FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachochschule Aachen, Campus Jülich

Fachbereich 09 - Medizintechnik und Technomathematik
Studiengang Angewandte Mathematik und Informatik

Seminararbeit

Untersuchung der Anwendbarkeit von Python FEM Bibliotheken mit integriertem automatischen Differenzieren für inverse Probleme

Autor:
Lukas Maximilian Sträche, 3273378

Betreuer:
Prof. Dr. Mehdi Behbahani
Eugen Salzmann, M. Sc.

Aachen, 20. Januar 2023

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema

Untersuchung der Anwendbarkeit von Python FEM Bibliotheken mit integriertem automatischen Differenziesen für inverse Probleme

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Lukas Maximilian Ströche

Aachen, den 20.01.2023

L. Ströche

Unterschrift der Studentin / des Studenten

Abstract

Diese Seminararbeit befasst sich mit der Untersuchung der Anwendbarkeit von Python Bibliotheken, welche die Berechnung der Finite Elemente Methode, sowie dem automatischen Differenzieren für inverse Probleme bereitstellen. Als Grundlage zur späteren Bewertung wird mit beiden Bibliotheken dieselbe gegebene transiente Wärmegleichung eins zweidimensionalen Stabes mithilfe der gegebenen Finiten Elemente Methode programmiert und berechnet. Dazu werden zunächst die theoretischen Grundlagen, sowie die Bewertungen gelegt, die Implementierungen der beiden Bibliotheken unter Verwendung des Python Interpreters der Version 3.9.13 mit zusätzlichem Postprocessing vorgestellt und die Genauigkeit mithilfe einer bekannten analytischen Lösung bewertet. Dadurch erhalten wir eine Bewertungsgrundlage, anhand welcher ein abschließendes Urteil über die Verwendbarkeit der Bibliotheken gefällt werden kann.

Inhaltsverzeichnis

1 Einleitung

Heutzutage sind computergestützte Simulationen für die Entwicklung neuer mechanischer Komponenten, wie Autokolben oder Stützträger, aufgrund ihrer präzisen Vorhersage nicht mehr wegzudenken. So durchlaufen neue Produkte eine Vielzahl von verschiedenen Simulationen, eine davon befasst sich mit der Wärmeverteilung. Dabei kann mithilfe von computerunterstützten Simulationen die Wärmeleitfähigkeit sowie das Wärmeverhalten mithilfe von angenommenen oder bekannten Randbedingungen genaustens bestimmt werden, ohne dabei viel Zeit und Kapital in die Entwicklung eines physischen Prototyps zu stecken. In dieser Seminararbeit wird die Wärmeverteilung mithilfe der Finiten Elemente Methode unter Verwendung von zwei dafür ausgelegten Bibliotheken analysiert. Dafür wird ein Beispiel aus einem fachkundigen Buch herangezogen. Anhand dessen werden die beide Bibliotheken FEniCS (Version 2022.05) und Nutils (Version 7.1) auf Grundlage von später definierten Messgrößen untersucht und bewertet. Zusätzlich dazu wird das gegebene Beispiel reverse engineered, um so das automatische Differenzieren der Bibliotheken zu untersuchen. Hierbei kann anhand einer gegebenen Endwärmeverteilung, welche vorher selber errechnet wird, die Wärmeleitfähigkeit κ des Stabes angenähert werden. Dafür verwenden wir den Optimierungsalgorithmus des Gradientenverfahren unter Verwendung des automatischem Differenzieren. Alle Simulationen und einige Postprocessing Schritte werden in der Programmiersprache Python (Version 3.9.14) programmiert. Zusätzlich zu dem Python Postprocessing werden auch weitere Plots über ParaView erstellt.
In Kapitel 2 werden zunächst die physikalischen und mathematischen Grundlagen für das Verständnis der Wärmegleichung, der Finite Elemente Methode und des inversen Problems erläutert. Das 3. Kapitel umfasst alle programmtechnischen Aspekte der Seminararbeit, von den Anforderungen an die Bibliotheken über das automatische Differenzieren bis hin zu der Implementierung. Anschließend werden im 4. Kapitel die Skripte anhand der Anforderungen aus Kapitel 3 untersucht und die Ergebnisse dieser vorgestellt. Zum Schluss wird anhand der erarbeiteten Ergebnisse aus Kapitel 4 ein Fazit über die Verwendbarkeit der beiden Bibliotheken gefällt.

2 Grundlagen

In diesem Kapitel werden die physikalischen und mathematischen Grundlagen gelegt, welche beim Verständnis der nachfolgenden Kapitel helfen. Hierbei wird das Hauptaugenmerk auf der Finiten Elementen Methode liegen, welche für die Implementierungen der Bibliotheken für ein transienten Wärmeleitungsproblem in einem zweidimensionalen Stab wichtig ist.

2.1 Wärmeleitung

Die Wärmeleitung beschreibt zunächst nur die Übertragung von Energie zwischen oder in Feststoffen, Flüssigkeiten oder Gasen. [4] Dabei gilt nach dem zweiten Hauptsatz der Thermodynamik, dass Energie, hier Wärme, stets vom Ort höherer Temperatur zum Ort niedriger Temperatur fließt. Des Weiteren geht bei diesem Vorgang nach dem Satz der Energieerhaltung auch keine Wärme verloren. [12]
Der Wärmefluss $\dot{Q}$ wird durch das Gesetz nach Fourier beschrieben

$$\dot{Q} = \lambda \cdot A \cdot \frac{T_1 - T_2}{\triangle x}, \tag{2.1}$$

mit $\dot{Q}$ für den Wärmefluss, λ die Wärmeleitfähigkeit des Stoffes, A die Fläche, durch welche die Wärme strömt, T_1 und T_2 für die wärmere und kältere Oberfläche, sowie $\triangle x$ für die Dicke des Körpers.
Die Temperaturverteilung eines beliebigen Körpers kann aus der sogenannten Wärmeleitungsgleichung bestimmt werden. Diese ist eine partielle Differenzialgleichung, welche die Änderung der Temperatur über die Zeit mit der Wärmeleitung im Körper und der erzeugten Wärme einer möglichen Quelle bilanziert. [3]
Sei u: [0,T] $\times$ Ω $\rightarrow$ $\mathbb{R}^+$ ein Temperaturfeld mit $\Omega \subset \mathbb{R}^d$ und f: [0,T] $\times$ Ω $\rightarrow$ $\mathbb{R}$ eine Wärmequelle dann gilt für die Wärmegleichung

$$\partial_t u(t,x) - \nabla \cdot (\kappa(u(t,x)) \nabla u(t,x)) = f(t,x), \tag{2.2}$$

wo κ die Temperaturleitfähigkeit des gegebenen Materials beschriebt. Sie wird in dieser Seminararbeit als konstante Größe angenommen.

Ein außerdem wichtiger Teil zum Lösen des partiellen Differenzialgleichungsproblems ist die Annahme von Randbedingungen als auch eines Anfangswertes. Es gibt drei verschiedene Arten von Randbedingungen:

- Dirichlet (beschreiben die Funktionswerte an den jeweiligen Definitionsrändern)
- Neumann (beschreiben die Funktionswerte für die Normalableitung der jeweiligen Definitionsränder)
- Robin (Kombination der Dirichlet- und Neumann-Randbedingungen) [5]

Da partielle Differenzialgleichungen selten analytisch gelöst werden können, benötigen wir ein numerisches Verfahren, welches uns eine gute Approximation der Lösungen der partiellen Differenzialgleichungen gibt.

2.2 Anwendungsfall transiente Wärmeleitung in einem Stab

Das folgende Beispiel ist aus [7] und befasst sich mit einer transienten Wärmeleitung in einem zweidimensionalen Stab, für welchen es auch eine analytische Lösung gibt, mit welcher wir die approximierten Lösungen der Bibliotheken überprüfen und vergleichen können. Der Stab besitzt im Beispiel die relativen Abmessungen von einer Einheit in y-Richtung und 20 Einheiten in x-Richtung. Als gegebene Randbedingungen haben wir zum einen den von Links kommenden Wärmefluss, welcher stetig ist und mit 1 angenommen wird, während die restlichen Seiten abisoliert sind und somit als adiabatische Randbedingungen eine Wärmeübertragung von 0 besitzen. Die Anfangstemperatur des Stabes wird als 0°C angenommen. [7]

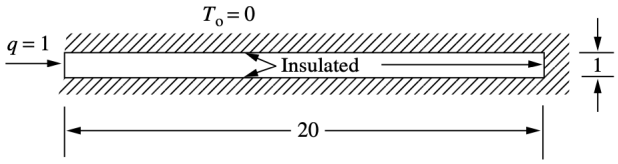

Abbildung 2.1: 1D Zeichnung der Wärmeverteilung(Quelle: [7])

Dadurch erhalten wir folgende Problemstellung

$$\begin{cases} \partial_t u(t,x) = \nabla \cdot (\kappa \nabla u(t,x)) + f \\ \partial_n u_l(t,x) = g_{N,l} \\ \partial_n u_r(t,x) = 0. \end{cases} \tag{2.3}$$

$\partial_n u_l(t,x)$ beschreibt die Randbedingung auf der linken Seite und $u_r(t,x)$ auf allen anderen Seiten des Stabes. Obwohl dieses Problem zweidimensional aufgestellt und auch berechnet wird, ist die Lösung aufgrund der Geometrie und der Randbedingungen in y-Richtung konstant. Dadurch erhalten wir eine zweidimensionale Lösung, welche praktisch eindimensional betrachtet werden kann. [7]

Die erwähnte analytische Lösung zum Überprüfen der Genauigkeit der beiden Bibliotheken für das Problems von Carslaw und Jager ist:

$$T(\mathbf{x},t) = 2(t/\pi)^{1/2}[exp(-\mathbf{x}^2/4t) - (1/2)\mathbf{x}\sqrt{\frac{\pi}{t}}erfc(\frac{\mathbf{x}}{2\sqrt{t}})] \qquad ((2.4)\ [7])$$

Die Temperaturverteilung der analytischen Lösung über den Stab sieht bei t=2, also nach 2 Zeiteinheiten, wie folgt aus.

Abbildung 2.2: Temperaturverteilung des Stabes zum Zeitpunkt t=2s

2.3 Finite Elemente Methode

Für die Berechnung der Wärmeverteilung in unserem Stab, verwenden wir das numerische Verfahren der Finiten Elemente Methode. Exemplarisch wird die Finite Elemente Methode an einem eindimensionalen stationären Wärmeleitungsproblem erklärt. Der spätere Unterschied ist marginal und zum Verständnis der Grundlagen der Methode irrelevant. Grundsätzlich lässt sich die Finite Elementen Methode in zwei Schritte unterteilen:

- Überführung des Problems in die schwache Formulierung
- Diskretisierung

Im ersten Schritt wird das Problem in die schwache Formulierung umgeformt, dafür wird die Differenzialgleichung mit einer Testfunktion (auch Gewichtsfunktion genannt) multipliziert und über das Rechengebiet integriert. Die Testfunktion kann jede beliebige Funktion annehmen, solang die starke Form der Gleichung noch erfüllt ist. Bei der Berechnung der Finiten Elemente Methode kann die schwache Formulierung nach dem Fundamentalsatz der Variationsrechnung hinzugezogen werden, da davon ausgegangen werden kann, dass jede Lösung der starken Formulierung auch eine Lösung der schwachen Formulierung und umgekehrt ist. [2] [5]
Sei f(x) mit $x\epsilon[a,b]$ unsere Differentialgleichung und w(x) eine beliebige Testfunktion muss gelten:

$$f(x) = 0 \Leftrightarrow \int_a^b w(x)f(x)dx = 0, \quad \forall w(x) \qquad (2.5)$$

Der Vorteil gegenüber der starken Formulierung liegt begründet in der integralen Mittelung, wodurch einzelne Unstetigkeiten in der gesuchten Funktion irrelevant für die Formulierung werden. Demnach hat die Variationsformulierung schwächere Anforderungen an die Eigenschaften der gesuchten Funktion. Ein weiterer Vorteil der schwachen Formulierung ist die Anwendbarkeit der partiellen Integration, wodurch es ermöglicht wird, den Grad der Ableitung in der Differenzialgleichung zu verkleinern. Die Ableitungsgrade gehen dabei nicht verloren, sondern gehen auf die Testfunktion über. Dadurch verringert sich die Ordnung der gewählten Ansatzfunktionen, was eine Verringerung des Rechenaufwands mit sich zieht. Somit kann die Gleichung 2.5 noch um die partielle Integration erweitert werden und es entsteht die schwache Form [2] [5]

$$\int_a^b w(x)f'(x) = -\int_a^b w'(x) \cdot f(x)dx + [w(x) \cdot f(x)]_a^b. \tag{2.6}$$

Im nächsten Schritt wird die nun in schwacher Form vorliegende partielle Differentialgleichung diskretisiert. Dafür wird das komplette Rechengebiet in die sogenannten Elemente, welche meistens dreieckig oder viereckig sind, aufgeteilt. Diese Elemente besitzen dann an ihren Eckpunkten die Knoten N, welche sie sich mit ihren Nachbarelementen teilen, sie verbindet und so ein Rechengitter darstellen. Durch dieses Rechengitter ergibt sich nun für die Approximation der gesuchten Funktion $\bar{f}(x)$ eine Summe bestehend aus den Werten der gesuchten Funktion an der diskreten Stelle im Rechengitter multipliziert mit der jeweiligen Interpolationsfunktion über alle im Rechengitter vorhandenen Knoten N. [2] [5]

$$\bar{f}(x) = \sum_{k=1}^{N} f_k \phi_k(x) \tag{2.7}$$

Damit die Interpolationsfunktion die einzelnen Knoten im Rechengitter repräsentieren kann muss gelten, dass sie an dem jeweiligen Knoten den Wert 1 annimmt und für alle anderen Knoten den Wert 0. [2] [5]
Im eindimensionalen ist die lineare Interpolationsfunktion auch als Hutfunktion mit folgendem Graphen bekannt. [7]

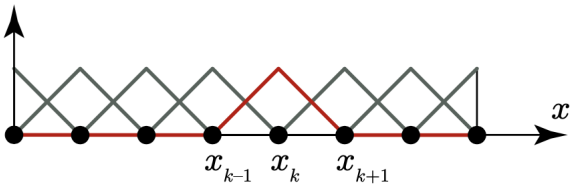

Abbildung 2.3: Lineare Interpolationspolynome in 1D (Hutfunktionen) (Quelle: [2])

Durch die Betrachtung der Abbildung 2.3 wird deutlich, dass dadurch nur die Elemnte, welche den Knoten besitzen oder umgeben einen mathematischen Zusammenhang besitzen.

Schließlich fehlt zur vollständigen Diskretisierung noch die Festlegung der Testfunktion. Die Testfunktion muss so gewählt werden, dass sie die Bedingung aus Gleichung 2.5 erfüllt. Diese wird nach der Galerkin-Methode vollständig gleich der Interpolationsfunktion gesetzt, sodass gilt $w(x) = \phi_i(x)$, wobei $\phi_i(x)$ den Interpolationspolynomen der Knoten entspricht. [2] [5] Dadurch gilt für die partielle Differentialgleichung

$$\int_a^b \phi_i(x)(\sum_{k=1}^{N} f_k \phi_k(x))dx = 0. \tag{2.8}$$

Dies kann nun als lineares Gleichungssystem der Form $A \cdot f = b$ intepriert werden, wobei f den gesuchten Approximationen an den Knoten entspricht. Ausgeschrieben ergibt das Gleichungssystem:

$$\begin{pmatrix} \int_0^L \phi_1(x)\phi_1 dx & \int_0^L \phi_1(x)\phi_2 dx & ... & \int_0^L \phi_1(x)\phi_N dx \\ \int_0^L \phi_2(x)\phi_1 dx & \int_0^L \phi_2(x)\phi_2 dx & ... & \int_0^L \phi_2(x)\phi_N dx \\ \vdots & \vdots & \vdots & \vdots \\ \int_0^L \phi_N(x)\phi_1 dx & \int_0^L \phi_N(x)\phi_2 dx & ... & \int_0^L \phi_N(x)\phi_N dx \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{2.9} \text{ [2]}$$

Aufgrund der gegebenen Eigenschaften der Interpolationspolynome und der Form der Matrix A aus 2.9, werden ein Großteil der Einträge gleich 0. Denn bei der Multiplikation der Interpolationsfunktionen gibt es nur wenige Einträge, die zeitgleich ungleich 0 sind. Dadurch schrumpft diese auf eine dünnbesetzte Matrix, welche so performanter berechnet werden kann. Seien nun die Knoten N innerhalb der Gitterstruktur von Links nach Rechts durchnummeriert, ergibt sich eine typisch tridiagonale Matrixstruktur, welche sich dadurch auszeichnet Einträge auf der Hauptdiagonalen sowie den beiden Nebendiagonalen zu haben. [2] [5]

$$\begin{pmatrix} \int_0^L \phi_1(x)\phi_1 dx & \int_0^L \phi_1(x)\phi_2 dx & 0 & ... & ... \\ \int_0^L \phi_2(x)\phi_1 dx & \int_0^L \phi_2(x)\phi_2 dx & \int_0^L \phi_2(x)\phi_3 dx & 0 & ... \\ 0 & \int_0^L \phi_3(x)\phi_2 dx & \int_0^L \phi_3(x)\phi_3 dx & \int_0^L \phi_3(x)\phi_4 dx & \vdots \\ \vdots & ... & ... & ... & \ddots \\ 0 & ... & 0 & \int_0^L \phi_N(x)\phi_{N-1} dx & \int_0^L \phi_N(x)\phi_N dx \end{pmatrix} \tag{2.10} \text{ [2]}$$

Damit ist eine Diskretisierung im Raum möglich, aber es muss außerdem noch einer Diskretisierung der Zeit stattfinden, welche mithilfe des impliziten Euler-Rückwerts-Verfahrens berechnet wird. Die unbekannten f_i werden nun zeitabhängig und bekommen einen weiteren Index n, welcher eine diskrete Zeitauswertung darstellt, mit $f_i^n = f_i(t^n)$ und $t^n = n \cdot \triangle t$. Dazu wird zur Approximation der ersten zeitlichen Ableitung, die 2-Punkte-Differenz-Formel, bestehend aus den beiden oben genannten Zeitschritten, sowie Δt, verwendet. [5]

2.4 Inverse Probleme

Bei einem Inversen Problem wird ausgehend von einer angenommenen Wirkung auf ein System und dem Ergebnis auf die Ursachen geschlossen. [10] Für das inverse Problem wird auf Grundlage eine Referenztemperaturverteilung $u_{ref}(T,x)$, welche aus der vorher berechnete Forwärtssimulation besteht, der Parameter κ minimiert.
Die daraus resultierende Kostenfunktion lautet

$$\begin{aligned} J(u;\kappa) &= \int_\Omega \langle u_{ref}(T,x) - u(T,x;\kappa), u_{ref}(T,x) - u(T,x;\kappa)\rangle \, dx \\ &= \left\| u_{ref}(T,x) - u(T,x;\kappa) \right\|^2_{L^2(\Omega)}, \end{aligned} \tag{2.11}$$

mit $u(T,x;\kappa)$ als Temperaturverteilung mit einem geschätzten κ.
Es gibt viele verschiedene Algorithmen und Methoden, um ein Inverses Problem zu lösen. In dieser Seminararbeit wird das Gradientenverfahren verwendet.
Das Gradientenverfahren ist ein Minimierungsalgorithmus für eine Funktion *f(x)*, bei welchem sich mithilfe des negativen Gradientens der Funktion $\nabla f(x)$ einem (lokalen) Minimum der Funktion angenähert wird. Die Iterationsvorschrift für das Standard Gradientenverfahren ist [11]

$$x^{n+1} = x^n - \eta \cdot \nabla f(x^n), \tag{2.12}$$

wobei η die Schrittweite ist und entscheident die Kovergenz beeinflusst. Mit dem gegebenen Optimierungsproblem aus 2.11 für κ und der Iterationsvorschrift aus Gleichung 2.12 ergibt sich folgende Iterationsvorschrift für das betrachtete Beispiel:

$$\kappa^{n+1} = \kappa^n - \alpha * \frac{dJ(u;\kappa^n)}{d\kappa^n} \tag{2.13}$$

η kann entweder konstant sein oder für jede Iteration in verschiedensten Ausführungen berechnet werden. Ist sie zu klein angesetzt, kann es unter Umständen sehr lange dauern, bis das Verfahren konvergiert. Ist sie zu groß gewählt, kann es zum einen dazu führen, dass wir, wie in der Abbildung rechts zu sehen, über den Konvergenzpunkt hin und her springen oder der Algorithmus sogar divergieren. [11]

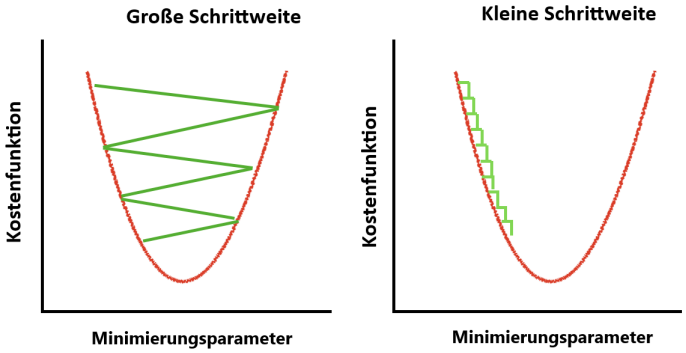

Abbildung 2.4: Auswirkung der Schrittweite auf das Gradientenverfahren (Quelle: [6], bearbeitet)

3 Tools

3.1 Anforderungen an die Bibliotheken

Das Definieren der Anforderungen an die Bibliotheken ist die Grundlagen dieser Seminararbeit. Erst dadurch lassen sich die beiden Bibliotheken im Nachhinein anhand ihrer Eigenschaften und Leistungen bewerten. Hierbei können die Anforderungen in zwei Arten unterschieden werden. Zum einen die technischen Anforderungen und anderseits die Handhabbarkeit der Bibliotheken.
Technische Anforderungen der Bibliotheken sind rein objektive Gesichtspunkte. Hierzu zählt zum einen die Genauigkeit der berechneten Lösung, welche wir anhand einer gegebenen analytischen Lösung bestimmen können, oder wie im Falle des inversen Problems anhand unserer gegebenen Referenzberechnung. Zusätzlich dazu ist für Simulationen noch die benötigte Zeit, welche sich in der Performanz der benutzten Algorithmen und Methoden der Bibliotheken widerspiegelt, eine weitere wichtige Größe. Da beide zu untersuchende Bibliotheken im Spektrum der Berechnung von Finiten Elemente Methoden bekannt sind, kann davon ausgegangen werden, dass sie, bis auf geringfügige Genauigkeiten, korrekte Berechnungen durchführen, wodurch die zeitliche Betrachtung in Relation einen höheren Stellenwert erhalten kann. Zur genauen Zeitmessung wird für beide Bibliotheken die Default Python Bibliothek time verwendet. Um eine von Hintergrundprozessen unabhängige Zeit zu erhalten, wird eine Funktion verwendet, welche nur die Zeit der Prozessoren misst, welche auch an der Berechnung des Skriptes arbeiten. Des Weiteren wird zum Eliminieren von größeren Schwankungen anstatt einer Einzelprobe, der Mittelwert von 10 Zeitmessungen genommen.
Die Handhabbarkeit ist eine schwer zu fassende Größe, da sie schlecht bis gar nicht objektiv, sondern vor allem subjektiv bestimmt ist. Hauptaugenmerk der Handhabbarkeit ist zum einen die Komplexität der Implementierung zum Lösen des Anwendungsbeispiels als auch die benötigten Vorleistungen zum Benutzen der Bibliotheken. Als Vorleistungen können vor allem die Schritte der Installation sowie deren Abhängigkeiten und deren Komplexität berücksichtigt werden. Leider ist wiederum auch die Komplexität relativ subjektiv, weshalb wir zur groben Bestimmung immer den Standardnutzer, welcher in unserem Fall ein Ingenieur ist, heranziehen. Dadurch erhalten wir für eine eigentlich subjektive Größe, die bestmögliche objektive Sichtweise.

3.2 Automatisches Differenzieren

Dieser Abschnitt befasst sich mit dem Verfahren des automatischen Differenzierens. Das automatische Differenzieren ist ein programmiertechnisches Verfahren, welches Programmen ermöglicht Funktionen abzuleiten. Dabei werden die komplexen Funktionen in mehrere einfachere Teile unterteilt. Dadurch können diese einfachen Teilfunktionen und ihre Ableitungen analytisch berechnet werden. Um im Anschluss die Ableitung der komplexen Funktion zu bestimmen, muss die Kettenregel sukzessiv auf alle Ableitungen der Teilfunktionen angewendet werden. Die Schachtelung der Teilfunktionen kann als gerichteter azyklischer Graph dargestellt werden. [9]
Es gibt zwei verschiedene Berechnungsverfahren

- den Vorwärts-Modus und
- den Rückwerts-Modus

Bei dem Vorwärts-Modus des automatischen Differenzierens wenden wir die Kettenregel ausgehend von der innersten zur äußersten partiellen Ableitung an, das heißt im gerichteten azyklischen Graph nach dem Bottom-Up-Prinzip. Ein Nachteil des Vorwärts-Modus ist, dass die Funktion nur nach einem Eingabeparameter differenziert werden kann und sich somit nur für Funktionen mit wenigen Eingabeparametern, aber vielen Ausgabeparametern eignet.
Währenddessen wird beim Rückwerts-Modus die Kettenregel ausgehend von der äußersten zur innersten partiellen Ableitung angewandt und so ein Top-Down-Prinzip im Graphen widerspiegelt. Im Gegensatz zum Vorwärts-Modus bekommen wir hier die Ableitungen aller Eingabeparameter, jedoch immer nur für einen speziellen Ausgabeparameter. Somit ist die Komplexität des Rückwerts-Modus gering für die Berechnung einer Funktion mit vielen Eingabeparametern für nur einen oder wenige Ausgabeparameter. Technisch ist er in zwei Schritte unterteilt, zunächst werden die Zwischenergebnisse der einzelnen Teilfunktionen an dem gewählten Funktionswert berechnet und zwischengespeichert, damit im nächsten Schritt diese Zwischenergebnisse verwendet werden können, um die inneren partiellen Ableitungen zu bestimmen. Dieses Vorgehen ist auch unter den Englischen Akronymen “recording the tape” und “interpreting the tape” bekannt. [9]

3.3 FEniCS

Die erste zu untersuchende Bibliothek ist die freie Open Source Bibliothek FEniCS (Version 2022.05), welche in der Programmiersprache C++ geschrieben ist. Ihr Hauptaufgabenbereich ist die Lösung von partiellen Differenzialgleichungen und sie beinhaltet eine Interface-Schnitstelle für C++ und Python.
Somit kann sie ganz einfach per pip-Modul installiert und in einer Entwicklungsumgebung verwendet werden. Außerdem bietet FEniCS bei auftretenden Problemen mit dem pip-Modul, ein vorgefertigtes Docker Image mit der jeweils aktuellsten stabil laufenden Version. [1] [8]

3.3.1 Vorwärtsproblem

Die komplette Berechnung ist im Skript rod_fenics.py zusammengefasst und wurde dabei in Funktionen ausgelagert. Zur besseren Verständlichkeit wird das Skript anhand kleiner Codeauschnitte erklärt.
Das Skript kann thematisch in zwei Teile geteilt werden, zum einen die Verwendung von FEniCS zur Berechnung des Problems und auf der anderen Seite das Postprocessing. Zunächst müssen aber, um FEniCS und seine Abhängigkeiten korrekt verwenden zu können, diese importiert werden. Danach werden erst Parameter gesetzt, um die Gitterstruktur (auch Mesh genannt) sowie den Funktionsraum V zu definieren. Dafür verwenden wir eine vorgefertigte Methode namens *RectangeMesh*, welche uns ein Gitternetz unseres Stabes mit einer Unterteilung von nx Elementen in x-Richtung und ny Elementen in y-Richtung erstellt. Auf diesem Mesh können wir dann unseren Funktionsraum V definieren, welcher gebraucht wird, um die von ihm abhängigen Elemente, das heißt die Testfunktion v, Lösungsfunktion u und unseren Anfangswert u_n, wie in den Zeilen 3 bis 7 der Abbildung 3.1 zu sehen, zu definieren. Auch hier greifen wir auf vorgefertigte FEniCS Methoden zurück. In Zeile 9 finden wir die Aufstellung der partiellen Differenzialgleichung als Residuum. Zuletzt müssen wir noch, um die Berechnung der Finiten Elemente Methode in FEniCS durchführen zu können, die sogenannten Left-hand side(LHS)/Right-hand side(RHS) definieren. Diese repräsentieren das ganze Gleichungssystem, wobei alle Unbekannten auf die LHS und die Bekannten auf die RHS verteilt werden.

```
1 #Define varitional problems
2 V = FunctionSpace(mesh, 'P', 1)
3 u = TrialFunction(V)
4 v = TestFunction(V)
5
6 #Set initial values to T0
7 u_n = Interpolate(Expression('T0', degree=0, T0=T0), V)
8
9 F = u * v * dx + kappa * dt *(dot(grad(u),grad(v))* dx - g * v * ds) - (u_n + dt * f) * v * dx
10 a, L = lhs(F), rhs(F)
```

Abbildung 3.1: Definierung der schwachen Form

Das Problem ist in Abhängigkeit von Raum und Zeit gestellt, aber da die Bibliothek nur die räumlichen Komponenten berücksichtigt, muss das ganze eigenhändig einpflegt werden. Aus den in Kapitel 2.3 gelernten Grundlagen zu der Finiten Elemente Methode haben wir gelernt, dass eine Zeitdiskretisierung hauptsächlich aus der Verwendung des Euler-Rückwerts-Verfahren besteht. Programmtechnisch können wir das ganze durch eine einfache For-Schleife umsetzten. Dabei iterieren wir wie in Abbildung 3.2 zu sehen über alle Zeitschritte. Innerhalb der Loop berechnen wir zunächst die Lösung für unseren aktuellen Zeitschritt, um dann im nächsten Schritt unsere Lösung aus dem vorherigen Zeitschritt u_n damit zu aktualisieren. In der ersten Iteration entspricht u_n der Anfangsbedingung.

```
for n in range(num_steps):
  #Compute Solution
  solve(a == L, u, bc)
  #Update previous Solution
  u_n.assign(u)
```

Abbildung 3.2: Zeitdiskretisierung

Nach der eigentlichen Berechnung der Werte folgt, das Postprocessing. Im Postprocessing werden die errechneten Daten aufbereitet und in lesbare Darstellungen umgewandelt. Da der Code nicht relevant für die weiteren Untersuchungen ist, werden nur die Schritte grob beschrieben. Das Skript schreibt die errechneten Daten in eine sogenannten .pvd Datei zur weiteren Verarbeitung oder Anschauung in ParaView.

3.3.2 Inversesproblem mit AD

Die Berechnung des inversen Problems wird in einem anderen Python Skript inverse_fenics.py durchgeführt. Dafür wird neben FEniCS selbst noch die Bibliothek dolfin-adjoint eingebunden, welche als Erweiterung fungiert. Dolfin-adjoint überläd die dafür notwendigen FEniCS Methoden, wie solve oder assign zur Verwendung des Rückwerts-Modus der automatischen Differenzierung. Da wir in unserem inversen Problems ausgehend von einer bekannten Temperaturverteilung lösen, benötigen wir zunächst einmal die Berechnung der Finiten Elemente Methode, wie sie im vorherigen Kapitel 3.3.1 erläutert wurde.
Neben der Referenztemperaturverteilung sehen wir in den Zeilen 2 f. der Abbildung 3.3 die benötigten Parameter, wie einem Anfangs Kappa und der Schrittweite, zur Aufstellung des Inversen Problems sehen. Das inversen Problems wird durch eine While-Scheife repräsentiert, die bei zwei Bedingungen abgebrochen wird und somit fertig ist. Dies geschieht zum einen, wenn die Veränderung von κ unter eine gewisse Toleranzgrenze fällt, wo wir davon ausgehen, dass wir nah genug am (lokalen) Minimum sind oder wenn wir eine gewisse Anzahl an Iterationen berechnet haben und durch die Schrittweite entweder divergiert sind oder zu

langsam konvergieren. Innerhalb der Schleife wird dann neben der Forwärtssimulation für das geschätzte κ, zu sehen in Zeile 7, auch die Kostenfunktion J und daraus der Gradienten, unter Verwendung des automatischen Differenzierens berechnet, mithilfe welchem dann die Iterationsvorschrift in Zeile 13 berechnet werden kann.

```
1  #Inverse problem parameters
2  initial_guess = 0.3
3  step_size = 1e-1
4  ...
5  while (diff_kappa >= tol) and i > max_iter:
6    #Compute solution
7    u.assign(forward_simulation(V,T,num_steps,kappa))
8    #Compute cost function and gradient
9    J = assemble(inner(temp_ref-u, temp_ref-u)* dx)
10   dJ_dkappa = compute_gradient(J, Control(kappa))
11   #Update kappa
12   kappa_new = float(kappa- step_size * dJ_dkappa)
13   diff_kappa = np.abs(kappa_new - float(kappa))
14   i = i +1
15   ...
```

Abbildung 3.3: Implementierung des Inversen Problems in FEniCS

3.4 Nutils

Die zweite Bibliothek ist Nutils in der Version 7.1. Ebenso wie FEniCS ist Nutils eine freie Open Source Bibliothek, jedoch im Gegensatz zu FEniCS vollständig in Python geschrieben und auch nur per pip-Modul zu installieren. [13]

3.4.1 Vorwärtsproblem

Ebenso wie bei FEniCS wurde auch die Berechnung der Finiten Elemente Methode in einem Python Skript rod_nutils.py zusammengefasst. Die ersten Schritte vom Importieren der Bibliotheken bis hin zu der Definierung des Funktionsraums ist bis auf die Verwendung von Bibliothekseigener Methoden und Semantik erst einmal sehr ähnlich. Außer, dass Nutils, die Topografie und Geometrie, zu sehen in den Zeilen 1 bis 5 der Abbildung 3.4, als getrennte Größen ansieht. Um unsere gewollten Berechnungen als Zeichenkette angegeben zu können, werden zunächst alle relevanten Funktionen und Parameter, wie unser u oder u_n in den Zeilen 6 und 7 derselben Abbildung auf dem Namespace definiert. Somit dient er als ein Raum, in welchem alle geometrischen Simulationsparameter über eine mathematische Schreibweise definiert werden.

```
1 domain, geom = mesh.rectilinear(verts)
2
3 ns = Namespace()
4 ns.x = geom
5 ns.basis = domain.basis(std, degree=1)
6 ns.u = function.dotarg('lhs', ns.basis)
7 ns.un = function.dotarg('lhsun', ns.basis)
8 ...
```

Abbildung 3.4: Definierung des Stabes und des Namespaces

Anschließend werden zunächst in den Zeilen 2 bis 6, in der auf der nächsten Seite folgenden Abbildung 3.5, die Anfangsbedingungen und danach die partielle Differenzialgleichung mit den vorher definierten Zeichen definiert. Daraufhin wird das Residuum unter Berücksichtigung der Randbedingungen vervollständigt. Zu guter Letzt findet sich in den Zeilen 11 bis 13 nur noch die aus FEniCS bekannte For-Schleife zur Berechnung des Euler-Rückwert-Verfahrens für die zusätzliche Zeitdiskretisierung.

```
1 #Define T0=0
2 sqr = domain.integral('(un - 0)^2 dV')
3 lhsun = solver.optimize('lhsun', sqr)
4
5 #Define the PDE
6 res = domain.integral('(basis_n u + kappa dt ∇_i(basis_n) ∇_i(u) - basis_n un) dv' @ ns,
7                       degree = degree * 2)
8 #Add boundary conditions
9 res -= domain.boundary['left'].integral('dt kappa basis_n dS' @ ns, degree = degree * 2)
10
11 for n in range(num_steps):
12   lhs = solver.solve_linear('lhs', res, arguments = dict(lhsun = lhsun))
13   lhsun = lhs
```

Abbildung 3.5: Berechnung der Finite Elemente Methode in Nutils

3.4.2 Inversesproblem mit AD

Bei der Anwendung des automatischen Differenzierens in Nutils sind Probleme aufgekommen. Es hat sich herausgestellt, dass es für inverse Probleme nicht anwendbar ist. Aus Autorenbeispielen zeigt sich, dass die Implementierung der automatischen Differenzierung für nichtlineare Probleme gedacht ist.
Wie bei der automatischen Differenzierung erhalten wir über die Finite Differenz die Ableitung, jedoch in diesem Falle über eine Approximation mit einstellbarer Fehlertoleranz. Die Berechnung beruht auf den Grundsätzen der Differenzialrechnung, denn für die Berechnung unserer Ableitung $\frac{\partial J}{\partial \kappa}$ gilt:

$$\frac{\partial J}{\partial \kappa} = \lim_{\triangle\kappa \to 0} \frac{J(\kappa + \triangle\kappa) - J(\kappa)}{\triangle\kappa} + O(\triangle\kappa) \tag{3.1}$$

Somit erhalten wir einen Fehler in der Größenordnung von $\triangle\kappa$. [7]
Demnach müssen wir zur Berechnung unserer Ableitung $\frac{\partial J}{\partial \kappa}$ zum einen die Fehlerfunktion oder auch Kostenfunktion genannt $J(\kappa)$ an zwei verschiedenen Stellen auswerten. Dafür müssen wir im Gegensatz zu FEniCS, im Skript inverse_nutils.py zunächst zwei Forwärtssimulationen berechnen, um mit diesen dann, wie in den Zeilen 12 und 13 der folgenden Abbildung, mithilfe der von Nutils stammenden eval Funktion, die jeweiligen Fehlernormen einzeln zu bestimmen. Anschließend können wir dann die Approximation unserer Ableitung in Zeile 15 berechnen. Damit haben wir alles gegeben, um das neue κ laut den Iterationsvorschriften des Gradientenverfahrens zu berechnen.

```python
#Inverse problem parameters
kappaGuess = 0.3
step_size = 1e-1
...
fd_kappa = 1e-5

while dkappa > tol or it < it_max:
  it += 1
  lhs = forward_problem(ns, degree, domain, num_steps, kappa)
  dlhs = forward_problem(ns, degree, domain, num_steps, kappa + fd_kappa)

  Jeval = J.eval(lhs = lhs,lhsref = lhsref)
  dJeval = J.eval(lhs = dlhs, lhsref = lhsref)

  dJdkappa = (dJeval - Jeval)/ fd_kappa
  dkappa = -step_size * dJdkappa
```

Abbildung 3.6: Implementierung des Inversen Problems über die Finiten Differenzen

4 Untersuchung der Bibliotheken

In diesem Kapitel werden die erzielten Ergebnisse der Bibliotheken aufgezeigt und miteinander verglichen. Dadurch erhalten wir eine Beurteilungsgrundlage zur Ausarbeitung des Fazits.
Der erste Negativpunkt trat im Zuge der Installation von FEniCS auf, denn obwohl FEniCS ein pip Modul anbietet, konnte dieses auch bei mehreren verschiedenen Versuchen und Intepreterversionen nicht ans Laufen gebracht werden. Demnach musste auf die ausschließlich Benutzung des Dockercontainers innerhalb der Kommandozeile ausgewichen werden. Im Dockercontainer traten weiteren Schwierigkeiten auf. Bei der Berechnung von sehr genauen Meshes, vielen Zeitschritten oder auch der Kombination beider kam es dazu, dass der Dockercontainer aus unbekannten Gründen den Vorgang terminierte. Dies konnte auch nicht durch die Zuweisung von mehr Arbeitsspeicher oder ähnlichem umgangen werden. Zusätzlich sei zu dem pip-Modul gesagt, dass obwohl es für die Untersuchungen auf einem verwendbaren Stand gewesen wäre, die Bibliothek seid längerer Zeit nicht mehr gepflegt wurde. Im Vergleich dazu hat die Installation von Nutils per pip ohne irgendwelche Schwierigkeiten funktioniert und wird ebenso noch aktiv gewartet. Während der Untersuchungen wurde eine neue Version veröffentlicht, die innerhalb weniger Tage auch über pip nutzbar war. Im Gegensatz zu der Einbindung eines pip-Moduls in eine laufende Entwicklungsumgebung ist die Variante über den Dockercontainer schwieriger.
Bei der zeitlichen Auswertung, wie in Kapitel 3.1 beschrieben, sind für die Berechnung des Forwärtsproblems große Unterschiede festzustellen, während FEniCS für die Berechnung des Stabes mit einem Mesh von 1600 Unterteilungen in X-Richtung und 2 in Y-Richtung durchschnittlich 2,85 Sekunden gebraucht hat, waren es bei Nutils 108,42 Sekunden oder 1 Minute und rund 46 Sekunden.
Relevant für die Einordnung der zeitlichen Diskrepanzen sind die erzielten Genauigkeiten der Lösungen.

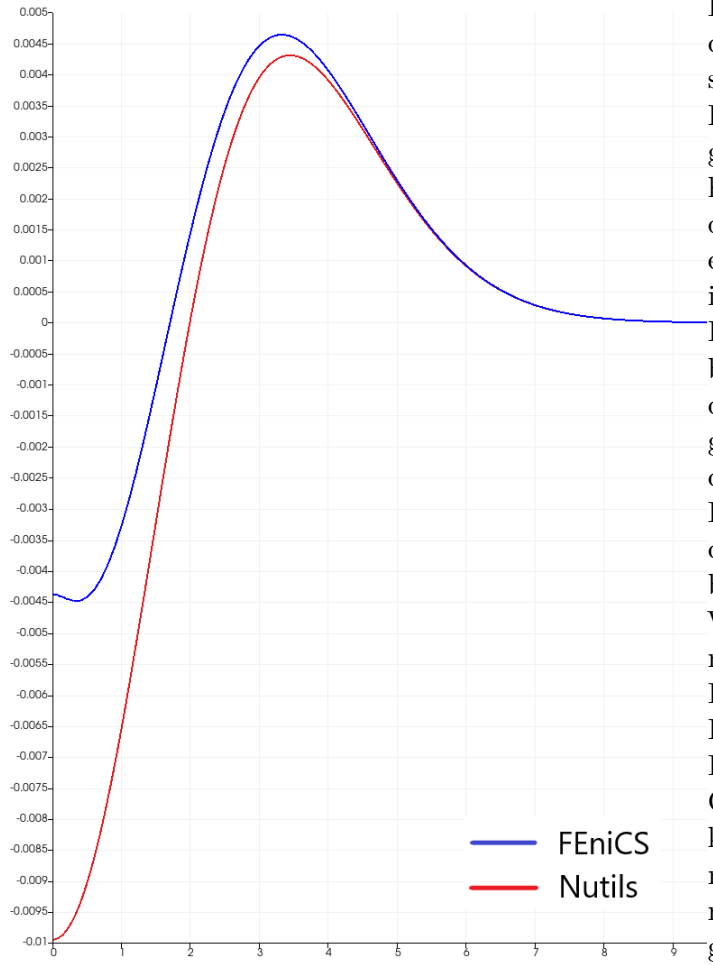

Abbildung 4.1: Fehler im Verhältnis zur Analytischen Lösung

In der nebenstehenden Grafik 4.1 wurde der Fehler zwischen der analytischen und der jeweils berechneten Lösung bis zur 9ten Längeneinheit gegeneinander geplottet. Interessant ist hierbei der Unterschied zwischen beiden Lösungen, während Nutils auf den ersten 2 Längeneinheiten des Stabes im Verhätlnis deutlich schlechter als FEniCS ist, hat Nutils von der 2ten bis zur 5ten Längeneinheiten einen durchgehenden geringen Vorteil gegenüber FEniCS. Anschließend sind die Fehler beider Lösungen über den Rest des Stabes nahe zu identisch und deshalb nicht in der Abbildung 4.1 berücksichtigt. Um nun aber einen Vergleich über den kompletten Stab zu machen, wird im folgendenden die L2-Fehlernorm zum Zeitpunkt t=1 der Berechnung betrachtet. Denn der L2-Fehler ist das Integral der jeweiligen Graphen, aus Abbildung 4.1, über den kompletten Stab und repräsentiert somit die gesamt Genauigkeit der Berechnungen. Da bei den Berechnungen zwei Diskretisierungen vorgenommen werden kann der Fehler in einen örtlichen und zeitlichen Fehler unterteilt werden.

Somit kann im nächsten Schritt mithilfe von im folgenden beschriebenen Tests, bestimmt werden ab welcher Fehlergenauigkeit, der Fehler von dem Ort oder der Zeit dominiert wird. Im ersten Test wird die Veränderung des L2-Fehlers bei stetig feiner werdendem Gitter und konstantem $\triangle t$ betrachtet. Beim zweiten Test besitzt dann das Gitter eine feste Größe, während $\triangle t$ kleiner wird. In der Grafik 4.2 ist die logarithmische Fehlernorm im Zusammenhang zu dem immer feiner werdenen Mesh für 128 Zeitschritte abgebildet. Grafik 4.3 zeigt logarithmisch den Einfluss der Zeitdiskretisierung auf den Fehler für verschiedene $\triangle t$. Beim betrachten der beiden Grafiken wird deutlich, dass Nutils die genaueren Berechnungen hat. Zunächst zeigen die Grafiken 4.2 und 4.3 nur die L2-Fehlernorm in Abhängigkeit von den oben genannten Größen. Betrachtet man aber nun die Grafik 4.2 fällt auf, dass der L2-Fehler beider Bibliotheken ab einem Wert von rund 10^{-6} anfängt zu konvergieren oder nur noch marginal besser wird. Somit kann keine Verbesserung der Genauigkeit für feiner werdende Gitter als 10^{-6} , bei konstantem $\triangle t$ erreicht werden. In der nebenstehenden Grafik 4.3 sinken die Fehler, bei Betrachtung eines konstant großen Gitters und immer kleiner werdendem $\triangle t$ für FEniCS bis knapp unter 10^{-6} und für Nutils definitiv bis 10^{-7} , wenn nicht noch weiter. Diese beiden Grafiken können nun

ins Verhältnis gesetzt werden, da sie größentechnisch mit den selber Werten arbeiten. Daraus zeigt sich, dass jeweils der zeitliche Fehler ab rund 10^{-6} dominiert und zusätzlich dazu der Vergleich der Genauigkeit. Nun zeigt sich im Minimum des blauen Graphens von Abbildung 4.3, die geringste Fehlernorm, welche FEniCS für die ausgewählten Größen erreichen kann. Im Gegensatz dazu ist erkennbar, dass die Fehlernorm für Nutils noch nicht konvergiert ist und somit eine bessere numerische Lösung erreicht.

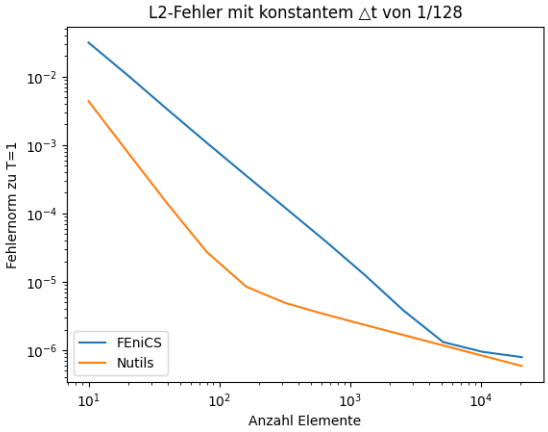

Abbildung 4.2: Fehlernorm Test 1

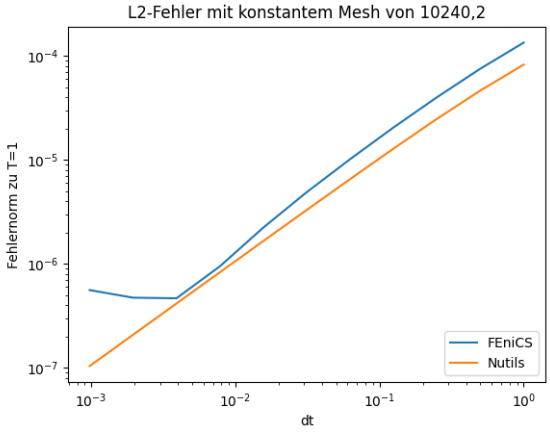

Abbildung 4.3: Fehlernorm Test 2

Auch für die Inversen Problem verändert sich der zeitliche Unterschied nicht, sondern wird aufgrund der verwendeten Finiten Differenzen bei Nuitls noch weiter verstärkt. Denn nicht nur braucht Nutils generell länger für die Berechnungen einer Forwärtssimulation, sondern benötigt zusätzlich dazu eine Forwärtssimulation mehr pro Iteration als FEniCS.
Die folgenden Plots 4.4 und 4.5 verdeutlichen neben dem zeitlichen Aspekt auch die Genauigkeit der beiden Bibliotheken. Alle Tests werden mit einer Referenztemperatur unter der gegebene Annahme von $\kappa = 1$ durchgeführt. In der ersten Grafik 4.4, auf welche sich im folgenden bezogen wird, sehen wir die Berechnung des Inversen Problems in FEniCS, also unter Verwendung des automatischen Differenzierens und unter Verwendung von verschiedenen konstanten Schrittweiten. Die Berechnungen mit $\alpha = 1$ und $\alpha = 0.7$ divergieren in Richtung eines negativen κ ,was physikalisch nicht möglich ist. Währenddessen konvergiert der rote Graph aufgrund des hohen $\alpha = 0.5$ in Sprüngen um den Konvergenzpunkt erst sehr spät. Für $\alpha = 0.1$ sehen wir das genaue Gegenstück, nämlich eine relativ langsame Konvergenz aufgrund der kleinen Schrittweite. Der beste konstante Wert aus unseren ausgewählten Schrittweite für α ist 0.3, da dieser schon nach rund 3 Iterationen einen sehr genauen Wert liefert.
Im Vergleich dazu zeigt die gegenüber gestelte Grafik 4.5 die Berechnung des Inversen Problems in Nutils über die Finite Differenz mit einer Schrittweite der Finiten Differenzen von $1e{-}5$. Auch bei Nutils divergieren die Berechnungen für $\alpha = 1$ und $\alpha = 0.7$, wobei die unterschiedlichen Werte aufgrund dessen nur geringe Aussagekfraft haben. Dafür sieht man für den orangen Graphen mit einem $\alpha = 0.1$ sehr gut, wie langsam die Nutils Lösung im Vergleich zu FEniCS konvergiert. Interessant dagegen ist der Vergleich der Graphen für $\alpha = 0.3$ und $\alpha = 0.5$ zwischen FEniCS und Nutils. Denn hier sieht man, wie unterschiedlich die Näherungslösung von $\frac{\partial J}{\partial \kappa}$ über die Finiten Differenzen im Vergleich zu der genauen Lösung über das automatische Differenzieren sein kann und so die Konvergenz bei Nutils beeinflusst.

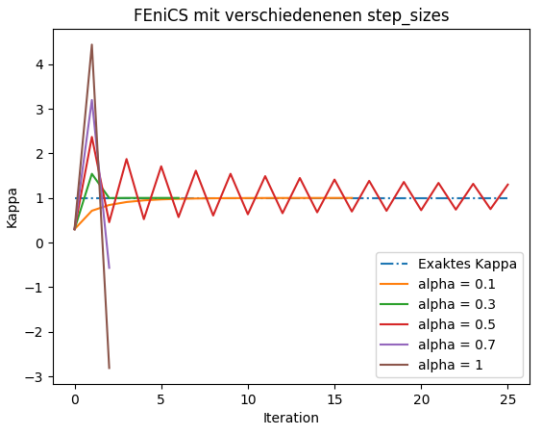

Abbildung 4.4: FEniCS

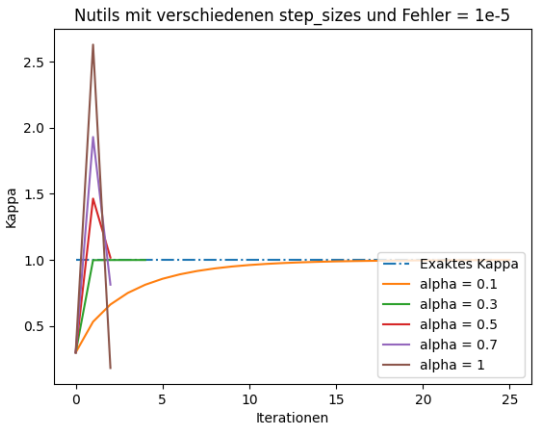

Abbildung 4.5: Nutils

Die oben benutzte Fehlertoleranz ist mit bedacht ausgewählt, da sie in Relation zum Zeitaufwand gute Daten produziert.

5 Fazit

In dieser Seminararbeit wurde mithilfe zweier Bibliotheken die Implementierung einer Finiten Elemente Methode, sowie einem Inversen Problem unter Verwendung des Gradientenverfahren für ein Wärmeleitungsbeispiel entwickelt. Die verwendeten externen Bibliotheken sind FEniCS und Nutils und wurden jeweils mithilfe von Python eingebunden.
Für die Bewertung der Bibliotheken sind vor allem die Rechengeschwindigkeit, sowie die Genauigkeit der Lösung wichtig, aber ebenso die äußeren Merkmale, wie die Installation und Bedienungsfreundlichkeit. Beim Punkt der Installation und der Bedienungsfreundlichkeit hat Nutils einen klaren Vorteil gegenüber FEniCS, denn nicht nur ist es über jede beliebige Entwicklungsumgebung zu bedienen, sondern auch die Pythonanwendung der Finiten Elemente Methode ist bei Nutils simpler.
Die Genauigkeit beider Bibliotheken im Vergleich zu der analytischen Lösung sind beide gut, sodass eine klar bessere Bibliothek nicht eindeutig bestimmt werden kann. Jedoch kommt genau hier der zeitliche Aspekt zur Geltung. Denn wie in Kapitel 4 aufgezeigt, ist FEniCS zeittechnisch im direkten Vergleich viel performanter als es Nutils ist. In Zahlen ausgedrückt brauchte Nutils rund 38 Mal länger für dieselbe Berechnung als FEniCS. Wenn nun aber der Verbrauch an Ressourcen, wie Strom oder auch die Zeit, vernachlässigt werden, hat Nutils aufgrund seiner genaueren Lösungen einen Vorteil gegenüber FEniCS. Da wir dies aber nicht unbedingt für jeden Standardnutzer voraussetzen können, ist FEniCS besser zu bewerten als Nutils.
Dies wird noch deutlicher, wenn wir als weiteren Faktor das Inverse Problem hinzuziehen. Denn auch wenn wir am Ende durch beide Bibliotheken zu Lösungen gekommen sind, konnte sich FEniCS aufgrund des verwendbaren automatischen Differenzierens auch hier gegen Nutils durchsetzen. Denn wie schon in 3.4.2 gezeigt, mussten wir in Nutils unser Inverses Problem durch die Verwendung von Finiten Differenzen bestimmen, da das automatische Differenzieren nicht für dieses konkrete Beispiel einsetzbar ist. Eine weitere Verbesserung für das inverse Problem in FEniCS kann durch die Verwendung einer eingebauten Methode für die Minimierung durch verschiedene Algorithmen erreicht werden.
Auch wenn die Seminararbeit gezeigt hat, dass beide Bibliotheken zur Berechnung der Lösung, sei es direkt oder indirekt über Umwege, fähig sind, überwiegt aufgrund der o.g. Aspekte die Bibliothek FEniCS.

A Literaturverzeichnis

[1] G. N. Wells et al A. Logg, K.-A. Mardal. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[2] Marek Behr. Simulationstechnik (teil ii). Notizen zur Lehrveranstaltung, 2021.

[3] Cengel. *Heat transfer: A practical approach*. McGraw Hill Higher Education, Maidenhead, England, 2 edition, November 2002.

[4] Yunus Cengel and Afshin Ghajar. *Heat and mass transfer, 6th edition, Si units*. McGraw-Hill Education, Singapore, Singapore, 6 edition, September 2020.

[5] Jean Donea and Antonio Huerta. *Finite Element Methods for Flow Problems*. Wiley, April 2003.

[6] Ray Hsu. Gradient descent. https://medium.com/geekculture/gradient-descent-a453b1943c8b am 12.01.2023, 2022.

[7] Roland W Lewis, Perumal Nithiarasu, and Kankanhalli N Seetharamu. *Fundamentals of the finite element method for heat and fluid flow*. John Wiley & Sons, Ltd, Chichester, UK, April 2004.

[8] J. Hake M. S. Alnaes, J. Blechta. The FEniCS project version 1.5. *Archive of Numerical Software*, 3, 2015.

[9] Uwe Naumann. *The Art of Differentiating Computer Programs*. Society for Industrial and Applied Mathematics, 2011.

[10] Andreas Rieder. *Einführung: Was ist ein inverses Problem?*, pages 1–19. Vieweg+Teubner Verlag, Wiesbaden, 2003.

[11] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[12] Peter Stephan, Karlheinz Schaber, Karl Stephan, and Franz Mayinger. *Thermodynamik*. Springer-Lehrbuch. Springer, Berlin, Germany, 18 edition, January 2009.

[13] J.S.B. van Zwieten, G.J. van Zwieten, and W. Hoitinga. Nutils 7.0, 2022.

B Abbildungsverzeichnis