OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

ARGES

BACHELORARBEIT

Simon Schindler

Anwendung und Vergleich verschiedener nichtlinearer Optimierungsalgorithmen für die optimale Steuerung von Galvanometer-Spiegel Systemen

2. April 2024

Fakultät:	Elektro- und Informationstechnik
Studiengang:	B.Eng. Elektro- und Informationstechnik
Abgabefrist:	15. September 2020
Betreuung:	Prof. Dr. Hans Meier
Zweitbegutachtung:	Prof. Dr. Norbert Balbierer
Externe Betreuung:	M.Sc. Bernhard Kiesbauer

Inhaltsverzeichnis

1. Einleitung

Aufgrund ihrer hohen Präzision und Prozessgeschwindigkeit spielen Laser unter anderem in der Materialbearbeitung, der additiven Fertigung und auch in der Medizintechnik eine wichtige Rolle. Zur korrekten Positionierung und Fokussierung von Laserstrahlen werden dabei Laserscanköpfe, wie sie die Fa. Arges entwickelt und fertigt, eingesetzt. In diesen sorgen auf Galvanometern montierte Spiegel für die geforderte Umlenkung des Lasers. Um eine zu jedem Zeitpunkt möglichst geringe Abweichung der Position des Laserstrahls von seiner geforderten Position zu erreichen und dabei eine hohe Dynamik zu bewahren, eignet sich der Einsatz digitaler Regelungssysteme. Für diese ist eine exakte Modellierung der Galvanometer-Spiegel Systeme von großem Vorteil. In verschiedenen Verfahren zur Systemidentifikation werden während der Produktion jedes Laserscankopfes die relevanten Modellparameter ermittelt und ein mathematisches Modell erstellt. Oftmals müssen die Systeme dabei möglichst schnelle Sprünge durchführen. Da die Ansteuerungselektronik der Galvanometer nicht unbegrenzt hohe Spannungen und Ströme liefern kann, ist es nötig, das Steuersignal zu optimieren, um die maximalen auftretenden Spannungen und Ströme zu minimieren. Ein solcher Minimax Sprung, so schnell wie möglich und die Systemgrenzen maximal ausreizend, zeigt die elektromechanischen Grenzen des Gesamtsystems auf und findet damit Anwendung sowohl bei der Bewertung der implementierten digitalen Regelungsalgorithmen als auch bei der Beurteilung der Systemdynamik.
Des Weiteren kann auch die Modellqualität durch Anwendung dieser auf Basis des Modells berechneten Sprünge evaluiert werden, indem nur das Steuersignal angelegt und auf eine Regelung verzichtet wird. Der Fehler zwischen Soll- und Ist-Zustand während und nach dem Sprung kann Aufschluss über etwaige Modellierungsfehler geben.
Zu guter Letzt können aus den „minimax“-Sprüngen auch optimale Koeffizienten zur digitalen Signalverarbeitung im Regelungssystem berechnet werden.
Die Kalkulation dieser Sprünge erfolgt dabei in Matlab und ist deshalb nur auf Rechnern mit einer entsprechenden Lizenz möglich.
Im Rahmen dieser Arbeit soll ein möglichst robuster und effizienter Algorithmus zur statischen, nichtlinearen Optimierung des Steuersignal in der Programmiersprache C auf einem ARM-Cortex-M7 Microcontroller implementiert werden.
Dies dient dem Zweck, eine lizenzunabhängige Erstellung der Sprünge auf einem Microcontroller zu ermöglichen und dabei die dafür benötigte Rechenzeit zu ermitteln. Somit kann Aufschluss darüber gegeben werden, ob die Filterkoeffizienten auch im laufenden Betrieb aktualisiert werden könnten.

Diese Arbeit verfolgt in erster Linie das Ziel, eine zur Optimierung des Steuersignals durch die Matlabfunktion „fminimax“ equivalente Implementierung zu erstellen. Weniger liegt eine dynamische Optimierung von Steuersignalen zur Laufzeit im Fokus, weshalb sich die Arbeit bewusst ausschließlich mit statischen Optimierungsverfahren beschäftigt.

Struktur der Arbeit

Die grundsätzliche Vorgehensweise zur Bearbeitung und Lösung von Optimierungsproblemen stellen Papageorgiou et al. in [8] durch Abbildung 1 dar. Dieser

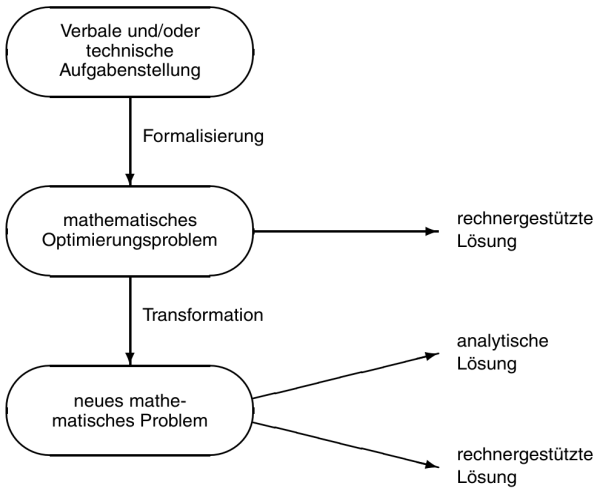

Abbildung 1: Vorgehensweise zur Lösung von Optimierungsproblemen aus [8]

Vorgehensweise folgend, führt Kapitel 3 zunächst über die Idee für die Berechnung des optimalen Sprungs zur technischen Aufgabenstellung hin und formalisiert im ersten Ansatz schon ein mathematisches Optimierungsproblem. Anschließend wird in Kapitel 4 zur Vollständigkeit und besseren Verständlichkeit der Arbeit auf die Theorie der nichtlinearen Optimierung eingegangen. Darauf folgend wird in Kapitel 5 der erste Ansatz für das mathematisch beschriebene Optimierungsproblem in eine passende Kostenfunktion umgewandelt und weitere Aspekte für den Optimierungsalgorithmus eruiert. Schließlich werden in Kapitel 6 verschiedene Algorithmen zur rechnergestützten Lösung erläutert und angepasst. Im letzten Kaptitel 7 werden sie auf dem Microcontroller miteinander verglichen. Die aus diesem Vergleich entstehenden Ergebnisse sollen als Anhaltspunkt für zukünftige Arbeiten mit den Optimierungsalgorithmen auf dem verwendeten Microcontroller dienen. Außerdem wird in Kapitel 2 kurz auf die für diese Arbeit verwendete Hard- und Software eingegangen.

2. Hard- und Softwareumgebung

2.1. Hardware

In dieser Arbeit werden alle Versuche zu Implementierungen in C auf dem Entwicklungsboard NUCLEO-H755ZI-Q durchgeführt. Dieses verfügt mit dem STM32-H755ZI sowohl über einen ARM-M4 als auch über einen ARM-M7 Rechenkern, wobei ausschließlich letzterer aufgrund seiner höheren maximalen Taktrate von 480 MHz und seiner 64-bit Fließkomma-Recheneinheit genutzt wird. Die Messun-

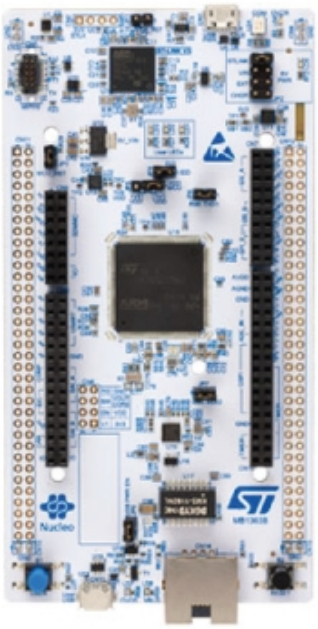

Abbildung 2: NUCLEO-H755ZI-Q Board

gen von Rechenzeiten erfolgen dabei durch Nutzung des General-Purpose-Timers TIM2 mit einer Messungenauigkeit $\leq 1\mu s$.
Der auf dem Entwicklungsboard integrierte zweite ARM-Prozessor STM32F723 fungiert als Programmierschnittstelle, als Debugger und als virtual COM-Port.
Über die USB-Schnittstelle eines PCs werden dem Board 5 V und max. 0.5 A bereitgestellt, wobei der STM32H755ZI über einen Low-Dropout Regler mit 3.3 V Betriebsspannung versorgt wird.

2.2. Software

Als Entwicklungsumgebung für den Microcontroller wird die auf Eclipse basierende STMCubeIDE, zum Kompilieren und Linken des Codes GNU GCC verwendet. Neben C in VS Code wird für die konzeptionellen Implementierungen der Algorithmen, sowie für die Systemsimulation der Galvanometer Matlab verwendet.

3. Der optimale Steuervektor

Die zugrundeliegende Idee zur Berechnung des optimalen Steuersignals für einen Sprung beschreibt Christian Reil in [1], verfolgt sie jedoch nicht weiter. Jene Idee wird in dieser Arbeit wieder aufgegriffen und für eine bessere Verständlichkeit und Vollständigkeit im folgenden Kapitel erörtert.
Die Systembetrachtung erfolgt dabei in der diskreten Zustandsraumdarstellung:

$$\vec{x}(k+1) = A\vec{x}(k) + Bu(k) \tag{1}$$
$$y(k) = C\vec{x}(k) + Du(k) \tag{2}$$

$\vec{x}(k)$ entspricht dem Systemzustand zum Abtastzeitpunkt k. Gleichung (1) beschreibt die Entwicklung des Zustandes $\vec{x}(k)$ zum Zustand $\vec{x}(k+1)$, abhängig von der Systemmatrix A und dem mit B gewichteten Eingangssignal $u(k)$ zum Abtastzeitpunkt k.
Der Ausgangsvektor $\vec{y}(k)$ bildet sich aus dem Systemzustand $\vec{x}(k)$, gewichtet mit der Ausgansmatrix C und, falls das System sprungfähig ist, dem Eingangssignal $\vec{u}(k)$, gewichtet mit der Durchgangsmatrix D.
Zur Berechnung eines optimalen Stellgrößenverlaufs ist zunächst ein Stellsignal $\vec{u} = [u(0), u(1), ..., u(N-1)]^T$ gesucht, mit dem das SISO System der Ordnung m von einem Ausgangszustand $\vec{x}(0)$ zu einem Endzustand $\vec{x}(N)$ gebracht werden kann.
Der Endzustand $\vec{x}(N)$ zum Abtastzeitpunkt N kann durch eine Iteration der Zustandsgleichung (1), ausgehend von dem Zustand $\vec{x_0} = \vec{x}(0)$ und dem Eingangssignal $\vec{u} = [u(0), u(1), ..., u(N-1)]^T$, bestimmt werden:

$$\begin{array}{rcrcrcrcl}
\vec{x}(1) = A\vec{x_0} & + & Bu(0) & & & & & & \\
\vec{x}(2) = A^2\vec{x_0} & + & ABu(0) & + & Bu(1) & & & & \\
\vec{x}(3) = A^3\vec{x_0} & + & A^2Bu(0) & + & ABu(1) & + & Bu(2) & & \\
\vdots & \vdots & \vdots & \vdots & & & & & \\
\vec{x}(N) = A^N\vec{x_0} & + & A^{N-1}Bu(0) & + & A^{N-2}Bu(1) & + & \dots & & \\
 & + & ABu(N-2) & + & Bu(N-1) & & & &
\end{array} \tag{3}$$

Die letzte Iteration kann auch in Summenschreibweise durch die Bewegungsgleichung der diskreten Zustandsraumdarstellung dargestellt werden:

$$\vec{x}(N) = A^N \vec{x_0} + \sum_{n=0}^{N-1} A^n B u(N-1-n) \tag{4}$$

Durch Zuhilfenahme der Matrix $Q_N = [A^{N-1}B, A^{N-2}B, ..., A^2B, AB, B]$ ist eine Schreibweise der Bewegungsgleichung auch in Matrixnotation möglich:

$$\vec{x}(N) = A^N \vec{x_0} + Q_N \vec{u} \tag{5}$$

3.1. Dead Beat Lösung

Bei Q_N handelt es sich im Falle $N = m$ um die Steuerbarkeitsmatrix des Systems. Ist ihr Rang maximal, $rg(Q_N) = N$, und die Matrix damit regulär, also invertierbar, gilt das System nach dem Steuerbarkeitskriterium von Kalman als vollständig steuerbar [7]. Unter der vereinfachenden Annahme $\vec{x_0} = 0$ gilt in diesem Fall:

$$\begin{aligned} \vec{x}(N = m) &= Q_N \vec{u} \\ Q_N^{-1} \vec{x}(N) &= \vec{u} \end{aligned} \tag{6}$$

Bei der aus (6) berechneten Stellgröße u handelt es sich um die sog. Dead Beat Lösung. Derartige Dead Beat Lösungen finden z.B. in der Prozesssteuerung Anwendung. Ein entscheidender Nachteil dabei ist jedoch, dass die Stellgröße teilweise nicht realisierbare Amplituden annimmt.
In Abbildung 3 ist eine Simulation der Dead Beat Lösung für das Modell der Ordnung $m = 3$ (Modell: A.1) eines Galvanometer-Spiegel Systems bei einer Abtastrate von 20 kHz zu sehen. Innerhalb der ersten drei Samples führt das System einen Sprung der Höhe 1° durch. Die dafür maximal benötigte Spannung beträge jedoch ca. 20,6 kV, der dabei maximal fließende Strom ca. 3 kA. Natürlich sind derartig große Werte durch die eingesetzte Treiberelektronik nicht realisierbar. Außerdem ist fraglich, ob das System die bei einer solch hohen Beschleunigung auftretenden Kräfte aushalten würde.
Um die maximal benötigte Spannung zu verringern, muss deshalb entweder die Sprunghöhe verringert werden oder der Sprung innerhalb einer größeren Zeitspanne, also innerhalb mehrerer Samples ($N > m$), erfolgen.

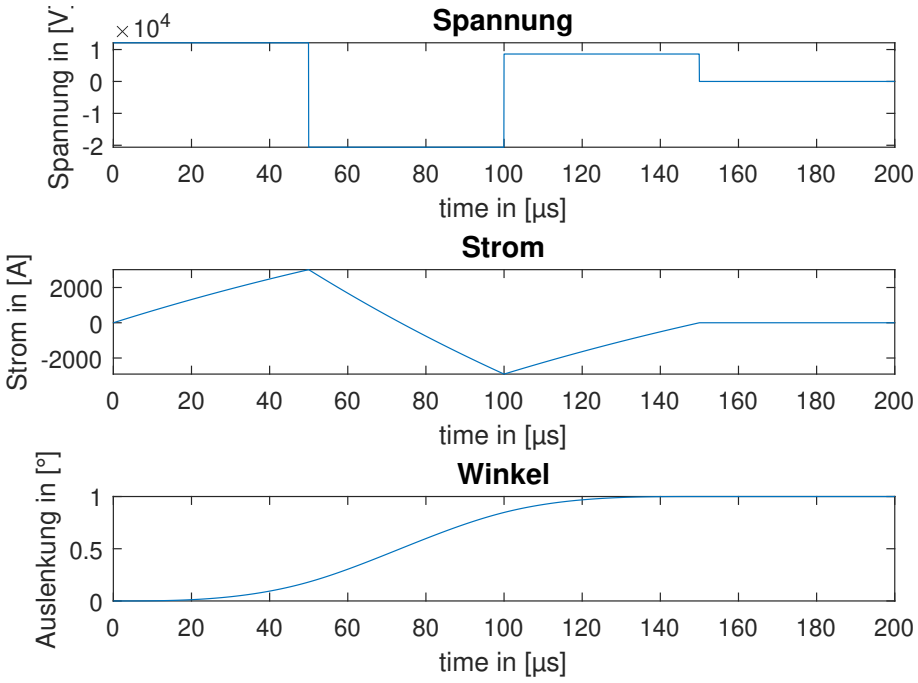

Abbildung 3: Dead Beat Lösung

3.2. Least Squares Lösung

Soll ein Sprung über eine längere Zeitspanne und damit über mehr Samples ($N > m$) ausgeführt werden, existiert ein ganzer Raum an möglichen Steuervektoren $\vec{u}$, um das unterbestimmte $m \times N$ Gleichungssystem

$$\vec{x}(N > m) = Q_N \vec{u} \tag{7}$$

zu erfüllen, weiterhin unter der Annahme $\vec{x_0} = 0$.
Eine Möglichkeit stellt die Least Squares Lösung (LS Lösung) dar:

$$\vec{x}(N) = Q_N \vec{u}_{LS} \tag{8}$$

Diese minimiert das Funktional:

$$J_N = \|Q_N \vec{u} - \vec{x}(N)\|^2 \tag{9}$$

Erweitert man (7) um Q_N^T und löst nach $\vec{u}$ auf, erhält man eine Beschreibung der Stellgröße $\vec{u}_{LS}$ durch das $N \times N$ Gleichungssystem:

$$Q_N^T Q_N \vec{u}_{LS} = Q_N^T \vec{x}(N) \tag{10}$$

$\vec{u}_{LS}$ ist durch (10) eindeutig bestimmbar, falls die Spalten von Q_N voneinander linear unabhängig sind und damit die resultierende $N \times N$ Matrix $Q_N^T Q_N$ invertierbar ist:

$$\vec{u}_{LS} = (Q_N^T Q_N)^{-1} Q_N^T \vec{x}(N) \tag{11}$$

Anderenfalls kann $\vec{u}_{LS}$ auch über Q_N^+, die Moore Penrose Pseudoinverse von Q_N, bestimmt werden:

$$\vec{u}_{LS} = Q_N^+ \vec{x}(N) \tag{12}$$

$$\text{mit } Q_N^+ = UD^{-1}V^T \tag{13}$$

Wobei man U,V und D durch die Singulärwertzerlegung von Q_N erzeugt. Ab-

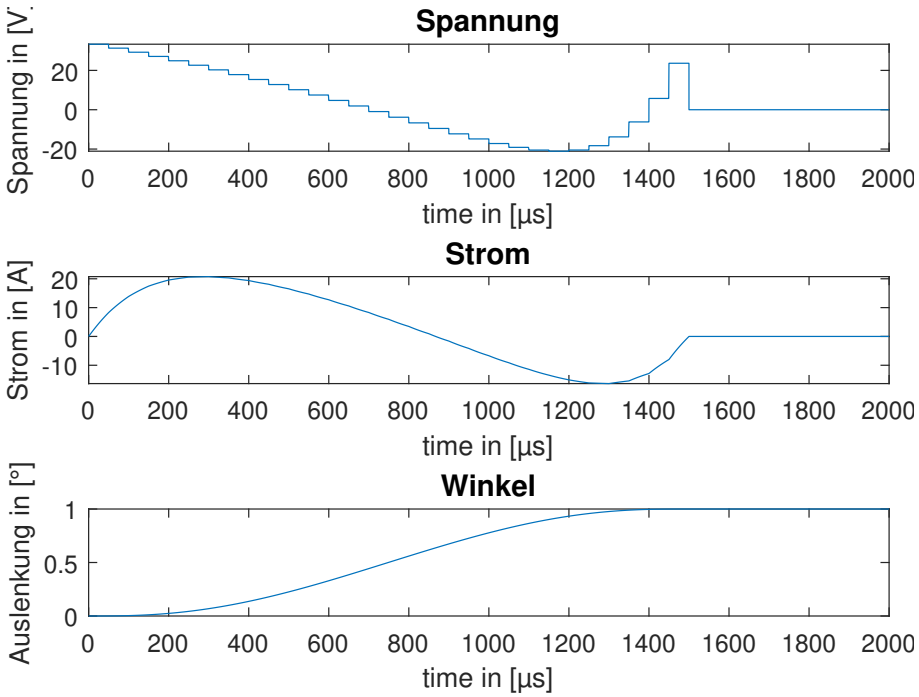

Abbildung 4: Least Squares Lösung

bildung 4 zeigt einen über die Moore Penrose Inverse berechneten Sprung des selben Systems wie in 3.1 (Modell: A.1) und der Sprunghöhe 1° innerhalb von 30 Samples mit einer Abtastrate von 20kHz. Die dabei maximal benötigte Spannung beträgt 33,32 V und der maximal benötigte Strom 20,75 A. Diese Werte liegen innerhalb der Systemschranken der bei Arges üblicherweise verbauten Treiberelektronik. Ein derartiger Sprung wäre folglich realisierbar.

3.3. Minimax Lösung

Das in 3.2 beschriebene Eingangssignal $\vec{u}_{LS}$ der Länge N kann für die Suche nach einem optimalen Steuervektor $\vec{u}_{opt}$ verwendet werden, wobei $\vec{u}_{opt}$ aus folgender Gleichung entsteht:

$$\vec{u}_{opt} = \vec{u}_{LS} + Z\vec{u}_{var} \tag{14}$$

Bei $Z \in \mathbb{R}^{N\times(N-m)}$ handelt es sich um den Nullraum von Q_N. Eine Variation von $\vec{u}_{var} \in \mathbb{R}^{N-m}$ ändert deshalb nichts an dem Endzustand $\vec{x}(N)$, da gilt:

$$\begin{aligned}
& Q_N Z = 0 \\
& Q_N Z\vec{u}_{var} = 0 && (15)\\
\text{mit (15) in (8)} \quad & Q_N\vec{u}_{LS} + Q_N Z\vec{u}_{var} = \vec{x}(N) + 0 \\
& Q_N(\vec{u}_{LS} + Z\vec{u}_{var}) = \vec{x}(N) \\
& Q_N\vec{u}_{opt} = \vec{x}(N) && (16)
\end{aligned}$$

Ziel der Optimierung durch Variation von $\vec{u}_{var}$ ist eine Minimierung der maximalen Amplitude einer ausgewählten, im System auftretenden Größe (z.B. Strom, Spannung oder Winkelbeschleunigung).
Das dabei zu lösende Minimax Problem kann folgendermaßen beschrieben werden:

$$\begin{gathered}
\min_{\vec{u}_{var}\in\mathbb{R}^{N-m}} \psi(\vec{u}_{var}), \\
\text{mit } \psi : \mathbb{R}^{N-m} \to \mathbb{R} \text{ definiert durch} \\
\psi(\vec{u}_{var}) = \max_{k\in K} f_k(\vec{u}_{var})
\end{gathered} \tag{17}$$

Die Abbildungen $f_k : \mathbb{R}^{N-m} \to \mathbb{R}, k \in K = [1, 2, 3..., N-1, N]$ beschreiben die Beträge der durch das Eingangssignal $\vec{u}_{opt}$ auftretenden Amplituden der zu minimierenden Größe. In der Simulation zu Abbildung 5 wurde als solche die Stellgröße $\vec{u}_{opt}$, also die am Galvanometer anliegende Spannung gewählt:

$$f_k(\vec{u}_{var}) = |u_{LS,k} + (z_{k,1}, z_{k,2}, ..., z_{k,N-m-1}, z_{k,N-m})\vec{u}_{var}| \tag{18}$$

$$\text{mit } \vec{u}_{LS} = \begin{pmatrix} u_{LS,1} \\ \vdots \\ u_{LS,N} \end{pmatrix} \tag{19}$$

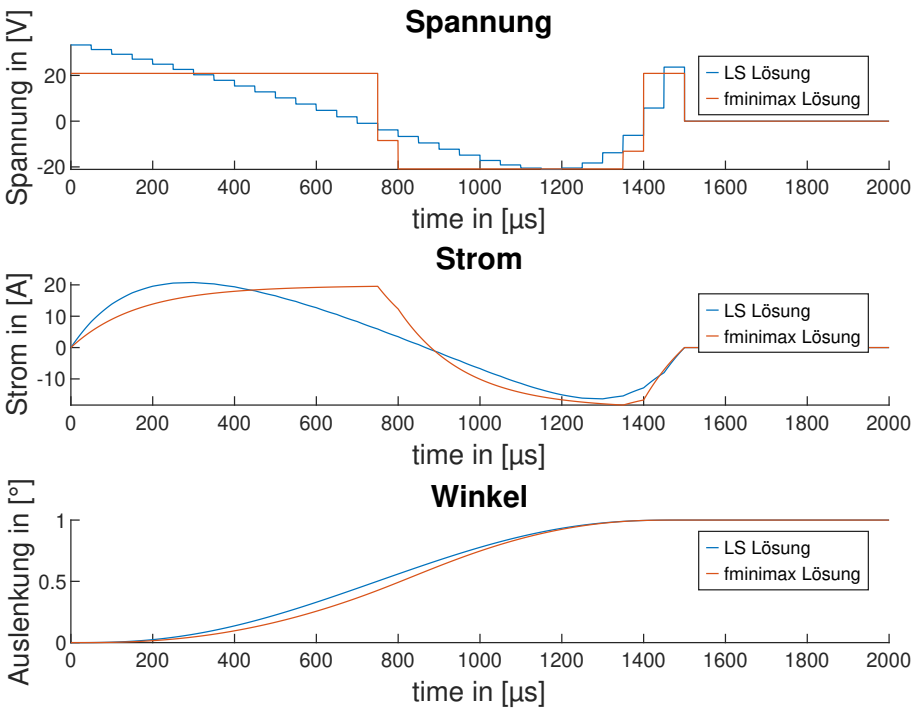

Abbildung 5: Minimax Lösung

Abbildung 5 zeigt die auf Basis der Least Squares Lösung aus 3.2 berechnete Minimax Lösung. Es ist deutlich erkennbar, dass die maximal benötigte Spannung der Minimax Lösung (orange) mit 20,89 V niedriger ist als die der LS Lösung (blau) mit 33,32 V. Zur Berechnung von $\vec{u}_{opt}$ wurde dabei die in der Optimization Toolbox enthaltene Matlab-Funktion „fminimax“ benutzt.

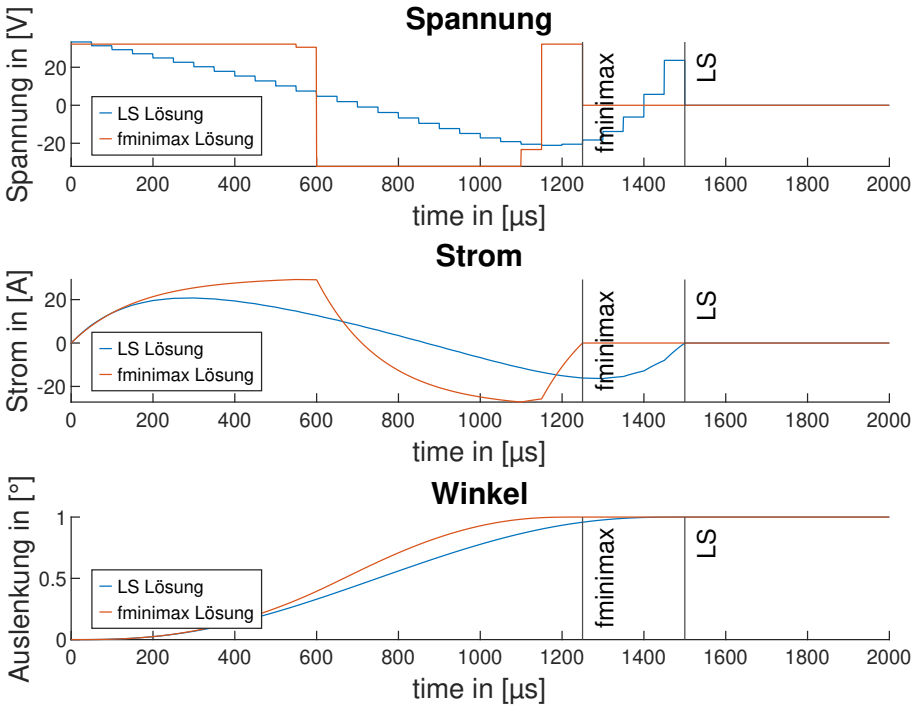

Abbildung 6: Minimax Lösung in kürzerer Zeit bei gleicher Maximalspannung

In Abbildung 6 ist ein weiterer Minimax Sprung im Vergleich zu der Least Squares Lösung zu sehen. In diesem Fall wurde der Sprung durch „fminimax"derart optimiert, dass er bei annähernd gleicher Maximalspannung in 16,66 % weniger Zeit als der Least Squares Sprung durchgeführt werden kann. Die vertikalen, schwarzen Striche markieren hierbei das Vollenden des jeweiligen Sprungs.

4. Grundlagen statischer, nichtlinearer Optimierung ohne Nebenbedingungen

In diesem Kapitel werden die für diese Arbeit relevanten Grundlagen der statischen, nichtlinearen Optimierung ohne Nebenbedingungen erörtert. Zunächst wird auf die Grundstruktur von Problemstellungen eingegangen, welche sich unter Zuhilfenahme von Optimierungsverfahren lösen lassen. Darauf folgend werden die für ein Optimum notwendigen und hinreichenden Bedingungen, die für einen Iterationsschritt hinreichenden Abstiegsbedingungen und die für den Algorithmus relevanten Abbruchbedingungen beleuchtet. Außerdem wird ein kurzer Überblick über die algorithmische Grundstruktur der in dieser Arbeit verwendeten numerischen, nichtlinearen Optimierungsverfahren gegeben.

4.1. Struktur von statischen Optimierungsproblemen

Aus mathematischer Sicht ist die Optimierung eine Minimierung oder Maximierung einer Funktion, deren Variablen gewissen Einschränkungen unterliegen [9]. Allgemein lässt sich die Problemstellung dabei folgendermaßen formulieren:

$$\min_{\vec{x}\in\mathbb{R}^n} f(\vec{x}), \text{ mit: } f(\vec{x}) : \mathbb{R}^n \to \mathbb{R} \tag{20}$$

unter Berücksichtigung der Gleichungsnebenbedingungen

$$c(\vec{x}) = 0, \, c \in \mathbb{R}^m \tag{21}$$

und der Ungleichungsnebenbedingungen

$$h(\vec{x}) \leq 0, \, h \in \mathbb{R}^q \tag{22}$$

Bei $f(\vec{x})$ handelt es sich um die sog. Kostenfunktion (bei Maximierung auch Gütefunktion genannt). Im Allgemeinen muss die Wahl des Punktes $\vec{x}$ die Gleichungs- und Ungleichungsnebenbedingungen erfüllen.
Das in dieser Arbeit behandelte Optimierungsproblem besitzt jedoch keine Nebenbedingungen, weshalb es sich allein durch (20) beschreiben lässt. Diese Tatsache verringert die Komplexität der einsetzbaren Optimierungsverfahren wesentlich. Im Vergleich zu den entsprechenden Verfahren für die nichtlineare Optimierung mit Nebenbedingungen sind diese Verfahren einfacher zu verstehen,

zu implementieren und anzuwenden.

4.2. Optimalitätsbedingungen

Bei der Suche nach der Lösung des vorliegenden Optimierungsproblems ist eine klare Definition für ein Optimum notwendig. Im Folgenden wird der Begriff Minimum statt Optimum verwendet, da von einem Minimierungsproblem ausgegangen wird.
Dabei ist zwischen lokalen, strikten lokalen und globalen Minima zu unterscheiden: Ein lokales Minimum an einem Punkt $\vec{x}^*$ erfüllt die Bedingung, dass in der hinreichend kleinen Nachbarschaft N von $\vec{x}^*$ gilt:

$$f(\vec{x}^*) \leq f(\vec{x}) \text{ für alle } \vec{x} \in N \tag{23}$$

Ein striktes lokales Minimum an einem Punkt $\vec{x}^*$ erfüllt die Bedingung, dass in der hinreichend kleinen Nachbarschaft N von $\vec{x}^*$ gilt:

$$f(\vec{x}^*) < f(\vec{x}) \text{ für alle } \vec{x} \in N \tag{24}$$

Ein globales Minimum an einem Punkt $\vec{x*}$ erfüllt die Bedingung, dass gilt:

$$f(\vec{x}^*) \leq f(\vec{x}) \text{ für alle } \vec{x} \in \mathbb{R}^n \tag{25}$$

Da man bei der Suche jedoch nur über lokale Informationen über $f(\vec{x})$ verfügt, ist es schwierig, ein globales Minimum mit Sicherheit zu bestimmen. Eine Ausnahme hierbei liegt vor, falls es sich bei $f(\vec{x})$ um eine konvexe Funktion handelt. Bei einer konvexen Funktion ist jedes lokale Minimum gleichzeitig auch ein globales Minimum [9].
Um ein lokales Minimum als solches zu identifizieren und die weitere Suche damit abbrechen zu können, ohne alle Punkte in dessen hinreichend kleinen näheren Umgebung zu untersuchen, ist es nötig, es auf die hinreichenden Optimalitätskriterien hin zu prüfen [9].
Ist die Funktion $f(\vec{x})$ an der Stelle $\vec{x}^*$ zweimal stetig differenzierbar, so kann man sie an dieser Stelle zu einer Taylorreihe entwickeln, woraus sich die notwendigen und die hinreichenden Optimalitätsbedingungen ergeben:

$$f(\vec{x}^* + \delta\vec{x}) = f(\vec{x}^*) + \nabla f(\vec{x}^*)^T \delta\vec{x} + \frac{1}{2}\delta\vec{x}^T \nabla^2 f(\vec{x}^*)\delta\vec{x} + O(||\delta\vec{x}||^3) \tag{26}$$

4.2.1. Notwendige Optimalitätsbedingung erster Ordnung

An einem lokalen Minimum $\vec{x}^*$ gilt für alle hinreichend kleinen $\delta\vec{x}$ eingesetzt in (23):

$$f(\vec{x}^* + \delta\vec{x}) \geq f(\vec{x}^*) \tag{27}$$

$$f(\vec{x}^* + \delta\vec{x}) - f(\vec{x}^*) \geq 0 \tag{28}$$

Stellt man die Taylorreihe (26) um

$$f(\vec{x}^* + \delta\vec{x}) - f(\vec{x}^*) = \nabla f(\vec{x}^*)^T \delta\vec{x} + \frac{1}{2}\delta\vec{x}^T \nabla^2 f(\vec{x}^*)\delta\vec{x} + O(||\delta\vec{x}||^3) \tag{29}$$

und setzt (28) darin ein, ergibt sich folgende Ungleichung:

$$\nabla f(\vec{x}^*)^T \delta\vec{x} + \frac{1}{2}\delta\vec{x}^T \nabla^2 f(\vec{x}^*)\delta\vec{x} + O(||\delta\vec{x}||^3) \geq 0 \tag{30}$$

Da in einer Taylorreihe für sehr kleine $\delta\vec{x}$ der Term der ersten Ableitung $\nabla f(\vec{x}^*)^T \delta\vec{x}$ überwiegt, gilt:

$$\nabla f(\vec{x}^*)^T \delta\vec{x} \geq 0 \tag{31}$$

Dies gilt an dem lokalen Minimum für alle $\delta\vec{x}$, woraus die auch oft als Stationärbedingung bezeichnete notwendige Bedingung erster Ordung folgt:

$$\nabla f(\vec{x}^*) = 0 \tag{32}$$

Im eindimensionalen Fall würde das einer Steigung von 0 am Punkt x^* entsprechen.

4.2.2. Notwendige Optimalitätsbedingung zweiter Ordnung

Die notwendige Bedingung erster Ordnung ist auch für Maxima und Sattelpunkte erfüllt. An einem lokalen Minimum hingegen gilt jedoch auch die notwendige Bedingung zweiter Ordnung.
Ist (32) erfüllt, muss gelten:

$$\frac{1}{2}\delta\vec{x}^T \nabla^2 f(\vec{x}^* + \delta\vec{x})\delta\vec{x} + O(||\delta\vec{x}||^3) \geq 0 \tag{33}$$

In diesem Fall dominiert der Term der zweiten Ableitung $\frac{1}{2}\delta\vec{x}^T\nabla^2 f(\vec{x}^*)\delta\vec{x}$ gegenüber dem Restglied $O(||\delta\vec{x}||^3)$, woraus folgt:

$$\frac{1}{2}\delta\vec{x}^T\nabla^2 f(\vec{x}^*)\delta\vec{x} \geq 0 \tag{34}$$

Um dies für jedes $\delta\vec{x}$ zu garantieren, muss die Hessematrix $\nabla^2 f(\vec{x}^*)$ positiv semidefinit sein. Es muss also die notwendige Bedingung zweiter Ordnung gelten:

$$\nabla^2 f(\vec{x}^*) \geq 0 \tag{35}$$

Im eindimensionalen Fall würde dies einer Krümmung ≥ 0 entsprechen.

4.2.3. Hinreichende Optimalitätsbedinungen

Sind die notwendigen Optimalitätsbedingungen erster und zweiter Ordnung an einem Punkt erfüllt, impliziert dies nicht, dass es sich bei diesem Punkt um ein striktes lokales Minimum handelt. Um dies garantieren zu können, müssen die hinreichenden Optimalitätsbedingungen erfüllt sein. Diese ergeben sich analog zu 4.2.1 und 4.2.2.
Bei einem Punkt $\vec{x}^*$ handelt es sich mit Sicherheit um ein lokales Minimum der Funktion $f(\vec{x}^*)$, falls gilt:

$$\nabla f(\vec{x}^*) = 0 \tag{36}$$

$$\nabla^2 f(\vec{x}^*) > 0 \tag{37}$$

4.3. Hinreichende Abstiegsbedingung

Ein Schritt in Richtung $\vec{d_i}$ der Schrittweite $\beta\vec{d_i}$ vom Punkt $\vec{x}_i$ führt zu einem Abstieg, wenn gilt:

$$f(\vec{x} + \beta\vec{d}) < f(\vec{x}) \tag{38}$$

Um dies gewährleisten zu können, muss die hinreichende Abstiegsbedingung erfüllt sein [8]. Diese sagt aus, dass das Skalarprodukt aus der Suchrichtung $\vec{d}$ und dem Gradienten der Kostenfunktion $\nabla f(\vec{x}_i)$ negativ sein muss.

$$\vec{d_i}\nabla f(\vec{x}_i) < 0 \tag{39}$$

Anschaulich interpretiert, muss der Winkel zwischen dem Gradienten und der Abstiegsrichtung stumpf sein.

4.4. Abbruchbedingung

Die in dieser Arbeit angewandten Algorithmen brechen ihre Suche ab, falls an einem Punkt $\vec{x}^*$ die folgende Abbruchbedingung erfüllt ist:

$$||\nabla f(\vec{x}^*)|| < \epsilon \tag{40}$$

Diese leitet sich ab aus der hinreichenden Optimalitätsbedingung erster Ordnung (36). Da es aufgrund der endlichen Rechengenauigkeiten des jeweiligen Prozessors sehr unwahrscheinlich ist, dass der Gradient der Kostenfunktion genau 0 ist, wird geprüft, ob dessen Betrag die Toleranzgrenze ϵ unterschreitet.

4.5. Algorithmische Grundstruktur

Manche Optimierungsprobleme lassen sich mithilfe der Optimalitätsbedingungen analytisch lösen. Da dies in vielen Fällen jedoch nicht möglich ist, müssen numerische Optimierungsverfahren angewendet werden.
Die grundsätzliche Vorgehensweise der in dieser Arbeit verwendeten Algorithmen lässt sich in folgende Schritte zusammenfassen:

1. Wahl eines Startpunktes $\vec{x}_0$, $i = 0$
2. Bestimmen der Suchrichtung $\vec{d}$ durch eine Subroutine
3. Bestimmen der Schrittweite β durch ein Liniensuchverfahren
4. Setzen von $\vec{x}_{i+1} = \vec{x}_i + \beta\vec{d}$ und $i = i + 1$
5. Prüfen auf Abbruchsbedingungen
    a) Falls erfüllt: Beenden der Suche
    b) Falls nicht erfüllt: Weitere Iteration ab Schritt 2

Durch ihre iterative Vorgehensweise nähern sich die Optimierungsalgorithmen schrittweise einem Optimum an. Dabei wird von jedem Schritt gefordert:

$$f(\vec{x}_{i+1}) < f(\vec{x}_i) \tag{41}$$

Die in Kapitel 6 behandelten Algorithmen unterscheiden sich im Wesentlichen nur in der Bestimmung der Suchrichtung und der Art des jeweils eingesetzten Liniensuchverfahrens zur Festlegung der Schrittweite.

5. Wahl der Kostenfunktion

Um bei der Berechnung des optimalen Eingangssignals auf dem eingesetzten Microcontroller zu gleichwertigen oder sogar besseren Ergebnissen zu kommen, wie durch die Anwendung der Matlab Funktion „fminimax“, gilt es, eine möglichst schnell berechenbare Kostenfunktion für das bestehende Problem zu formulieren. Im folgenden Kapitel werden verschiedenen Aspekte der Kostenfunktion behandelt. Dabei wird zunächst auf die Approximation der max-Funktion und der Betragsfunktion für die Wahl der Kostenfunktion eingegangen. Anschließend wird das zugrundeliegende Prinzip zur Anpassung der Präzision der sog. „exponential penalty function“ innerhalb der Kostenfunktion erläutert.

5.1. Approximation der max-Funktion

Viele Optimierungsalgorithmen nutzen in unterschiedlicher Weise die erste oder auch die zweite Ableitung des Kostenfunktionals in Form von Gradienten, partiellen Ableitungen und der Hessematrix. Um diese an jedem Punkt berechnen zu können, muss das Kostenfunktional zweifach stetig differenzierbar sein. Die partiellen Ableitungen des zu minimierenden Funktionals $\max_{k\in[1,2,3,\ldots,N]} f_k(x)$ sind jedoch unstetig an allen Punkten, an denen zwei oder mehr Funktionen $f_k(x)$ den gleichen Funktionswert wie $\max_{k\in[1,2,3,\ldots,N]} f_k(x)$ haben.
Wie in Abbildung 7 beispielhaft anhand von Sinus und Cosinus gezeigt befinden

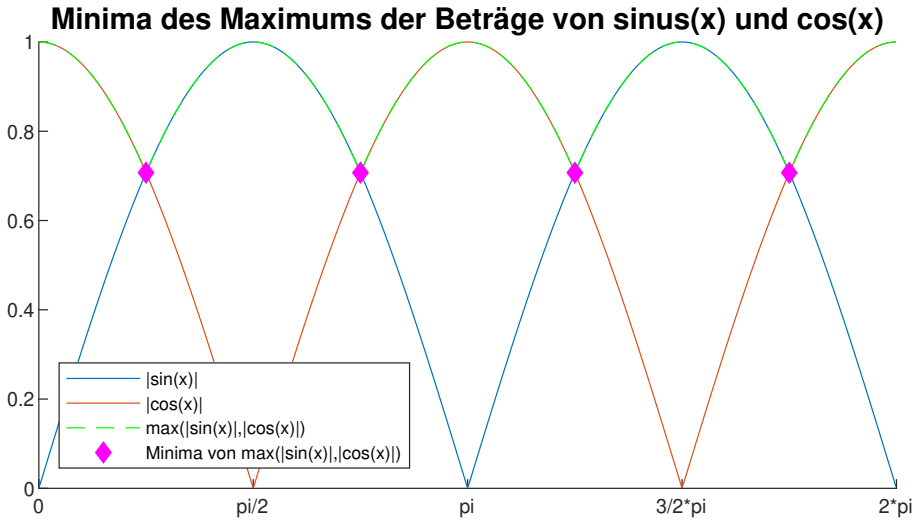

Abbildung 7: Minima des Maximums der Beträge von sinus(x) und cos(x)

sich die Minima des Maximums der Beträge mehrerer Funktionen stets an diesen Unstetigkeitsstellen.

5.1.1. Exponential Penalty Function

Um eine Berechnung der ersten und zweiten Ableitung des Funktionals an jedem Punkt zu ermöglichen, muss dieses durch eine zweifach stetig differenzierbare Funktion approximiert werden. Eine gängige Methode hierfür ist die Verwendung der sog. „exponential penalty function“ [12][6][10].

$$f_{appr}(x,p) = 1/p \ln \sum_{k=1}^{N} \exp(p f_k(x)) \tag{42}$$
$$f_{appr}(x,p) \geq \max_{k\in[1,2,3,\ldots,N]} f_k(x)$$

Der Präzisionsparameter $p > 0$ kann dabei in unterschiedlicher Art und Weise angepasst werden, um z.B. bei einem Gradientenverfahren einen steileren Abstieg und damit eine größere Schrittweite sicherzustellen oder auch, um die Größe von Zwischenergebnissen einzuschränken und damit Floating Point Overflows zu vermeiden. Genauer wird darauf in 5.3 eingegangen. Durch die Verwendung der

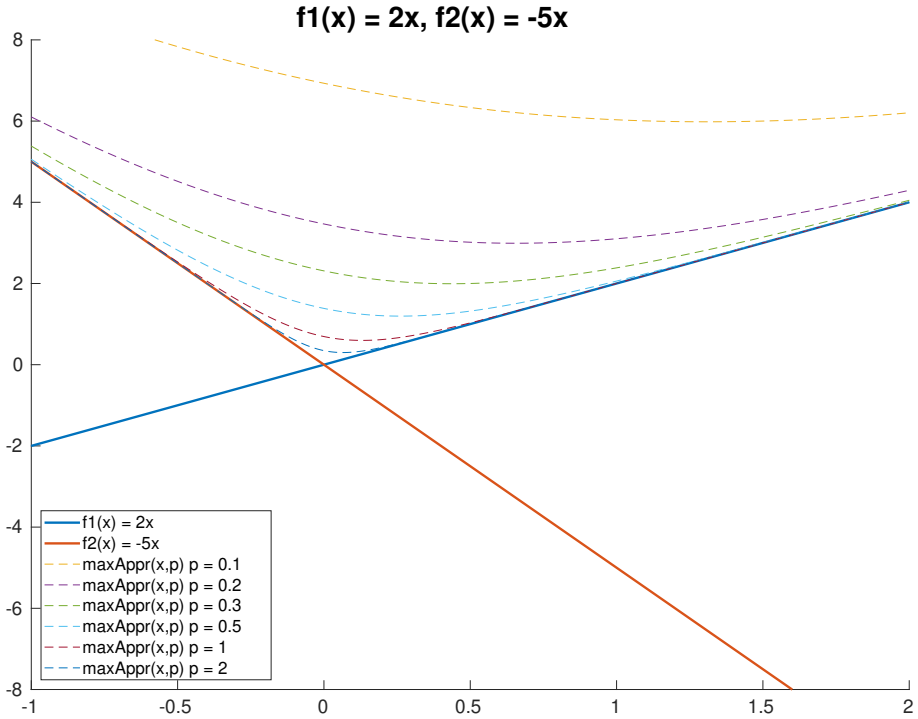

Abbildung 8: Approximation von $max(f1(x), f2(x))$ durch die „exponential penalty function“

Exponentialfuntion überwiegt der größte der Funktionswerte $f_k(x)$ deutlich.
Abbildung 8 zeigt die „exponential penalty function“ angewandt auf zwei lineare Funktionen nahe einer Sprungstelle. Es ist deutlich erkennbar, dass die „exponential penalty function“ näher an der „max Funktion“ liegt, je größer p gewählt wird.

$$\lim_{p\to\infty} f_{appr}(x,p) = \max_{k\in[1,2,3,\ldots,N]} f_k(x) \tag{43}$$

5.1.2. Gradient

Der Gradient der Approximationsfunktion $\nabla f_{appr}(x,p)$ ergibt sich aus der Summe aller Gradienten der evaluierten Funktionen $\nabla f_k(x)$, jeweils gewichtet mit dem Anteil $a_k(x,p)$ des exponentierten Funktionswertes $f_k(x)$ an der Summe aller exponentierten Funktionswerte:

$$\nabla f_{appr}(x,p) = \sum_{k=1}^{N} a_k(x,p)\nabla f_k(x) \tag{44}$$

$$a_k(x,p) = \exp(pf_k(x))/(\sum_{k=1}^{N}\exp(pf_k(x))) \tag{45}$$

Wie in Abbildung 9 zu sehen ist, ist der Anstieg des approximierten Gradienten um die Sprungstelle von $\nabla\max(f1(x), f2(x))$ steiler, je größer p gewählt wird.

5.2. Kostenfunktion

Um das vorliegende nichtlineare Optimierungsproblem möglichst effizient mit verschiedenen Optimierungsalgorithmen lösen zu können, gilt es, eine passende Kostenfunktion zu formulieren. Da diese bei allen vorhandenen Algorithmen in jeder Iteration berechnet werden muss, trägt die Berechnungsdauer der Kostenfunktion einen signifikanten Teil zur gesamten Laufzeit der Algorithmen bei.
Weil manche Algorithmen den Gradienten der Kostenfunktion ausnutzen, muss auch dieser so schnell wie möglich berechnet werden können.
Das für weitere Überlegungen zugrundeliegende Kostenfunktional wurde bereits in 3.3 durch (17) beschrieben, wird jedoch zur einfacheren Lesbarkeit hier noch

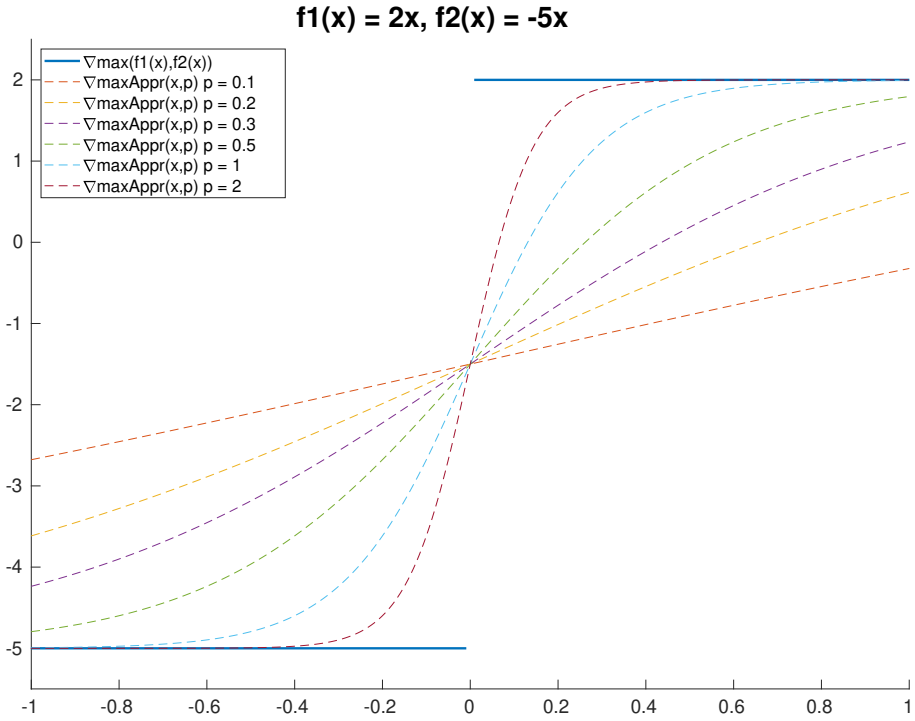

Abbildung 9: Approximation von $\nabla \max(f1(x), f2(x))$ durch die „exponential penalty function“

einmal aufgeführt.

$$
\begin{aligned}
\psi &: \mathbb{R}^N \to \mathbb{R} \\
\psi(\vec{u}_{var}) &= \max_{k \in K} f_k(\vec{u}_{var}) \\
\text{mit } f_k &: \mathbb{R}^{N-m} \to \mathbb{R}, k \in K = [1, 2, 3..., N-1, N] \\
f_k(\vec{u}_{var}) &= |u_{LS,k} + (z_{k,1}, z_{k,2}, ..., z_{k,N-m-1}, z_{k,N-m})\vec{u}_{var}| \qquad (46) \\
\text{und } \vec{u}_{LS} &= \begin{pmatrix} u_{LS,1} \\ \vdots \\ u_{LS,N} \end{pmatrix}
\end{aligned}
$$

Da die Optimierung des Steuervektors $\vec{u}_{opt} = \vec{u}_{LS} + Z\vec{u}_{var}$ als Ziel eine Minimierung der maximalen Amplitude, also des maximalen Betrags von $\vec{u}_{opt}$, hat, bestimmt $f_k(\vec{u}_{var})$ den Betrag von $\vec{u}_{opt,k}$. Die Verwendung der Betragsfunktion führt jedoch dazu, dass $f_k(\vec{u}_{var})$ nicht zweifach stetig differenzierbar ist.
Im Folgenden werden zur Lösung dieses Problems zwei Möglichkeiten vorgestellt und verglichen.

5.2.1. Approximation der Betragsfunktion

Die Betragsfunktion $f(x) = |x|$ kann durch die zweifach stetig differenzierbare Funktion $f_{abs}(x, \alpha) = \sqrt{x^2 + \alpha^2}$ angenähert werden, wobei es sich bei α um eine vernachlässigbar kleine Hilfsvariable handelt. Dabei gilt:

$$\sqrt{x^2 + \alpha^2} \geq |x| \tag{47}$$

$$\lim_{\alpha \to 0} \sqrt{x^2 + \alpha^2} = |x| \tag{48}$$

Die Approximation von Gleichung (46) ergibt sich damit zu:

$$f_{k.abs}(\vec{u}_{var}, \alpha) = \sqrt{(u_{LS,k} + (z_{k,1}, z_{k,2}, ..., z_{k,N-m-1}, z_{k,N-m})\vec{u}_{var})^2 + \alpha^2} \tag{49}$$

$$= \sqrt{(u_{LS,k} + z_{k,1}u_{var,1} + z_{k,2}u_{var,2} + ... + z_{k,N-m-1}u_{var,N-m-1} + z_{k,N-m}u_{var,N-m})^2 + \alpha^2} \tag{50}$$

Durch weiteres Umstellen von (50) lässt sich jede partielle Ableitung $df_{k.abs}/du_{var,i}$ mit $i \in [1, 2, 3, ..., N - m - 1, N - m]$ bestimmen. Zur Veranschaulichung wird im Folgenden die partielle Ableitung nach $u_{var,1}$ hergeleitet:

$$f_{k.abs}(\vec{u}_{var}, \alpha) = \Big\{ (z_{k,1}u_{var,1} + (u_{LS,k} + z_{k,2}u_{var,2} + ... + z_{k,N-m-1}u_{var,N-m-1} + z_{k,N-m}u_{var,N-m}))^2 + \alpha^2 \Big\}^{1/2}$$

$$= \Big\{ z_{k,1}^2 u_{var,1}^2 + 2z_{k,1}u_{var,1}(u_{LS,k} + z_{k,2}u_{var,2} + ... + z_{k,N-m-1}u_{var,N-m-1} + z_{k,N-m}u_{var,N-m}) + (u_{LS,k} + z_{k,2}u_{var,2} + ... + z_{k,N-m-1}u_{var,N-m-1} + z_{k,N-m}u_{var,N-m})^2 + \alpha^2 \Big\}^{1/2}$$

$$\frac{df_{k.abs}(\vec{u}_{var}, \alpha)}{du_{var,1}} = \frac{1}{f_{k.abs}(\vec{u}_{var}, \alpha)} \Big\{ z_{k,1}^2 u_{var,1} + z_{k,1}(u_{LS,k} + z_{k,2}u_{var,2} + ... + z_{k,N-m-1}u_{var,N-m-1} + z_{k,N-m}u_{var,N-m}) \Big\}$$

$$\frac{df_{k.abs}(\vec{u}_{var},\alpha)}{du_{var,1}} = \frac{z_{k,1}}{f_{k.abs}(\vec{u}_{var},\alpha)}\Big\{ z_{k,1}u_{var,1} + u_{LS,k} + z_{k,2}u_{var,2} + \ldots \\ + z_{k,N-m-1}u_{var,N-m-1} + z_{k,N-m}u_{var,N-m}\Big\} \tag{51}$$

$$\frac{df_{k.abs}(\vec{u}_{var},\alpha)}{du_{var,1}} = z_{k,1}\left\{\frac{(u_{LS,k} + (z_{k,1}, z_{k,2}, \ldots, z_{k,N-m-1}, z_{k,N-m})\vec{u}_{var}}{f_{k.abs}(\vec{u}_{var},\alpha)}\right\} \tag{52}$$

Wegen $f_{k.abs}(\vec{u}_{var},\alpha) > 0$ und
$\lim_{\alpha\to 0} f_{k.abs}(\vec{u}_{var},\alpha) = |(u_{LS,k} + (z_{k,1}, z_{k,2}, \ldots, z_{k,N-m-1}, z_{k,N-m})\vec{u}_{var}|$ gilt:

$$\lim_{\alpha\to 0}\frac{df_{k.abs}(\vec{u}_{var},\alpha)}{du_{var,1}} = \begin{cases} z_{k,1} & \text{, wenn } (u_{LS,k} + (z_{k,1}, z_{k,2}, \ldots, z_{k,N-m-1}, z_{k,N-m})\vec{u}_{var} \geq 0 \\ 0 & \text{, wenn } (u_{LS,k} + (z_{k,1}, z_{k,2}, \ldots, z_{k,N-m-1}, z_{k,N-m})\vec{u}_{var} = 0 \\ -z_{k,1} & \text{, wenn } (u_{LS,k} + (z_{k,1}, z_{k,2}, \ldots, z_{k,N-m-1}, z_{k,N-m})\vec{u}_{var} \leq 0 \end{cases} \tag{53}$$

Zur Anschaulichkeit zeigt Abbildung 10 die Approximation von $|f_k(u_{var})|$ mit $f_k(u_{var})$: $\mathbb{R} \to \mathbb{R}$ und deren erste Ableitung $f_{k.abs}(u_{var})/du_{var}$. Das Kostenfunktional berech-

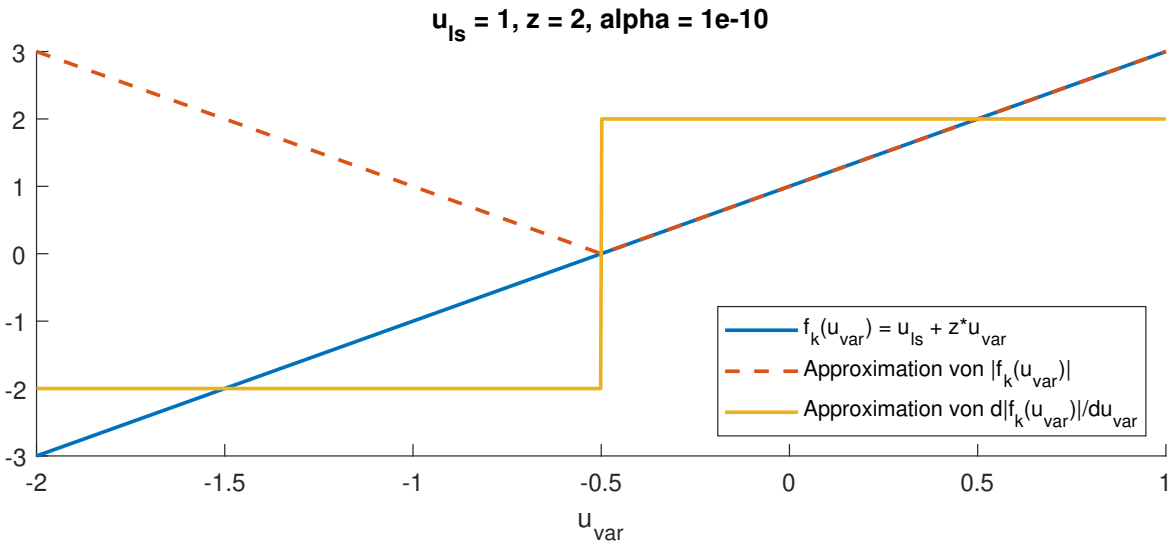

Abbildung 10: Approximation von $|f(x)|$ durch $\sqrt{f(x)^2 + \alpha^2}$

net sich damit sowohl durch Anwendung der „exponential penalty function“ als

auch durch die beschriebene Approximation der Betragsfunktion:

$$f_{abs}(\vec{u}_{var}, p, \alpha) = 1/p \ln \sum_{k=1}^{N} \exp(p f_{k.abs}(\vec{u}_{var}, \alpha)) \tag{54}$$

$$\text{mit } f_{k.abs}(\vec{u}_{var}, \alpha) = \sqrt{(u_{LS,k} + (z_{k,1}, z_{k,2}, ..., z_{k,N-m-1}, z_{k,N-m})\vec{u}_{var})^2 + \alpha^2} \tag{55}$$

Auch der Gradient bildet sich aus beiden Approximationen:

$$\nabla f_{abs}(\vec{u}_{var}, p, \alpha) = \sum_{k=1}^{N} a_k(\vec{u}_{var}, p) \nabla f_{k.abs}(\vec{u}_{var}, \alpha) \tag{56}$$

$$a_k(\vec{u}_{var}, p, \alpha) = \exp(p f_{k.abs}(\vec{u}_{var}, \alpha)) / (\sum_{k=1}^{N} \exp(p f_{k.abs}(\vec{u}_{var}, \alpha))) \tag{57}$$

Die partiellen Ableitungen erster Ordnung $\frac{df_{k.abs}(\vec{u}_{var},\alpha)}{du_{var,i}}$, $i \in [1, 2, 3, ..., N-m-1, N-m]$ werden analog zu (53) berechnet.
Jede Berechnung von $f_{abs}(\vec{u}_{var}, p)$ als auch von $\nabla f_{abs}(\vec{u}_{var}, p)$ führt durch die Ausführung von Logarithmus-, Exponential- und Wurzeloperationen zu einem erheblichen Rechenaufwand. Inwiefern sich dieser auf die gesamte Evaluierung des Funktionals an einem Punkt auswirkt, wird in 5.2.3 betrachtet.

5.2.2. Umformulierung des Kostenfunktionals

Zum gleichen Endergebnis wie durch die Minimierung des in 5.2.1 beschriebenen Kostenfunktionals kommt man auch durch die im Folgenden vorgestellte Vorgehensweise:
Statt den maximalen Betrag der Samplewerte zu minimieren, wird das Maximum der Samplewerte sowie das Maximum der Samplewerte multipliziert mit -1 minimiert. Somit umgeht man den Rechenaufwand durch die Approximation in 5.2.1. Ob dies den hinzukommenden Rechenaufwand durch die doppelte Anzahl an zu evaluierenden Samples rechtfertigt, wird in 5.2.3 untersucht.

$$f_{k.alt}(\vec{u}_{var}) = \begin{cases} u_{LS,k} + (z_{k,1}, z_{k,2}, ..., z_{k,N-m-1}, z_{k,N-m})\vec{u}_{var} \text{ für } 1 \leq k \leq N \\ -1 \cdot \quad (u_{LS,k} + (z_{k,1}, z_{k,2}, ..., z_{k,N-m-1}, z_{k,N-m})\vec{u}_{var}) \text{ für } N+1 \leq k \leq 2N \end{cases} \tag{58}$$

$$\nabla f_{k.alt}(\vec{u}_{var}) = \begin{cases} \begin{pmatrix} z_{k,1} \\ z_{k,2} \\ \vdots \\ z_{k,N-m-1} \\ z_{k,N-m}) \end{pmatrix} & \text{für } 1 \leq k \leq N \\ \begin{pmatrix} -z_{k,1} \\ -z_{k,2} \\ \vdots \\ -z_{k,N-m-1} \\ -z_{k,N-m}) \end{pmatrix} & \text{für } N+1 \leq k \leq 2N \end{cases} \tag{59}$$

5.2.3. Vergleich Rechenzeiten

Die Berechnungszeiten der beiden Kostenfunktionen $f_{abs}(\vec{u}_{var}, \alpha, p)$ und $f_{alt}(\vec{u}_{var}, p)$ sowie deren Gradienten aus 5.2.1 und 5.2.2 werden im Folgenden ermittelt. Die Berechnungen finden dabei ausschließlich auf dem M7-Kern des STM32H755ZI unter Verwendung der 64-bit Fließkommaeinheit und bei der maximalen Taktrate von 480 MHz statt. Die zugrundeliegenden Rechenoperationen sind weder auf den Microcontroller optimiert, noch wurde die Implementierung durch den Compiler optimiert (Optimization Level: -O0).
Um eine klare Aussage treffen zu können, welche der beiden Kostenfunktionen die effizientere ist, wird die Anzahl der Samples N und damit die Größe des Nullraums $Z \in \mathbb{R}^{N\times(N-m)}$ variiert. Des Weiteren werden die zu evaluierenden Punkte $\vec{u}_{var}$ durch die Funktion rand() aus der Standartbibliothek math.h zufällig gewählt. Die Präzisionsparameter $p = 1$ und $\alpha = 1e-10$ werden dabei nicht variiert.

Samples N	$f_{abs}(\vec{u}_{var}, \alpha, p)$	$f_{alt}(\vec{u}_{var}, p)$	$\nabla f_{abs}(\vec{u}_{var}, \alpha, p)$	$\nabla f_{alt}(\vec{u}_{var}, p)$
10	51 μs	74 μs	139 μs	168 μs
20	127 μs	222 μs	374 μs	520 μs
30	208 μs	360 μs	678 μs	960 μs
50	450 μs	881 μs	1562 μs	2433 μs
100	1426 μs	3179 μs	5302 μs	9052 μs
200	6329 μs	16015 μs	21357 μs	39012 μs

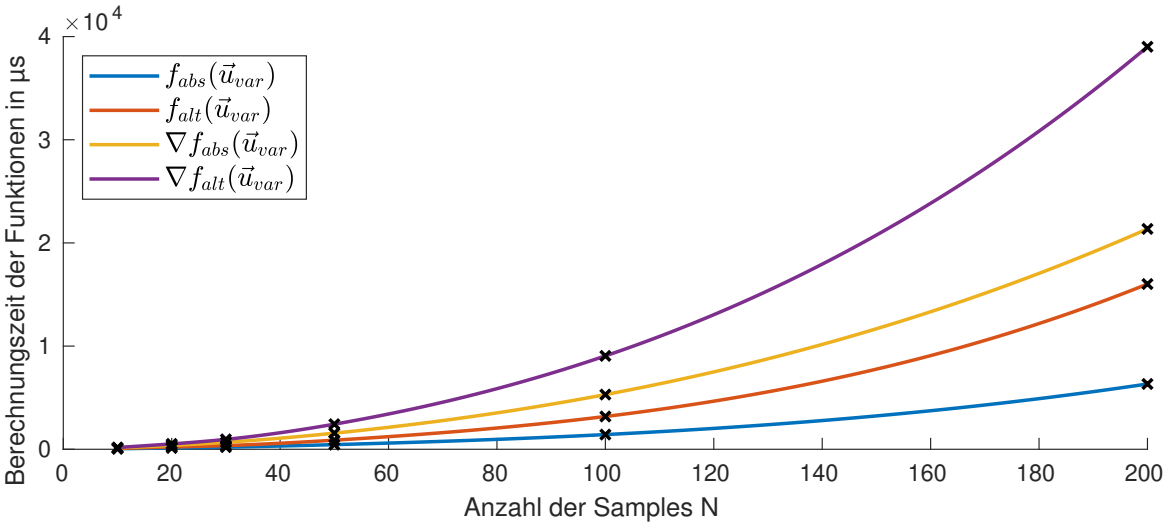

Abbildung 11: Vergleich der Berechnungszeiten der Kostenfunktionen aus 5.2.1 und 5.2.2

Aus den gemessenen Werten ist ersichtlich, dass die Kostenfunktion $f_{abs}(\vec{u}_{var}, \alpha, p)$ aus 5.2.1 sowie deren Gradient deutlich schneller berechnet werden als die Variante der Kostenfunktion aus 5.2.2.
Des Weiteren wird deutlich, dass die Berechnungsdauer exponentiell mit der Anzahl der Samples N steigt. Dies ist auf die exponentielle Vergrößerung des Nullraums $Z \in \mathbb{R}^{N\times(N-m)}$ zurückzuführen. Zum einen müssen mit steigender Anzahl an Samples mehr Teilfunktionen $f_k(\vec{u}_{var}), k \in K = [1, 2, 3..., N-1, N]$ evaluiert werden, zum anderen steigt mit der Anzahl an Samples auch die Anzahl der Elemente des Vektors $\vec{u}_{var} \in \mathbb{R}^{N-m}$.
Im weiteren Verlauf dieser Arbeit wird folglich $f_{abs}(\vec{u}_{var}, \alpha, p)$ als Kostenfunktion und $\nabla f_{abs}(\vec{u}_{var}, \alpha, p)$ in Implementierungen genutzt.

5.3. Präzisionsanpassung

Ein grundlegendes Problem bei der Verwendung der „exponential penalty function“ ist, dass das approximierte Problem deutlich schlechter konditioniert wird. Durch die „exponential penalty function“ kann eine zu große Skalierung der Werte und der Zwischenergebnisse entstehen. Dies kann sowohl zu Überläufen als auch zu Rundungsfehlern führen. Letztendlich wird das Problem für den jeweiligen Algorithmus aufwendiger zu lösen, je genauer die Approximation erfolgt [10]. Polak et al. nutzen in [12] in jeder Iteration innerhalb der behandelten Optimierungsalgorithmen deshalb eine spezielle Subroutine zur Anpassung des Präzisionsparameters p in (42). Aus den numerischen Untersuchungen in [12] geht

hervor, dass diese Vorgehensweise eine bessere Robustheit und meist schnellere Konvergenz der Optimierungsalgorithmen ermöglicht als die Nutzung eines konstanten, bzw. eines sich linear vergrößernden Präzisionsparameters.
Dabei wird der Präzisionsparameter klein gehalten, solange die Suche noch weit von einem Optimum entfernt ist und erst in der Nähe eines stationären Punktes vergrößert. Polaks et al. Präzissionsanpassung wird in Algorithmus 1 durch Pseu-

Algorithmus 1: Präzisionsanpassung in [12]

Ergebnis: Berechnung des Präzisionsparamters p_{i+1} für die nächste Iteration
Parameter: y, $\hat{p}$, i, k
Unterfunktionen: $\epsilon_a(p), \epsilon_b(p), \tau(p) : \mathbb{R}_+ \to \mathbb{R}_+$, wobei gilt: $\epsilon_a(p) \geq \epsilon_b(p) \geq \tau(p)$ für $p > 0$

```
wenn ||∇f_appr(x_i, p_i)||^2 > τ(p_i) dann
    p_{i+1} = p_i ;
    i = i + 1;
sonst
    finde p* sodass gilt: ε_a(p_i) ≤ ||∇f_appr(x_i, p*)||^2 ≤ ε_b(p_i);
    wenn p* ≤ p̂ und y == 1 dann
        // Anfangsstadium
        p_{i+1} = max(p*, p_i + 1);
        i = i + 1;
        k = k + 1;
    sonst
        wenn y == 1 dann
            // Zwischenstadium
            y = max(2, (p̂ + 2)/(k + 1));
            p_{i+1} = y(k + 2);
            i = i + 1;
            k = k + 1;
        sonst
            // Endstadium
            p_{i+1} = y(k + 2);
            i = i + 1;
            k = k + 1;
        Ende
    Ende
Ende
```

docode beschrieben. Sie weist dem Präzisionsparameter p_{i+1} für die nächste Iteration nur dann einen zu p_i unterschiedlichen Wert zu, wenn $f_{appr}(x_i, p_i)$ bereits nahe an einem Minimum liegt und damit das Quadrat der euklidische Norm des Gradienten $||\nabla f_{appr}(x_i, p_i)||^2$ kleiner als $\tau(p_i)$ ist.
Der Präzisionsparameter p^* wird dabei durch ein beliebiges Verfahren derart ge-

wählt, dass das Quadrat der euklidische Norm des Gradienten in dem Bereich $\epsilon_a(p_i) \leq ||\nabla f_{appr}(x_i, p^*)||^2 \leq \epsilon_b(p_i)$ liegt. Abhängig von der Größe von p^* wird p_{i+1} im nächsten Schritt festgelegt:
Je näher x_i an einem Minimum liegt und je kleiner damit $||\nabla f_{appr}(x_i, p_i)||^2$ ist, desto größer wird p^* gewählt. Im Anfangsstadium des Optimierungsalgorithmus, wenn x_i noch relativ weit von einem Minimum entfernt ist, gilt $p^* \leq \hat{p}$.
Polak et al. [12] empfehlen $\hat{p} = \ln(N)/t$ zu setzen, wobei N der Anzahl an Funktionen $f_k(x)$ entspricht und t die geforderte Toleranz zu der Minimax Lösung darstellt. t wird in der Implemtierung auf Empfehlung von Polak et al. auf $t = 100/p^2$ gesetzt. p_{i+1} wird dabei entweder auf $\hat{p}$ oder auf $p_i + 1$ gesetzt und damit in jedem Fall vergrößert.
Für den Fall, dass p^* zum ersten Mal größer ist als $\hat{p}$, wird p_{i+1} unabhängig von p^* festgelegt und in allen folgenden Iterationen nur noch um ein Vielfaches von y erhöht. Dadurch gilt stets $p_{i+1} \geq p_i$ und damit $f_{appr}(x_i, p_{i+1}) - f_{appr}(x_i, p_i) \leq 0$. So wird sichergestellt, dass sich der Wert an der Stelle x_i nicht durch die Wahl von p_{i+1} verschlechtert.

Anpassung wegen FPU Überläufen

Durch die Verwendung der Exponentialfunktion zur Berechnung der „exponential penalty function“ muss die Fließkommaeinheit des Prozessors in Zwischenschritten mit sehr großen Zahlen rechnen. Die größte Zahl entsteht bei der Aufsummierung der exponenzierten Beträge der einzelnen Samplewerte zur Berechnung des Gradienten in (57):

$$\sum_{k=1}^{N} \exp(p f_k(x))) \tag{60}$$

Je größer der Präzisionsparameter p während der Subroutine gewählt wird, desto wahrscheinlicher treten trotz der 64-Bit Fließkommaeinheit Überläufe auf. Der größte in einer IEEE 754 64 Bit Fließkommazahl speicherbare Wert beträgt ca. 1.8×10^{308} [4].
Um einen solchen Überlauf zu erkennen und in den relevanten Rechenoperationen zu vermeiden, wurde die Routine 1 entsprechend erweitert: Schon während der Wahl von p^* durch einen Bisectionsalgorithmus wird darauf geachtet, dass durch eine mögliche Wahl von p_{i+1} zu p^* kein Überlauf entsteht.
Außerdem wird auch nach dem Ausführen der Routine 1 auf einen möglichen Überlauf durch p_{i+1} geprüft und, falls vorhanden, alle durch die Routine berechneten Werte auf ihren vorherigen Zustand zurückgesetzt ($p_{i+1} = p_i$).

6. Auswahl der Optimierungsalgorithmen

In diesem Kapitel wird zunächst eine kurze Begründung zur Auswahl der Algorithmen dargelegt. Anschließend werden die einzelnen Algorithmen vorgestellt.

Obwohl das Verfahren des steilsten Abstiegs nicht als das schnellste der Gradientenverfahren bekannt ist, wurde es für diese Arbeit aufgrund seiner simplen Struktur und der damit einhergehenden leichten Implementierbarkeit als persönlicher Einstieg in die Thematik verwendet. Beschrieben wird es in Abschnitt 6.1. Da bekannt ist, dass es nicht sonderlich performant ist, dient es im weiteren Verlauf der Arbeit vor allem als guter Vergleich zu den beiden anderen Algorithmen. Liu et al. wenden in [6] erfolgreich ein konjugiertes Gradientenverfahren auf Minimax Probleme an, welches sich im Wesentlichen gegenüber dem Verfahren des steilsten Abstiegs nur in der Wahl der Suchrichtung und der Art des Liniensuchverfahrens unterscheidet. Aufgrund dieser Ähnlichkeit und der schnelleren Konvergenz gegenüber dem Verfahren des steilsten Abstiegs wird es in Abschnitt 6.2 behandelt und eine Implementierung aus [14] angepasst auf das Problem angewandt.
In [12], [15] und [10] werden Newton- bzw. Quasi Newton Verfahren zum Lösen von Minimax Problemen eingesetzt. Die Matlab Funktion „fminimax“, mit welcher das Problem in 3.3 gelöst wird, verwendet Sequentielle Quadratische Programmierung (SQP). SQP verhält sich für den Fall, dass keine Nebenbedingungen vorhanden sind, wie ein Newton Verfahren. Da Newton Verfahren jedoch mehr Rechenzeit kosten als Quasi-Newton Verfahren, wird in diesem Kapitel in Abschnitt 6.3 auch eine Implementierung eines Quasi Newton Verfahrens, des BFGS Verfahrens aus [14], verwendet.

6.1. Verfahren des steilsten Abstiegs

Bei dem Verfahren des steilsten Abstiegs handelt es sich um ein iteratives Gradientenverfahren zur Minimierung einer Kostenfunktion $f(\vec{x})$. Es sucht in jeder Iteration nach einem Minimum in der Richtung des steilsten Abstiegs an der momentanen Position.
Ausgehend von einem Punkt $\vec{x}_i$, wird bei dem Liniensuchverfahren in jeder Iteration versucht, einen Punkt $\vec{x}_{i+1}$ zu finden, an welchem gilt:

$$f(\vec{x}_{i+1}) < f(\vec{x}_i) \tag{61}$$

Dafür wird zunächst eine Richtung $\vec{d}$ festgelegt. Hierbei ist $\vec{d}$ stets die Richtung des steilsten Abstiegs an der Stelle $\vec{x}_i$:

$$\vec{d} = -\nabla f(\vec{x}_i) \tag{62}$$

Damit ist auch die hinreichende Abstiegsbedingung (39) erfüllt:

$$-\nabla f(\vec{x}_i) f(\vec{x}_i) < 0 \tag{63}$$

Daraufhin wird in der Schrittweitenbestimmung versucht, ein Minimum in der Richtung $\vec{d}$ zu finden. Es gilt also das eindimensionale Optimierungsproblem

$$\min_{\beta>0} f(\vec{x}_i + \beta\vec{d}) \tag{64}$$

zu lösen. Eine exakte Lösung zu finden ist aufgrund des hohen Rechenaufwands durch häufige Berechnung des Funktionswertes in diesem Fall nicht sinnvoll [9]. Stattdessen wird nach einem Punkt gesucht, welcher nach einer bestimmten Bedingung eine ausreichende Verringerung des Funktionswertes erzeugt.
Die in der Implementierung des Verfahrens des steilsten Abstiegs in dieser Arbeit verwendete Bedingung für eine passende Schrittweite ist die sog. Armijo-Goldstein Bedingung:

$$f(\vec{x}_i + \beta\vec{d}) \leq f(\vec{x}_i) + \beta c m \tag{65}$$

Hierbei gilt:

$$m = \nabla f(\vec{x}_i)^T \vec{d} \tag{66}$$

$$c \in (0, 1) \tag{67}$$

Sie stellt sicher, dass durch den Schritt $s = \beta\vec{d}$ eine ausreichend große Verringerung des Funktionswertes stattfindet.
Dafür wird die Schrittweite β ausgehend von einer relativ großen Schrittweite β_0 solange verkleinert, bis die Armijo-Goldstein Bedingung erfüllt ist.
Anschließend wird die vorangegangene Prozedur ab dem gefundenen Punkt $\vec{x}_{i+1}$ wiederholt, bis entweder die Länge des Schrittes $s = ||\vec{x}_{i+1} - \vec{x}_i||$ unter der minimal geforderten Länge s_{min} liegt oder die Abbruchbedingung (40) erfüllt ist.
Wird dieses Verfahren auf die Kostenfunktion $f_{abs}(\vec{u}_{var}, \alpha, p)$ angewandt, muss nach jeder Iteration zusätzlich noch der Präzisionsparameter p_{i+1} durch Algorithmus 1 aus 5.3 berechnet werden.

Algorithmus 2: Backtracking Liniensuche aus [9]

Ergebnis: Bestimmung von $\vec{x}_{i+1}$
Parameter: $\vec{d}$, $\vec{x}_i$, $\beta_0 > 0$, $\tau \in (0,1)$, $c \in (0,1)$
$\beta = \beta_0$;
$m = \nabla f(\vec{x}_i)^T \vec{d}$;
wiederhole
  $\beta = \tau\beta$;
bis $f(\vec{x}_i + \beta\vec{d}) \leq f(\vec{x}_i) + \beta cm$;
zurück $\vec{x}_{i+1} = \vec{x}_i + \beta\vec{d}$

Algorithmus 3: Gradientenverfahren mit Backtracking Liniensuche

Ergebnis: Minimierung von $f_{abs}(\vec{u}_{var}, \alpha, p)$ aus 5.2.1
Parameter: $s_{min} > 0$, $\vec{u}_{var.0}$, $\beta_0 > 0$, $\tau \in (0,1)$, $c \in (0,1)$, $p_0 > 0$, $\alpha > 0$
$i = 0$;
$p = p_0$;
wiederhole
  `// Berechnung der Richtung` $\vec{d}$
  $\vec{d} = -\nabla f_{abs}(\vec{u}_{var.i}, \alpha, p)$;
  Berechnung des nächsten Punktes $\vec{u}_{var.i+1}$ durch Algorithmus 2 mit $\vec{x}_i = \vec{u}_{var.i}$;
  Berechnung des Präzisionsparamaters p durch Algorithmus 1 ;
  `// Berechnung der Schrittlänge` s
  $s = ||\vec{u}_{var.i+1} - \vec{u}_{var.i}||$;
  $i = i + 1$;
bis $s \leq s_{min}$;
zurück $u_{var.i}$

Bewertung des Algorithmus

Die in diesem Kapitel beschriebene Variante des Verfahrens des steilsten Abstiegs ist verglichen mit den anderen vorgestellten Optimierungsalgorithmen relativ leicht zu implementieren. Des Weiteren ist der Rechenaufwand in jeder Iteration durch die Backtracking Liniensuche mit der Armijo-Goldstein Bedingung überschaubar gegenüber der exakten Liniensuche, die z.B. in 6.2 Anwendung findet. Wie Marcos et al. in [8] beschreiben, ist bei dem Verfahren in vielen Fällen ein relativ schnelles Erreichen eines Bereichs nahe des Optimums zu beobachten.
Eine genauere Betrachtung der Rechenzeiten und Konvergenzgeschwindigkeiten der Algorithmen wird in 7 vorgenommen.
Das simple Verfahren kann im weiteren Verlauf der Arbeit gut als Vergleich zu den anderen Algorithmen dienen.
Das Verfahren ist neben seiner simplen Struktur auch für seine langsame Konvergenz bekannt. Gerade in Tälern einer Kostenfunktion entstehen durch die inexakte Liniensuche in Verbindung mit der Richtungswahl teils sehr geringe Fortschritte. Wie in Abbildung 12 am Beispiel der Rosenbrock Funktion zu sehen ist, führt der Algorithmus in einem Tal sehr kleine Schritte durch, was sich im Endeffekt durch eine langsame Konvergenz ausdrückt.
In Abbildung 13 sind die durch das Verfahren des steilsten Abstiegs berechneten Spannungsverläufe für den gleichen Sprung wie in 3.3 nach verschiedenen Optimierungsiterationen zu sehen. Zum Vergleich ist auch die durch „fminimax"berechnete Lösung abgebildet. Anhand dieser Grafik ist deutlich erkennbar, dass der Algorithmus, wie Marcos et al. in [8] beschreiben, schon innerhalb kurzer Zeit, der ersten 50 Iterationen, relativ nahe an das durch „fminimax"berechnete Optimum kommt und anschließend sehr langsam konvergiert.

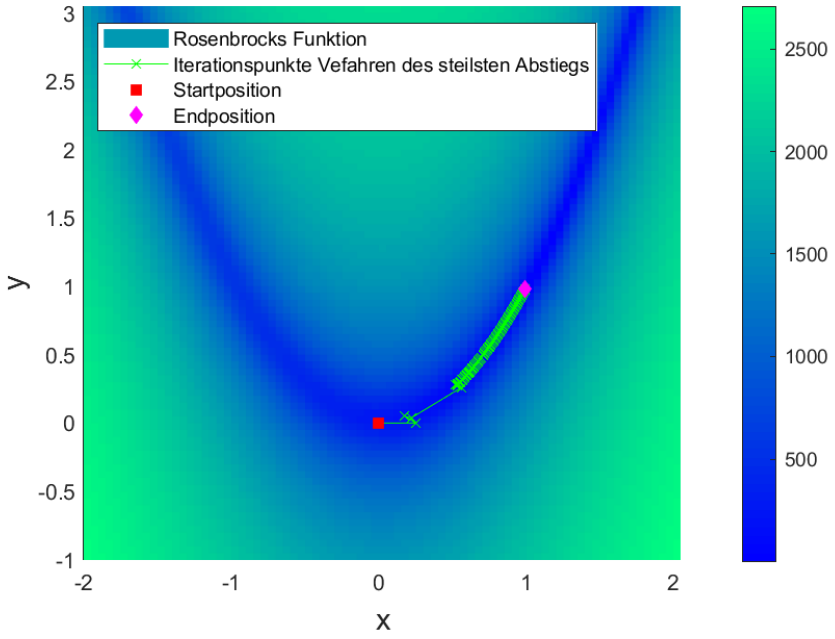

Abbildung 12: Verfahren des steilsten Abstiegs angewandt auf die Rosenbrock Funktion

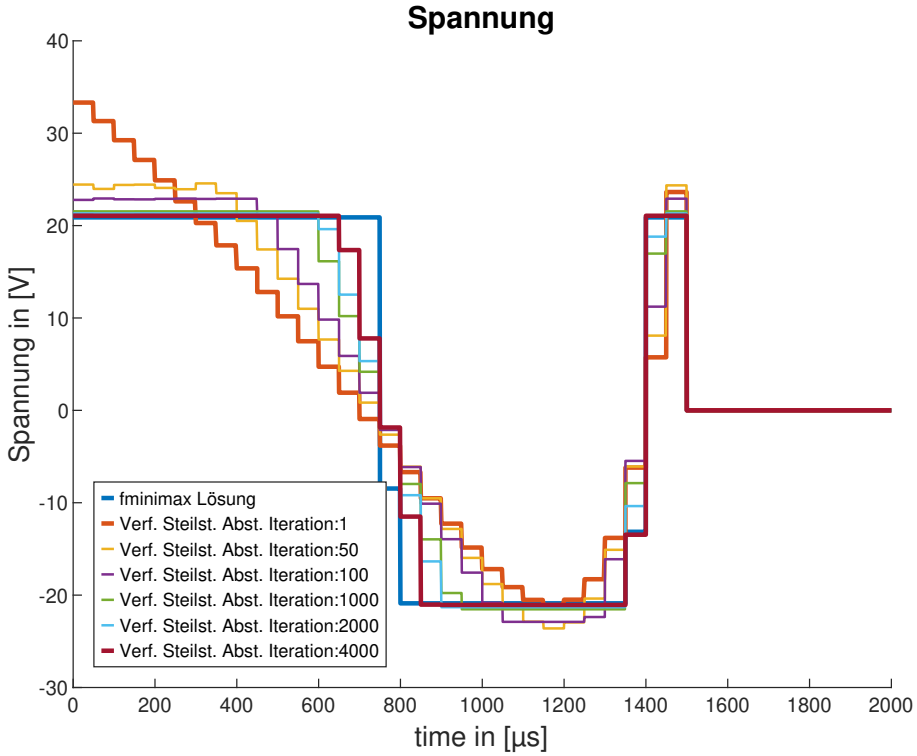

Abbildung 13: Optimierung eines Steuervektors durch das Verfahren des steilsten Abstiegs und fminimax

6.2. Verfahren der konjugierten Gradienten

Ursprünglich wurde das Verfahren der konjugierten Gradienten 1952 von Hestenes und Stiefel in [3] zur Lösung von linearen Gleichungssystemen der Form

$$A\vec{x} = \vec{b} \tag{68}$$

vorgeschlagen. Bei A handelt es sich um eine positiv definite $n \times n$ Matrix. Da das Lösen dieses Gleichungssystems zu dem selben Ergebnis wie das Minimieren der quadratischer Kostenfunktion

$$\phi(\vec{x}) = \frac{1}{2}\vec{x}^T A\vec{x} - \vec{b}^T\vec{x} \tag{69}$$

führt, liegt die Überlegung, es auch zur Optimierung quadratischer Kostenfunktionen zu verwenden, nahe. Zur Optimierung nichtlinearer Kostenfunktionen wurde die Methode 1964 erstmals von Fletcher et al. in [2] verwendet.
In seiner iterativen Vorgehensweise ähnelt es dem Verfahren des steilsten Abstiegs und unterscheidet sich von dem in 6.1 beschriebenen Gradientenverfahren im Wesentlichen nur in der Wahl der Suchrichtungen und der Genauigkeit der Liniensuche.
Die zugrundeliegende Idee zur Minimierung von (69) ist die Wahl von n Suchrichtungen $\vec{d_i} \in (1, n)$, welche zueinander A-konjugiert (A-orthogonal) sind. Für zwei A-konjugierte Vektoren $\vec{d_1}$ und $\vec{d_2}$ gilt:

$$\vec{d_1}^T A\vec{d_2} = 0 \tag{70}$$

Die Minimierung von (69) entlang einer dieser Richtungen hat dadurch keinen Einfluss auf die Minimierung entlang aller anderen A-konjugierten Richtungen. So kann das Minimum $\vec{x}_{opt}$ und damit auch die Lösung von (68) innerhalb von n Iterationen gefunden werden. Als konjugierte Suchrichtungen könnten z.B. die Eigenvektoren von A verwendet werden. Doch die Berechnung von Eigenvektoren großer Matrizen fordert einen erheblichen Rechenaufwand. Eine Alternative dazu ist die Berechnung der Suchrichtung $\vec{d_i}$ aus dem Gradienten von (69) an der aktuellen Position $\vec{x}_i$

$$\nabla\phi(\vec{x}_i) = A\vec{x}_i - \vec{b} \tag{71}$$

und der jeweiligen letzten Suchrichtung $\vec{d}_{i-1}$.

$$\vec{d_i} = -\nabla\phi(\vec{x}_i) + \beta_i\vec{d}_{i-1} \tag{72}$$

Bei β_i handelt es sich um einen Skalar, welcher sich aus den Gradienten an den Punkten $\vec{x}_i$ und $\vec{x}_{i-1}$ bildet:

$$\beta_i = \frac{\nabla\phi(\vec{x}_i)^T \nabla\phi(\vec{x}_i)}{\nabla\phi(\vec{x}_{i-1})^T \nabla\phi(\vec{x}_{i-1})} \tag{73}$$

Formel (73) wurde urspünglich von Fletcher und Reeves in [2] vorgeschlagen. Alle daraus berechneten Richtungen sind nicht nur zu der letzten Richtung, sondern auch zu allen anderen vorhergegangenen Richtungen A-konjugiert. Die in dieser Arbeit eingesetzte Implementierung aus [14] verwendet anstelle von (73) folgende von Polak und Ribiere in [11] entwickelte Formel:

$$\beta_i = \frac{(\nabla\phi(\vec{x}_i) - \nabla\phi(\vec{x}_{i-1}))^T \nabla\phi(\vec{x}_i)}{\nabla\phi(\vec{x}_{i-1})^T \nabla\phi(\vec{x}_{i-1})} \tag{74}$$

Diese erwies sich in numerischen Analysen als eine gute Alternative zu (73) und stellte sich als die robustere und effizientere Variante bei der Verwendung von nicht exakten Liniensuchverfahren heraus [9].
In der ersten Iteration wird statt (72) wie auch beim Verfahren des steilsten Abstiegs das Minimum in Richtung des negativen Gradienten der nichtlinearen Kostenfunktion $f(\vec{x})$ gesucht: $\vec{d_0} = -\nabla f(\vec{x}_0)$
Bei dem Verfahren der konjugierten Gradienten können durch Verwendung von nichtexakten Liniensuchverfahren Fehlsituationen entstehen [8]. Diese äußern sich unter anderem durch eine übermäßige Anzahl an Iterationen bei der Liniensuche oder durch eine ineffiziente Wahl der Suchrichtung. Aus diesem Grund wird hierbei zur Liniensuche kein Backtrack Algorithmus mit Armijo-Goldstein Bedingung, sondern ein exaktes Liniensuchverfahren eingesetzt. Die verwendete Implementierung aus [14] nutzt dazu aus Gründen der Effizienz das Brent Verfahren, welches eine Kombination aus Bisectionsverfahren, linearer Interpolation und quadratischer Interpolation darstellt.
In allen folgenden Iterationen berechnet sich die Suchrichtung $\vec{d_i}$ wie folgt:

$$\vec{d_i} = -\nabla f(\vec{x}_i) + \beta_i \vec{d}_{i-1} \tag{75}$$

$$\beta_i = \frac{(\nabla f(\vec{x}_i) - \nabla f(\vec{x}_{i-1}))^T \nabla f(\vec{x}_i)}{\nabla f(\vec{x}_{i-1})^T \nabla f(\vec{x}_{i-1})} \tag{76}$$

Um das Verfahren auf die Kostenfunktion $f_{abs}(\vec{u}_{var}, \alpha, p)$ aus 5.2.1 anzuwenden, muss nach jeder Iteration zusätzlich noch der Präzisionsparameter p_{i+1} durch Algorithmus 1 aus 5.3 berechnet werden.

Bewertung des Algorithmus

In erster Linie überzeugt das konjugierte Gradientenverfahren durch seine schnellere Konvergenz verglichen mit dem Verfahren des steilsten Abstiegs (vgl. Abschnitt 7). In Abbildung 14 wird diese Eigenschaft deutlich.

Abbildung 15 zeigt das Verfahren der konjugierten Gradienten angewandt auf

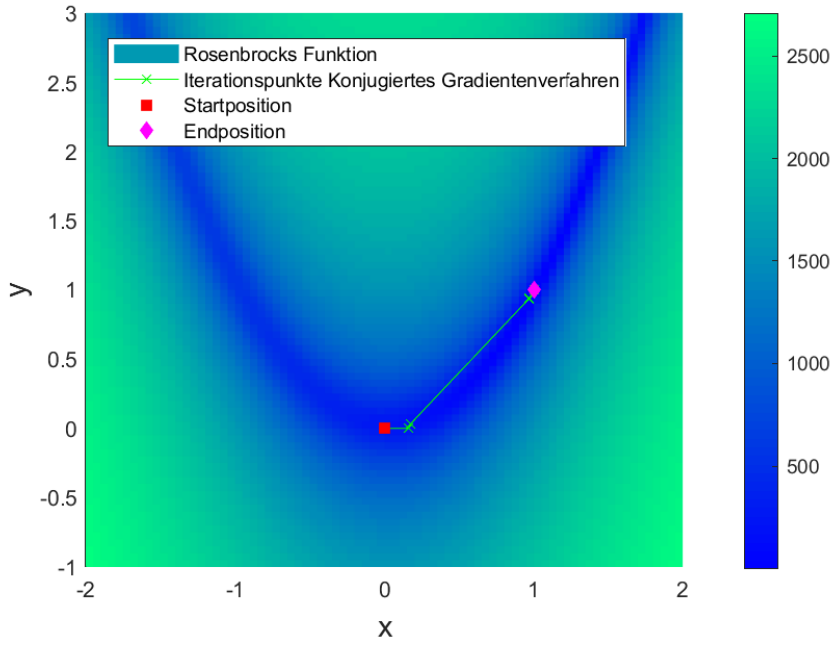

Abbildung 14: Verfahren der konjugierten Gradienten angewandt auf die Rosenbrock Funktion

das selbe Problem wie in Abbildung 13. Wie auch schon beim Verfahren des steilsten Abstiegs ist eine anfänglich sehr schnelle Konvergenz in Richtung des durch „fminimax"berechneten Optimums zu beobachten.
Vergleicht man Abbildungen 14 und 15 mit den entsprechenden Abbildungen 12 und 13 wird die schnellere Konvergenz des Verfahrens der konjugierten Gradienten gegenüber dem Verfahren des steilsten Abstiegs deutlich.

Angewandt auf quadratische Kostenfunktionen ist das konjugierte Gradientenverfahren quadratisch konvergent: Mit jeder Iteration verdoppelt sich die Anzahl der richtigen Dezimalstellen der momentanen Lösung.
Da nach dem Taylor Theorem auch eine nichtlineare, nichtquadratische Kostenfunktion nahe eines Minimums gut durch eine quadratische Näherung beschrieben werden kann und damit gerade dort eine nahezu quadratische Konvergenz vorliegt, ist das Verfahren für den Anwendungsfall dieser Arbeit geeignet [8].
Darüber hinaus verzichtet das Verfahren auf aufwendige Matrizenoperationen

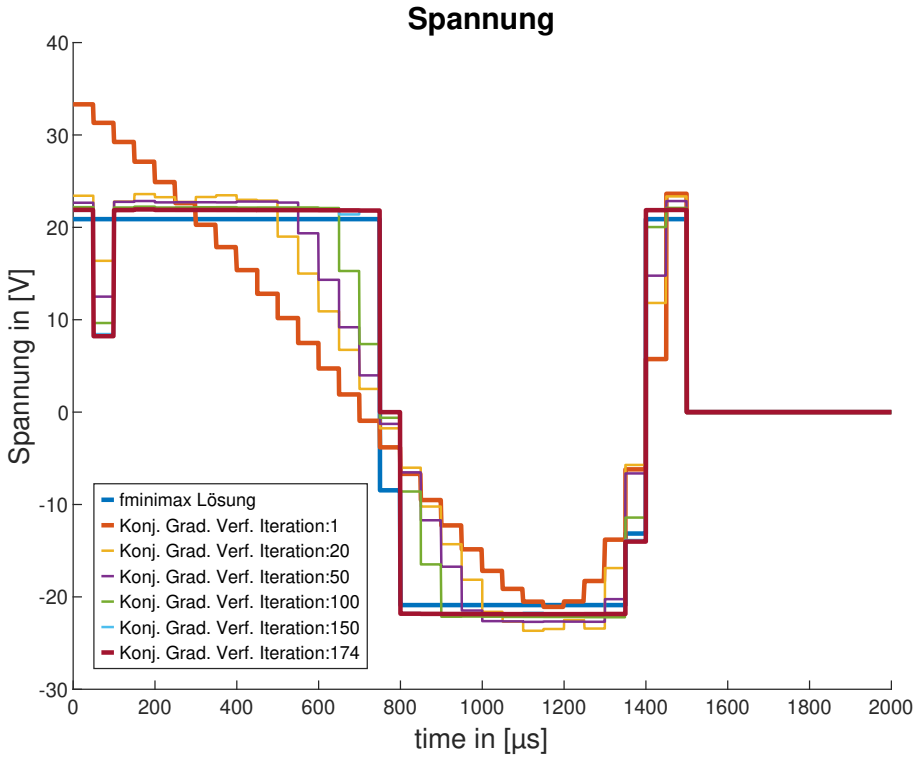

Abbildung 15: Optimierung eines Steuervektors durch das Verfahren der konjugierten Gradienten und fminimax

und die Speicherung großer Matrizen, wodurch es auch auf beschränkter Hardware einsetzbar ist.

Dass der Fortschritt bei der Minimierung entlang der durch (72) und (74) berechneten Suchrichtungen von den vorhergegangenen Suchrichtungen unabhängig ist, gilt nur für Kostenfunktionen quadratischer Form. Nichtlineare Kostenfunktionen können nahe ihres Minimums zwar gut durch konvexe, quadratische Funktionen beschrieben werden, können sich weiter entfernt davon jedoch auch gänzlich nichtquadratisch verhalten. Dies kann zur Folge haben, dass sich die Suchen entlang der berechneten Suchrichtungen gegenseitig beeinflussen und der mit jeder Iteration erzielte Fortschritt abnimmt.

Um dem entgegenzuwirken, empfiehlt sich ein Neustart des Algorithmus nach einer gewissen Anzahl an Iterationen [13]. In der ursprünglichen Implementierung aus [14] wird kein Neustart eingesetzt. Deshalb wird die Implementierung um eine Neustartstrategie ergänzt:

Dabei wird nach N Iterationen $\beta_i = 0$ gesetzt und damit als Suchrichtung der steilste Abstieg am aktuellen Punkt gewählt. Somit wird sichergestellt, dass die in den darauf folgenden Iterationen verwendeten Suchrichtungen unabhängig von den anfänglichen Suchrichtungen sind und die Konvergenzgeschwindigkeit

wieder zunimmt.

6.3. Broyden–Fletcher–Goldfarb–Shanno Verfahren

Bei dem Broyden-Fletcher-Goldfarb-Shanno Algorithmus, kurz BFGS Algorithmus, handelt es sich ebenfalls um ein iteratives Verfahren zur Lösung nichtlinearer Optimierungsprobleme. Als Quasi Newton Verfahren ähnelt es in seiner Funktionsweise dem Newton Verfahren, mit dem Unterschied, dass es keine Berechnungsvorschriften für die Hessematrix $\nabla^2 f(\vec{x}_i)$, der Ableitung zweiten Grades, der Kostenfunktion benötigt. Statt die Hessematrix in jeder Iteration genau zu berechnen und mit ihrer Inversen die Suchrichtung zu bestimmen, nutzen Quasi Newton Verfahren Gradienteninformationen zur sukzessiven Approximation dieser Inversen. Das Newton Verfahren ist allgemein für seine schnelle Konvergenz, gerade in der näheren Umgebung von Minima, bekannt [8].
Im Folgenden wird zunächst in Abschnitt 6.3 die grundsätzliche Funktionsweise des Newton Verfahrens zur nichtlinearen Optimierung erläutert. Anschließend wird auf den Unterschied des BFGS Verfahrens zum Newton Verfahren, die Approximation der Hessematrix, eingegangen.

Allgemeine Funktionsweise Newton Verfahren

In seinem grundlegenden Aufbau unterscheidet sich auch das Newton Verfahren nicht von der in Kapitel 4.5 beschriebenen algorithmischen Grundstruktur.
Mit jeder Iteration nimmt das Verfahren eine Approximation der Kostenfunktion am aktuellen Punkt $\vec{x}_i$ vor und sucht mit einem Liniensuchverfahren deren Minimum. Dies äußert sich in der Wahl der Suchrichtung $\vec{d_i}$ am Punkt $\vec{x}_i$ durch:

$$\vec{d_i} = -\nabla^2 f(\vec{x}_i)^{-1} \nabla f(\vec{x}_i) \tag{77}$$

Falls die Hessematrix $\nabla^2 f(\vec{x}_i)$ positiv definit ist, ist sie invertierbar wodurch (77) berechnet werden kann. Außerdem ist dadurch die Abstiegsbedingung

$$-\nabla f(\vec{x}_i) \nabla^2 f(\vec{x}_i)^{-1} \nabla f(\vec{x}_i) < 0 \tag{78}$$

erfüllt.
Da die positive Definitheit der Hessematrix nicht an jedem Punkt gewährleistet

werden kann, muss in jeder Iteration darauf geprüft und bei Nichtvorhandensein reagiert werden. Die sog. modifizierten Newton Verfahren gehen damit unterschiedlich um: Im einfachsten Fall wird die nicht positiv definite Hessematrix durch eine Einheitsmatrix ersetzt, wodurch sich die Suchrichtung in (77) zur Richtung des steilsten Abstiegs ergibt [8].

Approximation der Hessematrix

Sowohl das Prüfen auf die positive Definitheit als auch die Berechnung der Hessematrix erfordern einen erheblichen Rechenaufwand in jeder Iteration. Um diesen zu umgehen, verwenden Quasi Newton Verfahren verschiedene Methoden zur iterativen Approximation der Inversen der Hessematrix $\nabla^2 f(\vec{x})^{-1}$ durch eine Reihe an Matrizen H_i, welche die Bedingung

$$\lim_{i \to \infty} H_i = \nabla^2 f(\vec{x})^{-1} \tag{79}$$

erfüllen.
In diesem Abschnitt wird die Approximation durch das BFGS Verfahren erörtert:
Um garantieren zu können, dass die Suchrichtung die notwendige Abstiegsbedingung (39) auch weit entfernt von einem Minimum erfüllt, müssen die symmetrischen Matrizen H_i positiv definit sein. Bei der 0-ten Iteration wird zunächst durch die Wahl der Approximationsmatrix H_0 zur Einheitsmatrix I in Richtung des steilsten Abstiegs nach einem Minimum gesucht.

$$\vec{d_0} = -H_0 \nabla f(\vec{x}_0) \tag{80}$$

$$H_0 = I \tag{81}$$

In der verwendeten Implementierung aus [14] wird hierfür eine approximierende Liniensuche verwendet.
Alle folgenden Approximationsmatrizen H_i berechnen sich bei dem BFGS Verfahren nach der Formel:

$$H_{i+1} = H_i + \left(1 + \frac{y_i^T H_i y_i}{\delta_i^T y_i}\right) \frac{\delta_i \delta_i^T}{\delta_i^T y_i} - \frac{\delta_i y_i^T H_i + H_i y_i \delta_i^T}{\delta_i^T y_i} \tag{82}$$

Dabei gilt:

$$\delta_i = \vec{x}_{i+1} - \vec{x}_i \tag{83}$$

$$y_i = \nabla f(\vec{x}_{i+1}) - \nabla f(\vec{x}_i) \tag{84}$$

Damit lässt sich die Suchrichtung $\vec{d}_i$ folgendermaßen berechnen:

$$\vec{d}_{i+1} = -H_{i+1} \nabla f(\vec{x}_{i+1}) \tag{85}$$

Bewertung des Algorithmus

Das BFGS Verfahren bietet eine vergleichbar schnelle Konvergenz wie das Newton Verfahren, wobei es nur Informationen über die Kostenfunktion und deren Gradienten benötigt. Durch die Approximation der Hessematrix wird der erhöhte Rechenaufwand des Newton Verfahrens vermieden und es kann eine positive Definitheit dieser approximierten Matrix gesichert und damit bei exakter Liniensuche ein Abstieg von jedem Punkt außer einem Minimum garantiert werden.
Um die Suchrichtung durch (85) berechnen zu können, ist das Zwischenspei-

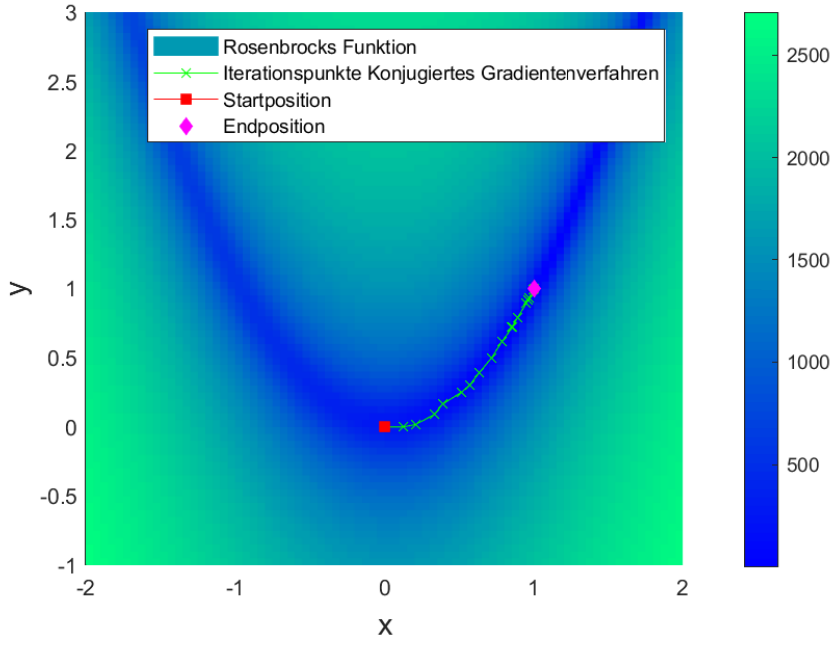

Abbildung 16: BFGS Verfahren angewandt auf die Rosenbrock Funktion

chern der Approximationsmatrix H_i der Größe $n \times n$ notwendig, was gerade auf

beschränkten Systemen wie auf dem in dieser Arbeit verwendeten Microcontroller von Relevanz sein kann. Abbildung 16 veranschaulicht die Anwendung des BFGS Verfahrens auf die Rosenbrock Funktion.
In Abbildung 17 kann man wie auch bei den vorangegangenen Algorithmen die anfänglich sehr schnelle und danach langsamer werdende Konvergenz des BFGS Verfahrens feststellen.

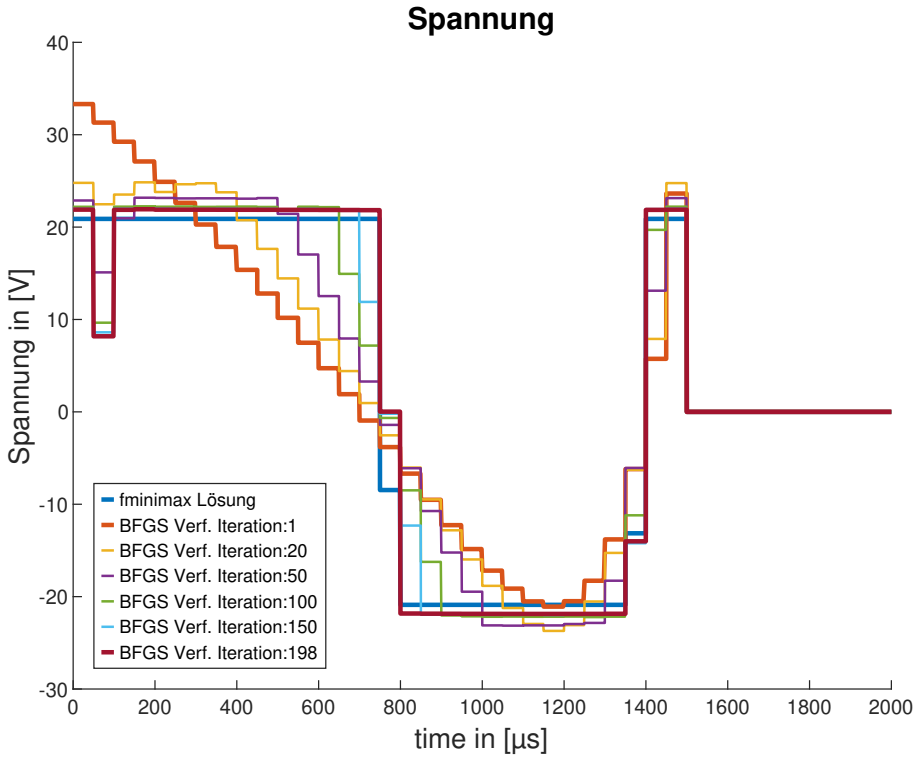

Abbildung 17: Optimierung eines Steuervektors durch das BFGS Verfahren und fminimax

7. Vergleich der Algorithmen

Im Folgenden werden die im vorangegangenen Kapitel erläuterten Algorithmen auf das vorliegende Minimax Problem angewandt und miteinander verglichen.

7.1. Versuchsaufbau und -durchführung

Um die Algorithmen vergleichen zu können, werden Sprünge der Höhe 1° und der verschiedenen Längen 10, 20, 30, 50, 100 und 200 Samples jeweils für das Modell 1 (m=3) und das Modell 2 (m=7) optimiert. Der Startpunkt, also die Least-Squares Lösung und deren Nullraum werden für jeden Sprung im Voraus in Matlab berechnet und fest im Code hinterlegt. Die maximalen Iterationen, die die Algorithmen jeweils durchlaufen dürfen, werden auf 50000 festgelegt. Die Algorithmen verhalten sich abhängig von der Codeoptimierung durch den Compiler unterschiedlich, weshalb die Versuche jeweils ohne (Compilerflag -O0) und mit höchster Optimierungsstufe (Compilerflag -O3) durchgeführt werden.
Zur Messung der Rechenzeiten wird der General-Purpose-Timer TIM2 mit einer Messungenauigkeit $\leq 1\mu s$ eingesetzt.

7.2. Ergebnisse

Abbildungen 18, 19, 20 und 21 zeigen die benötigten Rechenzeiten der Algorithmen und die daraus resultierenden Maximalspannungen.
Die kürzeste Rechenzeit für den jeweiligen Sprung ist grün hinterlegt, blau die niedrigste Maximalspannung. Felder, welche rot eingefärbt sind, kennzeichnen fehlgeschlagene Optimierungsversuche.

	Verfahren des steilsten Abstiegs		Verfahren der Konjugierten Gradienten		BFGS Verfahren	
Sprunglänge in Samples	Rechenzeit in [µs]	Max. Spannung in [V]	Rechenzeit in [µs]	Max. Spannung in [V]	Rechenzeit in [µs]	Max. Spannung in [V]
10	3115770	**379,6101**	**241393**	381,9088	/	/
20	24305167	**56,1698**	**1256993**	56,6393	1774886	56,6393
30	29551735	**21,0174**	1603890	21,9782	**1306648**	21,9782
50	102773169	**6,8594**	2753484	7,3107	**1956846**	7,3026
100	19117192	**1,8985**	8209064	1,9966	**5729780**	1,9706
200	44293534	**0,6906**	**24099559**	0,7031	/	/

Abbildung 18: Test der Algorithmen für verschiedene Sprunglängen (Modell 1, Codeoptimierung -O0)

	Verfahren des steilsten Abstiegs		Verfahren der Konjugierten Gradienten		BFGS Verfahren	
Sprunglänge in Samples	Rechenzeit in [µs]	Max. Spannung in [V]	Rechenzeit in [µs]	Max. Spannung in [V]	Rechenzeit in [µs]	Max. Spannung in [V]
10	979755	**379,6101**	**86684**	381,9088	/	/
20	8204075	**56,1686**	**225481**	58,7770	604312	56,6393
30	7486336	**21,0202**	728934	21,9782	**507964**	21,9782
50	14255086	**6,8664**	**1485387**	7,3081	/	/
100	7749506	**1,8960**	**2230682**	1,9966	/	/
200	115287988	**0,6852**	**16933250**	0,7031	/	/

Abbildung 19: Test der Algorithmen für verschiedene Sprunglängen (Modell 1, Codeoptimierung -O3)

	Verfahren des steilsten Abstiegs		Verfahren der Konjugierten Gradienten		BFGS Verfahren	
Sprunglänge in Samples	Rechenzeit in [µs]	Max. Spannung in [V]	Rechenzeit in [µs]	Max. Spannung in [V]	Rechenzeit in [µs]	Max. Spannung in [V]
10	/	/	/	/	/	/
20	2106547	**7,6934**	833670	7,9974	**251961**	7,9973
30	1739978	1,8738	606182	1,8710	**274978**	**1,8629**
50	3001007	0,3598	2976538	**0,3577**	**646426**	0,3605
100	8074343	**0,0478**	1850628	0,0724	**974708**	0,0644
200	16375133	**0,0086**	**6503549**	0,0118	/	/

Abbildung 20: Test der Algorithmen für verschiedene Sprunglängen (Modell 2, Codeoptimierung -O0)

	Verfahren des steilsten Abstiegs		Verfahren der Konjugierten Gradienten		BFGS Verfahren	
Sprunglänge in Samples	Rechenzeit in [µs]	Max. Spannung in [V]	Rechenzeit in [µs]	Max. Spannung in [V]	Rechenzeit in [µs]	Max. Spannung in [V]
10	/	/	/	/	/	/
20	776156	**7,6934**	141956	8,0517	**102856**	7,9973
30	805001	1,8738	150161	1,8755	**100917**	**1,8629**
50	927118	0,3598	821523	**0,3577**	**212182**	0,3605
100	**333726**	**0,0533**	625335	0,0724	509585	0,0644
200	10176570	**0,0086**	**1647748**	0,0118	/	/

Abbildung 21: Test der Algorithmen für verschiedene Sprunglängen (Modell 2, Codeoptimierung -O3)

7.3. Auswertung und Folgerungen

Um eine klare Aussage darüber treffen zu können welcher der Algorithmen für kommende Projekte zu empfehlen ist, gilt es, die Ergebnisse nach unterschiedlichen Kriterien auszuwerten.

7.3.1. Robustheit

Zunächst spielt die Robustheit eines Algorithmus vor allem im laufenden Betrieb einer Anwendung eine große Rolle. Bei den rot eingefärbten Feldern stieß der jeweilige Algorithmus bei der Optimierung auf bestimmte Probleme, mit welchen er nicht umgehen konnte, und brach deshalb ab. Diese Probleme traten aus verschiedenen Gründen auf:
Das Abbrechen der Algorithmen bei den Tests für die Sprunglängen kleiner 200 ist auf eine schlechte Skalierung der zu optimierenden Variablen und der Kostenfunktion zurückzuführen, wodurch zum einen Rundungsfehler bei zu kleiner Skalierung und zum anderen Überläufe der Fließkommaeinheit bei zu großer Skalierung auftreten können. Press et al. empfehlen in [14] in einem solchen Fall die Variablen und die Kostenfunktion soweit möglich auf die Größenordnung 10^0 zu skalieren.
Außerdem scheiterte der BFGS Algorithmus immer bei Sprüngen der Länge 200. Dies lag daran, dass der Arbeitsspeicher des Microcontrollers nicht ausreichte, um die im BFGS Verfahren benötigten Approximationsmatrizen der Hessematrix zwischenzuspeichern. Abhilfe schaffen kann hierbei eine Erweiterung des Arbeitspeichers sowie die Verwendung einer angepassten Variante des BFGS Verfahrens, des Limited Memory BFGS Verfahrens. Im Hinblick auf die Robustheit ist die verwendete Implementierung des BFGS Verfahrens ohne weitere Modifikationen für die vorliegende Anwendung aus den beschriebenen Gründen nicht zu empfehlen.

7.3.2. Laufzeit

Vergleicht man die ermittelten Laufzeiten der Algorithmen, wird deutlich, dass das Verfahren des steilsten Abstiegs bis auf eine Ausnahme stets am langsamsten war. In den Fällen, in denen weder das Verfahren der konjugierten Gradienten noch das BFGS Verfahren zu einem Abbruch kamen, ist erkennbar, dass die Rechenzeiten des BFGS Verfahrens die der beiden anderen Verfahren in 11 von 14

Fällen deutlich unterschritten. Des Weiteren kann man erkennen, dass keiner der verwendeten Algorithmen für eine in Echtzeit durchzuführende Optimierung des Steuersignals auf dem verwendeten Microcontroller geeignet ist. Hierfür müsste der Sprung in weniger Zeit erstellt und optimiert werden als er in der Ausführung dauert. Dieses Kriterium wird von keinem der Testergebnisse erfüllt, denn schon allein die Optimierung durch die Algorithmen dauerte teilweise um ein Vielfaches länger als der Sprung an sich. Eine echtzeitfähige, ständige Aktualisierung von Filterkoeffizienten, wie in der Einleitung kurz angemerkt, ist mit den verwendeten Verfahren in Kombination mit dem STM32H755ZI deshalb nicht möglich.

7.3.3. Endergebnis

Das jeweils beste Ergebnis erreichte fast immer das Verfahren des steilsten Abstiegs. Bemerkenswert ist, dass die beiden anderen Algorithmen häufig bei einem sehr ähnlichen Wert ihre Suche nach einem Minimum abbrechen, während das Verfahren des steilsten Abstiegs zu einem besseren Minimum konvergiert. Diese Tatsache äußerte sich bereits in vorangegangenen Versuchen mit den Algorithmen. Vergleicht man die Abbildungen 13, 15 und 17 aus Kapitel 6, erkennt man, dass die Maximalwerte der Ergebnisse von „fminimax“ und des Verfahrens des steilsten Abstiegs nahezu identisch sind, wohingegen sowohl das BFGS Verfahren als auch das Verfahren der konjugierten Gradienten an einem höheren Minimum ihre Suche abbrechen.

7.3.4. Empfehlungen und Verbesserungsvorschläge

Aus den gesammelten Ergebnissen kann man keine eindeutige Empfehlung abgeben, welcher der Algorithmen für weiterführende Arbeiten zur Optimierung von Steuervektoren am besten geeignet ist.
Das Verfahren des steilsten Abstiegs stellte sich zwar aller Erwartung nach als das langsamste aller in dieser Arbeit vorgestellten Verfahren heraus, lieferte jedoch in fast allen Fällen das beste Ergebnis. Des Weiteren erwies es sich als äußerst robust. Spielt im Anwendungsfall die Rechenzeit keine Rolle und liegt das Hauptaugenmerk auf dem bestmöglichen Ergebnis, ist dieser Algorithmus den anderen beiden vorzuziehen.
Liegt der Fokus der Anwendung jedoch auf einer möglichst schnellen Konvergenz zu einem Minimum, sollte man die Verwendung eines der anderen beiden Algorithmen in Betracht ziehen.

Das Verfahren der konjugierten Gradienten konnte im Test mehr der zu optimierenden Probleme lösen als das BFGS Verfahren, wobei das letztere meist, manchmal um ein Vielfaches, schneller war. Da für den laufenden Betrieb einer Anwendung ein robuster, langsamerer Algorithmus natürlich einem instabilen vorzuziehen ist, sollte das Verfahren der konjugierten Gradienten gewählt werden, nicht zuletzt da es gerade bei längeren Sprüngen weit weniger Arbeitsspeicher verbraucht als das BFGS Verfahren.
Falls eine Weiterentwicklung der vorgestellten Algorithmen in Frage kommt und der Arbeitsspeicher nicht derart eingeschränkt ist wie auf dem verwendeten Entwicklungsboard, würde es sich aufgrund der besseren Rechenzeiten empfehlen, das Hauptaugenmerk auf das BFGS Verfahren zu legen. Um die Abbrüche aufgrund der schlechten Skalierung der zu optimierenden Variablen und der Kostenfunktion zu vermeiden bzw. abzufangen, wäre es sinnvoll eine Routine zur Neuskalierung des Problems zu entwerfen oder einen anderen Algorithmus ab dem letzten Punkt vor dem Abbruch einzusetzen.
Das Limited Memory BFGS Verfahren wurde in dieser Arbeit nicht behandelt, weshalb keine Aussage über dessen Konvergenzgeschwindigkeit, Robustheit und über die Qualität der Endergebnisse bei dem vorliegenden Optimierungsproblem getroffen werden kann. Grundsätzlich würde sich eine Verwendung dieses Verfahrens jedoch ebenfalls anbieten, um gerade auf eingebetteten Systemen mit wenig Arbeitsspeicher die hohe Konvergenzgeschwindigkeit von Quasi Newton Verfahren auszunutzen.

8. Fazit

Das Ziel dieser Arbeit eine lizenzunabhängige Erstellung von Minimax Sprüngen auf einem Microcontroller zu ermöglichen und dabei die dafür benötigte Rechenzeit zu ermitteln wurde erreicht.
Hierzu wurde zunächst eine Kostenfunktion formuliert und anschließend drei verschiedene Algorithmen erläutert, angepasst und miteinander verglichen. Aus den Testergebnissen ging hervor, dass mit der vorliegenden Implementierung auf dem verwendeten Microcontroller keine echtzeitfähige Optimierung von Steuervektoren möglich ist. Dies liegt einerseits an den zu langen Rechenzeiten der Algorithmen, als auch an wiederholten Stabilitätsproblemen. So entstanden vor allem bei dem BFGS Verfahren Abbrüche durch schlechte Skalierung der Kostenfunktion und derer Funktionswerte. Auch der eingeschränkte Arbeitsspeicher auf dem Microcontroller stellte sich bei längeren Sprüngen für das BFGS Verfahren als ein Problem heraus.
Überraschender Weise konvergierte das simpelste der Verfahren, das Verfahren des steilsten Abstiegs, in den meisten Testfällen zu einem besseren Optimum als die anderen zwei Algorithmen.
Aus diesen gesammelten Erfahrungen wurden deshalb für weitere Arbeiten Empfehlungen abgegeben, welcher der Algorithmen wann eingesetzt werden sollte und inwiefern noch Verbesserungsmöglichkeiten in der vorliegenden Implementierung bestehen.
Ob sich das BFGS Verfahren für den Anwendungsfall stabilisieren lässt und ob es dann im Hinblick auf die Rechendauer tendenziell immer noch schneller ist als das Verfahren der konjugierten Gradienten, muss noch geklärt werden.

A. Modelle

A.1. Modell 1

Kiesbauer erstellt in [5] das Modell dritter Ordnung eines Galvanometer-Spiegel Systems der Firma Cambridge Technologies anhand dessen grundlegenden physikalischen Zusammenhängen. Das Modell setzt sich dabei aus drei Komponenten zusammen:

- Einem RL-Kreis, der den Stromfluss durch die Spule modelliert
- Dem an der Galvanometerachse anliegenden Drehmoment
- Der durch Drehung der Achse induzierten Gegenspannung

Die Übertragungsfunktion des Systems ergibt sich daraus zu

$$H(s) = \frac{\phi(s)}{U(s)} = \frac{1}{s} \frac{\alpha}{(L\Theta)s^2 + (R\Theta + L\rho)s + (R\rho - \alpha\beta)} \tag{86}$$

mit den folgenden Werten:

- Spezifisches Drehmoment $\alpha = 1,31 \cdot 10^{-2} \frac{Nm}{A}$
- Gegenspannungskoeffizient $\beta = 229 \cdot 10^{-6} \frac{Vs}{^\circ}$
- Massenträgheitskoeffizient $\Theta = 0,97 \cdot 10^{-7} kgm^2$
- Induktivität $L = 173 \cdot 10^{-6} H$
- Widerstand $R = 1,07\Omega$
- Gleitreibungskoeffizient $\rho = 6,63 \cdot 10^{-5} \frac{kgm^2}{s}$

A.2. Modell 2

Dieses Modell siebter Ordung wurde aus einer mit Matlab durchgeführten Systemidentifikation gewonnen. Die Übertragungsfunktion der Eingangsspannung U

zur Auslenkung ϕ lautet:

$$H(s) = \frac{\phi(s)}{U(s)} =$$
$$10^{-3} \cdot \frac{-0.1097s^6 - 0.2806s^5 + 0.3239s^4 + 2.223s^3 + 2.663s^2 + 0.4556s + 0.5126}{s^7 + 1.428s^6 + 1.632s^5 + 0.882s^4 + 0.6089s^3 + 0.02643s^2 + 0.000256s + 6.882e - 08} \quad (87)$$

B. Implementierung

Die Implementierung der Algorithmen in C ist auf der beiliegenden CD im Ordner „Implementierung“ als VS Code Projekt zu finden.
Der C Sourcecode befindet sich innerhalb des Unterordners „Code“. Zum Kompilieren und Ausführen muss lediglich GCC vorhanden sein und gegebenenfalls „task.json“ und „launch.json“ angepasst werden.
Da der Fokus dieser Arbeit auf den Laufzeiten der Optimierungsalgorithmen liegt und nicht auf der effizienten Erstellung der Kostenfunktion wird die zur Berechnung der Kostenfunktion benötigte Nullmatrix und die Least Squares Lösung innerhalb des Matlab Skripts „u0_Z.m“ erzeugt. Dies ließe sich im Anwendungsfall natürlich auch in C umsetzen. Das Skript befindet sich im Unterordner „Matlab“. Es berechnet nach Festlegen der Sprunglänge und des Modells die genannten Parameter und aktualisiert anschschließend die Dateien „matrix_declarations.h“ und „matrix_declarations.c“. Diese Dateien beinhalten auch die globalen Variablen für die Berechnung der „exponential penalty function“.
Für den Anwendungsfall ist diese Art der Implementierung aufgrund der globalen Variablen jedoch nicht unbedingt zu empfehlen und müsste gegebenenfalls angepasst werden.
Die Kostenfunktion und deren Gradient werden durch die Funktionen „fAppr()“ und „fGradAppr()“ in „minimax_util.c“ kalkuliert.
Das Verfahren des steilsten Abstiegs ist in „minimaxGD.h“ und „minimaxGD.c“ implementiert. Sowohl das Verfahren der konjugierten Gradienten, umgesetzt in „minimaxCG.h“ und „minimaxCG.c“, als auch das BFGS Verfahren, umgesetzt in „minimaxVm.h“ und „minimaxVM.c“, stammen aus [14], weshalb sie die Routinen aus „nrutil.h“ und „nrutil.c“ benötigen.
Über „nrutil.h“ rufen die Algorithmen auch weitere Routinen, wie „linmin()“ zur Liniensuche, auf, welche sich ebenfalls im Ordner Code befinden.

Abbildungsverzeichnis

Literatur

[1] Christian Reil. Implementierung eines Optimaltrajektoriengenerators für einen Galvanometerscanner auf einem DSP. Master's thesis, OTH Amberg Weiden, 2013.

[2] R. Fletcher. Function minimization by conjugate gradients. *The Computer Journal*, 1964.

[3] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 1952.

[4] IEEE. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, 2019.

[5] B. Kiesbauer. Analyse einer digitalen Zustandsraumregelung für Galvanometerscanner. Master's thesis, OTH Amberg Weiden, 2011.

[6] J K Liu and L Zheng. A smoothing iterative method for the finite minimax problem. *Journal of Computational and Applied Mathematics*, 374, 2020.

[7] Jan Lunze. *Regelungstechnik 2. Mehrgroessensysteme, Digitale Regelung*. Springer Vieweg, Bochum, 8 edition, 2014.

[8] Papageorgiou Markos, Marion Leibold, and Martin Buss. *Optimierung Statische, dynamische, stochastische Verfahren für die Anwendung*. Springer Vieweg, München, 3 edition, 2012.

[9] Jorge Nocedal and Stephen J. Wright. *Numerical optimization 2nd edition*. 2000.

[10] E. Y. Pee and J. O. Royset. On Solving Large-Scale Finite Minimax Problems Using Exponential Smoothing. *Journal of Optimization Theory and Applications*, 148(2), 2011.

[11] E. Polak and G. Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Revue française d'informatique et de recherche opérationnelle. Série rouge*, 1969.

[12] E. Polak, J. O. Royset, and R. S. Womersley. Algorithms with adaptive smoothing for finite minimax problems. *Journal of Optimization Theory and Applications*, 119(3), 2003.

[13] M. J.D. Powell. Restart procedures for the conjugate gradient method. *Mathematical Programming*, 12(1), 1977.

[14] William Press, Saul Teukolski, William Vetterling, and Brian Flannery. *Numerical Recipes in C: The Art of Scientific Computing*, volume 1. Press Syndicate of the University of Cambridge, Cambridge, second edi edition, 1989.

[15] Song Xu. Smoothing method for minimax problems. *Computational Optimization and Applications*, 20(3), 2001.

Erklärung

1. Mir ist bekannt, dass dieses Exemplar der Bachelorarbeit als Prüfungsleistung in das Eigentum der Ostbayerischen Technischen Hochschule Regensburg übergeht.

2. Ich erkläre hiermit, dass ich diese Bachelorarbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum und Unterschrift

Vorgelegt durch:	Simon Schindler
Matrikelnummer:	3114816
Studiengang:	B.Eng. Elektro- und Informationstechnik
Bearbeitungszeitraum:	15. Juni 2020 – 15. September 2020
Betreuung:	Prof. Dr. Hans Meier
Zweitbegutachtung:	Prof. Dr. Norbert Balbierer
Externe Betreuung:	M.Sc. Bernhard Kiesbauer