HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

FAKULTÄT ELEKTROTECHNIK
UND INFORMATIK

Regelungsentwurf mittels maschinellen Lernens für ein Wärmeleitermodell

Abschlussarbeit
zur Erlangung des akademischen Grades
Bachelor of Engineering

im Studiengang
Elektrotechnik und Informationstechnik

vorgelegt von
Bettina Schull
Matrikelnummer 30996

im August 2021

Erstprüfer: Prof. Dr.-Ing. Lothar Berger
Zweitprüfer: Stephan Scholz M.Sc.

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit einem eindimensionalen Wärmeleitungsproblem: Das Ende eines Stabes wird auf eine vorgegebene Temperatur erwärmt, diesen Prozeß kontrolliert eine PI-Regelung. Es werden verschiedene Randbedingungen für das Stabende betrachtet. Anforderungen an die Regelung sowie deren Parametrisierung für die verschiedenen Randbedingungen werden diskutiert.
Es folgt die numerische Simulation des Vorgangs, wobei die Programmiersprache Julia [1] Anwendung findet. Hierzu wird die instationäre Wärmeleitungsgleichung zunächst manuell mit dem Finite-Differenzen-Verfahren FTCS (Abkürzung für engl.: Forward-Time Central-Space) numerisch gelöst und anschließend kommt die vertikale Linienmethode in Verbindung mit generischen Lösern zum Einsatz.
Es folgt der Vergleich des zuvor verwendeten expliziten Euler-Verfahrens mit ausgewählten Julia-Lösern. Ein Datensatz wird erstellt, um die Temperaturverläufe der verschiedenen Abschnitte des Stabes während der Erwärmung zu erfassen. Die so gewonnenen Temperaturverläufe dienen dazu, mithilfe von maschinellem Lernen die Reglerparameter zu finden, welche die PI-Regelung bei der Erwärmung des Stabes verwendet hat. Dabei werden verschiedene Optimierungsalgorithmen verglichen. Außerdem wird untersucht, wie viel Information, d.h. die Temperaturverläufe von wie vielen Ortspunkten, zum sicheren Auffinden der Parameter erforderlich sind.

Abstract

This thesis deals with a one dimensional heat conduction problem: the end of a rod is heated to a given temperature. This process is controlled by a PI controller. Different boundary conditions for the end of the rod are considered. Requirements for the control as well as the parametrization for the different boundary conditions are discussed.
Afterwards the process is simulated numerically using the Julia programming language. Therefore the instationary heat conduction equation is firstly solved manually using the numerical finite difference method FTCS (Forward-Time Central-Space) and after that the vertical line method in connection with generic solvers is used.
The previously used explicit Euler-method is compared with selected Julia solvers. A data set is created that contains the temperature distribution of the different sections of the rod during heating. After that the collected temperature distributions are used to identify through the application of machine learning the control parameters that the PI controller has used during heating. Different optimization algorithms are compared. In addition it is examined how much information, i.e. the temperature distribution of how many location points is needed to reliably identify the parameters.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Ziel dieser Arbeit ist es, mithilfe von maschinellem Lernen (engl.: Machine Learning, ML) anhand des Verlaufs der Temperaturverteilung in einem Stab feststellen zu können, welche Reglerparameter von einer PI-Temperaturregelung verwendet wurden, um diesen Stab zu erwärmen. Bekannt sind dabei die Materialeigenschaften des Stabes, die Umgebungstemperatur sowie Anfangstemperatur und Temperaturverlauf an verschiedenen Orten des Stabes.

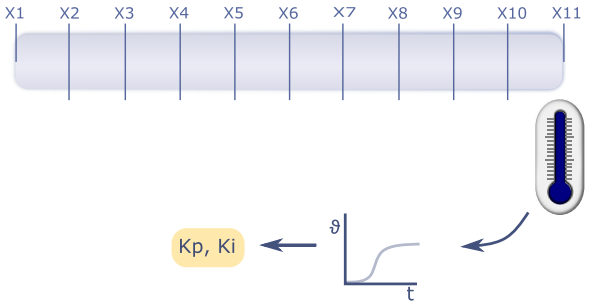


Abb. 1.1: Ablauf

Als Regelstrecke wurde ein Kupferstab der Länge l = 0.3 Meter angenommen. Aufgrund der im Vergleich zur Länge sehr geringen Dicke des Stabes, kann dieser als eindimensional betrachtet werden. Wie in Abb. 1.1 dargestellt soll dieser am linken Ende ($x_1, x = 0m$) erwärmt werden mit dem Ziel, das rechte Ende ($x_{11}, x = 0.3m$) auf eine vorgegebene Temperatur ϑ_{ref} zu heizen. Der Mantel des Stabes (x_2 bis x_{10}) wurde als thermisch perfekt isoliert betrachtet, innere Quellen oder Senken waren nicht vorhanden. Somit ist ein Wärmeübergang an die Umgebung lediglich an den Rändern x_1 und x_{11} möglich und die Wärme breitet sich ausschließlich in x-Richtung aus.
Im Rahmen dieser Arbeit wurden die Projekte PiSlider [33] und PI_OdeSolve_ML [32] in der Programmiersprache Julia [1] erstellt.

Die Temperaturverteilungen, anhand derer die Software den Proportional- und Integralanteil der Regelung erlernt, mussten zunächst durch Simulation erzeugt werden. Dazu wurde das System in einem ersten Schritt ohne Zuhilfenahme eines numerischen Lösers oder Julia-Paketes für Finite Differenzen oder ähnliche Methoden diskretisiert. Die örtlichen und zeitlichen Ableitungen der Wärmeleitungsgleichung wurden nach der FTCS-Methode (Forward-in-Time-Centered-in-Space) approximiert, wie in Abschnitt 4.1 beschrieben. Das so erhaltene lineare Gleichungssystem wurde im Projekt PiSlider verwendet, um verschiedene rechte Randbedingungen zu simulieren und jeweils passende Werte für die Regelparameter zu ermitteln. Folgende Randbedingungen (RB) wurden, wie in Abschnitt 2.2 diskutiert, für den rechten Grenzübergang untersucht:
- Neumann-RB (hohe angenommene thermischen Verluste)
- adiabate RB (thermische Isolierung) und

- natürliche RB (Wärmeübergang durch Strahlung, Konvektion und Konduktion).

Im nächsten Schritt wurde die vertikale Linienmethode (englisch: method of lines) verwendet (siehe z.B. [8, S. 151]) und das Projekt PI_OdeSolve_ML [32] erstellt. Hierbei wurde nur bezüglich der örtliche Variablen x mit der Finite-Differenzen-Methode diskretisiert (Semidiskretisierung, vgl Abschn. 4.2). Das so entstandene System gewöhnlicher Differentialgleichungen bezüglich der Zeit wurde einem generischen Löser übergeben. Im Unterschied zum FTCS-Verfahren bot diese Methode Flexibilität in der Wahl des Verfahrens für die Approximation der Zeitableitung. Beispielsweise kann durch adaptive Steuerung der Zeitschrittweite der Rechenaufwand reduziert werden. Der Euler-Solver (explizites Euler-Verfahren) wurde mit Verfahren mit adaptiver Zeitschrittweite verglichen.
Es wurde das Programm pi_solver_datagen im Projekt PI_OdeSolve_ML [32] entwickelt, um die Temperaturverläufe für alle Ortspunkte des Stabes zu berechnen. Dabei verwendet der Algorithmus die zuvor mithilfe von PiSlider gefundenen Reglerparameter.

Schließlich wurden die so gewonnenen Temperaturverläufe benutzt, um mithilfe von ML zu ermitteln, welche Reglerparameter die Temperaturregelung beim Erwärmen des Stabes verwendet hat. Hierbei wurden verschiedene Optimierungsverfahren hinsichtlich ihrer Geschwindigkeit und Genauigkeit verglichen. Außerdem wurde untersucht, wie viel Informationen (Anzahl der Ortspunkte, deren Temperaturverläufe verwendet werden) der ML-Algorithmus benötigt, um ein Ergebnis von hoher Genauigkeit zu erreichen.

Anmerkung zu den vorgenommenen Vereinfachungen

In dieser Arbeit wird davon ausgegangen, dass die Materialeigenschaften des Stabes in allen Stabsegmenten gleich sind und der Kupferstab auch bei großen Temperaturunterschieden seine Temperaturleitfähigkeit nicht verändert.

Weiterhin wird die zeitliche (linke Seite der Wärmeleitungsgleichung) oder die räumliche Ableitung (rechte Seite der Wärmeleitungsgleichung oder aber der Wärmefluss) gelegentlich separat betrachtet. Auch wenn hierbei nur nach einer Größe abgeleitet wird, soll die Notation für partielle Differentialgleichungen ($\partial\vartheta/\partial...$) Anwendung finden, da diese Teilbetrachtungen in den Kontext der Wärmeleitungsgleichung (WLG) einzuordnen sind.

Genauigkeit und Wertebereich aller berechneten Größen sind durch das verwendete Format für Dezimalzahlen (Float64) begrenzt.

2 Grundlagen und Rahmenbedingungen

2.1 Modell für das eindimensionale Wärmeleitungsproblem

Modelliert wird die Temperaturverteilung in einem eindimensionalen Kupferstab der Länge $l = 0.3m$, $\overline{\Omega} = [0, l]$, der über einen Zeitraum von $t_f = 2100s$ (35 Minuten) erwärmt wird. Der Wärmetransport im Stab wird durch die eindimensionale Wärmeleitungsgleichung (WLG) in ihrer homogenen, instationären Form beschrieben.

2.1.1 Wärmeleitungsgleichung

Die Wärmeleitungsgleichung ist eine partielle Differentialgleichung, mit der sich die Energiebilanz in einem Körper ($V \subseteq \mathbb{R}^3$) mit zeitlich veränderlichem ($T \in \mathbb{R}$) Temperaturfeld (instationäre Wärmeleitung) beschreiben lässt. In ihrer homogenen Form beschreibt sie das Temperaturfeld eines festen Körpers ohne innere Wärmequellen oder -senken

$$\rho\ c\ \frac{\partial\vartheta(\boldsymbol{s},t)}{\partial t} = -\nabla q(\boldsymbol{s},t) \qquad ,(\boldsymbol{s},t)\ \in\ V \times (0,T)\ .$$

Dabei ist ρ die Dichte in $[\frac{kg}{m^3}]$, c die spezifische Wärmekapazität in $[\frac{J}{kg\,K}]$ und q die Wärmeflussdichte in $[\frac{W}{m^2}]$. Siehe auch [20, S. 1ff]

Im dreidimensionalen Fall mit $\boldsymbol{s} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ ist der Gradient $\nabla q(s,t) = \begin{pmatrix} \frac{\partial q(s,t)}{\partial x} \\ \frac{\partial q(s,t)}{\partial y} \\ \frac{\partial q(s,t)}{\partial z} \end{pmatrix}$.

Nach dem Fourier'schen Gesetz

$$q = -\lambda \cdot \nabla\vartheta \tag{2.1}$$

kann die Wärmeflussdichte ersetzt werden. ϑ ist hierbei die Temperatur in [K], λ die Wärmeleitfähigkeit in $\left[\frac{W}{K\,m}\right]$.

Somit wird die Divergenz des Gradienten gebildet entsprechend

$$\nabla \cdot \Big(\nabla\vartheta(\boldsymbol{s},t)\Big) = div(\nabla\vartheta(\boldsymbol{s},t) = \frac{\partial^2\vartheta(\boldsymbol{s},t)}{\partial x^2} + \frac{\partial^2\vartheta(\boldsymbol{s},t)}{\partial y^2} + \frac{\partial^2\vartheta(\boldsymbol{s},t)}{\partial z^2} = \Delta\vartheta(\boldsymbol{s},t)$$

und man erhält

$$\rho c\frac{\partial\vartheta(\boldsymbol{s},t)}{\partial t} = \lambda \cdot \Delta\vartheta(\boldsymbol{s},t) \tag{2.2}$$

mit dem Laplaceoperator Δ.

Während die rechte Seite der WLG die Änderung des Wärmeflusses in Abhängigkeit vom Ort beschreibt, drückt die linke Seite die Änderung der inneren Energie über die Zeit aus.

Die Wärmeenergie, welche benötigt wird um einen Körper von einer Temperatur ϑ_1 auf die Temperatur ϑ_2 zu erwärmen, wird beschrieben durch

$$Q = mc\,\Delta\vartheta \quad \text{mit } \Delta\vartheta = (\vartheta_2 - \vartheta_1)$$

Drückt man die Masse m des Stababschnitts durch sein Volumen V und seine Dichte ρ aus, wobei das Volumen wiederum durch die Fläche A und den betrachteten Streckenabschnitt Δx beschrieben werden kann, ergibt sich für den betrachteten Abschnitt

$$Q_n = \overbrace{\underbrace{A\,\Delta x}_{V}\,\rho}^{m}\, c\,\Delta\vartheta \quad \text{mit der Differenz } \Delta x = (x_{out} - x_{in})\ .$$

Somit kann die Änderung der inneren Energie pro Volumeneinheit im Stab ausgedrückt werden durch

$$\frac{\partial Q_n}{\partial t} = \rho c \frac{\partial \vartheta}{\partial t}\ . \tag{2.3}$$

Mit der Wärmemenge Q_n ist dabei der Nettowärmestrom gemeint, der dem betrachteten Stababschnitt insgesamt zu- oder abgeführt wurde [11]

$$\frac{\partial Q_n}{\partial t} = \frac{\partial Q_{in}}{\partial t} - \frac{\partial Q_{out}}{\partial t}\ .$$

Fließt also mehr Wärme in ein Element (auch der Stab als Ganzes kann als solches betrachtet werden) hinein als am anderen Ende herausströmt, erhöht sich dessen thermische Energie und er erwärmt sich entsprechend (2.3). Dann liegt der instationäre Fall vor.
Ist dagegen die Solltemperatur erreicht, darf am linken Ende nur noch so viel Energie zugeführt werden, wie am rechten Ende abgegeben wird, damit die Temperatur im Stab konstant bleibt, d.h. im stationären Fall gilt

$$\frac{\partial Q}{\partial t} = \frac{\partial \vartheta}{\partial t} = 0.$$

In dieser Arbeit wird die instationäre Form der WLG entsprechend (2.2) untersucht. Bei eindimensionaler Betrachtung vereinfacht sich der Laplace-Operator zur zweiten Ableitung in x-Richtung und die Änderungsrate der Temperatur kann wie folgt berechnet werden.

$$\begin{aligned}
&\frac{\partial \vartheta(t,x)}{\partial t} &&= \alpha \cdot \frac{\partial^2 \vartheta(t,x)}{\partial x^2} && \text{für } (t,x) \in (0,tf) \times \Omega \\
&\text{mit Anfangsbedingung} \quad \vartheta(0,x) &&= \vartheta_0(x) && \text{für } x \in \overline{\Omega} \\
&\text{und den Randbedingungen} \quad \lambda \frac{\partial \vartheta(t,x)}{\partial x}\,\vec{n} &&= \begin{cases} q_{in}(t,x) & \text{für } x = 0 \\ q_{end}(t,x) & \text{für } x = L \end{cases}
\end{aligned}$$

Dabei ist
$\alpha = \frac{\lambda}{\rho c}$ in $\left[\frac{m^2}{s}\right]$ die Temperaturleitfähigkeit und
$\vec{n}$ der Normaleneinheitsvektor auf dem jeweiligen Rand bei $x = 0$ bzw. $x = l$.

Eine detaillierte Beschreibung der betrachteten Randbedingungen findet sich in Abschnitt 2.2 .

Diskretisierung

Der Stab ist durch 11 Ortspunkte ($nx = 11$, x_1 bis x_{11}) in 10 Segmente ($(nx - 1) = 10$) unterteilt.

Die rechte Seite der WLG (zweite Ableitung nach der Variablen x) wurde mithilfe von Finiten Differenzen approximiert durch den zentralen Differenzenquotienten zweiter Ordnung (vgl. Abschn. 4.1.1).
Zur Diskretisierung der linken Seite (erste Ableitung nach der Variable t) wurde
- im Projekt PiSlider [33] zunächst das explizite Euler-Verfahren (vgl. Abschn. 4.1.3) und
- im Projekt PI_OdeSolve_ML [32] verschiedene generische Solver (vgl. Abschn. 4.2)

verwendet.
Das Simulationsgebiet zeigt Abb. 2.1.

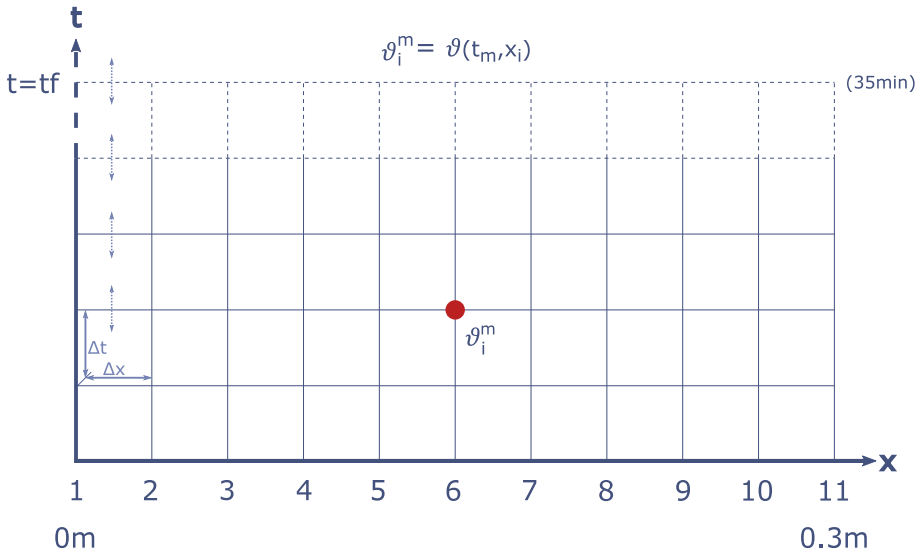


Abb. 2.1: Simulationsgebiet

Anfangsbedingung

Zu Beginn des Heizvorgangs, d.h. zum Zeitpunkt t_0, soll der gesamte Stab Umgebungstemperatur (ϑ_{amb}) haben. Diese entspricht annähernd Raumtemperatur:

$$\vartheta(0, x) = \vartheta_{amb} = 293K \text{ , d.h. } \vartheta_i^1 = 293K \text{ für } i = 1...11.$$

2.2 Randbedingungen

Wie bereits festgestellt, soll sich die Wärme nur entlang der x-Achse ausbreiten, d.h. ein Energieaustausch zwischen Stab und Umgebung kann nur an den Rändern des Stabes (x_1 und x_{11}) stattfinden. Es gibt außerdem keine inneren Quellen oder Senken.

Unterscheidet sich die Umgebungstemperatur von der des Stabes, kann Wärme über die Ränder hinein- oder herausströmen. Eine solche Wärmeübertragung ist quantitativ durch die Wärmeflussdichte q beschrieben. Diese wird definiert als die über den Rand übertragene

thermische Energie, bezogen auf die Randfläche und die Zeit (sie wird daher in der Literatur auch häufig mit $\dot{q}$ bezeichnet). Die Wärmeflussdichte wird, abgeleitet vom Fourier'schen Wärmeleitungsgesetz (Formel 2.1), berechnet nach

$$q_n = \lambda \cdot \nabla\vartheta\big|_{x_n} \cdot \vec{n}\,.$$

Im eindimensionalen Fall wird der Gradient jedoch zu einer partiellen Ableitung nach x

$$q_n = \lambda \cdot \frac{\partial\vartheta}{\partial x}\Big|_{x_n} \cdot \vec{n} \qquad \text{in } \left[\frac{W}{m^2}\right]\,. \tag{2.4}$$

Dabei ist

- λ die Wärmeleitfähigkeit des Stabes in $\left[\frac{W}{K\,m}\right]$,
- $\frac{\partial\vartheta}{\partial x}\Big|_{x_n}$ die Änderung der Temperatur in Abhängigkeit vom Ort an der Stelle x_n und
- $\vec{n}$ der nach außen weisende Normalenvektor der Randfläche (Einheitsnormale, l=1).

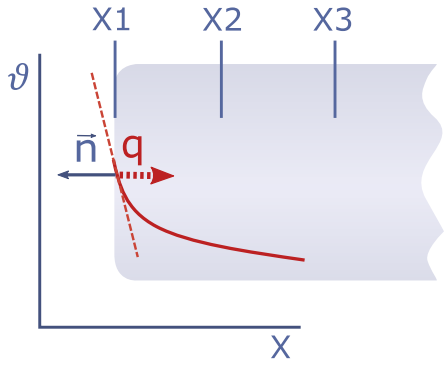


Abb. 2.2: Randbedingungen

Alle Randbedingungen (RB) werden über die Änderungsrate der Temperatur bzw. die Flussdichte durch den Rand definiert, Dirichlet-RB werden nicht betrachtet (siehe Abb. 2.2).

2.2.1 Linke Seite: Neumann-Randbedingung

Bei allen verwendeten Szenarien findet ein Wärmefluss in das System über den linken Rand statt. Mithilfe der so zugeführten Wärmemenge wird die Temperatur am rechten Stabende geregelt.
Im Falle einer Neumann-RB ist der Gradient einer Größe auf dem Rand gegeben, bei eindimensionaler Betrachtung also die partielle Ableitung nach x. Im hier untersuchten Fall ist die Wärmeflussdichte durch das linke Stabende bekannt. Da der Wärmefluss eine zum Normaleneinheitsvektor entgegengesetzte Orientierung hat, erhält sie ein negatives Vorzeichen.

$$q_1(t) = -\lambda \frac{\partial\vartheta}{\partial x}\Big|_{x=0} \tag{2.5}$$

Mithilfe der PI-Regelung wird die Regeldifferenz [K] vom rechten Stabende auf die Wärmeflussdichte $\left[\frac{W}{m^2}\right]$ am linken Rand abgebildet (vgl. Abschnitt 3.2).

$$\begin{aligned} q_1(t) &= u(t) \cdot \left[\tfrac{W}{m^2 \cdot K}\right] \\ \text{mit} \quad u(t) &= max\Big(\Big(\tilde{u}_p(t) + \tilde{u}_i(t)\Big), 0\Big) \end{aligned} \tag{2.6}$$

Für das Stabende (rechter Rand) wurden 3 Szenarien betrachtet.

2.2.2 Rechte Seite, Szenario 1: Adiabate Randbedingung (homogene Neumann-RB)

In diesem Fall kann nur über den linken Rand thermische Energie übertragen werden, der rechte Rand wird – wie die Seiten des Stabes – als thermisch perfekt isoliert betrachtet.

$$q|_{x=l} = 0$$

Abb. 3.1 (oberer Graph) zeigt das mit PI_Slider ermittelte Verhalten des Systems. Es handelt sich um eine integrierende Regelstrecke. Die Wärmeenergie kann sich im Stab verteilen, jedoch nicht in die umgebende Luft abströmen.

2.2.3 Rechte Seite, Szenario 2: Natürliche Randbedingung

Hier ist eine Wärmeübertragung über den rechten Rand möglich. Wird das Stabende auf eine höhere Temperatur erwärmt als die umgebende Luft, kann Wärme in die Umgebung abströmen. Für diesen Wärmetransport sind drei Mechanismen verantwortlich:

Konduktion und Konvektion

Konduktion (Wärmeleitung): Wärmeleitung findet im Inneren eines Körpers (Wärmedurchgang) oder zwischen zwei angrenzenden Körpern (Wärmeübergang) statt. Sie beschreibt den Übergang der kinetischen Energie von Partikeln von Bereichen höherer zu Bereichen niedrigerer Temperatur. [4]
Zwischen Stabende und Umgebung entsteht eine dünne laminare Grenzschicht mit einer stagnierenden Luftschicht direkt am Stabende. Durch diesen „stehenden Film“ wird thermische Energie durch Konduktion transportiert. (siehe [34])

Als **Konvektion (Wärmeströmung)** wird der Wärmetransport durch ein strömendes Fluid (hier: Luft) bezeichnet. Eine erzwungene Konvektion (z.B. durch Druckunterschiede, Wind) ist in unserem Szenario nicht vorhanden. Durch den Temperaturunterschied entsteht jedoch in den zum stehenden Film angrenzenden Luftschichten Auftrieb und Wärme strömt vom System weg.

Die Effekte von Konduktion und Konvektion in der angrenzenden laminaren Luftschicht werden zusammengefasst und anhand des linearen Newton'schen Gesetzes der Wärmeleitung wird die Wärmeflussdichte wie folgt berechnet (Robin-Randbedingung)

$$q_{cc}(t) = h\Big(\vartheta(t,x) - \vartheta_{amb}\Big)\Big|_{x=l} \tag{2.7}$$

mit q_{cc} durch Konduktion und Konvektion erzeugte Wärmeflussdichte über den rechten Rand (Fluss hier in Richtung der Flächennormalen),
h gemeinsamer Wärmeüberganskoeffizient für Konduktion und Konvektion, verwendet wurde $h = 10\frac{W}{m^2 \cdot K}$ (Quelle: RWU, Stephan Scholz),
ϑ_{end} Temperatur des Stabendes und
ϑ_{amb} Umgebungstemperatur in einiger Entfernung vom Stab .

Wärmestrahlung

Das erwärmte Stabende emittiert elektromagnetische Wellen an die kühlere Umgebung (Wärmeübertragung ohne Stofftransport). Sie wird unter Verwendung des Stefan-Boltzmann-Gesetzes wie folgt berechnet:

$$q_r(t) = k_r\Big(\vartheta(t,x)^{\,4} - {\vartheta_{amb}}^4\Big)\Big|_{x=l} \tag{2.8}$$

mit q_r $\left[\frac{W}{m^2}\right]$ durch Strahlung erzeugte Wärmeflussdichte,
k_r Strahlungskoeffizient, der sich mit $k_r = \epsilon \cdot \sigma$ errechnet aus:
$\epsilon = 0.78$ Emissionsgrad Kupfer (stark oxidiert bei 20°C , Quelle: [40]) und
$\sigma = 5.67 \cdot 10^{-8}$ Stefan-Boltzmann Konstante $\left[\frac{W}{m^2 \cdot K^4}\right]$

Für die Wärmeflussdichte durch das rechte Stabende ergibt sich somit für die natürliche Randbedingung:

$$q_{end}(t) = q_{cc}(t) + q_r(t)$$

Wie Abb. 3.2 zu entnehmen ist, zeigt auch diese Strecke integrierendes Verhalten. Die abgegebene Wärmemenge ist gegenüber der zugeführten Energie gering.

2.2.4 Rechte Seite, Szenario 3: Neumann-Randbedingung

In Abschn. 2.2.3 sind die Mechanismen berücksichtigt, die unter realen Bedingungen für einen Wärmetransport vom Stab an die Umgebung infrage kommen. Unter realen Bedingungen würde jedoch wesentlich mehr Energie abfließen, da auch über die Mantelfläche Wärmeenergie abgegeben würde. Daher soll ein Szenario konstruiert werden, bei dem sich das simulierte System wie ein reales System mit Ausgleich (PT_n-Glied) verhält. Am rechten Rand muss also ein wesentlich höherer Wärmeverlust entstehen.
Die Wärmeflussdichte am rechten Grenzübergang kann, wie an jedem anderen Punkt des Stabes auch, mithilfe des Fourier'schen Wärmeleitungsgesetzes (Formel 2.1) beschrieben werden. Angewendet auf den rechten Rand und im eindimensionalen Fall heißt das

$$q_{end}(t) = \lambda \frac{\partial \vartheta(t)}{\partial x}\Big|_{x=l} \ , \tag{2.9}$$

wobei das positive Vorzeichen der gleichen Orientierung von Wärmefluss und Normalenvektor auf dem rechten Rand geschuldet ist.

Tatsächlich bildet sich zwischen Stab und umgebender Luft eine Grenzschicht mit einem räumlich kontinuierlichen Wärmeübergang heraus. Die Änderungsrate der Temperatur an der Stelle $x = l$ ist dabei unbekannt. Bekannt ist lediglich die Temperatur des Stabendes und die Umgebungstemperatur, die theoretisch in unendlich großer Entfernung vom Stab erreicht wird, ϑ_{amb}. Für die Berechnungen der natürlichen Randbedingung werden diese Ausgleichsvorgänge zwischen Stab und umgebendem Medium durch Koeffizienten ausgedrückt. Diese sind von den Medien abhängig, welche die Grenze bilden. Dazu gehören k_r für die Strahlung in Gleichung (2.8) und h für Konduktion und Konvektion in Gleichung (2.7).
Im Falle der Neumann-RB wird dagegen direkt die Änderungsrate der Temperatur an der Stelle $x = l$ vorgegeben. Da diese möglichst groß sein sollte, wurde die volle Temperaturdifferenz des Stabendes zur Umgebung auf ein Δx bezogen. Dies entspricht nicht der physikalischen Realität, stellt jedoch sicher, dass sich am rechten Stabende ein stabiler Zustand einstellt. Die Wärmeflussdichte am rechten Rand berechnet sich dann durch

$$q_{end}(t) = \lambda \left. \frac{\vartheta(t,x) - \vartheta_{amb}}{\Delta x} \right|_{x=l} \quad . \tag{2.10}$$

Wie in Abb. 3.3 zu erkennen ist, zeigt diese Regelstrecke das angestrebte Verhalten eines PT_n-Gliedes.

In Abb. 2.3 ist für jede der drei Randbedingungen für das rechte Stabende jeweils der Temperaturverlauf aller elf Ortspunkte während der Erwärmung des Stabes als Heatmap dargestellt.

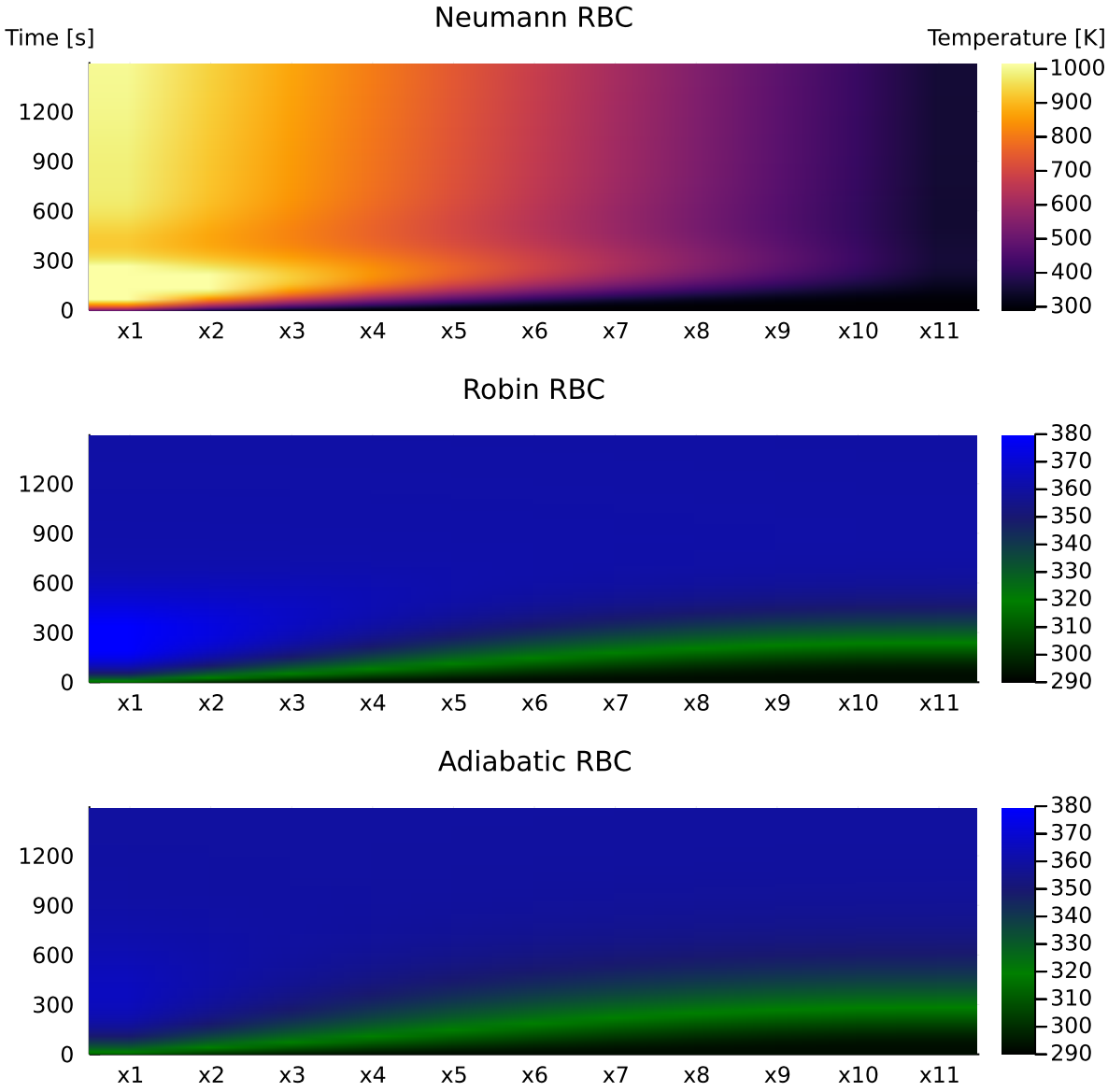


Abb. 2.3: Heatmaps für die verschiedenen rechten RB

3 Regelung

Im konkreten, in dieser Arbeit simulierten Fall, war das Ziel die Regelung des rechten Stabendes von einer

Anfangstemperatur	$\vartheta(0,x) = 293K$	auf eine
Solltemperatur	$\vartheta_{ref}(t) = 360K$	.

Verwendet wurde ein einschleifiger Regler mit Ausgangsrückführung mit

Regelgröße	$y(t) = \vartheta(t,x)$	bei $x = l$	und
Stellgröße	$u(t) = q(t,x)$	bei $x = 0$	.

Es wurde eine Festwertregelung (konstanter Sollwert ϑ_{ref}) simuliert. Geregelt wird kontinuierlich, die Stellgroße kann jeden Zwischenwert annehmen, soweit Genauigkeit und Wertebereich des verwendeten Formats für Dezimalzahlen (Float64) dies zulassen.
Der verwendete Algorithmus ist der einer PI-Regelung (siehe Abschn. 3.2). Da aufgrund der Trägheit des Systems die Änderungsrate der Regeldifferenz sehr gering ist, war ein Differentialanteil nicht sinnvoll. Der Algorithmus des PI-Reglers wird in Abschnitt 3.2 erläutert.

3.1 Anforderungen

Bei der Wahl der Regelparameter wurde nicht die optimale Dimensionierung des Reglers angestrebt, es sollte jedoch ein realistischer Verlauf für den Regelungsvorgang erzeugt werden. Folgende Anforderungen wurden an die Regelung gestellt:

Geschwindigkeit und Genauigkeit

Innerhalb von 20min ($T_1 = 1200s$) soll sich am Stabende eine Abweichung <5% vom Referenzwert eingestellt haben, d.h.

$$\lim_{t \to T_1} \left| \frac{\vartheta_{ref}(t) - y(t)}{\vartheta_{ref}(t)} \right| < 0.05 = 5\% \qquad \text{und} \tag{3.1}$$

$$\lim_{t \to T_1} \left| \vartheta_{ref}(t) - y(t) \right| < 3.35K \qquad . \tag{3.2}$$

Im Fall der natürlichen RB wird dies bereits nach weniger als 10 Minuten und bei der adiabaten RB nach knapp 15 Minuten erreicht. Obwohl das System mit natürlicher RB am rechten Ende Wärmeenergie abgibt, wird der stabile Zustand also schneller erreicht als bei der adiabaten RB. Das liegt daran, dass die Regelung hier mit einem höheren Proportionalbeiwert arbeiten kann.

Am Ende des Simulationszeitraumes (tf=2100s) soll die Abweichung vom Referenzwert <2% sein

$$\lim_{t \to tf} \left| \frac{\vartheta_{ref}(t) - y(t)}{\vartheta_{ref}(t)} \right| < 0.02 = 2\% \qquad \text{und} \tag{3.3}$$

$$\lim_{t \to tf} |\vartheta_{ref}(t) - y(t)| < 1.34K \qquad . \tag{3.4}$$

Ein dauerhaftes Überschreiten des Referenzwertes aufgrund von Integration wird nicht toleriert.

3.2 Regelalgorithmus

Die Temperatur am rechten Stabende aus der Simulation des Prozesses dient als Regelgröße. Aus der Differenz zum Sollwert, der Regeldifferenz

$$\Big(\vartheta_{ref}(t) - \vartheta(t,l)\Big)$$

wird der Wärmefluss berechnet, der am linken Rand in den Stab hineinströmt. Wie in Abschnitt 3 festgestellt, wurde das Modell eines PI-Reglers verwendet mit

Proportionalanteil	$\tilde{u}_p(t) = K_p\Big(\vartheta_{ref}(t) - \vartheta(t,l)\Big)$	und
Integralanteil	$\tilde{u}_i(t) = K_i \int_0^\tau \Big(\vartheta_{ref}(t) - \vartheta(t,l)\Big) d\tau$	mit $K_i = \frac{K_p}{T_n}$.

Somit ergibt sich als Zwischenergebnis für die Stellgröße

$$\tilde{u}(t) = \tilde{u}_p(t) + \tilde{u}_i(t) \ .$$

Da als Stellglied ein Heizelement angenommen wird, welches den Stab erwärmen, nicht aber kühlen kann, darf die Stellgröße nicht negativ werden. Sie wird daher wie folgt korrigiert:

$$u(t) = max(\tilde{u}(t), 0) \ .$$

Dabei ist

K_p	Proportionalbeiwert	
K_i	Integrierbeiwert	
T_n	Nachstellzeit	
$\vartheta(t,l)$	simulierte Temperatur am rechten Rand zum aktuellen Zeitpunkt,	Ist-Wert
$\vartheta_{ref}(t)$	vorgegebener Referenzwert für das Stabende (hier: konstant)	Soll-Wert
$\tilde{u}(t)$	Zwischenergebnis,	Stellgröße
$u(t)$	Endergebnis, Ausgangsgröße des Reglers,	Stellgröße

Durch die Diskretisierung des Systems wird bei der Berechnung des K_i-Anteils das Integral zu einer Summe. Bei einem Abtastintervall von $T_{samp} > \Delta t$ ergibt sich

$$\tilde{u}_i(t_m) = Ki \sum_{k=1}^{n} (\vartheta_{end}^{t_k} - \vartheta_{ref}^{t_k}) \cdot T_{samp} \quad \text{mit} \quad t_k = k \cdot T_{samp} \ . \tag{3.5}$$

Abtastrate: Aufgrund der Diskretisierung mit 11 Ortspunkten in x-Richtung (10 Stababschnitte) kann das Steuersignal frühestens nach 11 Zeitintervallen Δt einen Effekt auf die Regelgröße haben.
Zunächst wurde daher eine Samplezeit von $T_{samp} = (nx) \cdot \Delta t$ für die Berechnung des erforderlichen Wärmestroms durch den linken Rand implementiert sowie eine Ansteuerung mit Impulsen. Dafür waren zusätzliche Funktionen notwendig sowie ein Zähler bzw. – bei Verwendung eines generischen Lösers - ein *discretecallback*.
Es stellte sich jedoch heraus, dass die Temperaturänderung am rechten Ende so langsam erfolgt, dass selbst bei integrierender Regelstrecke (adiabate und natürliche RB) eine zeitkontinuierliche Ansteuerung und eine Berechnung des Steuersignals in jedem Zeitintervall keine Nachteile hat.
Im Programm Pi_Slider wird die Samplezeit verwendet, um die Messintervalle eines Sensors abzubilden und das Modell realistischer zu gestalten. Die Regelung liefert allerdings im Unterschied zum ersten Ansatz ein zeitkontinuierliches Ausgangssignal (halten des berechneten Ausgangswertes für die Dauer von T_{samp}).
Im Projekt PI_OdeSolv_ML wird dagegen für jeden Zeitschritt Δt auch die Stellgröße berechnet. Das vereinfacht nicht nur den Programmfluss, sondern ermöglicht es auch, die Regeldifferenz als Ableitung des Fehlerintegrals in das Differentialgleichungssystem einzubeziehen (siehe Abschnitt 4.2).

Eine weitere Besonderheit des Projekts Pi_Slider ist, dass hier die Steuergröße begrenzt wurde und bei Erreichen dieses Maximums die Integration des Fehlers gestoppt wird (ähnlich Clamping). In allen folgenden Projekten wurde im Unterschied dazu vereinfachend angenommen, dass das Heizelement am linken Ende eine unbegrenzte Leistung liefern kann.

3.3 Reglerparametrierung für die verschiedenen Randbedingungen

Die Parameter wurden mit dem Programm PiSlider.jl (Bestandteil des Projekts Pi_Slider) iterativ anhand der grafischen Darstellung des Regelungsvorgangs und der Wertetabellen ermittelt. Dabei haben folgende Überlegungen zum Verhalten des geschlossenen Regelkreises eine Rolle gespielt:

3.3.1 Vorüberlegungen

Zur adiabatischen und natürlichen RB

Die Wärme kann sich im Stab verteilen, jedoch nicht (adiabatische) bzw. nur in sehr geringem Maß (natürliche RB) an die Umgebung abfließen. Ein sehr schneller Anstieg mit Überschwingen der Temperatur am rechten Stabende kann nicht toleriert werden, da dies eine dauerhafte Überschreitung des Sollwertes zur Folge hätte. Diese Systeme werden daher im

Vergleich zur Neumann-RB mit einem deutlich geringeren Proportionalbeiwert angesteuert. Das Verhalten beider Systeme zeigen die Abbildungen 3.1 und 3.2 anhand von Sprungantwort und Regelungsvorgang.

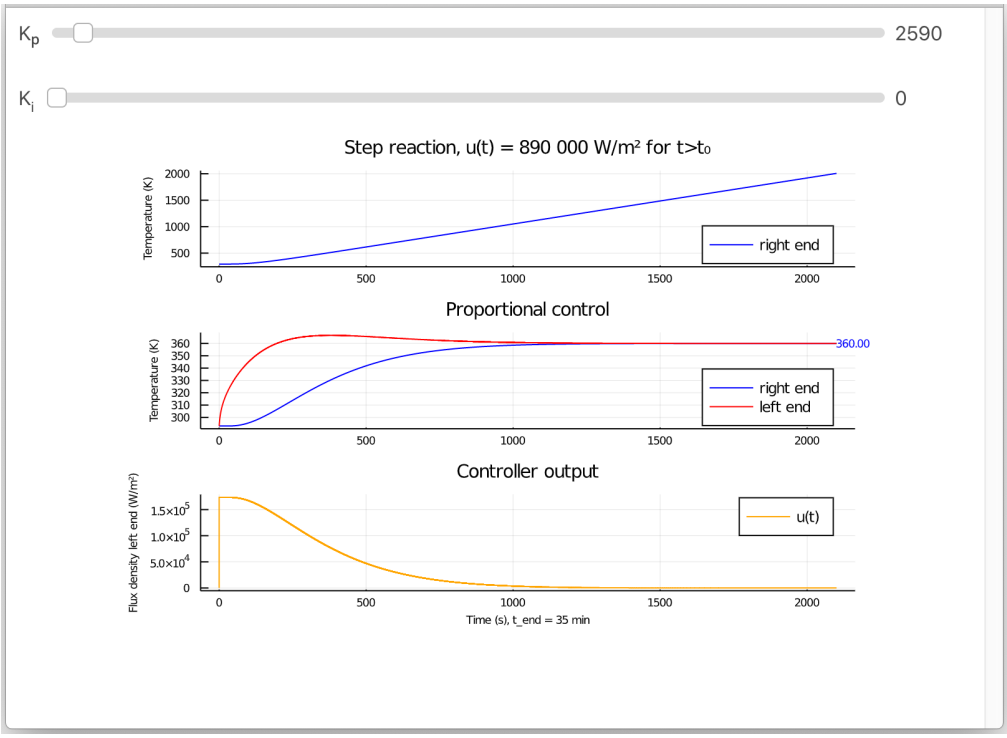


Abb. 3.1: Adiabate RB, erstellt mit PiSlider

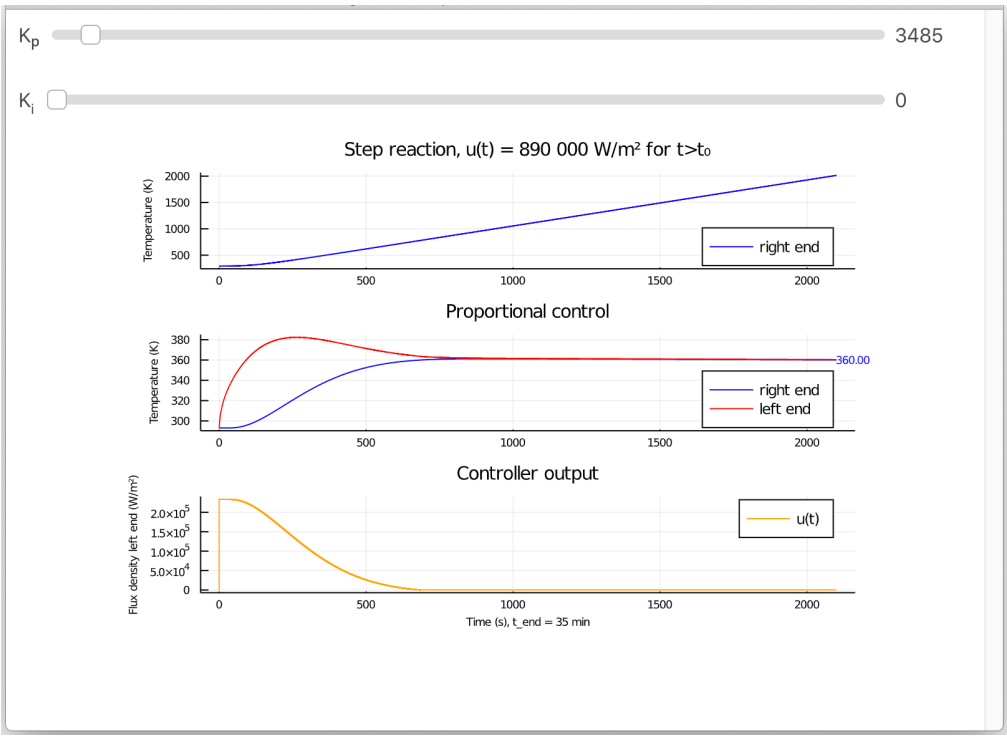


Abb. 3.2: Natürliche RB, erstellt mit PiSlider

Im Falle einer integrierenden Regelstrecke muss außerdem der Integralanteil deaktiviert sein ($K_i = 0$). Hier tritt auch bei einem reinen P-Regler keine bleibende Regelabweichung auf, der I-Anteil ist also verzichtbar. Im Gegenteil, solange die Temperatur des Stabendes niedriger als die Zieltemperatur ist, würde der Fehler integriert, d.h. der Integralanteil steigt und trägt zum Wärmestrom bei. Mit Erreichen der Zieltemperatur verschwindet der proportionale Anteil, der integrale würde jedoch weiterhin einen Wärmestrom erzeugen. Erst ein Überschreiten der Zieltemperatur würde zu einer negativen Regeldifferenz führen und den I-Anteil wieder verringern. Da jedoch am rechten Rand keine bzw. nur sehr wenig Wärme abfließt, kann das Stabende nicht oder nur extrem langsam abkühlen und die Zieltemperatur würde wiederum langandauernd überschritten.

Zur Neumann-RB

Hier handelt es sich um ein System mit Ausgleich. Aufgrund der hohen Wärmeverluste am rechten Grenzübergang werden am linken Stabende im Vergleich mit den anderen RB die größten Flussdichten benötigt, wie auch Abb. 3.3 zu entnehmen ist.

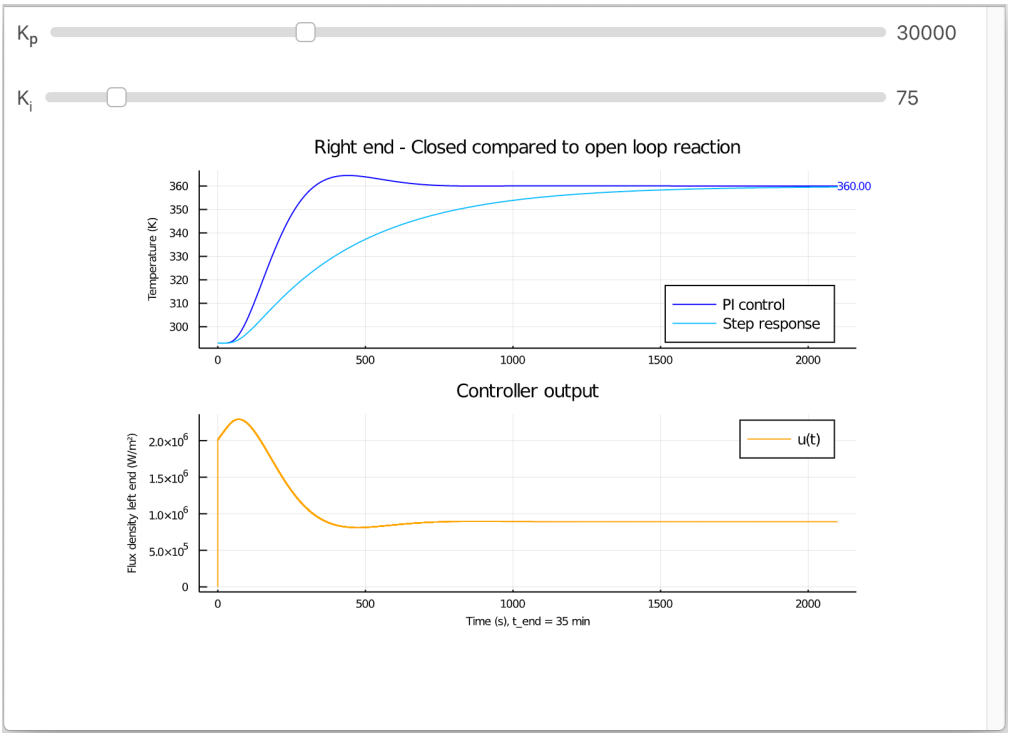


Abb. 3.3: Neumann-RB, erstellt mit PiSlider

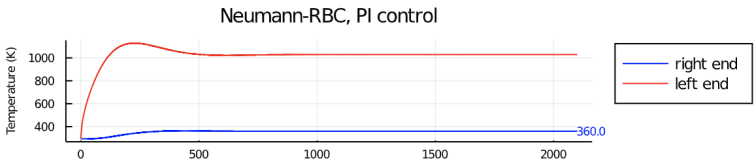


Abb. 3.4: Linkes/rechtes Stabende bei Neumann-RB

Hier wird ein sehr viel höherer Proportionalbeiwert verwendet. Ein mäßiges Überschwingen der Regelgröße kann toleriert werden. Es gleicht sich vor Erreichen des stationären Endwertes wieder aus. Ein I-Anteil wird benötigt, um die bleibende Regelabweichung des P-Reglers zu korrigieren. Nach Erreichen des Sollwertes muss eine konstante, vergleichsweise große Energiemenge zugeführt werden um zu verhindern, dass sich der Stab wieder abkühlt.
Wie Abb. 3.4 zeigt, wurden in der Simulation am linken Stabende tatsächlich Temperaturen von über 1000 Kelvin erreicht. Unter diesen Bedingungen würde dieser Teil des Stabes bereits glühen. Der Einfachheit halber wurde dies bei der Reglerauslegung jedoch ignoriert.

3.3.2 Regelparameter für die jeweiligen RB

Folgende Werte wurden für die drei RB des rechten Stabendes ermittelt, um die oben genannten Anforderungen zu erfüllen:

	AdiabatischeRB	Natürliche RB	Neumann-RB
K_p	2 590	3 485	40 000
K_i	0	0	50

Tab. 3.1: Reglerparameter

4 Diskretisierung der instationären Wärmeleitungsgleichung

4.1 Diskretisierung nach dem Forward-Time-Centered-Space-Verfahren (FTCS)

Das FTCS-Verfahren ist ein Finite-Differenzen-Verfahren. Der eindimensionale Stab wird dabei durch endlich viele äquidistante Punkte in x-Richtung angenähert und der zeitliche Verlauf durch endlich viele äquidistante Zeitpunkte (s. Abb. 4.1). Die partiellen Ableitungen werden durch Differenzenquotienten approximiert.
Die Temperaturwerte für alle Ortspunkte werden dabei aus den bekannten Werten zum vorangegangenen Zeitpunkt ermittelt (explizites Verfahren). Siehe [5]

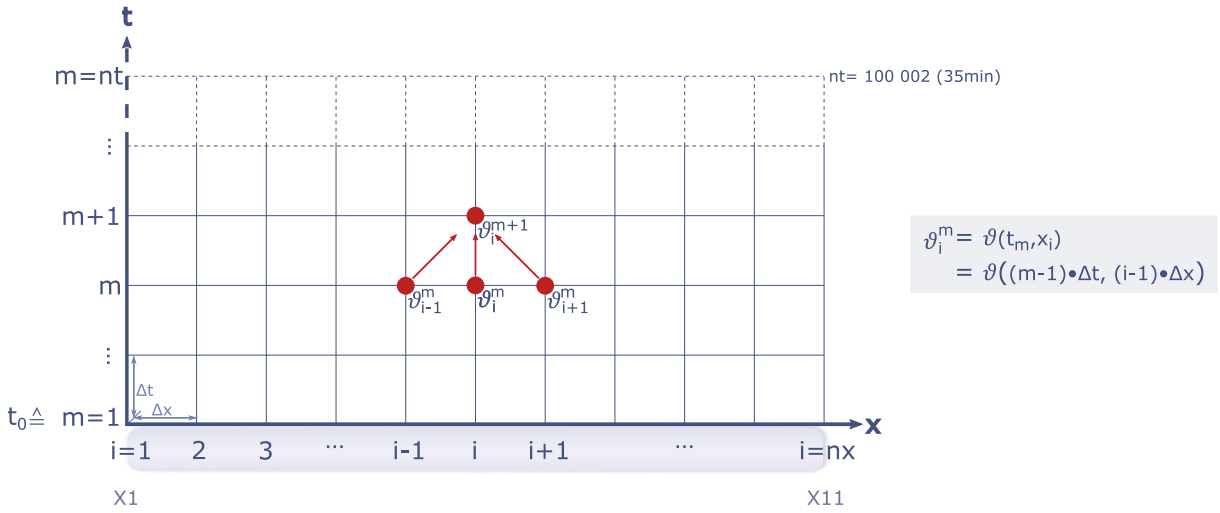


Abb. 4.1: Diskretisierung

Dabei ist $\vartheta_i^m = \vartheta(t_m, x_i)$ mit $x_i = (i-1) \cdot \Delta x$ und $t_m = (m-1) \cdot \Delta t$, wobei das Δ für die Schrittweite steht.
Es wird $(m-1)$ verwendet, weil aufgrund der eins-basierten Array-Indizierung bei Julia auch die Ortspunkte nicht bei x_0, sondern bei x_1 und Simulationszeit bei t_1 beginnt. Somit ist beispielsweise nach einem Δx der Punkt x_2 erreicht.

4.1.1 Räumliche Diskretisierung

Wie eingangs festgestellt, wird die zweite Ortsableitung in der rechten Seite der WLG durch finite Differenzen approxomiert, d.h. mithilfe einer endlichen Anzahl von gleichmäßig entlang der x-Achse angeordneten Punkten.

Betrachtet werden zunächst die drei benachbarten Punkte ϑ_{i-1}^m , ϑ_i^m und ϑ_{i+1}^m, die jeweils im Abstand von Δx angeordnet sind. Für eine analytische Funktion lässt sich der Funktionswert an einer benachbarten Stelle mithilfe der Taylorreihenentwicklung an der Stelle x_i darstellen durch

$$\begin{aligned} f(x_{i+1}) &= f(x_i + \Delta x) \\ &= f(x_i) + \sum_{n=1}^{\infty} \frac{(\Delta x)^n}{n!} \frac{\partial^n f(x_i)}{\partial x^n} \qquad (4.1) \\ &= f(x_i) + \Delta x \frac{\partial f(x_i)}{\partial x} + \frac{(\Delta x)^2}{2!} \frac{\partial^2 f(x_i)}{\partial x^2} + \frac{(\Delta x)^3}{3!} \frac{\partial^3 f(x_i)}{\partial x^3} + \mathcal{O}(\Delta x)^4, \qquad (4.2) \end{aligned}$$

nachzulesen beispielsweise bei [28, S. 4ff.]). Analog erhält man für die rückwärtige Taylorreihenentwicklung

$$\begin{aligned} f(x_{i-1}) &= f\Big(x_i + (-\Delta x)\Big) \\ &= f(x_i) + \sum_{n=1}^{\infty} \Big[\pm \frac{(\Delta x)^n}{n!} \Big] \frac{\partial^n f(x_i)}{\partial x^n} \quad \text{mit} \begin{cases} + \text{ für gerade } n \\ - \text{ für ungerade } n \end{cases} \\ &= f(x_i) - \Delta x \frac{\partial f(x_i)}{\partial x} + \frac{(\Delta x)^2}{2!} \frac{\partial^2 f(x_i)}{\partial x^2} - \frac{(\Delta x)^3}{3!} \frac{\partial^3 f(x_i)}{\partial x^3} + \mathcal{O}(\Delta x)^4. \qquad (4.3) \end{aligned}$$

Durch $\mathcal{O}(\Delta x)^4$ wird der durch das Abschneiden nach dem dritten Glied entstandene Fehler durch den für x < 1 führenden Fehlerterm bezeichnet [6]. Das Polynom 3. Grades wurde verwendet, da dann im nächsten Schritt sichtbar wird, dass das 3. Glied - wie auch die nachfolgenden Glieder mit ungeraden Exponenten - verschwindet und nicht zum Fehler beiträgt. Addiert man nun Gleichung 4.2 und 4.3 zu

$$f(x_{i+1}) + f(x_{i-1}) = 2f(x_i) + \frac{(\Delta x)^2 \, \partial^2 f(x_i)}{\partial x^2} + \mathcal{O}(\Delta x)^4$$

und löst nach $\frac{\partial^2 f(x_i)}{x^2}$ auf, ergibt sich

$$\frac{\partial^2 f(x_i)}{x^2} = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{(\Delta x)^2} + \mathcal{O}(\Delta x^2) \; .$$

Das Verfahren erhält die Konsistenzordnung 2 (vgl. auch [10, S. 77]) .
Entsprechend kann der zentrale Differenzenquotient zweiter Ordnung für ϑ_i^m wie folgt formuliert werden

$$\begin{aligned} \frac{\partial^2 \vartheta_i^m}{\partial x^2} &= \frac{\vartheta_{i+1}^m - 2\vartheta_i^m + \vartheta_{i-1}^m}{\Delta x^2} \qquad (4.4) \\ &= \frac{1}{(\Delta x)^2} \cdot \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} \cdot \begin{bmatrix} \vartheta_{i-1}^m \\ \vartheta_i^m \\ \vartheta_{i+1}^m \end{bmatrix} \; . \qquad (4.5) \end{aligned}$$

4.1.2 Einbau der Randbedingungen

Die in Gleichung (4.4) beschriebene Diskretisierung der Ortsableitung ist für die Ortspunkte auf den Rändern nicht anwendbar, da die Temperaturen unmittelbar jenseits der Ränder unbekannt sind. Bestimmbar ist dagegen die Wärmeflussdichte des durch den Rand strömenden Flusses.

Zunächst soll der **linke Rand** bei x_1 betrachtet werden.
Entwickelt man nach Formel (4.1) die Taylorreihe an der Stelle x_1 vorwärts bis zum zweiten Glied

$$\vartheta_{x_2} = \vartheta_{x_1} + \Delta x \, \frac{\partial \vartheta_{x_1}}{\partial x} + \frac{(\Delta x)^2}{2} \cdot \frac{\partial^2 \vartheta_{x_1}}{\partial x^2}$$

und stellt dieses Polynom 2. Grades nach $\frac{\partial^2 \vartheta_{x_1}}{\partial x^2}$ um, erhält man

$$\frac{\partial^2 \vartheta_{x_1}}{\partial x^2} = \frac{2}{(\Delta x)^2} \left(\vartheta_{x_2} - \vartheta_{x_1} - \Delta x \, \frac{\partial \vartheta_{x_1}}{\partial x} \right) .$$

Nach Gleichung 2.4 kann im letzten Term $\frac{\partial \vartheta_{x_1}}{\partial x}$ ersetzt werden

$$\frac{\partial \vartheta_{x_1}}{\partial x} = -q \, \frac{x_1}{\lambda} .$$

Somit wird die 2. Ableitung nach x an der Stelle x_1 zu

$$\frac{\partial^2 \vartheta_{x_1}}{\partial x^2} = \frac{1}{(\Delta x)^2} \begin{bmatrix} -2 & 2 \end{bmatrix} \begin{bmatrix} \vartheta_{x_1} \\ \vartheta_{x_2} \end{bmatrix} + \frac{2 q_{x_1}}{\Delta x \; \lambda} . \tag{4.6}$$

Für den **rechten Rand** bei x_{nx} ergibt sich unter Verwendung der rückwärtigen Taylorreihenentwicklung und bei Beachtung der gleichen Orientierung von Flussvektor und Flächennormaler

$$\frac{\partial^2 \vartheta_{x_{nx}}}{\partial x^2} = \frac{1}{(\Delta x)^2} \begin{bmatrix} 2 & -2 \end{bmatrix} \begin{bmatrix} \vartheta_{x_{nx-1}} \\ \vartheta_{x_{nx}} \end{bmatrix} - \frac{2 q_{x_{nx}}}{\Delta x \; \lambda} . \tag{4.7}$$

Im Fall der adiabatischen rechten Randbedingung entfällt der rechte Term, d.h. $q_{x_{11}} = 0$.

Im folgenden wird zur Bezeichnung der Temperatur an einem bestimmten Ortspunkt anstelle von ϑ_{x_i} vereinfachend ϑ_i verwendet. Analog wird die Wärmeflussdichte am Ort x_i mit q_i bezeichnet.

4.1.3 Zeitliche Diskretisierung

Die linke Seite der Wärmeleitungsgleichung wird nach dem expliziten Euler-Verfahren oder wiederum anhand der Taylorreihenentwicklung angenähert, nun allerdings für die zeitliche Dimension. Für die erste Ableitung nach der Zeit wird jedoch nur das Taylorpolynom 1.

Grades benötigt $\Big($vgl. Formel (4.2)$\Big)$. Entwickelt wird am Zeitpunkt $t = t_m$, allerdings wird für ϑ^{t_m} ebenfalls verkürzt ϑ^m geschrieben, d.h.

$$\vartheta_i^{m+1} = \vartheta_i^m + \frac{\partial \vartheta_i^m}{\partial t}\,\Delta t + \mathcal{O}(\Delta t)^2 \ .$$

Der loakle Fehler für einen einzelnen Schritt ist propotional zu $(\Delta t)^2$, das explizite Euler-Verfahren hat somit die lokale Fehlerordnung 2. Laut [10, S. 33f.] erhält ein Einschrittverfahren wie das explizite Euler-Verfahren Konsistenzordnung p, wenn es die lokale Fehlerordnung $p+1$ hat. Die Methode hat also nur Konsistenzordnung 1. Das Auflösen nach der ersten Ableitung führt zu

$$\frac{\partial \vartheta_i^m}{\partial t} = \frac{\vartheta_i^{m+1} - \vartheta_i^m}{\Delta t} + \mathcal{O}(\Delta t) \ .$$

Der globale Diskretisierungsfehler für dieses Verfahren hat ebenfalls die Größenordnung $(\Delta t)^p$, ist also proportional zu Δt. Der Beweis ist bei von Harrach nachzulesen [10, S. 33ff]. Das explizite Euler-Verfahren verfügt somit über die Konvergenzordnung 1.

Die meisten in Julia verfügbaren Löser (häufig Runge-Kutta-Verfahren) können wesentlich höhere Genauigkeiten liefern, Toleranzen für den lokalen Fehler können explizit vorgegeben werden.

Das numerische Schema für die homogene WLG mit dem FTCS-Verfahren lautet somit

$$\frac{\vartheta_i^{m+1} - \vartheta_i^m}{\Delta t} = \frac{\alpha}{(\Delta x)^2}\,(\vartheta_{i-1}^m - 2\vartheta_i^m + \vartheta_{i+1}^m)$$

und als Algorithmus für die unbekannte Temperatur zum neuen Zeitpunkt (t_{m+1}) am Ort x_i ergibt sich nach Umstellen

$$\vartheta_i^{m+1} = \vartheta_i^m + \frac{\alpha\,\Delta t}{(\Delta x)^2}(\vartheta_{i-1}^m - 2\vartheta_i^m + \vartheta_{i+1}^m) \ . \tag{4.8}$$

Dies gilt jedoch nur für das Stabinnere ($\Omega = (0, l)$). In Abschn. 4.1.5 wird das lineare Gleichungssystem für das gesamte System aufgestellt, inklusive der Randbedingungen. Zuvor soll jedoch die Stabilität der Methode betrachtet werden.

4.1.4 Stabilität des FTCS-Verfahrens

Gleichung (4.8) kann auch folgendermaßen geschrieben werden

$$\vartheta_i^{m+1} = \frac{\alpha\ \Delta t}{(\Delta x)^2}\ \vartheta_{i-1}^m + \left(1 - 2\ \frac{\alpha\ \Delta t}{(\Delta x)^2}\right)\ \vartheta_i^m + \frac{\alpha\ \Delta t}{(\Delta x)^2}\ \vartheta_{i+1}^m \ .$$

Der Koeffizient wird zur Neumann-Zahl Ne zusammengefasst

$$Ne = \tfrac{\alpha\ \Delta t}{(\Delta x)^2}$$

und die homogene Wärmeleitungsgleichung wird zu

$$\vartheta_i^{m+1} = Ne\ \vartheta_{i-1}^m + (1 - 2Ne)\ \vartheta_i^m + Ne\ \vartheta_{i+1}^m \ .$$

Die Neumann-Zahl beschreibt den in einem Zeitschritt aufgrund der Wärmeleitung zurückgelegten Weg, bezogen auf die Ortsschrittweite [20, S. 151]. Sie dient als Stabilitätskennzahl für das FTCS-Verfahren und andere explizite Verfahren([19, S. 140f]). Stabilität ist gegeben, wenn die Bedingung

$$Ne \ \leq \frac{1}{2}$$

erfüllt ist (vgl. [24, S.253], hier wird die Konstante als Diffusionszahl bezeichnet). Für die Zeitschrittsteuerung muss daher gelten

$$\Delta t \leq \frac{(\Delta x)^2}{2\alpha} \ . \tag{4.9}$$

Im Projekt PI_Slider wird explizit geprüft, ob die verwendeten Werte diese Bedingung erfüllen. Gegebenenfalls wird der Nutzer in einer Fehlermeldung informiert, welchen Wert Δt maximal annehmen darf. Bei Verwendung eines Julia-Solvers ist diese Stabilitätsprüfung nicht notwendig, ggf. führt der Solver einen Stabilitätscheck durch.

Im Abschnitt 4.2 wird die Linienmethode beschrieben. Diese ermöglicht den Einsatz von generischen Lösern für die Approximation der zeitlichen Ableitung. Hier stehen mit Julia Verfahren zur Verfügung, die dem expliziten Euler-Verfahren unter anderem in Bezug auf numerische Stabilität und Genauigkeit überlegen sind. So wurde im Projekt PI_Ode_Solve beispielsweise der Rodas4-Solver verwendet, ein A-stabiles Verfahren 4. Ordnung, das die Rosenbrock-Methode verwendet (siehe Abschnitt 5.2.1).

4.1.5 Aufstellen des linearen Gleichungssystems

Zunächst wird die zweite örtliche Ableitung unter Einbeziehung der Randbedingungen in Matrixschreibweise aufgestellt. Diese ergibt sich aus Gleichung (4.5) für das Stabinnere, (4.6) für den linken ($x = x_1$) und (4.7) für den rechten Rand ($x = x_{nx}$).

$$\frac{\partial^2}{\partial x^2}\begin{bmatrix}\vartheta_1\\ \vartheta_2\\ \vdots\\ \\ \vartheta_{nx}\end{bmatrix} = \frac{1}{\Delta x^2}\begin{bmatrix}-2 & 2 & 0 & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1\\ & & 0 & 2 & -2\end{bmatrix}\begin{bmatrix}\vartheta_1\\ \vartheta_2\\ \vdots\\ \\ \vartheta_{nx}\end{bmatrix} + \frac{1}{\Delta x\ \lambda}\begin{bmatrix}2q_1\\ 0\\ \vdots\\ 0\\ -2q_{nx}\end{bmatrix} \tag{4.10}$$

Die zweite Ortsableitung wird anschließend in den Algorithmus gemäß (4.8) eingefügt.

$$\begin{bmatrix}\vartheta_1^{m+1}\\ \vartheta_2^{m+1}\\ \vdots\\ \\ \vartheta_{nx}^{m+1}\end{bmatrix} = \underbrace{\begin{bmatrix}1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1\end{bmatrix}}_{\mathbf{M}}\begin{bmatrix}\vartheta_1^{m}\\ \vartheta_2^{m}\\ \vdots\\ \\ \vartheta_{nx}^{m}\end{bmatrix} + \alpha\,\Delta t\left(\underbrace{\frac{1}{(\Delta x)^2}\begin{bmatrix}-2 & 2 & 0 & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1\\ & & 0 & 2 & -2\end{bmatrix}}_{A}\begin{bmatrix}\vartheta_1^{m}\\ \vartheta_2^{m}\\ \vdots\\ \\ \vartheta_{nx}^{m}\end{bmatrix} + \frac{1}{\Delta x\ \lambda}\underbrace{\begin{bmatrix}2q_1\\ 0\\ \vdots\\ 0\\ -2q_{nx}\end{bmatrix}}_{\tilde{\boldsymbol{q}}}\right)$$

Dabei ist M die Massematrix der Zeitableitung [19, S. 140],
A die Matrix für die Ortspunkte des zentralen Differenzenquotienten und
$\tilde{\boldsymbol{q}}$ der Vektor für die Rb mit Wärmeflussdichte
q_1, die dem System bei $x = x_1$ zuströmt und
q_{nx}, die bei $x = x_{nx}$ abströmt.

In kompakter Schreibweise und mit $Ne = \frac{\alpha\,\Delta t}{(\Delta x)^2}$ und $\alpha = \frac{\lambda}{c\,\rho}$ ergibt sich

$$\boldsymbol{\vartheta}^{m+1} = (M + Ne\,A)\,\boldsymbol{\vartheta}^m + \frac{\cancel{\lambda}\;\Delta t}{\cancel{c\rho}\;\Delta x}\,\tilde{\boldsymbol{q}}$$

mit fettgedruckten Minuskeln für Vektoren und Majuskeln für Matritzen.
Da im Fall der Wärmeleitung nicht Masse sondern Energie gespeichert wird, beschreibt die Massematrix den zeitlichen Verlauf der Temperatur und damit die gespeicherte Wärmeenergie. Die Matrix A mit dem Vorfaktor Ne wird auch als Diffusionsmatrix ($D = Ne\,A$) bezeichnet [19, S. 140]. Sie beschreibt die Temperaturänderung aufgrund des Temperaturgradienten.

Für die gesamte homogene WLG (FTCS) in Matrixschreibweise erhält man dementsprechend

$$\begin{bmatrix}\vartheta_1^{m+1}\\ \vartheta_2^{m+1}\\ \vdots\\ \vartheta_{nx}^{m+1}\end{bmatrix} = \left(\underbrace{\begin{bmatrix}1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1\end{bmatrix}}_{\mathbf{M}} + Ne \underbrace{\begin{bmatrix}-2 & 2 & 0 & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1\\ & & & 0 & 2\;-2\end{bmatrix}}_{D} \right) \begin{bmatrix}\vartheta_1^{m}\\ \vartheta_2^{m}\\ \vdots\\ \vartheta_{nx}^{m}\end{bmatrix} + \frac{\Delta t}{c\rho\;\Delta x} \underbrace{\begin{bmatrix}2q_1\\ 0\\ \vdots\\ 0\\ -2q_{nx}\end{bmatrix}}_{\tilde{\boldsymbol{q}}} . \quad (4.11)$$

Für einige Julia-Anwendungsfälle wird eine Massenmatrix benötigt, für unser Problem können M und D allerdings zusammengefasst werden und der Algorithmus für die Temperatur zum neuen Zeitpunkt für alle Ortspunkte wird zu

$$\begin{bmatrix}\vartheta_1^{m+1}\\ \vartheta_2^{m+1}\\ \vdots\\ \vartheta_{nx}^{m+1}\end{bmatrix} = \begin{bmatrix}1-2Ne & 2Ne & 0 & & \\ Ne & 1-2Ne & Ne & & \\ & \ddots & \ddots & \ddots & \\ & & Ne & 1-2Ne & Ne\\ & & 0 & 2Ne & 1-2Ne\end{bmatrix} \begin{bmatrix}\vartheta_1^{m}\\ \vartheta_2^{m}\\ \vdots\\ \vartheta_{nx}^{m}\end{bmatrix} + \frac{\Delta t}{c\rho\;\Delta x} \begin{bmatrix}2q_1\\ 0\\ \vdots\\ 0\\ 2q_{nx}\end{bmatrix} .$$

Auf diese Weise wurde das gesamte Wärmeleitungsproblem „von Hand“ diskretisiert. Im Projekt PI_Slider wird dieses Verfahren in einer Schleife verwendet und die ermittelten Temperaturwerte für $\boldsymbol{\vartheta}^{m+1}$ werden, wie auch die Ausgabewerte der Regelung (q_1^m), in präallozierten Arrays gespeichert. Dabei wird q_1 wegen $T_{samp} = 11\,\Delta t$ nur bei jedem elften

Durchlauf aktualisiert und gespeichert.
Das Projekt wurde unter anderem verwendet, um adäquate Werte für K_p und K_i festzulegen. Mit diesen Parametern wird im nächsten Schritt mit PI_OdeSolve_ML ein Temperaturverlauf für alle $nx = 11$ Orte im Stab erstellt, der die Zielwerte für das maschinelle Lernen (engl. Machine Learning, ML) liefert.

Alternative Vorgehensweise: Auf die numerischen Löser für die zeitliche Diskretisierung wurde bereits verwiesen. Möglichkeiten zur Diskretisierung der rechten Seite der WLG bieten Julia-Pakete wie DiffEqOperators.jl [29] zur softwaregestützten Differenzierung mit finiten Differenzen oder AppoxFun.jl . Zum Vorgehen mit DiffEqOperators.jl siehe auch [35].

4.1.6 Dünnbesetzte Matritzen in Julia

Die Systemmatrix ebenso wie der Vektor für die Randbedingungen sind dünnbesetzt. Das Modul der Julia-Standardbibliothek SparseArrays [3] bietet die Möglichkeit, nur die Werte ungleich Null sowie deren Position zu speichern und keinen Speicherplatz für die Nullen zu verschwenden. Die zusammengefasste $M + D$ -Matrix wurde als *spdiagm* (sparse diagonal matrix) definiert, wobei nur die drei Diagonalen gespeichert werden.
Zunächst wurde auch der dünnbesetzte Vektor $\tilde{\boldsymbol{q}}$ als SparseVector definiert. Dies hat sich jedoch nicht bewährt. Während die Multiplikation der SparseMatrix A mit dem dichtbesetzten Vektor ϑ einen *dense vector* ergibt, ist das Ergebnis der Addition mit einem SparseVector jedoch immer ein SparseArray. In diesem Fall ist das weder im Hinblick auf Speichereffizienz noch auf die Berechnung vorteilhaft. Dies fiel beim Benchmarking auf und ist in der Dokumentation nicht beschrieben. Im Forum fand sich schließlich eine Diskussion zu diesem Problem [14].
Die Matrix bleibt im Verlauf der Berechnungen unverändert. Daraus lässt sich allerdings kein Vorteil im Hinblick auf die Zugriffsgeschwindigkeit ziehen. Auch ist die Matrix mit nur 11 Zeilen so klein, dass der eingesparte Speicherplatz für die Nullen kaum den Overhead zum Speichern der Indizes lohnt. Für eine so kleine Matrix wäre ein Stack allokiertes SArray (Paket StaticArrays, immutable) sicherlich effizienter (vgl. [31]).
Darauf wurde jedoch verzichtet, da das Programm die Option eröffnen sollte, mit wenigen Anpassungen auf eine größere Anzahl (>100) von Ortspunkten erweitert zu werden.

4.2 Vertikale Linienmethode (engl.: method of lines)

Eine gute Definition dieses Verfahrens liefert das Wolfram Documentation Center:
"The numerical method of lines is a technique for solving partial differential equations by discretizing in all but one dimension and then integrating the semi-discrete problem as a system of ODEs or DAEs. A significant advantage of the method is that it allows the solution to take advantage of the sophisticated general-purpose methods and software that have been developed for numerically integrating ODEs and DAEs."[16]
Nach dieser Methode wurden die Ableitungen bezüglich der räumlichen Variablen in der WLG wie in Abschnitt 4.1 beschrieben approximiert, die Zeit blieb dagegen kontinuierlich. Nachdem das Wärmeleitungsproblem auf diese Weise in ein System von Differentialgleichungen erster Ordnung umgewandelt wurde, kann es als Zustandsraummodell dargestellt werden.

4.2.1 Zustandsraumdarstellung

Das Zustandsraummodell beschreibt den Systemzustand mithilfe von dessen Zustandsgrößen, die immer Ausgangsgröße eines Integrators sind.
Für die Darstellung der allgemeinen Form des Modells wurde [17, S. 74ff.] verwendet.

Ein lineares System mit einer Eingangsgröße und einer Ausgangsgröße wird beschrieben durch die System- oder Zustandsgleichung

$$\dot{x}(t) = A\ \mathbf{x}(t)\ + \mathbf{b}\ u(t) \quad \text{mit den Anfangsbedingungen} \quad \boldsymbol{x}(0) = \boldsymbol{x_0}$$

und die Mess- oder Ausgabegleichung

$$y(t) = \boldsymbol{c^T}\,\boldsymbol{x}(t) + d\ u(t)\ .$$

Die WLG in ihrer semidiskretisierten Form lautet für das betrachtete System

$$\frac{\partial}{\partial t}\begin{bmatrix}\vartheta_1\\ \vartheta_2\\ \vdots\\ \\ \vartheta_{nx}\end{bmatrix} = \frac{\lambda}{c\rho\,(\Delta x)^2}\begin{bmatrix}-2 & 2 & 0 & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1\\ & & 0 & 2 & -2\end{bmatrix}\begin{bmatrix}\vartheta_1\\ \vartheta_2\\ \vdots\\ \\ \vartheta_{nx}\end{bmatrix} + \frac{1}{c\rho\ \Delta x}\begin{bmatrix}2q_1\\ 0\\ \vdots\\ 0\\ -2q_{nx}\end{bmatrix}\ . \tag{4.12}$$

Dies entspricht der **Systemgleichung** in Zustandsraumdarstellung

$$\dot{\boldsymbol{x}}(t)\ =\ A\ \mathbf{x}(t)\ +\ \mathbf{b}\ u(t) + \mathbf{e}\ s(t)$$

mit

$\dot{\boldsymbol{x}}(t)$	Zeitableitung des Zustandsvektors[I]	
A	Systemmatrix, s. oben	$A \in \mathbb{R}^{nx\times nx}$
$\mathbf{x}(t) = \begin{bmatrix}\vartheta_1\\ \vartheta_2\\ \vdots\\ \vartheta_{nx}\end{bmatrix}$	Zustandsvektor $\boldsymbol{\vartheta}(t)$	$\boldsymbol{\vartheta} \in \mathbb{R}^{nx\times 1}$
$\boldsymbol{b} = \begin{bmatrix}1\\ 0\\ \vdots\\ 0\end{bmatrix}$	Steuermatrix, Zeilenvektor	$\boldsymbol{b} \in \mathbb{R}^{nx\times 1}$
$\boldsymbol{e} = \begin{bmatrix}0\\ 0\\ \vdots\\ 1\end{bmatrix}$	Matrix für den Wärmeverlust, Zeilenvektor	$\boldsymbol{c} \in \mathbb{R}^{nx\times 1}$
$u(t) = q_1(t)$	Eingangsgröße (Regleroutput q_1 s. Formel (2.6))	
$s(t) = q_{nx}(t)$	$q_{nx}(t)$ in die Umgebung abströmende Flussdichte an $x = x_{11}$, vgl. die Abschnitte 2.2.2 bis 2.2.4	
$\frac{2}{c\rho\,\Delta x}$	Vorfaktor für $\boldsymbol{b}\,u(t)$	
$-\frac{2}{c\rho\,\Delta x}$	Vorfaktor für $\boldsymbol{e}\,s(t)$	

[I] Hier wird ausnahmsweise der üblichen Darstellung folgend die Ableitung nach der Zeit durch einen Punkt über dem Vektor gekennzeichnet.

Im folgenden soll der rechte Term der Systemgleichung $\mathbf{b}u(t) + \mathbf{e}s(t)$ genauer betrachtet werden. Exemplarisch soll für den Term $\mathbf{e}\,s(t)$ die Neumann-RB herangezogen werden, um das System im Zustandsraum darzustellen. Im Fall der adiabaten rechten Randbedingung verschwindet $\mathbf{e}\,s(t)$. Im Fall der natürlichen Randbedingung wird er aus der vierten Potenz der Temperaturen berechnet, was die Integration in das Zustandsraummodell erschwert.

Der rechte Term $\mathbf{b}\,u(t) + \mathbf{e}\,s(t)$ setzt sich für die Neumann-Rb zusammen aus

$$\boldsymbol{b}\ u(t) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \frac{2}{c\rho\ \Delta x} \left(K_p\Big(\vartheta_{ref}(t) - \vartheta_{end}(t)\Big) + K_i \underbrace{\int_0^t \Big(\vartheta_{ref}(t) - \vartheta_{end}(t)\Big)dt}_{v(t)} \right) \tag{4.13}$$

$$\text{und} \quad \boldsymbol{e}\ s(t) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \frac{-2}{c\rho\,\Delta x} \cdot \lambda \frac{\vartheta_{end}(t) - \vartheta_{amb}(t)}{\Delta x} \quad = \quad \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \frac{-2\lambda}{c\rho\,(\Delta x)^2}\Big(\vartheta_{end}(t) - \vartheta_{amb}(t)\Big) \quad .$$

Die **Messgleichung** für die Ausgangsgröße y in der Zustandsraum-Darstellung bei einem Eingrößensystem ohne Durchschaltung lautet

$$y(t) = \boldsymbol{c^T}\ \boldsymbol{x}(t)\ .$$

Für ϑ_{end} erhält man entsprechend

$$\vartheta_{end}(t) = \boldsymbol{c^T}\ \boldsymbol{\vartheta}(t)\ . \tag{4.14}$$

Dabei ist $\boldsymbol{c^T}$ der Beobachtungsvektor ($1 \times nx$ Spaltenvektor), $\boldsymbol{c}$ entspricht im hier untersuchten Fall $\boldsymbol{e}$.

Die Systemgleichung kann nun umformuliert werden.

$$\dot{\boldsymbol{\vartheta}}(t) = \frac{\lambda}{c\rho(\Delta x)^2} A\,\boldsymbol{\vartheta}(t) + \underbrace{\frac{2}{c\rho(\Delta x)} K_p\,\boldsymbol{b}\,\dot{v}(t) + \frac{2}{c\rho(\Delta x)} K_i\,\boldsymbol{b}\,v(t)}_{\boldsymbol{b}\,u(t)} - \underbrace{\frac{2\lambda}{c\rho(\Delta x)^2}\,\boldsymbol{e}\Big(\vartheta_{end}(t) - \vartheta_{amb}(t)\Big)}_{\boldsymbol{e}\,s(t)} \tag{4.15}$$

In Gleichung (4.13) wird das Fehlerintegral durch die Variable $v(t)$ ersetzt. Dieser Integrator repräsentiert eine weitere Zustandsgröße, die wir dem Differentialgleichungssystem hinzufügen. Die Regeldifferenz wird somit zu

$$\dot{v}(t) = \Big(\vartheta_{ref}(t) - \vartheta_{end}(t)\Big) \tag{4.16}$$

und durch Ersetzen gemäß Formel (4.14) erhält man

$$\dot{v}(t) = \Big(\vartheta_{ref}(t) - \boldsymbol{c^T}\boldsymbol{\vartheta}(t)\Big)\ . \tag{4.17}$$

Nach Ersetzen von $\dot{v}(t)$ entsprechend Formel (4.17) und Ausmultiplizieren ergibt sich für die

Temperaturänderungsrate

$$\dot{\boldsymbol{\vartheta}}(t) \quad = \quad \overbrace{\tfrac{\lambda}{c\rho(\Delta x)^2} A\,\boldsymbol{\vartheta}(t)} \quad + \quad \underbrace{\tfrac{2}{c\rho\,\Delta x} K_p\,\vartheta_{ref}(t)\;\boldsymbol{b} - \overbrace{\tfrac{2}{c\rho\,\Delta x} K_p\,\boldsymbol{b}\,\boldsymbol{c^T}\boldsymbol{\vartheta}(t)} + \tfrac{2}{c\rho\,\Delta x} K_i\,v(t)\,\boldsymbol{b}}_{\mathbf{b}u(t)} \quad \ldots$$

$$\ldots \quad \underbrace{+\tfrac{2\lambda}{c\rho(\Delta x)^2}\,\vartheta_{amb}(t)\;\boldsymbol{e} - \overbrace{\tfrac{2\lambda}{c\rho(\Delta x)^2}\,\boldsymbol{e}\,\boldsymbol{c^T}\boldsymbol{\vartheta}(t)}}_{\mathbf{e}s(t)} \; .$$

Fasst man die Terme zusammen, die $\boldsymbol{\vartheta}(t)$ enthalten (obere geschweifte Klammern in (4.2.1)), erhält man

$$\dot{\boldsymbol{\vartheta}}(t) \quad = \quad \left(\tfrac{\lambda}{c\rho(\Delta x)^2} A - \tfrac{2}{c\rho\,\Delta x} K_p\,\boldsymbol{b}\,\boldsymbol{c^T} - \tfrac{2\lambda}{c\rho(\Delta x)^2}\,\boldsymbol{e}\,\boldsymbol{c^T}\right)\boldsymbol{\vartheta}(t) \quad + \quad \tfrac{2}{c\rho(\Delta x)} K_p\,\vartheta_{ref}(t)\;\boldsymbol{b} \quad \ldots$$

$$\ldots \quad + \tfrac{2}{c\rho(\Delta x)} K_i\,v\,\boldsymbol{b} \quad + \tfrac{2\lambda}{c\rho(\Delta x)^2}\,\vartheta_{amb}(t)\;\boldsymbol{e} \; .$$

Wegen

$$\boldsymbol{b}\;\boldsymbol{c^T} = \begin{bmatrix} 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \quad \text{und} \quad \boldsymbol{e}\;\boldsymbol{c^T} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

ergibt sich für die Systemmatrix des geschlossenen Regelkreises mit dem Wärmeverlust durch die Neumann-RB

$$A_{Cl} = \begin{bmatrix} -2\gamma & 2\gamma & 0 & \cdots & -\frac{2K_p}{c\rho\,\Delta x} \\ \gamma & -2\gamma & \gamma & & \\ & \ddots & \ddots & \ddots & \\ & & \gamma & -2\gamma & \gamma \\ & & & 2\gamma & -4\gamma \end{bmatrix} \quad \text{mit} \quad \gamma = \tfrac{\lambda}{c\rho(\Delta x)^2} \; .$$

Die Systemgleichung kann nun folgendermaßen geschrieben werden

$$\tfrac{\partial}{\partial t}\begin{bmatrix} \vartheta_1 \\ \vdots \\ \vartheta_{11} \\ v \end{bmatrix} = \left[\begin{array}{c|c} A_{Cl} & \frac{2K_i}{c\rho\,\Delta x}\begin{bmatrix} \boldsymbol{b} \end{bmatrix} \\ \hline \begin{bmatrix} -\boldsymbol{c^T} \end{bmatrix} & 0 \end{array}\right] \cdot \begin{bmatrix} \vartheta \\ v \end{bmatrix} + \left[\begin{array}{c|c} \frac{2K_p}{c\rho\,\Delta x}\begin{bmatrix} \boldsymbol{b} \end{bmatrix} & \\ \hline & \begin{bmatrix} 1 \end{bmatrix} \end{array}\right] \cdot \boldsymbol{\vartheta_{ref}} + \tfrac{2\lambda}{c\rho(\Delta x)^2}\,\vartheta_{amb}\begin{bmatrix} \boldsymbol{e} \end{bmatrix} \; .$$

4.2.2 Exkurs: Eigenwerte der Systemmatrix A

Es wurden die Eigenwerte der Systemmatrix des offenen Regelkreises

$$A = \frac{\lambda}{c\rho(\Delta x)^2} \begin{bmatrix} -2 & 2 & 0 & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 2 & -2 \end{bmatrix}$$

ermittelt. Diese zeigt Abbildung 4.2. Es fällt auf, dass ein Eigenwert der Systemmatrix bei Null liegt. Sofern keine Kürzung zwischen Null- und Polstellen durchgeführt wird, entsprechen die Eigenwerte der Systemmatrix den Polstellen der Übertragungsfunktion. Eine Polstelle der Streckenübertragungsfunktion im Ursprung passt auch zu dem integrierenden Verhalten des Systems, wie es bei der adiabatischen Randbedingung in Abschnitt 2.2.2 beschrieben wurde.

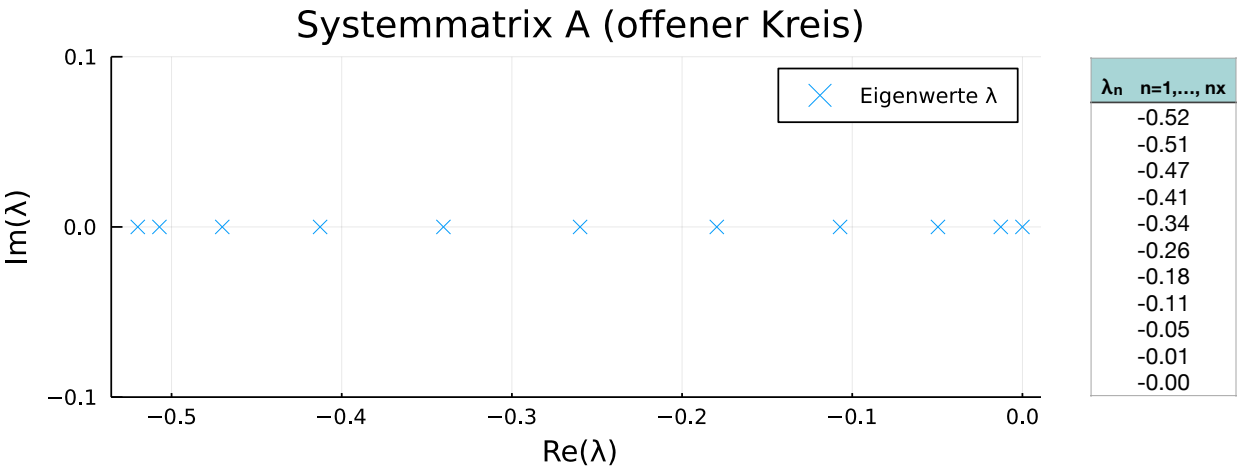


Abb. 4.2: Eigenwerte der Systemmatrix A

5 Zieldaten und generische Löser

5.1 Programm zur Datenerhebung

Das Programm pi_solver_datagen im Projekt PI_OdeSolv_ML berechnet die Temperaturen für alle nx Ortspunkte über ein gegebenes Zeitintervall t_f. Verwendet wurde wie zuvor $nx = 11$ und $t_f = 2100s$. Hierbei wird allerdings die vertikale Linienmethode benutzt, die in Abschnitt 4.2 beschrieben ist. Das heißt, die semidiskretisierte WLG mit der diskretisierten zweiten Ortsableitung einschließlich Randbedingungen wurde - wie in Gleichung (4.12) gezeigt - zu einem System von nx Differentialgleichungen erster Ordnung. Diesem Differentialgleichungssystem wurde - wie in Abschnitt 4.2.1 beschrieben - eine Zeile für das Fehlerintegral mit der Variablen v, siehe (4.16), hinzugefügt. Hier wird bei Aufruf der Funktion die Regeldifferenz $\dot{v}$ und das Fehlerintegral v berechnet, die wiederum für die Berechnung des Reglerausgangs und damit der zugeführten Wärmeflussdichte q_1 verwendet werden, wie in Formel (4.15) zu sehen ist.
Dieses Gleichungssystem kann mit verschiedenen generischen Lösern berechnet werden. Aus dem Softwarepaket DifferentialEquations.jl [29] wurden drei Solver-Algorithmen ausgewählt, um sie hinsichtlich ihrer Geschwindigkeit zu vergleichen. Diese sind in Abschnitt 5.2 beschrieben.
Anschließend kann ein Plot erstellt werden, in dem der Temperaturverlauf am linken (x_1) und am rechten (x_{nx}) Stabende sowie die zugeführte Wärmeflussdichte dargestellt sind. Der verwendete Solver-Algorithmus (dem Lösungsobjekt entnommen) und die verwendete rechte RB werden ebenfalls festgehalten. Ein Beispiel zeigt Abb. 5.1

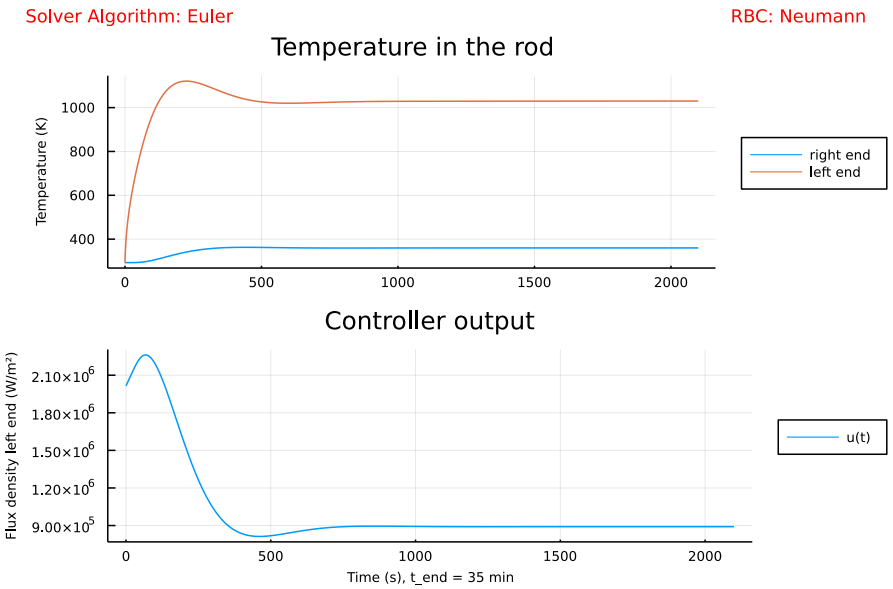


Abb. 5.1: Plot der Temperaturverteilung, erstellt mit pi_solver_datagen

Schließlich werden die berechneten Daten sowie die verwendeten Bedingungen und Parameter in einer tsv-Datei gespeichert. Einen Ausschnitt einer solchen Datentabelle zeigt Tab. 5.1. Der hier ausschnitthaft dargestellte Datensatz wird später im maschinellen Lernen verwendet. Um die Tabelle interpretierbar zu machen, wurde ein Stringarray als Kopfzeile (blaue Zeile) hinzugefügt. Die Tabellen wurden mithilfe von DelimitedFiles [21] erstellt, ein Modul der Julia-Standardbibliothek.

Euler-algorithm
Neumann-RBC
Kp=40000 Ki=50.0 9_ref=360.0 l=0.3

sol.t	$9(x_1)$	$9(x_2)$	$9(x_3)$	$9(x_4)$	$9(x_5)$	$9(x_6)$	$9(x_7)$	$9(x_8)$	$9(x_9)$	$9(x_{10})$	$9(x_{11})$	je	u(t)
0.0	293.0	293.0	293.0	293.0	293.0	293.0	293.0	293.0	293.0	293.0	293.0	67.0	2.6833e6
1.0	339.551526640514 85	295.841417708752 2	293.11578106238375	293.0034934580859	293.0000828260059 5	293.0000016039199 17	293.0000002801740	293.0000000003607	293.0000000000043	293.0	293.0	133.999999999999 77	2.6867e6
2.0	376.660388452627	302.826566881977 266	293.7975726339535 6	293.0489756707595 5	293.0024036115629	293.0000097785357 5	293.0000033842890 6	293.0000001015825 4	293.0000000002684	293.0000000000631	293.0000000000012 5	200.999999999999 97	2.6900499999995e6
3.0	407.220257257828 1	312.184311054749 46	295.2875960295044	293.2093315711713	293.0154526073945 4	293.0095227100155	293.0000502199525	293.0000002309223 76	293.0000000939403	293.0000000034203	293.0000000001121	267.999999999972 94	2.6933999999955148e6
4.0	433.140551001407 1	322.834782424878 9	297.6217767598776 6	293.5564579098231 7	293.0544663892129	293.0044745854870	293.0003158859118	293.0000010693385	293.0000010639385	293.0000000052602	293.0000000023403	334.999999999210 33	2.6967499999063485e6
5.0	455.699525340076 5	334.122597426021 74	300.7379495937138	294.1453674060161	293.1388266413451 3	293.0141912839187 6	293.0012509820478	293.0000096748126 85	293.0000066547704	293.0000004116531	293.0000002305044 6	401.999999989868 67	2.7000999990773216e6
6.0	475.764626334506 7	345.653985488175 3	304.5347325795201 5	295.0109409289738	293.2890929831838	293.0352123279371 3	293.0037106614229	293.0003439105258	293.0000284053155	293.0000021133911	293.0000001424099	468.999999923700 6	2.7034499942997 9e6
7.0	493.933967797723 65	357.195453409508 7	308.9019848869661 5	296.1700653786270 4	293.5246972741906	293.0739141386465	293.0090368958224	293.0009973402 27	293.0000093732512	293.0000081372684	293.0000006397939 4	535.999999590678 8	2.7067999743877 76e6
8.0	510.627825781162 7	368.612717209897 5	313.7352650327227 4	297.6253823769127	293.8624395227 47	293.1374749114079 3	293.0190787082 77	293.0023395539173 5	293.0002565932961	293.0000254237594 4	293.0000022807527 7	602.999998295979 4	2.7101499086846 88e6
9.0	526.146503110169 6	379.831382521424 2	318.9422428424649	299.3694171601853	294.3158046783281 6	294.3158047832816	293.2334074504095	293.0361589734576	293.0049609504131	293.0000678106916 3	293.0000006823643	669.999994145866 8	2.7134997267615735e6
10.0	540.709118053249	390.813811172726 05	324.4442535868534	301.3879344871073	294.8947534701743	293.3691301552875 6	293.0629980009823	293.0095434918065 4	293.0012975403818 7	293.0001598009859 5	293.0000177968670 4	736.999982698494 8	2.7168492872602 43e6
11.0	554.478745531277	401.544443423657 9	330.1758551821878 3	303.6626464149257 5	295.6058838916326	293.5516278396728	293.1026109615787 6	293.0169808429120 6	293.0025264233 24	293.0003409300 76	293.0000415665766	803.999954697494 4	2.7201983350718105e6
12.0	567.578963081035 2	412.020701440129 7	336.0834798567625	306.1732990635576 4	296.4528441005518 7	293.7872309827680 5	293.1582021688651	293.0283481861922	293.0045747691869 5	293.0006704832716	293.0000868999777	870.999892586646 6	2.7235464470294216e6
13.0	580.105509739856 2	422.247569609441 8	342.1236248165088	308.8989603917268	297.4367211882301 4	294.0814436224785 6	293.2330494482099	293.0448825936461 5	293.0077981140284	293.0012320654385 6	293.0001754824367	937.999765793062	2.7268929689814975e6
14.0	592.133759451800 4	432.234116073678 76	348.2612323711540 5	311.8190414198724	298.5565559684384	294.4388869457216 6	293.3304099456968 3	293.0679587073304 3	293.0126319440755 7	293.0021378662266	293.0003257460633 7	1004.99952305615 89	2.7302369463102734e6
15.0	603.724086190907 8	441.991430998011	354.4681550471430 6	314.9138801632043	299.8097861447014 3	294.8632880085183	293.4534377538437 3	293.0990594767981	293.0195911005159	293.0035322721093 4	293.0005725590299 3	1071.99904861827 434	2.7335770519479397e6
16.0	614.925731627093 6	451.531145829624 566	360.7218357684928	318.1650317147268	301.1925997117002 7	295.3574864886204	293.6051078904409	293.1397413278924 7	293.0292651679081	293.0055943533634 5	293.0009959658944	1138.99833786793 37	2.7369115182576 2e6
17.0	625.779272202715 2	460.866160918401 9	367.0042490167 97	321.5555147657254 3	302.7003346619905	295.9235244634378 4	293.7881787919529 3	293.1916063980238	293.0423132432630 4	293.0085397975015 6	293.0015446145137	1205.99711085575 84	2.7402380709822395e6
18.0	636.318693407827 5	470.007189925211 55	373.3009936669936	325.0698031006838	304.3277086800899	296.5626941148250 4	294.0051465284547	293.2562685154634 3	293.0594545134405	293.0126621242116	293.0023970470879	1272.99517458671 72	2.7435538768458 19e6
19.0	646.572746938901 4	478.965634409691 4	379.6005838232144	328.6938250675956	306.0690608448893	297.2756280539468 3	294.2582260304419 6	293.3353260560364	293.0814584969722 3	293.0181277902472 6	293.0036027290032	1339.99222080317 96	2.7468555018800316e6
20.0	656.565933113685 8	487.751870247957 9	385.8938688052067	332.4149204052072 35	307.9185587371798	298.0624044102537 5	294.5493517668287	293.4303420009403	293.1091359047092	293.0253842802798	293.0052628555767	1406.98784787196 77	2.7501388781705316e6
21.0	666.319416485787 8	496.375604056811 893	392.1735729505497	336.2217006638884	309.8702854506001	298.9225961122874	294.8801602694420 6	293.5428163515673 5	293.1433251472128	293.0347469601556	293.0074944687574	1473.98154556475 88	2.7533992985279425e6
22.0	675.851524945633	504.845805896443 84	398.4339242892000 4	340.1039835050798 4	311.9183882696626	299.855379560 21	295.2520126805511	293.6741760205501	293.1848837334677 6	293.0466036900700 7	293.0104305716356 5	1540.97267855263 73	2.7566314110622057e6
23.0	685.178258587535 8	513.170759446890 3	404.6703509998 1	344.0526696173533 3	314.0571399640247 7	300.8595969015092	295.6660026321586	293.8257603004282	293.2346773158 65	293.0613677188328 6	293.0142196736209	1607.96047071291 54	2.7598292365908 09e6
24.0	694.313695295368	521.358125467693 3	410.8792920298269	348.0596257051536	316.2809773647574	301.9338070321345	296.1229638693796	293.9988071428997	293.2935680814212	293.0794742683795 6	293.0190248116315 4	1674.94399091422 54	2.7629862070804 5e6
25.0	703.270205899994	529.411924311597 1	417.0579150954214	352.1176089859605	318.5845755976926	303.0763731285570 7	296.6235075173 94	294.1944562160308	293.3624098360994	293.1013774905091 5	293.0250229263805 3	1741.92213697148 6	2.7660951897933536e6
26.0	712.058769828764 8	537.347641001485 5	423.2040586326003	356.2201476633594	320.9628337931298	304.2854809138428 3	297.1680225880876	294.4137356037879	293.4420359731734	293.1275445672661	293.0324030782248 3	1808.89362463997 49	2.7691485581030054e6
27.0	720.689138150774 6	545.162237817164 4	429.3160616561848	360.3614681495363	323.4109070111539	305.5591965162177 5	297.7567054983352	294.6575650187 55	293.5332544769566 4	293.1584519958736 3	293.0413651231525	1875.85697532198 83	2.7721382483899 9e6
28.0	729.169965760089 3	552.864178873196 8	435.3926699390465	364.5364209907957	325.9242204126255	306.8955119512339	298.3895871723476 6	294.9267606184158	293.6368442529464	293.1945822452609	293.0521183755632	1942.81050421153 4	2.7750557901880 476e6
29.0	737.509011483821 2	560.458515489789	441.4329677603671 5	368.7403967685521 6	328.4984411234841	308.2923491180540 6	299.0665338915428	295.2220264623791	293.7535451308365	293.2364174621560 6	293.0648790921358 7	2009.75231614578 32	2.7778924521218548e6
30.0	745.713182536805 2	567.949869771179 6	447.4362978589267	372.9692785454127	331.1295028403684 5	309.7476135892 38	299.7872860982455 4	295.5439695927554	293.8840598122795 3	293.2844380758237 4	293.0798695070765	2076.68029436771 17	2.7806392344353246e6

Tab. 5.1: Ausschnitt aus der mit pi_solver_datagen erstellten Datentabelle

5.2 Numerische Löser

Ausgewählt wurden drei Algorithmen aus dem Softwarepaket DifferentialEquations.jl [29]. Das explizite Euler-Verfahren ist Gegenstand dieser Arbeit. Es wurde bei der manuellen Diskretisierung verwendet, weil es die einfachste Methode der numerischen Approximation darstellt. Es ist ebenfalls als generischer Solver verfügbar und soll nun mit fortgeschritteneren Algorithmen wie Rodas4 und dem verbesserten impliziten Euler-Verfahren verglichen werden. Folgende Kriterien wurden bei der Auswahl berücksichtigt:

Stabilität: Wie in Abschnitt 4.1.4 beschrieben führt der Einsatz der expliziten Euler-Methode dazu, dass das Verfahren nur dann stabil ist, wenn es die Bedingung (4.9) erfüllt. Im Unterschied dazu liefern Rodas4 und das implizite Euler-Verfahren stabile numerische Lösungen für jede beliebige Schrittweite Δt, beide Verfahren sind mindestens A-stabil.

Adaptive Schrittweitensteuerung: Rodas4 und der ImplicitEuler-Löser verfügen außerdem über eine adaptive Schrittweitensteuerung. Dabei wird eine Fehlerabschätzung durchgeführt und die Schrittweite für jeden Zeitschritt so angepasst, dass der geschätzte lokale Fehler die vorgegebenen Toleranzen nicht überschreitet. Hierbei wurden die Standardwerte (abstol=1e-6 und reltol=1e-3) verwendet, die in der Dokumentation des Pakets DifferentialEquations [38] beschrieben sind.

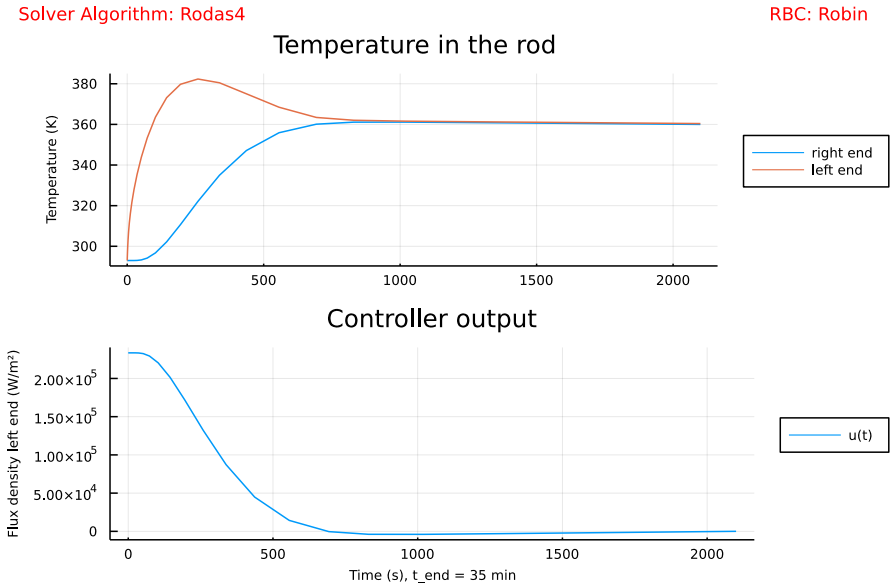


Abb. 5.2: Adaptive Schrittweitensteuerung mit Rodas4 (ohne interpolierte Werte)

In Abb. 5.2 ist das Ergebnis des Rodas4-Algorithmus für die Robin-RB am rechten Stabende zu sehen. Dabei wurden nur 25 Lösungsschritte benötigt, d.h. das Lösungsobjekt enthält 26 Elemente mit dem Startwert bei $t = 0$. Jedes dieser 26 Elemente enthält wiederum die Werte für alle Zeilen des Gleichungssystems (hier: $nx + 1$ für nx Ortspunkte und eine Zeile für das Fehlerintervall). In der Darstellung wurden nur diese 26 Werte verwendet, um zu verdeutlichen, dass der Löser für die einzelnen Schritte unterschiedliche Schrittweiten wählt. Anmerkung: Für die im Vergleichstest verwendete Neumann-RB benötigt das Verfahren 33 Lösungsschritte (34 gespeicherte Elemente).

Die Julia-eigenen Algorithmen mit adaptiver Schrittweitensteuerung speichern jedoch standardmäßig zusätzliche Informationen über die berechnete Funktion und sind in der Lage, eine kontinuierliche Lösung zu interpolieren. Beispielsweise wird für Rodas4 eine auf steife Gleichungssysteme spezialisierte freie Interpolation 3. Ordnung angegeben [37].
Anmerkung: Falls keine Interpolation erwünscht ist, kann diese Option ausgeschaltet werden, indem dem Solver das Argument $dense = false$ übergeben wird.

Eignung für steife Differentialgleichungssysteme: Die Diskretisierung von parabolischen partiellen Differentialgleichungen wie der Wärmeleitungsgleichung führt typischerweise zu steifen Differentialgleichungssystemen [41]. Daher wurden die beiden alternativen Solver entsprechend ausgewählt.

5.2.1 Verwendete Algorithmen

Euler (Explizites Euler-Verfahren)

Diese Methode ist in Abschnitt 4.1.3 beschrieben. Wie dort ausgeführt wird, handelt sich um ein Verfahren erster Ordnung (Konvergenzordnung 1). Es führt nur zu einer bedingt stabilen Lösung, was in Abschnitt 4.1.4 erläutert wird.
Die Methode ist unter den drei verwendeten Verfahren die einzige mit einer festen Zeitschrittweite. Verwendet wurden $nt = 100\,002$ Zeitschritte und $\Delta t \approx 0.021s$. Dieses geringe Δt ist bei einer Erweiterung auf 100 Ortspunkte immer noch ausreichend klein, damit das Verfahren nach (4.9) konvergiert. Mit der höheren Anzahl von Zeitschritten wird allerdings ein höherer Rechenaufwand und ein größerer Einfluss von Rundungsfehlern in Kauf genommen.

Rodas4

Hierbei handelt es sich um ein Verfahren vierter Ordnung, das besonders für steife Differentialgleichungssysteme geeignet ist. Rosenbrock-Methoden wie Rodas4 werden für kleinere Gleichungssysteme mit weniger als 50 Differentialgleichungen empfohlen [39], Rodas4 ist für linearisierte parabolische Differentialgleichungen geeignet [37].
Anmerkung: Bei einer Erweiterung auf $nx > 100$ wären andere Löser vorteilhafter. Hierfür empfiehlt die Dokumentation ESDIRK-Verfahren (explicit singly diagonally implicit Runge-Kutta) wie beispielsweise KenCarp (ebenda). Diese wurden in dieser Arbeit allerdings nicht verwendet.

Implizites Euler-Verfahren

Häufig ist in der Literatur die Aussage zu finden, dass steife Differentialgleichungen implizite Lösungsverfahren erfordern [9, S. 1]. Daher wurde dieser Algorithmus als implizites Verfahren mit einbezogen. Es ist ein Verfahren erster Ordnung, das allerdings unbedingt stabil ist (A-B-L-stabil).

5.3 Ergebnisse des Benchmarkings

Die drei Lösungsalgorithmen wurden hinsichtlich der Geschwindigkeit, mit der die Daten generiert werden können, verglichen.
Um eine Beeinflussung der Messung - beispielsweise durch Kompilierungszeiten oder im Hintergrund laufende Prozesse des Betriebssystems - zu vermeiden, wurde hierfür das Paket BenchmarkTools.jl [26] mit dem Makro *@benchmark* verwendet. Mit diesem Softwaretool wurde für jeden Solver eine Stichprobe von n=1000 Durchläufen der Funktion zur Datengenerierung mit der Neumann-RB untersucht. Die Funktion ruft immer einen Callback auf, der für jeden Zeitschritt zusätzlich den Reglerausgabewert speichert. Es werden keine Werte angezeigt und keine Plots erstellt. Für den Impliziten Euler-Algorithmus lautet diese Funktion beispielsweise

```
function generate_data(func_gen, Z₀_gen, tspan_gen,G_gen)            # for adaptive solver
    prob = ODEProblem(func_gen, Z₀_gen,tspan_gen, G_gen)
    sv_values = SavedValues(Float64, Float64)
    cb = SavingCallback((u,t,integrator)->(integrator.p[1]*(θ_ref - u[end-1])+ integrator.p[2]*u[end]), sv_values)
    solution = solve(prob, ImplicitEuler(),save_everystep=true, callback=cb)
    return sv_values, solution
end
```

Das explizite Eulerverfahren wurde mit dem Argument $saveat = sv\_steps$ verwendet, um den Speicheraufwand zu reduzieren. Dieses Argument bietet die Möglichkeit, nur die (ggf. interpolierten) Lösungen für bestimmte ausgewählte Zeitpunkte oder regelmäßige Intervalle zu speichern. Es wurde das Intervall $sv\_steps = 0.0 : 1.0 : t_f$ benutzt. Somit wird nur ein Wert pro Sekunde gespeichert (2100 Elemente).
Beim Impliziten Euler-Verfahren und Rodas4 wurde dagegen jeder mit der adaptiven Schrittweitensteuerung berechnete Wert gespeichert. Das Implizite Euler-Verfahren verwendet 154 Zeitschritte (155 Lösungselemente einschließlich des Startwertes bei $t = 0$) für die Neumann-RB.
Da der Speicheraufwand beim expliziten Euler-Algorithmus extrem viel höher ist als bei den beiden anderen Verfahren, wurde eine weitere Messung für das Rodas4-Verfahren aufgenommen, bei der ebenfalls Werte in 1s-Intervallen gespeichert werden. Dabei interpoliert der Algorithmus die zwischen den vom Solver berechneten adaptiven Zeitschritten liegenden Werte während der Lösungs-Phase. Dies geht aus der Dokumentation der Argumente *tstops*, *saveat* und *dense* hervor [37].
In Tabelle 5.2 sind die verschiedenen Algorithmen gegenübergestellt. Sie zeigt aus den Ergebnissen von BenchmarkTools den Median und den arithmetischen Mittelwert für die benötigte Zeit aus allen 1000 Durchläufen sowie die benötigte Zeit für den langsamsten und den schnellsten Durchlauf. Hinzugefügt wurde jeweils die Anzahl der gespeicherten Elemente, die dem Lösungsobjekt entnommen wurde. Dabei wird die Überlegenheit der beiden Verfahren mit adaptiver Zeitschrittweitensteuerung sehr deutlich. Betrachtet man die arithmetischen Mittelwerte, so benötigt das explizite Euler-Verfahren mehr als dreißig mal so viel Zeit wie das implizite und 76-mal so viel wie das Rodas4-Verfahren. Selbst bei gleichem Speicheraufwand benötigt der explizite Euler-Algorithmus beinahe 20-mal soviel Zeit wie Rodas4. Als Ursache ist anzunehmen, dass das explizite Euler-Verfahren - aus Stabilitätsgründen mit kleinen äquidistanten Zeitschritten - eine sehr viel größere Anzahl von Funktionsauswertungen durchführen muss, im vorliegenden Beispiel $nt = 100\,001$ (nach jedem Δt). Das

Rodas4-Verfahren dagegen benötigt nur 33 Zeitschritte. Auch wenn zusätzlich die Schrittweite ermittelt werden muss und die Werte nach einem komplexeren Verfahren berechnet werden, ist dies wesentlich effizienter. Von den drei verwendeten generischen Lösern lieferte Rodas4 die besten Ergebnisse.

	Algorithmus			
Stichprobenumfang jeweils n=1000	**Euler** (saveat=sv_steps)	**ImplicitEuler** (save_everystep=true)	**Rodas4** (save_everystep=true)	**Rodas4** (saveat=sv_steps)
Minimum t	23.250 ms	615.102 µs	294.597 µs	1.176 ms
Median t	23.817 ms	666.594 µs	303.731 µs	1.217 ms
Arithmetisches Mittel t	25.458 ms	831.093 µs	333.381 µs	1.326 ms
Maximum t	38.461 ms	16.264 ms	14.410 ms	15.085 ms
geschätzter Speicherbedarf	14.39 MiB	214.05 KiB	154.61 KiB	739.36 KiB
geschätzte Allokationen	302596	2949	2724	3741
Anzahl gespeicherter Elemente	2101	155	34	2101

Tab. 5.2: Vergleich der numerischen Löser

Unter den getesteten Verfahren hat sich Rodas4 als besonders effizient beim Lösen des vorliegenden Wärmeleitungsproblems erwiesen. Bei diesem Vergleich wurde nur die Geschwindigkeit berücksichtigt, mit der das Problem gelöst wurde, nicht die Genauigkeit des Ergebnisses. Es kann jedoch davon ausgegangen werden, dass Verfahren höherer Ordnung auch genauere Ergebnisse liefern und dass die Überlegenheit von Verfahren mit höherer Konvergenzordnung bezüglich der Effizienz mit steigenden Anforderungen an die Genauigkeit immer deutlicher hervortritt [30]. Rackauckas rät von der Verwendung von Verfahren mit weniger als Konvergenzordnung 3 generell ab (ebenda).

Mögliche Verbesserungen: Neben den Werten für die benötigte Zeit liefert der Test mit BenchmarkTools.jl auch eine Schätzung des benötigten Speichers und der vorgenommenen Allokationen, wie Tabelle 5.2 ebenfalls zeigt. Diese Werte legen nahe, dass der aktuelle Code noch verbessert werden kann. Möglicherweise fallen diese Allokationen beim Euler-Algorithmus besonders ins Gewicht, da dieser das Gleichungssystem viel häufiger auswerten muss als Verfahren mit adaptiver Schrittweite. Es gilt zu untersuchen, wo diese Allokationen auftreten und ob sie vermeidbar sind. Beispielsweise haben eine Reihe von Variablen wie $\vartheta_{ref}, \vartheta_{amb}, \lambda$ und Δx einen globalen Gültigkeitsbereich. Sie sind als *const* deklariert. Diese lassen sich nicht ohne Weiteres als Argument an die Methode ODEProblem übergeben. Möglicherweise kann hier eine Datenstruktur in der Art eines DEDataArray Abhilfe schaffen.

Im nächsten Schritt wird das explizite Eulerverfahren verwendet, um mittels ML aus den Temperaturverteilungen im Stab die Regelparameter zu erlernen, mit denen der Heizvorgang kontrolliert wurde. Die Zieldaten werden ebenfalls mit diesem Algorithmus erstellt. Verwendet wurde die mit dem Euler-Verfahren und der Neumann-Rb erzeugte Datentabelle, die auszugsweise in Tab. 5.1 zu sehen ist.

6 Ablauf und Methoden der Optimierung

In Abschnitt 5.3 ist beschrieben, wie im Verlauf der Erwärmung des Stabes verschiedene Daten - darunter die Temperaturverläufe an den nx Ortspunkten - erhoben werden. Der simulierte Heizvorgang wird dabei durch eine PI-Regelung kontrolliert, wie in Kapitel 3 erläutert.
Nun soll anhand dieser Temperaturverteilungen mithilfe von maschinellem Lernen (ML) der Proportional- und Integralanteil dieser Regelung ermittelt werden. Dabei werden drei Optimierungsalgorithmen hinsichtlich ihrer Geschwindigkeit und und Genauigkeit verglichen. Schließlich wird auch der Frage nachgegangen, wie viel Information - d.h. wie viele Ortspunkte - ein geeigneter Algorithmus benötigt, um die Parameter erlernen zu können. Zu diesem Zweck wurde das Programm pi_solver_ml geschrieben, das ebenfalls zum Projekt PI_Ode_Solv [32] gehört.

6.1 Vorgehen

Der Solver startet mit vorgegebenen Werten für K_p und K_i, die erheblich von den Zielparametern abweichen. Um die Leistung verschiedener Optimierungsalgorithmen vergleichen zu können, wurden jeweils die gleichen Startwerte für die Regelparameter verwendet.

	K_p	K_i
Startwerte	10 000	10
Zielwerte	40 000	50

Ausgehend von diesen Startwerten für die Regelparameter wird das Gleichungssystem mit dem Euler-Löser für alle nx Ortspunkte und den gesamten Zeitraum tf wiederholt berechnet und jeweils eine Vorhersage $\hat{y}$ über die Lösung generiert. Für die Vorhersage der Wärmeleitung im Stab und über die Ränder wird erneut das mit der vertikalen Linienmethode erstellte System gewöhnlicher Differentialgleichungen herangezogen, wie es in Abschnitt 5.1 beschrieben ist.
Für jede dieser Modellprognosen wird anschließend anhand einer Kostenfunktion der Fehler dieser Vorhersage ermittelt, d.h. wie weit sie von den Zieldaten y aus der Datentabelle abweicht. Diese Kostenfunktion $J = f(K_p, K_i)$ wird in Abschnitt 6.2 beschrieben.
Mithilfe eines geeigneten Optimierungsverfahrens sollen die Parameter K_p und K_i schrittweise dahingehend verändert werden, dass der Fehler minimiert wird. Hier werden also nicht Gewichte und Bias eines neuronalen Netzes optimiert, sondern die beiden unbekannten Regelparameter der Funktion, welche das Wärmeleitungsproblem beschreibt.
Die getesteten Optimierer werden in Abschnitt 6.3 kurz vorgestellt.

Zieldaten: Genutzt werden nur die Temperaturdaten für die ausgewählten Ortspunkte. Außerdem wird die rechte RB aus der Datentabelle für das ML übernommen. Die Wer-

te des Fehlerintegrals oder des Reglerausgangs sind dem ML-Algorithmus dagegen nicht bekannt. Verwendet wurden die mit dem expliziten Euler-Verfahren und für die Neumann-RB erhobenen Daten, d.h. zur Verfügung stehen die Werte für nx=11 Ortspunkte. Für jeden Ortspunkt sind 2101 Temperaturwerte in 1s-Intervallen gespeichert, da die Tabelle mit $sv\_steps = 0.0 : 1.0 : t_f$ und $t_f = 2100s$ aufgenommen wurde. Für einen Einblick in die Zieldaten sei erneut auf Tab. 5.1 verwiesen.
Die Daten der gewünschten Ortspunkte werden zunächst - wiederum unter Verwendung des Moduls DelimitedFiles [21] - aus der tsv-Datei ausgelesen und in einem zweidimensionalen Array gespeichert (bzw. in einem Zeilenvektor, falls nur ein Ortspunkt verwendet wird).

6.2 Kostenfunktion

Das auf diese Weise aus der Datentabelle erstellte Array wird dem ML-Algorithmus als Ziel-Lösung y vorgegeben.
In Abhängigkeit von den Werten für K_p und K_i, die der Optimierungsalgorithmus in einem iterativen Prozess liefert, wird nun mit dem bereits bekannten Modell aus Absatz 5.1 ein Lösungsobjekt erstellt. Von diesem Lösungsobjekt werden ebenfalls die Temperaturverläufe der oben ausgewählten Ortspunkte als Prognose $\hat{y}$ verwendet.
Die Kostenfunktion dient nun dazu, den Fehler dieser Prognose im Vergleich zu den Zieldaten zu quantifizieren. Dies wurde mithilfe der Least Squares Loss Funktion erreicht.

$$J(y, \hat{y}) = \sum_{k=1}^{n} (\hat{y}_k - y_k)^2 \quad , \; J : \Omega \to \mathbb{R}_0^+$$

Ω bezeichnet dabei die Menge aller möglicher Lösungen für y.
Von den Werten der Vorhersage werden die korrespondierenden Werte der Zieldaten subtrahiert, d.h. die Temperaturdaten, die jeweils für den gleichen Orts- und Zeitpunkt erhoben wurden. Um zu vermeiden, dass positive und negative Fehler einander ausgleichen, wird diese Differenz quadriert. Die quadratische Funktion hat gegenüber der Betragsfunktion den Vorteil, dass sie an allen Stellen stetig differenzierbar und streng konvex ist. Dies ist für den Einsatz von Gradientenverfahren (Abschn. 6.3) nützlich.
Anschließend wird die Summe über alle n verwendeten Werte aus den Lösungsobjekten berechnet. Die Kostenfunktion liefert somit für jede Prognose einen skalaren Rückgabewert.
Die so gebildete Kostenfunktion $J(y, \hat{y}) = f(K_p, K_i)$ ist - da sie aus strikt konvexen Summanden gebildet wird - wiederum eine steng konvexe Funktion. Zu lösen ist ein konvexes Optimierungsproblem [2, S. 7]. Da die Parameter theoretisch jeden Wert annehmen können und keine Rand- oder Nebenbedingungen erfüllen müssen, handelt es sich um eine Optimierung ohne Nebenbedingungen.
Aufgrund von

$$J(y, \hat{y}) = \begin{cases} 0 & \text{für } \hat{y} = y \\ > 0 & \text{sonst} \end{cases}$$

muss die Funktion über genau ein (lokales und globales [25, S. 7]) Optimum bei $\hat{y} = y$ verfügen. Es reicht also aus, mit Methoden der lokalen Optimierung ein lokales Minimum zu finden, da dieses zugleich das globale Optimum darstellt.

Die Kostenfunktion für das vorliegende Wärmeleitungsproblem mit den Zielparametern $K_p = 40\,000$ und $K_i = 50$ für den gesamten beim ML genutzten Bereich ist in Abb. 6.1 abgebildet. In Abb. 6.2 ist der Bereich um das Minimum der Kostenfunktion noch einmal detaillierter dargestellt um zu zeigen, dass in der Nähe des Minimums kleine Veränderungen von K_i einen großen Einfluss auf den Fehler haben. In dieser Arbeit werden die Begriffe Kosten- und Verlustfunktion (engl.: loss function) synonym verwendet.

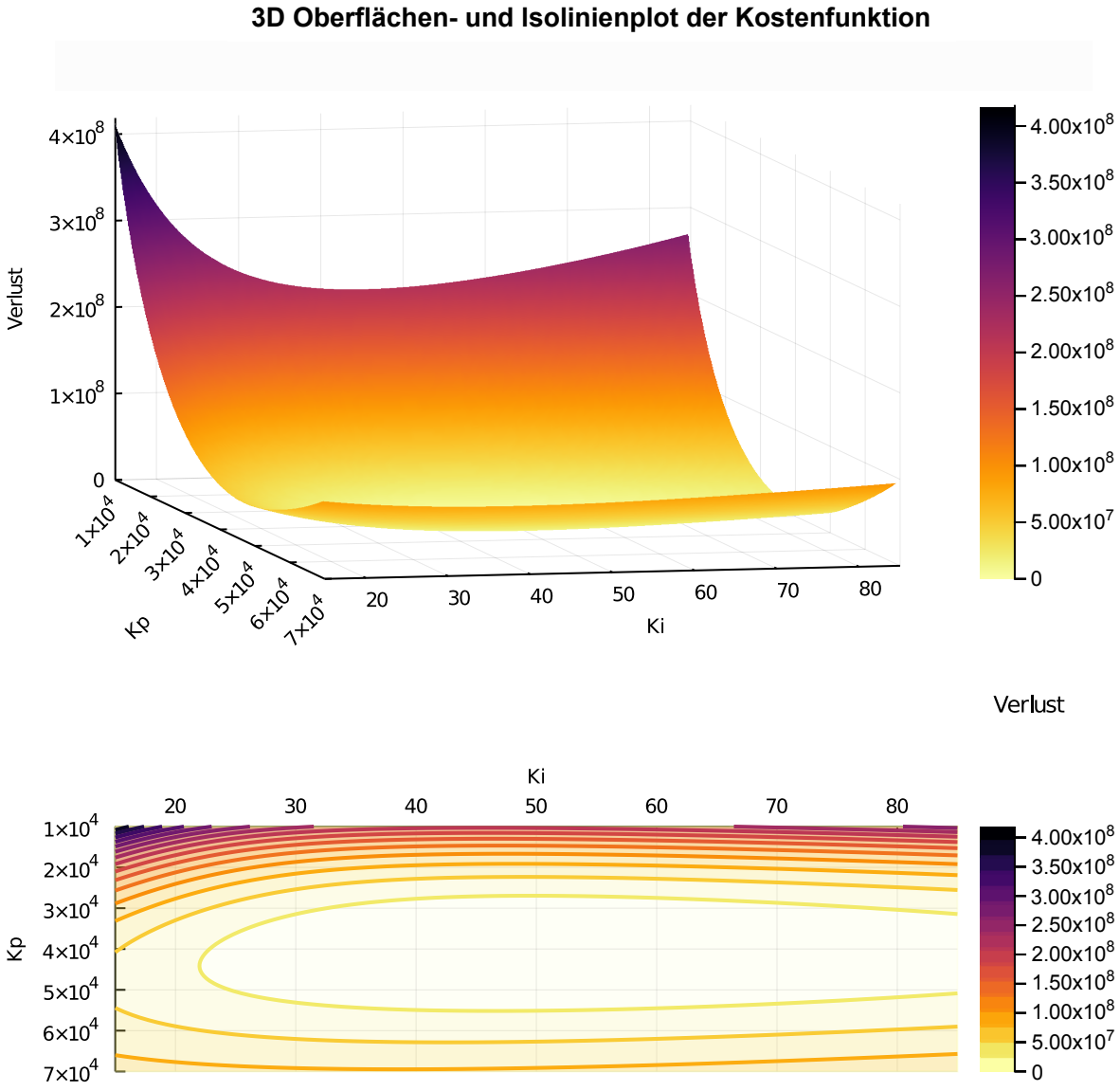


Abb. 6.1: Kostenfunktion, gesamter genutzter Bereich, erstellt mit Plots.jl und dem Backend GR

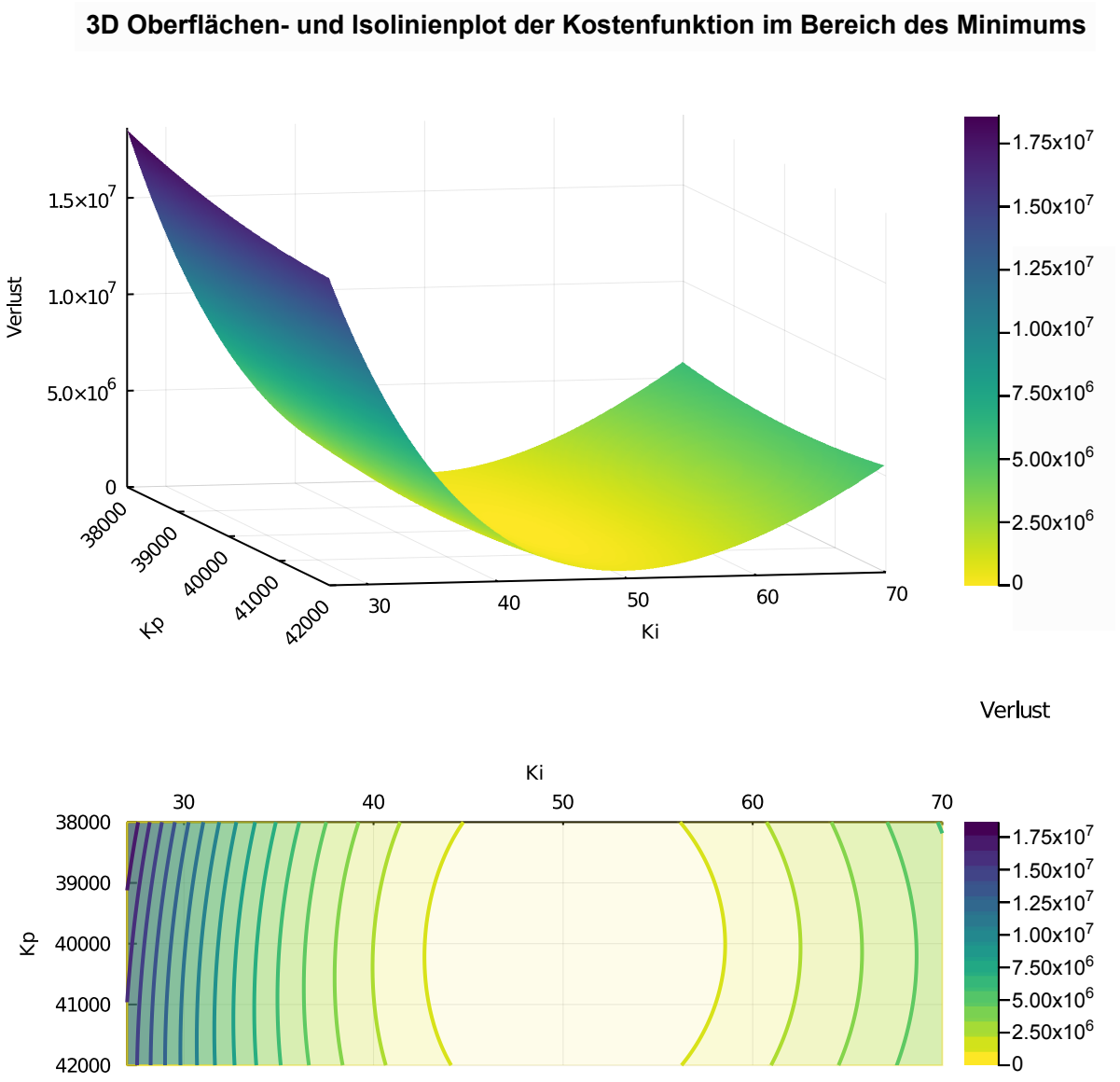


Abb. 6.2: Kostenfunktion im Bereich des Minimums, erstellt mit Plots.jl und dem Backend GR

6.3 Verwendete Optimierungsverfahren

Ziel des Optimierungsverfahrens ist es, die Werte für die Parameter K_p und K_i zu finden, die den Fehler minimieren. In Anlehnung an [7, S. 80] und mit $\boldsymbol{g} = \begin{pmatrix} K_p \\ K_i \end{pmatrix}$ ist der folgende Parametervektor gesucht

$$\boldsymbol{g}^* = arg\,min\, J(\boldsymbol{g}) \ .$$

Es wurden drei verschiedene gradientenbasierte Optimierungsverfahren getestet. Darunter der weithin gebräuchliche Adam-Optimierer aus dem Paket Flux.jl [13]. Außerdem der Gradient Descent-Algorithmus sowie das BFGS-Verfahren aus dem Paket Optim.jl [23] [22]. Die

numerischen Löser dieses Softwarepakets eignen sich besonders für Optimierungsprobleme ohne Nebenbedingungen und lokale Optimierung [27].

6.3.1 Gradient Descent (Gradientenabstieg)

Der Gradient der Kostenfunktion in Bezug auf $\mathbf{g}$ an einer Stelle g_i

$$\nabla_{\boldsymbol{g}} J(\boldsymbol{g_i}) = \begin{pmatrix} \frac{\partial J(\boldsymbol{g_i})}{\partial K_p} \\ \frac{\partial J(\boldsymbol{g_i})}{\partial K_i} \end{pmatrix} \tag{6.1}$$

weist in die Richtung, in der von Punkt g_i aus der Funktionswert am steilsten ansteigt. Der Algorithmus soll allerdings die Richtung finden, in welcher der Funktionswert der Kostenfunktion minimiert wird, die Richtung des steilsten Abstiegs. Dazu müsste der Vektor in die entgegengesetzte Richtung zeigen, wie in [7, S. 83] gezeigt wird. Beim Gradient Descent-Algorithmus werden daher für jede Modellprognose die Parameter in Richtung des negativen Gradienten aktualisiert, d.h.

$$\boldsymbol{g_{i+1}} = \boldsymbol{g_i} - \alpha \ \nabla_{\boldsymbol{g}} J(\boldsymbol{g_i}) \tag{6.2}$$

mit einem positiven Skalar als Lernrate α. Die Lernrate bestimmt dabei die Größe der Schritte, mit denen sich die Parameter den optimalen Werten $\boldsymbol{g}^*$ annähern. [7, S.83f.]
Die Wahl der Lernrate ist entscheidend für den Erfolg dieses Verfahrens. Bei einer zu großen Lernrate kann das Minimum verpasst werden. Ein kleiner Wert erhöht die Genauigkeit der Annäherung, gleichzeitig aber auch die Anzahl der benötigten Iterationen und damit die für das Auffinden benötigte Zeit.
Der hier eingesetzte Algorithmus Gradient Descent verwendet - im Unterschied zum klassischen stochastischen Gradientenabstieg - keine feste Lernrate, sondern passt diese im Rahmen einer Liniensuche bei jedem Schritt so an, dass sich der Funktionswert in ausreichendem Maß verringert. Hierauf wurde kein Einfluss genommen, sondern der Standard-Algorithmus (Hager-Zhang, [27, Stichwort: Line search] verwendet.

In Abb. 6.3 ist das Gradientenfeld der in dieser Arbeit verwendeten Kostenfunktion $J(K_p, K_i)$ dargestellt. Die Gradienten an den ausgewählten Stellen wurden mithilfe von Zygote [12] berechnet. Auffällig ist, dass es sich um ein eher flaches Minimum mit - zumindest im näheren Umfeld des Minimums - geringer Krümmung in K_p-Richtung handelt, was auch aus Abb. 6.2 deutlich wird.

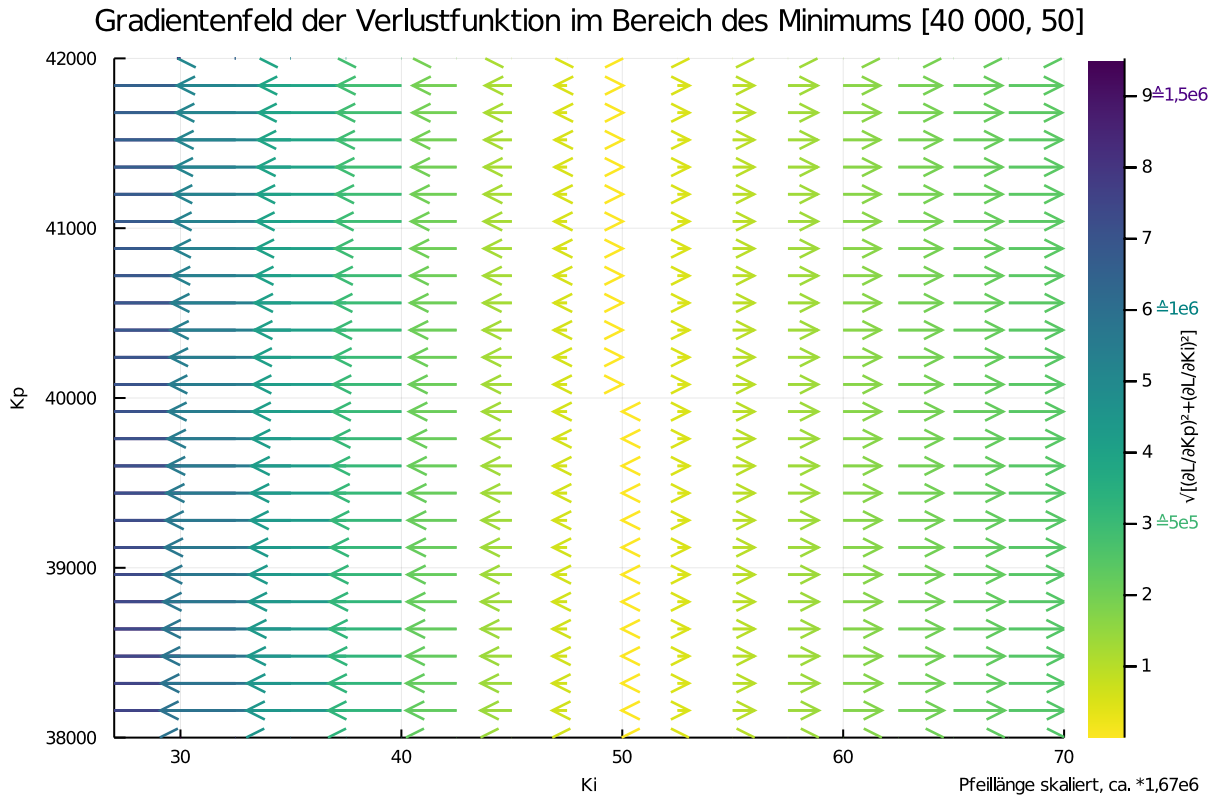


Abb. 6.3: Gradientenfeld im Bereich des Minimums, erstellt mit Plots.jl und dem Backend GR

6.3.2 Adam (Adaptive Moment Estimation)

Der Adam-Algorithmus ist ein sehr etabliertes Verfahren im maschinellen Lernen. Es verbindet die Ideen zweier anderer Optimierungsmethoden, dem Momentum Optimierer und RMS Prop [7, S. 305].
Während das Gradientenabstiegsverfahren die gleiche skalare Lernrate für alle Parameter verwendet, berechnet der Adam-Algorithmus unabhängige Lernraten für jeden einzelnen Parameter. Dabei wird nicht nur der aktuelle lokale Gradient verwendet, sondern zusätzlich sowohl der Durchschnitt der Gradienten der vorangegangenen Iterationen (wie beim Momentum Optimierer), als auch den Durchschnitt der quadrierten Gradienten (wie bei RMSProp). Verwendet werden gleitende Mittelwerte mit exponentiell abnehmender Wichtung. Außerdem wird bei den Momentvektoren erster und zweiter Ordnung bei Trainingsbeginn eine Bias-Korrektur vorgenommen. [7, S. 305f.]).
Neben der Anfangs-Schrittweite α wird der Algorithmus daher mit den Hyperparametern $\beta_1, \beta_2 \in [0, 1)$ als Abklingraten für die beiden Mittelwere gesteuert. Hier wurden die Standardeinstellungen [15, S. 2]

α	$= 0.001$	Lernrate	
β_1	$= 0.9$	exponentielle Abklingrate der Momentschätzung erster Ordnung	sowie
β_2	$= 0.999$	exponentielle Abklingrate der Momentschätzung zweiter Ordnung	

verwendet.

6.3.3 BFGS (Broyden–Fletcher–Goldfarb–Shanno-Verfahren)

Hierbei handelt es sich um eine Quasi-Newton-Methode, die besonders für konvexe Optimierungsprobleme geeignet ist. Für globale Optimierungen wird es gelegentlich auch in Kombination mit dem Adam-Algorithmus verwendet [42, S. 49]. Das BFGS-Verfahren benötigt in der Regel deutlich weniger Schritte für die Annäherung an das Minimum als die anderen Verfahren. Das bedeutet aber nicht zwangsläufig, dass es auch weniger Zeit braucht [42, S. 25].
Während die beiden zuvor beschriebenen Verfahren die partiellen Ableitungen erster Ordnung zur Bestimmung der Parameter für die nächste Iteration berücksichtigen, verwenden Newton-Methoden zusätzlich auch die Ableitungen zweiter Ordnung, die das Krümmungsverhalten der Kostenfunktion beschreiben. Dazu muss allerdings für jeden Datenpunkt die inverse Hesse-Matrix berechnet werden, was einen hohen Rechen- und Speicheraufwand bedingt. [7, S. 307ff.])
Quasi-Newton-Verfahren berechnen jedoch nicht die tatsächliche Hesse-Matrix. Stattdessen nähern sie diese durch eine positiv definite Matrix P an, die in jedem Iterationsschritt auf der Basis der für die letzten Schritte berechneten Informationen (Änderung der Gradienten) aktualisiert wird [27, Stichwort: (L)BFGS]. Der Algorithmus ist

$$\boldsymbol{g_{i+1}} = \boldsymbol{g_i} - \alpha \; P^{-1} \; \nabla_{\boldsymbol{g}} \, J(\boldsymbol{g_i}) \tag{6.3}$$

mit	$\boldsymbol{g}_i$	aktueller Punkt, $\boldsymbol{g_i} = (K_{p_{(i)}}, K_{i_{(i)}})$
	$\boldsymbol{g_{i+1}}$	neuer Punkt, aktualisierte Parameter $K_{p_{(i+1)}}$ und $K_{i_{(i+1)}}$
	α	Lernrate
	P^{-1}	approximierte Hesse-Matrix .

Wie beim Gradient Descent-Verfahren passt auch dieser Algorithmus die Lernrate mit einer Liniensuche für jeden Schritt an. Auch hier wurde das Standardverfahren verwendet.

6.3.4 Sensitivitätsanalyse

Mittels Sensitivitätsanalyse wird bestimmt, wie empfindlich die Lösung auf Veränderungen der einzelnen Parameter reagiert [36]. Für die Berechnung der partiellen Ableitungen der Lösung in Bezug auf die Parameter an einer Stelle bietet Julia eine Vielzahl verschiedener Optionen. Im Rahmen dieser Arbeit wurde für alle Tests von Optimierungsverfahren der Sensitivitätsalgorithmus $ForwardDiffSensitivity$ mit dem Argument $convert\_tspan = true$ verwendet. Das Argument $convert\_tspan = true$ wird bei der Verwendung mit Callbacks benötigt.
$ForwardDiffSensitivity$ ist ein Verfahren zur diskreten lokalen Sensitivitätsanalyse, das mittels automatischer Differenzierung umgesetzt wurde. Es wird als schnellster Algorithmus für kleine Systeme mit weniger als 100 Parametern und Gleichungen (ODEs) empfohlen [18, S. 1].

7 Ergebnisse des Maschinellen Lernens

7.1 Vergleich der Optimierungsalgorithmen unter Verwendung des Euler-Lösers

Hierfür wurden die Temperaturverläufe aller $nx = 11$ Ortspunkte verwendet. Die Zieldaten wurden - wie in Abschnitt 6.1 beschrieben - der mit dem expliziten Euler-Algorithmus erstellten Lösung entnommen. Auch die Vorhersagen wurden für diesen Vergleich mit dem expliziten Euler-Verfahren berechnet.
Alle Verfahren starten mit den Standardeinstellungen. Das bedeutet für die beiden Liniensuchverfahren Gradient Descent und BFGS eine Startlernrate von $\alpha = 1$ (dem Workspace entnommen) und für den Adam-Algorithmus die bereits in Abschnitt 6.3.2 erwähnten Startwerte von $\alpha = 0.001$ für die Lernrate sowie $\beta_1 = 0.9$ und $\beta_2 = 0.999$ für die Abklingraten der Momentschätzungen. Die Verfahren Adam und Gradient Descent erhielten außerdem das Argument $maxiters = 1000$. Da beide Verfahren sich zwar stetig, aber sehr kleinschrittig annähern, wurde nach 1000 Iterationen abgebrochen. Für den Adam-Algorithmus wurde ein weiterer Versuch mit einer größeren Lernrate und geringerer Dämpfung für die Momentschätzungen aufgenommen. Tabelle 7.1 zeigt die jeweils am Ende bzw. bei Abbruch des Prozesses erreichten Werte. Alle Verfahren nähern sich kontinuierlich an, so dass die letzten Werte auch dem minimalen erreichten Fehler entsprechen. In der Fehlerzeile sind die Funktionswerte der Kostenfunktion

$$J(y, \hat{y}) = \sum_{k=1}^{n} (\hat{y} - y_k)^2$$

am Ende der Optimierung festgehalten. Zu Beginn (erste Prognose mit den für K_p und K_i vorgegebenen Startwerten) des Prozesses beträgt der Funktionswert $J(y, \hat{y}) \approx 5.88e8$.

Endwerte	Gradient Descent	Adam (α=0.001, β_1=0.9, β_2=0.999)	Adam (α=1, β_1=0.4, β_2=0.5)	BFGS
Iterationen	1000	1000	1000	10
K_p	≈ 22 109.41	≈ 10 000.98	≈ 10 999.72	39 999.99999999998
K_i	≈ 48.89	≈ 10.98	≈ 45.86	49.99999999999985
Fehler	≈ 5.39e7	≈ 5.47e8	≈ 1.94e8	≈ 8.84e-22
benötigte Zeit (ø aus 100 Durchläufen)	- (Abbruch)	- (Abbruch)	- (Abbruch)	2,536s

Tab. 7.1: Vergleich der Optimierungsalgorithmen

Bei dieser speziellen Aufgabe zeigt sich eine deutliche Überlegenheit des BFGS-Algorithmus. Dieser Algorithmus ist besonders gut für steife Differentialgleichungssysteme und deterministische Kostenfunktionen geeignet. Da er auch die zweiten Ableitungen aus der approximierten Hesse-Matrix nutzt, konvergiert die Lösung bei dieser kleinen Zahl von Parametern

sehr schnell gegen das Minimum. Abbildung 7.1 zeigt den Verlauf der Annäherung an das Minimum der Kostenfunktion.

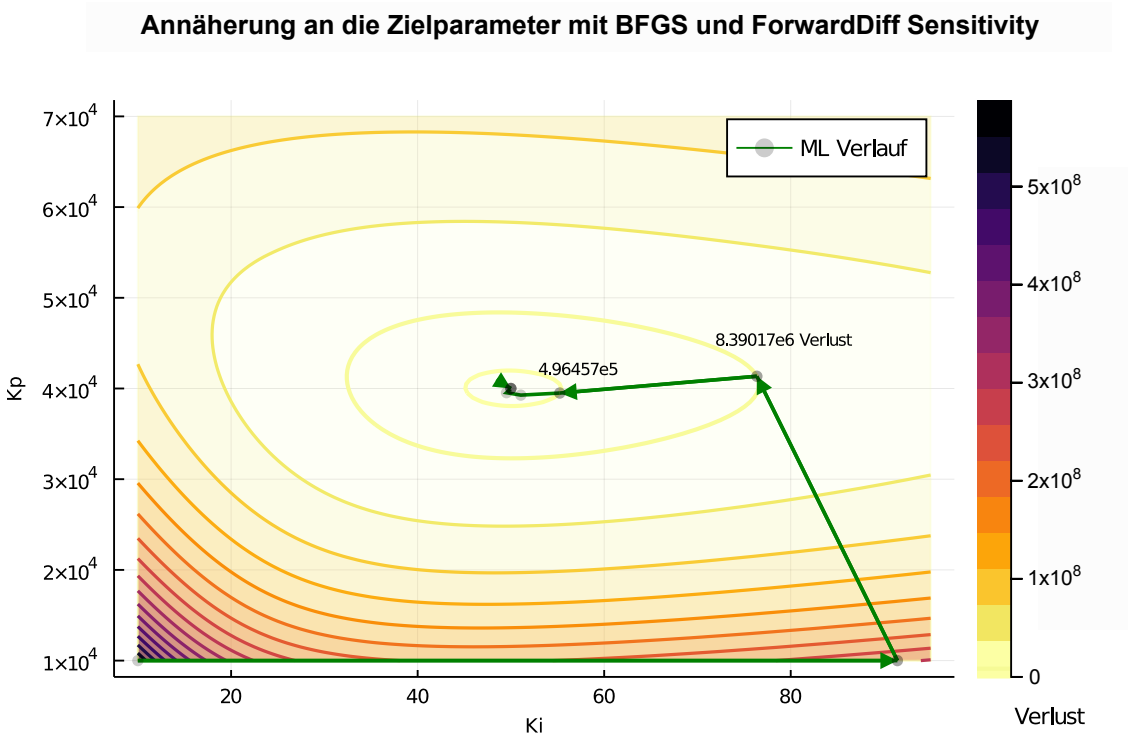


Abb. 7.1: Annäherung des BFGS-Algorithmus an das Minimum der Kostenfunktion

Iterationen	Kp	Ki	$\sum_{k=1}^{n}(\hat{y}_k - y_k)^2$	∂L/∂Kp	∂L/∂Ki
1.0	≈ 10000.00	≈ 10.00	≈ 5.88e8	≈ -46419.58	≈ -4.45e7
2.0	≈ 10000.09	≈ 91.44	≈ 2.87e8	≈ -30498.77	≈ 2.58e6
3.0	≈ 41348.62	≈ 76.37	≈ 8.39e6	≈ 255.87	≈ 497991.96
4.0	≈ 39497.20	≈ 55.24	≈ 496456.84	≈ -128.84	≈ 169531.16
5.0	≈ 39270.21	≈ 51.06	≈ 88100.17	≈ -188.14	≈ 37319.52
6.0	≈ 39537.57	≈ 49.53	≈ 32567.88	≈ -121.39	≈ -19489.16
7.0	≈ 40004.90	≈ 49.99	≈ 4.09	≈ 40004.90	≈ -274.25
8.0	39999.9992011812	49.99999636325434	≈ 3.39e-7	≈ -0.00022	≈ -0.14
9.0	40000.00000007049	49.99999999989074	≈ 8.40e-16	≈ 1.78e-8	≈ -3.89e-6
10.0	39999.99999999998	49.99999999999985	≈ 8.84e-22	≈ -1.42e-11	≈ -1.83e-9

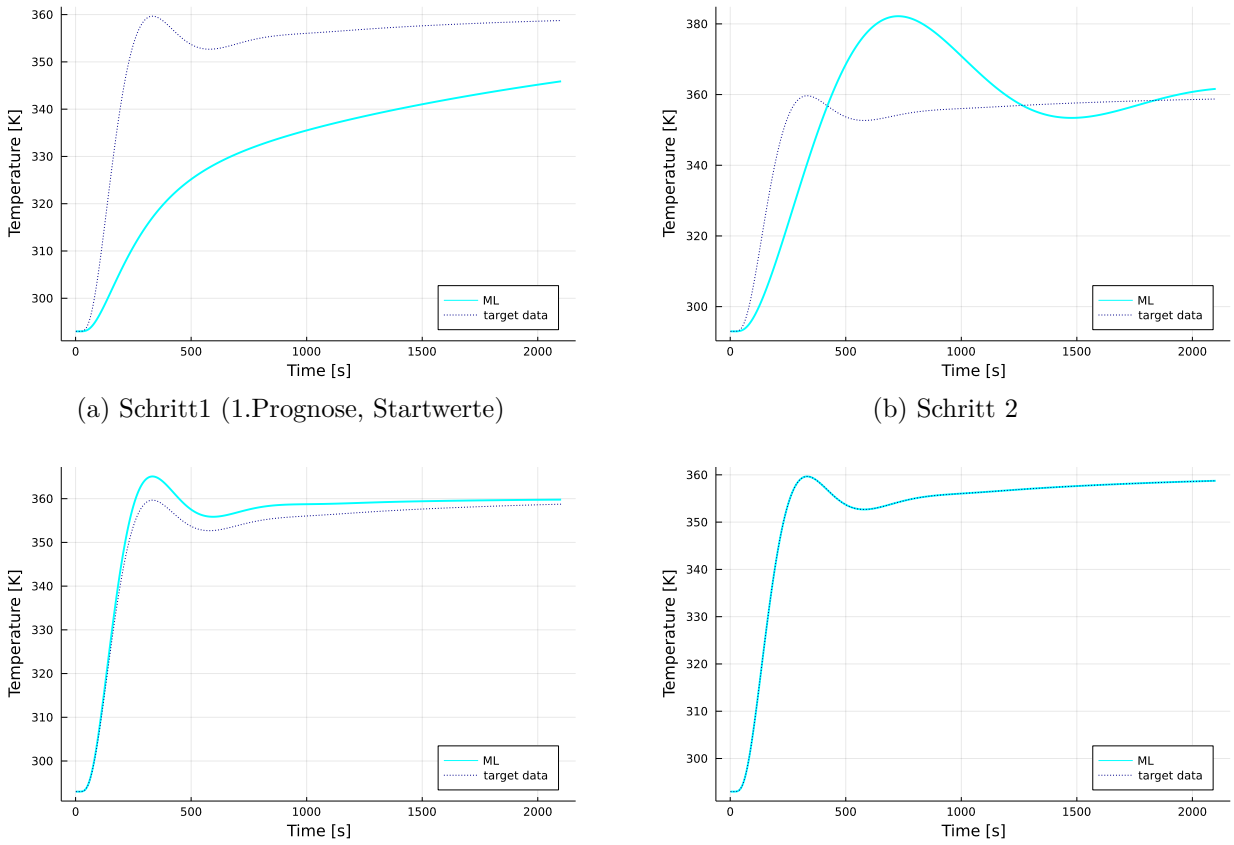
Daten erzeugt mit: Euler(), saveat: 1/s, RBC:Neumann, Kp=40000 Ki=50.0

ML mit: Euler(), saveat: wie Daten, RBC:Neumann (wie Daten), Optim.: BFGS(), sensealg=ForwardDiffSensitivity(convert_tspan=true), Start-Parameter: Kp=10000 Ki=10

Tab. 7.2: Parameter, Funktionswert der Kostenfunktion und Gradienten für alle Datenpunkte des BFGS

In Tabelle 7.2 sind die Parameter, der Funktionswert der Kostenfunktion und die partiellen Ableitungen bezüglich beider Parameter für alle Datenpunkte dieser Annäherung zusammengestellt. Parameter und Fehler sind tatsächlich verwendete, mittels Callback ausgegebene Werte, während die partiellen Ableitungen der Kostenfunktion an den Datenpunkten

nachträglich mit Zygote [12] berechnet wurden. In Abb. 7.2 ist zu sehen, wie sich der Temperaturverlauf am Stabende in wenigen Iterationen an die vorgegebenen Zielwerte annähert.



(a) Schritt1 (1.Prognose, Startwerte)

(b) Schritt 2

(c) Schritt 3

(d) Schritt 10

Abb. 7.2: Temperatur am Stabende ($x = x_{11}$) im Verlauf des ML mit BFGS und ForwardDiffSensitivity

Der Adam-Algorithmus hat im Unterschied dazu seine Stärken im Training tiefer neuronaler Netze, wobei nicht die tatsächlichen Parameter des Modells optimiert werden, welches das Problem mathematisch und physikalisch beschreibt. Stattdessen werden Gewichte und Bias eines neuronalen Netzes in Bezug auf ein Testset optimiert, um die Leistung dieses Neuronalen Netzes für gleichartige Probleme zu verbessern. Es handelt sich also eher um eine indirekte Optimierung [7, S. 272].

Es fällt auf, dass sowohl Gradient Descent als auch die Adam-Variante mit großer Lernrate und geringerer Dämpfung der Momentschätzungen große Schwierigkeiten bei der Annäherung des K_p-Wertes haben. Dies mag verwundern, betrachtet man Abb.6.1 oder auch Abb. 7.1. Hier entsteht der Eindruck, der steilste Abfall müsse - orthogonal zu den Konturlinien - etwa in diagonaler Richtung erfolgen. Somit könnte man auch für den K_p-Wert eine deutlichere Korrektur in Richtung des negativen Gradienten erwarten. Man beachte allerdings die unterschiedliche Skalierung von X- und Y-Achse in der Abbildung. Betrachtet man dagegen den ersten Datenpunkt aus Tabelle 7.2 hinsichtlich der partiellen Ableitungen der Kostenfunktion (hier mit L für *loss* bezeichnet) wird deutlich, dass der Gradient [I] am Startpunkt tatsäch-

[I] Man beachte den Vorzeichenwechsel bei den partiellen Ableitungen, da die Parameter in Richtung des

lich sehr viel stärker in K_i-Richtung weist als in K_p-Richtung. Aufgrund der streng konvexen Form der Kostenfunktion muss deren partielle Ableitung in Bezug auf K_p bei Annäherung an das Minimum betragsmäßig kontinuierlich noch weiter abnehmen. Dieser insgesamt eher flache Verlauf der Kostenfunktion in K_p-Richtung erschwert möglicherweise das Auffinden des K_p-Wertes für die beiden Verfahren erster Ordnung.

7.2 Reduktion der Zieldaten für den BFGS-Algorithmus

Für den Vergleich der Optimierungsverfahren wurden die Temperaturverläufe aller $nx = 11$ Ortspunkte verwendet. Im nächsten Schritt soll festgestellt werden, ob der BFGS-Algorithmus auch mit weniger Daten - d.h. weniger Ortspunkten - das Minimum der Kostenfunktion auffinden kann. In Tabelle 7.3 ist das Ergebnis des Optimierungsalgorithmus für verschiedene Kombinationen von Ortspunkten erfasst.

Messpunkte	Zeit (s)	$\frac{1}{N_x} J(y, \hat{y})$	Kp	Ki
$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}$	4.32	8.04E-23	40000.00	50.00
$x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}$	4.49	1.12E-23	40000.00	50.00
$x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}$	4.68	2.81E-24	40000.00	50.00
$x_4x_5x_6x_7x_8x_9x_{10}x_{11}$	4.69	9.07E-23	40000.00	50.00
$x_5x_6x_7x_8x_9x_{10}x_{11}$	4.59	2.86E-22	40000.00	50.00
$x_6x_7x_8x_9x_{10}x_{11}$	4.54	9.97E-24	40000.00	50.00
$x_7x_8x_9x_{10}x_{11}$	4.82	1.80E-24	40000.00	50.00
$x_8x_9x_{10}x_{11}$	4.53	2.42E-23	40000.00	50.00
$x_9x_{10}x_{11}$	4.95	9.47E-22	40000.00	50.00
$x_{10}x_{11}$	4.47	1.55E-23	40000.00	50.00
x_{11}	59.72	1.71E+09	63542.70	14.83
x_1	26.62	4.33E+10	36276.42	46.11
x_6	7.54	4.23E+10	60354.72	23.88
x_1x_{11}	4.33	9.00E-22	40000.00	50.00
x_1x_2	4.42	6.54E-22	40000.00	50.00
$x_1x_6x_{11}$	4.62	3.79E-22	40000.00	50.00

Daten erzeugt mit: Euler(), saveat: 1/s, RBC: Neumann, Kp=40000 Ki=50.0
ML mit: Euler(), saveat: wie Daten, RBC: Neumann, kein cb, Optim.: BFGS( ) mit sensealg=ForwardDiffSensitivity(convert_tspan=true), Start-Parameter: Kp=10000 Ki=10

Tab. 7.3: BFGS Optimierung mit verschieden Kombinationen von Ortspunkten

negativen Gradienten aktualisiert werden.

In der ersten Spalte ist die für den Optimierungsprozess benötigte Zeit festgehalten. Hierfür wurde allerdings das arithmetische Mittel aus nur 20 Durchläufen festgehalten, weshalb die Werte nur als grobe Orientierung, nicht als exakte Messung zu verstehen sind.
In der zweiten Spalte wurde der Fehler des Endergebnisses für die jeweilige Kombination von Ortspunkten festgehalten, d.h. der Funktionswert der Kostenfunktion für die letzte Prognose. Um diese Werte vergleichbar zu machen wurde der Fehler jeweils auf die Anzahl der verwendeten Ortspunkte bezogen

$$\frac{1}{N_x} J(y, \hat{y}) = \frac{1}{N_x} \sum_{k=1}^{n} (\hat{y}_k - y_k)^2$$

mit	N_x	Anzahl der verwendeten Ortspunkte,
	$n = N_x \cdot N_t$	Anzahl der Werte insgesamt, mit
	N_t	Anzahl Zeitpunkte (hier $N_t = 2101$) .

Abbildung 7.3 zeigt diese Werte nochmals grafisch.
In den letzten beiden Spalten von Tabelle 7.3 sind schließlich die erlernten Werte für K_p und K_i am Ende der Optimierung festgehalten.
Es zeigt sich, dass bereits die Temperaturverläufe von zwei Ortspunkten ausreichen, um die Parameter aus diesem Datensatz mit dem BFGS-Algorithmus mit hoher Genauigkeit zu ermitteln.

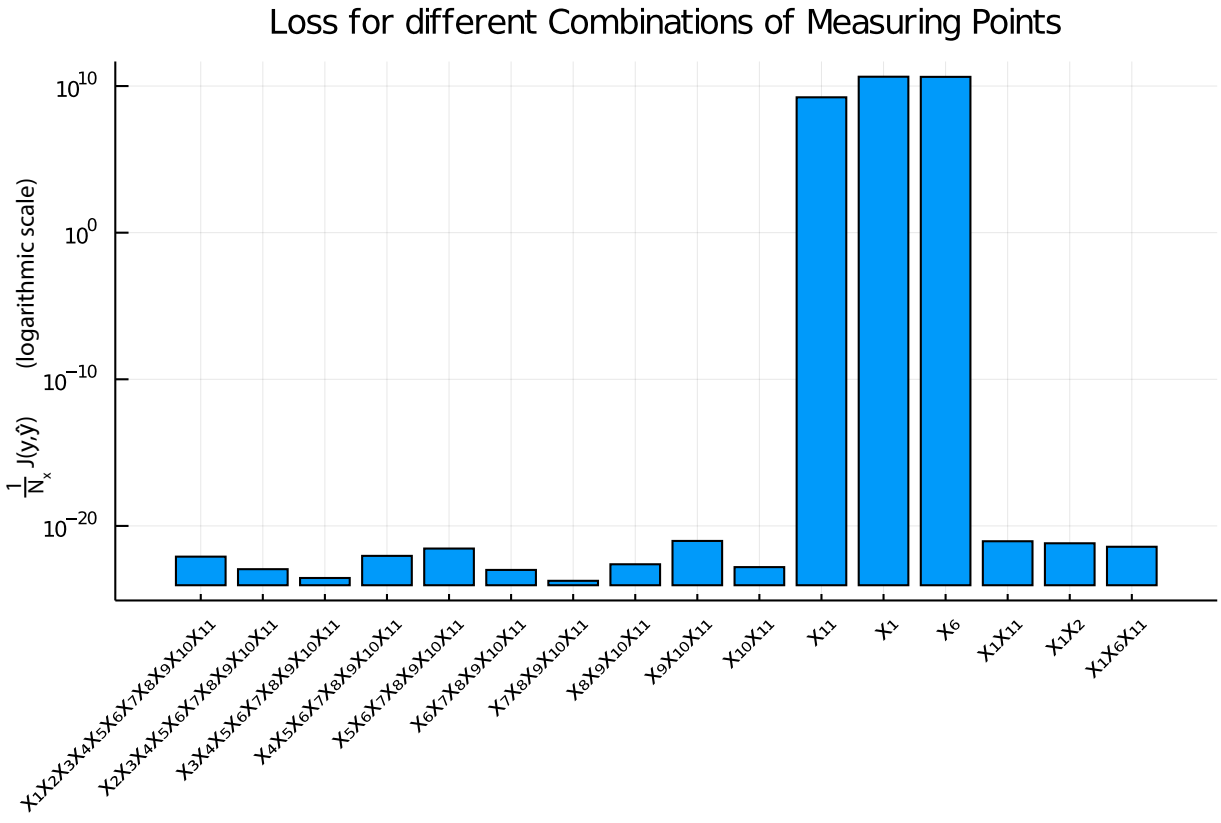


Abb. 7.3: Fehler nach Optimierung mit BFGS für verschiedene Kombinationen von Ortspunkten

7.3 Umrisshafter Vergleich mit dem Rodas4-Algorithmus

Abschließend wurde zum Vergleich die Optimierung mit der Datentabelle des adaptiven Rodas4-Lösers durchgeführt. Die Vorhersagen wurden ebenfalls mit Rodas4 erstellt. Hierbei wurden wiederum alle $nx = 11$ Ortspunkte verwendet. Allerdings benötigt der adaptive Algorithmus nur 34 Zeitpunkte (33 Lösungsschritte), was in Abschnitt 5.2 beschrieben ist. Im Gegensatz dazu enthält die Datentabelle des expliziten Euler-Algorithmus 2101 Zeitpunkte. Um die Fehler der beiden numerischen Löser vergleichbar zu machen, wurden die Funktionswerte der Kostenfunktion am Ende der Optimierung hier auf die verwendete Anzahl von Zeitpunkten bezogen. Die Ergebnisse zeigt Tabelle 7.4.

Numerischer Löser		**Optimierungsverfahren**			
		Gradient Descent	**Adam** (α=0.001, β_1=0.9, β_2=0.999)	**Adam** (α=1, β_1=0.4, β_2=0.5)	**BFGS**
Euler explizit	K_p	22 109.401	10000.98	10 999.72	40 000
	K_i	48.89	10.98	45.86	50
	$\frac{1}{N_t}J(y,\hat{y})$	25 658.97	260 409.92	92 511.3	4.21E-25
Rodas4	K_p	23 001.04	10 000.99	10 999.81	40 156.03
	K_i	67.52	10.98	73.44	50.20
	$\frac{1}{N_t}J(y,\hat{y})$	45 425.77	304 269.60	158 160.43	3.70
beachte: Eulerverfahren N_t = 2101 Rodas4 N_t =34					
Zielwerte: Kp = 40 000, Ki = 50 Startwerte: Kp = 10 000, Ki = 10					

Tab. 7.4: Erster Vergleich der Optimierungsverfahren mit dem Euler- und dem Rodas4-Löser

Auch hier liefert der BFGS-Optimierer das beste Ergebnis im Vergleich der drei Optimierungsalgorithmen. Allerdings erreicht die Annäherung an die Zielwerte ein nicht ganz so hohes Maß wie beim Euleralgorithmus (Fehler in der Größenordnung 10^-25 für 11 Ortspunkte). Inwieweit dies jedoch auf den verwendeten Lösungsalgorithmus zurückzuführen ist und inwieweit die Größe des verwendeten Datensatzes ($N_t = 34$ Zeitpunkte bei Rodas4 gegenüber $N_t = 2101$ beim Euler-Verfahren) das Ergebnis beeinflusst, kann anhand dieses orientierenden Vergleichs nicht festgestellt werden. Es erscheint jedoch wahrscheinlich, dass die sehr gute Leistung des BFGS-Algorithmus bei der Optimierung der Parameter für dieses System auch auf andere numerische Löser übertragbar ist.

7.4 Fazit zum BFGS-Verfahren

Der BFGS-Algorithmus findet im maschinellen Lernen eher selten Anwendung. Er ist für stochastische Differentialgleichungen nicht gut geeignet, ebenso wenig für Mini-Batching. Da die inverse Hessematrix gespeichert werden muss, ist er auch für komplexe neuronale Netzwerke mit einer sehr großen Parameteranzahl unvorteilhaft [7, S. 313].
Für diese spezielle Optimierungsaufgabe mit einer streng konvexen, deterministischen Optimierungsfunktion und sehr kleiner Parameterzahl liefert er jedoch hervorragende Ergebnisse

und ist dem Gradient Descent- und dem Adam-Algorithmus weit überlegen.

Im Rahmen dieser Arbeit wurden die drei Optimierungsalgorithmen hinsichtlich eines speziellen Optimierungsproblems verglichen. Es wurden Parameter des Modells, welches das Wärmeleitungsproblem numerisch beschreibt, direkt optimiert.
Für spätere Untersuchungen wäre es beispielsweise interessant, für das maschinelle Lernen nicht das gesamte Modell zu verwenden. Stattdessen könnte ein neuronales Netz darauf trainiert werden, die fehlenden Teile des Gleichungssystems zu ersetzen und auf dieser Basis die Lösung zu optimieren.

Literatur

[1] Jeff Bezanson u. a. „Julia: A Fresh Approach to Numerical Computing“. In: *CoRR* abs/1411.1607 (2014). arXiv: 1411.1607. URL: http://arxiv.org/abs/1411.1607.

[2] Stephen Boyd und Lieven Vandenberghe. *Convex Optimization*. Hrsg. von Cambridge University Press. Version 2. Aufl. 2009. URL: https://web.stanford.edu/~boyd/cvxbook/. zuletzt eingesehen: 31.07.2021. ISBN 0 521 83378 7.

[3] The Julia Project Contributors. *Sparse Arrays. In: Julia 1.6. Documentation*. 24. Apr. 2021. URL: https://docs.julialang.org/en/v1/stdlib/SparseArrays/. zuletzt eingesehen: 04.07.2021.

[4] Thermal Engeneering Contributors. *Wärmeübertragung und Massentransfer*. URL: https://www.thermal-engineering.org/de/waermetechnik/. zuletzt eingesehen: 25.06.2021.

[5] Benjamin Gilde. *Explizite und implizite Differenzenformeln*. TU Berlin. 16. Dez. 2000. URL: http://www.cfd.tu-berlin.de/Lehre/tfd_skript/node16.html. zuletzt eingesehen: 28.06.2021.

[6] Benjamin Gilde. *Explizite und implizite Differenzenformeln*. TU Berlin. 16. Dez. 2000. URL: http://www.cfd.tu-berlin.de/Lehre/tfd_skript/node11.html. zuletzt eingesehen: 17.07.2021.

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

[8] Michael Günther und Ansgar Jüngel. „Numerische Lösung parabolischer Differentialgleichungen“. In: *Finanzderivate mit MATLAB®: Mathematische Modellierung und numerische Simulation"*. Wiesbaden: Vieweg+Teubner, 2010, S. 146–194. ISBN: 978-3-8348-9786-2. DOI: 10.1007/978-3-8348-9786-2_6. URL: https://doi.org/10.1007/978-3-8348-9786-2_6.

[9] Ernst Hairer und Gerhard Wanner. „Stiff Problems — One-Step Methods“. In: *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, S. 1–254. ISBN: 978-3-662-09947-6. DOI: 10.1007/978-3-662-09947-6_1. URL: https://doi.org/10.1007/978-3-662-09947-6_1.

[10] Bastian von Harrach. *Numerik von Differentialgleichungen*. 15. Juni 2021. URL: https://www.math.uni-frankfurt.de/~harrach/lehre/Numerik_von_Differentialgleichungen.pdf. zuletzt eingesehen: 30.06.2021.

[11] Andreas Höfler. *Herleitung der Wärmeleitungsgleichung*. 1. Feb. 2020. URL: https://www.tec-science.com/de/thermodynamik-waermelehre/waerme/warmeleitungsgleichung-diffusionsgleichung/#Herleitung_der_Waermeleitungsgleichung. Zuletzt eingesehen: 10.07.2021.

[12] Michael Innes. *Don't Unroll Adjoint: Differentiating SSA-Form Programs*. 2019. arXiv: 1810.07951 [cs.PL].

[13] Michael Innes u. a. „Fashionable Modelling with Flux". In: *CoRR* abs/1811.01457 (2018). arXiv: 1811.01457. URL: https://arxiv.org/abs/1811.01457.

[14] Discourse JuliaLang. *Why does Julia promote full array to sparse array?* https://discourse.julialang.org/t/why-does-julia-promote-full-array-to-sparse-array/44205. Zuletzt eingesehen: 06.07.2021.

[15] Diederik P. Kingma und Jimmy Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014. URL: http://arxiv.org/abs/1412.6980.

[16] Wolfram Language und System Documentation Center. *The Numerical Method of Lines (Tutorial)*. URL: https://reference.wolfram.com/language/tutorial/NDSolveMethodOfLines.html#2081642391. zuletzt eingesehen: 07.07.2021.

[17] Jan Lunze. „Beschreibung linearer Systeme im Zeitbereich". In: *Regelungstechnik 1: Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, S. 59–121. ISBN: 978-3-662-52678-1. DOI: 10.1007/978-3-662-52678-1_4. URL: https://doi.org/10.1007/978-3-662-52678-1_4.

[18] Yingbo Ma u. a. *A Comparison of Automatic Differentiation and Continuous Sensitivity Analysis for Derivatives of Differential Equation Solutions*. 2021. arXiv: 1812.01892 [cs.NA].

[19] Andreas Malcherek. *Numerische Methoden der Strömungsmechanik*. 2014. URL: https://bookboon.com/premium/books/numerische-methoden-der-stromungsmechanik. E-Book (PDF).

[20] Helmut Martin. „Finite-Volumen-Methode". In: *Numerische Strömungssimulation in der Hydrodynamik: Grundlagen und Methoden*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 141–152. ISBN: 978-3-642-17208-3. DOI: 10.1007/978-3-642-17208-3_8. URL: https://doi.org/10.1007/978-3-642-17208-3_8.

[21] Julia Standard Library Module. *DelimitedFiles*. URL: https://docs.julialang.org/en/v1/stdlib/DelimitedFiles/. zuletzt eingesehen: 25.07.2021.

[22] Patrick Kofod Mogensen und Asbjørn Nilsen Riseth. „Optim: A mathematical optimization package for Julia". In: *Journal of Open Source Software* 3.24 (2018), S. 615. DOI: 10.21105/joss.00615.

[23] Patrick Kofod Mogensen u. a. *JuliaNLSolvers/Optim.jl: v1.4.1*. Version v1.4.1. Aug. 2021. DOI: 10.5281/zenodo.5151681. URL: https://doi.org/10.5281/zenodo.5151681.

[24] Claus-Dieter Munz und Thomas Westermann. „Differenzenverfahren". In: *Numerische Behandlung gewöhnlicher und partieller Differenzialgleichungen: Ein anwendungsorientiertes Lehrbuch für Ingenieure*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, S. 233–286. ISBN: 978-3-662-55886-7. DOI: 10.1007/978-3-662-55886-7_7. URL: https://doi.org/10.1007/978-3-662-55886-7_7.

[25] Jorge Nocedal und Stephen J. Wright. *Numerical Optimization*. 2. Aufl. New York, NY: Springer New York, 2006. ISBN: 978-0-387-40065-5. DOI: 10.1007/978-0-387-40065-5_2. URL: https://doi.org/10.1007/978-0-387-40065-5_2.

[26] Julia Package. *BenchmarkTools.jl*. URL: https://juliaci.github.io/BenchmarkTools.jl/dev/reference/. zuletzt eingesehen: 25.07.2021.

[27] Julia Package. *Optim.jl, part of JuliaNLSolvers family*. URL: https://julianlsolvers.github.io/Optim.jl/stable/. zuletzt eingesehen: 02.08.2021.

[28] Suraj Pawar und Omer San. „CFD Julia: A Learning Module Structuring an Introductory Course on Computational Fluid Dynamics“. In: *Fluids* 4.3 (2019). ISSN: 2311-5521. DOI: 10.3390/fluids4030159. URL: https://www.mdpi.com/2311-5521/4/3/159.

[29] Chris Rackauckas und Qing Nie. „DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia“. In: *Journal of Open Research Software* 5 (Mai 2017). DOI: 10.5334/jors.151.

[30] Chris Rackauckas u. a. *Why you shouldn't use Euler's method to solve ODEs*. 18. Juni 2019. URL: https://nextjournal.com/ChrisRackauckas/why-you-shouldnt-use-eulers-method-to-solve-odes. Zuletzt eingesehen: 27.07.2021.

[31] Christopher Rackauckas. *Optimizing DiffEq Code*. 25. Mai 2021. URL: https://tutorials.sciml.ai/html/introduction/03-optimizing_diffeq_code.html. zuletzt eingesehen: 06.07.2021.

[32] Bettina Schull. *PI_Ode_Solve_ML*. 21. Feb. 2021. URL: https://gitlab.com/pi_simulationeequ/pi_odesolv_ml.git. zuletzt eingesehen: 23.06.2021.

[33] Bettina Schull. *PiSlider*. 31. Mai 2021. URL: https://gitlab.com/pi_simulationeequ/PiSlider.git. zuletzt eingesehen: 26.06.2021.

[34] Nick Connor Thermal Engeneering. *Thermal Engeneering. Was ist Konvektion – konvektive Wärmeübertragung – Definition*. Thermal Engineering. URL: https://www.thermal-engineering.org/de/was-ist-konvektion-konvektive-warmeubertragung-definition/. zuletzt eingesehen: 25.06.2021.

[35] Christopher Rackauckas u.a. *DiffEquOperators.jl*. http://diffeqoperators.sciml.ai/stable/operators/derivative_operators/. Zuletzt eingesehen: 06.07.2021.

[36] Christopher Rackauckas u.a. *Local Sensitivity Analysis (Automatic Differentiation) In: DifferentialEquations.jl v6.19.0 Documentation*. URL: https://diffeq.sciml.ai/stable/analysis/sensitivity/. Zuletzt eingesehen: 29.07.2021.

[37] Christopher Rackauckas u.a. *ODE Solvers In: DifferentialEquations.jl v6.19.0 Documentation*. URL: https://diffeq.sciml.ai/stable/solvers/ode_solve/. Zuletzt eingesehen: 23.07.2021.

[38] Christopher Rackauckas u.a. *Timestepping Method Descriptions In: DifferentialEquations.jl v6.19.0 Documentation*. URL: https://diffeq.sciml.ai/stable/extras/timestepping/. Zuletzt eingesehen: 23.07.2021.

[39] Christopher Rackauckas u.a. *Tutorial: Solving Stiff Equations*. https://diffeq.sciml.ai/stable/tutorials/advanced_ode_example/. Zuletzt eingesehen: 24.07.2021.

[40] viZaar industrial imaging AG. *Emissionsgrad-Tabelle*. URL: https://vizaar-xtra.de/emissionsgrad-tabelle/. zuletzt eingesehen: 23.06.2021.

[41] Karen Willcox und Qiqi Wang. *16.90 Computational Methods in Aerospace Engineering. Spring 2014*. Abschnitt: Stiffness from PDE discretization of diffusion problem. Massachusetts Institute of Technology: MIT OpenCourseWare. License: Creative Commons BY-NC-SA. URL: https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-90-computational-methods-in-aerospace-engineering-spring-2014/numerical-integration-of-ordinary-differential-equations/stiffness-and-implicit-methods/. Zuletzt eingesehen: 24.07.2021.

[42] Kirill Zubov u. a. *NeuralPDE: Automating Physics-Informed Neural Networks (PINNs) with Error Approximations*. 2021. arXiv: 2107.09443 [cs.MS].