



Fachbereich VI Informatik und Medien

**Masterarbeit**

von

Radmir Gesler

Zur Erlangung  
des akademischen Grades  
Master of Engineering (M.Eng.)

Im Studiengang  
Technische Informatik

Thema:

Physics Informed Neural Networks (PINN) zur Lösung eines inversen Problems der  
Wärmeleitungsgleichung

Betreuer: Prof. Dr. Frank Haußer

Gutachterin: Prof. Dr. Yin Amy Siu

Eingereicht am: 07. Juli 2025

## **Eidesstattliche Erklärung**

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbstständig verfasst und nur unter Zuhilfenahme der angegebenen Quellen und Hilfsmitteln angefertigt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach anderen gedruckten oder im Internet verfügbaren Werken entnommen sind, habe ich durch genaue Quellenangaben kenntlich gemacht.

Berlin, 6. Juli 2025

---

Radmir Gelser

## Abstract

This thesis investigates the inverse heat conduction problem for determining the spatially dependent thermal conductivity. The theoretical foundation is provided by the partial differential equation of heat conduction. The aim is to reconstruct the spatially varying thermal conductivity from temperature measurements induced by a heat source. Since the determination of thermal conductivity using classical numerical methods is highly challenging, a modern approach from machine learning is adopted: Physics-Informed Neural Networks (PINNs). By incorporating physical laws into the training of the PINNs, it is demonstrated that this approach yields acceptable results. The overall outcome of this work is a software framework that can be easily applied to four variants of the problem. Both stationary and non-stationary inverse heat conduction problems up to the second spatial dimension are investigated.

## Zusammenfassung

Diese Arbeit untersucht das inverse Problem der Wärmeleitung zur Bestimmung der ortsabhängigen Wärmeleitfähigkeit. Den theoretischen Ausgangspunkt bildet die partielle Differentialgleichung der Wärmeleitung. Ziel ist es, aus Temperaturmessungen, die durch eine Wärmequelle induziert werden, die ortsabhängige Wärmeleitfähigkeit zu rekonstruieren. Da die Bestimmung der Wärmeleitfähigkeit auf klassischem Wege mit Methoden der numerischen Mathematik sehr anspruchsvoll ist, wird ein moderner Ansatz aus dem maschinellen Lernen verfolgt: Physics-Informed Neural Networks (PINNs). Durch die Einbeziehung physikalischer Gesetzmäßigkeiten in das Training der PINNs wird gezeigt, dass dieser Ansatz akzeptable Ergebnisse liefert. Das Gesamtergebnis der Arbeit ist ein Software-Framework, das unkompliziert auf vier Varianten des Problems angewendet werden kann. Untersucht werden stationäre und instationäre inverse Wärmeleitungsprobleme bis zur zweiten Raumdimension.

# Inhaltsverzeichnis

<b>1. Einführung</b>	<b>1</b>
1.1. Was ist Wärmeübertragung . . . . .	1
1.2. Wärmeleitung: Direktes und inverses Problem . . . . .	2
1.3. Ziel: PINNs zur Lösung des inversen Problems . . . . .	3
<b>2. Inverses Problem der Wärmeleitung</b>	<b>6</b>
2.1. Herleitung der Wärmeleitungsgleichung . . . . .	6
2.2. Vorwärtsproblem der Wärmeleitungsgleichung (WLG) . . . . .	11
2.2.1. Schwache Formulierung . . . . .	11
2.2.2. Beispiel einer Vorwärtssimulation mit FEniCS . . . . .	12
2.3. Inverses Problem der Wärmeleitung . . . . .	13
<b>3. Physics Informed Neural Networks (PINNs)</b>	<b>18</b>
3.1. Statistisches Lernen . . . . .	18
3.1.1. Nichtlineare Modelle und neuronale Netze . . . . .	20
3.1.2. Beispiel: PINNs für direkte Lösung der Wärmeleitungsgleichung . . . . .	24
3.2. PINNs für das inverse Wärmeleitungsproblem . . . . .	27
3.2.1. Einfluss zusätzlicher Terme in der Verlustfunktion auf den Lösungsraum . . . . .	30
<b>4. Implementierung und Tests</b>	<b>32</b>
4.1. Architektur . . . . .	32
4.1.1. Implementierung . . . . .	36
4.2. Testergebnisse . . . . .	42
4.2.1. Die synthetischen Beispiele aus Anhang B . . . . .	42
4.2.2. Test an verrauschten Daten . . . . .	49
4.2.3. Hyperparametersuche . . . . .	51
<b>5. Diskussion und Ausblick</b>	<b>55</b>
<b>A. Benutzte Frameworks</b>	<b>58</b>
A.1. DeepXDE . . . . .	58
A.2. FEniCS . . . . .	59

Inhaltsverzeichnis	v
<b>B. Betrachtete Beispiele</b>	<b>62</b>
<b>C. Inhalte der vollen Masterarbeit</b>	<b>66</b>
C.1. Sourcecode . . . . .	66
C.2. PDF-Dateien . . . . .	66
C.3. Python Code . . . . .	66
<b>Quellenverzeichnis</b>	<b>68</b>
Literatur . . . . .	68

# Kapitel 1

## Einführung

### 1.1 Was ist Wärmeübertragung

Der physikalische Prozess bei dem die Wärme übertragen wird, gehört in den Bereich der Thermodynamik und wird als Wärmevertragung bezeichnet. Man hat festgestellt, wenn in einem Medium zwei unterschiedliche Temperaturniveaus auftreten, so fließt die Wärme aus dem Bereich des höheren Niveau in den mit der niedrigeren. Nach einiger Zeit tritt ein vollkommener Temperaturausgleich statt. Dieses Prinzip ist grundlegend und wird in vielen natürlichen sowie technischen Systemen beobachtet. Dabei erfolgt die Wärmeübertragung über drei grundsätzliche Mechanismen, die sich in *Wärmekonvektion*, *Wärmeleitung* und *Wärmestrahlung* aufteilen. Im Folgendem werden diese Begriffe knapp vorgestellt, eine umfangreichere Einführung bietet beispielsweise [8, Kap. 19-11].

#### Wärmekonvektion

Bei der *Konvektion* handelt es sich um Transport der Stoffmengen. Dieser Prozess findet in festen, flüssigen und gasförmigen Materialzuständen statt. Zum Beispiel bei Flüssigkeiten und Gasen tritt eine Ausgleichsströmung durch den Dichteunterschied zwischen warmen und kühleren Gebieten auf.

#### Wärmeleitung

Man stelle sich vor, wie ein Metallstab an einem Ende erhitzt wird. Nach einiger Zeit würde man feststellen, dass an dem anderen Ende eine Temperaturerhöhung aufgetreten ist. Im Umkehrschluss hieße es, dass die Wärme von dem Stabende, wo eine Wärmequelle angelegt wurde, zu dem anderem gewandert ist. Der physikalische Prozess, der im Hintergrund auftritt, ist auf die atomare Struktur des Metalls zurückzuführen. Also die Atome, wo eine Wärmequelle angelegt ist, erhöhen ihre Schwingungsamplituden, sie bekommen mehr Energie. Durch die Gitterstruktur im Metall regen die schon stärker schwingende Atome ihre Nachbarn an und so weiter, bis die Atome an dem anderem End-

de auch stärker schwingen. Somit wird die Wärmeenergie in Form schwingender Atome durch einen Festkörper transportiert, in solchen Fällen spricht man von *Wärmeleitung*.

Im Gegensatz zur Konvektion findet bei Wärmeleitung nur Energietransport statt, jedoch kein Massentransport. Zusätzlich unterscheidet man zwischen der Wärmeleitung in Festkörpern, Flüssigkeiten und Gasen. Die Unterschiede fußen auf der atomaren Struktur jeweiliger Stoffzustände.

## Wärmestrahlung

Im Vakuum, da wo keine Materie vorhanden ist, kann die Wärmeübertragung weder durch Wärmeleitung noch durch Konvektion stattfinden. Nehmen wir ein Beispiel der Wärmeübertragung von der Sonne zur Erde an. Trotzt der Tatsache, dass der Raum zwischen der Erde und Sonne (fast) leer von Materie ist, findet die Wärmeübertragung von der Sonne auf die Erde statt. Es ist die *Wärmestrahlung*, die für die Energieübertragung sorgt. Im Allgemeinen werden solche Prozesse durch das Konzept der elektromagnetischen Strahlung erklärt. Fakt ist, dass die elektromagnetische Strahlung kein Medium braucht um sich im Raum auszubreiten.

Einen tiefen Einblick in das Thema des Wärmetransports findet man beispielsweise in [6, Kap. 10.2].

## 1.2 Wärmeleitung: Direktes und inverses Problem

Die mathematische Beschreibung des Wärmeleitungsprozesses erfolgt mit der sogenannten *Wärmeleitungsgleichung*, also einer partiellen Differentialgleichung (DGL). Diese ermöglicht raumzeitliche Temperaturverteilung  $u$  in verschiedenen Materialien zu simulieren, zum Beispiel, um den Wärmefluss  $j$  in elektronischen Bauteilen zu optimieren. Damit können beispielsweise energieeffiziente Materialien gestaltet oder sicherheitsrelevante Prozesse im Bauwesen beeinflusst werden.

Für die Anschaulichkeit stelle man sich folgendes Setting vor: Eine quadratische Metallplatte wird von einer Wärmequelle  $f$  (z.B. einem Laserstrahl) an bestimmter Stelle erhitzt. In Abhängigkeit von den Materialeigenschaften wie Dichte  $\rho$ , Wärmekapazität  $c$  und der Wärmeleitfähigkeit  $a$  wird sich die der Platte zugeführte Wärme unterschiedlich verteilen.

Sind die materialspezifischen Eigenschaften und die Wärmequelle  $f$  bekannt, so kann die Wärmeleitungsgleichung (vorwärts) gelöst werden und man bekommt die Temperaturverteilung  $u$ . Diese Vorgehensweise nennt man das *direkte* Problem der Wärmeleitung. Eine andere Situation ist gegeben, wenn die Wärmequelle  $f$  bekannt ist und die Temperaturverteilung als Messwertreihe nur an einigen Stellen gemessen ist. Hier stellt sich die Frage, ob sich aus diesen Werten die Materialeigenschaften, wie beispielsweise die Wärmeleitfähigkeit bestimmen lassen. Das ist das sogenannte *inverse* Problem der Wärmeleitung.

meleitung, man spricht auch von einer Parameteridentifikation aus gemessenen Daten. Der beschriebene Prozess ist nicht der einzige relevante, es gibt eine ganze Reihe inverser Wärmeleitungsprobleme, siehe beispielsweise [17, Kap. 1.4].

Oft wird die Wärmeleitfähigkeit  $a$  als (räumlich) konstanter Wert angenommen, was für homogene Materialien zutrifft und für viele Anwendungen ausreichend ist. Umgekehrt kann eine ortsabhängige Wärmeleitfähigkeit  $a(x, y)$  Aufschlüsse über die Homogenität des Materials geben. In dem Beispiel der erhitzten Metallplatte könnte eine ortsabhängige Wärmeleitfähigkeit Hinweise über Anomalien im gegebenen Material liefern, zum Beispiel Lufteinschlüsse oder Fremdkörper. Das könnte bei der Qualitätssicherung der Materialien gute Dienste leisten.

### 1.3 Ziel: PINNs zur Lösung des inversen Problems

Die vorliegende Arbeit macht es sich zu Eigen, die *Wärmeleitung* zu untersuchen. Die Erkenntnisse aus der Analyse dieses Prozesses spielen in zahlreichen wissenschaftlichen und technischen Disziplinen eine tragende Rolle, insbesondere in der Materialforschung.

Diese Arbeit beschäftigt sich mit dem inversen Problem der Wärmeleitung zur Bestimmung einer ortsabhängigen Wärmeleitfähigkeit. Aus der mathematischen Perspektive sind inverse Probleme keineswegs trivial zu lösende Aufgaben. Die hier vorliegende kann mit Hilfe der *optimalen Kontrolltheorie für elliptische Operatoren* [4] zuverlässig gelöst werden. Der zitierte Ansatz beruht auf einer speziellen Lagrange-Methode für Sattelpunktprobleme welche letztendlich mit der FEM-Methode diskretisiert und gelöst werden können. Somit sind fundierte Kenntnisse der Funktionalanalysis fürs Verständnis dieses Ansatzes unabdingbar. Hinzu kommt, dass die zitierte Arbeit sich mit einem stationären Problem der Wärmeleitung<sup>1</sup> beschäftigt.

Ein anderer Ansatz das Problem zu lösen, ist ein moderner und kommt aus dem *Maschinellen Lernen*. Die sogenannten *Physics Informed Neural Networks* (PINN) wurden erst 2017 von Raissi et al. in [15] eingeführt. Im Wesentlichen beruht dieser Ansatz auf der universellen Approximationseigenschaft neuronaler Netze (NNs) und dem automatischen Differenzieren (AutoDiff)<sup>2</sup>. Letzteres ist ein schöner technischer Nebeneffekt der Optimierungstechniken der NNs. Anfänglich wurden PINNs zum lösen partieller Differentialgleichungen eingeführt. Erst die Hinzunahme der Informationen über physikalische Gesetzmäßigkeiten, die in der zu lösenden DGL enthalten sind, in das Training der NNs macht sie zu PINNs. Schon 2018 wird in [14] der Einsatz der PINNs zum lösen inverser Probleme erwähnt. Der Ansatz der PINNs hat sich seit dem stetig weiterentwickelt und es existieren inzwischen einige Software-Bibliotheken (Anhang A), die es ermöglichen

---

<sup>1</sup>Eigentlich wird in [4] ein Parameter des elliptischen Operators bestimmt, was aber direkt aus das stationäre Problem der Wärmeleitung übertragbar ist.

<sup>2</sup>AutoDiff macht die PINNs erst überhaupt möglich, denn neuronale Netze gab es auch lange bevor AutoDiff effizient umgesetzt werden konnte.

PINNs problembezogen einzusetzen.

In dieser Masterarbeit wird ein Lösungsansatz für das inverse Problem der Wärmeleitung mit PINNs vorgeschlagen. Genauer ausgedrückt, wird hier die Bestimmung der ortsabhängigen Wärmeleitfähigkeit  $a$  aus gemessenen Daten der Temperaturverteilung  $u$  und der Wärmequelle  $f$  mit PINNs, behandelt. In der ausgearbeiteten Methode werden vier Fälle des Problems betrachtet. Konkret heißt es, das inverse Problem wird in einer oder in zwei Raumdimensionen betrachtet und jeweils stationär oder instationär. Aus der Perspektive des maschinellen Lernens kann das vorliegende Problem unter die Regressionsprobleme eingeordnet werden. Dabei stellen die PINNs des inversen Problems der Wärmeleitung einen hybriden Lernansatz dar, bei dem ein Teil datengetrieben trainiert wird (hier gemessene Werte für  $u$  und  $f$ ) also *supervised*, und ein Teil nur die Struktur der Physik lernt (PDE-Residuum, Gl. 3.17), also *unsupervised*.

Als zentrales Gesamtergebnis dieser Arbeit ist ein eigenes, modular aufgebautes Framework (**InverseHeatSolver**) zur Lösung inverser Wärmeleitungsprobleme entstanden. Dieses Framework versteht sich als konsequente Erweiterung der bestehenden Open-Source-Bibliothek **DeepXDE** (Anhang A) und macht die modernen Konzepte der Physics-Informed Neural Networks für das konkrete Anwendungsfeld der Wärmeleitung nutzbar. Durch seine modulare Struktur kann das Framework flexibel erweitert und für weitere inverse Problemstellungen angepasst werden. Ein besonderer Vorteil besteht in der überschaubaren API, die auch für Nutzende ohne tiefe Vorkenntnisse in der Implementierung von PINNs einen unkomplizierten Einstieg ermöglicht. Das Framework ist so konzipiert, dass die Implementierung neuer Spezialfälle, zusätzlicher Regularisierungen oder alternativer Trainingsverfahren mit überschaubarem Aufwand erfolgen kann.

Die theoretische und praktische Ausarbeitung dieser Schrift befindet sich in den Kapiteln zwei bis vier. Im zweiten Kapitel wird auf Grundlage physikalischer Gesetze und mathematischer Zusammenhänge die partielle Differentialgleichung (pDGL, oder engl. PDE) der Wärmeleitung hergeleitet und anschließend das Problem der Wärmeleitung aufgestellt. Dann wird gezeigt, wie dieses Problem vorwärts gelöst werden kann um die Temperaturverteilung  $u$  zu bekommen und mit einem praktischen Beispiel bekräftigt. Mit der Kenntnis des Vorwärtsproblems wird nun sein Gegenstück, das inverse Problem eingeführt. Hier wird gezeigt, dass die Bestimmung der Wärmeleitfähigkeit  $a$ , selbst im stationären Fall, mathematisch kein eindeutig zu lösendes Problem ist. Eine Umformulierung des Problems kann helfen und wird kurz skizziert.

Im Kapitel drei wird der alternative Ansatz zum klassischen, also die PINNs vorgestellt. Zum Verständnis des Ansatzes werden hier zunächst die nötigen Grundlagen aus dem maschinellen Lernen in einer sehr komprimierten Fassung angegeben. Danach werden die PINNs für das inverse Wärmeleitungsproblem eingeführt und die Rolle der zu definierenden Verlustfunktion für physikalisch-informiertes Lernen hervorgehoben. Auf dieser Grundlage wird im Kapitel vier der endgültige PINN-Ansatz zum Lösen des inversen Problems der Wärmeleitung für eine ortsabhängige Wärmeleitfähigkeit vorgestellt.

Nach der Einführung des Ansatzes werden im vierten Kapitel die vier genannten Problemfälle mit Hilfe des Frameworks gelöst und im Anschluss ausführlich analysiert. Die Leistungsfähigkeit der entwickelten Methode wird anhand der Ergebnisse somit offen gelegt. Es offenbaren sich im praktischen Einsatz sowohl die Stärken als auch die aktuellen Grenzen dieses Ansatzes: Während die Anwendbarkeit und Verständlichkeit sowie die direkte und unkomplizierte Integration der Physik in den Ansatz einen erheblichen Fortschritt gegenüber klassischen Verfahren bedeuten, gibt es nach wie vor Herausforderungen. So zeigt sich, dass die Genauigkeit der Rekonstruktion oder der zeitliche Aufwand für das Anlernen der Modelle beträchtlich ist. Eine weitere Schwäche des PINN-Ansatzes ist die Generalisierbarkeit der trainierten Netzwerke, diese können noch nicht gesehene Problemstellungen nicht lösen. Also für jedes Problem, muss ein neues Netzwerk antrainiert werden. Nichtsdestotrotz verbirgt das Forschungsfeld der PINNs ein großes Potenzial komplexe Probleme ohne der komplexen Umformulierung zu lösen, was insbesondere für die Weiterentwicklung und den Transfer in andere Anwendungsfelder ein großer Vorteil ist.

## Kapitel 2

# Inverses Problem der Wärmeleitung

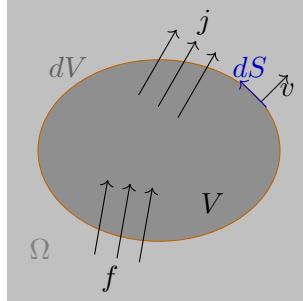
In diesem Kapitel wird die mathematische Grundlage des Wärmeleitungsproblems hergeleitet und analysiert. Den Ausgangspunkt der Analyse bildet die klassische Wärmeleitungsgleichung, die den zeitlichen Verlauf der Temperaturverteilung in einem Medium beschreibt. Die Herleitung erfolgt aus physikalischen Grundprinzipien wie dem Energieerhaltungssatz und dem Fourier'schen Gesetz. Im Anschluss wird das sogenannte Vorwärtsproblem behandelt, bei dem aus gegebenen Materialeigenschaften, wie die Wärmeleitfähigkeit und äußeren Einflüssen einer Wärmequelle die Temperaturverteilung bestimmt werden kann. Dieses Problem ist gut gestellt und mit klassischen numerischen Methoden wie Finite-Differenzen oder Finite-Elemente zuverlässig lösbar. Im Kontrast dazu steht das inverse Problem, das im dritten Abschnitt behandelt wird. Hier sollen aus gemessenen Temperaturen Rückschlüsse auf unbekannte Materialparameter, wie die Wärmeleitfähigkeit, gezogen werden. Aus der analytisch-formalen Perspektive gesehen handelt es sich um sogenannte schlecht gestellte Probleme, was einen direkten Zugang zur Lösung erschwert oder gar unmöglich macht. Ziel dieses Kapitels ist es, die Herausforderungen und Besonderheiten solcher inverser Wärmeleitungsprobleme theoretisch einzuordnen und die Grundlage für die in den Folgekapiteln entwickelten Lösungsansatz mit einer Methode des maschinellen Lernens zu legen.

### 2.1 Herleitung der Wärmeleitungsgleichung

Im Allgemeinen beschreibt die Wärmeleitungsgleichung die Ausbreitung der Wärme in einem Medium, welches beispielsweise ein räumlicher Gegenstand sein kann, etwa eine rechteckige Metallplatte. Man stelle sich vor, der Platte wird an einer Stelle Wärme, beispielsweise in Form eines Laserstrahls zugeführt. Folglich sagt unsere alltägliche Intuition, dass die erhöhte Temperatur auch an einer anderen Stelle der Platte zu spüren sein wird. Also man kann sagen, dass die Wärme sich von der erhitzen Stelle zu einer beliebigen auf der Platte ausgebreitet hat, oder sich darüber verteilt hat. Dieser Prozess wird Wärmeleitung genannt und ist im Kapitel 1.1 etwas anschaulicher beschrieben.

Die Frage nach der Temperaturverteilung in einem Medium kann mit einer Modellierung

der physikalischen Prozesse beantwortet werden. Das Modell wird *Wärmeleitungsgleichung* genannt und beruht auf dem Prinzip der *lokalen Bilanzgleichung* für eine Volumendichte einer *extensiven* Größe des gegebenen Systems [22, S.107].



**Abbildung 2.1.:** Schematische Darstellung zur Herleitung der Bilanzgleichung der Wärmeleitung. Wobei  $\Omega$  das Gebiet und  $V$  das Teilgebiet eines Mediums,  $dV$  Rand (orange) von  $V$ ,  $j$  Der Wärmestrom durch  $V$ ,  $v \in \mathbb{R}^3$  Normalevektor an  $dV$  und  $dS$  Tangente an  $dV$  sind.

Für den Prozess der Herleitung der Wärmeleitungsgleichung soll die Abbildung 2.1 als Gedankenstütze dienen. Betrachtet man nun auf dem Teilstück  $V \subset \Omega$  die Energiebilanz (Energieerhaltung) der Wärme in der Zeit, hieße es also:

*Die Gesamtänderung der Wärmeenergie auf  $V$  ist gleich der Differenz einfließender zu ausfließender Energimenge.*

Bevor dieser Zusammenhang formal aufgestellt werden kann, bedarf es einer Definition der darin vorkommenden Begrifflichkeiten, die hier ohne physikalische Einheiten betrachtet werden. Nun wird festgehalten:

**Definition 1.** Auf dem Gebiet  $\Omega \subset \mathbb{R}^2$  und einem Zeitintervall  $T = [t_0, t_1] \subset \mathbb{R}$  seien folgende Abbildungen definiert:

- (i) Die Wärmeenergie  $e : \Omega \times T \rightarrow \mathbb{R}$ ,
- (ii) Die Wärmequelle  $f : \Omega \times T \rightarrow \mathbb{R}$ ,
- (iii) Die Temperaturverteilung  $u : \Omega \times T \rightarrow \mathbb{R}$ ,
- (iv) Der Wärmefluss  $j : \Omega \times T \rightarrow \mathbb{R}^2$ .

Um die Physik zu berücksichtigen müssen gewisse Modellannahmen getroffen werden. In diesem Falle ist es die Energieänderung und das *Fourier'sche Gesetz*, die wie folgt aufgefasst werden können:

1. Die Energieänderung wird nur durch die Temperaturänderung hervorgerufen, damit gilt folgende Beziehung

$$\frac{\partial e}{\partial t} = \frac{\partial e}{\partial u} \frac{\partial u}{\partial t} = \rho c \frac{\partial u}{\partial t}. \quad (2.1)$$

Wobei  $\rho$  die Dichte und  $c$  die spezifische Wärmekapazität bezeichnen.

2. Wie der Wärmefluss  $j$  und die Temperaturverteilung  $u$  von einander abhängen, gibt das *Fourier'sche Gesetz*<sup>3</sup> wieder:

*Die Wärmeenergie strömt immer von Wärmeren Regionen zu den kälteren. Die Flussrate des Stroms ist proportional zum Temperaturgradienten.*

Eine gewisse Eigenschaft, dass jedes Material individuell hat, ist die *Wärmeleitfähigkeit*  $a$ . Sie ordnet jedem Punkt im Körper eine positive Zahl zu, also  $a : \Omega \rightarrow \mathbb{R}_+$ . Beide Überlegungen führen zu der Beziehung:

$$j = -a\nabla u. \quad (2.2)$$

**Bemerkung 1.** *Dass die Wärmeleitfähigkeit nur von den Raumkoordinaten abhängt, ist eine vereinfachende Annahme, im Allgemeinen hängt sie noch von der Temperatur  $u$  ab.*

Wir kehren zurück zu der Energiebilanz. Möchte man die Gesamtänderung der Wärmeenergie durch den Wärmefluss messen, so ist es sinnvoll diesen am Rand  $dV$  eines Teilstücks  $V \subset \Omega$  in Richtung der Normalen  $v$  zu berechnen. Formal kann es mit Hilfe des Skalarprodukts angegeben werden, also  $\langle j, v \rangle$  (Abb. 2.1).

Nun zur Aufstellung der Wärmeleitungsgleichung, summiert man die Wärmeenergieänderung über alle  $V \subset \Omega$  ergibt sich folgender Ausdruck

$$\frac{\partial}{\partial t} \int_{\Omega} e dx = \int_{\Omega} f dx - \int_{\partial\Omega} \langle j, v \rangle dS. \quad (2.3)$$

Verlangt man, dass die Wärmeenergie eine stetig differenzierbare Funktion ist, also  $e \in C^1(\Omega \times T)$ , dann kann der Differentialoperator  $\frac{\partial}{\partial t}$  unter das Integral in der Gleichung (2.3) gezogen werden. Mit dem Einsetzen der Gleichungen (2.1) in die obige (2.3) bekommt man folgenden Ausdruck

$$\int_{\Omega} \rho c \frac{\partial u}{\partial t} dx = \int_{\Omega} f dx - \int_{\partial\Omega} \langle j, v \rangle dS. \quad (2.4)$$

Da das Gebiet, auf dem die Temperaturverteilung nicht weiter spezifiziert war, würden die Integrale wegfallen, wenn man das letzte Randintegral in ein Gebietsintegral in (2.4) umschreiben könnte. An dieser Stelle kann der *Gauß'schen Satz* die nötige Hilfe leisten.

---

<sup>3</sup> J. B. Fourier, 1768-1830

**Definition 2. Der Gauß'sche Satz**

Sei  $\mathcal{M} \subset \mathbb{R}^n$  kompakt mit abschnittsweise glattem Rand  $S = \partial\mathcal{M}$ , der Rand ist orientiert durch äußeres Normaleneinheitsvektorfeld  $n$ . Das Vektorfeld  $F$  sei stetig differenzierbar auf einer offenen Menge  $\mathcal{K}$  mit  $\mathcal{M} \in \mathcal{K}$ . Dann gilt

$$\int_{\mathcal{M}} \langle \nabla, F \rangle dV = \int_{\partial\mathcal{M}} \langle F, n \rangle dS. \quad (2.5)$$

Nimmt man an, dass das Gebiet  $\Omega$  und der Wärmefluss  $j$  so spezifiziert sind, dass sie die Voraussetzungen des Satzes erfüllen, dann folgt

$$\int_{\Omega} \rho c \frac{\partial u}{\partial t} dx = \int_{\Omega} f dx - \int_{\Omega} \langle \nabla, j \rangle dx. \quad (2.6)$$

Setzt man die Annahme (2.2) in (2.6) ein und lässt die Integrale weg, folgt daraus die besagte Wärmeleitungsgleichung

$$\rho c u_t - \langle \nabla, a \nabla u \rangle = f. \quad (2.7)$$

Wobei hier der zeitliche Differenzialoperator der Temperatur  $\frac{\partial u}{\partial t}$  der Bequemlichkeit halber durch  $u_t$  ersetzt wurde.

Die Gleichung 2.7 beschreibt ein noch unbestimmtes System. Die sogenannten Anfangs- und Randbedingungen würden das Problem mehr spezifizieren. Jedes physikalische System besitzt zu Beginn der Messung oder der Beobachtung einen gewissen Anfangszustand  $u_0$ . Oft wird die Zeit  $t$  zu Beginn der Beobachtung auf den Wert 0 gesetzt, somit erhält man eine Anfangsbedingung.

$$u(x, 0) = u_0(x), \quad x \in \Omega. \quad (2.8)$$

Die Randbedingungen geben die Verteilung der gesuchten Größe am Rand  $\partial\Omega$  des Gebiets  $\Omega$  an. Im Falle der Wärmeleitungsgleichung erhält man also:

$$u(x, t) = g(x, t), \quad x \in \partial\Omega \quad (2.9)$$

Nun kann man die Gleichungen 2.7, 2.8 und 2.9 zusammen in einer Rand- und Anfangswertaufgabe (RAWP) der Wärmeleitung zusammenfassen, also:

**Definition 3. Rand- und Anfangswertaufgabe der Wärmeleitung mit nicht-konstantem Wärmeleitungskoeffizienten  $a$ .**

Auf einem Gebiet  $\Omega \subset \mathbb{R}^d$ ,  $d = 1, 2, 3$  beschreibt das folgende Modell

$$\begin{aligned} \rho c u_t - \langle \nabla, a \nabla u \rangle &= f \\ u(x, 0) &= u_0(x), \quad x \in \Omega \\ u(x, t) &= g(x, t), \quad x \in \partial\Omega \end{aligned} \quad (2.10)$$

die Rand- und Anfangswertaufgabe der Wärmeleitung zur Bestimmung von  $u$  für alle  $t \in T \subset \mathbb{R}$ .

Die Definition 2.10 stellt somit eine partielle Differentialgleichung (pDGL, eng. PDE) zweiter Ordnung mit Nebenbedingungen dar, oder ein *instationäres* Wärmeleitungsproblem. Aus der Theorie der Quadriken geht hervor, dass diese Gleichung einem *parabolischen* Typ entspricht. Die Typisierung der pDGL'n gibt Aufschlüsse über ihre Lösungsmengen und Lösungsansätze. Eben die Theorie über lineare parabolische DGL'n behandelt die Lösungsansätze dieses Problems.

Ist die zeitliche Ableitung  $u_t = 0$ , so spricht man von einer *stationären* Randwertaufgabe, auf einem Gebiet  $\Omega \subset \mathbb{R}^d$ ,  $d = 1, 2, 3$  folgende Form annimmt:

$$\begin{aligned} -\langle \nabla, a \nabla u \rangle &= f \\ u(x) &= g(x), \quad x \in \partial\Omega \end{aligned} \tag{2.11}$$

(2.11) fällt unter die lineare elliptische Theorie der DGL'n. Im Bezug auf *inverses* Problem der Wärmeleitung im Abschnitt 2.3, wird diese Gleichung einige Aufschlüsse eben zum inversen Problem liefern.

Die oft verbreitete Annahme, dass der Wärmeleitfähigkeit konstant ist, also  $a' = 0$ , führt auf die Gleichung von der Form:

$$\begin{aligned} &\rho c u_t - \langle \nabla, a \nabla u \rangle \\ &= \rho c u_t - \nabla a \nabla u - a \Delta u \\ &= \rho c u_t - a \Delta u = f. \end{aligned}$$

Somit vereinfacht sich das Problem ein wenig, zumal die Wärmeleitfähigkeit  $a$  nicht im DGL-Operator ist, sondern als Multiplikator auf ihn wirkt. Diese Gleichung wird auch mit den Methoden der elliptischen Theorie behandelt werden..

In beiden Fällen, ob es der *Divergenzoperator*  $\langle \nabla, \cdot \nabla \cdot \rangle$  oder der *Laplace-Operator*  $\Delta$  ist, wenn die Gleichung nach der Temperaturverteilung  $u$  gelöst werden muss, liegen symmetrisch, positiv definite Operatoren vor. Diese sind invertierbar und führen zur Lösung, also der Temperaturverteilung  $u$ . Diese Art die DGL zu lösen, nennt man auch *Vorwärtsproblem* und ist in der Theorie sehr umfangreich untersucht worden. Eine sehr weitreichende Untersuchung zu diesem Thema ist beispielsweise in [19], in den Kapiteln über lineare Theorie dargestellt.

**Bemerkung 2.** Die Materialkonstanten  $\rho$  und  $c$  werden der Einfachheit halber auf 1 gesetzt. Diese Vereinfachung beeinträchtigt die qualitative Untersuchung des Problems nicht und wird auf diese Weise unten im Text fortgeführt.

Im Folgenden wird das Problem (2.10) näher untersucht. Dabei wird es nicht um den umfangreichen mathematischen Formalismus handeln, vielmehr um das Aufgreifen der Ideen, wie man an dieses Problem praktisch herangehen könnte.

## 2.2 Vorwärtsproblem der Wärmeleitungsgleichung (WLG)

Eine etablierte Methode zur numerischen Lösung elliptischer und parabolischer partieller DGL'n ist die Finite-Elemente-Methode (FEM). Dieser Ansatz beruht auf der schwachen Formulierung des zugrunde liegenden Problems, welche ihren Ursprung in der Variationsrechnung hat. Ziel der schwachen Formulierung ist es, die Anforderungen an die Regularität der Lösung zu reduzieren, insbesondere durch die Vermeidung höherer Ableitungen. Im Falle der Wärmeleitungsgleichung (2.10) betrifft dies den Diffusionsterm  $\langle \nabla, a \nabla u \rangle$ , der in der klassischen Formulierung die Existenz von zweiten partiellen Ableitungen der Temperaturverteilung  $u$  voraussetzt.

Formal führt schwache Formulierung zu einer Erweiterung des Lösungsraums: Während die klassische (starke) Formulierung Lösungen in einem Raum wie  $C^2(\Omega)$  verlangt, erlaubt die schwache Formulierung Lösungen im Sobolev-Raum  $H^1(\Omega)$ , also Funktionen mit quadratintegrierbaren ersten Ableitungen. Dies ist eine Verallgemeinerung, auf der numerisch robuste Verfahren wie die FEM anwendbar sind.

### 2.2.1 Schwache Formulierung

Die Variationsformulierung (schwache Form) wird durch Multiplikation der PDE mit einer Testfunktion  $v$  und Integration über das Gebiet  $\Omega$  hergeleitet. Dieser Schritt erlaubt es, auch Lösungen  $u$  zu berücksichtigen, die möglicherweise nicht überall differenzierbar sind. Wendet man das Beschriebene auf die Gleichung 2.10 an, so ergibt sich zunächst

$$\int_{\Omega} v \frac{\partial u}{\partial t} dx - \int_{\Omega} v \langle \nabla, a \nabla u \rangle dx = \int_{\Omega} v f dx. \quad (2.12)$$

Im nächsten Schritt wird durch die Anwendung des Gaußschen Satzes aus dem Volumenintegral ein Oberflächenintegral umgewandelt:

$$\int_{\Omega} v \langle \nabla, a \nabla u \rangle dx = \int_{\partial\Omega} v \langle a \nabla u, \mathbf{n} \rangle dS - \int_{\Omega} \nabla v \cdot (a \nabla u) dx. \quad (2.13)$$

Hier ist  $\partial\Omega$  der Rand des Gebiets und  $\mathbf{n}$  der nach außen gerichtete Normalenvektor. Verlangt man von der Lösung, dass diese auf dem Rand verschwinden soll (Neumann-Randbedingung), dann fällt das Integral über den Rand  $\partial\Omega$  zu Null und man erhält durch das Einsetzen von (2.13) in (2.12) nun die schwache Formulierung der Wärmeleitungsgleichung

$$\int_{\Omega} v \frac{\partial u}{\partial t} dx + \int_{\Omega} a \langle \nabla u, \nabla v \rangle dx = \int_{\Omega} vf dx. \quad (2.14)$$

Die Form (2.14) ist die Grundlage für Lösungsverfahren wie FEM. Für die Anwendung des Verfahrens muss im nächsten Schritt eine Raumdiskretisierung stattfinden. In der Praxis kommt oft eine Dreieckdiskretisierung des Gebiets  $\Omega$  zum Einsatz. Ist das Problem nicht stationär, so muss die zeitliche Komponente auch diskretisiert werden. Beispielsweise könnte die Gleichung (2.14) in jedem diskreten Schritt die folgende Gestalt annehmen

$$\int_{\Omega} \frac{u^{n+1} - u^n}{\Delta t} v dx + \int_{\Omega} a^{n+1} \nabla u^{n+1} \cdot \nabla v dx = \int_{\Omega} vf^{n+1} dx. \quad (2.15)$$

Dabei wurde jeder Zeitschritt  $t_{n+1}$  mit dem Backward-Euler-Verfahren approximiert

$$\frac{\partial u}{\partial t} \approx \frac{u^{n+1} - u^n}{\Delta t}. \quad (2.16)$$

Insgesamt führt dieser Ansatz zu einem System von Gleichungen, das für jeden Zeitschritt numerisch gelöst werden muss. Eine eigenhändige Implementierung des Verfahrens ist unter Umständen sehr aufwändig, jedoch existieren zahlreiche freie und kommerzielle Software-Bibliotheken. Eine dieser etablierten frei verfügbaren Bibliotheken ist FEniCS und wird in dieser Arbeit eingesetzt. Wie man die Bibliothek unter Windows benutzen kann ist in dem Abschnitt A.2 beschrieben. An dieser Stelle soll es nun mit einem konkreten Beispiel implementiert mit FEniCS weiter gehen.

### 2.2.2 Beispiel einer Vorwärtssimulation mit FEniCS

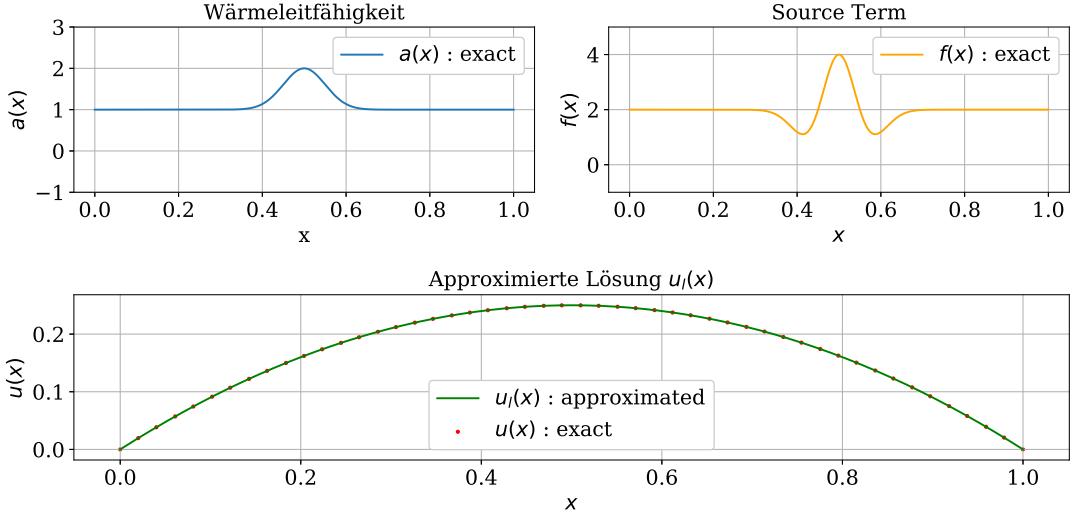
Seien hier die Funktionen (B.1) als raumabhängiger Wärmeleitkoeffizient  $a(x)$  und (B.3) die Wärmequelle  $f(x)$  aus dem stationären eindimensionalen Fall betrachtet. Die Implementierung kann in der Datei `FEniCS_1D_ti.py` eingesehen werden<sup>4</sup>. Im Wesentlichen folgt die allgemeine Vorgehensweise diesen vier Schritten:

1. Definiere die Geometrie und die Zeitschritte,
2. Definiere die Eingabeparameter, solche wie  $a$  und  $f$  und die Bedingungen, wie Anfangs- und Randwerte,
3. Definiere das Variationsfunktional, hier (2.15)
4.
  - i) Löse das Funktional aus 3 direkt, falls stationäres, oder
  - ii) Löse das Funktional für jeden Zeitschritt  $\Delta t$ , falls ein instationäres Problem vorliegt.

---

<sup>4</sup>Es liegt eine eigene Implementierung vor, da konkret für das hier betrachtetes Problem keine Internetquelle gefunden werden konnte.

Die Lösung für die eingangs festgelegten Inputs (B.1) und (B.2) ist in der Abbildung 2.2 zu sehen. Das Intervall  $\Omega = [0, 1]$  wurde hier mit 1100 äquidistanten Punkten diskretisiert.



**Abbildung 2.2.:** Es sind abgebildet: Die approximierte Lösung  $u_l(x)$  der eindimensionalen stationären WLG (2.11) (unten), die vorgegebene ortsabhängige Wärmeleitfähigkeit  $a(x)$  (oben links) und die ortsabhängige Heizdichte  $f(x)$  (oben rechts). Das Intervall  $\Omega = [0, 1]$  wurde mit 1100 Punkten diskretisiert, die Rechenzeit betrug dabei etwa 5.16s und der  $L^2$  Normabstand der Lösung zur exakten Funktion ist  $\|u(x) - u_l(x)\|_{L^2} = 1.6 \cdot 10^{-14}$ . Die exakte Funktion wurde der Übersichtlichkeit halber mit 50 Punkten wiedergegeben. Dieses Beispiel wurde mit der FEniCS-Library im Skript `FEniCS_1D_ti.py` gelöst.

Die Ergebnisse dieses Beispiels sollen als Referenzwerte für spätere Untersuchung dienen. Insbesondere der stationäre Fall für die Gleichungen (B.2) und (B.3), hier hat man mit einer Gitterauflösung von 1100 Punkten über  $\Omega$  eine Rechenzeit von 5.16s und eine Genauigkeit gegenüber der exakten Lösung von  $1.6 \cdot 10^{-14}$  erzielt. Die entstandenen Ergebnisse können für Testzwecke des inversen Problems, das im nächsten Abschnitt vorgestellt werden soll, benutzt werden.

### 2.3 Inverses Problem der Wärmeleitung

Bei der klassischen Lösung des Problems (5) auf einem Gebiet  $\Omega \subset \mathbb{R}^n, n = 1, 2$  und über die Zeit  $T \subset \mathbb{R}_+$  wird die unbekannte Temperaturverteilung  $u : \Omega \times T \rightarrow \mathbb{R}$  gesucht. Dabei ist davon auszugehen, dass  $f : \Omega \times T \rightarrow \mathbb{R}$  und  $a : \Omega \rightarrow \mathbb{R}_+$  bekannt sind. Dieses Vorgehen nennt man das *direkte Problem*, oder *Vorwärtsproblem*.

Für die Lösung solcher Probleme gibt es eine Fülle bewährter Verfahren, zum Beispiel die Finite-Elemente-Methode (Eigenes Beispiel in 2.2.2 oder eine ausführliche theoretische Diskussion in [2]).

In der Praxis ist es oft so, dass man gewisse Messwerte vorliegen hat. Messwerte sind die Ergebnisse einer *Ursache* und werden *Wirkungen* genannt. Oft steht die Frage eher so: Welche Ursache hat eine Wirkung hervorgerufen? Konkret hier, am Beispiel der Metallplatte, an die eine Wärmequelle  $f$  angesetzt wurde, wird die resultierende Temperaturverteilung  $u$  gemessen. Nun kann man sich fragen, welche intrinsischen Eigenschaften der Metallplatte wirkten auf die Wärmequelle  $f$ , so dass die gemessene Temperaturverteilung  $u$  zustande gekommen ist? Die Antwort ist, es sind die physikalischen Eigenschaften der Metallplatte. Im Zusammenhang mit der Wärmeleitung sind es solche wie die Dichte  $\rho$ , die spezifische Wärmekapazität  $c$  und die Wärmeleitfähigkeit  $a$ . Wenn man die Vereinbarung aus der Bemerkung 2 weiterhin bestehen lässt, bleibt es hier nur die Wärmeleitfähigkeit  $a$ , die die spezifische Temperaturverteilung  $u$  aus der Wärmequelle  $f$  verursacht. Die Bestimmung der Wärmeleitfähigkeit  $a$  aus einer z.B. durch Messungen bekannten Temperaturverteilung  $u$ , wird als *inverses Problem* bezeichnet. Man spricht in diesem Fall auch von dem Problem der Parameteridentifizierung, also von der Bestimmung von Materialeigenschaften wie der Wärmeleitfähigkeit. Eine abstrakte Formulierung, in Anlehnung an [17, S. 14], fasst diesen Zusammenhang folgendermaßen auf:

**Definition 4.** Ein *mathematisches Modell* ist eine Abbildung

$$A : X \rightarrow Y$$

von der Menge der Ursachen (Parameter)  $X$  in die Menge der Wirkungen (Daten)  $Y$ . Als **direktes Problem** bezeichnet man die Bestimmung der Wirkung  $y = A(x)$  bei bekannter Ursache  $x$ . Im Falle des **inversen Problems** wird zu einer Wirkung  $y \in Y$  die Ursache  $x \in X$  gesucht, sodass  $Ax = y$  erfüllt ist.

In der Funktionalanalysis werden Werkzeuge bereitgestellt, die es erlauben, Operatorgleichungen zu untersuchen. Das kann insbesondere Informationen zum Verhalten des Operators bei inversen Fragestellungen liefern. Daher soll die DGL der Wärmeleitung aus der Definition 5 als eine Operatorgleichung aufgefasst werden. Dafür kann die obige Definition 4 herangezogen werden und man erhält

**Definition 5.** Sei  $\Omega \subset \mathbb{R}^d$ ,  $d = 1, 2, 3$  und  $t \in T \subset \mathbb{R}$ , zu gegebener Temperaturverteilung  $u : \Omega \times T \rightarrow \mathbb{R}$  und bekannter Wärmequelle  $f : \Omega \times T \rightarrow \mathbb{R}$  soll die Wärmeleitfähigkeit  $a : \Omega \rightarrow \mathbb{R}$  so bestimmt werden, dass das Problem 2.10 erfüllt ist. Dieses Vorgehen wird als **inverse Problem der Wärmeleitung für die ortsabhängige Wärmeleitfähigkeit  $a$**  bezeichnet.

In der Theorie stellt sich leider heraus, dass alleine die Untersuchung des stationären Problems

$$-\langle \nabla, a \nabla u \rangle = f \tag{2.17}$$

recht aufwendig ist. Im Allgemeinen lässt sich die Gleichung (2.11) in die Theorie der *Parameteridentifizierung in elliptischen Systemen* einordnen. Das heißt, bezogen auf das inverse Problem der Wärmeleitfähigkeit  $a$ , verantworten die Eigenschaften des Operators  $\langle \nabla, \cdot \nabla \cdot \rangle$  die Lösbarkeit des Problems. Um den Operator in seinem Verhalten bezüglich der Lösung einordnen zu können, wurde in der Theorie der inversen Problemen der Begriff der *Gutgestelltheit (well-posedness)* eingeführt, seine Definition lautet (nach [17, S. 15]):

**Definition 6.** Sei  $A : X \rightarrow Y$  eine Abbildung zwischen den topologischen Räumen  $X$  und  $Y$ . Das Problem  $(A, X, Y)$  heißt gut gestellt (well-posed), wenn folgende Eigenschaften erfüllt sind.

- (a) Die Gleichung  $Ax = y$  hat für jedes  $y \in Y$  eine Lösung.
- (b) Diese Lösung ist eindeutig bestimmt.
- (c) Die inverse Abbildung  $A^{-1} : Y \rightarrow X$  ist stetig, d.h. die Lösung  $x$  hängt stetig von den Daten  $y$  ab (kleine Störungen in  $y$  bewirken kleine Störungen in  $x$ ).

Ist eine der Bedingungen verletzt, so heißt das Problem schlecht gestellt (ill-posed).

In der Praxis stellt sich leider heraus, dass praktisch alle inverse Probleme schlecht gestellt sind. Und oft ist die Bedingung (c) der obigen Definition verletzt, da inverse Probleme im Zusammenhang mit gemessenen Daten Auftreten und bekannterweise sind diese oft verrauscht. Das hieße, wenn ein Operator ohnehin unstetig ist, wird seine Datenempfindlichkeit durch verrauschte Daten nur noch erhöht. Anschaulich äußert sich die Datenempfindlichkeit dadurch, dass der Graph der Funktion  $a$  wellig sein kann und nicht glatt ist.

**Bemerkung 3.** In einem stationären Fall auf einem Gebiet  $\Omega \subset \mathbb{R}$ , also  $-\langle \nabla, a \nabla u \rangle = f$ , existiert keine eindeutige Lösung, wenn  $\nabla u(\Omega) = 0$ . Das heißt für jedes beliebigen  $a$  würde die Gleichung erfüllt sein. Damit ist in der Definition 6 die Bedingung (a) verletzt, aus der praktischen Sicht hieße es, man hat eine konstante Temperaturverteilung  $u$  gemessen.

Die unstetige Abhängigkeit von den Daten kann an einem konkreten Beispiel nachvollzogen werden ([7, S. 334]). Auf  $\Omega = (0, 1)$  soll folgendes Problem

$$(a(x)u'(x))' = 0 \quad \text{für } x \in \Omega, \quad u(0) = 0, \quad u(1) = 1,$$

dessen Lösung  $u(x) = x$  ist, betrachtet werden. Nun soll ein weiteres Problem mit einer periodischen Wärmeleitfähigkeit

$$a^\varepsilon(x) = a\left(\frac{x}{\varepsilon}\right) \quad \text{mit } a : \mathbb{R} \rightarrow \mathbb{R} \text{ 1-periodisch,}$$

mit  $\varepsilon = \frac{1}{n}$ ,  $n \in \mathbb{N}$ , in Betracht gezogen, also

$$(a^\varepsilon(x)u^\varepsilon(x))' = 0 \quad \text{für } x \in \Omega, \quad u^\varepsilon(0) = 0, \quad u^\varepsilon(1) = 1.$$

Die Integration der DGL und Berücksichtigung der Randbedingungen führt zum folgenden Ausdruck:

$$u^\varepsilon(x) = \frac{\int_0^x \frac{1}{a^\varepsilon(y)} dy}{\int_0^1 \frac{1}{a^\varepsilon(y)} dy}$$

Die Integration periodischer Funktionen und Transformation der Variablen erlauben es, den obigen Ausdruck wie folgt umzuschreiben

$$u^\varepsilon(x) = \frac{\int_0^{nx} \frac{1}{a(y)} dy}{n \int_0^1 \frac{1}{a(y)} dy}.$$

Eine Erweiterung dieses Ausdrucks mit einem Nullterm  $x - x = 0$  ergibt

$$\begin{aligned} u^\varepsilon(x) &= \frac{\int_0^{nx} \frac{1}{a(y)} dy}{n \int_0^1 \frac{1}{a(y)} dy} + (x - x) \\ &= x + \frac{1}{n \int_0^1 \frac{1}{a(y)} dy} \left( \int_0^{nx} \frac{1}{a(y)} dy - xn \int_0^1 \frac{1}{a(y)} dy \right). \end{aligned}$$

Man sieht, dass im Grenzwertfall für  $\varepsilon \rightarrow 0$ , also  $n \rightarrow \infty$  konvergiert  $u^\varepsilon \rightarrow u$  gleichmäßig, weil  $\frac{1}{n \int_0^1 \frac{1}{a(y)} dy} \rightarrow \infty$  gilt und  $\int_0^{nx} \frac{1}{a(y)} dy - xn \int_0^1 \frac{1}{a(y)} dy$  beschränkt ([7, S. 335])<sup>5</sup> ist.

Dieses Verhalten zeigt, dass man für zwei bestimmte Wärmeleitfähigkeiten  $a$  und  $a^\varepsilon$  ein und das selbe  $u$  bekommen kann.

Die schlechte Gestelltheit des Problems verlangt nach alternativen Ansätzen. Oft ist es so, dass man die Lösung eines nicht invertierbaren Systems durch eine Approximation annähert. In diesem Zusammenhang entstehen sogenannte *Minimierungsaufgaben*, die durch eine Umformulierung des ursprünglichen Problems zustande kommen, ähnlich der schwachen Formulierung aus dem Abschnitt 2.2.1. Einer solcher Ansätze ist beispielsweise in einer Arbeit von Kueng und Zou [10]<sup>6</sup> vorgeschlagen. Hier wird folgendes äquivalentes zu 5 Problem behandelt

---

<sup>5</sup>Die Beschränktheit beruht auf der Periodizität von  $a$ . Mit der Mittelung  $M := \int_0^1 \frac{1}{a(y)} dy$  erhält man die Abschätzungen  $-1 \leq \lfloor nx \rfloor - nx \leq 0 \implies |M(\lfloor nx \rfloor - nx)| \leq |M|$  und  $\left| \int_0^{nx} \frac{1}{a(y)} dy - M \lfloor nx \rfloor \right| \leq C$  wonach dann gilt:  $\left| \int_0^{nx} \frac{1}{a(y)} dy - xn \int_0^1 \frac{1}{a(y)} dy \right| \leq |M| + C$ .

<sup>6</sup>In der zitierten Arbeit, wird ausdrücklich gesagt, dass der Operator in der Gleichung 2.17 für  $a$  schlecht-gestellt ist.

$$J(a, u) = \frac{1}{2} \int_{\Omega} |\nabla u - \nabla \tilde{u}|^2 dx + \lambda N(a)$$

$$(a, u) \in K \times V \text{ und } e(a, u) = 0,$$

das zu minimieren ist. Wobei  $u$  die Lösung von (2.17) und  $\tilde{u}$  die Messwerte sind.  $\lambda$  ist ein Gewichtsparameter der Regularisierung  $N(a)$ , die noch neben dem Funktionenraum  $V$  zu spezifizieren ist. Allerdings wird hier eine Bedingung an  $a$  gestellt, es muss gelten :  $K = \{a \in L^1(\Omega) : |a| < \infty \wedge \forall x \in \Omega : \alpha_1 \leq a(x) \leq \alpha_2\}$ , das heißt  $a$  liegt in einem positiven Bereich  $0 < \alpha_1 \leq \alpha_2$ .  $e(a, u)$  ist das Residuum der Gleichung (2.17), welches in der schwachen Formulierung die Form annimmt :

$$(\nabla e(a, u), \nabla v) = \int_{\Omega} \langle a \nabla u, \nabla v \rangle dx - \int_{\Omega} v f dx \text{ mit } v \in V.$$

Nach den Vorbereitungen wird gezeigt, dass das *Lagrange-funktional* (erweiterte Lagrange-Methode) die Bedingung  $e(a, u) = 0$  aufweicht und die Konvexität des zu minimierenden Ziel-Funktional verstärkt:

$$\mathcal{L}_r(a, u, v) = J(a, u) + (\nabla e(a, u), \nabla v) + \frac{1}{2} \|\nabla e(a, u)\|_{L^2(\Omega)}^2$$

Das obere Funktional ist zu minimieren, wobei für die Diskretisierung der schwachen Formulierung der FEM Ansatz gewählt wurde. Dies ist eine sehr grobe Darlegung des klassischen Ansatzes und lässt annehmen, dass dieser ein Feingefühl für Funktionenräume, Normen und Regularisierungstechniken verlangt und dass man bei der Umsetzung gegebenenfalls mit höherem Rechenaufwand zu rechnen hat.

In den letzten Jahren hat sich jedoch ein vielversprechender neuer Ansatz herauskristallisiert: die Nutzung moderner Methoden des maschinellen Lernens, insbesondere sogenannter Physics-Informed Neural Networks. Diese kombinieren datengetriebene Modellierung mit physikalischem Vorwissen in Form partieller Differentialgleichungen und eröffnen damit einen alternativen Zugang zur Lösung inverser Probleme. Das folgende Kapitel widmet sich daher der Einführung und Untersuchung dieses Ansatzes bezogen auf das Problem 5.

## Kapitel 3

# Physics Informed Neural Networks (PINNs)

Der methodische Einsatz der PINNs zum Lösen der pDGL's wurde etwa 2017 von Raissi et al. in [15] vorgestellt. Die grundlegende Idee der PINNs beruht auf zwei wesentlichen Aspekten, einerseits auf der Fähigkeit neuronaler Netze, als universelle Funktionsapproximatoren zu wirken, und andererseits auf dem Einsatz von automatischem Differenzieren<sup>7</sup> während des Trainingsprozesses. Das Letztere ist eher der technische Aspekt, der bei der Implementierung des Gradientenabstiegsverfahrens für NNs entsteht. Das Innovationspotenzial des Ansatzes der PINNs heben die Autoren von [15] wie folgt hervor:

*This simple yet powerful construction allows us to tackle a wide range of problems in computational science and introduces a potentially disruptive technology leading to the development of new data-efficient and physics-informed learning machines, new classes of numerical solvers for partial differential equations, as well as new data-driven approaches for model inversion and systems identification.*

Wie diese Aspekte nun tatsächlich zur Sache beitragen, soll eine formale Darlegung der dahinterstehenden Ideen des statistischen Lernens zumindest skizzenhaft im Folgendem angedeutet werden.

### 3.1 Statistisches Lernen

In der *statistischen Entscheidungstheorie* versucht man das *Risiko R* einer *Entscheidungsregel f* so zu minimieren, dass die zugrunde liegende Verteilung  $\mathbb{P}^{(X,Y)}$  der Daten  $X, Y \in (\Omega, \mathcal{A}, \mathbb{P})$  möglichst gut approximiert wird. Diese Art des statistischen Lernens wird auch *supervised learning* genannt.

---

<sup>7</sup> Der Unterschied zwischen dem *automatischen Differenzieren* (AD) bei neuronalen Netzen und dem *algorithmischen Differenzieren* (Algorithmic Differentiation) liegt nicht im Prinzip, sondern im Kontext und der konkreten Implementierung. Beide beruhen auf denselben mathematischen Grundlagen – aber sie unterscheiden sich in ihrer Anwendung und Optimierung.

In den meisten Anwendungsfällen liegen die Daten in Form von Beobachtungen oder Messungen vor, es sind die sogenannten, womöglich fehlerbehafteten und unvollständigen Trainingsdaten  $T_n = \{(x_i, y_i) | x_i \in \mathcal{X} \subset X(\Omega), y_i \in \mathcal{Y} \subset Y(\Omega), i = 1, \dots, n\}$ . Aufgrund dieser Unvollkommenheit ist die Kenntnis von exaktem  $\mathbb{P}^{(X,Y)}$  und  $f$  unmöglich. Das Ziel ist nun einen *Algorithmus* (nach [16, S. 7])  $\hat{f}$  basierend auf gegebenen Trainingsdaten zu finden, also

**Definition 7. Algorithmus**

Sei  $(\Omega, \mathcal{A}, \mathbb{P})$  ein Wahrscheinlichkeitsraum,  $X : \Omega \rightarrow \mathcal{X}$ ,  $Y : \Omega \rightarrow \mathcal{Y}$  die Zufallsvariablen und  $T_n$  Trainingsdaten. Dann heißt eine messbare Abbildung  $\hat{f}_n : Q \times \mathcal{X} \rightarrow \mathcal{Y}$  **Algorithmus**, falls es eine messbare Abbildung  $A : (\mathcal{X} \times \mathcal{Y})^n \times \mathcal{X} \rightarrow \mathcal{Y}$  gibt, so dass für alle  $q \in Q$ ,  $x \in \mathcal{X}$  gilt

$$\hat{f}_n(q, x) = A(T_n(q), x). \quad (3.1)$$

Dabei ist  $q \in \Omega$  eine bestimmte Realisierung von  $\hat{f}_n$  zu gegebenen Daten  $T_n$ .

Das eingangs erwähnte Risiko  $R$  von  $f$  wird als Erwartungswert der Verlustfunktion  $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_+$  definiert, also:

**Definition 8.** Sei  $(X, Y)$  eine Stichprobe aus einer unbekannten Verteilung  $\mathbb{P}^{(X,Y)}$ . Das Risiko einer Entscheidungsregel ist dann gegeben durch:

$$R(f) = \mathbb{E}[\mathcal{L}(Y, f(X))] \quad (3.2)$$

Mit der gleichen Argumentation, dass die wahre Verteilung  $\mathbb{P}^{(X,Y)}$  unbekannt ist, kann (3.2) nicht direkt berechnet werden. So wird das Risiko durch das *empirische* Risiko approximiert

$$R_n(f) = \frac{1}{n} \sum_{i=1}^n L(y_i, \hat{f}_n(x_i)). \quad (3.3)$$

wobei  $(x_i, y_i) \in T_n$  die (beobachteten) Trainingsdaten sind.

**Bemerkung 4.** Es kann gezeigt werden [16, Bem. 1.17], dass aufgrund des Gesetzes der großen Zahlen gilt:  $R_n(f) \rightarrow R(f)$  für  $n \rightarrow \infty$ .

Mit obigen Annahmen ist es klar, dass Die Zielsetzung des statistischen Lernens darin besteht, den Algorithmus  $\hat{f}_n$  so zu bestimmen, dass das Risiko  $R$  durch die Verlustfunktion  $L$  minimiert wird, dass also gilt

$$\hat{f}_n = \arg \min_{f \in \mathcal{F}} R_n(f). \quad (3.4)$$

Wobei hier  $\mathcal{F} = \{f : \mathcal{X} \rightarrow \mathcal{Y}\}$  eine Menge von Funktionen mit beschränkter Komplexität ist. Es ist auch naheliegend zu sagen, dass  $\hat{f}_n$  ein Element von  $\mathcal{F}$  auswählt.

**Bemerkung 5.** Das oberste Ziel des statistischen Lernens ist, dass das Modell  $\hat{f}_n$  generalisierbar ist, d. h., es kann auf neue, unbekannte Daten angewendet werden.

Die Bemerkung 5 lässt anmuten, dass die Generalisierbarkeit von  $\hat{f}_n$  noch überprüft werden muss. Wenn das Verfahren (3.4) passiert ist, bedarf es noch sogenannter *Validierungsdaten*  $T_m \neq T_n$ <sup>8</sup>, um die Güte der Generalisierung festzustellen. Prinzipiell wird hier das empirische Risiko  $R_m$  auf den Validierungsdaten berechnet. In der Praxis des supervised ML werden die Beobachtungsdaten  $T_B$  in zwei Teile aufgeteilt, so dass  $T_B = T_n \cup T_m$  gilt. Die Phase in der (3.4) gelöst wird heißt *Trainingsphase*, und die wo  $R_m$  bestimmt wird heißt *Testphase*.

Sogenannter *Predictor*  $\hat{f}_n$  kann auf unterschiedliche Weisen gefunden werden. Es gibt lineare, sowie nicht lineare Ansätze. Ein klassisches Beispiel eines linearen Modells wäre eben *lineare Regression*. Eine nicht lineare Modellierung im Bezug auf statistische Lerntheorie stellen die *neuronale Netze* dar. Aus dem Kontext der vorliegenden Arbeit sollen diese im Folgenden genauer betrachtet werden.

### 3.1.1 Nichtlineare Modelle und neuronale Netze

Ein neuronales Netz lässt sich als eine Funktion auffassen, die durch eine Verknüpfung von linearen Transformationen und nichtlinearen Aktivierungsfunktionen konstruiert wird. Diese Funktion hängt von einer im Allgemeine sehr großen Menge von geeigneten zu wählenden Parametern ab. Ein mehrschichtiges NN (*Multilayer Perceptron MLP*) kann folgendermaßen definiert werden [16, S. 224]:

**Definition 9.** Ein **neuronales Netzwerk (NN)** mit der Netzwerkarchitektur  $(L, p)$ , wobei  $L \in \mathbb{N}_0$ ,  $p = (p_0, \dots, p_{L+1})^T \in \mathbb{N}^{L+2}$  und Aktivierungsfunktion  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  ist eine Funktion  $g : \mathbb{R}^{p_0} \rightarrow \mathbb{R}^{p_{L+1}}$  mit

$$g(x) = W^{(L+1)} \cdot \sigma(W^{(L)} \sigma(W^{(L-1)} \dots \sigma(W^{(1)}x + b^{(1)}) \dots + b^{(L-1)}) + b^{(L)}). \quad (3.5)$$

Dabei bezeichnen  $W^{(l)} \in \mathbb{R}^{l \times \mathbb{R}^l}$ ,  $l = 1, \dots, L+1$  die Gewichtsmatrizen und  $b^{(l)} \in \mathbb{R}^{p_l}$  die Verschiebungsvektoren. Die Netzwerkarchitektur bezieht sich auf  $L$ , die Anzahl der hidden layer und  $p_k$  die Breite des width Vektors<sup>9</sup>, mit  $k = 1, \dots, L$ .

Die schon erwähnte Eigenschaft eines NN's, ist die der universellen Approximation. Dazu existiert eine Reihe von Arbeiten die zeigen, dass die Netzwerkarchitektur und eine nicht lineare und nicht *polynomiale* Aktivierungsfunktion ein NN als universellen Approximator fungieren lassen. Beispielsweise wurde schon 1989 in [5] gezeigt, dass bereits ein einlagiges NN ein universeller Approximator ist. Diese Untersuchungen erstrecken sich bis in die heutige Zeit, wobei ein Trend zu vermerken ist. Die Untersuchungen an der Breite mit fester Tiefe hielten etwa bis in die 2000-er Jahre an. Danach kamen vermehrt die Untersuchungen der tiefen Netzwerke auf, wie etwa [12] oder [3] es zeigen.

---

<sup>8</sup>Streng formal wäre es besser zu sagen, dass  $T_m$  und  $T_n$  unabhängige Stichproben aus  $\mathbb{P}$  sind.

<sup>9</sup>Auch Breite des Layers genannt.

Für die Übersichtlichkeit soll hier der Satz der universellen Approximation aus [5] wiedergegeben werden und sein Bezug auf die Approximation einer stetigen Funktion hergestellt werden.

**Satz 1.** *Sei  $I \subset \mathbb{R}$  kompakt,  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  ist eine stetige Aktivierungsfunktion, dann liegt die endliche Summe*

$$G(x) = \sum_{j=0}^N \alpha_j \sigma(w_j x + b_j) \quad (3.6)$$

dicht in  $C(I)$ , also gilt

$$\forall f \in C(I) : \exists \varepsilon > 0 : \forall x \in I : |G(x) - f(x)| < \varepsilon. \quad (3.7)$$

Wobei die Gleichung (3.6) ein einlagiges NN mit  $N$  Neuronen abbildet. In der referenzierten Arbeit konnte gezeigt werden, dass für eine sigmoide Aktivierungsfunktion  $\sigma$  ein einlagiges NN eine beliebige Funktion  $f \in C(I)$  im Sinne einer Supremumsnorm approximieren kann (Gl. 3.7). Prinzipiell ist es eine Superposition nicht linearer Funktionen und strukturell ähnelt es einer Reihendarstellung von Funktionen mit einer Taylor<sup>10</sup>- oder Fourier<sup>11</sup>-Reihe, die auch unter Anderen zu den universellen Approximatoren gezählt werden können.

Der zweite wichtige Aspekt der mit den NNs mitgeliefert wird, ist der des Automatischen-Differenzierens (AutoDiff). Diese Fähigkeit spielt in Bezug auf PINNs eine wesentliche Rolle. Angenommen, dass ein NN die Lösung  $u(x, t)$  zum Problem 5 liefert, so erlaubt seine technische Umsetzung, beispielsweise in TensorFlow effizient die Ableitungen, solche wie  $u_x$  oder  $u_t$  zu berechnen. Diese wiederum können direkt in die DGL eingesetzt werden, um das Ergebnis zu überprüfen. Im Folgendem soll es kurz skizziert werden, wie dieser Zusammenhang steht. Es bleibt nur noch hervorzuheben, dass Neuronale Netze selbst kein AutoDiff enthalten, schließlich sind es nur Funktionen.

Neuronale Netze werden bekannterweise mit Gradientenabstiegsverfahren optimiert [16, Kap. 7]. Dabei sieht der  $t + 1$  Iterationsschritt der Gewichtsaktualisierung wie folgt aus

$$W_{[t+1]}^{(l)} = W_{[t]}^{(l)} - \lambda \nabla_{W_{[t]}^{(l)}} \mathcal{L}. \quad (3.8)$$

Der Parameter  $\lambda$  wird meistens als Lernrate bezeichnet. Anschaulich ist es die Schrittweite mit der der Gradient in Richtung eines Minimums schreitet. Im gesamten Trainingsprozess ist man bestrebt die Lernrate so anzupassen, dass das Verfahren im globalen Minimum endet.

---

<sup>10</sup>Brook Taylor, 1685-1731

<sup>11</sup>Joseph Fourier, 1768-1830

Die Verlustfunktion  $\mathcal{L}$  misst dabei den Fehler zwischen den Daten  $y$  und den Outputs des Netzwerks  $g(x)$  und kann beispielsweise als Mean-Squared-Error (MSE) definiert werden

$$\mathcal{L}(W) = \frac{1}{n} \sum_{i=1}^n (y_i - g(x_i))^2 = \frac{1}{n} \|\tilde{Y} - g(\tilde{X})\|_2^2, \quad (3.9)$$

mit  $(x_i, y_i) \in T_n = \tilde{Y} \times \tilde{X}$  und  $i = 1, \dots, n$  als Trainingsdatensatz.

Da das neuronale Netz  $g(x)$  als Kompositum mehrerer Funktionen aufgefasst werden kann, also

$$g(x) = a^{(L+1)} \circ a^{(L)} \circ \dots \circ a^{(1)}(x), \text{ mit } a^{(l)}(z) = \sigma^{(l)}(W^{(l)}z + b^{(l)}), \quad (3.10)$$

kann der Gradient der Verlustfunktion  $\mathcal{L}$  bezüglich der Gewichte  $W^{(l)}$  und Ausnutzung der Kettenregel als

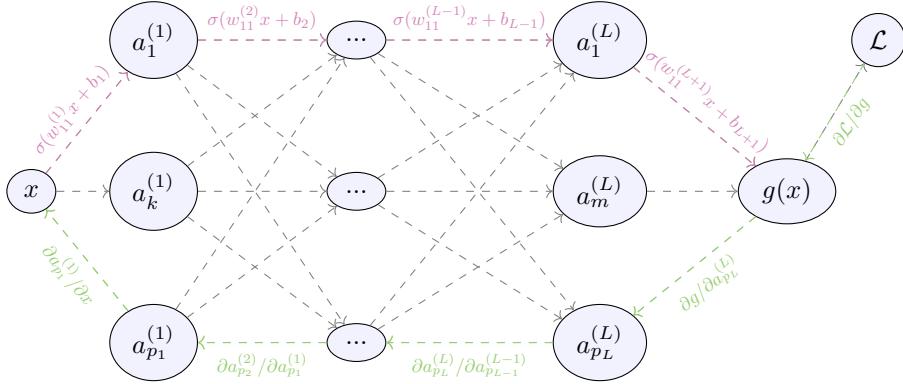
$$\nabla_{W^{(l)}} \mathcal{L} = \frac{\partial g}{\partial a^{(L+1)}} \cdot \frac{\partial a^{(L+1)}}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial a^{(L-1)}} \cdot \dots \cdot \frac{\partial a^{(l)}}{\partial W^{(l)}} \quad (3.11)$$

geschrieben werden. NNs lassen sich als gerichtete azyklische Graphen (DAG) effizient realisieren. In der Abbildung 3.1 ist DAG Struktur anschaulich verdeutlicht. Hier kann beispielsweise der Abwärtslauf, also der grüne Pfad, als Ableitung von  $g(x)$  nach  $x$  Verstanden werden, somit gilt

$$\frac{\partial g(x)}{\partial x} = \frac{\partial g}{\partial a^{(L+1)}} \cdot \frac{\partial a^{(L+1)}}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial a^{(L-1)}} \cdot \dots \cdot \frac{\partial a^{(1)}}{\partial x}.$$

In der Praxis bedeutet dies, dass während des Vorwärtsdurchlaufs des Netzwerks die Ausgaben jedes Knotens berechnet und gespeichert werden. Anschließend werden im Rückwärtsdurchlauf (Backpropagation) die Gradienten berechnet, indem die gespeicherten Werte und die Kettenregel verwendet werden. Dies ermöglicht eine effiziente und genaue Berechnung der Gradienten. Moderne Deep-Learning-Frameworks (hier wird TensorFlow verwendet, andere sind z.B. PyTorch oder JAX) implementieren diese Konzepte, indem sie die Rechenoperationen als DAGs modellieren und AutoDiff-Mechanismen bereitstellen, die auf dieser Struktur basieren.

Es kann also gesagt werden, dass AutoDiff ein schöner Nebeneffekt der Implementierung der Optimierungsverfahren für NNs ist. Eben dieser Effekt, den Ausgang des NNs bezüglich des Eingangs ableiten zu können ist grundlegend für den Einsatz der NNs bei der Lösung der Differentialgleichungen als PINN-Modelle.



**Abbildung 3.1.:** Schematische Darstellung eines NNs als gerichteter azyklischer Graph (DAG: Directed Acyclic Graph). Der rosa Pfad soll den Vorrwärtsdurchlauf (forward propagation) und der grüne den Abwärtsdurchlauf (back propagation) darstellen.

Auch höhere Ableitungen werden in einem *persistente*n Graph möglich. Konkret ab der zweiten *Iteration* im Trainingsprozess kann der Gradientenabstieg des ersten Durchlaufs gemacht werden, sprich

$$\frac{\partial}{\partial x} \left( \frac{\partial g(x)}{\partial x} \right) = \frac{\partial^2 g(x)}{\partial x^2}.$$

**Bemerkung 6.** Der Begriff der **Iteration** kann hier verwirrend wirken, da man im Kontext des maschinellen Lernens eher den Begriff **Epoche** benutzt. Bedeuten soll es aber das Gleiche, nämlich die Anzahl der Schritte, bei denen die Gewichte des NN aktualisiert werden. Und zumindest im Kontext der PINNs das hier eingesetzte Framework DeepXDE benutzt die Bezeichnung *Iteration* anstatt *Epoche*.

Um neuronale Netze als PINNs verwenden zu können, müssen physikalische Gesetzmäßigkeiten sowie Rand- und Anfangsbedingungen explizit in die Verlustfunktion  $\mathcal{L}$  eingebunden werden. Diese formuliert nicht nur die Diskrepanz zwischen Modellvorhersage und beobachteten Daten, sondern auch die Abweichung vom physikalischen Modell in Form von Residuen der Differentialgleichungen. Die Gewichtsanpassung des Netzes erfolgt dann, wie eben beschrieben, über Gradientenverfahren anhand von  $\mathcal{L}$ .

Die Art der verwendeten Verlustfunktion legt dabei fest, ob es sich um ein *supervised*, *unsupervised* oder ein *hybrides* Lernverfahren handelt. In klassischen *supervised learning*-Szenarien besteht  $\mathcal{L}$  aus einem Fehlermaß zwischen der Modellvorhersage und bekannten Ausgabewerten. Im Gegensatz dazu verwendet *unsupervised learning* keine expliziten Zielgrößen, sondern orientiert sich an internen Strukturen oder Zusatzannahmen (z. B. Clustern, Verteilungen oder physikalischen Gleichungen).

Die PINNs lassen sich nicht eindeutig supervised oder unsupervised ML unterordnen, sie kombinieren beide Ansätze. Einerseits wird die Vorhersage auf beobachtete Datenpunkte kalibriert (supervised), andererseits durch die Minimierung von Residuen der Differentialgleichungen auf physikalische Konsistenz hin optimiert (unsupervised). Das hängt stark vom dem Problem-Setting ab. Liegt also nur das Gleichungssystem mit Rand- und Anfangswerten vor, wie es in der Definition 5 der Fall ist, dann hat man einen unsupervised ML Fall vor sich. Liegen aber zusätzlich Messwerte in Form von Trainingsdaten  $(x_i, y_i) \in T_n$  vor, dann ist es weder supervised noch unsupervised ML. Das nachfolgende Beispiel für direkte Lösung des Problems 5 wird beispielsweise ohne der Menge  $\{y_i\}$  gelöst und stellt somit den unsupervised ML Fall dar.

### 3.1.2 Beispiel: PINNs für direkte Lösung der Wärmeleitungsgleichung

Die Bildungsvorschrift der Verlustfunktion  $\mathcal{L}$  ist also wesentlich dafür, welche Zusammenhänge ein NN lernen soll. Zum Aufstellen von  $\mathcal{L}$  soll hier zunächst ein Beispiel eines stationären Problems der Wärmeleitung auf dem Intervall  $\Omega = [0, 1]$  betrachtet werden. So bekommt die DGL aus der Definition 5 folgende Gestalt

$$\begin{aligned} -\frac{\partial}{\partial x} \left( a(x) \frac{\partial}{\partial x} u(x) \right) &= f(x), \\ u(0) = u(1) &= 0. \end{aligned} \tag{3.12}$$

Um die exakte Lösung dieses Problems für eine Validierung des eingelernten PINNs zur Verfügung zu haben, wurden eine Wärmeleitfähigkeit  $a(x)$  und die exakte Lösung  $u(x)$  vorgegeben und dann die rechte Seite  $f(x)$  gemäß der obigen Gleichung 3.12 berechnet. Die genannten Funktionen sind im Anhang B unter (B.2) und (B.3) angegeben.

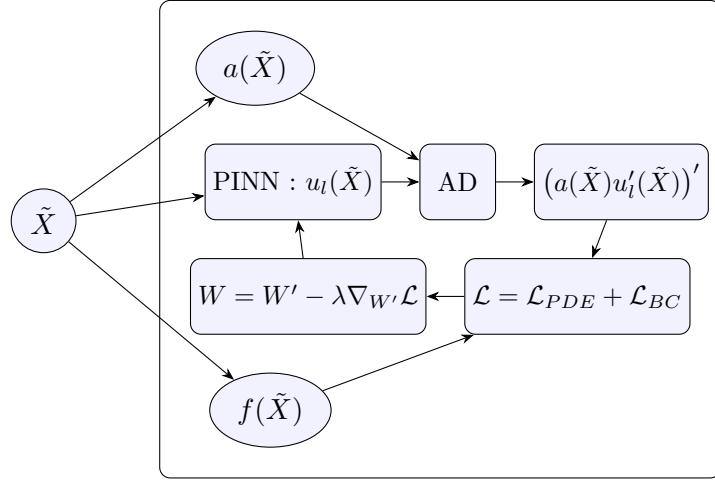
Ausgehend von (3.12) kann nun die Verlustfunktion für die PINNs, wie folgt, aufgestellt werden:

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{BC}} \\ &= \frac{1}{n} \| - (a(\tilde{X}) u'_l(\tilde{X}))' - f(\tilde{X}) \|_2^2 + \frac{1}{p} \| u_B - u_{LB} \|_2^2, \end{aligned} \tag{3.13}$$

mit  $\tilde{X} = \{x_i | i = 1, \dots, n\}$  als Menge der Trainingspunkte. Der zweite Term  $\mathcal{L}_{\text{BC}}$  bezieht sich nur auf die Randwerte der Lösung  $u(x)$ , also für  $x = 0$  und  $x = 1$ , damit ist  $p = 2$ .

Mit der Gleichung (3.13) liegt nun ein Minimierungsproblem vor, welches mit dem Gradientenabstiegsverfahren auf PINNs optimiert werden kann. Den Plan wie das Verfahren ablaufen soll gibt die Abbildung 3.2 wieder. Die Vorgehensweise ist relativ einfach, es wird eine gewisse Anzahl an Iterationen, Trainingsdaten und die Funktionen  $a$  und  $f$  vorgegeben. Dann wird in jedem Iterationsschritt das Schema aus der genannten Abbildung ausgeführt. Am Ende wird geschaut, ob das Ergebnis zufriedenstellend aussieht.

Damit ist hier die  $L^2$  Abstandsnorm des Ergebnisses  $u_l$  zur exakten  $u$  Funktion gemeint.



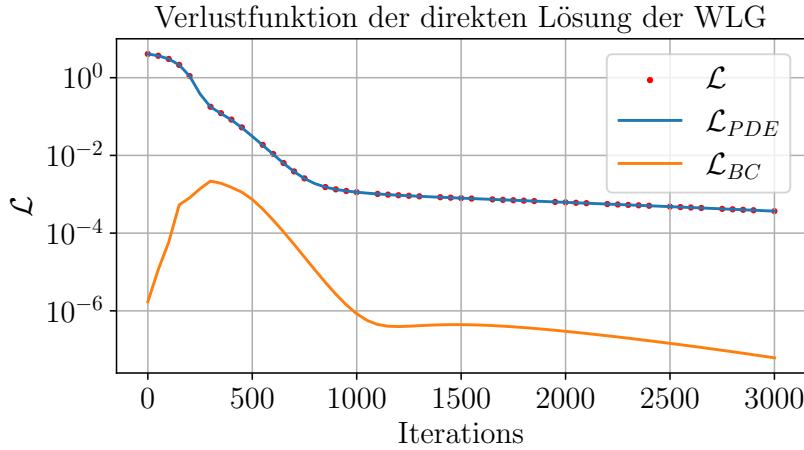
**Abbildung 3.2.:** Ablaufdiagramm des Trainingsloops eines PINNs zur direkten Lösung der stationären Wärmeleitungsgleichung. Das Diagramm stellt eine Iteration dar, die folgendermaßen abläuft: Zuerst werden an einer Menge  $\tilde{X}$  von Trainingspunkten die Funktionen  $a$  und  $f$  sowie die durch das PINN approximierte Lösung  $u_l$  ausgewertet, im Anschluss wird mittels AutoDiff  $(au'_l)'$  berechnet, danach wird der Ausdruck samt  $f$  auf der Trainingsmenge  $\tilde{X}$  ausgewertet und (3.13) berechnet, im Anschluss erfolgt die Backpropagation mit Gewichtsaktualisierung.

Mit dem im Kapitel A.1 vorgestellten Python Framework für Scientific Deeplearning *DeepXDE* lässt sich das in der Abbildung 3.2 dargestelltes Schema leicht umsetzen. Mit einem PINN von der Größe  $p = (1, 5, 5, 1)^T$  und der Aktivierungsfunktion

$$\tanh(x) := \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3.14)$$

lässt sich die DGL (3.12) in 3000 Iterationen bis auf einen Fehler  $\|u(x) - u_l(x)\|_{L^2} \approx 3.79 \cdot 10^{-5}$  in einer Zeit von  $7.97s$  lösen. Dabei wurde für die Optimierung der weit verbreitete Adam-Optimizer und eine Lernrate von  $\lambda = 1^{-3}$  benutzt, wobei der Optimierer die Lernrate im weiterem Verlauf des Trainings adaptiv anpasst.

Die Abbildungen 3.3 zeigt die Verlustfunktion der direkten Lösung der WLG und 3.4 die zugehörige Lösung an sich. In 3.3 erkennt man, dass die Wirkung von  $\mathcal{L}_{BC}$  auf die gesamte Verlustfunktion  $\mathcal{L}$  sehr gering ist. Und in der Abbildung 3.4 sieht man, dass die Lösung des Problems identisch zu der aus dem Beispiel 2.2.2 ist. Das liefert einen praktischen Nachweis, dass beide Verfahren äquivalent sind. Auf die Darstellung der Inputfunktionen  $a(x)$  und  $f(x)$  für  $\mathcal{L}_{PDE}$  wurde verzichtet, denn es sind dieselben wie in der Abbildung 2.2.

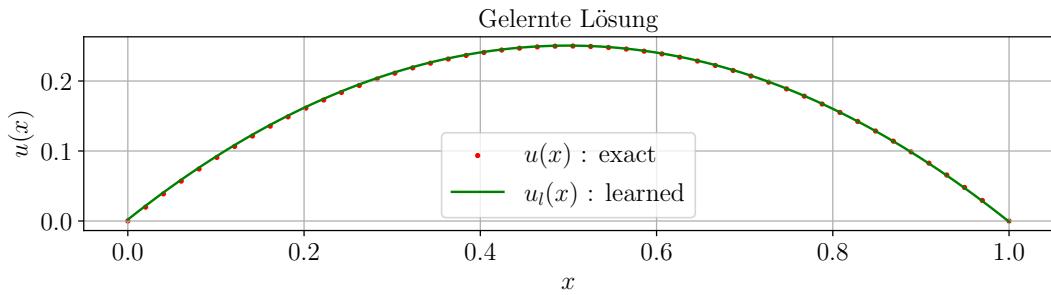


**Abbildung 3.3.:** Verlauf der Verlustfunktion des PINN für die direkte Lösung der stationären eindimensionalen WLG aus (3.12). Die Rechenzeit für 3000 Iterationen betrug etwa 7.97s. Aufgrund geringes Unterschieds der Gesamtverlustfunktion  $\mathcal{L}$  und  $\mathcal{L}_{PDE}$  wurde  $\mathcal{L}$  an nur 50 Punkten dargestellt.

Die somit erzielten Ergebnisse können denen aus dem Abschnitt 2.2.2 gegenüber gestellt werden. Hier wurde der Trainingsprozess für 1100 Punkte über  $\Omega$  in 3000 Iterationen durchgeführt. Im Beispiel aus 2.2.2 wurde das Intervall  $\Omega$  für 1100 Punkte diskretisiert. Die Rechenzeiten sind vergleichbar, also hier bei etwa 7.97s die numerische Lösung verlangte etwa 5.15s, dagegen ist die Genauigkeit gegenüber der exakten Lösung hier bei  $3.79 \cdot 10^{-5}$  und die der numerischen ist bei  $1.6 \cdot 10^{14}$ . Damit ist das Vorgehen mit PINNs umstritten langsamer und ungenauer, aber im Vergleich zum numerischen Verfahren musste hier keine Linearisierung durchgeführt werden und man braucht den Formalismus der schwachen Formulierung auch nicht mehr.

Abschließend, darf nicht unerwähnt bleiben, dass die mit dem PINN gefundene Lösung  $u_l(x)$  eine kontinuierliche Funktion ist. Das heißt, sie kann an jedem Punkt auf dem Intervall  $\Omega = [0, 1]$  ausgewertet werden. Im Vergleich liefern die numerische Lösungsverfahren wie FEM auch eine kontinuierliche Funktion, aber für das Verfahren selbst muss immer ein Gitter erzeugt werden und das kann recht aufwendig sein.

Im Bezug auf die Art des maschinellen Lernens hat hier ein unsupervised Learning stattgefunden. Das heißt soviel, dass die Verlustfunktion  $\mathcal{L}$  keine Informationen über die Messdaten  $\tilde{Y}$  erhalten hat, sondern nur Informationen über die Struktur des Problems (3.12) bekommen hat. Man spricht manchmal auch von *physics-constrained learning* oder *residual-based learning* statt klassisch unsupervised.



**Abbildung 3.4.:** Die Lösung der stationären eindimensionalen WLG (3.12). Der  $L^2$ -Abstand der exakten und der gelernten Lösung beträgt  $\|u(x) - u_l(x)\|_{L^2} \approx 3.79 \cdot 10^{-5}$ .

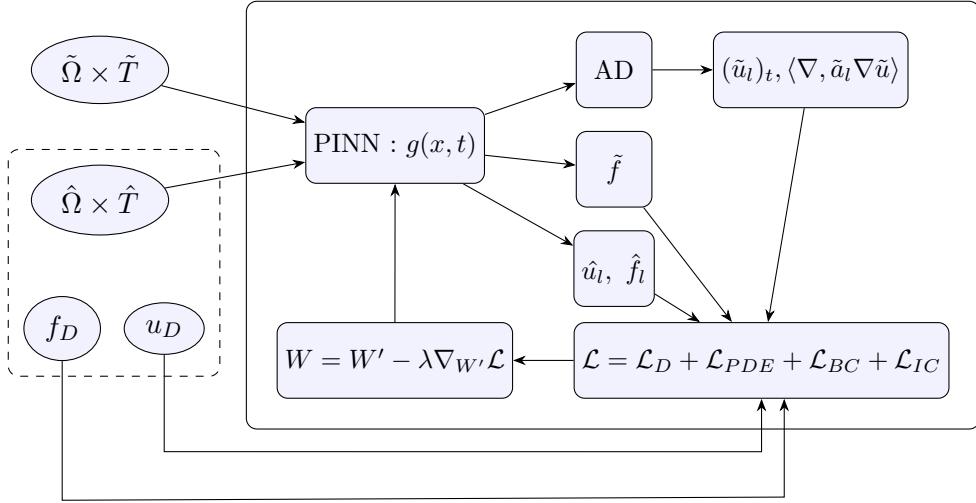
Die Implementierung zu diesem Beispiel findet man in der Datei `dde_forward_1D_ti.py` wieder.

## 3.2 PINNs für das inverse Wärmeleitungsproblem

Im Abschnitt 2.3 wurde angedeutet, dass es herausfordernd sein kann, den Operator  $\langle \nabla, a \nabla u \rangle$  zu invertieren, um aus einer bekannten Lösung  $u$  die Wärmeleitfähigkeit  $a$  zu bestimmen. PINNs kommen ohne eine solche Invertierung aus, weil hier der Ausdruck der Verlustfunktion minimiert wird und daher keine direkte Inversion des Operators stattfinden muss. Das ist die Hauptmotivation dafür, das inverse Wärmeleitungsproblem mit PINNs zu lösen.

Das einleitende Beispiel legt es nahe, dass die Struktur des Problems dem NN durch die Verlustfunktion vermittelt wird. Das heißt, die Gewichte des NNs werden je nach Gestalt der Verlustfunktion angepasst. Möchte man dass, PINN das Problem der Wärmeleitung lernt, soll die zugehörige PDE als *Residuum* in der Verlustfunktion ergänzt werden. Mit den Randbedingungen passiert das Gleiche. Zusätzlich hat man bei der Bestimmung des inversen Problems für die Wärmeleitfähigkeit die Messwerte der Temperaturverteilung und eventuell noch von der Wärmequelle, also der rechten Seite der PDE. Das PINN selbst soll jetzt die Raumkoordinaten und Zeit als Argumente bekommen und die Wärmeleitfähigkeit  $a(x)$ , die Lösung  $u(x, t)$  und zusätzlich, falls die Daten vorhanden sind, die Wärmequelle  $f(x, t)$  ausgeben. Es ist davon auszugehen, dass die Trainingsdaten  $\{\tilde{\Omega} \times \tilde{T}\}$  ( $\tilde{\Omega}$  soll auch die Ränder enthalten) und Messdaten  $(\hat{x}_i, \hat{t}_j) \in \{\tilde{\Omega} \times \tilde{T}\}$  mit Messwerten  $(u_D, f_D) \in \{u(\hat{x}_i, \hat{t}_j), f(\hat{x}_i, \hat{t}_j)\}$  schon vorliegen. Wobei, wie eben angedeutet, die Messwerte  $f_D$  nicht vorliegen müssen. Die graphische Darstellung 3.5 zeigt eine mögliche Umsetzung dieses Trainings-Procedere. Für die Klarheit muss gesagt werden, dass beispielsweise solche Ausdrücke wie  $(\tilde{u}_l)_t$  oder  $\langle \nabla, \tilde{a}_l \nabla \tilde{u} \rangle$  den Bezug zu den Daten  $\tilde{\Omega} \times \tilde{T}$  meinen, ebenso  $\hat{u}_l$ ,  $\hat{f}_l$  bezieht sich auf die Auswertung an  $\tilde{\Omega} \times \tilde{T}$ .

Die zugehörige Verlustfunktion bekommt somit die Daten der Messungen, die interne



**Abbildung 3.5.:** Das Schema für den Trainingsloop für PINNs des inversen Wärmeleitungsproblems zur Bestimmung der Wärmeleitfähigkeit  $a(x)$  aus vorgegebenen Daten  $f_D$  und  $u_D$ . Der Loop startet mit der Auswertung von  $(\tilde{u}_l)_t$  des Diffusionsterms  $\langle \nabla, \tilde{a}_l \nabla \tilde{u} \rangle$  (beides nach AD) und von  $\tilde{f}$  an den Trainingsdaten  $\{\tilde{\Omega} \times \tilde{T}\}$ .  $\hat{u}_l, \hat{f}_l$  werden entsprechend an den Messdaten  $\{\hat{\Omega} \times \hat{T}\}$  ausgewertet. Danach passiert die übliche Prozedur der Backpropagation und Gewichtsanpassung.

Abhängigkeiten des Problems über die DGL damit auch die Gesetze der Physik und die Rand- und Anfangsbedingungen. Somit stellt dies den hybriden Fall des maschinellen Lernens dar. Insgesamt hat die Verlustfunktion des inversen Problems der Wärmeleitung folgende Form

$$\begin{aligned} \mathcal{L} &: \Omega \times T \rightarrow \mathbb{R} \\ \mathcal{L} &= \mathcal{L}_D + \mathcal{L}_{PDE} + \mathcal{L}_{BC} + \mathcal{L}_{IC} \end{aligned}, \quad \text{mit } \Omega \subset \mathbb{R}^n, n = 1, 2; T \subset \mathbb{R} \quad (3.15)$$

Die obere Darstellung (3.15) ist noch zu allgemein und gibt keine Auskunft darüber, wie die einzelnen Terme der Verlustfunktionen aussehen. Daher ist es wesentlich zu klären, welche Daten von welchen Termen verarbeitet werden.

- **Datenanpassung**

Die Datenanpassung findet auf den gemessenen Werten statt, das heißt an  $\{\hat{\Omega} \times \hat{T}\} \subset \Omega \times T$ . Nimmt man an, dass die Messwerte der Temperaturverteilung  $\hat{u}$  und der Wärmequelle  $\hat{f}$  an selben Stellen aufgenommen wurden, so haben sie als Vektoren gleiche Länge. Für  $m$  Messwerte gilt dann:

$$\mathcal{L}_D = \frac{1}{m} \|\hat{u}_l - \hat{u}\|_2^2 + \frac{1}{m} \|\hat{f}_l - \hat{f}\|_2^2, \quad (3.16)$$

- **PDE-Residuum**

Das Sampling des PDE Residuums bei PINNs ist ein Thema für sich. Oft wird beim ML die Trainingsmenge in *Batches* (Stapel) unterteilt und der Gradient der Verlustfunktion an verschiedenen Batches pro Epoche (hier Iteration) ausgewertet, um die Generalisierungsfähigkeit des NN zu gewährleisten. Die jüngeren Untersuchungen an PINNs (z.B. in [18], [9], [20]) zeigten, dass die Wahl der Batch-Größe beim Training von PINNs signifikante Auswirkungen auf die Trainingsdynamik hat. Während kleinere Batches eine bessere Generalisierung ermöglichen können, führen sie zu längeren Trainingszeiten. Größere Batches bieten effizienteres Training, können jedoch die Generalisierungsfähigkeit beeinträchtigen. Die optimale Batch-Größe hängt daher von der spezifischen Problemstellung und den verfügbaren Ressourcen ab. Die hier verwendete Methode bezieht sich auf das benutzte deep learning Framework DeepXDE und benutzt die ganze Trainingsmenge als Batch. Dieser Ansatz hat zu Resultaten geführt, die zufriedenstellend sind und wurde nicht weiter hinterfragt.

Nun, geht man von der Trainingspunkte  $\{\tilde{\Omega} \times \tilde{T}\} \subset \Omega \times T$  mit der Kardinalität  $n$  aus, so ist die Auswertung des PINNs  $g$  auf dieser Menge gerade  $\{\tilde{u}_l, \tilde{a}_l, \tilde{f}_l\}$  mit jeweiliger Länge  $n$ . Dank des AutoDiffs heben  $\{(\tilde{u}_l)_t, \langle \nabla, \tilde{a} \nabla \tilde{u} \rangle\}$  auch die Länge  $n$ . Der MSE-Ausdruck für das PDE-Residuum hat entsprechend folgende Gestalt:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{n} \|(\tilde{u}_l)_t - \langle \nabla, \tilde{a} \nabla \tilde{u} \rangle - \tilde{f}_l\|_2^2 \quad (3.17)$$

- **Rand- und Anfangswerte**

Es wurde erwähnt, dass die Randwerte für das Training in  $\tilde{\Omega}$  enthalten sind. Formal sei festgehalten  $\{(\tilde{\Omega}_B \subset \tilde{\Omega}) \subset \tilde{\Omega} \times \tilde{T}\}$  ist eine beliebige Menge der Punkte am Rand mit  $p$  Elementen. Und es soll noch eine Menge der Anfangswerte  $\{\tilde{\Omega}_I \times 0\}$  mit  $k$  Elementen geben.

$$\mathcal{L}_{BC} + \mathcal{L}_{IC} = \frac{1}{p} \|\tilde{u}_B - \tilde{u}_{l_B}\|_2^2 + \frac{1}{k} \|\tilde{u}_I - \tilde{u}_{l_I}\|_2^2 \quad (3.18)$$

Als letztes kann nun die Kernaussage des gesamten Vorhabens aufgestellt werden: Das Training des PINN findet durch Minimierung der Verlustfunktion statt und in einem Idealfall werden die Gewichte  $W$  so angepasst, dass die Bedingung

$$W = \arg \min_W \mathcal{L}. \quad (3.19)$$

erfüllt wird. Wobei  $W \in \mathbb{R}^{(p_1 + \dots + p_L)}$  Vektor mit allen Gewichten  $w_j$  ist.

**Bemerkung 7.** *In der praktischen Realität wird die in Gleichung (3.19) formulierte Optimierungsaufgabe nur näherungsweise gelöst. Aufgrund der typischen Nichtkonvexität der Verlustfunktion  $\mathcal{L}$  ist es nicht garantiert, dass ein eindeutiges globales Minimum existiert. Vielmehr kann die Optimierung in einem lokalen Minimum oder Sattelpunkt enden, das stark von der Initialisierung der Gewichte abhängt.*

*Bei inversen Problemen ist die Lösungssensitivität gegenüber Rauschen oder Datenlücken oft ein zusätzlicher erschwerender Faktor, sodass der resultierende Gewichtvektor  $W$  möglicherweise nur eine von vielen annähernd gültigen Lösungen darstellt.*

Ungeachtet angedeuteter Schwierigkeiten lässt sich zusammenfassend sagen, dass PINNs eine geschickte Methode zur Lösung des inversen Wärmeleitungsproblems bereitstellen. Durch die Integration physikalischer Gesetze in die Verlustfunktion, gepaart mit gemessenen Daten, entsteht ein hybrider Ansatz, der sowohl datengetrieben als auch modellbasiert ist. Die Flexibilität dieses Vorgehens äußert sich durch Erweiterung oder schnelle Änderung der Verlustfunktion  $\mathcal{L}$ . Dadurch lassen sich komplexe, räumlich variable Wärmeleitfähigkeiten (Kap. 4) aus den Temperaturmessungen zuverlässig rekonstruieren. Allerdings bleiben Herausforderungen hinsichtlich der optimalen Wahl der Netzwerkarchitektur, der Regularisierung sowie der Auswahl effizienter Trainingsstrategien bestehen. Im Kapitel 4 werden neben der praktischen Umsetzung des Verfahrens auch die genannten Aspekte Platz für Diskussion finden.

### 3.2.1 Einfluss zusätzlicher Terme in der Verlustfunktion auf den Lösungsraum

Im Bezug auf die Lösungstheorie in diesem konkreten Fall des inversen Problems der Wärmeleitung liefert hier definierter PINN-Ansatz Lösungen aus dem Raum, wo das Problem direkt definiert wurde, nämlich  $u \in C^2$  und  $a \in C^1$ . Im Vergleich dazu sucht man bei einem klassischen Ansatz, wie FEM nach Lösungen mit einem äquivalent umformulierten Problem in einem anderem Funktionenraum, beispielsweise in einem Sobolev-Raum wie  $H_0^1(\Omega)$  oder  $H^2(\Omega)$ . Anschaulich bedeutet es, dass man den Lösungsraum erweitert, in dem man die Bedingungen des Problems abschwächt. Im Falle der PINNs ist dieser Umstand nicht nötig, weil NNs eine beliebige Funktion aus dem gegebenen Funktionenraum approximieren können, was auf ihre universellen Approximationseigenschaften zurückzuführen ist.

#### Regulierende Einschränkung durch Regularisierungsterme

Die mit PINNs gefundene Lösungen können zusätzlich reguliert werden, in dem man einen Regularisierungsterm der Form

$$\mathcal{L}_R = \lambda_R \int_{\Omega} |\nabla a(x)|^2 dx \quad (3.20)$$

(z.B. Tikhonov-Regularisierung erster Ordnung) in die Verlustfunktion hineinbringt, wobei  $\lambda_R > 0$  ein Gewichtsparameter des Terms ist. Solche Terme fördern spezifische strukturelle Eigenschaften wie Glattheit und wirken der Lösungsinstabilität in schlecht gestellten inversen Problemen entgegen. Die resultierende Lösung liegt dann typischerweise in einem regulierten Teilraum  $\mathcal{F}_\lambda \subset \mathcal{F}$ , der durch die Regularisierungsbedingungen bestimmt wird.

### Gradient-enhanced PINNs (gPINNs)

Ein weiterer Ansatz zu Regularisierung der PINNs wurde in [21] vorgeschlagen. Hier wurden *gPINNs* als eine Erweiterung der klassischen PINNs eingeführt. Die zentrale Idee besteht darin, nicht nur das PDE-Residuum in die Verlustfunktion einzubeziehen, sondern auch die Gradienten dieses Residuums. Dadurch wird dem neuronalen Netz mehr Information über die lokale Struktur der Lösung vermittelt.

$$\mathcal{L}_g = \lambda_g \frac{1}{d} \sum_{i=1}^d \left\| \frac{\partial}{\partial x_i} \mathcal{L}_{\text{PDE}}(x_i) \right\|^2, \quad (3.21)$$

$\lambda_g > 0$  spielt dieselbe Rolle des Gewichtsparameters wie oben. Diese Erweiterung soll zu einer stärkeren Regularisierung der Lösung führen.

## Kapitel 4

# Implementierung und Tests

Die Anfangsidee des Vorhaben vorliegender Arbeit bestand darin, ein zweidimensionales zeitabhängiges inverses Problem der Wärmeleitung zu lösen. Im Zuge der Entwicklungen hat es sich herausgestellt, dass man eigentlich ein ganzes Framework als Lösung bereitstellen kann. Diese Entstehung kam durch den üblichen Ansatz zu erst ein einfacheres Problem zu untersuchen. Somit lag zuerst ein eindimensionales stationäres inverses Problem der Wärmeleitung vor. Weitere Fälle kamen in Folge hinzu. Am Ende ist ein Framework entstanden, dass es den Nutzenden ermöglicht, verschiedene Varianten des inversen Problems flexibel zu formulieren. Beispielsweise kann man durch einfache Anpassungen der Eingabedaten ein eindimensionales Problem ohne Zeitabhängigkeit, genauso wie ein komplexeres zweidimensionales zeitabhängiges Problem berechnen lassen. Das Framework ist modular aufgebaut und kann leicht weiterentwickelt werden. Die genaueren Aspekte der Implementierung und deren Herausforderungen mögen im nächsten Abschnitt vorgestellt werden.

### 4.1 Architektur

Im Laufe der Entwicklungen hast es sich herausgestellt, dass das Schema aus der Abbildung 3.5 im Kontext eines flexiblen Frameworks nicht direkt umsetzbar. Grund dafür ist die spezielle Struktur des vorgenommenen Problems. Bei der Betrachtung des Schema 3.5 ist diese Schwierigkeit nicht offensichtlich, zumindest für den Verfasser nicht. Dies zu erkennen hat es seinen zeitlichen Anspruch genommen und ist hier einer Darstellung Wert.

#### Ein Fehlversuch

Den Ausgangspunkt für die praktische Umsetzung des *Lösers* des inversen Problems der Wärmeleitung bildet das Schema aus der Abbildung 3.5. Die Implementierung für den Fall des stationären eindimensionalen Problems setzte genau dieses Schema um. Aus der anwenderspezifischen Sicht, bestand die Hoffnung ein NN zu trainieren, welches die drei gesuchten Größen direkt liefert, also:

$$g(x) = (a(x), u(x), f(x))^T, \quad x \in \Omega \subset \mathbb{R}.$$

Mit den Trainingsdaten  $\tilde{\Omega} \subset \Omega$ , Messdaten  $\hat{x}_i \in \tilde{\Omega}$  und passenden Messwerten  $(\hat{u}, \hat{f}) \in \{u(\hat{x}_i), f(\hat{x}_i)\}, i = 0, \dots, m = |\tilde{\Omega}|$  nimmt die Verlustfunktion folgende Gestalt an

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_D + \mathcal{L}_{PDE} + \mathcal{L}_{BC} \\ &= \frac{1}{m} \left( \|\hat{u}_l - \hat{u}\|_2^2 + \|\hat{f}_l - \hat{f}\|_2^2 \right) + \frac{1}{n} \| - (\tilde{a}\tilde{u}')' - \tilde{f}_l \|_2^2 + \frac{1}{2} \|\tilde{u}_B - \tilde{u}_{l_B}\|_2^2.\end{aligned}$$

Die Tests für diesen Fall haben auch gute Resultate gezeigt, sprich die Abstandsnorm der gelernten Wärmeleitfähigkeit  $a_l$  zu der analytischen  $a$  ist immer kleiner geworden. Wohlgemerkt nur wenn, alle gesuchten Größen  $(a(x), u(x), f(x))$  ortsabhängig sind.

Sei nun jetzt der eindimensionale zeitabhängige Fall betrachtet. Im Hinblick auf das Schema 3.5 werden am PINN nur die Anzahl der Inputs angepasst

$$g(x, t) = (a(x), u(x, t), f(x, t))^T, \quad (x, t) \in \Omega \times T \subset \mathbb{R} \times \mathbb{R}_+.$$

Und die Verlustfunktion sieht entsprechend anders als im ersten Fall aus (ohne expliziten Aufzählung der Trainings- Messwert-, Anfangs- und Randwertmengen)

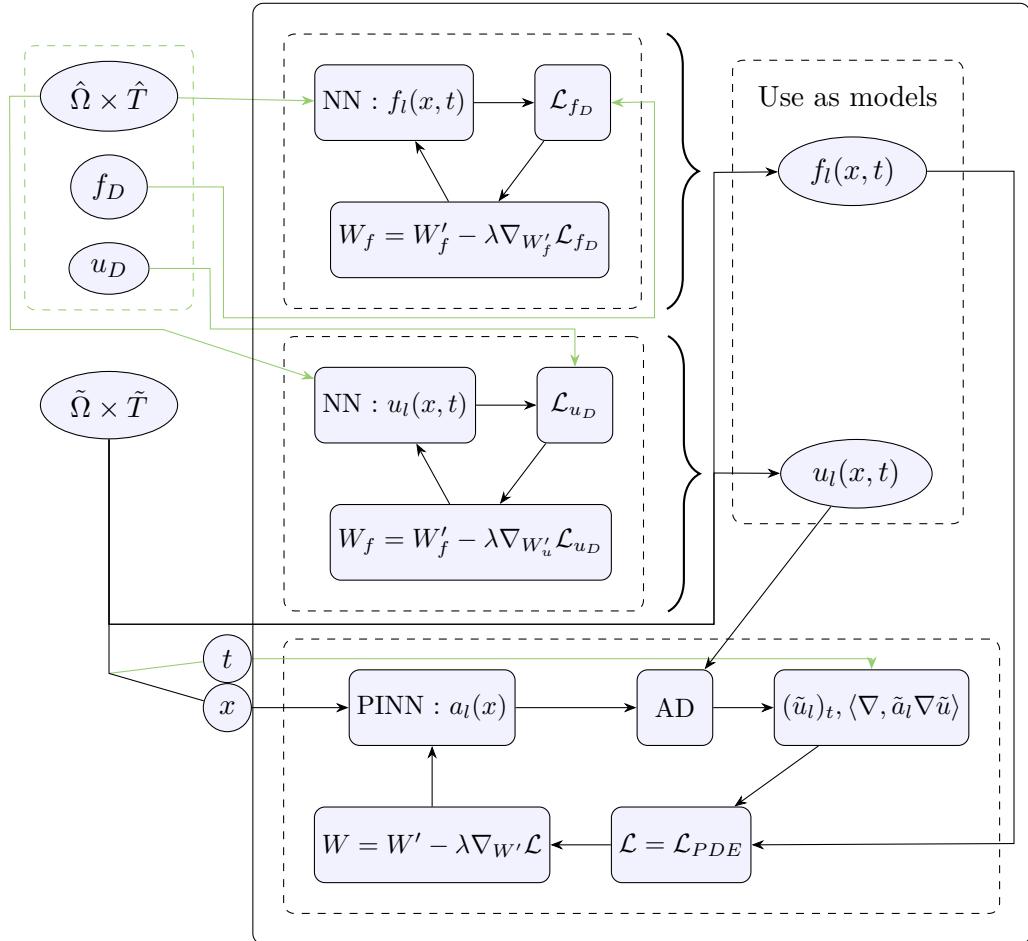
$$\begin{aligned}\mathcal{L} &= \mathcal{L}_D + \mathcal{L}_{PDE} + \mathcal{L}_{BC} + \mathcal{L}_{IC} \\ &= \frac{1}{m} \left( \|\hat{u}_l - \hat{u}\|_2^2 + \|\hat{f}_l - \hat{f}\|_2^2 \right) + \frac{1}{n} \|\hat{u}_t - (\tilde{a}\tilde{u}')' - \tilde{f}_l\|_2^2 \\ &\quad + \frac{1}{2} \|\tilde{u}_B - \tilde{u}_{l_B}\|_2^2 + \frac{1}{k} \|\tilde{u}_I - \tilde{u}_{l_I}\|_2^2.\end{aligned}$$

Die entsprechenden Tests haben gezeigt, dass trotz der Struktur der Verlustfunktion, lernte das PINN eine zeitabhängige Wärmeleitfähigkeit  $a(x, t)$ , was nicht zielführend war. Grund dafür ist, dass das PINN  $g(x, t)$  selbst raum- und zeitabhängig ist und diese intern weiter auf  $a$  propagiert. Also kann es in der Form keine zeitunabhängige Lösungen  $a(x)$  liefern.

Um dieses Problem zu beheben könnte man in die PINN Struktur auf der Implementierungsebene eingreifen und versuchen die Neuronen, die die zeitliche Abhängigkeit auf den Ausgang  $a$  projizieren, ausschalten. Dieser Ansatz wäre sehr aufwendig. Ein eleganterer ist, die Zeitabhängigkeit von  $a$  in der Verlustfunktion mit einem extra *Bestrafungsterm* etwa  $\mathcal{L}_a = \frac{1}{n} \|\frac{\partial a}{\partial t}\|_2^2$  zu versehen. Das würde im Allgemeinen jedoch zu *fast* keiner Zeitabhängigkeit führen, das ist durch die Gleichung (3.19) bedingt. Eine andere Lösung ist, dass man ein *multioutput* PINN mit Subnetzen definiert. Dieser Ansatz ist tatsächlich geläufig und wird beispielsweise in [11] diskutiert. Tatsächlich ist Letzteres zur Wahl gefallen, da es eine saubere Trennung zwischen zu lernenden Größen garantiert. Einen inspirierenden Beitrag hat hierzu die Arbeit von A. Tewes [1] geliefert. Jedoch wurde hier die WLG für einen homogenen instationären eindimensionalen Fall und einem Zwischenschritt mit Integration des PDE Residuums über das Intervall  $I \subset \mathbb{R}$ , gelöst.

### Neuer Ansatz

Die Überlegung das PINN in mehrere Subnetzwerke zu unterteilen würde den Vorteil bringen die Argumente einzelnen Netzwerken getrennt zu übergeben. Bezogen auf das konkrete Problem hier, würden drei Netzwerke entstehen. Die ersten zwei für  $u_l(x, t)$  und  $f_l(x, t)$  als Interpolationsnetzwerke. Und das dritte ist das PINN selbst für  $a(x)$ .



**Abbildung 4.1.:** Das separierte Modelltraining des PINNs des inversen Wärmeleitungsproblems für Wärmeleitfähigkeit. Die NNs für  $u_l$  und  $f_l$  werden jeweils in eigenem Trainingsloop an den Daten  $\hat{\Omega} \times \hat{T}$  und Messwerten  $f_D$ ,  $u_D$  angelernt. Danach findet das eigentliche Training des PINN auf der Trainingsmenge  $\tilde{\Omega} \times \tilde{T}$ , wo die vorgeholteten Modelle  $u_l$  und  $f_l$  zum Berechnen des PDE-Residuums eingesetzt werden. Wobei das PINN selbst nur Elemente von  $\tilde{\Omega}$  bekommt. Der grüne Pfeil von dem Knoten  $t$  stellt die Konsistenz der Trainingsdaten wieder her und liefert die passenden Zeitwerte für die Auswertung der Ableitungen von  $u_l$ .

Der neue Ansatz besteht somit aus drei Minimierungsaufgaben. Genauer bedeutet es, dass die Verlustfunktion 3.16 separiert werden muss. Ein Blick auf die Abbildung 4.1 erklärt diesen Umstand. In der linken oberen Ecke des Schemas sind die Messdaten  $\hat{\Omega} \times \hat{T}$  mit der Kardinalität  $|\hat{\Omega} \times \hat{T}| = m$  sowie  $f_D$  und  $u_D$  zu sehen. Diese fungieren als Trainingsdaten für die Interpolationsnetzwerke  $f_l$  und  $u_l$  ( $\hat{u}_l, \hat{f}_l$  ausgewertet an den Trainingspunkten  $\hat{\Omega} \times \hat{T}$ ). Beide werden unabhängig von einander jeweils in eigenem Loop trainiert. Also erhält jedes Interpolator-Netzwerk jeweils die Verlustfunktionen  $\mathcal{L}_{f_D}$  und  $\mathcal{L}_{u_D}$ . Sodass pro Netzwerk folgende Ausdrücke minimiert werden müssen:

$$\mathcal{L}_{u_D} = \frac{1}{m} \|\hat{u}_l - u_D\|_2^2 \quad (4.1)$$

$$\mathcal{L}_{f_D} = \frac{1}{m} \|\hat{f}_l - f_D\|_2^2 \quad (4.2)$$

Beide Funktionen sind aber gerade der Daten-Anteil  $\mathcal{L}_D$  der Gesamtverlustfunktion (3.15), daher muss dieser Anteil bei der Berechnung des Gesamtverlusts nicht mehr berücksichtigt werden. Geht man davon aus, dass die Messdaten auch die Rand- und Anfangswerte enthalten, so fallen auch die Terme  $\mathcal{L}_{BC}$  und  $\mathcal{L}_{IC}$  aus dem Gesamtverlust des PINNs weg.

Wenn das Anlernen von  $u_l$  und  $f_l$  stattgefunden hat beginnt das eigentliche Training des PINNs  $a_l$ . Erst hier kommt das AutoDiff-Konzept zum Tragen um die verbliebene Verlustfunktion (PDE-Residuum)

$$\mathcal{L}_{PDE} = \frac{1}{n} \|(\tilde{u}_l)_t - \langle \nabla, \tilde{a} \nabla \tilde{u} \rangle - \tilde{f}\|_2^2 \quad (4.3)$$

auszuwerten. Zunächst werden die Ausdrücke  $(\tilde{u}_l)_t$ ,  $\langle \nabla, \tilde{a} \nabla \tilde{u} \rangle$  im AD Block (Abb. 4.1) berechnet. Danach folgt die Auswertung an den Trainingspunkten  $\tilde{\Omega} \times \tilde{T}$  mit der Kardinalität  $|\tilde{\Omega} \times \tilde{T}| = n$  und anschließende Backpropagation-Procedere.

**Bemerkung 8.** *In Bezug auf PINNs wird tatsächlich empfohlen die Trainingspunkte  $\tilde{\Omega} \times \tilde{T}$  während des Trainings zu resampeln. Dafür bietet hier eingesetzte Software-Bibliothek DeepXDE entsprechende Funktionalität an. Konkret, werden die Trainingspunkte nach einer gewissen Anzahl von Iterationen wieder neu initialisiert.*

Interessant ist, dass bei diesem Ansatz am Ende als Lösung des Problems 5 drei kontinuierliche Funktionen  $u_l$ ,  $a_l$  und  $f_l$  vorliegen, die unabhängig von einander differenziert werden können. Ähnlich ist es im analytischen Fall, wo man theoretisch davon ausgeht, dass differenzierbare Funktionen vorliegen müssen um die DGL erfüllen zu können.

Aus der praktischen Sicht können die Gewichte einzelner NNs nach dem Training gespeichert und zu beliebigen Zeitpunkten wieder geladen und ausgewertet werden. Die Inhalte dieses Abschnitts bleiben jedoch recht abstrakt und liefern keine konkrete Informationen und Hinweise zur späteren Wiederverwendung der trainierten Modelle. Gerade diese

Aspekte sind entscheidend, wenn es um die Bequemlichkeit der Nutzung eines Frameworks geht. Um diesen Anforderungen gerecht zu werden, ist es unverzichtbar, über die theoretische Grundlage hinauszugehen und auch die praktische Umsetzung zu betrachten. Daher ist es wichtig, im weiteren Verlauf dieses Kapitels auch die konkrete Implementierung näher vorzustellen. Nur so kann nachvollzogen werden, welche Schnittstellen und Routinen zur Verfügung stehen und wie die einzelnen Komponenten ineinander greifen.

### 4.1.1 Implementierung

Die Implementierung wurde in der Programmiersprache Python aufgrund vorhandener Open-Source Bibliotheken durchgeführt. Allem voran diente die deep learning Software-Bibliothek DeepXDE (Anhang A) als Grundlage der Implementierung. Neben DeepXDE spielen die Frameworks TensorFlow und Keras eine wesentliche Rolle bei dem eigentlichen ML Prozess. Da bei der Benutzung des implementierten Frameworks nicht ersichtlich ist wie das Zusammenspiel zwischen diesen Bibliotheken stattfindet, ist es Grund genug, es zu in knapper Weise zu beschreiben.

#### Beziehung zwischen DeepXDE, TensorFlow und Keras

Ein wesentliches Merkmal von DeepXDE ist seine Fähigkeit, verschiedene Backend-Engines zur numerischen Berechnung und zum Training neuronaler Netze zu unterstützen. Zu den unterstützten Backends zählen unter anderem TensorFlow und PyTorch. In dieser Arbeit wurde TensorFlow als Backend gewählt.

**TensorFlow** ist eine von Google entwickelte, weit verbreitete Softwarebibliothek im dem ML Bereich. Sie bietet eine umfassende Infrastruktur zur Definition, Optimierung und effizienten Ausführung rechenintensiver Algorithmen auf verschiedenen Hardwareplattformen (CPUs, GPUs, TPUs). TensorFlow arbeitet dabei als *Backend*, d. h. als technische Grundlage für die Ausführung und das Training der zugrunde liegenden neuronalen Netze.

Um die Entwicklung und Implementierung von Deep-Learning-Modellen für den Anwender zu vereinfachen, wurde **Keras** als High-Level-API eingeführt. Keras stellt eine intuitive Schnittstelle zum Aufbau, Training und zur Evaluation neuronaler Netze zur Verfügung, indem es die zugrundeliegende Komplexität von TensorFlow kapselt. Seit TensorFlow 2.x ist Keras als `tf.keras` nahtlos in TensorFlow integriert und bildet die empfohlene Schnittstelle für Deep-Learning-Anwendungen auf Basis von TensorFlow. Praktisch bedeutet dies, dass sämtliche Modellierungs- und Trainingsschritte, die in DeepXDE definiert werden, letztlich über die Keras-API an TensorFlow übergeben und dort ausgeführt werden.

Das Zusammenspiel der genannten Pakete lässt sich wie folgt zusammenfassen: DeepXDE übernimmt die domänen spezifische Modellierung des Differentialgleichungsproblems

sowie die Formulierung der physikbasierten Verlustfunktion. Für den eigentlichen Aufbau und das Training des neuronalen Netzes nutzt DeepXDE die Funktionalität von Keras, welche die Modellerstellung, Optimierung und das Model-Management kapselt. TensorFlow dient wiederum als Backend, das die numerische Ausführung der Trainings- und Optimierungsprozesse übernimmt und für die effiziente Auswertung der Netzwerkarchitektur auf der Hardware sorgt.

Die Struktur dieser Softwarearchitektur ermöglicht eine klare Trennung zwischen domänen spezifischer Problemformulierung (DeepXDE) und generischem maschinellen Lernen (Keras/TensorFlow). Dies trägt wesentlich zur Modularität und Erweiterbarkeit der Implementierung bei.

### Das Framework: InverseHeatSolver

Das Framework ist objektorientiert aufgebaut. Konkret gibt es sechs Klassen, die miteinander interagieren. Die Abbildung 4.2 spiegelt diese Beziehung in einem UML Diagramm wieder. Mit einem kurzen Überblick über die Klassen soll gezeigt werden, welche Funktionalitäten die einzelnen Klassen in sich kapseln.

#### InverseHeatSolver

Diese Klasse stellt den *Entry point* des Frameworks dar. Sie kann anhand der Eingabe entscheiden welches Problem gerade vorliegt um dann entsprechende Initialisierungen vorzunehmen. Wesentlich sind die Parameter `two_dim` und `time_dependent`, weil davon die Initialisierung der NNs abhängt. Vorausgesetzt dass die Messdaten `obs_dom`, `u_obs` und `f_obs` vorliegen, kann eine Initialisierung des InverseHeatSolvers wie im Listing 4.1 vorgenommen werden.

```

1 domain = {'x_domain': [0, 1, 1000], 'y_domain': None, 't_domain': None}
2 nn_dims = {'num_layers': 3, 'num_neurons': 40}
3 obs_values = {'dom_obs': obs_dom, 'u_obs': u_obs, 'f_obs': f_obs}
4 loss_weights = {'w_PDE_loss': 1/5, 'a_grad_loss': 0}
5 learning_rate = 1e-2
6 use_deepxde = True
7 inv_solver = InverseHeatSolver(domain, nn_dims, obs_values, loss_weights,
                                learning_rate, use_deepxde)
8 inv_solver.train(a_iterations=5000, u_iterations=10000, f_iterations=15000,
                   display_results_every=500, save_path="models/1D_ti_dde")
```

**Listing 4.1:** Beispiel einer Initialisierung des InverseHeatSolvers.

Im kurzen Beispiel 4.1 in der Zeile 1 wird der Definitionsbereich `domain` des Problems definiert, hier eindimensional und zeitunabhängig. `nn_dims` (2) stellt die Dimensionalität das PINNs dar, also 3 Lagen mit je 20 Neuronen. `obs_values` (3) ist selbstsprechend. Im ML ist es üblich, dass man die Verlustterme gewichtet um die Beeinflussung einzelner Terme auf den Gesamtverlust vorzunehmen, hier geschieht es mit `loss_weights` (4).

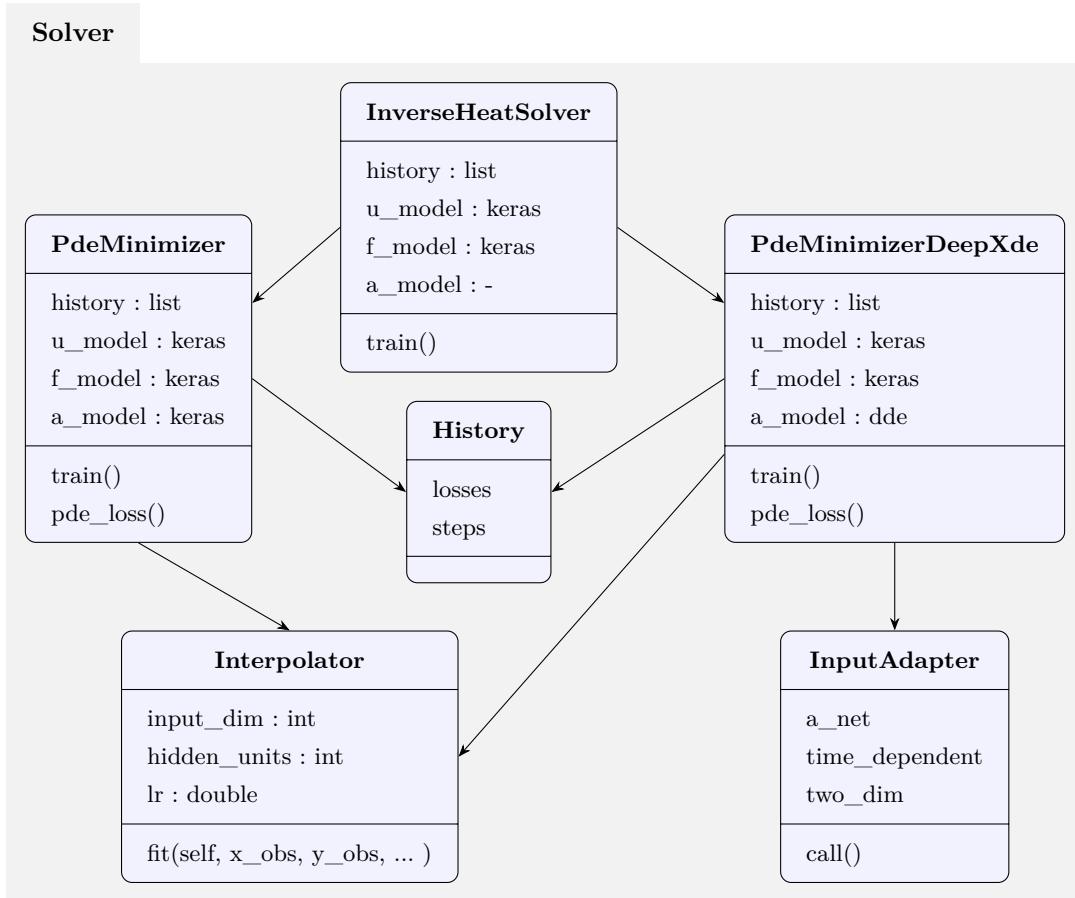


Abbildung 4.2.: Das UML-Diagramm des InverseHeatSolvers.

`learning_rate` (5) ist die Lernrate, oder die Schrittweite des Gradientenabstiegsverfahrens. `use_depxde` (6) definiert, ob DeepXDE als Berechnungs-Framework eingesetzt werden soll, oder ein selbst implementiertes (das wird später aufgeklärt). Danach folgt in der Zeile 7 schon die Initialisierung des Objekts selbst und anschließend in der Zeile 8 das Training.

### Interpolator

Wie der Name es schon verrät handelt es sich um Interpolation. Die Klasse basiert ganz auf `tf.keras` und stellt ein Interpolationsnetzwerk mit variabler Tiefe und zwei *hidden layers* bereit. Die Hauptaufgabe der Klasse ist die Funktionen  $u$  und  $f$  aus gemessenen Daten zu interpolieren. In dem InverseHeatSolver in der Methode `train(...)` wird das Training, wie in der Abbildung 4.1 gezeigt ist, für jede zu Lernende Größe einzeln durchgeführt.

```

1 self.u_model = Interpolator(self.dimension, lr=self.learning_rate)
2 u_loss, u_steps = self.u_model.fit(self.observed_domain, self.u_obs,
3         iterations=u_iterations, print_every=display_results_every)

```

**Listing 4.2:** Einsatz der Interpolator Klasse in dem InverseHeatSolver in der Methode `learn_u_and_f(...)`. Dabei bezieht sich `self` auf das instanzierte Objekt des InversHeatSolvers.

Ein der Trainings ist im Listing 4.2 gezeigt. Die Initialisierung in der Zeile (1) ist klar. Das Fitting (2) an die Daten benutzt im Wesentlichen die Messdaten `observed_domain`, `u_obs` und die Anzahl der Iterationen `u_iterations`. Die Methode `fit(...)` liefert ein NN das den kleinsten Wert der Verlustfunktion innerhalb des Trainingloops erzeugt hat.

Obwohl der Name „Interpolator“ vermuten lässt, dass lediglich eine exakte Anpassung an die gegebenen Datenpunkte erfolgt, handelt es sich mathematisch betrachtet um eine Regressionsaufgabe. Dies wird insbesondere dann relevant, wenn die gemessenen Daten verrauscht sind, wie es in praktischen Anwendungen häufig der Fall ist. Das Netzwerk approximiert in diesem Fall nicht jeden Messpunkt exakt, sondern findet eine glatte Funktion, die den Gesamtdatensatz im Sinne der gewählten Verlustfunktion bestmöglich beschreibt. Somit stellt das Interpolationsnetzwerk ein Werkzeug dar, um auch aus verrauschten Daten die zugrunde liegenden Funktionen  $u$  und  $f$  zuverlässig zu rekonstruieren (s. 4.2.2).

## PdeMinimizerDeepXde und InputAdapter

In dem Beschreibungsabschnitt des InverseHeatSolvers wurde ein Flag `use_depxde` erwähnt. Dieser Flag verweist auf die Benutzung der PdeMinimizerDeepXde wenn er `True` gesetzt ist, ansonsten wird eigene Implementierung `PdeMinimizer` benutzt.

Die Schwierigkeit die im Fehlversuch des Abschnitts 4.1 beschrieben wurde liefert das Framework DeepXDE mit. Das heißt, die Möglichkeit Multioutput-Networks zu benutzen ist gegeben, aber nicht die, die es erlaubt die einzelnen Inputs an bestimmte Netzwerke zu verabreichen. Somit ist das Problem 5 nicht direkt lösbar. Die Kapselung des DeepXDE-Workflows im PdeMinimizerDeepXde soll es zeigen.

### 1. PDE-Residuum $\mathcal{L}_{PDE}$ (`pde_loss`):

Die DeepXDE-API erwartet eine Callback-Methode, die dem PDE-Residuum entspricht

$$\text{pde\_loss}(\mathbf{x}, \mathbf{y}) = \mathcal{N}[u(x, t); a(x)] - f(x, t), \quad (x, t) \in \tilde{\Omega} \times \tilde{T}$$

wobei  $\mathcal{N}$  den PDE Operator,  $u$  die Lösung,  $a$  der unbekannte Parameter (Wärmeleitfähigkeit) und  $f$  die rechte Seite der PDE beschreiben.  $\mathbf{y}$  ist der Output des PINNs. In dieser Methode werden auch die interpolierten  $u$  und  $f$  an  $(x, t)$  ausgewertet.

### 2. PDE-Domain definieren (`prepare_domain`):

Hier wird festgelegt welche Dimensionalität (`pinn_domain`) das Problem hat

- Intervalle: `dde.geometry.Interval(x_s, x_e)`
- Rechtecke (2D): `dde.geometry.Rectangle([x_s, y_s], [x_e, y_e])`
- Zeitbereiche: `dde.geometry.TimeDomain(t_s, t_e)`
- Raumzeit: `dde.geometry.GeometryXTime(geom_domain, time_domain)`

### 3. Definition des Modells (`prepare_model`):

Diese Methode kapselt folgende Schritte:

- (i) Datensatz (`data`) kombiniert Domain, PDE sowie Messdaten:
  - Stationär: `dde.data.PDE(pinn_domain, pde_loss, ..., obs_domain)`
  - Zeitabhängig: `dde.data.TimePDE(pinn_domain, pde_loss, ..., obs_domain)`
- (ii) Neuronales Netz (`a_net`) definieren, etwa mittels:
  - `dde.maps.FNN([input_dim, layer1, ..., output_dim], ...)`
- (iii) PINN-Modell instanziieren:
  - `a_model = dde.Model(data, a_net)`

### 4. Kompilieren (Graph erstellen) und Training durchführen:

- `a_model.compile(optimizer, lr, loss_weights)`
- `a_model.train(iterations, callbacks)`

**Bemerkung 9.** Typischerweise sollten im Schritt 3 zusätzlich die Rand- und Anfangsbedingungen festgelegt werden, diese fallen hier weg, da die Funktionen  $u$  und  $f$  vorher schon interpoliert sind (solange Messungen am Rand vorhanden sind).

In diesem Setting wird während des Trainings die Callback-Methode `pde_loss` aufgerufen und an `pinn_domain` ausgewertet. Weil in `pde_loss` nach dem AD mitunter `a_net` an der ganzen Domain ausgewertet werden muss. Das heißt, ist die Dimensionalität von `pinn_domain` und die des Eingangs von `a_net` nicht dieselbe wird ein Fehler geworfen. Dieser Fall tritt auf, wenn `a_net` nicht zeitabhängig sein soll.

Sei nun `a_net` genauer betrachtet, das ist ein NN, das die Art und Weise des zugrundeliegenden Backends erbt. Im vorliegenden Fall ist es TensorFlow, das ist also ein Keras-Modell. Schreibt man nun einen Wrapper, der dem NN vorgibt, wie es sich bezüglich der Eingänge verhalten soll, hat man das Problem gelöst. Diese Aufgabe übernimmt der **InputAdapter**. Genauer gesagt, der Wrapper überschreibt im Wesentlichen die Call-Methode von `tf.keras.Model`.

```

1 class InputAdapter(tf.keras.Model):
2     def __init__(self, a_net, time_dependent=False, two_dim=False):
3         super().__init__()
4         self.time_dependent = time_dependent
5         self.two_dim = two_dim
6         self.regularizer = None
7         self.a_net = a_net
8

```

```

9     def call(self, inputs):
10        if not self.time_dependent:
11            a = self.a_net(inputs)
12        elif not self.two_dim and self.time_dependent:
13            x = inputs[:, :1]
14            a = self.a_net(x)
15        else:
16            xy = inputs[:, :2]
17            a = self.a_net(xy)
18        return a

```

**Listing 4.3:** Implementierung des InputAdapter zur Steuerung der Input-Dimensionalität des PINNs. Nur im Falle der zeitlichen Abhängigkeit muss unterschieden werden, welche Inputs das NN bekommt, Zeilen 12 und 15.

Der schritt 3.(iii) kann mit Benutzung des InputAdapters umgestaltet werden in

```

composite_net = InputAdapter(a_net, time_dependent, two_dim)
a_model = dde.Model(data, composite_net).

```

Das ist eine schlanke Implementierung mit der sich das Problem 5 im Kontext des Frameworks DeepXDE lösen lässt.

## PdeMinimizer

Diese Implementierung erfüllt prinzipiell das Gleiche wie PdeMinimizerDeepXde, jedoch wird hier direkt TensorFlow benutzt. Das heißt der Trainingsprozess wird hier in dem typischen ML Loop ausgeführt und die obere `call` Methode kann direkt in der Klasse implementiert werden. Diese Implementierung hat mehr einen didaktischen Wert und offenbart die Prozesse die unter der Haube des DeepXDE stattfinden.

**Bemerkung 10.** Beide Ausführungen `PdeMinimizer` und `PdeMinimizerDeepXde` implementieren auch das Konzept der Regularisierung aus dem Abschnitt 3.2.1. Zu beachten ist, dass in DeepXDE nicht die Möglichkeit besteht, die Regularisierungsterme als eigenständige Verlustfunktionsterme einzubinden. Deshalb enthält `PdeMinimizerDeepXde` in der Verlustfunktion des PDE-Residuums `pde_loss(...)` die Terme der Regularisierung, also `res + a_grad + gPINN`. Dieser Ansatz ist grundsätzlich nicht falsch, erlaubt aber nicht die Auseinanderhaltung einzelner Verlustfunktionen und somit die Beurteilung, welcher der Verluste wie viel Gewicht an dem Gesamtverlust hat.

Zusätzlich bietet Keras eine Regularisierung über die Gewichte des NNs an. Diese Funktionalität kann für  $L^1$  und/oder  $L^2$  aktiviert werden. Für alle benutzten NNs wurde `regularizers.L1L2()` genommen.

## History

History ist eine Datenstruktur die während des Trainingsprozesses gesampelt wird und dient zur Grafischen Darstellung der *Lernkurven* (oder Verlustfunktionen) einzelner NNs.

## 4.2 Testergebnisse

In diesem Abschnitt werden einige der durchgeführten Tests gezeigt. Als erstes sollen hier die vier behandelbaren Fälle, für die das Framework ausgelegt ist, betrachtet werden. Vor Allem geht es dabei, zu zeigen dass das Framework im Stande ist brauchbare Funktionen für  $u$ ,  $f$  und die Wärmeleitfähigkeit  $a$  zu liefern. Da die Messdaten für diese Beispiele aus einer Handrechnung stammen (Anhang B) konnte die Beurteilung der Outputs anhand der  $L^2$ -Norm durchgeführt werden.

Ein weiterer Test wird die Empfindlichkeit des Lösen auf verrauschte Daten untersuchen. Hier wird es sich zeigen, dass der Löser im Sinne des ML eine Regressionsaufgabe bezüglich der Daten von  $u$  und  $f$  löst.

Und der letzte Punkt der Tests nimmt sich der *Hyperparametrierung* an. Das heißt hier werden einigen Parameter des Lösen im Bezug auf die Genauigkeit und die Laufzeit untersucht.

### Eingesetzte Hardware

Es ist nicht unerheblich die eingesetzte Hardware zu erwähnen, da hier je nach Art noch Optimierungspotential bezüglich der Trainingszeiten besteht. Die verwendete CPU in allen Testfällen ist: Intel i7-7820HQ (2.90GHz) mit 4 physikalischen und 8 logischen Kernen. TensorFlow wurde folgendermaßen konfiguriert:

```
1 tf.config.threading.set_intra_op_parallelism_threads(4)
2 tf.config.threading.set_inter_op_parallelism_threads(2)
```

**Listing 4.4:** Konfiguration von TesnsorFlow für eine Parallelisierung.

Dabei ist obere Listing folgendermaßen zu deuten:

- `intra_op_parallelism_threads` (hier: 4) Diese Zahl legt fest, wie viele Threads zur parallelen Abarbeitung von Operationen verwendet werden, die intern parallelisiert werden können. Beispiele solcher Operationen sind Matrixmultiplikationen oder Reduktionen.
- `inter_op_parallelism_threads` (hier: 2) Diese Zahl bestimmt, wie viele Operationen parallel voneinander ausgeführt werden können, also Operationen, die unabhängig voneinander sind und sich nicht gegenseitig blockieren.

Diese Einstellung hat sich als relativ optimal herausgestellt und die Hardware nicht übersteuert.

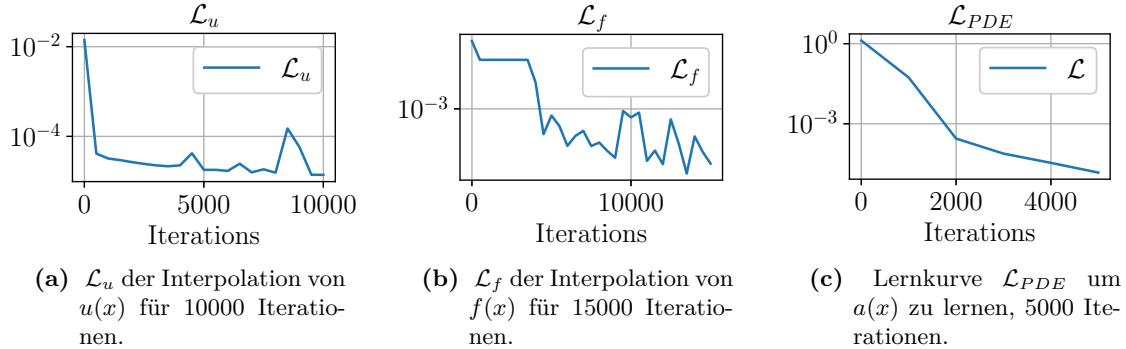
### 4.2.1 Die synthetischen Beispiele aus Anhang B

In diesem Abschnitt soll *Proof of Concept* vorgeführt werden. Das bedeutet, das Framework für alle betrachteten Fälle zu testen und die Güte der Ergebnisse mit der  $L_2$  Norm zu beurteilen. Für die Plausibilität der Ergebnisse wurden händisch gerechnete Beispiele

aus dem Anhang B benutzt. Für alle Fälle wurde hier der DeepXDE Wrapper PdeMinimizerDeepXde benutzt. Grundsätzlich haben dieselben Test auch mit der eigenen Implementierung PdeMinimizer funktioniert, der Unterschied bestand jedoch in deutlich längeren Laufzeiten, insbesondere für datenintensive zweidimensionale zeitabhängige Fälle, aber der PdeMinimizer hat sein Proof of Concept geliefert. Aus Zeitgründen wurden alle späteren Tests nur mit dem PdeMinimizerDeepXde durchgeführt.

### 1D zeitunabhängig

Der eindimensionale stationäre Fall ist durch die Gleichungen (B.1), (B.2) und (B.3) festgelegt. Für synthetische Messdaten wurden die Funktionen  $u(x)$  und  $f(x)$  an 25 Äquidistanten Punkten ausgewertet.

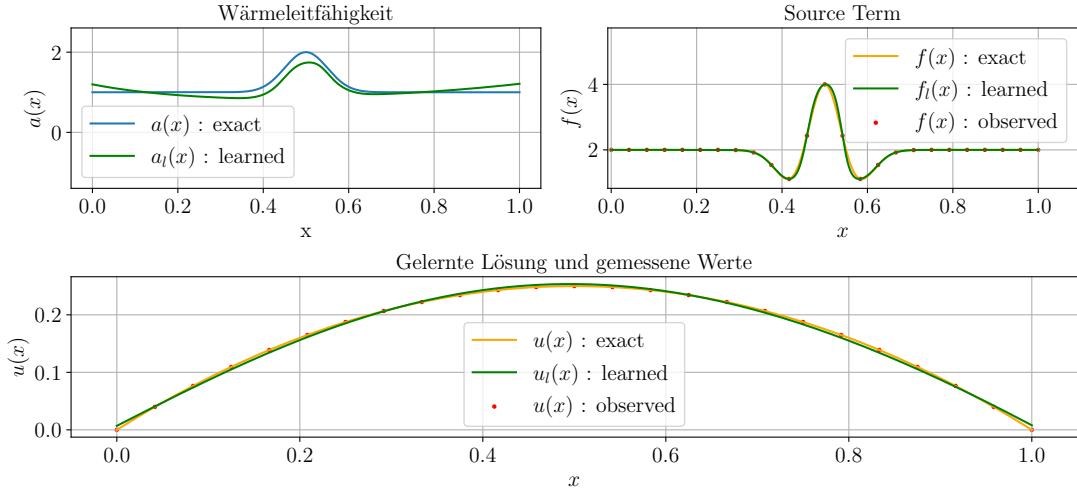


**Abbildung 4.3.:** Die Verlustfunktionen des eindimensionalen stationären Falls. Trainingszeit beträgt etwa 12.9 Minuten.

Die Abbildung 4.3 zeigt die Lernkurven für folgendes Setting:

- PDE-Domain:  $\tilde{\Omega} = [0, 1]$  mit 1000 äquidistanten Punkten,
- $NN_u$  und  $NN_f$  sind mit  $\{2, 20\}$  und PINN mit  $\{2, 10\}$  dimensioniert.
- Lernrate für alle NNs ist  $\lambda = 10^{-2}$ .

Die erzielten Ergebnisse zeigen die Abbildung 4.4. Die gemeinsame Trainingszeit betrug 12.9 Minuten. Dabei wurden folgende Normabstände auf einer Testmenge  $X_t$  mit 1000 äquidistanten Punkten erzielt:  $\|u(X_t) - u_l(X_t)\|_{L^2} \approx 1.036 \cdot 10^{-4}$ ,  $\|a(X_t) - a_l(X_t)\|_{L^2} \approx 4.1 \cdot 10^{-3}$  und  $\|f(X_t) - f_l(X_t)\|_{L^2} \approx 1.56 \cdot 10^{-3}$ . Diese Ergebnisse deuten auf eine Normkonvergenz hin und die optische Beurteilung des Ergebnisses lässt sagen, dass die gelernte Kurve  $a_l$  der gesuchten Wärmeleitfähigkeit  $a$  sehr stark nahe kommt.



**Abbildung 4.4.:** Die Lösung des eindimensionalen stationären Problems der inversen Wärmeleitung. Die gelernten Kurven  $u$  und  $f$  wurden an 25 äquidistanten Punkten  $\{\tilde{x}_i, \tilde{u}_i, \tilde{f}_i\}$  interpoliert. Das Ergebnis des PINN lieferete  $\|a(X_t) - a_l(X_t)\|_{L^2} \approx 4.1 \cdot 10^{-3}$ , ausgewertet an der Testmenge  $X_t$  mit 1000 äquidistanten Punkten.

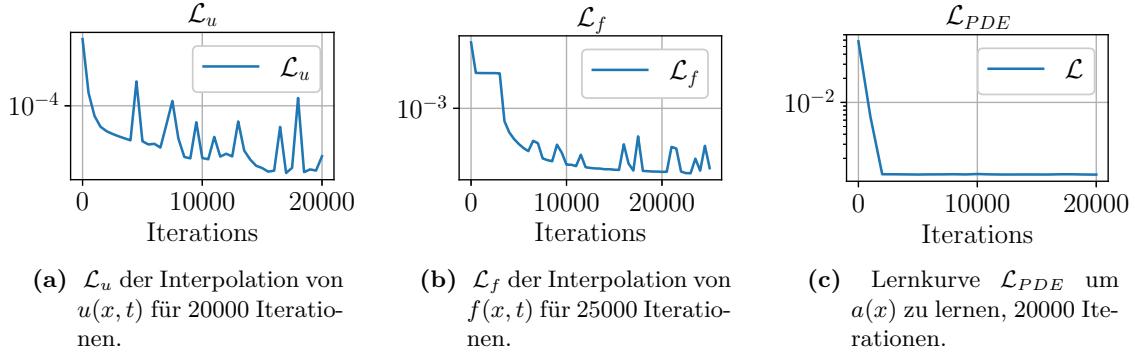
## 1D zeitabhängig

Referenzen hier sind die Funktionen B.4, B.5 und aufgrund der Ortsabhängigkeit der Wärmeleitfähigkeit wieder die Funktion B.2. Das sind gleich strukturierte Funktionen wie oben nur das  $u$  und  $f$  mit einer zeitlich abfallender Komponente  $e^{-t}$  komponiert sind. Für physikalische Systeme ist es typisch, dass eine Anregung exponentiell mit der Zeit abfällt.

Die in der Abbildung 4.5 dargestellten Verlustfunktionen wurden mit folgendem Setting erzielt:

- Messungen: 5500 Punkttripels  $\{\hat{u}, \hat{f}, (\hat{x}, \hat{t}_j)\}$  äquidistant über die Geometrie und Zeit verteilt,
- PDE-Domain:  $\tilde{\Omega} = [0, 1]$  mit 500 und  $\tilde{T} = [0, 6]$  mit 100 äquidistanten Punkten,
- $NN_u$  und  $NN_f$  sind mit  $\{2, 20\}$  und PINN mit  $\{3, 20\}$  dimensioniert.
- Lernrate für alle NNs ist  $\lambda = 10^{-2}$ .

In der Abbildung 4.6a sind die Ergebnisse für Interpolationen für  $u$  und  $f$  als räumliche Kurven über das Testintervall  $X_t$  (1000 Punkte) und Testzeit  $T_t$  (100 Punkte), jeweils äquidistant aufgelöst. Die Abbildung 4.6b zeigt  $u$  an  $X_t$  ausgewertet. Die zugehörigen Normabstände betragen:  $\|u_t - u_l\|_{L^2} \approx 1.18 \cdot 10^{-6}$ ,  $\|a(X_t) - a_l(X_t)\|_{L^2} \approx 6.47 \cdot 10^{-4}$  und  $\|f_t - f_{l_t}\|_{L^2} \approx 3.003 \cdot 10^{-3}$ . Dieser Fall gehört zu dem, wo die Herangehensweise



**Abbildung 4.5.:** Die Verlustfunktionen des eindimensionalen instationären Falls. Der Verlauf von  $\mathcal{L}_{PDE}$  bleibt ab ca 1800 Iteration bei einem relativ konstantem Wert von  $1.2 \cdot 10^{-3}$ , also konnte die Trainingsphase für  $a$  schon bei  $1/3$  der Iterationen abgebrochen werden. Die Gesamttrainingszeit lag bei etwa 26.41 Minuten.

mit dem InputAdapter ihren Beitrag zur Lösung leistet.

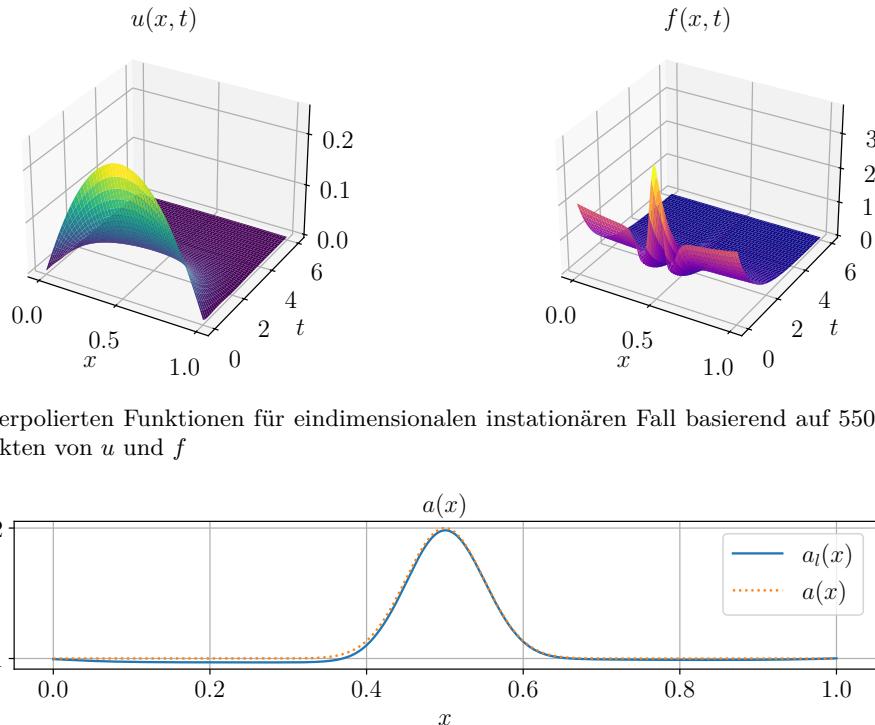
Durch eine zeitliche Auflösung liegt in diesem Fall eine viel größere Anzahl an Trainingspunkten vor, was die Güte der Lösung  $u$  beeinflusst. Vergleicht man die Normabstände für beide Fälle miteinander, so liegt der Unterschied um eine Potenz größer für den zeitabhängigen Fall vor. Besonders gut ist dieser Unterschied auch optisch zu sehen, in der Abbildung 4.6b schmiegt die gelernte Funktion  $a_l$  viel näher an die von  $a$  als in der Abbildung 4.4.

## 2D zeitunabhängig

Zur Beurteilung des stationären zweidimensionalen Falls wurden die Funktionen B.6 bis B.7 herangezogen. Für einen Trainingsdurchlauf aus der Abbildung 4.7 wurde folgendes Setting gesetzt:

- Messungen: 3025 Punkttripels  $\{\hat{u}, \hat{f}, (\hat{x}, \hat{y})\}$  äquidistant über die Geometrie verteilt.
- PDE-Domain:  $\tilde{\Omega} = [0, 1] \times [0, 1]$  mit jeweils 500 äquidistanten Punkten, also 250000 Trainingspunkte.
- $NN_u$  und  $NN_f$  sind mit  $\{2, 20\}$  und PINN mit  $\{3, 30\}$  dimensioniert.
- Lernrate für alle NNs ist  $\lambda = 10^{-2}$ .

Die erreichten Ergebnisse in diesem Fall sind in der Abbildung 4.8 dargestellt. Optisch ist zu erkennen, dass die Wärmeleitfähigkeit  $a$  an den Ecken des Gebiets  $\Omega = [0, 1] \times [0, 1]$  auf Null abfällt, was in der originalen Abbildung B.3 nicht zu beobachten ist. Eine Mögliche



(a) Interpolierten Funktionen für eindimensionalen instationären Fall basierend auf 5500 Messpunkten von  $u$  und  $f$

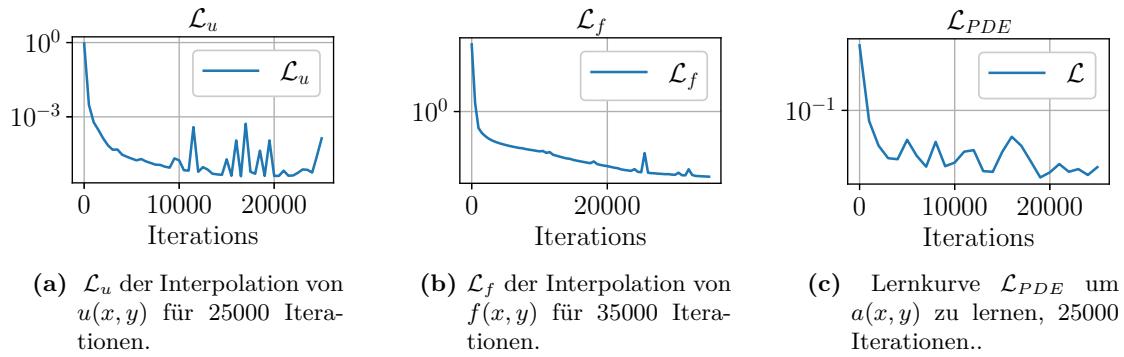
(b) Die Lösung  $a_l$  des inversen Wärmeleitungsproblems im eindimensionalen instationären Fall ist mit blau dargestellt. Die exakte Wärmeleitfähigkeit  $a$  ist zum Vergleich rot-punktiert abgebildet.

**Abbildung 4.6.:** Die Lösung des inversen Wärmeleitungsproblems für einen instationären eindimensionalen Fall. Alle Funktionen wurden an  $X_t = [0, 1] \times [0, 6] = T_t$  mit 1000 in  $X_t$  und 100 in  $T_t$  äquidistanten Punkten ausgewertet. Der Normabstand der Lösung beträgt  $\|a(X_t) - a_l(X_t)\|_{L^2} \approx 6.47 \cdot 10^{-4}$ .

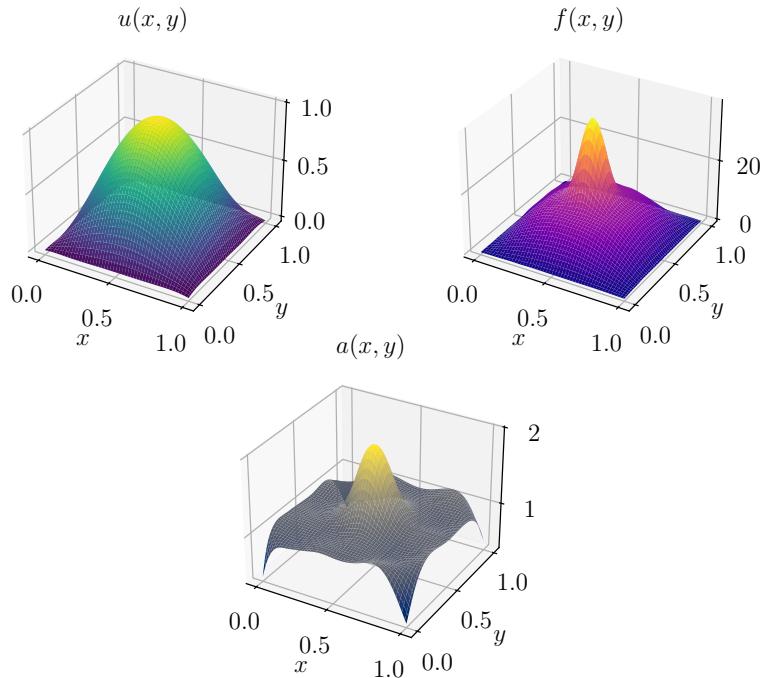
Erklärung dieses Verhaltens könnte so lauten: In diesem konkreten Fall ist es bekannt, dass  $\nabla u = 0$  und  $f = 0$  in den Ecken des Gebiets sind. Also erfüllt jedes  $a$  das PDE-Residuum  $-\langle \nabla a \nabla \bar{0} \rangle - \bar{0}$ . Die einfachste Wahl wäre  $a$  auch auf Null zu setzen. Klar ist aber, das Verfahren ist nicht eingeschränkt, was die Wahl von  $a$  in den Ecken in diesem konkreten Fall anbetrifft.

## 2D zeitabhängig

Für diesen Fall wurden die Funktionen (B.9) und (B.9) herangezogen. Ähnlich wie im Beispiel 4.2.1, sind es von der Struktur her, dieselben Funktionen, die aber mit einer zeitlich abfallenden Komponente  $e^{-t}$  versehen sind. Die Verlustfunktionen zu diesem Test sind in der Abbildung 4.9 zu sehen, dabei wurde der Löser folgendermaßen parametriert:



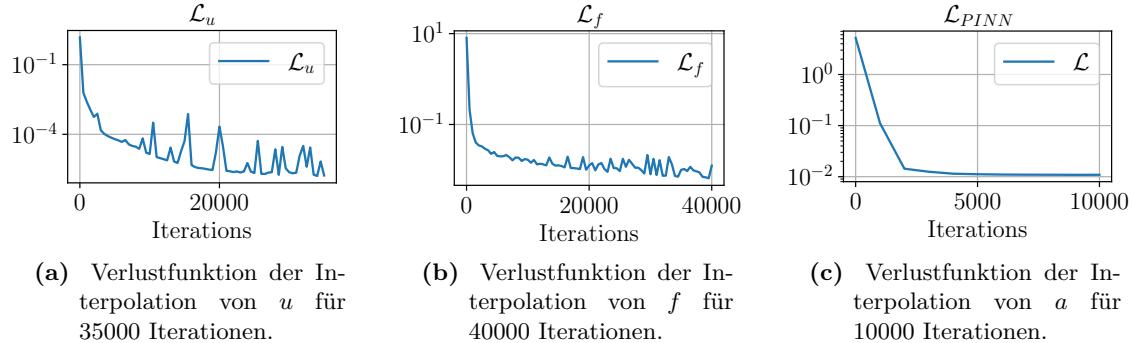
**Abbildung 4.7.:** Die Verlustfunktionen des zweidimensionalen stationären Falls. Die Gesamttrainingszeit lag bei etwa 31.22 Minuten.



**Abbildung 4.8.:** Lösung des inversen Problems für zweidimensionalen stationären Fall.  
Die erreichten Fehlerabstände ausgewertet auf einem Gitter von  $1000 \times 1000$  Punkten erreichten:  $\|u - u_l\|_{L^2} \approx 1.9 \cdot 10^{-3}$ ,  $\|f - f_l\|_{L^2} \approx 5.33 \cdot 10^{-2}$  und  $\|a - a_l\|_{L^2} \approx 9.15 \cdot 10^{-2}$ .

- Messungen: 166375 Punkttripels  $\{\hat{u}, \hat{f}, (\hat{x}, \hat{y}, \hat{t})\}$  äquidistant über die Geometrie und Zeit verteilt.

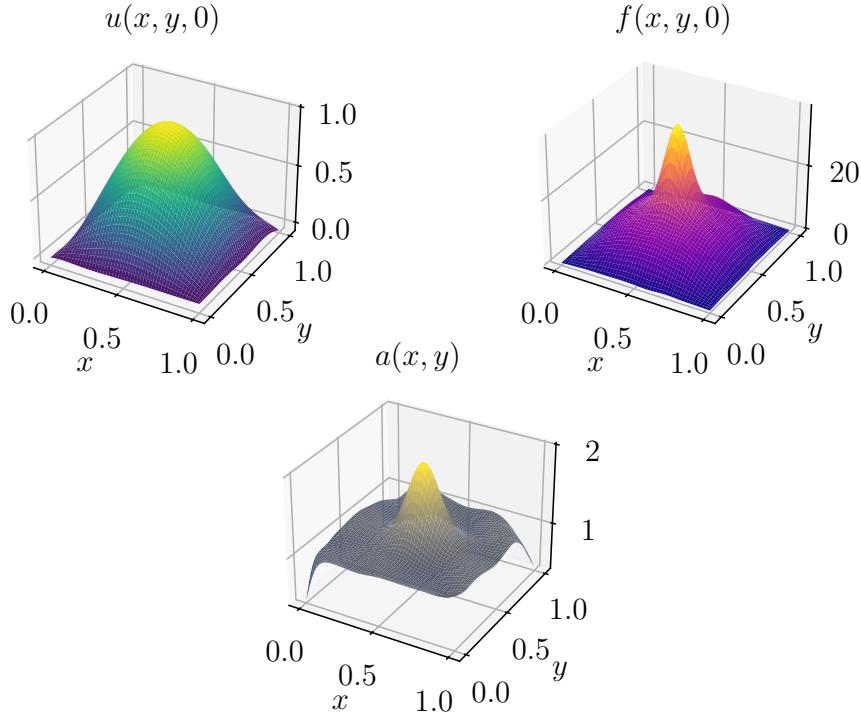
- PDE-Domain:  $\tilde{\Omega} \times \tilde{T} = [0, 1] \times [0, 1] \times [0, 6]$  mit jeweils 50 äquidistanten Punkten.
- $NN_u$  und  $NN_f$  sind mit  $\{2, 20\}$  und PINN mit  $\{3, 30\}$  dimensioniert.
- Lernrate für alle NNs ist  $\lambda = 10^{-2}$ .



**Abbildung 4.9.:** Die Verlustfunktionen des zweidimensionalen instationären Falls. Das Training für  $a$  ist bei etwa 6000 Iterationen auskonvergiert. Die Gesamttrainingszeit lag bei etwa 121.24 Minuten.

Die Ergebnisse dieses Tests sind in der Abbildung 4.10 gezeigt. Die Graphen gelernter Funktionen  $u(x, y, t)$  und  $f(x, y, t)$  sind zum Zeitpunkt  $t = 0$  ausgewertet. Man beobachtet auch hier ein ähnliches Verhalten von  $a$  wie in dem stationären Fall 4.2.1. Die Vermutung hier das Verhalten von  $a$  zu deuten, bleibt dieselbe wie oben.

Abschließend lässt sich festhalten, dass in allen in diesem Abschnitt betrachteten Beispielen ein Konvergenzverhalten hinsichtlich der  $L^2$ -Norm beobachtet werden konnte. Diese numerischen Resultate werden durch den optischen Vergleich der exakten und gelernten Funktionen bekräftigt. Auffällig ist, dass in den instationären Fällen die Verlustfunktionen viel frühere und vor allem stabilere MSE-Konvergenz, also ohne starken Ausreißern, als in den instationären Fällen zeigen. Dieses Verhalten ist der größeren Trainingsmenge zu verdanken, die durch die feine zeitliche Auflösung der Daten entsteht. Dadurch stehen dem Netzwerk deutlich mehr Informationspunkte zur Verfügung, was zu einer verbesserten Approximation und einer zuverlässigeren Rekonstruktion der gesuchten Funktionen führt, insbesondere der Wärmeleitfähigkeit  $a$ . Es ist zu erwarten, dass sich die Genauigkeit der Rekonstruktion durch eine gezielte Optimierung der Hyperparameter weiter verbessert lässt. Ein weiterer Aspekt besteht darin, dass sämtliche Berechnungen ohne zusätzliche Regularisierung durchgeführt wurden, wodurch die Robustheit der Methodik gegenüber der Überanpassung etwas heraussticht.



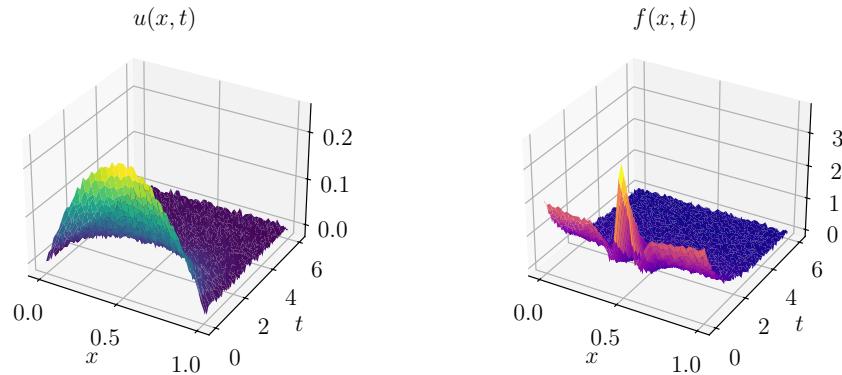
**Abbildung 4.10.:** Lösung des inversen Wärmeleitungsproblems für den zweidimensionalen instationären Fall. Die Fehlerabstände ausgewertet auf einem Raum-Zeit-Gitter von  $100 \times 100 \times 100$  Punkten betragen:  $\|u - u_l\|_{L^2} \approx 1.17 \cdot 10^{-2}$ ,  $\|f - f_l\|_{L^2} \approx 7.53 \cdot 10^{-1}$  und  $\|a - a_l\|_{L^2} \approx 5.4 \cdot 10^{-2}$ .

#### 4.2.2 Test an verrauschten Daten

Ein zentraler Aspekt bei der praktischen Anwendung inverser Methoden ist die Robustheit des Lösungsansatzes gegenüber Messrauschen. In realen Szenarien sind sowohl Temperaturdaten  $u$  als auch ggf. bekannte Wärmequelle  $f$  typischerweise mit Unsicherheiten behaftet, z.B. infolge begrenzter Messgenauigkeit, Sensorrauschen oder unvollständiger Modellierung physikalischer Effekte. In diesem Test soll untersucht werden, wie sich ein Rauschen in den Daten auf die Rekonstruktion der Wärmeleitfähigkeit  $a$  auswirkt.

Der entwickelte `InverseHeatSolver` verzichtet auf den Einsatz vorgelagerter Filtermechanismen, wie sie etwa in der klassischen Signalverarbeitung üblich sind. Stattdessen wird das zugrundeliegende Regressionsproblem direkt durch die Minimierung einer geeigneten Verlustfunktion adressiert. Insbesondere ist es der Interpolationsschritt für die Funktionen  $u$  und  $f$ , der diese Aufgabe übernimmt. Im `Interpolator` kommt dabei nativ die mittlere quadratische Abweichung (MSE) des Keras-Frameworks als Verlustfunktion zum Einsatz, welche äquivalent zur Methode der kleinsten Quadrate ist. Diese Vorgehensweise stellt somit explizit eine Regressionsmethode dar. Diese Eigenschaft des

**Interpolator** kommt bei Fehlerbehafteten Daten zum Tragen, was durch folgende Beispiele bekräftigt wird.



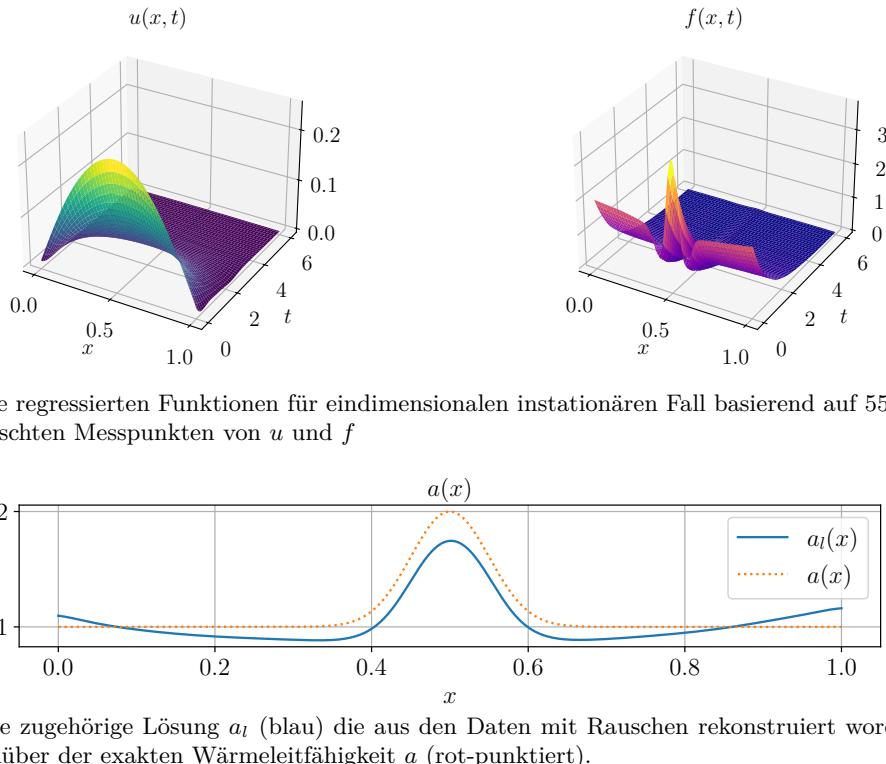
**Abbildung 4.11.:** Ein Beispiel mit verrauschten Daten (eindimensional und zeitabhängig), wo  $u(x, t)$  mit  $\varepsilon_u = 0,005$  und  $f(x, t)$  mit  $\varepsilon_f = 0,05$  Rauschen behaftet ist.

Die Abbildung 4.11 zeigt einen Fall mit Datenrauschen an  $u$  und  $f$ . Das Rauschen wurde mit der Standardnormalverteilung von `numpy.random.randn` erzeugt und entsprechend skaliert, in diesem Fall ist die Temperaturverteilung  $u$  mit  $\varepsilon_u = 0.005$  und die Wärmequelle  $f$  mit  $\varepsilon_f = 0.05$  behaftet. Die Skalierung des Rauschens wurde entsprechend des Definitionsbereichs jeweiliger Datenwerte gewählt, was eine zu starke Streuung vermeidet.

In der Abbildung 4.12 sind die Ergebnisse zu obigen Daten dargestellt. Diese Ergebnisse wurden mit selben Setting wie in dem Test 4.2.1 erstellt. Zusätzlich wurde hier die Regularisierung mitbenutzt und zwar die Tikhonov-Regularisierung (3.20) und die gPINN-Regularisierung (3.21).

In diesem Test erreichten  $L^2$ -Normabstände, ausgewertet an der gleichen Punktmenge wie im Test 4.2.1, folgende Werte:  $\|u_t - u_{l_t}\| \approx 2.53 \cdot 10^{-6}$ ,  $\|a(X_t) - a_l(X_t)\| \approx 3.84 \cdot 10^{-3}$  und  $\|f_t - f_{l_t}\| \approx 2.35 \cdot 10^{-5}$ . Während der Fehler für  $u$  und  $a$  etwa dieselbe Größenordnung hat, ist der Fehler für  $f$  um zwei Potenzen kleiner als im Test 4.2.1.

Das obige Beispiel motivierte einen weiteren Test zu erstellen. Im nächsten Schritt wurde geschaut wie der Löser auf eine stufenweise Erhöhung des Datenrauschens reagiert. Dafür wurde das Rauschen stufenweise bis zu 10% des Wertebereichs der Daten erhöht. Die somit entstandene Tabelle 4.1 gibt die Rauschempfindlichkeit des Lösen in Bezug auf die Rekonstruktion der Wärmeleitfähigkeit  $a$  an. Alle Ergebnisse wurden bis auf den Rauschanteil mit demselben Setting wie oben erstellt.



**Abbildung 4.12.:** Die Lösung des inversen Problems für den eindimensionalen instationären Fall an verrauschten Daten. Alle Funktionen wurden an  $X_t = [0, 1] \times [0, 6] = T_t$  mit 1000 in  $X_t$  und 100 in  $T_t$  äquidistanten Punkten ausgewertet. Der Normabstand der Lösung beträgt  $\|a(X_t) - a_l(X_t)\|_{L^2} \approx 3.84 \cdot 10^{-3}$ .

Die Ergebnisse in Tabelle 4.1 zeigen, dass der Löser auch bei zunehmendem Messrauschen stabile Rekonstruktionen liefert. Der Fehler in  $a$  steigt mit dem Rauschanteil zwar an, bleibt aber auch für 10% Rauschen in einem akzeptablen Bereich. Die Fehler in  $u$  und  $f$  zeigen ein vergleichbares Verhalten. Die Laufzeiten variieren mit dem Rauschanteil nur geringfügig, obwohl für alle Tests die gleiche Anzahl an Trainingspunkten verwendet wurde. Die Resultate belegen, dass die gewählte Regressionsmethode in Kombination mit Regularisierung in der Lage ist, das zugrundeliegendes inverses Problem auch an verrauschten Daten zu lösen.

### 4.2.3 Hyperparametersuche

Die Hyperparametersuche ist der Prozess zur Auswahl optimaler Modellparameter, die nicht durch Training gelernt werden. Beispielsweise ist es die Anzahl der Schichten, Neuronen pro Schicht, Lernrate oder die Gewichte einzelner Verlustfunktionen. Das sind

Noise Error	$\varepsilon_u = 0$ $\varepsilon_f = 0$	$\varepsilon_u = 0.005$ $\varepsilon_f = 0.05$	$\varepsilon_u = 0.008$ $\varepsilon_f = 0.08$	$\varepsilon_u = 0.01$ $\varepsilon_f = 0.1$
$\ u_t - u_{l_t}\ $	$9.28 \cdot 10^{-7}$	$2.53 \cdot 10^{-6}$	$2.02 \cdot 10^{-6}$	$2.17 \cdot 10^{-7}$
$\ f_t - f_{l_t}\ $	$6.45 \cdot 10^{-6}$	$2.35 \cdot 10^{-5}$	$4.26 \cdot 10^{-5}$	$5.77 \cdot 10^{-5}$
$\ a - a_l\ $	$6.16 \cdot 10^{-4}$	$3.84 \cdot 10^{-3}$	$2.08 \cdot 10^{-3}$	$2.06 \cdot 10^{-3}$
Runtime	30.81 min	30.01 min	31.18 min	31.58 min

**Tabelle 4.1.:** Fehlerwerte der Lösungen für verschiedene verrauschte Messdaten von  $\hat{u}$  und  $\hat{f}$  in eindimensionalen instationären Fall. Dabei wurden in allen Fällen die exakten und die gelernten Funktionen an einer äquidistanten Punktmenge  $X_t \times T_t = [0, 1] \times [0, 6]$  mit 1000 Punkten in  $X_t$  und 100 in  $T_t$  ausgewertet. Zusätzlich ist für jeden Fall auch die Laufzeit angegeben. Der Fehler wurden mit der  $L^2$ -Norm gemessen.

einige typische ML Parameter. Aus der Sicht des inversen Problems ist es auch interessant zu schauen wie sich der Löser im Bezug auf Datenknappheit verhält. Konkret will man wissen, wie die Güte der Lösung gegenüber der Datenreduktion sich entwickelt. Eben die Datenempfindlichkeit soll als erstes untersucht werden.

DP Error	11	15	21	25	31
$\Delta_u$	$4.4 \cdot 10^{-1}$	$1.33 \cdot 10^{-1}$	$1.26 \cdot 10^{-1}$	$1.87 \cdot 10^{-1}$	$5.34 \cdot 10^{-1}$
$\Delta_f$	3.07	1.84	$8.35 \cdot 10^{-1}$	1.14	1.18
$\Delta_a$	$2.1 \cdot 10^{-1}$	$1.34 \cdot 10^{-1}$	$2.34 \cdot 10^{-1}$	$2.35 \cdot 10^{-1}$	$1.31 \cdot 10^{-1}$
RT	33.56 min	45.41 min	51.68 min	52.45 min	54 min

**Tabelle 4.2.:** Untersuchung des Einflusses der Anzahl von Datenpunkten (DP), zweidimensionaler instationärer Fall. Hier bezeichnet DP die Anzahl der äquidistanten Punkte in jede Koordinatenrichtung von  $\hat{\Omega} \times \hat{T}$ . RT bezeichnet die Laufzeit für das Training und  $\Delta$  den Fehler, gemessen in der  $L^2$ -Norm.

Die Tabelle 4.2 gibt die Datenabhängigkeit des Lösers an, dabei wurden die Ergebnisse mit folgendem Setting erzielt:

- PDE-Domain:  $\hat{\Omega} \times \hat{T} = [0, 1] \times [0, 1] \times [0, 2]$  mit jeweils 100 äquidistanten Punkten in Raumauflösung und 5 Zeitschritte gesetzt.
- Für jeden Fall wurden die Messdaten mit  $\varepsilon_u = 0.002$ ,  $\varepsilon_f = 0.02$  als Rauschen versehen.
- $NN_u$  und  $NN_f$  und PINN sind mit  $\{3, 30\}$  dimensioniert.

- Lernrate für alle NNs ist  $\lambda = 10^{-2}$ .
- Regularisiert mit Tikhonov-Regularisierung (3.20) und mit gPINN-Regularisierung (3.21).
- Anzahl der Iterationen:  $I_u = 15000$ ,  $I_f = 25000$  und  $I_a = 7000$

In der Tabelle 4.2 ist zu erkennen, dass mit der steigenden Datenanzahl die Lösungen nicht unbedingt stark qualitativ besser sind. Im Vergleich zu Test 4.2.1 (ohne Regularisierung) hat der Löser bei einer Datenmenge von 166375 Punkten eine Genauigkeit für die Wärmeleitfähigkeit von  $\Delta_a \approx 5.4 \cdot 10^{-2}$  in der Zeit von 121.24 Minuten erreicht. Gemessen an der Datenmenge zu dem letzten Fall hier, ist es ein Verhältnis von mehr als 1:10, lässt sich sagen, dass das Ergebnis an vielen Daten nicht signifikant besser ausfällt.

Als nächstes soll nun die Empfindlichkeit der Wärmeleitfähigkeit zu der Dimensionierung der NNs untersucht werden. Zu hervorzuheben ist, dass die NNs für  $u_l$ ,  $f_l$  vortrainiert sind und für jeden Durchlauf dieses Tests gleich blieben. Das heißt es wird nur auf die Dimensionierung des PINNs geschaut und auf die daraus resultierende Empfindlichkeit  $\Delta_a$  in  $L^2$ -Norm der Lösung  $a_l$  zur exakten Funktion  $a$ . Bedingt durch lange Laufzeiten zum Anlernen des PINNs wurde dieser Test für eindimensionalen instationären Fall durchgeführt. Die NNs für  $u_l$  und  $f_l$  sind an Verrauschten Daten mit  $\varepsilon_u = 0.005$  und  $\varepsilon_f = 0.05$  mit 5500 Punkttripels  $\{\hat{u}, \hat{f}, (\hat{x}, \hat{t}_j)\}$  trainiert worden und liefern eine Genauigkeit von  $\Delta_u \approx 2.53 \cdot 10^{-6}$  und  $\Delta_f \approx 2.35 \cdot 10^{-5}$ , gemessen in der  $L^2$ -Norm. Die Parameter des PINN sind: PDE-Domain:  $\tilde{\Omega} = [0, 1]$  mit 15 und  $\tilde{T} = [0, 2]$  mit 50 äquidistanten Punkten, Lernrate  $\lambda = 10^{-2}$  und 5000 Iterationen. Somit wird nur die Anzahl der Layers und Neurone variiert. Die Ergebnisse sind in der Tabelle 4.3 angegeben.

Layers \ Neurons	10	20	30	40
2	$4.48 \cdot 10^{-3}$	$4.62 \cdot 10^{-3}$	$4.57 \cdot 10^{-3}$	$4.53 \cdot 10^{-3}$
3	$4.56 \cdot 10^{-3}$	$4.57 \cdot 10^{-3}$	$4.32 \cdot 10^{-3}$	$4.28 \cdot 10^{-3}$
4	$4.46 \cdot 10^{-3}$	$4.54 \cdot 10^{-3}$	$4.56 \cdot 10^{-3}$	$4.48 \cdot 10^{-3}$

**Tabelle 4.3.:** Einfluss der Dimensionierung des PINNs in einem eindimensionalen statio-nären Fall. Angegeben sind Werte von  $\Delta_a$  als  $L^2$ -Normabstand. Die NNs für  $u$  und  $f$  wurden vortrainiert und in allen Fällen gleich gehalten.

Anhand der Tabelle 4.3 zeichnet es sich heraus, dass die Dimensionierung des PINNs keine tragende Rolle bei der Lösungsgenauigkeit spielt.

Abschließend kann gesagt werden, dass die Güte der inversen Lösung stark von der Gü-te der Interpolierten Funktionen und von der Menge der Trainingsdaten abhängt. Die Interpolation selbst hängt natürlich von der Anzahl der Messdaten ab. Aber selbst bei

einer Messdatenanzahl von 11 Punkten, einer PINN Größe von (3, 20) und einer Trainingsmenge von etwa 55500 Punkten kriegt man eine Güte der inversen Lösung von der Größenordnung  $10^{-3}$  und das sogar für Verrauschte Daten für  $u$  und  $f$ .

Weitere Modellparameter wie Optimierer oder Aktivierungsfunktion wurden konstant gehalten, da ihre systematische Variation den Rechenaufwand unverhältnismäßig erhöht hätte. Ebenso wurden die Gewichte einzelner Verlustterme und Lernraten auf empirisch bewährten Werten belassen, da diese während des gesamten Entwicklungsprozesses systematisch angepasst worden sind.

## Kapitel 5

# Diskussion und Ausblick

Die vorliegende Arbeit hat nun das inverse Problem der stationären und instationären Wärmeleitung zur Bestimmung des ortsabhängigen Wärmeleitfähigkeit mit einem Verfahren des ML behandelt. Im Folgenden soll eine abschließende Reflexion der erzielten Ergebnisse, der Herangehensweise und der offenen Herausforderungen erfolgen.

### **Diskussion erreichter Ziele**

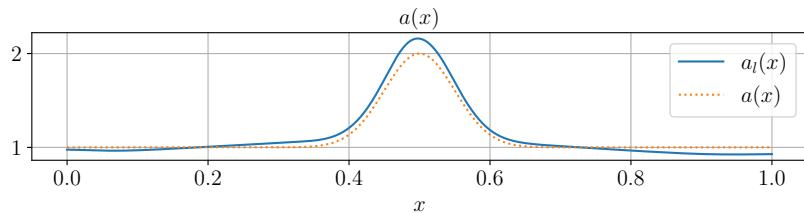
Es ist gelungen eine einführende Darstellung der theoretischen Grundlagen für das Problem der Wärmeleitung aufzustellen. Die auf diesem Wege hergeleitete Differentialgleichung der Wärmeleitung dient als Fundament aller folgender Ansätze.

Obwohl das Vorwärtsproblem der Wärmeleitung nicht im zentralen Fokus dieser Arbeit steht, musst es verstanden werden. Die Kenntnis über das Vorwärtsproblem ist im Kontext des inversen Problems nach Meinung des Autors unabdingbar, den es vermittelt zumindest die Grundlagen die bei dem numerischen Ansatz des inversen Problems auch tragend sind. Zudem konnte die Lösung auf kurzem Wege und unkompliziert mit der Software-Bibliothek FEniCS angegeben werden. Eine Besonderheit an dieser Stelle ist, dass FEniCS nur auf Linux-Derivaten funktioniert. Dieser Umstand konnte mithilfe eines Docker-Containers unter Windows aufgehoben werden. Auf der anderen Seite konnte dasselbe Problem mithilfe der DeepXDE Software-Bibliothek mit PINNs gelöst werden. Die dabei erzielten Ergebnisse sind plausibel und vergleichbar zu den Lösungen mit FEniCS. Jedoch ist es zu erwähnen, dass das Verfahren mit PINNs deutlich dem numerischen unterlegen ist. Das zeichnet sich besonders über die Laufzeiten und die Genauigkeit des Ergebnisses ab. Während die Laufzeit betrachteter Beispiele eine Differenz von etwa 3s aufzeigt, liegt die Differenz der Genauigkeit gemessen mit dem  $L^2$ -Abstand bei etwa  $10^{-9}$ . Dieser Unterschied ist signifikant und deutet darauf hin, dass der heutige Stand der PINN-Technik keinen Vorteil gegenüber den numerischen Verfahren beim Lösen des Vorwärtsproblems bietet.

Bei der Betrachtung des inversen Problems der Wärmeleitung zur Bestimmung der Wär-

meleitfähigkeit stoßt man zuerst auf dessen schlechte Gestelltheit, was die Berechnung der Wärmeleitfähigkeit deutlich erschwert. Die schlechte Gestelltheit des inversen Problems ist das Haupthindernis für einen strikten Ansatz, wie es bei dem Vorwärtsproblem der Fall ist. Der vorgestellte Ansatz der erweiterten Lagrange-Methode aus [4] liefert zwar zuverlässige Ergebnisse, ist aber nur für das stationäre Problem ausgelegt. In diesem Zusammenhang werden nun die PINNs des inversen Problems eingeführt. Der vorgestellte Ansatz mit PINNs konnte ohne theoretischen Umwege direkt auf das Problem angewandt werden. Das ist nach Meinung des Autors ein sehr spannender Punkt, der eventuell in der Lehre zum tragen kommen kann. Denn dieser Ansatz vereinfacht den Zugang zum Verständnis des Problems dadurch, dass das PDE-Residuum ohne Umformulierung in die Verlustfunktion des PINNs direkt aufgenommen werden kann. Mit dieser Herangehensweise und der Benutzung bestehender Software-Pakete, dessen Wahl nicht nur auf hier benutzte sich einschränkt, kann ein inverser Löser relativ schnell und unkompliziert umgesetzt werden. Hier konnte leider kein Vergleich zu dem numerischen Verfahren aufgestellt werden, da eine selbstständige Implementierung aufwendig wäre und den Rahmen dieser Arbeit bei weitem übertreffen würde.

Die vorgestellten Testergebnisse deuten auf die Tauglichkeit des Lösen für die vier behandelten Fälle hin, was sich vor Allem über die erreichten  $L^2$ -Normabstände zu den Referenzen exakter Funktionen äußert. In erster Linie zeigt es sich mit erreichten Ergebnissen für die exakten Messdaten. Der Test mit verrauschten Daten gilt auch als bestanden. Selbst bei relativ stark Verrauschten Daten (etwa 10% des Daten-Wertebereichs) bekommt man eine Wärmeleitfähigkeit  $a_l$ , dessen Profil der exakten Lösung relativ nahe kommt (Abb. 5.1).



**Abbildung 5.1.:** Die Wärmeleitfähigkeit als Lösung des inversen Problems mit 10% verrauschten Daten für einen eindimensionalen instationären Fall.

Bei der Hyperparametersuche stellte es sich heraus, dass der Löser nicht sensitiv auf Änderungen der PINN-Größe reagiert. Es hat sich gezeigt, dass die Vergrößerung des PINNs nur größeren zeitlichen Aufwand mit sich bringt, aber keinerlei mehr Lösungsgenauigkeit. Vielmehr ist es möglich mit relativ kleinen PINNs die Lösungsgüte von etwa  $10^{-3}$  erreicht werden.

Rückblickend lässt sich sagen, dass die zentrale Errungenschaft dieser Arbeit in der Implementierung eines Frameworks zur Lösung eines inversen Problems der Wärmelei-

tung liegt. Der `InverseHeatSolver` kann als eigenständiges Modul in anderer Software benutzt werden. Durch den modularen Aufbau, kann der Löser mit wenig Aufwand erweitert oder beispielsweise das Backend gewechselt werden. Diese Eigenschaften könnten eventuell bei Interessenten zum tragen kommen.

## Ausblick

Zu jedem Vorteil lässt sich vermutlich auch ein Nachteil finden, so ist es auch hier. In diesem Ausblick sollen die Schwachstellen dieses Ansatzes und seine Erweiterungsmöglichkeiten kurz genannt und diskutiert werden.

Als erstes soll ein nicht gelungener Test erwähnt werden. Der `InverseHeatSolver` ist leider nicht im Stande das homogene inverse Problem der Wärmeleitung zu lösen, also  $u_t - \langle \nabla, a \nabla u \rangle = 0$ . Dieser Umstand konnte aus Zeitgründen nicht näher geklärt werden. Vermutlich würde eine Umformulierung in eine schwache Formulierung des Problems an dieser Stelle helfen.

Eine richtige Schwachstelle die generell sich bei dem Ansatz der PINNs auszeichnet ist die der Generalisierung. Also im Vergleich zu einem numerischen Löser, vorausgesetzt seine Implementierung existiert, funktioniert der Ansatz der PINNs nur für angelernte Probleme. Somit muss jedes weitere Problem neu angelernt und als Modell gespeichert werden. Wo ein numerischer Algorithmus nach seiner Implementierung für ein ganze Klasse von Problemen funktionieren würde. Ein Versuch das PINN zu generalisieren wäre zum Beispiel der DeepONet-Ansatz. Hier wird versucht den Operator der vorliegenden DGL zu lernen (Vgl. [13]).

Ein weiterer Verbesserungsvorschlag wäre eine realistischere Abbildung der Physik. Zum Beispiel, dass die Wärmeleitfähigkeit  $a$  nicht nur Ortsabhängig sondern auch Temperaturabhängig ist, also  $a(x, u(x, t))$ . Und eine Ausweitung bis zur dritten Dimension wäre an dieser Stelle auch denkbar.

## Anhang A

# Benutzte Frameworks

### A.1 DeepXDE

**DeepXDE** (*Deep Learning Library for Differential Equations*) ist eine offene Python-Bibliothek zur Lösung von gewöhnlichen und partiellen Differentialgleichungen (ODEs und PDEs) mit Hilfe von *Physics-Informed Neural Networks* (PINNs). Sie wurde von Lu Lu und George Karniadakis entwickelt und unterstützt sowohl Vorwärts- als auch inverse Probleme in Wissenschaft und Technik.

Die Wahl dieses Framework zu nutzen ist mit folgenden Punkten motiviert worden:

#### 1. Breite Gleichungsunterstützung

- ODEs, PDEs, integro-differenzielle, fraktionale und stochastische Gleichungen.
- Vorwärts- und inverse Probleme.

#### 2. Modularer Aufbau

- Geometrien, Randbedingungen und Netzwerkarchitekturen sind getrennt definierbar.
- Unterstützung für Feedforward-Netze, DeepONets u. a.

#### 3. Mehrere Backend-Engines

- TensorFlow 1.x, 2.x
- PyTorch
- JAX
- PaddlePaddle

Neben DeepXDE existiert eine Reihe weiterer Frameworks/Bibliotheken die im selben Kontext benutzt werden könnten. Hier soll nur einige davon mit Vor- und Nachteilen aufgelistet werden.

#### TorchPhysics (PyTorch)

Modular aufgebautes Framework zur Lösung von PDEs über PINNs. Unterstützt

benutzerdefinierte Bedingungen und Geometrien. Ein vergleichbares Framework zu DeepXDE, aber keine ausführliche Dokumentation.

### NVIDIA Modulus

Hochoptimierte Plattform für industrielle PINN-Anwendungen, mit Fokus auf GPU-Nutzung und Deployment. Starkere Bindung an spezifische Hardware ist ein Nachteil gegenüber DeepXDE.

### PinnDE

Jüngere Python-Bibliothek mit einfacher API und Unterstützung für verschiedene Architekturen, u. a. DeepONets. Nach einer subjektiven Einschätzung des Verfassers, ist die Bibliothek noch stark in Entwicklung und der Zweck deutet mehr auf einen Einsatz in der Lehre.

Die oberen Ausführungen bekräftigten die Wahl des Frameworks DeepXDE für das Vorhaben der vorliegenden Arbeit.

## A.2 FEniCS

**FEniCS** ist ein freies Open-Source-Framework zur numerischen Lösung partieller Differentialgleichungen (PDEs) mit der *Finite-Elemente-Methode (FEM)*. Es ermöglicht auf low-level Programmierung die high-level Variationsformulierungen der PDEs zu definieren und zu lösen.

### Technische Kernmerkmale

- **High-Level-Formulierung:** Variationsformen können direkt in Python oder C++ beispielsweise als

$$a = \int_{\Omega} \nabla u \cdot \nabla v \, dx, \quad L = \int_{\Omega} f v \, dx$$

definiert werden.

- **Schwache Formulierung:** FEniCS basiert auf der Formulierung in Sobolev-Räumen ( $H^1, H^2$ ) und übernimmt interne Umformungen (z. B. partielle Integration).
- **Form-Compiler:** Das DSL-Backend UFL wird vom FFC (FEniCS Form Compiler) in effizienten C++-Code übersetzt.

### Komponenten

- **dolfin:** High-Level-Schnittstelle und numerisches Backend.
- **ufl:** Unified Form Language zur Variationsformulierung.
- **ffc:** Kompiliert schwache Formen in ausführbaren Code.
- **meshio, mshr:** Werkzeuge zur Mesh-Erstellung und -Verarbeitung.

## FEniCS unter Windows

Die native Ausführung von FEniCS ist ausschließlich auf Linux-Derivaten möglich, was eine direkte Nutzung unter Windows erschwert. Im Folgenden wird eine praxiserprobte Vorgehensweise beschrieben, mit der sich FEniCS dennoch unkompliziert unter Windows verwenden lässt. Das gewählte Verfahren basiert auf der Nutzung von **Docker-Containern**, welche eine vollständige Linux-Umgebung zur Verfügung stellen und die Ausführung von FEniCS-Skripten unabhängig vom verwendeten Betriebssystem ermöglichen.

## Einrichtung des FEniCS-Containers

### 1. Dockerfile

```
...
# official Ubuntu 22.04 base image
FROM ubuntu:22.04

# Update package index, install wget, bzip2 and Miniconda
RUN apt-get update && \
    apt-get install -y wget bzip2 && \
    wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh \
    -O miniconda.sh && \
    bash miniconda.sh -b -p /opt/conda && \
    rm miniconda.sh

# Miniconda to the PATH environment
ENV PATH="/opt/conda/bin:$PATH"

# Conda to install FEniCS and scientific Python libs
RUN conda install -c conda-forge fenics matplotlib numpy scipy plotly

# Container working directory
WORKDIR /app
...
```

Der Ausschnitt der bereitgestellten `FEniCS_Dockerfile` Datei erstellt einen Ubuntu-Container in dem eine `miniconda`-Umgebung mit entsprechenden Python Bibliotheken installiert wird. Die Datei sollte zusammen mit den zugehörigen Python-Skripten (z.B. `FEniCS_1D_ti.py`, `FEniCS_2D_td.py`, `test_cases_FEniCS.py`) in einem Arbeitsverzeichnis liegen (hier direkt im Verzeichnis `code/InverseHeatSolver/FEniCS_scripts`).

## 2. Docker-Image bauen:

Zum Bauen des Containers kann folgender Befehl beispielsweise in einer cmd Konsole ausgeführt werden:

```
docker build -f FEniCS_Dockerfile -t fenics_env .
```

Dieser Befehl erstellt ein Docker-Image mit dem Namen `fenics_env`.

## 3. Container starten und Skript ausführen:

Zum Ausführen eines bestimmten Skripts in einer PowerShell Konsole, beispielsweise `FEniCS_1D_td.py`, kann folgender Befehl verwendet werden:

```
docker run --rm `  
    -v "$(Resolve-Path ..):/workspace" `  
    -w /workspace/FEniCS_scripts `  
    -e PYTHONPATH=/workspace `  
    fenics_env python3 FEniCS_1D_td.py
```

In einer cmd Konsole sähe der Befehl folgendermaßen aus:

```
docker run --rm  
    -v <FULL\PATH\TO>\Code\InverseHeatSolver:/workspace  
    -w /workspace/FEniCS_scripts  
    -e PYTHONPATH=/workspace  
    fenics_env python3 FEniCS_1D_td.py
```

Damit werden wird der Parent-Path (hier `/InverseHeatSolver`) als Full-Path aufgelöst, als Arbeitsverzeichnis für den Container gemacht und die Ausgabedateien direkt auf dem Host im Aufrufordner geschrieben.

## 4. Hinweise

- **Dateiaustausch:** Der Parameter `-v` im obigen Befehl sorgt dafür, dass Änderungen an Skripten und erzeugte Daten sowohl auf dem Windows-System als auch im Container direkt sichtbar sind.
- **Persistenz:** Der Container wird nach der Ausführung automatisch gelöscht (`-rm`).
- **Grafische Ausgaben:** Plots werden wie gewohnt als Dateien (z. B. PNG oder EPS) gespeichert und sind nach der Ausführung im Arbeitsverzeichnis auf Windows zugänglich.

## Anhang B

# Betrachtete Beispiele

Für die Testszenarien der Implementierung wurden 4. Fälle konstruiert. Es wurden also jeweils ein oder zwei dimensionale Funktionen betrachtet und hinzu deren jeweilige Zeitabhängigkeit. Wobei die Wärmeleitfähigkeit immer als raumabhängig betrachtet wird, also  $a : \Omega \rightarrow \mathbb{R}$ . Den Ausdruck für  $f$  gewinnt man durch Einsetzen von  $u$  und  $a$  in 2.10 und durch das direkte Ausrechnen. Alle Handrechnungen und die dazugehörigen Plots der Funktionsgraphen sind übersichtlich in einem *Jupyter Notebook cases\_and\_plots.ipynb* zusammengefasst.

### 1. Eindimensionaler stationärer Fall:

Auf dem Intervall  $\Omega = [0, 1]$  und mit Konstanten  $\sigma = 0.05, \mu = 0.5$  wurden folgende Funktionen definiert:

$$u(x) = x(1 - x) \tag{B.1}$$

$$a(x) = 1 + e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{B.2}$$

$$f(x) = 2 + \frac{1}{\sigma^2} (2\sigma^2 + (1 - 2x)(x - \mu)) e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{B.3}$$

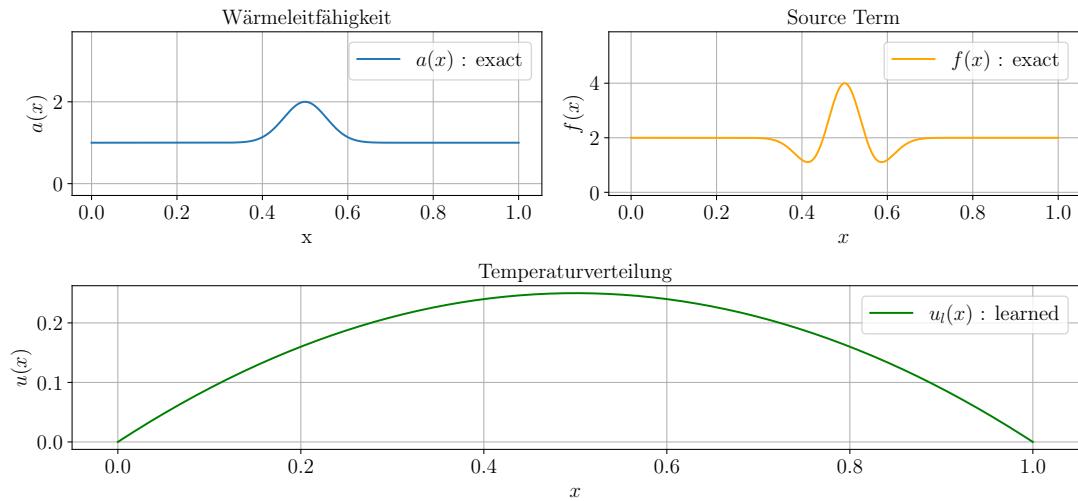
Die Grafen der Funktionen (B.1) bis (B.3) sind in der Abbildung B.1 zu sehen.

### 2. Eindimensionaler instationärer Fall:

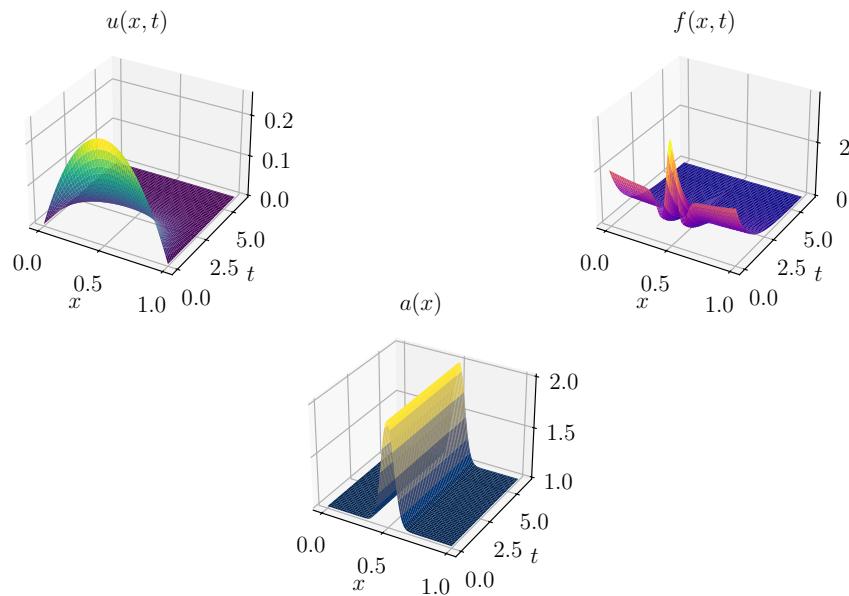
Hier wurden selben Parameter  $\mu$  und  $\sigma$  wie oben gewählt und der Definitionsbereich als  $\Omega \times [t_s, t_e]$  gesetzt. In der Abbildung B.2 sind die Verläufe folgender Funktionen zu sehen:

$$u(x, t) = x(1 - x)e^{-t} \tag{B.4}$$

$$f(x, t) = e^{-t} \left( \frac{1}{\sigma^2} ((1 - 2x)(x - \mu) + 2\sigma^2) e^{-\frac{(x-\mu)^2}{2\sigma^2}} - x(1 - x) + 2 \right). \tag{B.5}$$



**Abbildung B.1.:** Graphen exakter Funktionen für die Testung des eindimensionalen stationären Falls, Beispiel 1.



**Abbildung B.2.:** Exakte Funktionen für das Testbeispiel 2: Eindimensionaler instationärer Falls. Man beachte, dass  $a(x)$  nicht zeitabhängig ist und daher auf der  $t$ -Achse konstant ist.

### 3. Zweidimensionaler stationärer Fall:

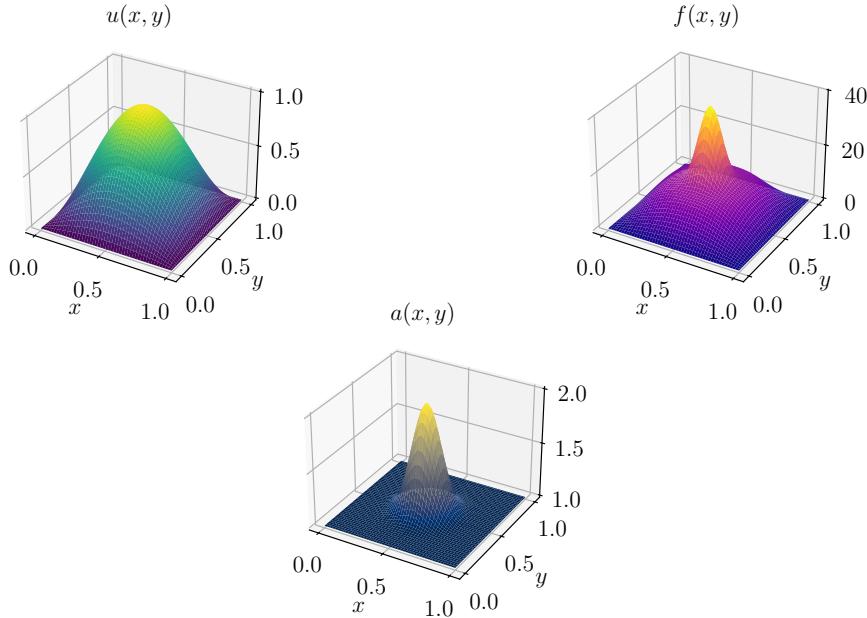
Dieser Fall ist auf der Menge  $I = [0, 1] \times [0, 1]$  mit entsprechenden Parametern

$\sigma = 0.05$ ,  $\mu_x = 0.5$ ,  $\mu_y = 0.5$  definiert. In diesem und im letzten Fall gilt für die Wärmeleitfähigkeit natürlich eine Anpassung  $a : I \rightarrow \mathbb{R}$ .

$$u(x, y) = \sin(\pi x) \sin(\pi y) \quad (\text{B.6})$$

$$a(x, y) = 1 + e^{-\frac{(x-\mu_x)^2+(y-\mu_y)^2}{2\sigma^2}} \quad (\text{B.7})$$

$$\begin{aligned} f(x, y) &= \frac{\pi}{\sigma^2} ((x - \mu_x) \cos(\pi x) \sin(\pi y) + (y - \mu_y) \sin(\pi x) \cos(\pi y) \\ &\quad + 2\pi\sigma^2 \sin(\pi x) \sin(\pi y)) e^{-\frac{(x-\mu_x)^2+(y-\mu_y)^2}{2\sigma^2}} + 2\pi^2 \sin(\pi x) \sin(\pi y) \end{aligned} \quad (\text{B.8})$$



**Abbildung B.3.:** Graphen exakter Funktionen für die Testung des zweidimensionalen stationären Falls des Beispiels 3.

#### 4. Zweidimensionaler instationärer Fall:

Hier sind die Parameter  $\sigma$ ,  $\mu_x$ ,  $\mu_y$  wie im 3. Fall geblieben, nur der Definitionsbereich wurde auf  $I \times [t_s, t_e]$  angepasst.

$$u(x, y, t) = e^{-t} \sin(\pi x) \sin(\pi y) \quad (\text{B.9})$$

$$\begin{aligned}
 f(x, y, t) = & e^{-t} \left( \frac{\pi}{\sigma^2} ((x - \mu_x) \cos(\pi x) \sin(\pi y) + (y - \mu_y) \sin(\pi x) \cos(\pi y) \right. \\
 & \left. + 2\pi\sigma^2 \sin(\pi x) \sin(\pi y) \right) e^{-\frac{(x-\mu_x)^2+(y-\mu_y)^2}{2\sigma^2}} + (2\pi^2 - 1) \sin(\pi x) \sin(\pi y)
 \end{aligned} \tag{B.10}$$

Eine Abbildung der Funktionen (B.9) und (B.10) ist insofern nicht notwendig, dass sie dieselbe Form wie in der Abbildung B.3 haben und mit der fortschreitender zeitlichen Komponente abklingen.

## Anhang C

# Inhalte der vollen Masterarbeit

### C.1 Sourcecode

Der gesamte Sourcecode der Masterarbeit ist in der GitHub Repository: **Radminir Gesler Master Arbeit** zu finden

### C.2 PDF-Dateien

- MA\_Gesler.pdf - gesamte Masterarbeit im PDF-Format

### C.3 Python Code

- Den Sourcecode der Implementierung findet man im Verzeichnis /code/InverseHeatSolver. Dabei sind unten aufgeführte Inhalte für gesamte Arbeit wesentlich.

```
|-- FEniCS_scripts
|   |-- direct_1D_fenics.eps
|   |-- FEniCS_1D_td.py
|   |-- FEniCS_1D_ti.py
|   |-- FEniCS_2D_td.py
|   |-- FEniCS_Dockerfile
|   |-- ForwardHeatSolver.py
|   |-- test_cases_FEniCS.py
|-- solver
|   |-- History.py
|   |-- Interpolator.py
|   |-- InverseHeatSolver.py
|   |-- ModelComposer.py
|   |-- PdeMinimizer.py
|   |-- PdeMinimizerDeepXde.py
|   |-- Visualizer.py
|-- cases_and_plots.ipynb
```

```
|-- dde_forward_1D_ti.py  
|-- dde_forward_2D_ti.py  
|-- functions.py  
|-- requirements.txt  
|-- use_cases.py  
|-- use_cases_deep_xde.py
```

Der Ordner **FEniCS\_scripts** enthält Skripte für die FEM Berechnung des Vorwärtsproblems mit FEniCS. Der Ordner **solver** enthält alle Objekte zur Lösung des inversen Problems.

# Quellenverzeichnis

## Literatur

- [1] Tewes Andreas. „Solutions of inverse problems of the one-dimensional heat conduction equation using Physics Informed Neural Networks“. In: (2024) (siehe S. 33).
- [2] Douglas N. Arnold und Hongtao Chen. „Finite element exterior calculus for parabolic problems“. In: *ESAIM: Mathematical Modelling and Numerical Analysis* (2016). URL: <http://dx.doi.org/10.1051/m2an/2016013> (siehe S. 14).
- [3] Yongqiang Cai. „Vocabulary for Universal Approximation: A Linguistic Perspective of Mapping Compositions“. In: *Proceedings of the 41st International Conference on Machine Learning*. 2024, S. 5189–5208. URL: <https://proceedings.mlr.press/v235/cai24a.html> (siehe S. 20).
- [4] Zhiming Chen und Jun Zou. „An Augmented Lagrangian Method for Identifying Discontinuous Parameters in Elliptic Systems“. In: *SIAM Journal on Control and Optimization* 37 (1999) (siehe S. 3, 56).
- [5] G. Cybenko. „Approximation by superpositions of a sigmoidal function“. In: *Mathematics of Control, Signals and Systems* (1989) (siehe S. 20, 21).
- [6] Wolfgang Demtröder. *Experimentalphysik I*. 4. Aufl. Springer, 2005 (siehe S. 2).
- [7] Knaber P. Eck C. Garke H. *Mathematische Modellierung*. 3. Auflage. Springer-Spektrum, 2017 (siehe S. 15, 16).
- [8] Halliday. *Physik*. 2. Aufl. Wiley-VCH, 2009 (siehe S. 1).
- [9] Teeratorn Kadeethum, Thomas M Jørgensen und Hamidreza M Nick. *Physics-informed Neural Networks for Solving Inverse Problems of Nonlinear Biot's Equations: Batch Training*. 2020. URL: <https://arxiv.org/abs/2005.09638> (siehe S. 29).
- [10] Yee Lo Keung und Jun Zou. „An Efficient Linear Solver for Nonlinear Parameter Identification Problems“. In: *SIAM Journal on Scientific Computing* (2001). URL: <https://doi.org/10.1137/S1064827598346740> (siehe S. 16).
- [11] R. Laubscher. „Simulation of multi-species flow and heat transfer using physics-informed neural networks“. In: *Physics of Fluids* (2021). URL: <http://dx.doi.org/10.1063/5.0058529> (siehe S. 33).

- [12] Zhou Lu u.a. *The Expressive Power of Neural Networks: A View from the Width*. 2017. arXiv: 1709.02540 [cs.LG]. URL: <https://arxiv.org/abs/1709.02540> (siehe S. 20).
- [13] Yifan Mei u.a. „Forward and inverse problems for Eikonal equation based on DeepONet“. In: (2023). URL: <https://arxiv.org/abs/2306.05754> (siehe S. 57).
- [14] M. Raissi, P. Perdikaris und G.E. Karniadakis. „Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving non-linear partial differential equations“. In: *Journal of Computational Physics* (2019). URL: <https://www.sciencedirect.com/science/article/pii/S0021999118307125> (siehe S. 3).
- [15] Maziar Raissi, Paris Perdikaris und George Em Karniadakis. „Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations“. In: *arXiv preprint arXiv:1711.10561* (2017) (siehe S. 3, 18).
- [16] Stefan Richter. *Statistisches und maschinelles Lernen*. Springer Spektrum, 2019 (siehe S. 19–21).
- [17] Andres Rieder. *Keine Probleme mit inversen Problemen*. 1. Aufl. Vieweg, 2003 (siehe S. 3, 14, 15).
- [18] Shyam Sankaran u.a. „On the impact of larger batch size in the training of Physics Informed Neural Networks“. In: (2022). URL: <https://openreview.net/forum?id=THCvohg1RV> (siehe S. 29).
- [19] Ben Schweizer. *Partielle Differentialgleichungen*. 2. Aufl. Springer Spektrum, 2018 (siehe S. 10).
- [20] Sifan Wang, Xinling Yu und Paris Perdikaris. *When and why PINNs fail to train: A neural tangent kernel perspective*. 2020. URL: <https://arxiv.org/abs/2007.14527> (siehe S. 29).
- [21] Jeremy Yu u.a. „Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems“. In: *CoRR* (2021). URL: <https://arxiv.org/abs/2111.02801> (siehe S. 31).
- [22] Frank Haußer Yuri Luchko. *Mathematische Modellierung mit MATLAB und Octave*. Springer Spektrum, 2019 (siehe S. 7).

# Abbildungsverzeichnis

2.1.	Schematische Darstellung zur Herleitung der Bilanzgleichung der Wärmeleitung . . . . .	7
2.2.	Die approximierte Lösung $u_l(x)$ der eindimensionalen stationären Wärmeleitungsgleichung (unten), die vorgegebene ortsabhängige Wärmeleitfähigkeit $a(x)$ (oben links) und die ortsabhängige Heizdichte $f(x)$ (oben rechts), gelöst mit der FEniCS-Library . . . . .	13
3.1.	Schematische Darstellung eines NNs als gerichteter azyklischer Graph . . . . .	23
3.2.	Ablaufdiagramm des Trainingsloops eines PINNs zur direkten Lösung der stationären Wärmeleitungsgleichung . . . . .	25
3.3.	Verlauf der Verlustfunktion beim Training des PINN zur direkten Lösung der stationären eindimensionalen WLG . . . . .	26
3.4.	Lösung der stationären eindimensionalen WLG . . . . .	27
3.5.	Ablaufdiagramm des Trainingsloops eines PINNs zur Lösung der inversen Problems zur Bestimmung der Wärmeleitfähigkeit. . . . .	28
4.1.	Das separierte Modelltraining des PINNs des inversen Wärmeleitungsproblems für Wärmeleitfähigkeit . . . . .	34
4.2.	Das UML-Diagramm des InverseHeatSolvers . . . . .	38
4.3.	Die Verlustfunktionen des eindimensionalen stationären Falls . . . . .	43
4.4.	Die Lösung des eindimensionalen stationären Problems der inversen Wärmeleitung . . . . .	44
4.5.	Die Verlustfunktionen des eindimensionalen instationären Falls . . . . .	45
4.6.	Die Lösung des inversen Wärmeleitungsproblems für einen instationären eindimensionalen Fall . . . . .	46
4.7.	Die Verlustfunktionen des zweidimensionalen stationären Falls . . . . .	47
4.8.	Lösung des inversen Problems für zweidimensionalen stationären Fall . . . . .	47
4.9.	Die Verlustfunktionen des zweidimensionalen instationären Falls . . . . .	48
4.10.	Lösung des inversen Wärmeleitungsproblems für den zweidimensionalen instationären Fall . . . . .	49
4.11.	Beispiel mit verrauschten Daten (eindimensional und zeitabhängig) . . . . .	50
4.12.	Die Lösung des inversen Problems für den eindimensionalen instationären Fall an verrauschten Daten . . . . .	51

5.1. Die Wärmeleitfähigkeit als Lösung des inversen Problems mit 10% ver- rauschten Daten für einen eindimensionalen instationären Fall. . . . .	56
B.1. Graphen exakter Funktionen für die Testung des eindimensionalen statio- nären Falls, Beispiel 1. . . . .	63
B.2. Testbeispiel 2: Eindimensionaler stationärer Fall . . . . .	63
B.3. Graphen exakter Funktionen für die Testung des zweidimensionalen sta- tionären Falls des Beispiels 3. . . . .	64

# Tabellenverzeichnis

4.1.	Fehlerwerte der Lösungen für verschiedene verrauschte Messdaten in eindimensionalen instationären Fall . . . . .	52
4.2.	Untersuchung des Einflusses der Anzahl von Datenpunkten, 2D, instationär.	52
4.3.	Einfluss der Dimensionierung des PINNs in einem eindimensionalen stationären Fall . . . . .	53