
dynamic	Stratified	gelu	5	50	4000	2000	1.8e13	**8.64e−4**	**2.70e−3**
dynamic	Sobol	gelu	5	50	4000	1000	**1.6e13**	**8.66e−4**	2.73e−3

(c) Most accurate results of the isentropic problem.

	Quadrature	σ	N	K	d_{EQ}	d_{BD}	FLOP	error[ρ]	error[ρv]
static	LHS	tanh	4	20	2000	1000	**7.6e11**	**1.63e−3**	**5.18e−3**
static	Stratified	tanh	4	20	2000	1000	**7.7e11**	1.76e−3	5.58e−3
static	Sobol	tanh	4	20	2000	1000	7.8e11	1.70e−3	5.41e−3
static	Gauss 3	tanh	4	20	2000	1000	7.8e11	**1.60e−3**	**5.10e−3**
dynamic	LHS	tanh	4	20	2000	1000	**7.7e11**	1.66e−3	5.31e−3
dynamic	Stratified	tanh	4	20	2000	1000	7.7e11	1.76e−3	5.52e−3
dynamic	Sobol	tanh	4	20	2000	1000	7.9e11	**1.59e−3**	**5.12e−3**

(d) Most efficient results of the isentropic problem.

Table 4.5.: Sampling strategies test results. Each row shows the most accurate or efficient result per quadrature rule. The three lowest values per column are highlighted.

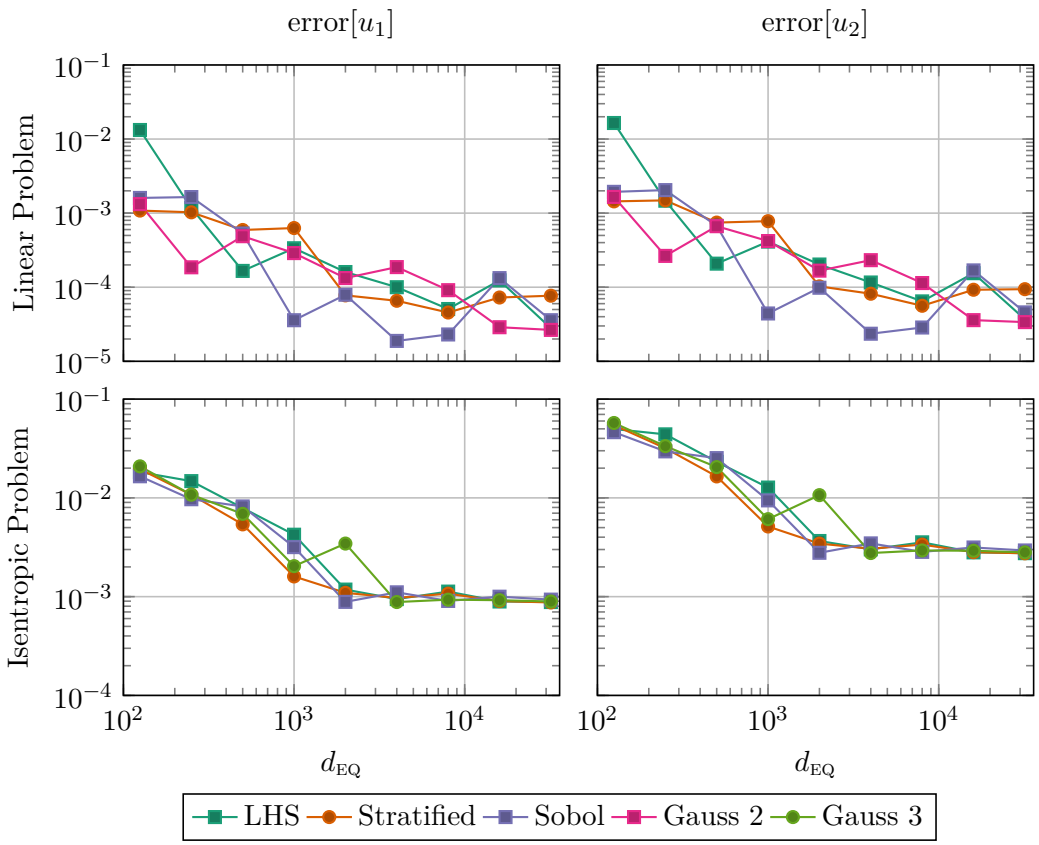


Figure 4.3.: Relative $\mathscr{L}_2$-errors for different sampling strategies and number of balance law sample points d_{EQ} when solving both test problems with physics-informed neural networks.

observe a small advantage of the Sobol points over the other strategies. This shows that the theory of the previous subsection, which decouples training and quadrature error, provides an incomplete picture and cannot fully describe the results obtained.

In the next subsection, we introduce loss balancing methods, which are another extension of physics-informed neural networks.

4.1.3. Loss balancing weighting

To train physics-informed neural networks, four loss functions are combined: $\overline{L}_{\mathrm{EQ}}$ for the balance law, $\overline{L}_{\mathrm{L}}$ for the left boundary condition, $\overline{L}_{\mathrm{R}}$ for the right boundary condition, and $\overline{L}_{\mathrm{I}}$ for the initial condition. They encode different properties, and each loss is necessary for a successful approximation. It appears that they are working together to achieve a common goal, but this is not the case. They compete for the approximation capabilities of the neural network, as they optimize for their own goal. As a result, the gradients of the losses do not necessarily point in the same direction.

From this viewpoint, it is remarkable that physics-informed neural networks actually work. This remains especially the case when we consider our system of coupled transport

equations: The information begins with the initial data, travels through the space-time domain, and is reintroduced at each boundary. Therefore, we are very dependent on each loss function to cooperate in the most beneficial way. In this subsection, we look at the interaction of the different losses and introduce loss balancing methods.

Weighted training problem. As briefly mentioned above, the training problem of physics-informed neural networks is a multi-objective optimization problem (4.6)

$$\min_{\theta} \left(\overline{L}_{\mathrm{EQ}}(\theta), \overline{L}_{\mathrm{I}}(\theta), \overline{L}_{\mathrm{L}}(\theta), \overline{L}_{\mathrm{R}}(\theta) \right).$$

Here, the conflicting objectives must be minimized simultaneously. This is in contrast to the training problem (4.4), where the sum of the objectives is minimized.

In general, there is no solution of (4.6) that minimizes all objective functions simultaneously. Therefore, we focus on Pareto optimal solutions, which are solutions that cannot be improved in any of the objectives without worsening at least one of the other objectives. Then, among all Pareto optimal solutions, one can decide which objectives are more important than others and choose the solution accordingly. The trade-offs between Pareto optimal solutions are visualized by the Pareto front. The Pareto front has been studied in the context of physics-informed neural networks in [48]. However, the study only focuses on one weight that mediates between the data residuals and the differential equation residual.

In our use case, visualizing the Pareto front is challenging and does not provide the information we need: Our primary concern is minimizing the generalization errors rather than achieving the lowest loss values. The error estimates suggest that lower loss values should imply reduced errors, however, each loss value affects the bound differently. This makes it challenging to describe the optimal prioritization.

Hence, we refocus on the more algorithmically tractable single-objective optimization problems. A prioritization between the objectives can be achieved by introducing weights into the loss function L_{DIF}. Here, we consider positive weights λ_{EQ}, λ_{I}, λ_{L}, λ_{R} for each objective and optimize the weighted sum of the weights and objectives. That is

$$\min_{\theta} L_{\mathrm{wDIF}} \quad \text{with} \quad L_{\mathrm{wDIF}} = \lambda_{\mathrm{EQ}} \overline{L}_{\mathrm{EQ}}(\theta) + \lambda_{\mathrm{I}} \overline{L}_{\mathrm{I}}(\theta) + \lambda_{\mathrm{L}} \overline{L}_{\mathrm{L}}(\theta) + \lambda_{\mathrm{R}} \overline{L}_{\mathrm{R}}(\theta). \tag{4.25}$$

The training problem (4.4), is only one possible instance of (4.25), where all weights are one. For each tuple of weights, we get another optimal solution, and changing the weights may improve one objective but can compromise another. The following example shows that choosing appropriate weights is critical.

Motivational example. Setting aside the theoretical challenges, selecting different weights can improve the results. See Figure 4.4 for an example. There, we compare two neural networks that were trained to approximate the linear simulation problem. The same sample points and initial parameters were used in the training process. Also, the relative $\mathscr{L}_2$-error was measured by the same sample points. However, different weights were used during the training process, which resulted in significantly reduced generalization errors. In the example, the weights have the following order: $\lambda_{\mathrm{EQ}} > \lambda_{\mathrm{I}} > \lambda_{\mathrm{L}} > \lambda_{\mathrm{R}}$. This shows that the balance law is the most important and that the right boundary is the least important.

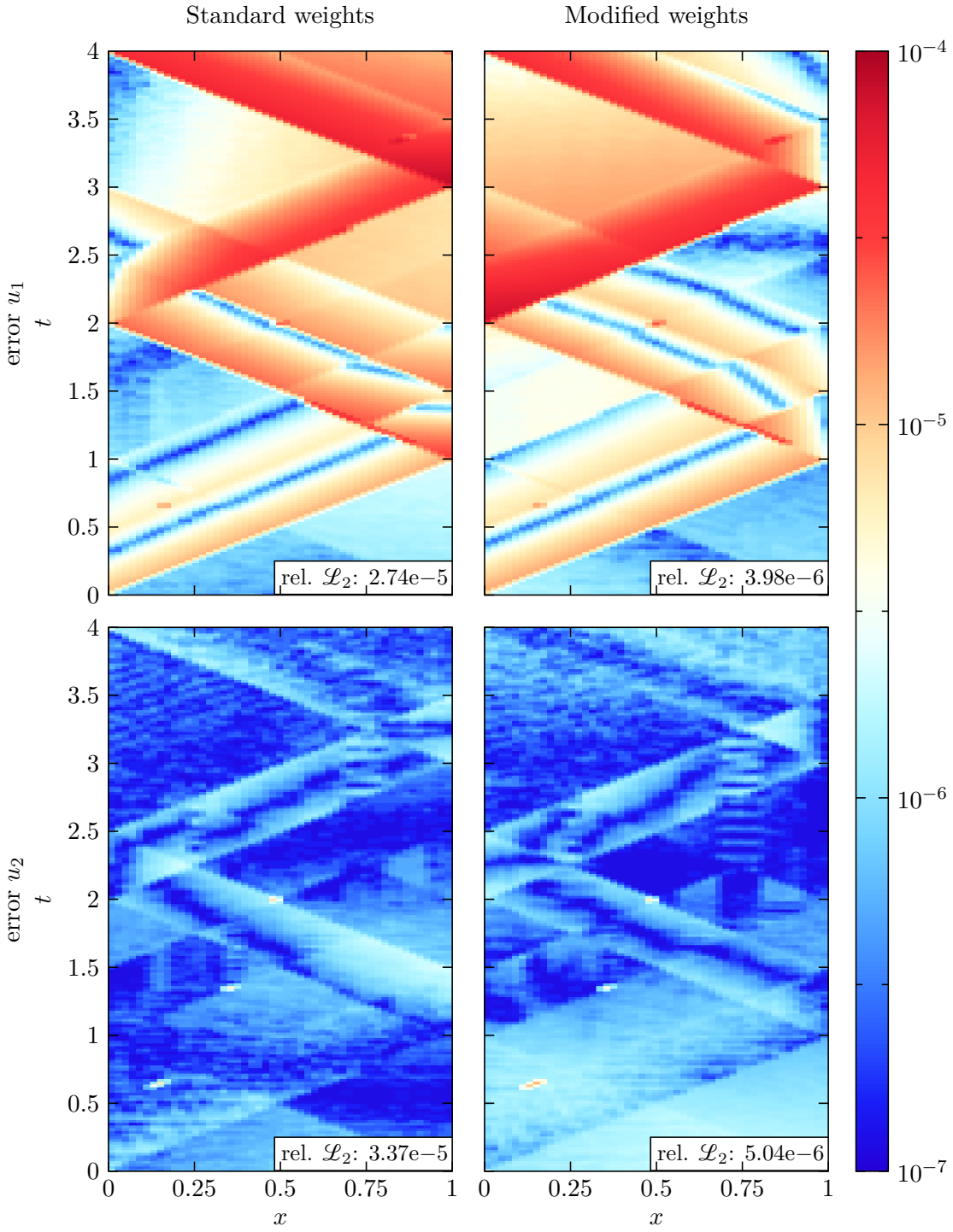


Figure 4.4.: The pointwise difference of two neural networks with $\sigma = \text{relu}$, $N = 2$, $K = 30$, and the exact solution for the linear problem are displayed. Both neural networks were trained with different loss weights. The left one was trained with unit weights and the right one with $\lambda_{\text{I}} = 2.25$, $\lambda_{\text{L}} = 3.69$, $\lambda_{\text{R}} = 1.12$, $\lambda_{\text{EQ}} = 4.65$.

Algorithmic approaches. In the previous example, the weights were not chosen arbitrarily but were determined through an algorithmic approach. Such techniques are often referred to as loss balancing methods in the literature. Loss balancing methods determine the weights λ_{EQ}, λ_{I}, λ_{L}, λ_{R} to achieve better approximations. Multiple studies have emphasized its significance and proposed various approaches [61, 64, 50]. We review some of these methods next.

We start with a random-search procedure in Subsubsection 4.1.3.a. This procedure is prohibitively expensive, so we turn our attention to computationally cheaper solutions. First, we will look at gradient-based balancing methods in Subsubsection 4.1.3.b. Second, we will look at an attention-based method in Subsubsection 4.1.3.c. After that, we will compare all approaches in Subsubsection 4.1.3.d.

4.1.3.a. Random-search

The results of the previous example were obtained by a random-search. This method is explained in this subsubsection. Using the Latin hypercube sampling, we generate a fixed number of random tuples $(\lambda_{\text{EQ}}, \lambda_{\text{I}}, \lambda_{\text{L}}, \lambda_{\text{R}}) \in [1,5] \times [1,5] \times [1,5] \times [1,5]$ and solve the training problem (4.25) for each tuple. Once each training has been conducted, we need to select the best approximation among all the approximations obtained. Here, we consider two different criteria: First, we use only the information available during training by selecting the approximation with the smallest loss value L_{DIF}. Second, we choose the approximation associated with the smallest sum of the generalization errors, $\text{error}[u_1] + \text{error}[u_2]$, and select the most accurate approximation.

A random-search is computationally very expensive since we have to solve the training problem multiple times. Therefore, we consider next loss balancing methods that determine the weights based on information available during the training process.

4.1.3.b. Gradient-based weighting

Gradient-based balancing methods determine the weights based on the gradients during the training process. This procedure is inspired by the ideas of the initialization schemes introduced in Chapter 3. There, we have discussed that initialization schemes consider the gradient entries of the loss and try to keep the mean close to zero and the variance close to one. The variance expresses the ability of the neural network parameters to minimize the loss. A variance closer to zero indicates that many elements are not very sensitive. Conversely, a higher variance indicates a greater number of parameters that are highly sensitive. In these cases, the loss is easier to minimize.

The variance depends on the specific loss function, and thus varies for each $\overline{L}_{\text{EQ}}$, $\overline{L}_{\text{I}}$, $\overline{L}_{\text{L}}$, $\overline{L}_{\text{R}}$. In their studies, the authors in [50, 64] identified these varying variances and interpreted them as indicators of imbalances between the different losses. They argue that such imbalances hinder successful training. Hence, they use the weights λ_{EQ}, λ_{I}, λ_{L}, λ_{R} to adjust each gradient such that the variances are approximately the same. This ensures that the losses are prioritized equally during the training process. We now describe the procedure in more detail.

In the following, we denote the empirical mean of a vector v by $\mu(v)$. Then $\mu(|\cdot|)$ measures the mean absolute deviation. For a vector with an empirical mean of zero,

this can be regarded as an approximation of the empirical variance. For this method, $\mu(|\nabla_\theta \overline{L}_{\text{EQ}}|)$ is the reference value. Therefore, we have $\lambda_{\text{EQ}} = 1$. The other gradients are scaled to have approximately the same mean absolute deviation. To achieve this, the intermediate weights are defined by

$$\hat{\lambda}_{\text{I}} = \frac{\mu\big(|\nabla_\theta \overline{L}_{\text{EQ}}(\theta)|\big)}{\mu\big(|\nabla_\theta \overline{L}_{\text{I}}(\theta)|\big)}\,, \quad \hat{\lambda}_{\text{L}} = \frac{\mu\big(|\nabla_\theta \overline{L}_{\text{EQ}}(\theta)|\big)}{\mu\big(|\nabla_\theta \overline{L}_{\text{L}}(\theta)|\big)}\,, \quad \hat{\lambda}_{\text{R}} = \frac{\mu\big(|\nabla_\theta \overline{L}_{\text{EQ}}(\theta)|\big)}{\mu\big(|\nabla_\theta \overline{L}_{\text{R}}(\theta)|\big)}\,. \tag{4.26}$$

During the optimization process, the intermediate weights fluctuate a lot and are therefore stabilized by a mean weighted average. This is similar to the approach used by the Adam optimizer. Consequently, let $\alpha \in (0, 1)$. Then, the weights are updated according to

$$\lambda_{\text{I}} \leftarrow (1-\alpha)\lambda_{\text{I}} + \alpha\hat{\lambda}_{\text{I}}\,, \quad \lambda_{\text{L}} \leftarrow (1-\alpha)\lambda_{\text{L}} + \alpha\hat{\lambda}_{\text{L}}\,, \quad \lambda_{\text{R}} \leftarrow (1-\alpha)\lambda_{\text{R}} + \alpha\hat{\lambda}_{\text{R}}\,. \tag{4.27}$$

Now, the method works as follows. First, the gradients $\nabla_\theta \overline{L}_{\text{EQ}}$, $\nabla_\theta \overline{L}_{\text{I}}$, $\nabla_\theta \overline{L}_{\text{L}}$, $\nabla_\theta \overline{L}_{\text{R}}$ are computed. Second, the weights are updated as defined by (4.26) and (4.27). Third, the weighted gradient

$$\nabla_\theta L_{\text{wDIF}} = \nabla_\theta \overline{L}_{\text{EQ}} + \lambda_{\text{I}} \nabla_\theta \overline{L}_{\text{I}} + \lambda_{\text{L}} \nabla_\theta \overline{L}_{\text{R}} + \lambda_{\text{L}} \nabla_\theta \overline{L}_{\text{R}}\,.$$

is built and used to update the parameters with a gradient descent step.

Usually, the weights $\lambda_{\text{EQ}}, \lambda_{\text{I}}, \lambda_{\text{L}}, \lambda_{\text{R}}$ are initially one. In [50] $\alpha = 0.9$ and in [64] $\alpha = 0.1$ is used. There is also a proposed variant that uses $\max(|\nabla_\theta \overline{L}_{\text{EQ}}|)$ in (4.26) instead of $\mu(|\nabla_\theta \overline{L}_{\text{EQ}}(\theta)|)$. However, this method was not successful in the study [54] and therefore we omit it here.

4.1.3.c. Attention-based weighting

In this subsubsection, we consider the attention-based mechanism that was introduced in [31]. Unlike the previous method, which determines weights based on the solution, this approach incorporates the weights directly into the training process. Originally, the authors proposed to assign one weight to each sample point in $\overline{D}_{\text{EQ}}$, $\overline{D}_{\text{I}}$, $\overline{D}_{\text{L}}$ and $\overline{D}_{\text{R}}$. However, this is associated with a very high computational cost. To avoid this, and for a more direct comparison, we modify this method to fit the problem (4.25).

For the following, we define $\lambda = (\lambda_{\text{EQ}}, \lambda_{\text{I}}, \lambda_{\text{L}}, \lambda_{\text{R}})$. Then, we focus on $L_{\text{wDIF}}(\theta, \lambda)$, the loss function associated with the training problem (4.25). Now, the goal of the weights λ is to penalize the losses $\overline{L}_{\text{EQ}}$, $\overline{L}_{\text{I}}$, $\overline{L}_{\text{L}}$, and $\overline{L}_{\text{R}}$ with the most significant values. This is accomplished by maximizing the loss function L_{wDIF} with respect to λ. Our overall goal remains to minimize L_{wDIF} with respect to the parameters θ. We, address both goals by considering the saddle point problem

$$\min_\theta \max_\lambda \; L_{\text{wDIF}}(\theta, \lambda)\,.$$

This problem can be solved by a gradient descent step for the parameters θ combined with a gradient ascent step for the weights λ. Specifically, we perform the update

$$\theta \leftarrow \theta - \nabla_\theta L_{\text{wDIF}}(\theta, \lambda) \qquad \text{and} \qquad \lambda \leftarrow \lambda + \nabla_\lambda L_{\text{wDIF}}(\theta, \lambda)\,.$$

At the beginning, we initialize the weights with $\lambda = (1, 1, 1, 1)$.

	σ	N	K	Quadrature rule	d_{BD}	d_{EQ}	η_{INIT}	η_{STEPS}
Most accurate	relu	30	2	Sobol	4000	8000	0.1	1000
Most efficient	relu	20	2	Stratified	1000	8000	0.01	1000

(a) Hyperparameter for the linear problem.

	σ	N	K	Quadrature rule	d_{BD}	d_{EQ}	η_{INIT}	η_{STEPS}
Most accurate	gelu	50	5	Stratified	1000	2000	0.01	10000
Most efficient	tanh	20	4	Stratified	1000	2000	0.01	10000

(b) Hyperparameter for the isentropic problem.

Table 4.6.: Most accurate and efficient hyperparameter identified in the previous tests.

4.1.3.d. Numerical results

In this subsubsection, we test the performance of the introduced loss balancing methods to solve the two simulation problems. We aim to improve the previous results by obtaining more efficient or more accurate approximations. To accomplish this, we select the hyperparameters that led to either the most accurate or the most efficient results in the previous tests, and apply each of the presented loss balancing methods. The hyperparameters include the neural network architectures and the training parameters, and are listed in Table 4.6 for the linear and the isentropic problem.

The gradient-based method is tested for $\alpha \in \{0, 1, 0.5, 0.9\}$. Training is conducted using the Adam method over 100 000 iterations. Each test is repeated three times and the average error is reported.

See Table 4.7a for the results of the linear problem and Table 4.7b for the results of the isentropic problem. We observe that using the random-search procedure consistently decreases the errors for both problems. The improvement is small when the weights are chosen based on the lowest L_{DIF} value. However, selecting the weights based on the smallest error leads to a substantial error reduction, particularly for the linear problem. This indicates that the selection of the weights is important, but optimal results are achieved using the solution itself, which is not practical.

The gradient-based method has the best results for $\alpha = 0.1$, but even these are worse than the standard weights. Therefore, the gradient-based method is not able to improve the results. The attention-based method only marginally improves the results for one test case. Consequently, neither approach offers significant advantages for our test problems.

In the following section, we present another physics-informed loss function that integrates the initial and boundary data, and thus mitigates the multi-objective optimization problem.

	Most accurate			Most efficient		
Method	FLOP	error$[u_1]$	error$[u_2]$	FLOP	error$[u_1]$	error$[u_2]$
standard weights	**1.0e12**	**2.30e−5**	**2.86e−5**	**4.0e11**	**5.10e−5**	**6.37e−5**
random-search L_{DIF}	**1.0e12**	**2.03e−5**	**2.55e−5**	**4.0e11**	**4.16e−5**	**5.18e−5**
random-search error	**1.0e12**	**4.87e−6**	**6.12e−6**	**4.0e11**	**2.41e−5**	**2.99e−5**
attention-based	1.0e12	1.55e−1	1.75e−1	4.0e11	8.54e−5	1.04e−4
gradient-based $\alpha = 0.1$	2.6e12	4.96e−1	6.10e−1	8.8e11	1.14e−2	1.50e−2
gradient-based $\alpha = 0.5$	2.6e12	4.85e−1	6.02e−1	8.8e11	1.07e−1	1.34e−1
gradient-based $\alpha = 0.9$	2.6e12	5.53e−1	7.20e−1	8.8e11	9.03e−2	1.14e−1

(a) Results of the linear problem.

	Most accurate			Most efficient		
Method	FLOP	error$[\rho]$	error$[\rho v]$	FLOP	error$[\rho]$	error$[\rho v]$
standard weights	**9.4e12**	**8.53e−4**	**2.59e−3**	**7.7e11**	1.74e−3	5.49e−3
random-search L_{DIF}	**9.4e12**	**8.12e−4**	**2.61e−3**	**7.7e11**	**1.43e−3**	**4.58e−3**
random-search error	**9.4e12**	**7.53e−4**	**2.30e−3**	**7.7e11**	**1.30e−3**	**4.09e−3**
attention-based	9.4e12	6.67e9	6.67e9	7.7e11	**1.70e−3**	**5.35e−3**
gradient-based $\alpha = 0.1$	2.1e13	2.33e−3	7.34e−3	3.0e12	5.40e−3	1.70e−2
gradient-based $\alpha = 0.5$	2.1e13	3.33e9	3.33e9	3.0e12	6.80e−3	2.16e−2
gradient-based $\alpha = 0.9$	2.1e13	3.33e9	3.33e9	3.0e12	4.76e−3	1.49e−2

(b) Results of the isentropic problem.

Table 4.7.: Results of the loss balancing tests. The three lowest values per column are highlighted.

4.2. Loss based on integral form

The physics-informed loss function L_{DIF}, introduced in the previous section, is based on the differential form of the balance law given by (1.1). In this section, we introduce a loss function that incorporates the integral form of the balance law. This results in a loss function that has different properties than the loss function L_{DIF}. This method was first proposed in [47].

Integral form. To reformulate the differential form of the balance law (1.1) into its integral form, we define the space-time flux function $G\colon \mathbb{R}^2 \to \mathbb{R}^{2\times 2}$ with

$$G(u) = \begin{pmatrix} F(u) & u \end{pmatrix} .$$

Using the space-time flux G we can rewrite the balance law (1.1) with the divergence

$$\partial_t u + \partial_x(F \circ u) = \partial_x(G_1 \circ u) + \partial_t(G_2 \circ u) = \operatorname{div}_{(x,t)}(G \circ u) \,. \tag{4.28}$$

Now, we integrate equation (4.28) on $D' \subset D_{\text{EQ}}$ with a piecewise smooth boundary and obtain by the divergence theorem

$$\int_{D'} \operatorname{div}_{(x,t)}(G \circ u) \,\mathrm{d}(x,t) = \int_{\partial D'} G(u) \cdot \vec{n} \,\mathrm{d}S \,,$$

where $\vec{n}$ denotes the outward-facing normal vector on the boundary $\partial D'$. Thus, if we assume that u fulfills (1.1), then

$$\int_{\partial D'} G(u) \cdot \vec{n} \, \mathrm{d}S = \int_{D'} g(u) \, \mathrm{d}(x,t) \quad \text{for all } D' \subset D_{\mathrm{EQ}} \text{ with piecewise smooth boundary.} \tag{4.29}$$

With sufficient regularity assumptions, the reverse also holds: If u satisfies (4.29), then u is also a solution of (1.1).

In the following, we define a loss function based on the integral form of the balance law (4.29). We start with the initial and boundary conditions.

Enforcing boundary conditions. In comparison to the differential form of the balance law (1.1), it is apparent that u needs less regularity to fulfill (4.29). It is therefore admissible to include the initial and boundary data naturally by altering the predicted state $\overline{u}$ of the neural network. For $\varepsilon = 10^{-6}$ let

$$\tilde{u}(h_1,h_2) = \begin{pmatrix} \tilde{u}_1 \\ \tilde{u}_2 \end{pmatrix}, \quad \tilde{u}_1 = \begin{cases} (b_{\mathrm{I}})_1(x) & \text{if } t \le t_{\mathrm{I}} + \varepsilon\,, \\ b_{\mathrm{L}}(t) & \text{if } x \le x_{\mathrm{L}} + \varepsilon\,, \\ \overline{u}(h_1,h_2) & \text{otherwise}\,, \end{cases} \quad \tilde{u}_2 = \begin{cases} (b_{\mathrm{I}})_2(x) & \text{if } t \le t_{\mathrm{I}} + \varepsilon\,, \\ b_{\mathrm{R}}(t) & \text{if } x \ge x_{\mathrm{R}} - \varepsilon\,, \\ \overline{u}(h_1,h_2) & \text{otherwise}\,, \end{cases}$$

where $(h_1,h_2) = h(x,t;\theta)$ is the output of the neural network. Compared to $\overline{u}$, $\tilde{u}$ always satisfies the initial, left, and right boundary conditions. We alter $\overline{F}$ and $\bar{g}$ in the same way, which leads to $\tilde{F}$ and $\tilde{g}$. In conclusion, we define the altered space-time flux $\tilde{G}$

$$\tilde{G}(h_1,h_2) = \begin{pmatrix} \tilde{F}(h_1,h_2) & \tilde{u}(h_1,h_2) \end{pmatrix}.$$

Control volume loss function. Now, we construct a loss function derived from the integral form of the balance law (4.29). The equation should be satisfied only with respect to a finite sequence of suitable subsets $D_1, \ldots, D_n \subset D_{\mathrm{EQ}}$. For these subsets, we aim to minimize the deviation from equation (4.29). Consequently, we consider

$$\min_{\theta} \; L_{\mathrm{INT}}(\theta)\,, \quad \text{with}$$

$$L_{\mathrm{INT}}(\theta) = \sum_{i=1}^{n} \left(\int_{\partial D_i} \tilde{G}\big(h(x,t;\theta)\big) \cdot \vec{n} \; \mathrm{d}S - \int_{D_i} \tilde{g}\big(h(x,t;\theta)\big) \, \mathrm{d}(x,t) \right)^2.$$

There are two remaining aspects to consider: the selection of the control volumes $D_1, \ldots, D_n$ and the approximation of the integrals in L_{INT} with quadrature rules.

Unlike the physics-informed approach, the authors of [47] recommend a deterministic quadrature rule. This was evaluated in [54], where deterministic rules gave the best results. Thus, we focus only on Gauss-Legendre quadrature rules with one, two, or three quadrature points to integrate the edges of ∂D_i. We extend the same rule to integrate the two-dimensional control volumes D_i.

The control volumes $D_1, \ldots, D_n$ are another degree of freedom in this method. In [47], the authors recommend partitioning D_{EQ} using either rectangles or triangles. Both

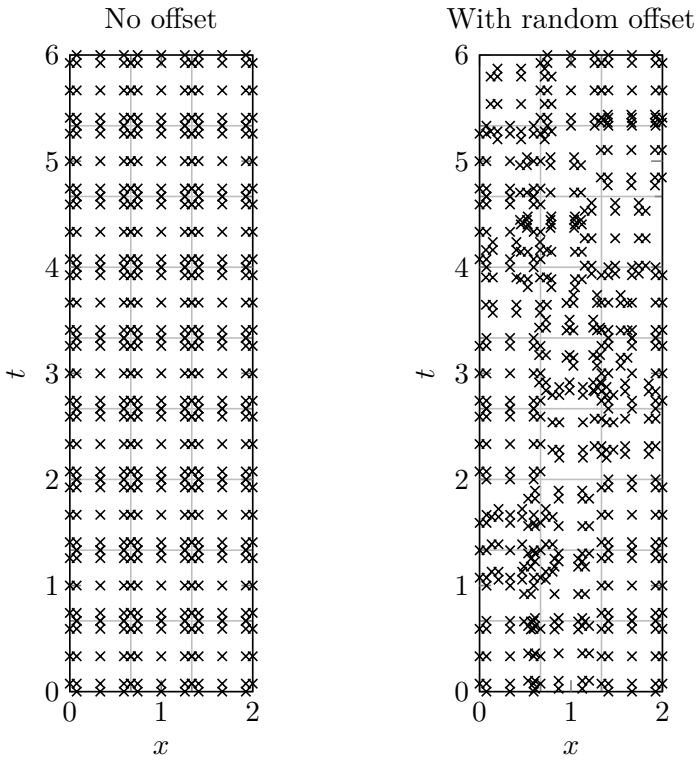


Figure 4.5.: Sampling points of the control volume physics-informed neural networks for the domain $[0, 2] \times [0, 6]$. Left for a regular partition and right for a regular partition with random offset. The x axis is divided into 3 intervals and the t axis is divided into 9 intervals. The sample points on the edges integrate the space-time flux G and the sample points in the interiors integrate the source g.

strategies were compared in [54], but neither was conclusively better. As a result, we exclusively use rectangular meshes.

In addition to the rectangular partition, which results in a regular pattern of sample points, we explore an alternative selection of control volumes and introduce randomness into the loss function. Suppose that $D_1, \dots, D_n$ is a rectangular partition of D_{EQ}. Then, we consider $D'_1, \dots, D'_n$, where each rectangle is slightly offset in a different random direction. We ensure that no rectangle is moved outside the domain D_{EQ} and consider two modes: maintaining a static offset throughout the training process or dynamically changing it in each iteration. See Figure 4.5 for a comparison between a partition and a randomly offset partition.

Comparison to PINNs. Most importantly, the loss function L_{INT} does not contain any derivatives. This leads to a considerable reduction in computational complexity. When the source term is zero, the dimension of the integration domain is reduced by one degree, again yielding significant efficiency gains. Otherwise, the space-time flux G and the source term g can be integrated with different quadrature rules.

By naturally incorporating the initial and boundary conditions, we have bypassed the

Mesh	σ	N	K	d_{INT}	q	FLOP	error$[u_1]$	error$[u_2]$
Partition	relu	2	30	**32000**	1	**1.9e12**	2.52e−3	2.95e−3
Static offset	relu	6	50	32000	2	1.3e13	3.39e−3	4.19e−3
Dynamic offset	relu	2	30	32000	2	2.0e12	**7.59e−4**	**9.34e−4**

(a) Most accurate results.

Mesh	σ	N	K	d_{INT}	q	FLOP	error$[u_1]$	error$[u_2]$
Partition	relu	2	30	16000	1	9.7e11	5.64e−3	7.29e−3
Static offset	relu	2	30	32000	1	2.0e12	3.62e−3	4.50e−3
Dynamic offset	relu	2	30	**8000**	3	**4.9e11**	**7.85e−4**	**1.00e−3**

(b) Most efficient results.

Table 4.8.: Results of the control volume physics-informed approach and the linear problem. The lowest value per column is highlighted.

Mesh	σ	N	K	d_{INT}	q	FLOP	error$[\rho]$	error$[\rho v]$
Partition	gelu	5	50	16000	1	1.3e13	1.05e−3	3.33e−3
Static offset	gelu	5	50	32000	1	2.6e13	1.03e−3	3.19e−3
Dynamic offset	gelu	5	50	**8000**	2	**6.4e12**	**8.60e−4**	**2.73e−3**

(a) Most accurate results.

Mesh	σ	N	K	d_{INT}	q	FLOP	error$[\rho]$	error$[\rho v]$
Partition	tanh	4	20	**2000**	1	**1.7e11**	4.00e−3	1.18e−2
Static offset	tanh	4	20	2000	1	1.9e11	7.85e−3	2.45e−2
Dynamic offset	tanh	4	20	2000	1	1.8e11	**2.39e−3**	**7.42e−3**

(b) Most efficient results.

Table 4.9.: Results of the control volume physics-informed approach and the isentropic problem. The lowest value per column is highlighted.

multi-objective optimization problem. The loss function L_{INT} consists only of a single objective function that enforces the same conditions for all control volumes D_i. Also, as with L_{DIF}, time is treated like any other dimension, and there is no explicit time discretization.

4.2.1. Numerical Results

In this subsection, we test the control volume physics-informed neural networks and compare their performance with the other results obtained in this chapter. We use the two test problems introduced in Chapter 2.

We denote the number of sample points with d_{INT} and the order of the Gauss-Legendre quadrature rule with q. The number of control volumes n is chosen depending on d_{INT} and q such that the number of sample points used is very close to d_{INT}. Furthermore,

since the source term g of the linear problem is always zero, we do not integrate this term and therefore do not need specific sample points. The implementation takes this into account.

We use the neural network architectures listed in Table 4.4 and consider the following combination of hyperparameters

$$d_{\text{INT}} \in \{2000, 4000, 8000, 16000, 32000\}\,, \quad q \in \{1, 2, 3\}\,.$$

The neural networks are trained with 100 000 Adam iterations, and we consider the following learning rate parameters

$$\eta_{\text{STEPS}} \in \{1000, 5000, 10000\}\,, \quad \eta_{\text{RATE}} = 0.9\,, \quad \eta_{\text{INIT}} \in \{0.1, 0.01, 0.001\}\,.$$

From these nine approximations, we select the approximation with the lowest loss value L_{INT} after training. To limit the influence of randomness, we repeat each test three times with different random values and report only the average error.

The results for the linear problem are listed in Table 4.8 and for the isentropic problem in Table 4.9. Most notably, dynamically offsetting the control volumes improves the results of this approach significantly. We get more accurate results, and in some cases, with fewer sample points. The best results of the isentropic problem have the same accuracy as the other results of this chapter, while requiring only two thirds of the computational cost. However, more sample points are required to achieve the results, which indicates that the derivative in the physics-informed approach provides more information. The results of the linear problem are significantly worse than the other results in this chapter.

In conclusion, our tests indicate that the results improve if the control volumes are not a partition of the domain. This choice was suggested in [47] and was most likely influenced by the finite volume method. However, control volume physics-informed neural networks work differently and therefore other considerations are necessary. Beyond the random offset of the control volumes, one could imagine control volumes that are randomly positioned, scaled, rotated, and skewed. This would free this approach from the partition. These changes could further improve the accuracy and may reveal the full potential of this very cost-effective approach.

4.3. Conclusion

At the beginning of this chapter, we looked at the fundamental properties of physics-informed neural networks, efficient implementations, and error estimates for a linear system of two transport equations. The estimates show that the generalization error is bounded by the quadrature and the training errors, the value of the loss functions $\overline{L}_{\text{EQ}}$, $\overline{L}_{\text{L}}$, $\overline{L}_{\text{R}}$, $\overline{L}_{\text{I}}$. The latter is composed of the optimization and the approximation error. The estimates indicate that problems with high characteristic speeds and simulations over long time spans pose particular challenges for physics-informed neural networks. However, for sufficiently small quadrature errors they do not suffer from overfitting.

Complementary to the theoretical analysis, we conducted extensive numerical tests to identify the most effective training strategies for physics-informed neural networks for two test problems introduced in Chapter 2. These experiments included different neural network architectures, sampling and optimization strategies, and loss function variants. Specific conclusions are outlined in the following paragraphs.

Linear problem. The linear problem has been approximated by highly accurate results, with a relative $\mathscr{L}_2$-error of 5×10^{-6}. This result actually surpasses those acquired in Chapter 3, where the training was conducted with the exact solution, and has been obtained with specific loss balancing weights. The error is potentially already limited by the 32-bit floating-point precision and is rather unexpected, since an investigation of different quadrature rules showed that the loss function L_{EQ} is more difficult to approximate for the linear problem than for the isentropic problem.

This can largely be attributed to the relu activation function, which is only continuous and piecewise linear. This inherent limited regularity is transferred to L_{EQ}, and thus a greater number of sample points is necessary. However, the relu activation function is very compatible with the solution of the linear problem, which is also continuous and piecewise linear. This provides an explanation for the small errors.

Isentropic problem. The isentropic problem did not yield errors as low as those of the linear problem: For ρ we have obtained a relative $\mathscr{L}_2$-error of 9×10^{-4} and for ρv of 3×10^{-3}. This shows that the nonlinear isentropic balance law poses a greater challenge than the linear balance law. This is further reinforced by the fact that we achieved this level of error through two distinct approaches: the physics-informed loss function L_{DIF} and the control volume physics-informed loss function L_{INT}.

Moreover, the errors are nearly an order of magnitude higher than those in Chapter 3, where the neural networks were trained with the exact solution. This indicates that there are parameters θ with lower relative $\mathscr{L}_2$-errors. However, we were not able to find them with a physics-informed approach. The limiting factors are unknown and a better theoretical understanding might help to overcome them. Furthermore, a different loss function that is based on reformulated Euler equations or a loss function that embeds more information about the nonlinear system might be promising. Crucially, improvements for the simpler problems could enable the simulation of real-world gas pipelines with a decent accuracy.

Also, smooth activation functions like tanh and gelu have yielded the best results. But we can assume that the exact solution of the isentropic problem is not smooth. It is thus challenging for neural networks with a smooth activation function to approximate the exact solution. However, the relu activation did not perform as good as the smooth alternatives. Therefore, activation functions that maintain a higher degree of regularity compared to relu, while not being smooth, might be beneficial.

Default hyperparameter. From a broader perspective, physics-informed neural networks apply successful deep learning methods to solve differential equations. Therefore, the training algorithms, neural network architectures, or sampling strategies are not specifically optimized to solve differential equations with high accuracy. They are optimized for tasks like image recognition or text generation, and there are certainly differences between classical machine learning tasks and differential equations.

Throughout this chapter, we have questioned the default hyperparameters for physics-informed neural networks with the goal of finding the most effective strategy, and we now outline the results.

Sampling strategies. We studied random-based sampling strategies, including the default Latin hypercube sampling. While the Latin hypercube sampling has some theoretical advantages over Monte Carlo sampling, strategies such as stratified sampling or Sobol points offer even higher convergence rates.

A convergence analysis has shown that the quadrature error can be easily minimized. Furthermore, if the neural network is sufficiently smooth, then higher-order quadrature rules converge faster. However, if the neural network is rather rough then higher-order methods have no benefit and a larger number of sample points is required. Importantly, we could not practically confirm that the convergence rate of the quadrature rule is transferred to the convergence rate of the physics-informed neural networks. Therefore, there were no significant advantages over the Latin hypercube sampling, and the decomposition of the generalization error into the quadrature error and the training error is not accurate enough to fully describe the generalization error of physics-informed neural networks.

Optimization strategies. In our tests, we explored several optimization strategies: the Adam method, the L-BFGS method, and a hybrid of both. The L-BFGS method features an integrated line search algorithm to find the optimal learning rate. In contrast, with the Adam method, we always tried different learning rates to obtain the best possible result.

Overall, the Adam method surpassed the other strategies when we trained for 100 000 iterations. However, these results were associated with a significantly higher computational cost as we tested different learning rates to find the most suitable one. This indicates that there is potential to develop optimization techniques tailored specifically for physics-informed neural networks.

Loss function variants. In our study, we also evaluated loss balancing techniques and the control volume loss function. Both are designed specifically for physics-informed neural networks.

For the former, a random-search procedure demonstrated that specific weights can improve the accuracy, but at a high computational cost. While other loss balancing methods are less expensive, they did not perform as well. For the latter, incorporating random offsets into the control volume approach enhanced the results, matching the performance of the physics-informed approach in one test case. Crucially, this was achieved with just two-thirds of the computational cost.

Outlook. In this chapter, we have covered a substantial subset of physics-informed neural networks, but many more research opportunities exist. Our findings suggest that physics-informed neural networks can solve differential equations with adequate accuracy, making them suitable for various applications, even though they are not a high-precision method. In the next chapter, we will build on these results to solve optimal control problems.

5. Physics-informed optimization

In the previous chapter, we studied physics-informed neural networks and conducted a thorough analysis. It became evident that physics-informed neural networks are neither a highly accurate simulation method nor a super fast algorithm. However, the strength of this approach lies in the flexibility. This allows to address problems that are difficult for classical methods. This is illustrated with numerous examples in the literature, such as solving inverse problems, discovering differential equations, or simulating high-dimensional physical problems. For an overview see [25].

In this chapter, we demonstrate the flexibility of the physics-informed approach with a gas transport example. Specifically, we focus on optimal boundary control problems. Previous research in [29, 11] approached such problems with a classical approach that decouples the simulation and optimization. To this end, this method unfolds in a loop of simulation, sensitivity calculation regarding the control (requiring another simulation), followed by an update of the control. Thus, the method necessitates many simulations, which are very costly for large gas networks.

Therefore, we want to explore different approaches that do not decouple the simulation and optimization. To achieve this, we utilize the physics-informed approach and the knowledge gained in the previous chapter. As a test case, we focus on optimal control problems with a single pipe. The specific problems are based on the simulation problems introduced in Chapter 2.

This chapter is structured as follows. In Section 5.1, we define the optimal control problem and introduce two specific instances. Then, in Section 5.2, we introduce a direct approach to solve this problem. However, this approach shows complications with our problems. To address these, we develop an indirect adjoint-based approach in Section 5.3. Finally, we conclude this chapter in Section 5.4.

5.1. Optimal control problems

Optimal control problems consist of several components. They include a simulation problem represented by a differential equation and additional initial and boundary conditions. Importantly, this simulation problem is influenced by a control. This control can vary the initial condition, the boundary condition, or the differential equation. The control thus affects the solution of the simulation problem. The objective function depends on this solution and is thus indirectly changed by the control. The goal is to find a control that minimizes the objective function. For further information on optimal control problems, see [59, p. 3].

In this chapter, we focus on optimal boundary control problems. As before, we consider a state vector $u(x,t)\colon D_{\text{EQ}} \to \mathbb{R}^2$ that should adhere to a balance law in a pipe. We control the left boundary data of the first state variable $u_1(x_{\text{L}},t)$ with a control function

$c(t)\colon D_{\mathrm{L}} \to \mathbb{R}$. Our aim is to match a target at the right boundary of the first state variable $u_1(x_{\mathrm{R}}, t)$. This target is denoted by $u_{\mathrm{OBJ}}(t)\colon D_{\mathrm{R}} \to \mathbb{R}$. Our objective function measures the distance $u_1(x_{\mathrm{R}}, t) - u_{\mathrm{OBJ}}(t)$ in the $\mathscr{L}_2(D_{\mathrm{R}})$-norm, as well as the deviation $c(t) - c_{\mathrm{OBJ}}(t)$ in the $\mathscr{L}_2(D_{\mathrm{L}})$-norm. The latter serves as a regularization term and also as a cost function that evaluates the cost associated with c deviating from c_{OBJ}. This cost is scaled by a factor $\lambda > 0$.

Consequently, we are concerned with the following optimal control problem

$$\min_{u(x,t),\, c(t)} \quad \frac{1}{2} \left\| u_1(x_{\mathrm{R}}, \cdot) - u_{\mathrm{OBJ}}(\cdot) \right\|^2_{\mathscr{L}_2(D_{\mathrm{R}})} + \frac{\lambda}{2} \left\| c(\cdot) - c_{\mathrm{OBJ}}(\cdot) \right\|^2_{\mathscr{L}_2(D_{\mathrm{L}})} \tag{5.1a}$$

$$\text{s.t.} \qquad \partial_t u + \partial_x (F \circ u) = g \circ u \quad \text{in } D_{\mathrm{EQ}} \tag{5.1b}$$

$$u(\cdot, t_{\mathrm{I}}) = b_{\mathrm{I}}(\cdot) \quad \text{on } D_{\mathrm{I}} \tag{5.1c}$$

$$u_1(x_{\mathrm{L}}, \cdot) = c(\cdot) \quad \text{on } D_{\mathrm{L}} \tag{5.1d}$$

$$u_2(x_{\mathrm{R}}, \cdot) = b_{\mathrm{R}}(\cdot) \quad \text{on } D_{\mathrm{R}}\,, \tag{5.1e}$$

where the objective function is given by (5.1a) and the simulation problem by (5.1b) – (5.1e) for a flux function $F(u)$, a source term $g(u)$, initial boundary data $b_{\mathrm{I}}(x)\colon D_{\mathrm{I}} \to \mathbb{R}^2$ and right boundary data $b_{\mathrm{R}}(t)\colon D_{\mathrm{R}} \to \mathbb{R}$. In this chapter, we denote by $u_c(x,t)$ the solution of the simulation problem (5.1b) – (5.1e) with respect to a control $c(t)$.

Usually, two approaches are considered to solve an optimal control problem: either directly or indirectly. The former approach minimizes the objective of (5.1) directly. The latter solves derived optimality conditions and thus approaches the problem (5.1) indirectly. In this chapter, we translate both approaches into a physics-informed formulation and then study the advantages and the disadvantages through numerical tests.

Before that, we introduce two instances of the optimal control problem (5.1). Both problems are based on a simulation problem introduced in Chapter 2.

Linear problem. This optimal control problem is based on the linear simulation problem defined in Chapter 2. The right boundary $b_{\mathrm{R}}(t)$ and the initial condition $b_{\mathrm{I}}(x)$ remain unchanged. This optimal control problem is specifically designed so that the left boundary data $b_{\mathrm{L}}(t)$ of the linear simulation problem is the exact control

$$\begin{aligned} c^*(t) = b_{\mathrm{L}}(t) = \iota(t; 0.0, \quad 0.5, \quad 1.0, 1.5, 2.0, \quad 2.5, 3.0, 4.0; \\ 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 0.0, 0.0)\,, \end{aligned}$$

where ι is the continuous piecewise linear interpolation defined by (2.2). We achieve this with a target function $u_{\mathrm{OBJ}}(t)$ that is equal to the exact solution of the simulation problem $u_{b_{\mathrm{L}}}(x,t)$ at the right boundary. Hence, $u_{\mathrm{OBJ}}(t) = (u_{b_{\mathrm{L}}})_1(x_{\mathrm{R}}, t) = (u_{c^*})_1(x_{\mathrm{R}}, t)$, or specifically

$$\begin{aligned} u_{\mathrm{OBJ}}(t) = \iota(t; 0.0, 1.0, \quad 1.5, \quad 2.0, 2.5, 3.0, 3.5, 4.0; \\ 0.0, 0.0, -1.0, -1.0, 1.0, 1.0, 0.0, 0.0)\,. \end{aligned}$$

Since the length of the spatial domain is one, the speed of the characteristics is one, and the length of the time interval is four, any control $c(t)$ does not affect the right boundary for $t > 3$. According to the definition of $b_{\mathrm{L}}(t)$, the control should be zero for $t > 3$, therefore we set $c_{\mathrm{OBJ}}(t) = 0$. Lastly, we set $\lambda = 5 \cdot 10^{-3}$.

Nonlinear problem. This problem is based on the isentropic simulation problem defined in Chapter 2. The right boundary $b_{\mathrm{R}}(t)$ and the initial condition $b_{\mathrm{I}}(x)$ remain unchanged. As described in the definition of the simulation problem, we want to keep the density ρ at the right boundary constant. Thus, we set $u_{\mathrm{OBJ}}(t) = (b_{\mathrm{I}})_1(x_{\mathrm{R}}) \approx 1.93$. We also set $c_{\mathrm{OBJ}}(t) = 2$ to measure the cost of changing the density at the left boundary. These costs are scaled by $\lambda = 5 \cdot 10^{-2}$. Therefore, the optimization method must decide between cost minimization and goal satisfaction at the right boundary.

5.2. Direct approach

This approach directly minimizes the objective function (5.1a) of the optimal control problem (5.1). This works by introducing a parameterized control into the physics-informed training problem (4.4) and by adding the objective function to the loss function. The approach has been described in [38]. A similar approach has been described in [14].

The state vector $u(x,t)$ is approximated by a neural network with randomly initialized parameters θ_u. Consequently, we have $u(x,t) \approx \overline{u}(h(x,t;\theta_u))$.

The control $c(t)$ is parameterized by a continuous piecewise linear function defined through (2.2). We consider d_c nodes, denoted as ϑ_k for $k = 1, ..., d_c$, each associated with a function value φ_k. The nodes are initially equidistant, thus we have

$$\vartheta_k = t_{\mathrm{I}} + (k-1)\frac{t_{\mathrm{E}} - t_{\mathrm{I}}}{d_c - 1} \qquad \text{and} \qquad \varphi_k = u_{\mathrm{OBJ}}(\vartheta_k) \qquad \text{for} \quad k = 1, ..., d_c\,.$$

The first and the last nodes, as well as the first weight, cannot be changed during the optimization process. In our numerical tests, we aim to differentiate between training only the weights φ_k and simultaneously training both the weights and the nodes ϑ_k. Therefore, the parameters for the control are given either by

$$\theta_c = (\vartheta_2, ..., \vartheta_{d_c-1}, \varphi_2, ..., \varphi_{d_c}) \qquad \text{or} \qquad \theta_c = (\varphi_2, ..., \varphi_{d_c})\,.$$

This allows for the control of either $d_c - 1$ weights and $d_c - 2$ nodes, or only $d_c - 1$ weights. Finally, we define the parameterized control

$$\bar{c}(t;\theta_c) = \iota(t;\vartheta_1, ..., \vartheta_{d_c};\varphi_1, ..., \varphi_{d_c})\,. \tag{5.2}$$

Note that [14] removes the control from the system and recovers the control from the state variables. For our problem this could be achieved with $c(t) = \overline{u}_1(h(x_{\mathrm{L}}, t;\theta_u))$.

We define the physics-informed loss function with respect to the simulation problem (5.1b) – (5.1e)

$$\begin{aligned} L_{\mathrm{DIF}}(\theta_u,\theta_c) \approx{}& \frac{1}{\mathrm{vol}(D_{\mathrm{EQ}})}\Big\|\partial_t(\overline{u}\circ h)(\,\cdot\,,\,\cdot\,;\theta_u) \\ &\qquad + \partial_x(\overline{F}\circ h)(\,\cdot\,,\,\cdot\,;\theta_u) - (\overline{g}\circ h)(\,\cdot\,,\,\cdot\,;\theta_u)\Big\|^2_{\mathscr{L}_2(D_{\mathrm{EQ}})} \\ &+ \frac{1}{\mathrm{vol}(D_{\mathrm{I}})}\,\|(\overline{u}\circ h)(\,\cdot\,,t_{\mathrm{I}};\theta_u) - b_{\mathrm{I}}(\,\cdot\,)\|^2_{\mathscr{L}_2(D_{\mathrm{I}})} \\ &+ \frac{1}{\mathrm{vol}(D_{\mathrm{L}})}\,\|(\overline{u}\circ h)_1(x_{\mathrm{L}},\,\cdot\,;\theta_u) - \bar{c}(\,\cdot\,;\theta_c)\|^2_{\mathscr{L}_2(D_{\mathrm{L}})} \\ &+ \frac{1}{\mathrm{vol}(D_{\mathrm{R}})}\,\|(\overline{u}\circ h)_2(x_{\mathrm{R}},\,\cdot\,;\theta_u) - b_{\mathrm{R}}(\,\cdot\,)\|^2_{\mathscr{L}_2(D_{\mathrm{R}})}\,. \end{aligned}$$

The approximation is obtained by a suitable quadrature rule, which we specify later. Compared to the definition of L_{DIF} in Chapter 4, the left boundary condition depends on the control $\bar{c}(t)$. We also construct a loss function for the objective function (5.1a)

$$L_{\text{OBJ}}(\theta_u, \theta_c) \approx \frac{1}{2} \left\| (\overline{u} \circ h)_1(x_{\text{R}}, \cdot\,; \theta_u) - u_{\text{OBJ}}(\cdot) \right\|^2_{\mathscr{L}_2(D_{\text{R}})} + \frac{\lambda}{2} \left\| \bar{c}(\cdot\,; \theta_c) - c_{\text{OBJ}} \right\|^2_{\mathscr{L}_2(D_{\text{L}})} .$$

In conclusion, we consider the training problem

$$\min_{\theta_u, \theta_c} \; L_{\text{DIF}} + \gamma L_{\text{OBJ}} \,. \tag{5.3}$$

The factor $\gamma > 0$ balances between the objective function, which we aim to minimize as much as possible, and the constraints, which should always be satisfied. Next, we want to assess the effectiveness of this approach.

5.2.1. Numerical tests

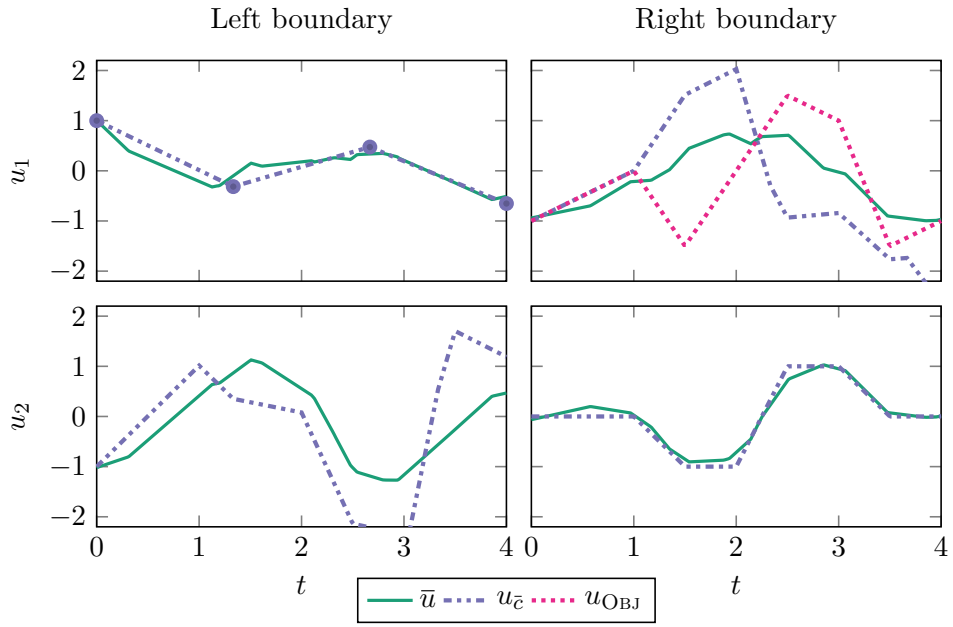
In this subsection, we test the direct approach with the linear optimization problem and a control with $d_c = 4$ control points. This problem was specifically chosen to test the direct approach in a setting where the exact control cannot be reconstructed. We want to evaluate, how well this approach satisfies the conditions while minimizing the objective. For this, we define the exact value of the objective function (5.1a) with respect to a control $c(t)$

$$\mathcal{J}(c) = \frac{1}{2} \left\| (u_c)_1(x_{\text{R}}, \cdot\,) - u_{\text{OBJ}}(\cdot) \right\|^2_{\mathscr{L}_2(D_{\text{R}})} + \frac{\lambda}{2} \left\| c(\cdot\,) - c_{\text{OBJ}}(\cdot) \right\|^2_{\mathscr{L}_2(D_{\text{L}})} .$$

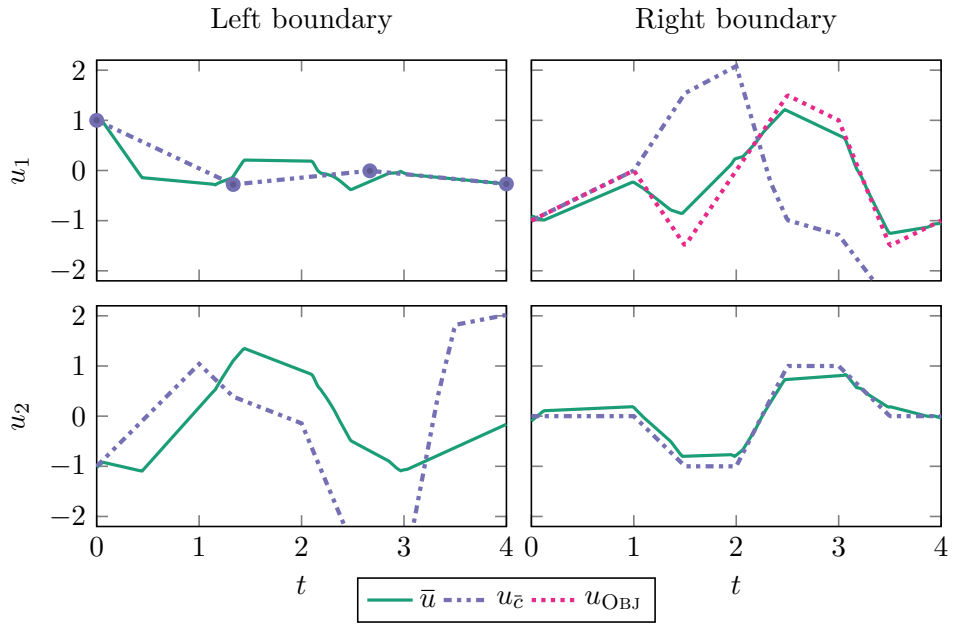
The test uses the hyperparameters that have resulted in the most accurate approximations in the previous chapter. These are listed in Table 4.6a. Then, we solve the training problem (5.3) with the Adam method for 150 000 iterations. We test the approach for $\gamma = 0.25$ and $\gamma = 1.5$. The results are shown in Figure 5.1.

We observe that the results depend strongly on γ: For $\gamma = 0.25$, the approximation $\overline{u}_1$ on the right boundary is a compromise between the exact solution of the simulation problem $u_{\bar{c}}$ and the target function u_{OBJ}. On the other hand, for $\gamma = 1.5$, the approximation $\overline{u}_1$ is very close to u_{OBJ} and far away from the exact solution. Remarkably, in both cases, the generalization error between the approximation and the exact solution is very large. This is also shown in the figures, where the approximation of $\overline{u}_2(x_{\text{L}}, \cdot\,)$ is very poor, since these values depend only on the balance law. There is also a significant difference between the loss value L_{OBJ} and the exact objective value $J(\bar{c})$. This shows that the direct training problem (5.3) does not adequately reflect the optimal control problem (5.1).

The obtained results can be explained by the loss function of the training problem (5.3). Here, both the constraints and objectives are treated equally, and the training process is unaware of the specific role of each loss function. Consequently, $\overline{u}_1(x_{\text{R}}, \cdot\,)$ is optimized such that the difference to u_{OBJ} is minimal. However, any changes in $\overline{u}_1(x_{\text{R}}, \cdot\,)$ should only be attributed to changes in the control $\bar{c}$. Furthermore, $\overline{u}_1(x_{\text{L}}, \cdot\,)$ should be optimized to be close to $\bar{c}$. But in the training problem (5.3), $\bar{c}$ is also optimized to be close to $\overline{u}_1(x_{\text{L}}, \cdot\,)$.



(a) Solution for $\gamma = 0.25$. The dots ● are the control points of the control $\bar{c}$. Furthermore, we have $\text{error}[u_1] = 6.43 \times 10^{-1}$, $\text{error}[u_2] = 7.23 \times 10^{-1}$, $\mathcal{J}(\bar{c}) = 1.13 \times 10^{0}$, $L_{\text{OBJ}} = 2.71 \times 10^{-1}$.



(b) Solution for $\gamma = 1.5$. The dots ● are the control points of the control $\bar{c}$. Furthermore, we have $\text{error}[u_1] = 8.91 \times 10^{-1}$, $\text{error}[u_2] = 1.03 \times 10^{0}$, $\mathcal{J}(\bar{c}) = 1.45 \times 10^{0}$, $L_{\text{OBJ}} = 3.14 \times 10^{-2}$.

Figure 5.1.: Solutions computed by the direct approach for the linear optimization problem, two values of γ and $d_c = 4$.

In conclusion, the direct training problem (5.3) fails to accurately reflect the interactions between the state variables $u(x,t)$ and the control $c(t)$ in the optimal control problem (5.1). As a consequence, both are optimized based on the wrong incentives, resulting in infeasible solutions. Adjusting γ can certainly help prioritize between objectives and constraints, but this does not solve the underlying issue. And it raises the question of how the choice of γ affects the result. Therefore, the direct approach is not suitable for the problems we consider.

In the next section, we construct an alternative approach based on the adjoint of the optimal control problem. We demonstrate that this method does not have the same limitations as the direct approach and show its effectiveness through numerical tests.

5.3. Indirect approach

In this section, we develop an indirect physics-informed approach, partially based on the work in [1], to solve the optimal control problem (5.1). In contrast to the direct approach, which directly minimizes the objective function (5.1a), the indirect approach solves optimality conditions of the optimal control problem. The optimality conditions describe a stationary point of the Lagrange function associated with the optimal control problem.

The optimality conditions include the so-called adjoint problem. The adjoint is a measurement tool that quantifies the deviation between the solution u and the target u_{OBJ} at the right boundary. This information is then transported to the left boundary, where the control can be updated accordingly. The adjoint is usually used to provide sensitivity information in an optimization procedure.

For the indirect approach, however, the introduction of the adjoint serves a different purpose and allows to effectively decouple the update rules for the parameters of the state vector θ_u and the control θ_c. We will see that, in contrast to the direct approach, this ensures that all constraints are satisfied and wrong incentives, such as the control adapting to the state or the state adapting to the objective function, are avoided. As a result, the indirect approach adequately reflects the optimal control problem, and does not require any balancing between the objective and the constraints.

The derivation of the adjoint-based optimality conditions is shown in Subsection 5.3.1. Then, in Subsection 5.3.2, we describe our physics-informed approach to solve the optimality conditions. In Subsection 5.3.3, we examine the approach with numerical tests.

5.3.1. Adjoint-based optimality conditions

In this subsection, we derive optimality conditions for the optimal control problem (5.1). We will see that the conditions consist of three separate components: The simulation problem, the adjoint problem, and a multiplier law. Each component will be used later to update the state, the adjoint, and the control separately.

For the following, we define the $\mathscr{L}_2$ scalar product by

$$\langle f, g \rangle_{\mathscr{L}_2(D)} = \int_D \big\langle f(x), g(x) \big\rangle_2 \,\mathrm{d}x$$

for functions $f, g \colon D \to \mathbb{R}^n$ and the euclidean scalar product $\langle \cdot\,, \cdot \rangle_2$.

We start by defining the Lagrange function

$$\begin{aligned}\mathcal{L}(u,c,\xi,\zeta,\nu,\mu) = \frac{1}{2}\left\|u_1(x_{\mathrm{R}},\,\cdot\,) - u_{\mathrm{OBJ}}(\,\cdot\,)\right\|^2_{\mathcal{L}_2(D_{\mathrm{R}})} + \frac{\lambda}{2}\left\|c - c_{\mathrm{OBJ}}\right\|^2_{\mathcal{L}_2(D_{\mathrm{L}})} \\ + \left\langle \partial_t u + \partial_x(F\circ u) - g\circ u, \xi\right\rangle_{\mathcal{L}_2(D_{\mathrm{EQ}})} \qquad (5.4) \\ + \left\langle u(\,\cdot\,,t_{\mathrm{I}}) - b_{\mathrm{I}}, \zeta\right\rangle_{\mathcal{L}_2(D_{\mathrm{I}})} \qquad (5.5) \\ + \left\langle u_1(x_{\mathrm{L}},\,\cdot\,) - c, \mu\right\rangle_{\mathcal{L}_2(D_{\mathrm{L}})} \qquad (5.6) \\ + \left\langle u_2(x_{\mathrm{R}},\,\cdot\,) - b_{\mathrm{R}}, \nu\right\rangle_{\mathcal{L}_2(D_{\mathrm{R}})} \qquad (5.7)\end{aligned}$$

for the Lagrange multiplier $\xi\colon D_{\mathrm{EQ}} \to \mathbb{R}$, $\zeta\colon D_{\mathrm{I}} \to \mathbb{R}$, $\mu\colon D_{\mathrm{L}} \to \mathbb{R}$ and $\nu\colon D_{\mathrm{R}} \to \mathbb{R}$. Consequently, we consider the minimization problem

$$\min_{u,c,\xi,\zeta,\nu,\mu} \mathcal{L}(u,c,\xi,\zeta,\nu,\mu)\,. \qquad (5.8)$$

Formal Lagrange calculus. In the following, we derive optimality conditions for the minimization problem (5.8). These conditions will describe a stationary point of the Lagrange function $\mathcal{L}$. The performed procedure is based on the formal Lagrange calculus described in [59, p. 67]. That is, we perform the calculations syntactically without providing evidence that they are semantically correct.

Especially, we will not provide function spaces for the solution, adjoint, and control. We just assume that the calculations can be performed and that all functions are at least square integrable. Note that specifying function spaces is inherently difficult for nonlinear transport equations, since discontinuities can occur in the underlying simulation problem.

Fréchet derivative. To formulate a stationary point of $\mathcal{L}$, we need to consider derivatives of functions with respect to function parameters. Here, the Fréchet derivative provides us with a generalization of the total derivative to normed function spaces [59, p. 46]. We denote the Fréchet derivative of a function $f(x)$ by f' or, in the case that f depends on multiple parameters, by f_x.

Let f and g be Fréchet differentiable, then the following statements hold for the Fréchet derivative [59, p. 47]:

(A) The Fréchet derivative is linear: $(\alpha f + \beta g)'h = \alpha f'h + \beta g'h$.

(B) For every bounded linear operator $f(x) = Ax$ we have $f'(x)h = Ah$.

(C) The chain rule holds
$$(f\circ g)'h = f'\big(g(x)\big)g'(x)h\,.$$

Stationary point. The Lagrange function $\mathcal{L}$ has a stationary point if the Fréchet derivative of $\mathcal{L}$ with respect to each parameter is zero. Hence, we seek for u, c, ξ, ζ, μ, and ν such that

$$\mathcal{L}_u = 0\,,\quad \mathcal{L}_c = 0\,,\quad \mathcal{L}_\xi = 0\,,\quad \mathcal{L}_\zeta = 0\,,\quad \mathcal{L}_\nu = 0\,,\quad \text{and}\quad \mathcal{L}_\mu = 0\,. \qquad (5.9)$$

In the following, each derivative in (5.9) is derived.

Simulation problem ($\mathcal{L}_\xi = 0$, $\mathcal{L}_\zeta = 0$, $\mathcal{L}_\nu = 0$, $\mathcal{L}_\mu = 0$). We start with the derivatives of $\mathcal{L}$ with respect to the Lagrange multipliers ξ, ζ, ν, and μ. We define $k_1(x) = \langle \chi, x \rangle_{\mathscr{L}_2}$ for an arbitrary χ. The function $k_1(x)$ is linear in x, and thus by (B) we have $k_1'(x)h = \langle \chi, h \rangle_{\mathscr{L}_2}$. In the following, we will refer to k_1 several times, but we will not specify χ each time.

With the derivative of k_1 we can directly conclude

$$\begin{aligned} \mathcal{L}_\xi h &= \big\langle \partial_t u + \partial_x(F \circ u) - g, h \big\rangle_{\mathscr{L}_2(D_{\text{EQ}})}, & \mathcal{L}_\zeta h &= \big\langle u(\,\cdot\,, t_{\text{I}}) - b_{\text{I}}, h \big\rangle_{\mathscr{L}_2(D_{\text{I}})}, \\ \mathcal{L}_\nu h &= \big\langle u_1(x_{\text{L}},\,\cdot\,) - c, h \big\rangle_{\mathscr{L}_2(D_{\text{L}})}, & \mathcal{L}_\mu h &= \big\langle u_2(x_{\text{R}},\,\cdot\,) - b_{\text{R}}, h \big\rangle_{\mathscr{L}_2(D_{\text{R}})}. \end{aligned} \tag{5.10}$$

To satisfy the conditions for a stationary point (5.9), the integrals $\mathcal{L}_\xi h$, $\mathcal{L}_\zeta h$, $\mathcal{L}_\nu h$ and $\mathcal{L}_\mu$ need to vanish for any h. To achieve this, we require the stronger condition that each integrand is pointwise zero. This is equivalent to the requirement that u satisfies the simulation problem (5.1b) – (5.1e).

Adjoint problem ($\mathcal{L}_u = 0$). Next, we derive the derivative $\mathcal{L}_u$. To describe the first term of $\mathcal{L}$, let $k_2(x) = \|x\|^2_{\mathscr{L}_2}$. Then, as described in [59, p. 45], we have $k_2'(x)h = 2\langle x, h \rangle_{\mathscr{L}_2}$. Further let $k_3(u) = u_1(x_{\text{R}},\,\cdot\,) - u_{\text{OBJ}}(\,\cdot\,)$. Here, by (B) we have $k_3'(u)h = h_1(x_{\text{R}},\,\cdot\,)$. Therefore, we obtain

$$(k_2 \circ k_3)(u) = \|u_1(x_{\text{R}},\,\cdot\,) - u_{\text{OBJ}}(\,\cdot\,)\|^2_{\mathscr{L}_2(D_{\text{R}})}\,.$$

By using the chain rule (C), we conclude

$$(k_2 \circ k_3)'(u)h = 2\big\langle u_1(x_{\text{R}},\,\cdot\,) - u_{\text{OBJ}}(\,\cdot\,), h_1(x_{\text{R}},\,\cdot\,)\big\rangle^2_{\mathscr{L}_2(D_{\text{R}})}\,.$$

Auxiliary calculation: Integration by parts for inner products. Before we continue with the derivative $\mathcal{L}_u$, we need the following auxiliary statement: For functions $f(t), g(t)\colon [a,b] \to \mathbb{R}^n$ with $a < b$ we have

$$\int_a^b \langle \partial_t f(t), g(t) \rangle_2 \,\mathrm{d}t = \langle f(b), g(b) \rangle_2 - \langle f(a), b(a) \rangle_2 - \int_a^b \langle f(t), \partial_t g(t) \rangle_2 \,\mathrm{d}t\,. \tag{5.11}$$

This follows by considering $e(t) = \langle f(t), g(t) \rangle_2$ with $\partial_t e(t) = \langle \partial_t f(t), g(t) \rangle_2 + \langle f(t), \partial_t g(t) \rangle_2$. Then, by the fundamental theorem of calculus we obtain

$$\int_a^b \langle \partial_t f(t), g(t) \rangle_2 + \langle f(t), \partial_t g(t) \rangle_2 \,\mathrm{d}t = \int_a^b \partial_t e(t) \,\mathrm{d}t = \langle f(b), g(b) \rangle_2 - \langle f(a), b(a) \rangle_2\,.$$

Rearranging now yields (5.11).

Adjoint problem continued. As the next intermediate step, we build the derivative of (5.4) with respect to u. Let $k_4(u) = \partial_t u$. Since k_4 is linear in u and by (B) we have $k_4'(u)h = \partial_t h$. Therefore, we can conclude

$$(k_1 \circ k_4)(u) = \big\langle \partial_t u, \xi \big\rangle_{\mathscr{L}_2(D_{\text{EQ}})} \qquad \text{and} \qquad (k_1 \circ k_4)'(u)h = \big\langle \partial_t h, \xi \big\rangle_{\mathscr{L}_2(D_{\text{EQ}})}.$$

Integration by parts now yields

$$\begin{aligned}\big\langle \partial_t h, \xi \big\rangle_{\mathscr{L}_2(D_{\mathrm{EQ}})} &= \int_{x_{\mathrm{R}}}^{x_{\mathrm{L}}} \int_{t_{\mathrm{I}}}^{t_{\mathrm{E}}} \big\langle \partial_t h, \xi \big\rangle_2 \,\mathrm{d}t \,\mathrm{d}x \\ &= \int_{x_{\mathrm{R}}}^{x_{\mathrm{L}}} \big\langle h(x,t_{\mathrm{E}}), \xi(x,t_{\mathrm{E}}) \big\rangle_2 \,\mathrm{d}x - \int_{x_{\mathrm{R}}}^{x_{\mathrm{L}}} \big\langle h(x,t_{\mathrm{I}}), \xi(x,t_{\mathrm{I}}) \big\rangle_2 \,\mathrm{d}x \\ &\quad - \int_{x_{\mathrm{R}}}^{x_{\mathrm{L}}} \int_{t_{\mathrm{I}}}^{t_{\mathrm{E}}} \big\langle h, \partial_t \xi \big\rangle_2 \,\mathrm{d}t \,\mathrm{d}x \\ &= \big\langle h(\,\cdot\,,t_{\mathrm{E}}), \xi(\,\cdot\,,t_{\mathrm{E}}) \big\rangle_{\mathscr{L}_2(D_{\mathrm{I}})} - \big\langle h(\,\cdot\,,t_{\mathrm{I}}), \xi(\,\cdot\,,t_{\mathrm{I}}) \big\rangle_{\mathscr{L}_2(D_{\mathrm{I}})} - \big\langle h, \partial_t \xi \big\rangle_{\mathscr{L}_2(D_{\mathrm{EQ}})} .\end{aligned}$$

Now let $k_5(u) = \partial_x u$ and similarly we obtain $k_5(u)'h = \partial_x h$. By the chain rule, we have $(k_5 \circ F)'(u)h = \partial_x(F'(u)h)$. Hence, we conclude

$$(k_1 \circ k_5 \circ F)(u) = \big\langle \partial_x F(u), \xi \big\rangle_{\mathscr{L}_2(D_{\mathrm{EQ}})} \quad \text{and} \quad (k_1 \circ k_5 \circ F)'(u) = \big\langle \partial_x (F'(u)h), \xi \big\rangle_{\mathscr{L}_2(D_{\mathrm{EQ}})} .$$

Integration by parts now yields

$$\begin{aligned}&\big\langle \partial_x (F'(u)h), \xi \big\rangle_{\mathscr{L}_2(D_{\mathrm{EQ}})} \\ &\quad = \int_{t_{\mathrm{I}}}^{t_{\mathrm{E}}} \int_{x_{\mathrm{R}}}^{x_{\mathrm{L}}} \big\langle \partial_x (F'(u)h), \xi \big\rangle_2 \,\mathrm{d}x \,\mathrm{d}t \\ &\quad = \int_{t_{\mathrm{I}}}^{t_{\mathrm{E}}} \big\langle (F'(u)h)(x_{\mathrm{R}},t), \xi(x_{\mathrm{R}},t) \big\rangle_2 \,\mathrm{d}t - \int_{t_{\mathrm{I}}}^{t_{\mathrm{E}}} \big\langle (F'(u)h)(x_{\mathrm{L}},t), \xi(x_{\mathrm{L}},t) \big\rangle_2 \,\mathrm{d}t \\ &\qquad - \int_{t_{\mathrm{I}}}^{t_{\mathrm{E}}} \int_{x_{\mathrm{R}}}^{x_{\mathrm{L}}} \big\langle F'(u)h, \partial_x \xi \big\rangle_2 \,\mathrm{d}x \,\mathrm{d}t \\ &\quad = \big\langle (F'(u)h)(x_{\mathrm{R}},\,\cdot\,), \xi(x_{\mathrm{R}},\,\cdot\,) \big\rangle_{\mathscr{L}_2(D_{\mathrm{I}})} - \big\langle (F'(u)h)(x_{\mathrm{L}},\,\cdot\,), \xi(x_{\mathrm{L}},\,\cdot\,) \big\rangle_{\mathscr{L}_2(D_{\mathrm{I}})} \\ &\qquad - \big\langle F'(u)h, \partial_x \xi \big\rangle_{\mathscr{L}_2(D_{\mathrm{EQ}})} .\end{aligned}$$

To conclude the derivative of (5.4) with respect to u, we observe

$$(k_1 \circ g)(u) = \big\langle g, \xi \big\rangle_{\mathscr{L}_2(D_{\mathrm{EQ}})} \quad \text{and thus} \quad (k_1 \circ g)'(u)h = \big\langle g'(u)h, \xi \big\rangle_{\mathscr{L}_2(D_{\mathrm{EQ}})} .$$

The remaining derivatives for $\mathcal{L}_u$ are the derivatives of (5.5), (5.6) and (5.7) with respect to u. Here, by performing similar calculations as before, we obtain

$$\langle h(\,\cdot\,,t_{\mathrm{I}}), \zeta \rangle_{\mathscr{L}_2(D_{\mathrm{I}})} , \quad \langle h_1(x_{\mathrm{L}},\,\cdot\,), \mu \rangle_{\mathscr{L}_2(D_{\mathrm{L}})} , \quad \text{respectively} \quad \langle h_2(x_{\mathrm{R}},\,\cdot\,), \nu \rangle_{\mathscr{L}_2(D_{\mathrm{R}})} .$$

Finally, we combine all derivatives and receive

$$\begin{aligned}\mathcal{L}_u h = &\big\langle u_1(x_{\mathrm{R}},\,\cdot\,) - u_{\mathrm{OBJ}}(\,\cdot\,), h_1(x_{\mathrm{R}},\,\cdot\,) \big\rangle^2_{\mathscr{L}_2(D_{\mathrm{R}})} \\ &+ \big\langle h(\,\cdot\,,t_{\mathrm{E}}), \xi(\,\cdot\,,t_{\mathrm{E}}) \big\rangle_{\mathscr{L}_2(D_{\mathrm{I}})} - \big\langle h(\,\cdot\,,t_{\mathrm{I}}), \xi(\,\cdot\,,t_{\mathrm{I}}) \big\rangle_{\mathscr{L}_2(D_{\mathrm{I}})} - \big\langle h, \partial_t \xi \big\rangle_{\mathscr{L}_2(D_{\mathrm{EQ}})} \\ &+ \big\langle (F'(u)h)(x_{\mathrm{R}},\,\cdot\,), \xi(x_{\mathrm{R}},\,\cdot\,) \big\rangle_{\mathscr{L}_2(D_{\mathrm{R}})} - \big\langle (F'(u)h)(x_{\mathrm{L}},\,\cdot\,), \xi(x_{\mathrm{L}},\,\cdot\,) \big\rangle_{\mathscr{L}_2(D_{\mathrm{L}})} \\ &- \big\langle F'(u)h, \partial_x \xi \big\rangle_{\mathscr{L}_2(D_{\mathrm{EQ}})} \\ &+ \big\langle g'(u)h, \xi \big\rangle_{\mathscr{L}_2(D_{\mathrm{EQ}})} \\ &+ \big\langle h(\,\cdot\,,t_{\mathrm{I}}), \zeta \big\rangle_{\mathscr{L}_2(D_{\mathrm{I}})} + \big\langle h_1(x_{\mathrm{L}},\,\cdot\,), \mu \big\rangle_{\mathscr{L}_2(D_{\mathrm{L}})} + \big\langle h_2(x_{\mathrm{R}},\,\cdot\,), \nu \big\rangle_{\mathscr{L}_2(D_{\mathrm{R}})} .\end{aligned} \tag{5.12}$$

The derivatives $g'(u)$ and $F'(u)$ can be represented by their Jacobians $J_g(u)$ and $J_F(u)$, respectively. Additionally, we have

$$J_F(u)^\top = \begin{pmatrix} (\partial_{u_1} F)^\top \\ (\partial_{u_2} F)^\top \end{pmatrix} .$$

With these identities, we rearrange the terms in (5.12) and obtain

$$\begin{aligned}
\mathcal{L}_u h = &\left\langle -\partial_t \xi - J_F(u)^\top \partial_x \xi + J_g(u)^\top \xi, h \right\rangle_{\mathscr{L}_2(D_{\text{EQ}})} \\
&+ \left\langle \zeta - \xi(\,\cdot\,, t_{\text{I}}), h(\,\cdot\,, t_{\text{I}}) \right\rangle_{\mathscr{L}_2(D_{\text{I}})} \\
&+ \left\langle \xi(\,\cdot\,, t_{\text{E}}), h(\,\cdot\,, t_{\text{E}}) \right\rangle_{\mathscr{L}_2(D_{\text{I}})} \\
&+ \left\langle \mu - \Big(\partial_{u_1} F\big(u(x_{\text{L}}, \,\cdot\,)\big)\Big)^\top \xi(x_{\text{L}}, \,\cdot\,), h_1(x_{\text{L}}, \,\cdot\,) \right\rangle_{\mathscr{L}_2(D_{\text{L}})} \\
&+ \left\langle -\Big(\partial_{u_2} F\big(u(x_{\text{L}}, \,\cdot\,)\big)\Big)^\top \xi(x_{\text{L}}, \,\cdot\,), h_2(x_{\text{L}}, \,\cdot\,) \right\rangle_{\mathscr{L}_2(D_{\text{L}})} \\
&+ \left\langle u_1(x_{\text{R}}, \,\cdot\,) - u_{\text{OBJ}}(\,\cdot\,) + \Big(\partial_{u_1} F\big(u(x_{\text{R}}, \,\cdot\,)\big)\Big)^\top \xi(x_{\text{R}}, \,\cdot\,), h_1(x_{\text{R}}, \,\cdot\,) \right\rangle_{\mathscr{L}_2(D_{\text{R}})} \\
&+ \left\langle \nu + \Big(\partial_{u_2} F\big(u(x_{\text{R}}, \,\cdot\,)\big)\Big)^\top \xi(x_{\text{R}}, \,\cdot\,), h_2(x_{\text{R}}, \,\cdot\,) \right\rangle_{\mathscr{L}_2(D_{\text{R}})} .
\end{aligned}$$

Again, we extract pointwise conditions, such that each integrand vanishes. Therefore, we conclude that ξ should fulfill

$$\partial_t \xi + J_F(u)^\top \partial_x \xi = J_g(u)^\top \xi \qquad \text{in } D_{\text{EQ}} , \tag{5.13a}$$

$$\xi(\,\cdot\,, t_{\text{E}}) = 0 \qquad \text{on } D_{\text{I}} , \tag{5.13b}$$

$$\Big(\partial_{u_2} F\big(u(x_{\text{L}}, \,\cdot\,)\big)\Big)^\top \xi(x_{\text{L}}, \,\cdot\,) = 0 \qquad \text{on } D_{\text{L}} , \tag{5.13c}$$

$$\Big(\partial_{u_1} F\big(u(x_{\text{R}}, \,\cdot\,)\big)\Big)^\top \xi(x_{\text{R}}, \,\cdot\,) = u_{\text{OBJ}}(\,\cdot\,) - u_1(x_{\text{R}}, \,\cdot\,) \qquad \text{on } D_{\text{R}} . \tag{5.13d}$$

We denote the equations (5.13) as adjoint equations and ξ as adjoint variables. The remaining equations vanish, if we choose suitable ζ, μ, and ν.

The adjoint is zero if the target function $u_{\text{OBJ}}(\,\cdot\,)$ is equal to $u_1(x_{\text{R}}, \,\cdot\,)$, since then the end condition (5.13b), the left boundary (5.13c), and the right boundary (5.13d) are zero. Otherwise, information about the deviation of the target is introduced through the right boundary condition (5.13d). This information is then transported through the domain by (5.13a). However, with the current equations, this information is not used and no update of the control can be performed. To achieve this, we will next derive the multiplier law, the missing piece of the puzzle.

Multiplier law ($\mathcal{L}_c = 0$). There are only two terms in $\mathcal{L}$ that depend on the control c. The derivatives can be derived using similar arguments as before. Consequently, we have

$$\mathcal{L}_c h = \left\langle \lambda(c - c_{\text{OBJ}}) + \Big(\partial_{u_1} F\big(u(x_{\text{L}}, \,\cdot\,)\big)\Big)^\top \xi(x_{\text{L}}, \,\cdot\,), h \right\rangle_{\mathscr{L}_2(D_{\text{L}})} .$$

Again, we want each integrand to vanish pointwise. Therefore, we require

$$\lambda(c - c_{\text{OBJ}}) + \Big(\partial_{u_1} F\big(u(x_{\text{L}}, \cdot)\big)\Big)^{\top} \xi(x_{\text{L}}, \cdot) = 0 \qquad \text{on } D_{\text{L}}\,. \tag{5.14}$$

We refer to equation (5.14) as the multiplier law. The multiplier law combines the state and the adjoint at the left boundary with the control into one equation. If this equation holds with respect to a state u and an adjoint ξ, then $\mathcal{L}_c = 0$ and thus the control cannot be improved anymore.

Stationary point revisited. In the previous paragraphs, we derived optimality conditions that imply a stationary point (5.9) of the Lagrange function $\mathcal{L}$. Specifically, if the state variables u, the adjoint variable ξ, and the control c satisfy the simulation problem (5.1b) – (5.1e), the adjoint equations (5.13), and the multiplier law (5.14), then the conditions for a stationary point (5.9) are fulfilled. Moreover, the Lagrange multipliers ζ, μ, and ν can be eliminated from the system. In the following subsection, we describe how these equations are solved using a physics-informed approach.

5.3.2. Physics-informed approach

In this subsection, we describe a physics-informed approach to solve the adjoint-based optimality conditions which we have derived in the previous subsection. This approach is inspired by the work in [1], but differs in important aspects which we will highlight.

Approximation of u, ξ, c. The state vector, the adjoint variables, and the control serve different purposes in the optimality conditions. To avoid any unwanted connections between them and to allow them to focus on different aspects during the training process, we intentionally use three different parameterized functions to approximate them. This is in contrast to the method proposed in [1], where one tailored neural network predicts the state vector, the adjoint variables, and the control.

Our approach builds on the definitions in Section 5.2. The approximation of the state vector $u(x,t)$ is defined by $\overline{u}(h(x,t;\theta_u))$ for parameters θ_u and a neural network h. The parameterized control $\bar{c}(t)$ is defined by (5.2).

Additionally, we have an approximation of the adjoint variables ξ, which we denote by $\bar{\xi}(x,t) = h(x,t;\theta_\xi)$ for the parameters θ_ξ. Note that the adjoint equations (5.13) and the multiplier law (5.14) can be evaluated for any ξ.

New loss functions. As a next step, we translate the adjoint equations (5.13) into a physics-informed loss function. Here, we have

$$\begin{aligned} L_{\text{ADJ}}(\theta_u, \theta_\xi) \approx \frac{1}{\operatorname{vol}(D_{\text{EQ}})} \Big\| & \partial_t \bar{\xi}(\,\cdot\,,\,\cdot\,;\theta_\xi) \\ & + J_F\big((\overline{u} \circ h)(\,\cdot\,,\,\cdot\,;\theta_u)\big)^{\top} \partial_x \bar{\xi}(\,\cdot\,,\,\cdot\,;\theta_\xi) \\ & - J_g\big((\overline{u} \circ h)(\,\cdot\,,\,\cdot\,;\theta_u)\big)^{\top} \bar{\xi}(\,\cdot\,,\,\cdot\,;\theta_\xi) \Big\|_{\mathscr{L}_2(D_{\text{EQ}})} \\ + \frac{1}{\operatorname{vol}(D_{\text{I}})} & \big\| \bar{\xi}(\,\cdot\,, t_{\text{E}}; \theta_\xi) \big\|_{\mathscr{L}_2(D_{\text{I}})} \end{aligned}$$

$$+ \frac{1}{\operatorname{vol}(D_{\mathrm{L}})} \left\| \Big(\partial_{u_2} F\big((\overline{u} \circ h)(x_{\mathrm{L}}, \cdot\,; \theta_u)\big) \Big)^{\top} \bar{\xi}(x_{\mathrm{L}}, \cdot\,; \theta_\xi) \right\|_{\mathscr{L}_2(D_{\mathrm{L}})}$$
$$+ \frac{1}{\operatorname{vol}(D_{\mathrm{R}})} \left\| \Big(\partial_{u_1} F\big((\overline{u} \circ h)(x_{\mathrm{R}}, \cdot\,; \theta_u)\big) \Big)^{\top} \bar{\xi}(x_{\mathrm{R}}, \cdot\,; \theta_\xi) \right.$$
$$\left. - u_{\mathrm{OBJ}}(\,\cdot\,) + (\overline{u} \circ h)_1(x_{\mathrm{R}}, \cdot\,) \right\|_{\mathscr{L}_2(D_{\mathrm{R}})} .$$

A suitable quadrature rule, which we will specify later, is used to obtain the approximation. The derivatives of ξ in L_{ADJ} are computed with forward mode automatic differentiation. However, the derivatives of F and g are implemented explicitly. This is another difference from the approach in [1], where every derivative is obtained by automatic differentiation.

The physics-informed loss function for the multiplier law (5.14) is given by

$$L_{\mathrm{MULT}}(\theta_u, \theta_\xi, \theta_c) \approx \frac{1}{\operatorname{vol}(D_{\mathrm{L}})} \left\| \lambda\big(\bar{c}(\,\cdot\,; \theta_c) - c_{\mathrm{OBJ}}\big) \right.$$
$$\left. + \Big(\partial_{u_1} F\big((\overline{u} \circ h)(x_{\mathrm{L}}, \cdot\,; \theta_u)\big) \Big)^{\top} \bar{\xi}(x_{\mathrm{L}}, \cdot\,; \theta_\xi) \right\|_{\mathscr{L}_2(D_{\mathrm{L}})} .$$

Training process. The approach in [1] considers the training problem

$$\min_{\theta_u, \theta_\xi, \theta_c} \; L_{\mathrm{DIF}}(\theta_u, \theta_c) + L_{\mathrm{ADJ}}(\theta_u, \theta_\xi) + L_{\mathrm{MULT}}(\theta_u, \theta_\xi, \theta_c) \,. \tag{5.15}$$

However, this training problem has the same underlying issue as the direct approach. The parameters θ_u, θ_ξ, and θ_c are still optimized based on incorrect incentives. Specifically, the state vector $\overline{u}$ still adjusts according to the condition $u_{\mathrm{OBJ}}(\,\cdot\,) + u_1(x_{\mathrm{R}}, \cdot\,)$ in L_{ADJ}. Furthermore, the control adapts to $u_1(x_{\mathrm{L}}, \cdot\,) - c(\,\cdot\,)$ in L_{DIF}. This issue also shows up in a practical implementation of (5.15). As a result, the training problem (5.15) is unable to distinguish between constraints and objective. This has also been noted and addressed by [1], who suggests to incorporate weights into the training problem.

But we will pursue a different strategy, which is inspired by the classical strategies described in [11, p. 63] and [29, p. 125]. There, the optimal control problem is decoupled and solved in a loop of the following steps. First, the simulation problem is solved with respect to a fixed control, resulting in a new state vector. Second, the adjoint equations are solved with respect to a fixed state vector, resulting in new adjoint variables. Third, the state vector and the adjoint variables are combined to form a so-called reduced gradient, which is used to update the control. Crucially, the state vector, the adjoint variables, and the control are each updated with their own update rule.

We transfer this strategy to obtain the following training procedure. The parameters of the state vector θ_u are trained on the simulation problem, encoded by L_{DIF}, the parameters of the adjoint variables θ_ξ are trained on the adjoint equations, encoded by L_{ADJ}, and the parameters of the control θ_c are trained on the multiplier law, encoded by L_{MULT}. Specifically, a training step works as follows. First, we calculate the gradients $\nabla_{\theta_u} L_{\mathrm{DIF}}(\theta_u, \theta_c)$, $\nabla_{\theta_\xi} L_{\mathrm{ADJ}}(\theta_u, \theta_\xi)$, and $\nabla_{\theta_c} L_{\mathrm{MULT}}(\theta_u, \theta_\xi, \theta_c)$. Second, we update the parameters according to

$$\theta_u \leftarrow \theta_u - \nabla_{\theta_u} L_{\mathrm{DIF}} \,, \qquad \theta_\xi \leftarrow \theta_\xi - \nabla_{\theta_\xi} L_{\mathrm{ADJ}} \,, \qquad \theta_c \leftarrow \theta_c - \nabla_{\theta_c} L_{\mathrm{MULT}} \,.$$

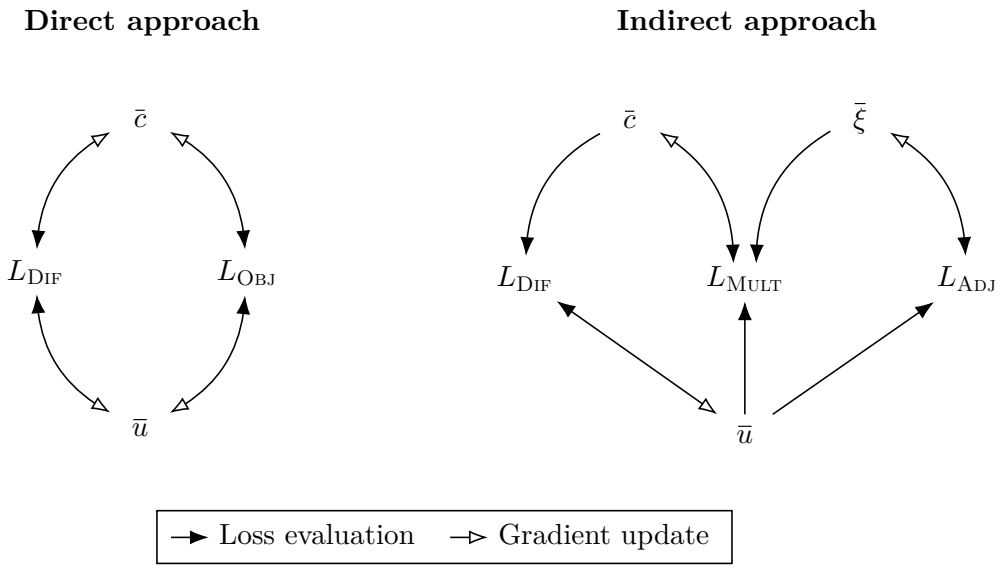


Figure 5.2.: Training process of the direct and indirect approach.

In summary, we have three different loss functions, one for each of the different parameters. These loss functions encode only the information relevant to the parameters. Thus, we have removed the wrong optimization incentives from the training problem. Importantly, this differs from the other approaches only when the target cannot be met exactly. See Figure 5.2, which compares the training process of the indirect approach from this section and the direct approach from Section 5.2. In the next subsection, we test the indirect approach.

5.3.3. Numerical results

Setup. The tests in this subsection are performed with the hyperparameters that led to either the most accurate or the most efficient results in the previous chapter, listed in Table 4.6 for the linear problem and the isentropic problem. In particular, the neural network architectures are used to approximate the state vector u and the adjoint variables ξ. The sampling strategies are used to approximate the norms in the loss functions L_{DIF}, L_{ADJ} and L_{MULT}. For the linear problem, we consider $d_c \in \{4, 6, 8, 9\}$, and for the isentropic problem, we consider $d_c \in \{4, 8, 12\}$. We also test two scenarios: one with constant nodes during the training process, and another where the nodes are trained.

Importantly, the loss function for θ_ξ depends on the parameters θ_u and the loss function for θ_c depends on the parameters θ_u, θ_ξ. To improve the learning process, we want to give θ_u an advantage over θ_ξ and θ_c, and also θ_ξ an advantage over θ_c. To accomplish this, we extend the exponential decaying learning rate (3.7) with a new variable that delays the learning rate for η_{DELAY} steps. Consequently, we consider the learning rate

$$\eta_k = \begin{cases} 0 & \text{if } k \leq \eta_{\text{DELAY}}, \\ \eta_{\text{INIT}} \cdot \eta_{\text{RATE}}^{\frac{k-\eta_{\text{DELAY}}}{\eta_{\text{STEPS}}}} & \text{otherwise,} \end{cases}$$

where the remaining learning rate variables are as before.

		Accurate Hyperparameter				Efficient Hyperparameter			
d_c	nodes	L_{OBJ}	$\mathcal{J}(\bar{c})$	error$[u_1]$	error$[u_2]$	L_{OBJ}	$\mathcal{J}(\bar{c})$	error$[u_1]$	error$[u_2]$
4	constant	9.43e−1	9.43e−1	2.90e−3	3.50e−3	9.41e−1	9.44e−1	8.33e−3	1.11e−2
4	trained	2.25e−1	2.26e−1	**1.02e−3**	**1.22e−3**	2.24e−1	2.26e−1	**6.73e−3**	**7.92e−3**
6	constant	9.51e−2	9.88e−2	1.50e−2	1.85e−2	9.28e−2	9.96e−2	4.10e−2	5.09e−2
6	trained	4.10e−2	4.35e−2	2.64e−2	3.16e−2	3.93e−2	4.64e−2	4.57e−2	5.54e−2
8	constant	7.30e−2	7.99e−2	3.48e−2	3.68e−2	6.07e−2	9.03e−2	9.63e−2	1.06e−1
8	trained	**3.47e−3**	**5.33e−3**	2.61e−2	3.25e−2	**3.40e−3**	**8.19e−3**	5.33e−2	6.65e−2
9	constant	**1.04e−3**	**1.04e−3**	**7.82e−4**	**9.79e−4**	**1.04e−3**	**1.04e−3**	**1.03e−3**	**1.19e−3**
9	trained	**1.04e−3**	**1.05e−3**	**2.47e−3**	**3.11e−3**	**1.06e−3**	**1.08e−3**	**5.17e−3**	**6.51e−3**

(a) Results for the linear optimal control problem.

		Accurate Hyperparameter				Efficient Hyperparameter			
d_c	nodes	L_{OBJ}	$\mathcal{J}(\bar{c})$	error$[\rho]$	error$[\rho v]$	L_{OBJ}	$\mathcal{J}(\bar{c})$	error$[\rho v]$	error$[\rho v]$
4	const	3.76e−3	3.86e−3	**1.40e−3**	6.00e−1	3.80e−3	3.84e−3	**1.49e−3**	6.00e−1
4	train	2.37e−3	2.32e−3	**9.71e−4**	5.98e−1	2.37e−3	2.32e−3	**1.50e−3**	5.98e−1
8	const	2.04e−3	2.01e−3	1.54e−3	**5.98e−1**	2.03e−3	2.02e−3	2.11e−3	**5.97e−1**
8	train	**1.84e−3**	**1.78e−3**	1.52e−3	**5.97e−1**	**1.77e−3**	**1.86e−3**	3.89e−3	**5.97e−1**
12	const	**1.78e−3**	**1.82e−3**	**1.47e−3**	**5.97e−1**	**1.82e−3**	**1.81e−3**	**1.45e−3**	**5.97e−1**
12	train	**1.42e−3**	**1.79e−3**	6.47e−3	5.98e−1	**1.61e−3**	**1.69e−3**	3.53e−3	5.97e−1

(b) Results for the isentropic optimal control problem.

Table 5.1.: Results of the indirect approach. The three lowest values per column are highlighted.

We use three learning rates, one for the parameters θ_u, one for θ_ξ, and one for θ_c. To limit the number of learning rates considered, we make the following assumptions. We set $\eta_{\text{RATE}} = 0.9$. The parameters θ_u and θ_ξ use the same values for η_{STEPS} and η_{INIT}, which we denote by $\eta_{\text{STEPS}-u\xi}$ and $\eta_{\text{INIT}-u\xi}$, respectively. Additionally, we have $\eta_{\text{STEPS}-c}$ and $\eta_{\text{INIT}-c}$. Then, we consider the following combinations

$$\eta_{\text{STEPS}-u\xi}, \eta_{\text{STEPS}-c} \in \{1000, 5000, 10000, 15\,000\}\,, \quad \eta_{\text{INIT}-u\xi}, \eta_{\text{INIT}-c} \in \{0.1, 0.01, 0.001\}\,.$$

Furthermore, we denote the delay steps for θ_ξ and θ_c by $\eta_{\text{DELAY}-\xi}$ and $\eta_{\text{DELAY}-c}$, respectively. Here, we consider the following choices

$$(\eta_{\text{DELAY}-\xi}, \eta_{\text{DELAY}-c}) \in \{(500, 1000), (500, 2000), (1000, 2000), (2000, 4000)\}\,.$$

We solve the training problem with the Adam method for 150 000 iterations. From all obtained results, we select the ten results with the lowest L_{DIF} values, and from these, we select the one with the lowest L_{OBJ} value. This ensures that the simulation problem is adequately solved, while also taking into account the results with a low objective value. Finally, we run each test three times and report the average results.

Quantitative evaluation. We start by quantitatively evaluating the results by analyzing the obtained objective values and generalization errors for different numbers of control points d_c and whether the nodes are constant or trained. The generalization error measures the relative $\mathscr{L}_2$-error between $\overline{u}$ and $u_{\bar{c}}$.

The results for the linear and the isentropic control problem are listed in Table 5.1. Most importantly, the generalization errors are much better than those of the direct approach. But the generalization errors are still higher than in the previous chapter, also and the error of $\overline{\rho v}$ is quite high.

Furthermore, the objective values decrease with a higher number of control points d_c and also when the nodes are trained. Crucially, L_{OBJ} and $\mathcal{J}(\bar{c})$ are very close. In conclusion, the results show that the indirect approach closely reflects the optimal control problem (5.1). There is only a small difference between the results obtained with the most accurate and the most efficient hyperparameters, and thus compute time can be saved by using the latter.

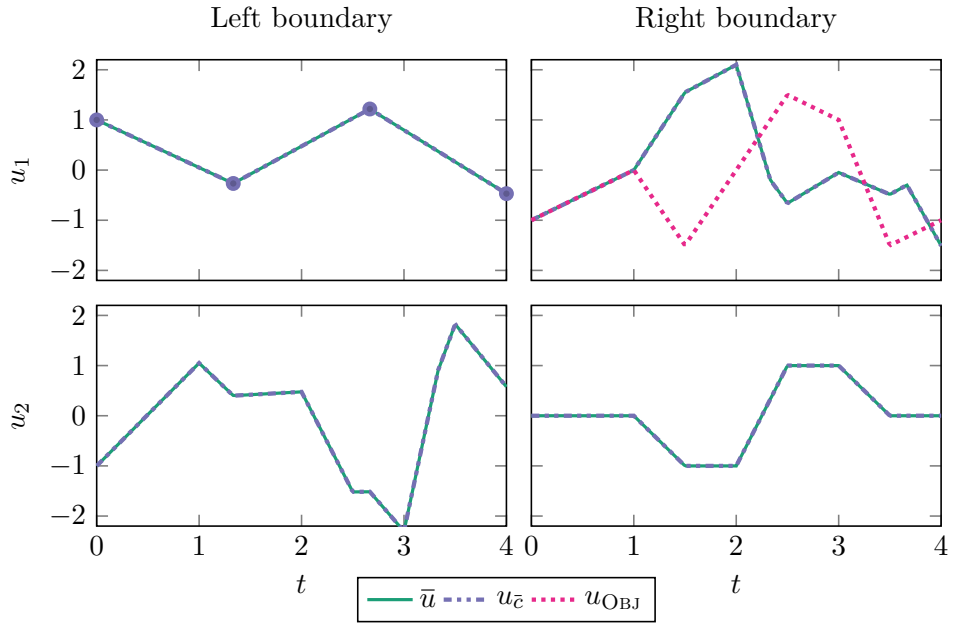
Qualitative evaluation. The second step of the test evaluation is the visual analysis of the obtained solutions. We visualize the solution with the lowest L_{OBJ} value, selected from the top ten solutions with the lowest L_{DIF} values among all solutions trained with different learning rates and random initializations. We begin by analyzing the test considered in Section 5.2. There, we considered the linear optimization problem with $d_c = 4$ and the nodes remained constant.

See Figure 5.3a for the results obtained by the indirect approach. Compared to the result of the direct approach, in Figure 5.1, the generalization errors are significantly reduced, and the achieved objective value $\mathcal{J}(\bar{c})$ is lower. This shows that the indirect approach does not have the same problems as the direct approach. See Figure 5.3b for the same test, but here the nodes are also trained. Importantly, training the nodes improves the control obtained, which is quantified by a lower objective value $\mathcal{J}(\bar{c})$ and qualified in the figure, since the target is better reached.

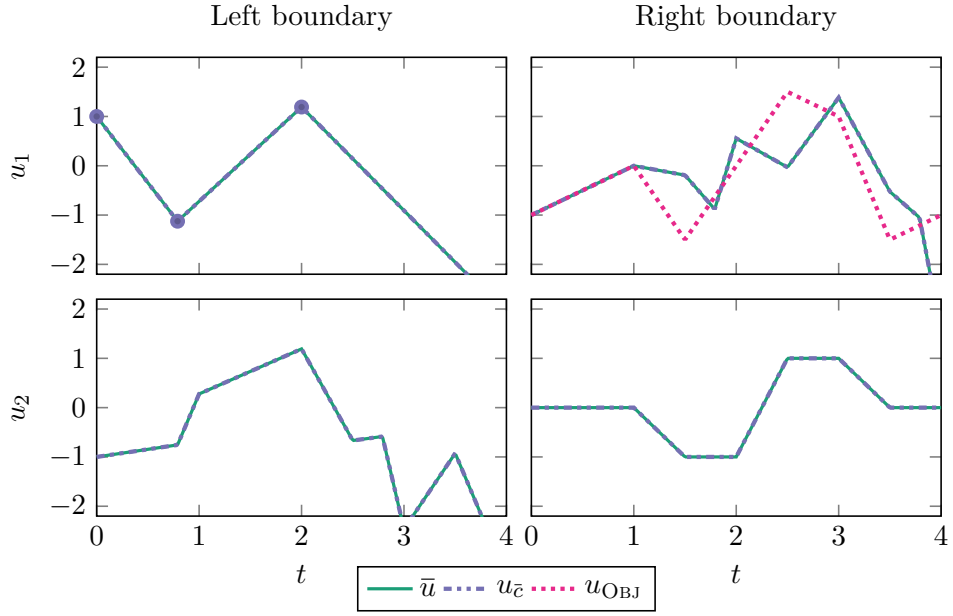
The remaining solutions for the linear problem are in Figure 5.4. It is evident that as the number of control points increases, the target at the right boundary is better fulfilled. In particular, with $d_c = 9$ control points, the exact boundary data from the linear simulation problem is reconstructed. This shows that the indirect approach successfully finds the optimal solution.

The solutions for the isentropic problem are shown in Figure 5.5. Crucially, the solutions maintain a constant density and improve with more control points. However, the solution with $d_c = 12$ control points, where the nodes are also trained, shows some deficiencies.

It is important to emphasize that the initial density drop cannot be addressed by any control due to the time it takes for the control to affect the right boundary. This demonstrates that the indirect approach respects the balance law and does not optimize the state to conform to the target function.



(a) Solution with constant nodes. The dots ● are the control points of the control $\bar{c}$. Furthermore, we have error$[u_1] = 6.99 \times 10^{-4}$, error$[u_2] = 1.01 \times 10^{-3}$, $\mathcal{J}(\bar{c}) = 9.43 \times 10^{-1}$, $L_{\text{OBJ}} = 9.43 \times 10^{-1}$.



(b) Solution with trained nodes. The dots ● are the control points of the control $\bar{c}$. Furthermore, we have error$[u_1] = 4.58 \times 10^{-5}$, error$[u_2] = 5.38 \times 10^{-5}$, $\mathcal{J}(\bar{c}) = 2.26 \times 10^{-1}$, $L_{\text{OBJ}} = 2.26 \times 10^{-1}$.

Figure 5.3.: Solutions computed by the indirect approach for the linear optimization problem with constant and trained nodes, and $d_c = 4$.

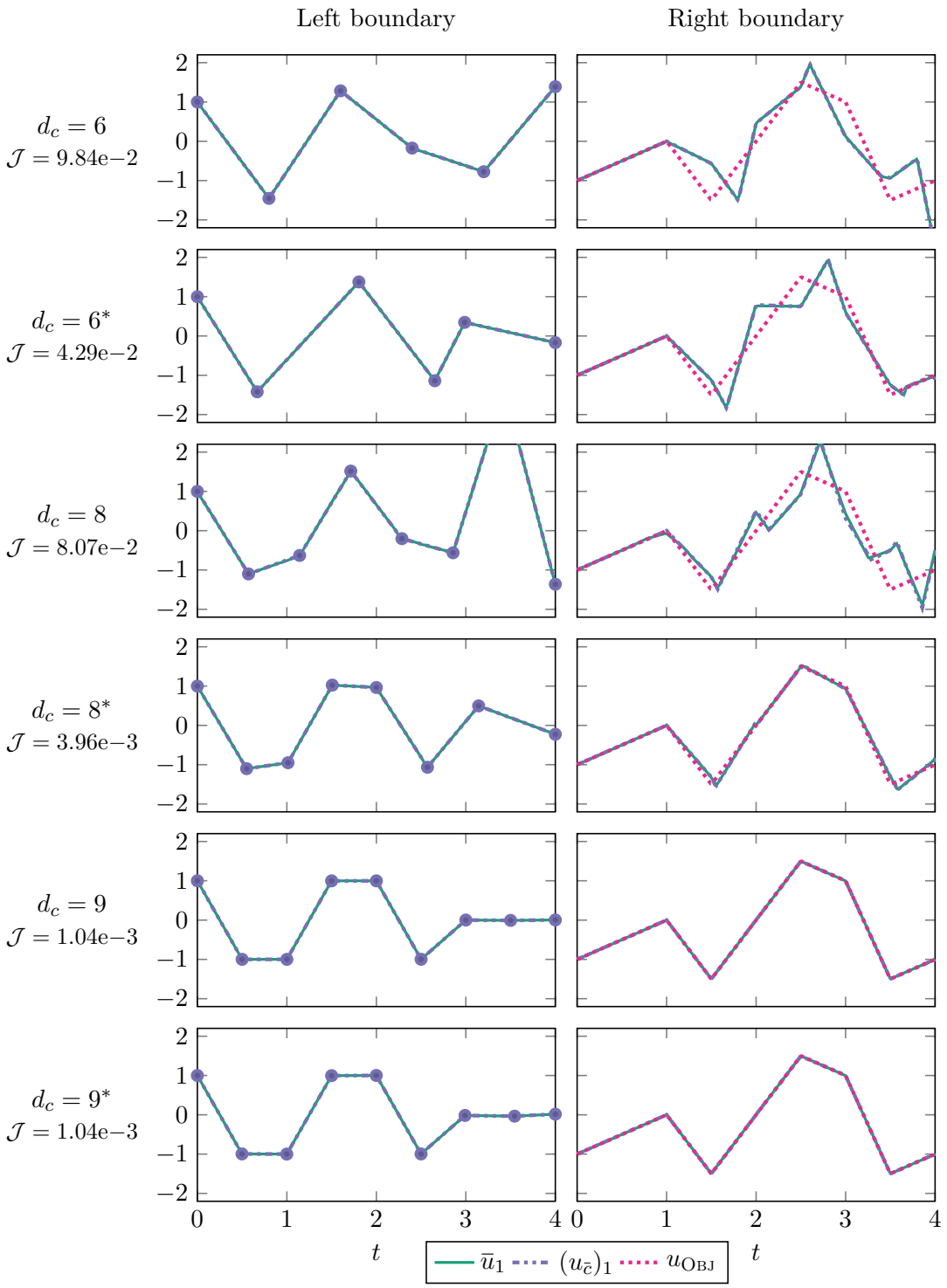


Figure 5.4.: Solution of the indirect approach for the linear problem with different values for d_c. An asterisk indicates the results where the nodes are also trained.

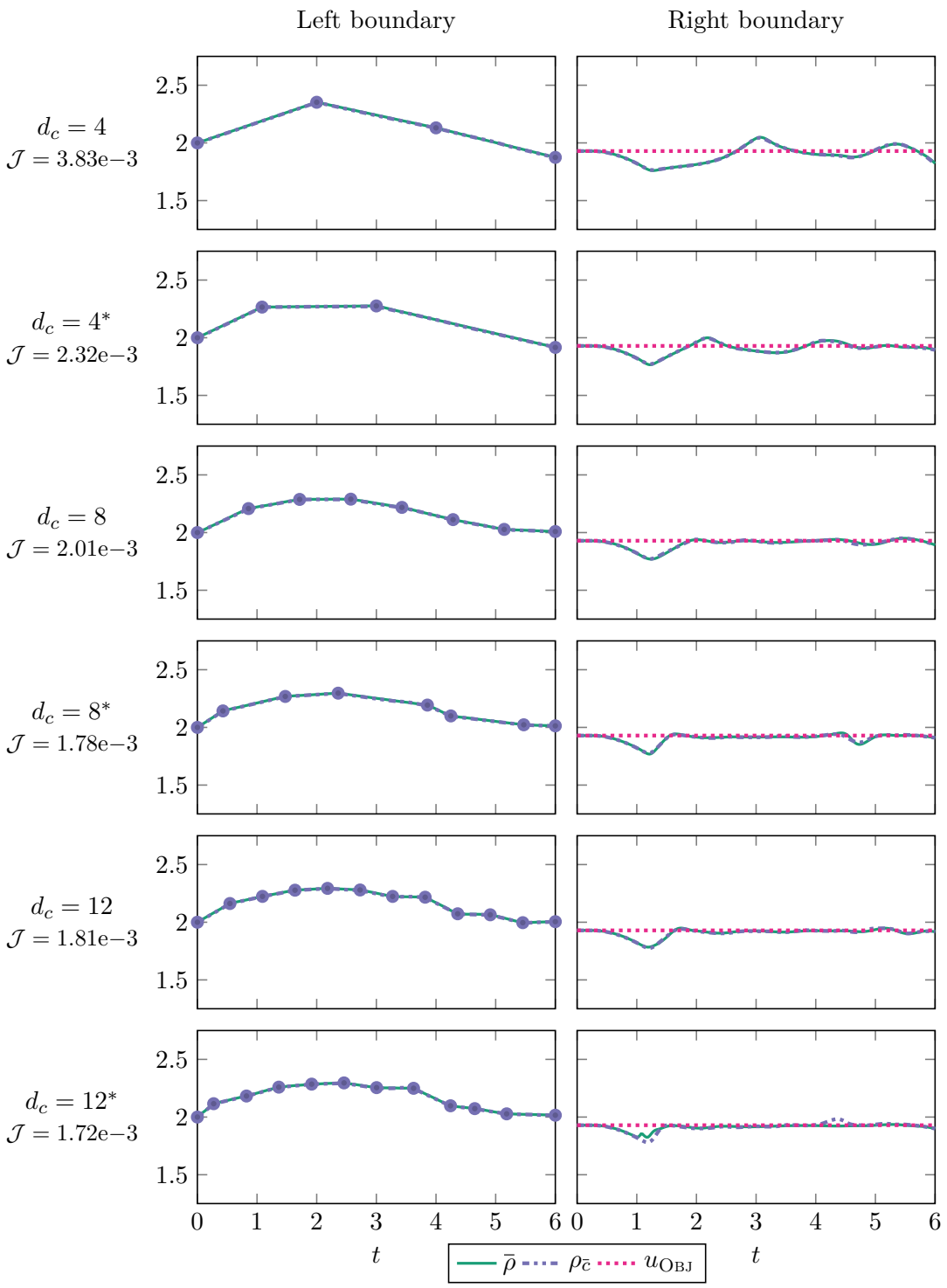


Figure 5.5.: Solution of the indirect approach for the isentropic problem with different values for d_c. An asterisk indicates the results where the nodes are also trained.

5.4. Conclusion

In this chapter, we have presented a direct and an indirect approach to solve optimal boundary control problems with physics-informed neural networks. The optimal boundary control problem includes a balance law constraint. We have tested both approaches with two test problems: one based on a linear balance law and one based on the isentropic Euler equations. These tests used the most accurate and efficient hyperparameters identified in the previous chapter.

Importantly, the results show that the direct approach does not distinguish between the constraints and the objective. This has been demonstrated practically by solutions which do not satisfy the balance law. Therefore, this approach is not suitable to solve the problems considered.

To circumvent this issue, we have developed an indirect approach. The indirect approach solves optimality conditions and introduces the adjoint variables, the adjoint equations, and the multiplier law of the optimal control problem. This allows us to formulate separate loss functions for the state, adjoint, and control parameters.

We tested the indirect approach with two test problems and a different number of control points. The results show that the method was very effective in solving the optimal control problems. The computed solutions satisfy the balance law and minimize the objective value. This shows that the indirect approach correctly reflects the dynamics of the original optimal control problem.

To obtain the results, we have tested many different learning rates to find the best solutions. There is certainly room for improvement to reduce the number of combinations tested. The computed solutions did not solve the simulation problem with the same level of accuracy as in the previous chapter. This implies that the simulation problem alone must be solved with decent accuracy to make this approach viable. The accuracy of the approach could be improved with a different update rule, where only the state, the adjoint, or the control is updated for a few iterations at a time.

The proposed indirect approach has a very general formulation, making it applicable to other optimal control problems. While our control was a continuous piecewise linear function, alternative parameterized functions can also be applied.

In addition to the approaches discussed in this chapter, physics-informed neural networks offer another direction to solve optimal control problems. Here, a reduced model is trained that predicts the state vector based on the coordinates and, crucially, a control. This reduced model can now be used in another optimization process to quickly predict a state with respect to the control and to obtain sensitivity information by automatic differentiation of the reduced model. Significant research efforts have been invested in this topic, including the development of deep operator networks [35] and their specialized adaptations for transport problems [33].

6. Conclusion

Physics-informed neural networks are a new numerical method to solve problems involving differential equations. Compared to classical numerical methods, a lot of knowledge needs to be developed. This includes the types of problems that can be solved, the advantages, the limitations, and the most effective strategies.

In this thesis, we developed knowledge on each of these aspects when physics-informed neural networks are applied to gas transport problems. The contributions can be grouped into three areas, which we outline below.

Advancements in fundamental knowledge. PINNs are a very flexible method that can be easily adapted to a wide variety of differential equations. However, the underlying dynamics of the differential equations are different, so specific knowledge of the method for gas transport problems is required.

Specifically, we derived error estimates for a system of two linear transport equations that shares important properties with the nonlinear Euler equations. Importantly, the estimates bound the generalization error by the loss function values, thus validating the physics-informed approach. However, the estimates also show that physics-informed neural networks have problems simulating over large time scales and with high characteristic speeds. *As a result, simulations of full-scale gas pipelines are not possible with this approach and require further research.* We also performed a practical convergence analysis, but the results could not be fully explained by the error estimates. This suggests that decomposing the loss value into a training error and a quadrature error is not sufficient.

In addition, we developed and compared different implementations of physics-informed neural networks. Our comparison shows that the commonly used implementation based on reverse mode automatic differentiation is inefficient. An implementation based on forward mode automatic differentiation, which also groups similar operations together, requires about half the computational cost of the reverse mode implementation.

Finding the most effective simulation strategy. Physics-informed neural networks have been extended in many directions. In this thesis, we studied many variants and performed intensive numerical tests to find the most effective training strategy. For the numerical tests, we considered a linear transport problem and a problem using the nonlinear isentropic Euler equations. Together, these problems provided a broad picture of the capabilities of physics-informed neural networks.

Our tests included different neural network architectures and activation functions. Here, fully connected deep neural networks provided the best results. For each test problem, the best approximations were obtained with a different activation function, number of layers, and number of neurons. Hence, the most effective neural network depends on the specific problem and its solution.

Furthermore, we tested different optimization strategies to solve the training problem. Here, the Adam method with a suitable learning rate outperformed the L-BFGS method and a hybrid of the two. We also considered different sampling strategies for the training procedure. The original formulation uses Latin hypercube sampling, but there are other strategies with more promising properties. However, the theoretical advantages do not lead to better approximations, and the overall difference between the strategies is rather small.

The training problem of physics-informed neural networks consists of multiple loss functions that need to be minimized simultaneously. Here, variants have been proposed that can compensate for imbalances in the loss function. In our tests, a random search is able to improve the accuracy. However, this method is computationally expensive and cheaper methods performed worse. By deriving a loss function using the integral form of the balance law, we only need to minimize one function. The original formulation of this approach did not achieve the same level of accuracy as PINNs. We showed that a modification of this loss function not only matched the accuracy of PINNs in one test case, but also required a smaller computational budget.

Our numerical tests demonstrated that physics-informed neural networks are not a highly accurate numerical method. Even with intensive hyperparameter tuning, the generalization errors cannot be reduced arbitrarily.

Solving optimal boundary control problems. PINNs can be easily extended to use cases beyond simulation problems. In this thesis, we followed this path and investigated physics-informed approaches to solve optimal boundary control problems.

Here, we considered an existing direct approach and our tests showed that the direct approach is not able to compute feasible solutions. Therefore, we developed an indirect adjoint-based approach that uses different loss functions for the different parameters in the training process. We performed extensive numerical tests using this approach and two test problems. In all cases, the solutions were feasible, and also the objective was minimized. *Hence, the indirect approach solves optimal control problems very effectively.*

Outlook. Physics-informed neural networks excel when applied to problems that involve differential equations, but have additional constraints that make it difficult to apply classical numerical methods. They are a complementary numerical method and can be easily adapted to many problems. Improving physics-informed neural networks not only benefits simulation problems, but many other problems as well, and thus can have a very broad impact. This illustrates the value of further research in physics-informed neural networks.

In this thesis, we identified several research opportunities, including specifically tailored training methods and neural network architectures. This research can be continued by developing a physics-informed approach to simulate and optimize real-scale gas pipeline systems with pipes and algebraic elements such as compressors and valves.

Bibliography

[1] J. Barry-Straume, A. Sarshar, A. A. Popov, and A. Sandu. Physics-informed neural networks for PDE-constrained optimization and control, 2022. arXiv: 2205.03377. Preprint.

[2] A. Bihlo and R. O. Popovych. Physics-informed neural networks for the shallow-water equations on the sphere. *Journal of Computational Physics*, 456:111024, 2022. DOI: 10.1016/j.jcp.2022.111024.

[3] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, version 0.4.15, 2018. URL: http://github.com/google/jax.

[4] I. Bremmer. How the World Must Respond to the AI Revolution. May 2023. URL: https://time.com/6283716/world-must-respond-to-the-ai-revolution/. Accessed on August 17, 2023.

[5] C. Himpe, S. Grundel, and P. Benner. Model order reduction for gas and energy networks. *Journal of Mathematics in Industry*, 11:13, 2021.

[6] E. Celledoni, M. J. Ehrhardt, C. Etmann, R. I. Mclachlan, B. Owren, C.-B. Schonlieb, and F. Sherry. Structure-preserving deep learning. *European Journal of Applied Mathematics*, 32(5):888–936, 2021. DOI: 10.1017/S0956792521000139.

[7] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18, New Orleans, Louisiana, USA. AAAI Press, 2018. DOI: 10.5555/3504035.3504378.

[8] A. Chaumet and J. Giesselmann. Efficient wPINN-Approximations to Entropy Solutions of Hyperbolic Conservation Laws, 2022. arXiv: 2211.12393. Preprint.

[9] A. Daw, J. Bu, S. Wang, P. Perdikaris, and A. Karpatne. Mitigating propagation failures in physics-informed neural networks using retain-resample-release (R3) sampling. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 7264–7302. PMLR, July 2023.

[10] T. De Ryck, S. Lanthaler, and S. Mishra. On the approximation of functions by tanh neural networks. *Neural Networks*, 143:732–750, 2021. DOI: 10.1016/j.neunet.2021.08.015.

[11] P. Domschke. *Adjoint-Based Control of Model and Discretization Errors for Gas Transport in Networked Pipelines*. Dr. Hut, München, 2011.

[12] P. Domschke, B. Hiller, J. Lang, V. Mehrmann, R. Morandin, and C. Tischendorf. Gas Network Modeling: An Overview, 2021. Preprint.

[13] L. C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2nd edition, 2010. DOI: https://doi.org/10.1090/gsm/019.

[14] C. J. García-Cervera, M. Kessler, and F. Periago. Control of Partial Differential Equations via Physics-Informed Neural Networks. *Journal of Optimization Theory and Applications*, 196(2):391–414, 2023. DOI: 10.1007/s10957-022-02100-4.

[15] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR, May 2010.

[16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[17] A. Griewank and A. Walther. *Evaluating Derivatives*. Society for Industrial and Applied Mathematics, second edition, 2008. DOI: 10.1137/1.9780898717761.

[18] M. D. Gunzburger and P. B. Bochev. *Least-Squares Finite Element Methods*. Applied Mathematical Sciences. Springer New York, NY, 1st edition, 2009. DOI: 10.1007/b13382.

[19] E. Haber and L. Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, Dec. 2017. DOI: 10.1088/1361-6420/aa9a90.

[20] S. Haber. A Modified Monte-Carlo Quadrature. *Mathematics of Computation*, 21(99):388–397, 1967. DOI: 10.1090/S0025-5718-1967-0234606-9.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. DOI: 10.1109/CVPR.2016.90.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, Los Alamitos, CA, USA. IEEE Computer Society, Dec. 2015. DOI: 10.1109/ICCV.2015.123.

[23] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus), 2016. arXiv: 1606.08415. Preprint.

[24] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure

prediction with alphafold. *Nature*, 596(7873):583–589, 2021. DOI: 10.1038/s41586-021-03819-2.

[25] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, June 2021. DOI: 10.1038/s42254-021-00314-5.

[26] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. DOI: 10.48550/arXiv.1412.6980.

[27] G. Kissas, Y. Yang, E. Hwuang, W. R. Witschey, J. A. Detre, and P. Perdikaris. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 358:112623, 2020. DOI: 10.1016/j.cma.2019.112623.

[28] D. K. Klein, M. Fernández, R. J. Martin, P. Neff, and O. Weeger. Polyconvex anisotropic hyperelasticity with neural networks. *Journal of the Mechanics and Physics of Solids*, 159:104703, 2022. DOI: doi.org/10.1016/j.jmps.2021.104703.

[29] O. Kolb. *Simulation and Optimization of Gas and Water Supply Networks*. Dr. Hut, München, 2011.

[30] A. S. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[31] L. McClenny and U. Braga-Neto. Self-adaptive physics-informed neural networks using a soft attention mechanism. In J. Lee, E. Darve, P. Kitanidis, M.W. Mahoney, A. Karpatne, M.W. Farthing, and T. Hesser, editors, *Proceedings of the AAAI 2021 Spring Symposium on Combining Artificial Intelligence and Machine Learning with Physical Sciences* (Stanford, CA, USA), volume 2964 of *CEUR Workshop Proceedings*, Aachen. CEUR-WS, 2021.

[32] P. L'Ecuyer. Randomized Quasi-Monte Carlo: An Introduction for Practitioners. In A. B. Owen and P. W. Glynn, editors, *Monte Carlo and Quasi-Monte Carlo Methods*, pages 29–52, Cham. Springer International Publishing, 2018.

[33] S. Lanthaler, R. Molinaro, P. Hadorn, and S. Mishra. Nonlinear Reconstruction for Operator Learning of PDEs with Discontinuities. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

[34] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511791253.

[35] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, Mar. 2021. DOI: 10.1038/s42256-021-00302-5.

[36] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. DOI: 10.1016/j.jcp.2018.10.045.

[37] R. Mojgani, M. Balajewicz, and P. Hassanzadeh. Kolmogorov n–width and Lagrangian physics-informed neural networks: A causality-conforming manifold for convection-dominated PDEs. *Computer Methods in Applied Mechanics and Engineering*, 404:115810, 2023. DOI: 10.1016/j.cma.2022.115810.

[38] S. Mowlavi and S. Nabi. Optimal control of PDEs using physics-informed neural networks. *Journal of Computational Physics*, 473:111731, 2023. DOI: 10.1016/j.jcp.2022.111731.

[39] O. Kolb, J. Lang, and P. Bales. An implicit box scheme for subsonic compressible flow with dissipative source term. *Numerical Algorithms*, 53:293–307, 2010. DOI: 10.1007/s11075-009-9287-y.

[40] J. A. A. Opschoor, P. C. Petersen, and C. Schwab. Deep ReLU networks and high-order finite element methods. *Analysis and Applications*, 18(05):715–770, 2020. DOI: 10.1142/S0219530519410136.

[41] A. B. Owen. Latin Supercube Sampling for Very High-Dimensional Simulations. *ACM Trans. Model. Comput. Simul.*, 8(1):71–102, Jan. 1998. DOI: 10.1145/272991.273010.

[42] A. B. Owen. *Monte Carlo theory, methods and examples.* https://artowen.su.domains/mc/, 2013.

[43] A. B. Owen. On dropping the first sobol' point, 2021. arXiv: 2008.08051. Preprint.

[44] P. Domschke, A. Dua, J.J. Stolwijk, J. Lang, and V. Mehrmann. Adaptive refinement strategies for the simulation of gas flow in networks using a model hierarchy. *Electronic Transactions on Numerical Analysis*, 48:97–113, 2018. DOI: 10.1553/etna_vol48s97.

[45] A. Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, 1999. DOI: 10.1017/S0962492900002919.

[46] A. F. Queiruga, N. B. Erichson, D. Taylor, and M. W. Mahoney. Continuous-in-depth neural networks, 2020. arXiv: 2008.02389. Preprint.

[47] R.G. Patel, I. Manickam, N.A. Trask, M.A. Wood, M. Lee, I. Tomas, and E.C. Cyr. Thermodynamically consistent physics-informed neural networks for hyperbolic systems. *Journal of Computational Physics*, 449:110754, 2021. DOI: 10.1016/j.jcp.2021.110754.

[48] F. M. Rohrhofer, S. Posch, C. Gößnitzer, and B. C. Geiger. Data vs. Physics: The Apparent Pareto Front of Physics-Informed Neural Networks. *IEEE Access*, 11:86252–86261, 2023. DOI: 10.1109/ACCESS.2023.3302892.

[49] S. Mishra and R. Molinaro. Estimates on the generalization error of physics-informed neural networks for approximating PDEs. *IMA Journal of Numerical Analysis*, 2022. DOI: 10.1093/imanum/drab093.

[50] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43:A3055–A3081, 2021. DOI: 10.1137/20M1318043.

[51] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. DOI: 10.1038/nature16961.

[52] M. Stein. Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics*, 29(2):143–151, 1987. DOI: 10.1080/00401706.1987.10488205.

[53] D. Steinkraus, I. Buck, and P. Simard. Using GPUs for machine learning algorithms. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, volume 2, pages 1115–1120, 2005. DOI: 10.1109/ICDAR.2005.251.

[54] E. L. Strelow, A. Gerisch, J. Lang, and M. E. Pfetsch. Physics informed neural networks: a case study for gas transport problems. *Journal of Computational Physics*, 481:112041, 2023. DOI: 10.1016/j.jcp.2023.112041.

[55] The Future of Hydrogen. Technical report, International Energy Agency, 2019. URL: https://www.iea.org/reports/the-future-of-hydrogen.

[56] The Role of Gas in Today's Energy Transitions. Technical report, International Energy Agency, 2019. URL: https://www.iea.org/reports/the-role-of-gas-in-todays-energy-transitions.

[57] A. Tooze. Welcome to the world of the polycrisis. Oct. 2022. URL: https://www.ft.com/content/498398e7-11b1-494b-9cd3-6d669dc3de33. Accessed: August 16, 2023.

[58] L. N. Trefethen. Computing numerically with functions instead of numbers. *Commun. ACM*, 58(10):91–97, 2015. DOI: 10.1145/2814847.

[59] F. Tröltzsch. *Optimale Steuerung partieller Differentialgleichungen: Theorie, Verfahren und Anwendungen*. Vieweg+Teubner Verlag Wiesbaden, 2nd edition, Oct. 2009, page 311. DOI: 10.1007/978-3-8348-9357-4.

[60] M. Ulbrich and S. Ulbrich. *Nichtlineare Optimierung*. Mathematik Kompakt. Birkhäuser Basel, 1st edition, 2012. DOI: 10.1007/978-3-0346-0654-7.

[61] R. van der Meer, C. Oosterlee, and A. Borovykh. Optimally weighted loss functions for solving PDEs with Neural Networks. *Journal of Computational and Applied Mathematics*, 405, 2022. DOI: 10.1016/j.cam.2021.113887.

[62] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. DOI: 10.5555/3295222.3295349.

[63] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[64] X. Jin, S. Cai, H. Li, and G.E. Karniadakis. NSFnets (Navier-Stokes flow nets): physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 426:109951, 2021. DOI: 10.1016/j.jcp.2020.109951.

[65] Z. Mao, A.D. Jagtap, and G.E. Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020. DOI: 10.1016/j.cma.2019.112789.

[66] H. Zheng, Z. Yang, W. Liu, J. Liang, and Y. Li. Improving deep neural networks using softplus units. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–4, 2015. DOI: 10.1109/IJCNN.2015.7280459.

Wissenschaftlicher Werdegang

Erik Laurin Strelow

geboren am 18.2.1995 in Flörsheim am Main, Deutschland

2023	**Promotion in Mathematik (Dr. rer. nat.)** Technische Universität Darmstadt
2020 – 2023	**Mitglied im Sonderforschungsbereich Transregio 154** *Mathematische Modellierung, Simulation und Optimierung am Beispiel von Gasnetzwerken*
2020 – 2023	**(Hilfs-)Wissenschaftlicher Mitarbeiter in der AG *Numerik und Wissenschaftliches Rechnen*** Fachbereich Mathematik, Technische Universität Darmstadt
2019	**Master of Science Mathematik** Technische Universität Darmstadt
2017	**Bachelor of Science Mathematik** Technische Universität Darmstadt
2013	**Abitur** Schillerschule, Frankfurt am Main

Stipendien

2021 – 2022	**Graduiertenschule Computational Engineering** Technische Universität Darmstadt
2020 – 2021	**Sonderforschungsbereich Transregio 154**