
Projektseminar Beschleunigertechnik: Ausarbeitung

Iterative Optimierung eines Hammerstein-Modells und Code-Design für ein
Barrier-Bucket-System

Artem Moskalew, Jonas Christ, Maximilian Nolte
Darmstadt, August 2018



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhaltsverzeichnis

1	Abstract	2
2	Einführung	3
2.1	Modell und Konvention	3
2.1.1	Hammerstein Modell	4
2.2	Motivation	5
2.3	Aufgabenstellung	5
3	Vorgehen	6
3.1	Vorhandene Implementierung	6
3.1.1	Python	6
3.1.2	MATLAB	6
3.2	Funktionalität reproduzieren	6
3.3	Funktionalität erweitern	7
3.3.1	Erste Optimierungs Idee	7
4	Gerätekommunikation	8
4.1	Laufzeitoptimierung	8
4.2	Anpassung der horizontalen Auflösung am Oszilloskop	9
5	Code-Design	10
5.1	Motivation	10
5.2	Aufbau	10
5.3	Methodik	11
5.3.1	Test Driven Development	11
5.3.2	Mock-System	12
6	Iterative Optimierung des Hammerstein-Modells	14
6.1	Optimierung der linearen Übertragungsfunktion H	14
6.1.1	Fehlerquellen	15
6.1.2	Umgang mit Fehlerquellen	16
6.1.3	Auswertung	17
6.2	Optimierung der nichtlinearen Kennlinie K	20
6.2.1	Erster Ansatz	22
6.2.2	Zweiter Ansatz	25
7	Fazit	26
8	Ausblick	27
8.1	Impressionen zum Code	27
8.2	Ausblick: Optimierung	27
9	Anhang	30

1 Abstract

Für die sinnvolle Nutzung der Barrier-Bucket-Kavität an den Speicherringen der GSI Darmstadt ist ein qualitativ hochwertiger Einzelsinus als Ausgangsspannung am Gap unabdingbar. Modelliert man den Messaufbau durch ein Hammerstein-Modell mit einer nichtlinearen Kennlinie im Zeitbereich und einer linearen Übertragungsfunktion im Frequenzbereich, so verlagert sich die Schwierigkeit auf das hinreichend genaue Bestimmen dieser Komponenten. Diese lassen sich zwar einzeln über automatisierte Messungen erfassen, sind dabei aber nicht unabhängig voneinander. Um die Modellierung zu verbessern liegt eine iterative Anpassung der beiden Bausteine nahe.

In einem ersten Ansatz zur Optimierung werden für Kennlinie und Übertragungsfunktion getrennte Algorithmen genutzt. Es zeigt sich am simulierten idealen Hammerstein-Modell, dass dieser Ansatz die richtige Richtung einschlägt. Für eine merkliche Verbesserung der Modellierung auch am realen System sind jedoch noch eine Reihe von Hürden zu nehmen. Diese sind teilweise auf Diskretisierung und Interpolation zurückzuführen. Jedoch liegen gerade bei der nichtlinearen Kennlinie auch konzeptionelle Probleme bezüglich des Definitionsbereichs vor, die eine genauere Betrachtung nicht nur für die Optimierung erfordern.

Unverzichtbare Grundlage der hier verfolgten Ansätze sowie Sprungbrett für die weitere Arbeit an einer Optimierung ist das Design einer Programmstruktur, die an den Kontext eines Messaufbaus angepasst ist. Als außerordentlich hilfreich erweist sich ein modularer Aufbau von Funktionalitäten. Zusätzlich bietet sich ein Programmier-Konzept an, das auf periodische Tests der Funktionalitäten ausgelegt ist. Ergänzend hierzu vereinfacht sich die Erweiterung der Funktionalität durch die Möglichkeit der Simulation des mathematischen Modells erheblich.

2 Einführung

Mit Barrier Bucket (BB) RF Systemen sollen am neu entstehenden Synchrotron SIS100 oder im Experimentier Speicherring (ESR) am GSI Helmholtzzentrum unterschiedliche longitudinale Manipulationen an Teilchenstrahlen vorgenommen werden. Der dazu notwendige Spannungspuls hat die in Abb. 2.1 dargestellte Form. Wenn die Wiederholfrequenz des Spannungspulses gleich der Umlauffrequenz ist, so führt dies zu einer stationären Potentialbarriere im Phasenraum. Diese Barrieren können dazu verwendet werden, den Strahl longitudinal einzuschließen und ermöglichen somit variable Bruchlängen. Unterscheiden sich Wiederholfrequenz des Spannungssignals und Umlauffrequenz, so gewegt sich die Barriere im Phasenraum ("moving barrier") und ermöglicht Kompression/Dekompression der Bunches. Eine Herausforderung dieser Systeme liegt darin, eine hohe Qualität des Pulses am Gap der Kavität zu erzeugen, um unerwünschte Effekte auf den Strahl zu verhindern. Laut Spezifikation für das EST System müssen die Nachschwinger nach dem Einzelsinus beispielsweise kleiner als 2,5% von \hat{U}_{BB} sein.

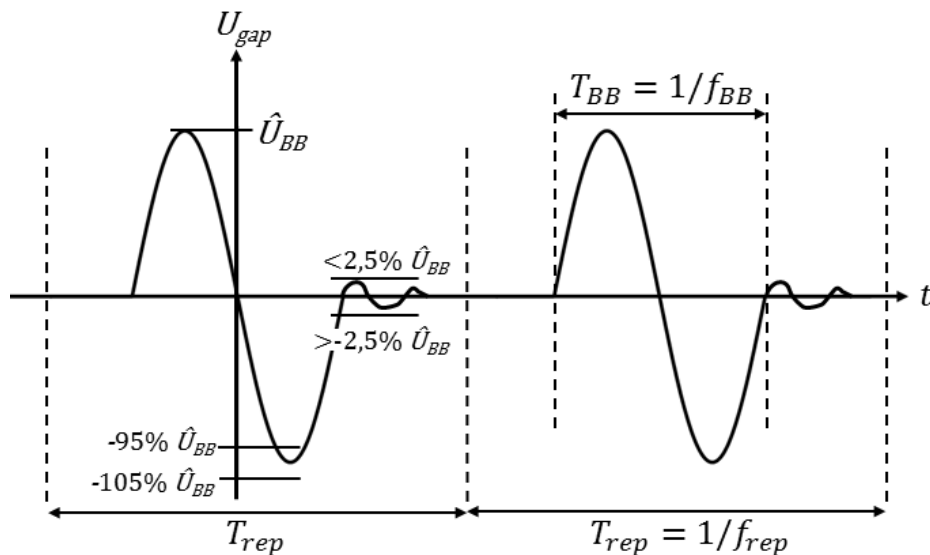


Abbildung 2.1: Ausgangssignal

Dabei wird im Folgenden f_{rep} für die Wiederholfrequenz des Spannungssignals verwendet. Die Barrier Bucket Frequenz f_{BB} bestimmt die Breite der Potential Barriere. Für die Messungen in diesem Projektseminar wurden $f_{rep} = 900\text{kHz}$ und $f_{BB} = 5\text{MHz}$ verwendet.

2.1 Versuchsaufbau und Modell

Der Prototyp des ESR BB besteht aus einem Funktionsgenerator (Keysight 3600A series 2-channel (50 Ω) AWG), einem Verstärker (AR1000A225) und der Breitband Ringkern-Kavität. Der Versuchsaufbau ist in Abb. 2.2 gezeigt. Ein Messablauf ist in Abb. 2.3 als Flow Chart Diagramm dargestellt.

Dabei kann angenommen werden, dass sich das System bis $\hat{U}_{BB} = 550\text{V}$ annähernd linear verhält und durch die Übertragungsfunktion \underline{H} beschrieben werden kann. Eine mathematische Modellierung ist ebenfalls in Abb. 2.2 gegeben, bei der zur linearen Übertragungsfunktion \underline{H} noch eine nichtlineare Kennlinie enthalten ist. Die Größen werden im nächsten Abschnitt erklärt.

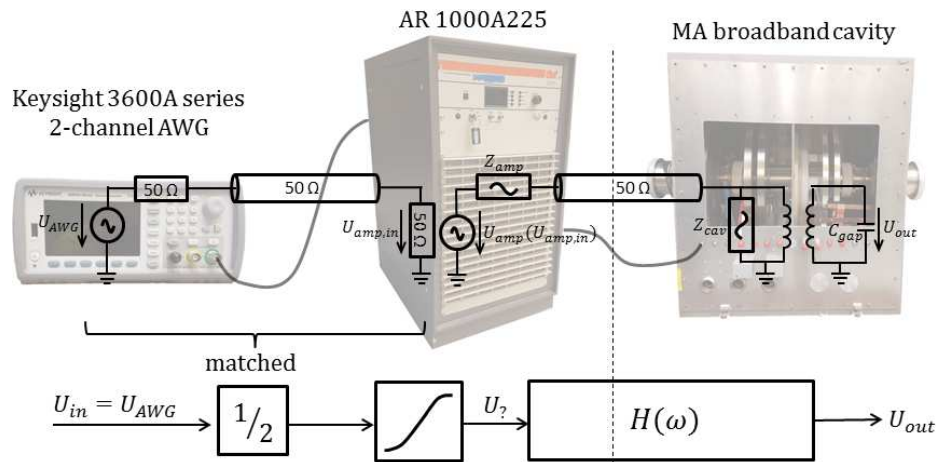


Abbildung 2.2: Versuchsaufbau und Modell siehe [2]

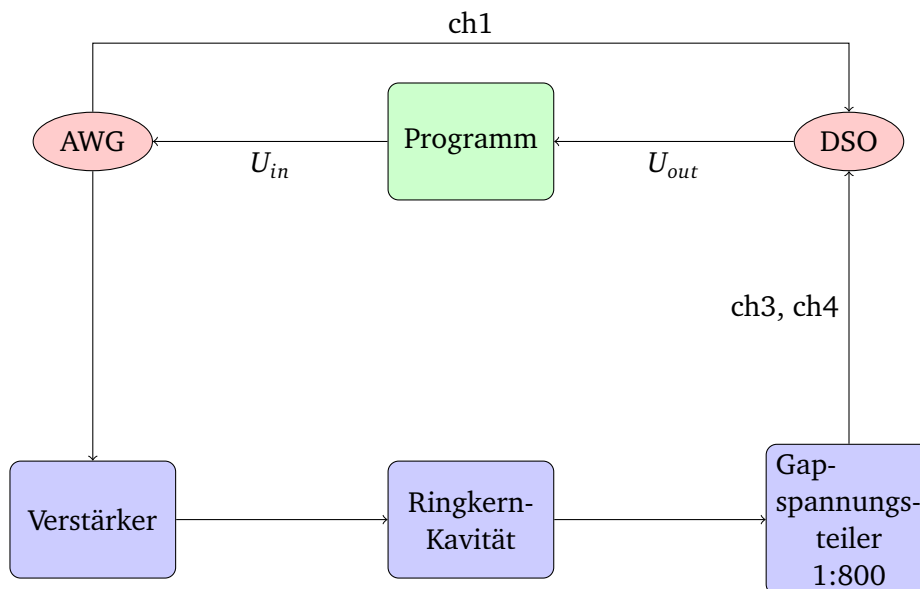


Abbildung 2.3: Ein Messdurchlauf

Das berechnete Eingangssignal U_{in} wird an das AWG übergeben davon wird es zum einen an das Oszilloskop geleitet und zu anderen an den Verstärker. Der Gapspannungsteiler hat ein Verhältnis von 1:800 dabei ergibt sich die Ausgangsspannung U_{out} als die Differenz der Kanäle 3 und 4. Es muss darauf geachtet werden, dass alle Eingang des Oszilloskops auf hoch-ohmig eingestellt sind, um Schäden am Gerät zu verhindern. Dabei müssen die Leitungen von der Kavität mit einem T-Stück und einem 50Ω Widerstand abgeglichen sein. Die Ausgänge des AWG müssen auch auf 50Ω eingestellt werden.

2.1.1 Hammerstein Modell

In Abb. 2.4 sind alle verwendeten Größen dargestellt. U_{in} ist die Eingangsspannung des Funktionsgenerators. Der erste Block in Abb. 2.4 stellt die nichtlineare Modellierung dar. Die Spannung $U_?$ ist eine rein virtuelle Größe und kann wie in Gl. (2.1) aus der Eingangsspannung über die Koeffizienten a_n berechnet werden. Bei der Potenzreihe hatte $N = 3$ in der MATLAB Implementierung schon gute Ergebnisse geliefert und wurde daher weiterhin so verwendet. Die nichtlinearen Kennlinie im Folgenden nur noch als K bezeichnet, wird als Look-Up Tabelle im Programm hinterlegt. Die Spannung U_{out} ist die Gapspannung in der Kavität.

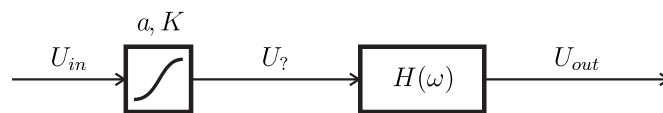


Abbildung 2.4: Hammerstein Modell

$$U_?(t) = \sum_{n=1}^N a_n [U_{in}(t)]^n \quad U_{out}(t) = \mathcal{F}^{-1} \{ \underline{H}(\omega) \cdot \underline{U}_{in}(\omega) \} \quad (2.1)$$

2.2 Motivation

Das Hammerstein Modell wurde bis auf die Berechnung von K schon erfolgreich in Python implementiert. Dabei sei auf das Projektseminar [1] verwiesen. Deshalb lag unsere Motivation darin, dieses Modell in Python zu vervollständigen, um dann einen möglichen Optimierungsansatz aufzustellen.

2.3 Aufgabenstellung

Im Rahmen dieses Projektseminars sollte eine iterative Optimierung der Vorverzerrung auf Basis einer Hammersteinmodellierung in Python implementiert werden. Dabei sollte das Tool die nichtlineare Kennlinie K und die Übertragungsfunktion \underline{H} wechselseitig optimieren, wobei eine optimale Gewichtung der Parameter zur Optimierung noch nicht erreicht werden musste.

3 Vorgehen

In diesem Kapitel werden die wichtigsten Punkte in der Weiterentwicklung des vorhandenen Tools erklärt. Es wird die Methodik erläutert, mit der an den vorhandenen Weg angeschlossen wurde. Dies geschieht auch in Hinblick darauf, dass andere Personen an den hier beschriebenen Weg anschließen werden.

3.1 Vorhandene Implementierung

Das Projekt setzt an 2 Punkten an. Zum einen an dem Projektseminar [1], in dem der Grundstein in Python gelegt wurde und zum anderen an einer funktionierende Implementierung des nichtlinearen Bestands des Modells aus Abb. 2.4 in MATLAB nach [2].

3.1.1 Python

Das vorhandene Python Tool kann mit einem Pseudorauschen die Übertragungsfunktion des Systems im linearen Bereich bestimmen. Diese Funktionalität wurde vollständig übernommen. Darüber hinaus waren folgende Funktionalitäten implementiert:

- \underline{H} auf das Spektrum von $U_{out,ideal}$ anzuwenden.
- Signale an das AWG zu senden und Messsignale aus dem Oszilloskop auszulesen.
- aus gegebenen Koeffizienten a_n eine Kennlinie K als Look-Up Tabelle zu berechnen.

3.1.2 MATLAB

Als zweiten Ansatzpunkt standen funktionierende Methoden aus der MATLAB Implementierung des nichtlinearen Teils mit folgenden Funktionalitäten zur Verfügung:

- Das Eingangssignal des linearen Blocks aus \underline{H} und $U_{out,ideal}$ zu berechnen.
- a_n für die nichtlineare Kennlinie zu berechnen.
- K als Look-Up Tabelle aufzustellen.
- ein verzerrtes Eingangssignal über K zu berechnen.

3.2 Funktionalität reproduzieren

Als erstes wurde versucht, die vorhandene Implementierung zu verstehen und letztendlich beide in Python zusammenzuführen. Dabei wurde eine starke Kopplung innerhalb der vorhandenen Implementierungen festgestellt, was die Fehlersuche und Fortführung erschwerte. Ein paar Funktionen hatten teilweise mehrere logische Aufgaben, die für eine bessere Verständlichkeit getrennt werden sollten. Mit dem Code wurden aber schon gute Ergebnisse erzielt und das erste Ziel war, den Code in eine neue Form zu überführen, ohne die Funktionalität zu beeinträchtigen.

Neues Design

Das in Abb. 2.4 dargestellte Modell wird in zwei Blöcke gegliedert und so wurde ein Code-Design gewählt, das diese Blöcke repräsentiert. Dafür bietet sich ein modulares Design an, das dieses Modell möglichst genau abbildet (Kap. 5). Dieses Design wurde auch in Hinblick auf die Teamarbeit gewählt, damit an mehreren logisch getrennten Abschnitten gleichzeitig gearbeitet werden kann. Dafür wurde eine Github Repository erstellt, um eine Versionenkontrolle vorzunehmen.

Tests

Aus vorherigen Messungen und den Ergebnissen beider Implementierungen konnten Unit-Tests erstellt werden, die die geforderte Funktionalität beschreiben und somit die Ergebnisse aus dem neuen Design mit schon bestehenden Ergebnissen evaluieren.

Ziel war es, den Code umzustrukturieren mit der Bedingung, dass die Funktionalität nicht verändert wird. Die bestehende Implementierung wurde in mehrere logische Abschnitte unterteilt, bei denen jeder Abschnitt für sich die geforderten Daten erzeugen sollte. Dabei sollte die funktionierenden Teile aus MATLAB in Python überführt werden.

Fehlersuche

Die Möglichkeiten der Fehlersuche soll an zwei Beispielen erklärt werden. In der Python Implementierung aus [1] war schon ein erster Entwurf zur Berechnung der Koeffizienten a_n gegeben, der aber noch keine korrekten Werte berechnete. Die Ergebnisse aus der MATLAB Implementierung wurden in den Unit Test importiert, sodass in Python mehrere Möglichkeiten getestet wurden, bis die Ergebnisse mit den schon bestehenden Werten übereinstimmen.

Das zweite Beispiel ist die Berechnung des Eingangssignals über K . Dabei wurden während einer Messung Daten gespeichert, die zu einem Programmabbruch führten. Diese Daten wurden in einem Unit Test zusammengeführt, damit der Fehler im Programm ohne Messaufbau behoben werden konnte. Einzelheiten dazu im Abschnitt 6.2. Des weiteren wird durch das neue Design eine strukturierte Fehlersucher ermöglicht. Wenn eine Veränderung an einem Block vorgenommen wird, so lässt sich diese isoliert betrachten und sollte keinen Einfluss auf die Logik anderer Blöcke haben.

3.3 Funktionalität erweitern

Nachdem die Funktionalitäten beider Implementierungen in Python zusammengeführt wurden konnten im nächsten Schritt Erweiterungen der vorhandenen Funktionalität vorgenommen werden. Dabei stellte das neue Design mit seinem modularen Aufbau eine gute Grundlage dar, um an den logisch getrennten Aufbau anzuschließen und schon vorhandene Funktionalität zu nutzen.

3.3.1 Erste Optimierungs Idee

Als nächster Schritt wurde ein erster Optimierungsansatz implementiert, wie er in der Aufgabenstellung aus Abschnitt 2.3 vorgesehen war. Genauere Erläuterungen folgen in Kap. 6. Der Ansatz wurde mit zwei Routinen getestet und mit dem RF-Tool [18] zur Güteberechnung des Ausgangssignals evaluiert. Bei der Nutzung des Programms ist eine längere Wartezeit für die Geräte aufgefallen, die bei kleineren Durchläufen nicht so ins Gewicht fällt aber im Hinblick auf eine höhere Zahl an Iterationsschritten noch optimiert werden sollte. Dazu werden im folgenden Abschnitt Ansätze vorgestellt.

4 Gerätekommunikation

Um Messergebnisse sinnvoll für die Verarbeitung in Python aufzunehmen, ist eine effiziente Kommunikation zwischen den Messgeräten und dem angeschlossenen Computer essentiell. Besonders relevant ist hier die zeitliche Abstimmung zwischen Computer und Gerät sowie die bestmögliche Nutzung der Genauigkeiten der verwendeten Geräte. Hierbei wurden die zu Beginn vorliegenden Implementierungen für die Gerätekommunikation erweitert, wobei vor allem

1. die Optimierung der Laufzeit beim Schreiben und Lesen von Gerätewerten sowie
2. die Anpassung der Darstellung des Oszilloskops an die gemessenen Daten für eine höhere Genauigkeit

Ziele der Anpassungen waren.

Die Fernsteuerung von Messgeräten, der `Remote`-Modus, wird standardmäßig durch das Visa-Protokoll und im vorliegenden Fall durch die Implementierung von National Instruments, NIVisa, geregelt. Die Einbindung in Python wird durch die Erweiterung PyVisa ermöglicht. Übertragen werden Befehle in Form standardisierter Befehlsstrukturen, den SCPI (Standard Commands for Programmable Instruments). Dem inneren Aufbau von Messgeräten zugrunde liegt der IEEE-488 Standard, sodass unabhängig vom konkreten Gerät oder der Art der Verbindung eine Reihe von Befehlen existiert, die gemäß IEEE-488.2 gebräuchliche Standards bieten. Diese Befehle sind in der Form `*COMMAND` gehalten [5, S. 224 ff.]. Weiterhin bieten die Geräte spezifische Befehle zur Manipulation der Einstellungen, die in großer Analogie zur direkten Benutzeroberfläche formuliert sind. Diese werden nach Kontext gegliedert in sogenannte Subsysteme oder Command Groups.

Es wird ein Keysight 33622A als Signalgenerator (AWG) und ein Oszilloskop der TDS 5000 Serie von Tektronix genutzt.

4.1 Laufzeioptimierung

Im vorliegenden Messaufbau ist es notwendig, beim Konfigurieren des AWGs alle Einstellungen zurückzusetzen und anschließend neu zu setzen. Dabei werden Befehle nach dem First-Come-First-Serve Prinzip aus dem internen Speicher abgearbeitet. Dies führt aufgrund begrenzter Speicherkapazität und Verarbeitungsgeschwindigkeit bei einer Reihe von Befehlen zur Notwendigkeit, das Senden von Befehlen im Programm mit dem Arbeitsstatus des Gerätes zu synchronisieren. Die Größenordnung dieser Wartezeiten schwankt dabei stark. So dauert ein `Reset` des AWGs bis zu 5 Sekunden und das Schreiben eines Arbiträrsignals benötigt ebenso mehr als eine Sekunde. Das Oszilloskop hingegen verarbeitet ohne im Kontext dieses Aufbaus messbare Verzögerung. Verbunden damit, dass das Oszilloskop auf absehbare Zeit durch das neuere MAUI von Teledyne LeCroy ersetzt wird, wird im Folgenden nicht weiter auf die Laufzeit dieses Gerätes eingegangen. Durch diese Feststellung konnte die Laufzeit je Lese-Aufruf bereits um ungefähr 15 Sekunden reduziert werden. Bei den zur Sicherheit implementierten Warte-Befehlen handelt es sich um Status-Abfragen ähnlich den nachfolgend für das AWG erläuterten.

In der zu Beginn vorliegenden Implementierung lagen pauschale Wartezeiten nach einigen als besonders zeitintensiv erwarteten Befehlen sowohl in der Ansteuerung des AWGs als auch des Oszilloskops vor. Insgesamt blieb die Verbesserung der Zugriffszeiten auf das AWG erfolglos. Es stellt nach wie vor das Nadelöhr des Programms dar mit etwa 25 Sekunden Laufzeit je Aufruf. Es wurden eine Reihe von IEEE-488 Standardbefehlen für diesen Zweck ausprobiert, nachfolgend mit den wichtigsten Eigenschaften:

-
1. `*WAI` ist in erster Linie ein interner Befehl, zuerst alle Befehle abzuarbeiten, bevor neue angegangen werden. Dies geschieht jedoch ohne Rückgabe an den aufrufenden Code.
 2. `*OPC?` zwingt das aufrufende Programm zum Warten auf Beenden aller Befehle im AWG und führt zu einem `VisaTimeoutError` im AWG, wenn zu lange auf die Rückgabe gewartet werden muss [7, S. 9]. Dieser Fehler konnte auch nicht im Programm aufgefangen werden, sodass er folgenlos für den Ablauf geblieben wäre.
 3. `*OPC` setzt das zugehörige Bit `Standard-Event-Register` des Gerätes, sobald alle Befehle abgearbeitet sind. Im Unterschied zu `*OPC?` wird damit jedoch nicht auch das aufrufende Programm geblockt. Jedoch muss mit einer Statusabfrage über `*ESR?` dieses Register ausgelesen werden. Die dafür benötigte Schleife konnte jedoch nicht derart über PyVisa implementiert werden, dass tatsächlich mit Sicherheit alle vorigen Befehle beendet wurden.

Nicht für das AWG definiert ist der Standardbefehl `*BUSY?`, der eine 0 zurückgeben würde, wenn das Gerät gerade nicht mit Ausführen eines Befehls beschäftigt ist. Ansonsten wäre die Rückgabe 1.

Als besonders erfolgversprechend erschien der letzte Ansatz 3, der jedoch auch zeitlich der letzte war. Mit einer Schleife über `*ESR?` könnte das zugehörige Bit erneut ausgelesen und eine sehr kleine pauschale Wartezeit im Programm eingebaut werden, bevor die nächste Abfrage stattfindet. Hier könnte es sich um ein Resultat der Codierung über PyVisa handeln, da hierbei mehrere Arten, einen Befehl an das Gerät zu senden, zur Verfügung stehen.

Weiterhin wurde die pauschale Wartezeit an der im Datenblatt [6, S. 21] für die Verbindung via USB angegebenen, gemessenen Ladedauer eines Arbiträrsignals orientiert. Dieser Richtwert von 1.25 s stellte sich jedoch als zu gering für den hier vorliegenden Fall heraus.

Letztlich sei erwähnt, dass die Abfrage der Fehler im AWG mittels `SYSTem:ERRor?` möglich ist und durch das dabei erfolgte Löschen der Fehler aus dem internen Speicher nicht nur die Abfrage sondern auch das Rücksetzen des Fehlerstatus ermöglicht. Dies ist insbesondere für das Debugging vorteilhaft, wenn Befehle ausprobiert werden und die gegebenenfalls auftretenden Fehler gehandhabt werden müssen [5, S. 454]. Dies vermeidet das unter Umständen mehrmalige Trennen und Wiederherstellen der Verbindung zum AWG sowie das Löschen der Fehler an der Benutzeroberfläche.

4.2 Anpassung der horizontalen Auflösung am Oszilloskop

Um die Auflösung des Oszilloskops ausnutzen zu können, ist die Anpassung der horizontalen wie auch vertikalen Auflösung an das zu messende Signal notwendig. Während die horizontale Auflösung durch die Wiederholfrequenz des Pulses mit f_{BB} sich nahezu unabhängig des betrachteten Signals setzen lässt, muss die vertikale Skalierung an das Signal angepasst werden. Um dies etwa während einer Optimierung mit wechselnden Spannungsamplituden zu ermöglichen, sind zwei direkt aufeinander folgende Messungen notwendig. Dabei wird aus der ersten Messung die Amplitude des Signals entnommen und die Skalierung für die zweite, dann genauere Messung, dahingehend angepasst, dass das Signal bildfüllender dargestellt wird. Dieser Gewinn an Genauigkeit in der Messung stellte sich insbesondere für die Optimierung der linearen Übertragungsfunktion H als notwendig heraus.

5 Code-Design

Die Aufgabe des Projektseminars war, ein bestehendes Tool weiterzuentwickeln. Es wurde nach einer Methode gesucht, um an die vorhandene Funktionalität anzuschließen und fortzusetzen. Um auf dieses Projekt aufbauende Arbeiten zu erleichtern, wird in diesem Kapitel die Namenskonvention und die allgemeine Struktur unseres Design vorgestellt.

5.1 Motivation

Ziel ist es, ein Design aufzustellen, das mit seinen Funktionen und seiner gesamten Dokumentation für sich spricht. Des weiteren wird eine lose Kopplung der einzelnen Blöcke angestrebt, damit Bestandteile ohne die Logik anderer Funktionen verstanden und bearbeitet werden können. Dies verbessert zum einen die Arbeitsaufteilung im Team, zum anderen aber auch die Verständlichkeit des gesamten Programms. Ohne ein modulares Design ist eine Optimierung der einzelnen Bestandteile nur schwer umzusetzen.

5.2 Aufbau

Der Aufbau des Programms orientiert sich an dem vorgegebenen Modell aus Abb. 2.4.

Namenskonvention

In der nachfolgenden Tabelle sind die verwendeten Präfixe erklärt.

Präfix	Erklärung
adjust	Anpassung einer Größe in einem Iterationsschritt
compute	Berechnung einer Größe, ohne Kommunikation der Geräte
determine	Bestimmung einer Größe mit Kommunikation der Geräte
evaluate	Routinen mit Bewertung der Ergebnisse mit idealen Ergebnissen
generate	Erstellung eines Datensatzes
get	Aus der Implementierung von [1] übernommen für get_H
loop	Eine Iterierungsschleife im Optimierungsalgorithmus
measure	Messung einer Größe

Tabelle 5.1: Erklärung der Präfixe von Methoden

Ordnerstruktur

In Tab. 5.2 sind die Inhalte der verwendeten Ordner aufgelistet.

Ordner	Inhalt
blocks	Implementierung der abstrakten Schritte im Modell
classes	ADTs, siehe 5.2
data	Alle Ergebnisse
helpers	Hilfsfunktionen nach [13]
tests	Alle Klassen mit Unit Tests
tools	RF-Tools

Tabelle 5.2: Namen der Ordner

Abstract Data Types

Bestimmte Datensätze wurden als eigene abstrakte Datentypen (ADTs) implementiert, siehe [14, Kap. 6.1]. Daraus ergeben sich folgende Vorteile:

- Allgemeine Funktionalität wird in der Klasse implementiert.
- Typüberprüfung wird ermöglicht.
- Implementierung generischer Klassen¹ beispielsweise `calculate_error()` ermöglicht.

Für die Klasse `signal_class()` ergeben sich speziell noch weitere Vorteile:

- Vermeidung von ungewollten Manipulationen an Datensätzen, die nur schwer zu debuggen sind
- Das Originalsignal ist immer gespeichert.
 - dadurch Vermeidung der Fortpflanzung von Interpolationsfehlern durch Resampling.

5.3 Methodik

Für diesen Typ Aufgabenstellung konnte viel Programmierarbeit ohne den Versuchsaufbau vorgenommen werden, sodass viele Fehler schon vor dem eigentlichen Testlauf am System entdeckt und behoben werden konnten. Dafür werden zwei Möglichkeiten vorgestellt, die diese Fehlersuche ermöglicht und erleichtert haben.

5.3.1 Test Driven Development

Nach der Methode TDD werden vor dem eigentlichen Programmieren Tests geschrieben, die mit der folgenden Implementierung der Funktionalität bestanden werden müssen. Diese Tests geben ein gutes Feedback darüber, ob eine Methode gut implementiert wurde. Die folgenden Vorteile waren besonders wichtig in der Entwicklung des Codes für diese Aufgabe:

- Testen von kleinen Abschnitten erzwingt einen modularen Aufbau.
- Tests dokumentieren Funktionalität.
- Refactoring² wird ermöglicht.
- Einfachere Fehlersuche auch ohne Versuchsaufbau möglich.

¹ Implementierungen, die unabhängig vom übergebenen Datentyp sind.

² Änderung der Struktur ohne Veränderung der Funktionalität.

- Reproduktion von Messvorgängen, die Fehler aufgedeckt haben.
- Arbeitsteilung an logisch getrennten Abschnitten ermöglicht.
- Hilft bei der Migration der Implementierung aus MATLAB nach Python.

Eine große Schwierigkeit dieser Methode liegt darin, gute Tests zu formulieren, die auch alle möglichen Fälle abdecken. Das kann einige Zeit in Anspruch nehmen, lohnt sich aber auf längere Sicht

Beispiel: Test für `apply_transfer_function`

Der übliche Aufbau eines Unit Tests wird am Beispiel `test_apply_transfer_function_2` erklärt. Diese Funktion wendet eine Übertragungsfunktion auf ein Signal an und gibt das Ausgangssignal zurück. Per Definition sollte bei zweifacher Anwendung der Funktion wieder das ursprüngliche Signal zurückgegeben werden.

Als erstes werden die Daten eingelesen, mit denen gearbeitet werden soll.

```
1 H = read_in_transfer_function(mock_data_path + 'H.csv')
2 Uout = read_in_signal(mock_data_path + 'Uout.csv')
```

Im nächsten Schritt wird die Funktion angewendet.

```
1 Utransferred = apply_transfer_function(Uout, H)
2 Uout_computed = apply_transfer_function(Utransferred, H.get_inverse())
```

Der letzte Schritt ist die Auswertung. Dabei ist die Rückgabe der Funktion `finalize_test` genau dann `True`, wenn das `U_computed` gleich dem `U_accepted` ist.

```
1 test_succeeded = finalize_test(Uout_computed, set_ideal_values=False)
2 self.assertTrue(test_succeeded)
```

`U_computed` -Wert wird festgelegt und gespeichert..muss genau (auf 1e-10 getroffen werden) - Alternative über die Norm: Schwierigkeit..zu große Toleranzen, keine universelle Fehlerberechnung möglich-

5.3.2 Mock-System

Das implementierte Mock System stellt ein typisches Beispiel für ein mock object nach [16] dar und simuliert das getestete reale System nach einem bestimmten Modell. In diesem Fall wird zur Simulation ein ideales Hammerstein Modell als Grundlage verwendet, weil dieses Modell verwendet wird, um das reale System abzubilden. Dieses Mock-System testet die Möglichkeiten der implementierten Methoden ein Hammerstein Modell darzustellen und zu optimieren.

Mit diesem System konnten auch Methoden getestet werden, die eigentlich mit den Geräten kommunizieren und somit nicht gut ohne Versuchsaufbau getestet werden können. Der Ablauf sieht wie folgt aus:

- Das Eingangssignal wird nicht an das AWG übergeben sondern an die Simulation
- Verzerrung über eine nichtlineare Kennlinie K
- Berechnung des Ausgangssignals über die Übertragungsfunktion H

-
- Das Ausgangssignal wird wieder an die eigentliche Methode übergeben

Daraus folgen für diese Zielsetzung folgende Vorteile:

- Unter der Voraussetzung, dass das Mock-System richtig implementiert ist und das reale System mit dem Hammerstein Modell beschrieben werden kann, können über die Konvergenz Aussagen getroffen werden. Wenn der Optimierungsalgorithmus am realen System konvergiert, so wird er auch am Mock-System konvergieren. Umgekehrt lässt sich von der Divergenz am Mock-System auf die Divergenz am realen System schließen.
- Der Algorithmus kann ohne Versuchsaufbau entwickelt werden, weil die Konvergenz am Mock-System eine notwendige Bedingung für die Konvergenz am realen System ist, das mit einem Hammerstein Modell beschrieben werden kann.
- Das Programm kann am Mock-System mit in der Praxis selten auftretenden Szenarien getestet werden. Zum Beispiel eine hohe Anzahl an Iterationen. Dabei konnte in diesem Projekt mit über 100 Iterationen ein unerwarteter Fehler beobachtet werden.
- Durch die Implementierung des Mock-Systems wird das Modell besser verstanden.

Die Optimierungsansätze aus Kap. 6 wurden bereits mit dem Mock-System getestet und es wurde festgestellt, dass diese für das ideale Modell tatsächlich zu einer Verbesserung führen. Die Frage dabei ist aber, inwieweit sich der Versuchsaufbau durch ein ideales Hammerstein-Modell beschreiben lässt und sich damit die Verbesserung auf den realen Aufbau übertragen lässt. Je nach Ergebnissen der Übertragung auf das reale System lassen sich Aussagen über die Grenzen des Modells für diesen Aufbau treffen.

Mit dem Testen an der Simulation können Aufgaben wie Debugging, Refactoring und Erweiterungen der Funktion auch ohne den Versuchsaufbau durchgeführt werden.

6 Iterative Optimierung des Hammerstein-Modells

Ziel der Optimierung von Übertragungsfunktion \underline{H} und Kennlinie K mit ihren Parametern a ist die Minimierung des Fehlers zwischen idealem und gemessenem Ausgangssignal, $U_{out,id}(t)$ und $U_{out,meas}(t)$. Die Minimierung des relativen Fehlers ist gegeben durch

$$\min f(t) = \min \left(\frac{U_{out,meas}(t) - U_{out,id}(t)}{U_{out,id}(t)} \right) = \min \left(\frac{U_{out,meas}(t)}{U_{out,id}(t)} - 1 \right). \quad (6.1)$$

Für das verwendete Hammerstein-Modell liegt die in [3] vorgeschlagene getrennte, iterative Optimierung von \underline{H} und K nahe. Die Auswertung der Qualität des Einzelsinus erfolgt dabei durch das RF-Tool [18] unter Verwendung des als `QGesamt1` geführten Qualitätswerts ¹. Dabei wird im Folgenden unter $U_{out,id}(t)$ ein durch die Routine `generate_BBsignal` erzeugter Einzelsinus betrachtet, vergleiche hierzu 8.1.

6.1 Optimierung der linearen Übertragungsfunktion H

Die Optimierung von $\underline{H}(\omega)$ beruht auf der Annahme, dass sich Gl. (6.1) auf die komplexen Darstellungen des berechneten und des gemessenen Ausgangssignals, $\underline{U}_{out,id}(\omega)$ und $\underline{U}_{out,meas}(\omega)$ fortsetzen lässt mit

$$f_{compl}(\omega) := \frac{\underline{U}_{out,meas}(\omega)}{\underline{U}_{out,id}(\omega)} - 1. \quad (6.2)$$

Ist im Spektrum des gemessenen Signals eine Frequenz stärker vertreten verglichen mit dem idealen Signal, wurde die Verstärkung des realen Systems bei der Berechnung von \underline{U} , unterschätzt. Folglich muss die Verstärkung im Hammerstein-Modell angehoben werden. Die Korrektur der Phase ist dabei in Gl. (6.2) mitinbegriffen. Iterativ mit einer Schrittweite $\sigma_H^i \in [0, 1]$ ausgeführt, folgt für den i -ten Schritt

$$\underline{H}^{i+1} = \underline{H}^i \cdot (1 + \sigma_H^i f_{compl}^i) \quad (6.3)$$

für $\underline{U}_{out,meas}^i$ in f_{compl}^i als gemessenem Ausgangssignal für das mit \underline{H}^i berechnete Eingangssignal. ² In Abb. 6.1 sind Betrag und Phase der durch FFT erhaltenen Spektren für gemessenes und ideales Ausgangssignal vor Durchführung einer Optimierung dargestellt. Die Transformation erfolgt dabei durch die Erweiterung `numpy` in Python mit `numpy.fft`. Die Routine nutzt einen Algorithmus von Cooley und Tukey [9, S. 297-301]. Klar erkennbar sind die Nulldurchgänge des Betragsspektrums sowie die große Streuung der Phasenwerte.

Mit diesen Überlegungen gestaltet sich der Programmablauf zur Optimierung der Übertragungsfunktion wie in Abb. 6.2 illustriert.

¹ Hierauf beziehen sich alle weiteren Angaben zur Qualität des Signals. Eine intensivere Befassung mit dem Tool hat nicht stattgefunden.

² Nachfolgend wird aus Gründen der Übersichtlichkeit f_{compl} statt dem länglichen Bruch genutzt

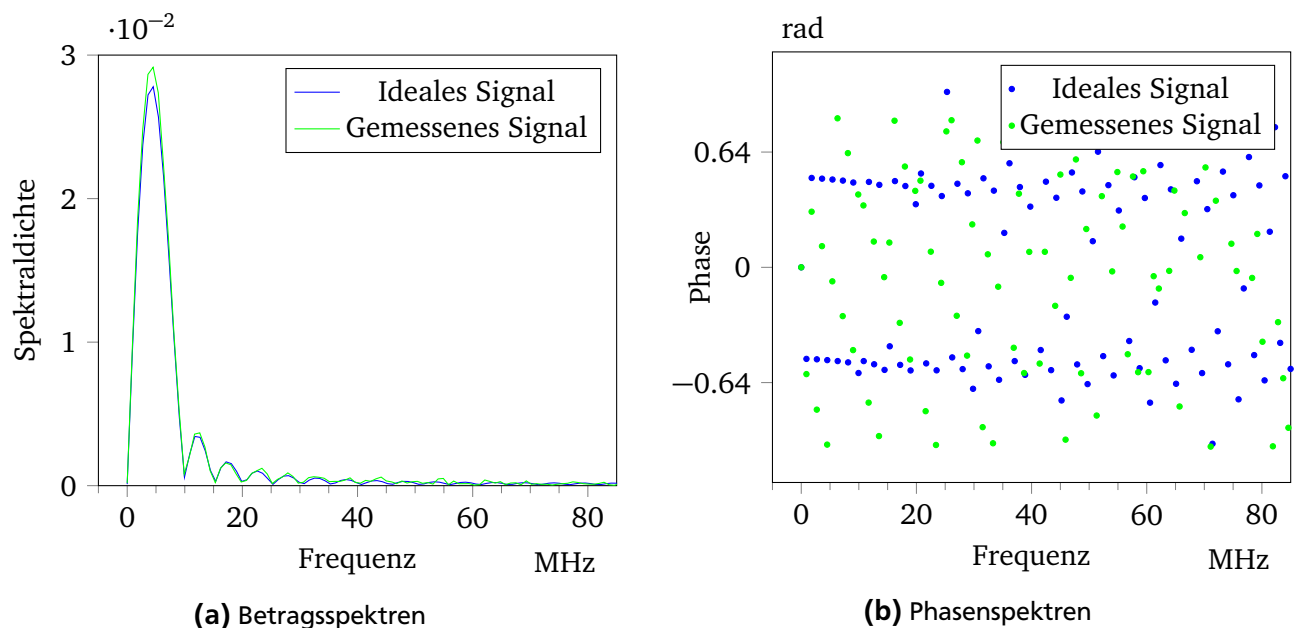


Abbildung 6.1: Spektrum des Einzelsinus-Signals, Berechnung der FFT mit jeweils 109 Punkten

6.1.1 Fehlerquellen

Neben den für kontinuierliche Funktionen problemlos definierbaren iterativen Zuweisungen ergeben sich in Messung und diskreter Ausführung jedoch Fehlerquellen. Problematisch sind insbesondere solche, die in Gl. (6.3) durch das Betragsverhältnis der Ausgangssignale verstärkt werden. Unterscheiden sich die Spektren hier um einen großen Faktor, resultiert dies in einer großen Anpassung der Übertragungsfunktion für die betreffende Frequenz. Dies ist folglich insbesondere bei kleinen Beträgen der Spektren problematisch, wenn Ungenauigkeiten und Störeinflüsse betrachtet werden.

Besondere Störeinflüsse ergeben sich also durch

- Rauschen: Unterschiedliche Rauscharten macht sich in allen Frequenzen bemerkbar mit kritischem Einfluss bei geringer Spektraldichte des Signals.[10, S. 205 ff.]
- Diskretisierungsfehler: Die FFT bedingt eine begrenzte Auflösung in den Spektren von \underline{H} und den gemessenen Signalen und liegt insbesondere im Allgemeinen an unterschiedlichen Frequenzen und mit unterschiedlich vielen Punkten vor.
- Interpolationsfehler: Die (hier lineare) Interpolation der Spektren zur Auswertung von f_{compl} an den Frequenzen von \underline{H} kann insbesondere den Einfluss oben genannter Punkte verstärken.

Weiterhin unterscheidet sich die Auflösung des Spektrums zwischen der FFT der Signale und der Darstellung von \underline{H} . Der Frequenzabstand zweier Einträge in den Signal-Spektren ist durch die Wiederholfrequenz f_{rep} gegeben. Hierdurch ergeben sich im betrachteten Bereich der Übertragungsfunktion etwa 100 Werte in den Spektren der Signale, während die Übertragungsfunktion mit über 200 Werten berechnet wird. Hieraus resultiert eine konzeptionelle Ungenauigkeit, auf die in 6.1.2 eingegangen wird.

Ein Aspekt, der sich hierzu in der Implementierung als entscheidend, aber nicht auf den ersten Blick intuitiv darstellt, liegt in der Periodendauer diskreter Signale. Bei diskreten Zeitsignalen ist zu beachten, dass die Periodendauer T nicht durch die Differenz der Zeitwerte des letzten und des ersten Wertes, sondern noch zuzüglich eines Zeitschritts zu berechnen ist, also $T = \text{time}[\text{end}] - \text{time}[0] + \text{delta_time}$, womit gelten sollte $T = N * \text{delta_time}$. Dies wirkt sich insbesondere auch auf die FFT aus, wenn die zugehörige Frequenzachse berechnet werden muss.

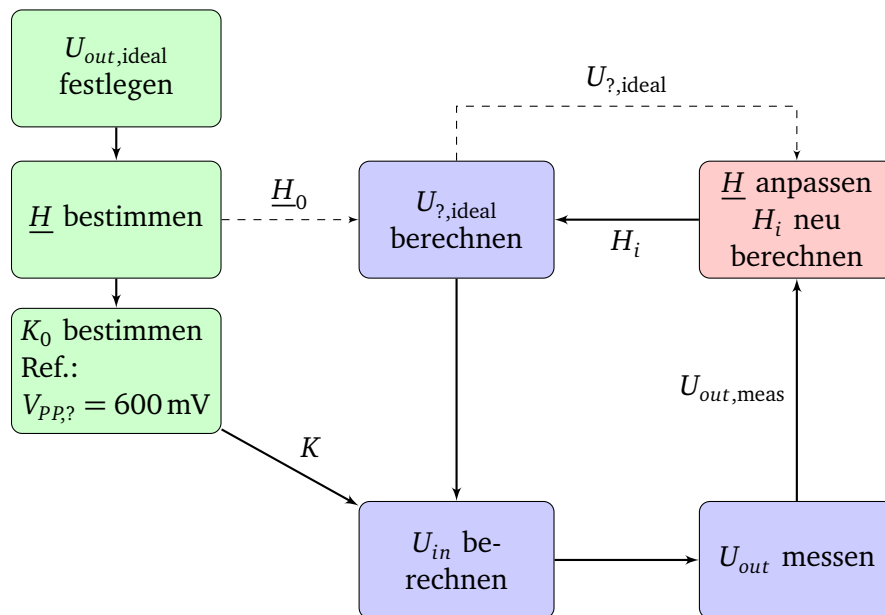


Abbildung 6.2: Algorithmus zur Optimierung von \underline{H}

6.1.2 Umgang mit Fehlerquellen

Ignorieren kleiner Beträge im Spektrum

Um Rauscheinflüsse und Probleme durch Nulldurchgänge zu dämpfen, wird ein erster intuitiver Ansatz vorgenommen: Bei den Betragsspektren der in f_{compl} eingehenden Signale, des gemessenen und idealisierten Spannungssignals, werden alle Anteile, die verglichen mit dem Maximalwert des betreffenden Spektrums besonders klein sind, auf einen vorgegebenen Wert, im Folgenden Default-Wert genannt, gesetzt. Dies führt an den betroffenen Frequenzen zu $f_{\text{compl}} = 0$ und damit zu keiner Änderung von \underline{H} . Dies bedeutet also, dass alle Einträge des Betragsspektrums von $\underline{U}_{\text{out,ideal}}$ mit weniger als zum Beispiel 3 ‰ der maximalen Amplitude auf den Default-Wert gesetzt werden. An den betroffenen Frequenzen wird auch das Spectrum von $\underline{U}_{\text{out,meas}}$ unabhängig der dort vorhandenen Werte auf den Default-Wert gesetzt, um $f_{\text{compl}} = 0$ an dieser Frequenz zu erreichen. Zu beachten bei letzterem Punkt ist die notwendige Rundung, falls die Einträge der FFT an unterschiedlichen Frequenzen vorliegen.

Mit Beschränkung auf 3 ‰ und dem globalen Minimum beider Spektren als Default-Wert werden insbesondere Einträge in den höheren Frequenzen des Spektrums 6.1a beeinflusst. Ab etwa 8 ‰ werden alle Frequenzen ab ungefähr 60 MHz auf den Default-Wert gesetzt. Mit dieser Methodik können folglich in erster Linie Korrekturen an hohen Frequenzen von \underline{H} verhindert werden. Weiterhin werden hiermit Punkte nahe der Nulldurchgänge des originalen Spektrums ignoriert, wodurch auch diese Fehlerquelle beeinflusst werden kann. Verglichen mit der nachfolgend beschriebenen Anpassung hat eine Grenze von 3 ‰ aufgrund der beschriebenen Bandbreite einen niedrigeren Einfluss auf die Qualität des Ausgangssignals. Da sie jedoch gleichzeitig nicht alle Anteile in höheren Frequenzen auf den Default-Wert setzt, wurde in den meisten Fällen mit dieser Grenze simuliert oder getestet.

Ignorieren großer Korrektur-Terme

Ein zweiter, sehr grober Ansatz liegt in der Beschränkung von f_{compl} auf Werte, die betragsmäßig unterhalb einer vorgegebenen Schwelle liegen. Zugrunde liegt die Annahme, dass die gerade an Nulldurchgängen des Spektrums aber auch bei vielen hohen Frequenzen auftretenden großen Werte durch die in

obiger Aufzählung genannten Fehlerquellen entstehen. Hier bedeutet dies insbesondere, dass die Diskretisierung die Nulldurchgänge nicht korrekt darstellen kann. Die Interpolation auf Frequenzen von \underline{H} ist dann aufgrund der großen Sprünge von Werten in direkter Umgebung der problematischen Frequenzen mit großer Ungenauigkeit behaftet. Dies kann zu den beschriebenen, großen Korrektur-Termen in f_{compl} führen. Ähnlich verhält es sich mit Änderungen durch Rauschen, die das Spektrum nicht in einen zu ignorierenden Bereich nach 6.1.2 drücken.

Listing 6.1: Pseudocode zur Veranschaulichung der Anpassung des Korrekturterms

```
rms_orig = root_mean_square( f_abs )
f_abs_to_use = f_abs[ where( abs(f_abs) >= 0.02 * rms_orig ) ]
rms_mod = root_mean_square( f_abs_to_use )
idx_to_clear = f_abs[ where( abs(f_abs) >= rms_mod ) ]
f_abs[ ix_to_clear ] = 0
```

Vereinfacht bedeutet der verfolgte Ansatz, ausnehmend große Werte von f_{compl} als unrealistisch zu bewerten. Eine Pseudo-Implementierung findet sich in Code (6.1), um die nachfolgende Erläuterung zu illustrieren. In der vorgenommenen Implementierung wurde f_{compl} an den ausgewählten Frequenzen beliebig auf 0 gesetzt, also keine Anpassung bei diesen Frequenzen ermöglicht. Dadurch werden bei dieser Anpassung verglichen mit dem Ignorieren kleiner Einträge in den Spektren wesentlich mehr Frequenzen nicht in der Optimierung berücksichtigt.

Als Grenze genutzt wird ein modifizierter Effektivwert, nachfolgend mit RMS (Root Mean Square) bezeichnet. Der reine RMS von f_{compl} unterliegt der Problematik, eine unproportional große Gewichtung von kleinen Einträgen zu enthalten.

Idealerweise enthält f_{compl} mit jeder Iteration betragsmäßig kleinere Einträge als zuvor. Es würden also bei Nutzung des reinen RMS unter Umständen mit zunehmender Schrittzahl zunehmend mehr Werte in f_{compl} ignoriert - was der Optimierung entsprechende Grenzen setzt. In Kombination mit den im vorigen Abschnitt erläuterten Anpassungen wäre die Problematik unumgänglich, da Frequenzen, die explizit nicht bei der Optimierung berücksichtigt werden sollen, den reinen RMS-Wert beeinflussen. Folglich muss der RMS modifiziert werden. In der vorliegenden Implementierung wurden zur Berechnung des modifizierten RMS nur die Werte einbezogen, die mehr als beliebig gewählte 2% des reinen RMS betragen. Es handelt sich bei dieser Anpassung um eine sehr grobe und größtenteils willkürliche Wahl der Parameter, die zu Zwecken der Illustration jedoch brauchbare Ergebnisse liefert.

Zero-Padding

Eine einfache Möglichkeit, den Frequenzabstand in der FFT der genutzten Signale zu verbessern, liegt in der Nutzung von Zero-Padding [11]. Hierbei werden einem Signal eine Reihe von zusätzlichen 0-Werten hinzugefügt. Dadurch wird die Anzahl Samples einer Periode künstlich vergrößert, woraus ein kleinerer Frequenzabstand bei der FFT folgt. Voraussetzung hierfür ist ein Signal, das bei 0 beginnt und zu 0 abfällt, damit keine Sprünge zwischen Zeitwerten auftreten. Hierdurch ist es möglich, ein Spektrum der zur Optimierung notwendigen Signale zu erreichen, dessen Frequenzabstand geringer ist als bei der Übertragungsfunktion. Dies resultiert in einer genaueren Interpolation der Spektren auf die Frequenzwerte der Übertragungsfunktion, an denen f_{compl} ausgewertet wird.³

6.1.3 Auswertung

³ Es sei darauf hingewiesen, dass die vorliegende Implementierung eine separate Interpolation von Betrag und Phase vornimmt.

In einer ersten Implementierung der Optimierung wurde aufgrund noch nicht behobener kritischer Fehler in Hilfs-Methoden die Anpassung nach Gl. (6.3) nur mit den Beträgen der betrachteten Komponenten vorgenommen. Eine Phasenanpassung fand in diesem Schritt nicht statt. Für den Korrekturterm dieser Implementierung wird im Folgenden f_{abs} verwendet, um den Unterschied zur komplexen Anpassung zu unterstreichen. Hierbei konnten die Fehlerquellen konkretisiert werden und die beschriebenen Ansätze zur Handhabung der Fehler getestet werden mit Ausnahme des unter 6.1.2 beschriebenen Zero-Paddings. Dieser Algorithmus führt jedoch nicht zu einer Verbesserung der Qualität des Ausgangssignals, wie schon der Vergleich bei Anwendung auf das Mock-System zeigt, illustriert in Abb. 6.5 mit dem Plot $\text{---}\bullet\text{---}$.

Wendet man die Beschränkung durch den RMS-Wert auf diese erste Implementierung an, wird die drastische Wirkungsweise auf den Korrekturterm f_{abs} ersichtlich, wie Abb. 6.3 illustriert. Die Einschränkung betrifft dabei nicht nur obere Frequenzen, sondern insbesondere auch solche in Umgebung der Nulldurchgänge des idealen Spektrums. Für die Einordnung der Größenordnung des Korrekturterms sei nochmals betont, dass aufgrund der Anpassung nach Gl. (6.3) ein Korrekturterm von $f_{\text{abs}} = \pm 0.5$ bei dem im Folgenden standardmäßig verwendeten $\sigma_H = 1/2$ zu einer Anpassung in \underline{H} in Höhe eines Viertels des ursprünglichen Wertes führt.

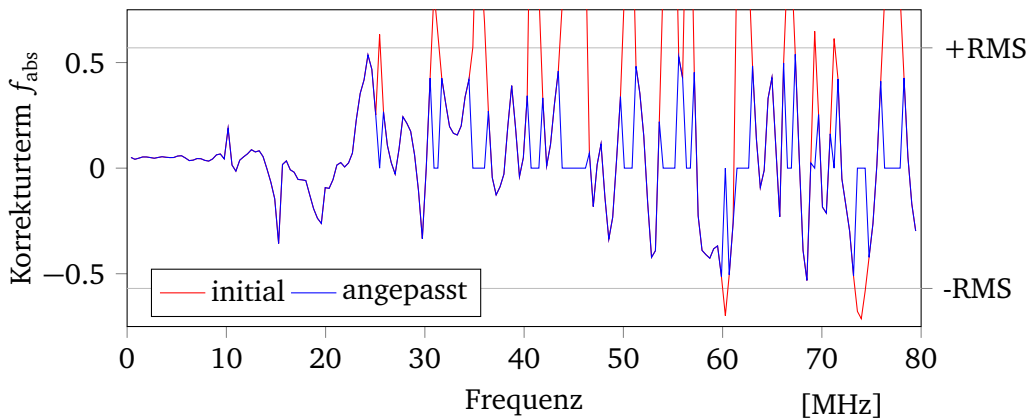


Abbildung 6.3: Korrekturterm in initialer und in angepasster Form mit RMS- Korrektur, am Messaufbau

Vergleicht man weiterhin die beiden mit identischem Setup innerhalb weniger Minuten entstandenen Messungen $\text{---}\times\text{---}$ und $\text{---}+\text{---}$ in Abb. 6.4, so wird die Bedeutung zufälliger Rauscheinflüsse auf den Erfolg der Optimierung deutlich. Trotz identischen Einstellungen schwankt die Güte des Signals bei einer Optimierung ohne jede Anpassung nach 6.1.2 derart, dass aufgrund der vorliegenden Daten eine mögliche Verbesserung über wenige Iterationsschritte nur mit großer Unsicherheit quantitativ zu beziffern ist. Auch wenn diese Anpassung in erster Linie Erkenntnisse über Fehlerquellen und die Implementierung eines funktionsfähigen Algorithmus lieferte, konnten Ergebnisse über den Unterschied zwischen der Anwendung auf das `mock`-System und auf die Kavität generiert werden. Es zeigt sich, dass der am `mock`-System nicht divergierende Algorithmus unter Nutzung der Anpassungen mit RMS und 3‰ am realen System zu einer leichten Verbesserung der Qualität führt. Dies geschieht jedoch in anderen Skalenordnungen, da die Qualität eines an der Kavität gemessenen Signals durch Rauscheinflüsse bei Gütewerten knapp über 2.1 liegt, während am `mock`-System generierte Signale mit Gütewerten von etwa 0.45 vorliegen. Die prozentuale Verbesserung ist jedoch vergleichbar.

Anwendung der komplexen Anpassung auf das Mock-System

Wie in 5.3.2 beschrieben, bietet das `mock`-System die Möglichkeit zur Simulation eines idealen Hammerstein-Modells. Bis auf zufällige Rauscheinflüsse lässt sich damit die Wirkung der Optimie-

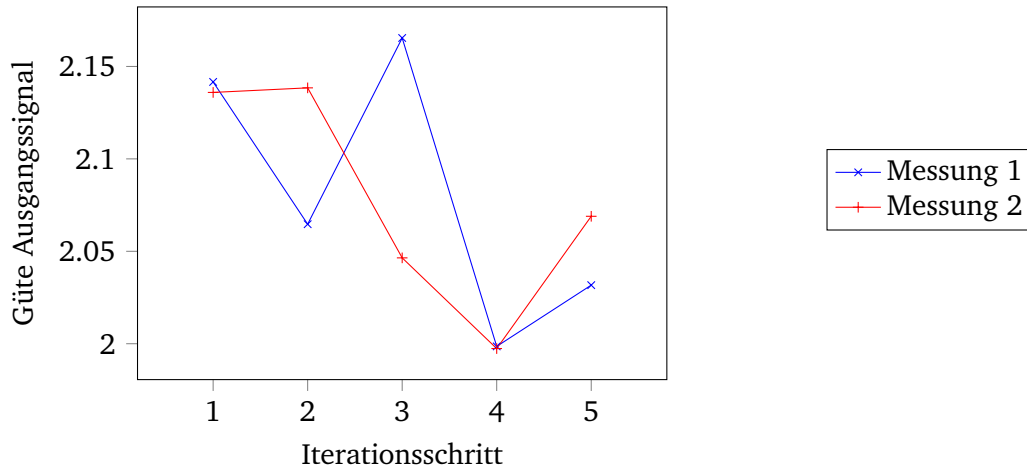


Abbildung 6.4: Entwicklung der Güte des Ausgangssignals bei erstem Algorithmus mit identischen Einstellungen. Schrittweite $1/2$, keine Manipulation des Korrekturterms.

rung qualitativ validieren. Dadurch wird es möglich, die Anwendung der komplexen Optimierung nach Gl. (6.3) zu testen und erste Aussagen zu treffen, bevor Tests am Messaufbau vorliegen. Gemäß [4] lässt sich die Kavität bis zu einer Gapspannung von ungefähr 570V gut als linear mit \underline{H} nähern. Durch den Spannungsteiler entspricht dies gemessenen Differenzspannungen von bis zu 680 mV.

Bei der Simulation unterschiedlicher Parameter für die Manipulation des Korrekturterms nach 6.1.2 zeigt sich, dass ohne eine solche Anpassung bereits im `mock`-System nicht von einer Verbesserung zu sprechen ist. Genutzt wird eine Optimierung über 10 Iterationen mit einer Ausgangsspannung von $0.3V V_{pp}$ für $U_{out,ideal}$ um die Simulation im linearen Bereich sicherzustellen und einen Vergleich zu späteren Messungen zu ermöglichen. Die Schrittweite wurde auf $\sigma_H = 0.5$ gesetzt. In Abb. 6.5 sind die Verläufe der Qualität des Ausgangssignals über 10 Iterationsschritte bei unterschiedlicher Wahl der Parameter zur Anpassung des Korrekturterms aufgetragen. Offensichtlich führen alle drei vorgestellten möglichen Anpassungen zu einer leichten Verbesserung der Qualität in den ersten Iterationsschritten. Weiterhin liegen auch beiden Anpassungen mit Nutzung einer 3‰-Grenze die Minima der Güte bei Schritt 7 beziehungsweise 8. Es lässt sich also festhalten, dass eine Anpassung des Korrekturterms notwendig ist, dabei jedoch wenige Iterationsschritte für eine Verbesserung genügen, während eine zu große Anzahl Schritte wieder zu Problemen führt.

Die Änderungen am Betrag der Übertragungsfunktion selbst stellt Abb. 6.6 nach 5 Iterationen dar. Gemäß Abb. 6.4 stellt dies einen Bereich dar, in dem die Qualität aller Anpassungen noch eine Verbesserung darstellen. Gezeigt sind die niedrigsten und höchsten Frequenzen der Übertragungsfunktion. Die Änderungen in diesen Bereichen unterscheiden sich offensichtlich signifikant je nach Parameterwahl. Diese Erkenntnisse konnten durch den ersten Algorithmus bereits am gemessenen System validiert werden. Als Vergleich dient eine Optimierung ohne zusätzliche Manipulation des Korrekturterms nach 6.1.2 sowie die tatsächlich vom `mock`-System repräsentierte Übertragungsfunktion. Die initiale Funktion wurde gegen die ideale Übertragung um den Faktor 1.05 im Betrag und 1.03 in der Phase versetzt. Während bei niedrigen Frequenzen größtenteils keine großen Änderungen an der Übertragungsfunktion vorgenommen werden, ändert sich das Spektrum von \underline{H} in hohen Frequenzen sowie einigen mittleren Frequenzbereichen wesentlich. Aufgrund der vorliegenden Daten lässt sich die Verschlechterung der Qualität in Abb. 6.5 also deuten als Einfluss der vor allem in höheren Frequenzen auftretenden starken Veränderung der Übertragungsfunktion. Während in den betragsmäßig stärker vertretenen niedrigen Frequenzen die Optimierung nach wenigen Iterationen keine nennenswerten Änderungen mehr bewirkt, verstärken sich die Ausreißer in höheren Frequenzen mit jedem zusätzlichen Schritt, sodass die Qualität wieder sinkt. Be-

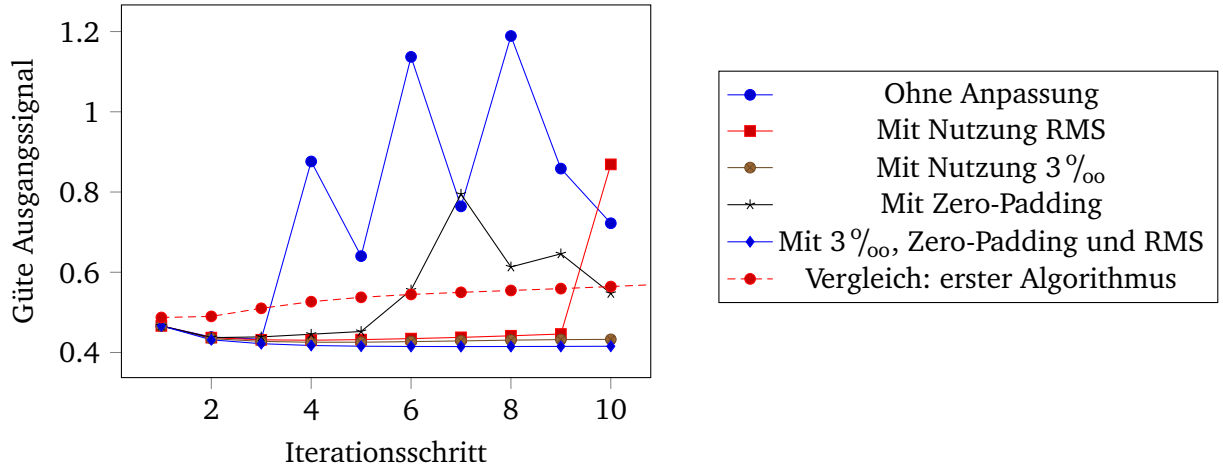


Abbildung 6.5: Entwicklung der Güte des Ausgangssignals bei Anwendung der komplexen Optimierung auf das `mock`-System bei Nutzung verschiedener Parameter zum Umgang mit Fehlern.

merkwürdig ist weiterhin, dass die Qualität des Ausgangssignals verbessert werden konnte, obwohl die berechnete Übertragungsfunktion nicht gegen die tatsächlich vom `mock`-System repräsentierte Funktion zu konvergieren scheint.

Es lässt sich bei der vorliegenden Anwendung der Optimierung am `mock`-System noch nicht auf eine Konvergenz gegen die tatsächliche Lösung schließen. Allerdings veranlassen die Ergebnisse aus dem ersten Algorithmus bezüglich der Vergleichbarkeit von `mock`-System und Messaufbau nicht dazu, eine erfolgreiche Optimierung am realen System auszuschließen. Die Messungen am realen System sind somit Gegenstand weiterer Tests. Ebenso verhält es sich mit einer Begründung der starken Anpassungen bei einigen Frequenzen im Spektrum von \underline{H} , die je nach Wahl der Parameter unterschiedlich stark und an unterschiedlichen Frequenzen auftreten. Es ist also nicht auszuschließen, dass bereits mit einer günstigen Wahl von σ_H , der Grenze für das Ignorieren kleiner Beträge in den Signal-Spektren sowie einer Schranke für den Korrektur-Term bereits nennenswerte Verbesserungen erreicht werden können.

6.2 Optimierung der nichtlinearen Kennlinie K

Die erste Kennlinie wird über ein linear vorverzerrtes Signal bestimmt. Dabei ist es möglich, dass sich die Kennlinie für ein nichtlinear vorverzerrtes Signal oder für eine Eingangssignal mit verändertem Frequenzspektrum ändert.

Für die Anpassung der Kennlinie K werden die beiden berechneten Signal $U_{?,\text{meas}}(t)$ und $U_{?,\text{ideal}}(t)$ mit einander verglichen. Diese Signale lassen sich wie folgt aus der Parametrisierung der nichtlinearen Kennlinie als Potenzreihe berechnen.

$$U_{?,\text{meas}}(t) = \sum_{n=1}^N \bar{a}_n [U_{in}(t)]^n \quad U_{?,\text{ideal}}(t) = \sum_{n=1}^N a_n [U_{in}(t)]^n \quad (6.4)$$

Dabei sind \bar{a}_n die Koeffizienten der neuen Kennlinie. Diese können mit der schon aus [2] bekannten Matrixmultiplikation berechnet werden. Bei diesem Verfahren werden einzelne Sample des Eingangssignals U_{in} und der Größe $U_?$ verglichen und das entstehende lineare Optimierungsproblem mit der Methode der kleinsten Quadrate gelöst.

Für eine Optimierung ist es notwendig eine Schrittweite zu definieren. Die Differenz der beiden Signale

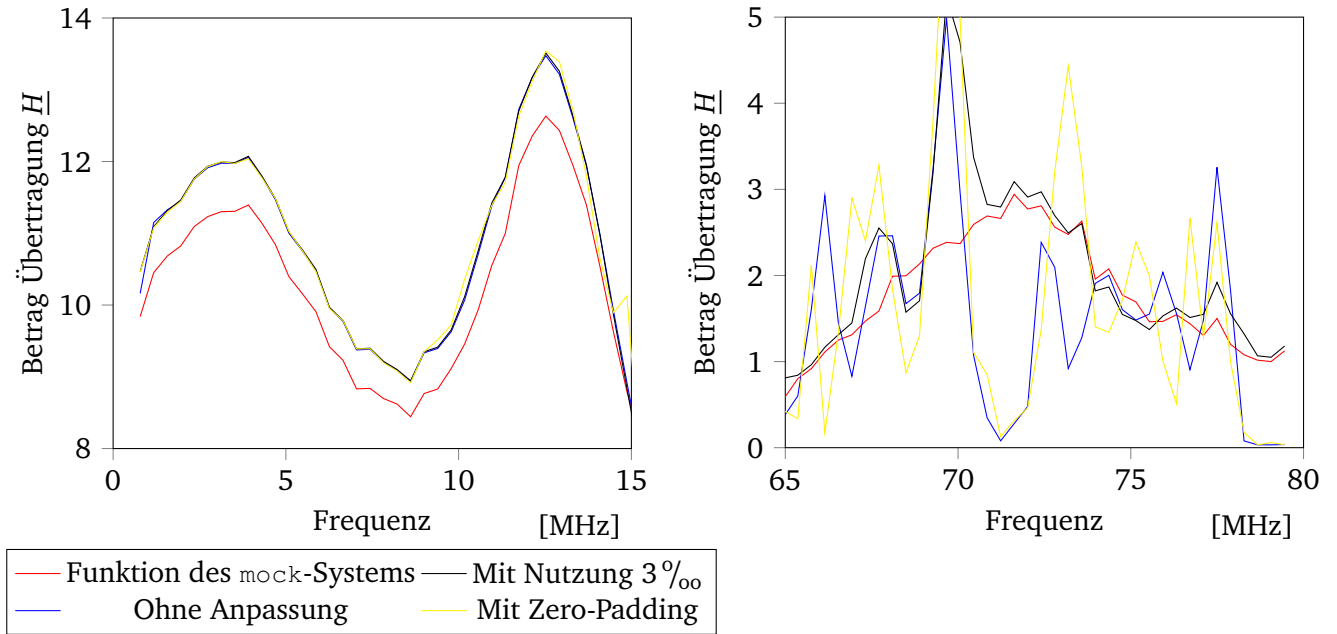


Abbildung 6.6: Vergleich unterschiedlicher Anpassungsparameter bei Anwendung der komplexen Optimierung auf das `mock`-System nach 5 Iterationen.

$U_{\gamma,\text{meas}}(t)$ und $U_{\gamma,\text{ideal}}(t)$ wird ebenfalls durch eine Potenzreihe der nichtlinearen Kennlinie parametrisiert.

$$\Delta U_{\gamma}(t) = U_{\gamma,\text{meas}}(t) - U_{\gamma,\text{ideal}}(t) = \sum_{n=1}^N (\bar{a}_n - a_n) [U_{in}(t)]^n = \sum_{n=1}^N \tilde{a}_n [U_{in}(t)]^n \quad (6.5)$$

Dafür können die Koeffizienten \tilde{a}_n direkt ohne die Berechnung von \bar{a}_n bestimmt werden. Die Berechnung der Koeffizienten \tilde{a}_n stellt ebenso ein lineares Optimierungsproblem dar. Dabei werden M Samples von $\Delta U_{\gamma,i} = \Delta U_{\gamma}(i \cdot \Delta t)$ mit zugehörigen Samples des Eingangssignals $U_{in,i} = U_{in}(i \cdot \Delta t)$ verglichen. Dieses Lösungsverfahren wird in [2] vorgestellt. Mit der Potenzreihe aus Gl. (6.5) ergibt sich folgendes Gleichungssystem

$$\begin{pmatrix} U_{in,1} & U_{in,1}^2 & \cdots & U_{in,1}^N \\ U_{in,2} & U_{in,2}^2 & \cdots & U_{in,2}^N \\ \vdots & \vdots & \ddots & \vdots \\ U_{in,M} & U_{in,M}^2 & \cdots & U_{in,M}^N \end{pmatrix} \cdot \begin{pmatrix} \tilde{a}_1 \\ \tilde{a}_2 \\ \vdots \\ \tilde{a}_N \end{pmatrix} = \begin{pmatrix} \Delta U_{\gamma,1} \\ \Delta U_{\gamma,2} \\ \vdots \\ \Delta U_{\gamma,M} \end{pmatrix} \quad (6.6)$$

Dieses Gleichungssystem ist mit normalerweise $M > N$ überbestimmt und wird mit der Methode der kleinsten Quadrate gelöst. Die Koeffizienten \tilde{a}_n werden nun wie folgt zur Anpassung der Koeffizienten a_n verwendet

$$a_n^{i+1} = a_n^i + \sigma_a^i \tilde{a}_n^i \quad (6.7)$$

Für die Schrittweite gilt $\sigma_a^i \in [0, 1]$. Der Ablauf des Algorithmus ist in Abb. 6.7 gezeigt. Dabei stellen die grünen Schritte die Initialisierung dar, blaue Schritte sind Berechnungen oder Messungen von Signalen und im roten Block findet die eigentliche Anpassung der Kennlinie statt. Durchgezogene Pfeile folgen dem Programmablauf und gestrichelte Pfeile stellen Parameterübergaben dar.

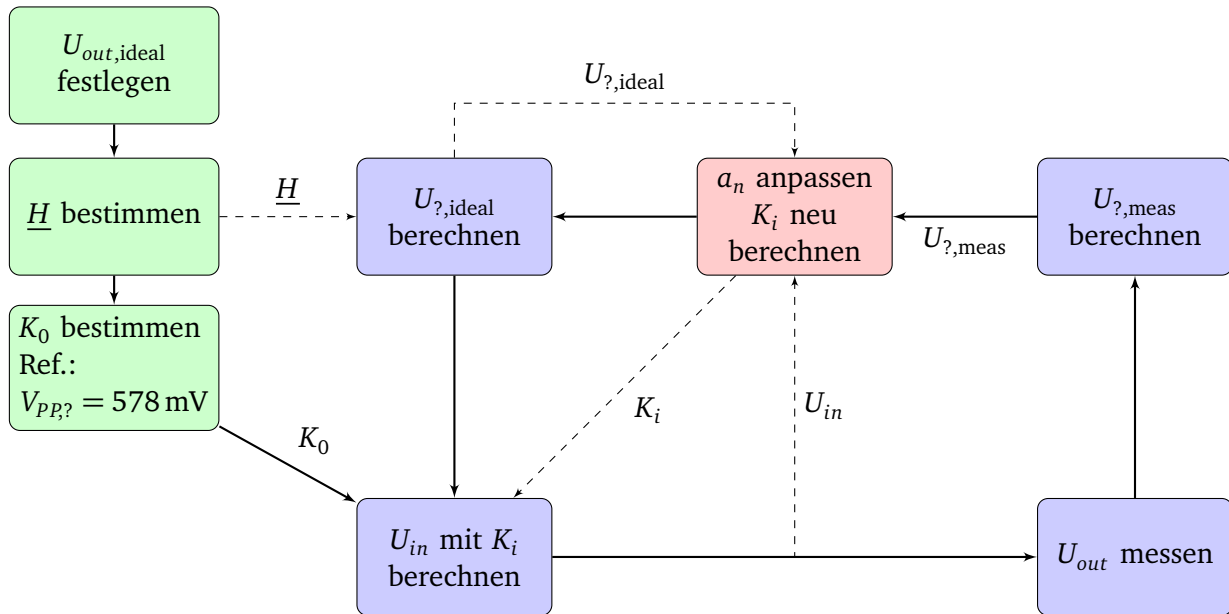


Abbildung 6.7: Algorithmus zur Optimierung von K

6.2.1 Erster Ansatz

Für die Berechnung der ersten Kennlinie K_0 wurde das ideale Ausgangssignal $U_{out,ideal}$ mit $V_{PP} = 6V$ über H^{-1} zurückgerechnet und als Eingangssignal verwendet $U_{in,initial} = U_{?,ideal}$. Dieser Wert von $V_{PP} = 6V$ wurde nicht gemessen, sondern lediglich zur Berechnung des ersten Eingangssignals verwendet, damit die Kennlinie im nichtlinearen Spannungsbereich des System berechnet werden kann. Für das berechnete Eingangssignal ergibt sich $V_{PP} \approx 587mV$.

Als erster Ansatz wurde versucht K über das selbe $U_{?,ideal}$ zu optimieren, mit dem es zuvor zur Berechnung verwendet wurde. Das heißt, dass die beiden Signale die gleiche Amplitude haben. Beim Zurückrechnen über die Kennlinie wurde festgestellt, dass K in diesem Bereich nicht mehr bijektiv ist. Für den Fall, dass das Signal zu groß für den gültigen Bereich ist wird es auf beliebig gewählte 80% des maximal zulässigen Wertes gesetzt. Zu diesem Zeitpunkt war schon bekannt, dass die Kennlinie nicht für 100% ihres gültigen Bereiches gültig ist, deshalb wurde ein kleinerer Wert gesetzt. Dabei könnten sich in jedem Iterationsschritt unterschiedliche Amplituden von $U_?$ ergeben. Ziel hierbei ist es den Bereich, in dem K bijektiv ist, schrittweise zu vergrößern.

Ergebnisse und Erkenntnisse

Das Ergebnis ist eine Schwankung der Qualität siehe Abb. 6.8b. Die Kennlinien, zu den markierten Iterationen sind in Abb. 6.8a eingezeichnet. Zu erkennen ist eine alternierende Anpassung der Kennlinie, die schon in den ersten drei Iterationen angefangen hat. Ergänzend wurden auch die letzten beiden Kennlinien eingezeichnet, um die große Anpassung innerhalb einer Iteration zu verdeutlichen. Ursache dafür könnte sein, dass die Kennlinie in einen nicht mehr sinnvollen Bereich genutzt wurde, um das Eingangssignal zu berechnen. Die Details dazu folgen im nächsten Abschnitt. Dadurch könnten zu große Änderungen in einem Iterationsschritt vorgenommen worden sein, die nicht mehr zielführend waren.

Ein weiteres Ergebnis ist auch, dass das hier verwendete Eingangssignal nicht die zur Berechnung verwendeten $V_{PP} = 6V$ über dem Gapspannungsteiler erreicht. Mit dem daraus berechneten Eingangssignal lassen sich $V_{PP} = 4.7V$ über dem Gapspannungsteiler erreichen, deshalb sollte im nächsten Ansatz zur Optimierung von einem Ausgangssignal mit höchstens diesem Wert von V_{PP} ausgegangen werden. Diese Wahl der Amplituden am Ausgang dienen lediglich der Referenz auf einen Wert. Man kann analog dazu

auch die Amplitude des Eingangssignal festlegen und damit als Referenz arbeiten.

Im nachfolgenden Diagramm der Güte wurden die Iterationen 1, 2, 3, 29 und 30 anders markiert, weil die dazu gehörigen Kennlinien K_i im nebenstehenden Diagramm eingezeichnet wurden. Die anderen Kennlinien hatten sich über dem Verlauf der Messung den Kennlinien K_{29} und K_{30} abwechselnd angenähert.

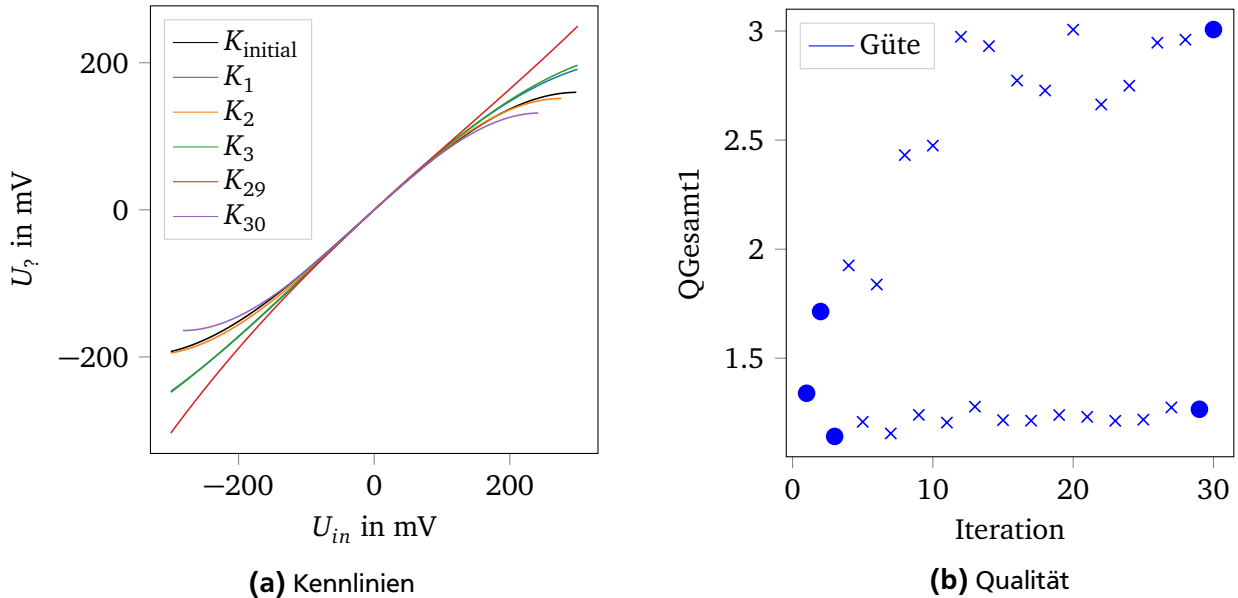


Abbildung 6.8: Erster Ansatz zur Anpassung von K

Grenzen der initial Kennlinie

Das Signal $U_{?,meas}$ wurde aus dem gemessenen $U_{out,meas}$ berechnet und ist in Abb. 6.9 eingezeichnet. Die sich daraus ergebende Kennlinie ist in Abb. 6.9a eingezeichnet. Das dazu gehörige Eingangssignal hat dabei die Form von $U_{?,ideal}$ mit $V_{pp} = 578 \text{ mV}$ und ist damit etwa doppelt so groß wie das in Abb. 6.9b eingezeichnete $U_{?,ideal}$.

Wenn man jetzt $U_{?,ideal}$ mit $V_{pp} = 300 \text{ mV}$ über K_0 zurückrechnet, um das erste nichtlinear vorverzernte Eingangssignal zu erhalten, so stellt man fest, dass dies an der Grenze von K_0 stößt, siehe auch das eingezeichnete Maximum von $U_?$ in Abb. 6.9a.

Deshalb kann damit zwar ein vorverzerntes Signal berechnen werden, aber Messungen zeigen, dass die Qualität in diesem Bereich abnimmt.

Eine Ursache dafür könnte sein, dass die Methode der kleinsten Quadrate, mit der die Parameter a_n für K bestimmt werden, die Extrema von $U_?$ nicht genug gewichtet, um in diesem Bereich noch gültig zu sein.

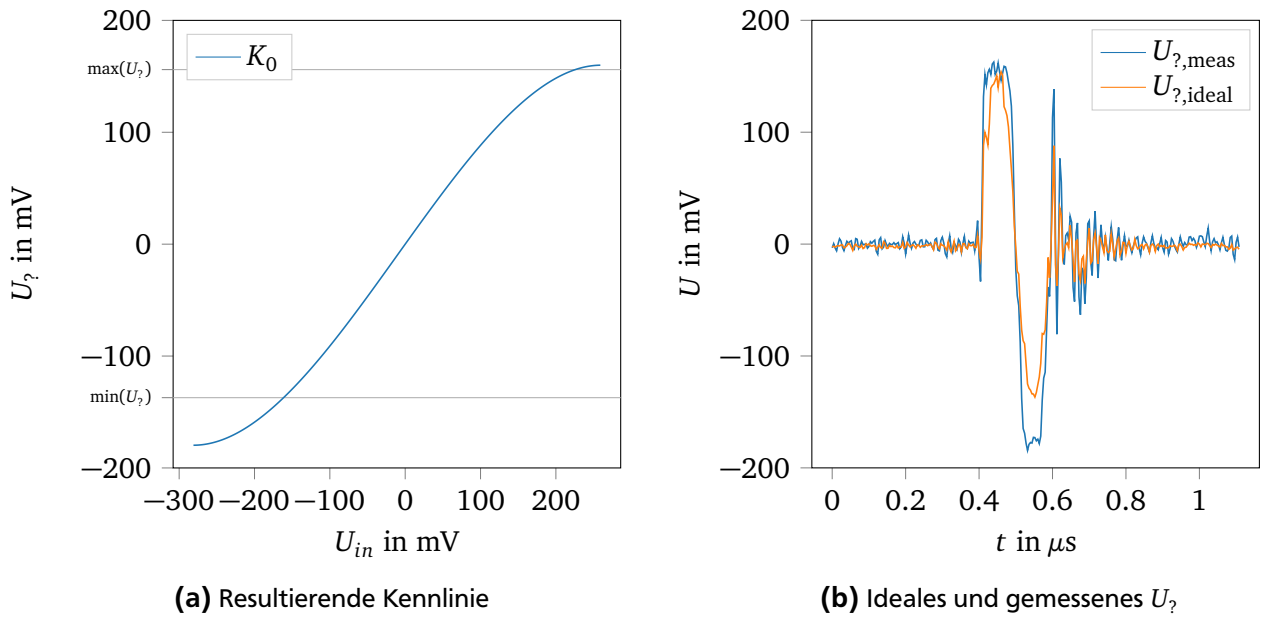


Abbildung 6.9: Nichtlineare Vorverzerrung

Für die Evaluierung der Grenzen wurden verschiedene Amplituden von $U_{?,ideal}$ über die Kennlinie nichtlinear vorverzerrt. Das Ausgangssignal wurde mit dem RF-Tool [18] bewertet und ist in Abb. 6.10b gezeigt. Dabei sind die Grenzen von $U_?$ in Abb. 6.10a mit $\min(U_?)$ und $\max(U_?)$ beschriftet dies entspricht mit $V_{pp} = 120$ mV etwa 34% des maximal möglichen V_{pp} , das von K im bijektiven Bereich zugelassen ist. Die anderen eingezeichneten Grenzen, die nicht beschriftet sind, stellen die Grenzen dar, bis zu denen die Güte des Ausgangssignals überprüft wurde.

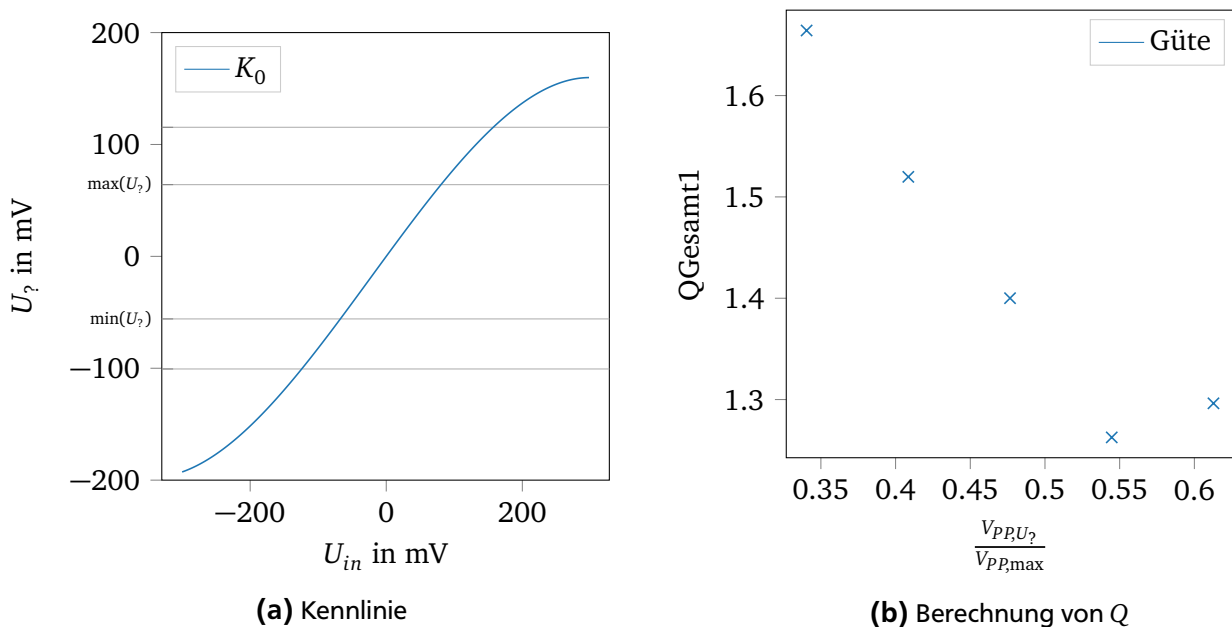


Abbildung 6.10: Bewertung des Ausgangssignals

Dabei ist der kleinste und damit der beste Wert der Güte mit 190 mV etwa 54% des maximal möglichen V_{pp} das von K aufgezeichnet worden. Diese Messung wurde nur für diese Kennlinie durchgeführt und lässt keine Schlüsse auf die allgemeine Gültigkeit für alle Kennlinien zu, deshalb sollte der optimale Wert für V_{pp} sicherheitshalbe für alle Kennlinien einzeln bestimmt werden.

6.2.2 Zweiter Ansatz

Aus den Ergebnissen aus 6.2.1 und der Evaluierung der Grenzen aus dem vorherigen Abschnitt wurde ein neuer Ansatz zur Optimierung aufgestellt, bei dem die Kennlinie K in einem kleineren Bereich optimiert wird als der, aus dem sie vorher zur Initialisierung berechnet wurde.

Als zweiter Ansatz wurde folgendes Setup verwendet:

- K_0 bestimmt über $U_{out,ideal}$ mit $V_{PP} = 6\text{ V}$
- K angepasst über $U_{out,ideal}$ mit $V_{PP} = 3\text{ V}$

Damit wurde K mit einem $U_{?,ideal}$ optimiert, bei dem V_{PP} auf 66% des maximal zulässigen Wertes gesetzt wurde. Der Bereich ist in Abb. 6.11a eingezeichnet.

Ergebnisse und Erkenntnisse

Für diesen Ansatz wurden aufgrund der späten Erkenntnis keine ausführlichen Tests durchgeführt, deshalb lässt sich darüber nur bedingt eine Aussage über die Konvergenz treffen. Aber diese niedrigen Werte der Güte konnten bisher nur mit diesem Ansatz erreicht werden. Im Vergleich zu den Ergebnissen des vorigen Ansatzes in Abb. 6.8b fällt jedoch auf, dass in den ersten Iterationen keine Verschlechterung der Qualität verglichen mit dem Initialwert auftritt.

Die Schrittweite wurde bei all diesen Messungen auf $\sigma_a = 0.5$ gelassen.

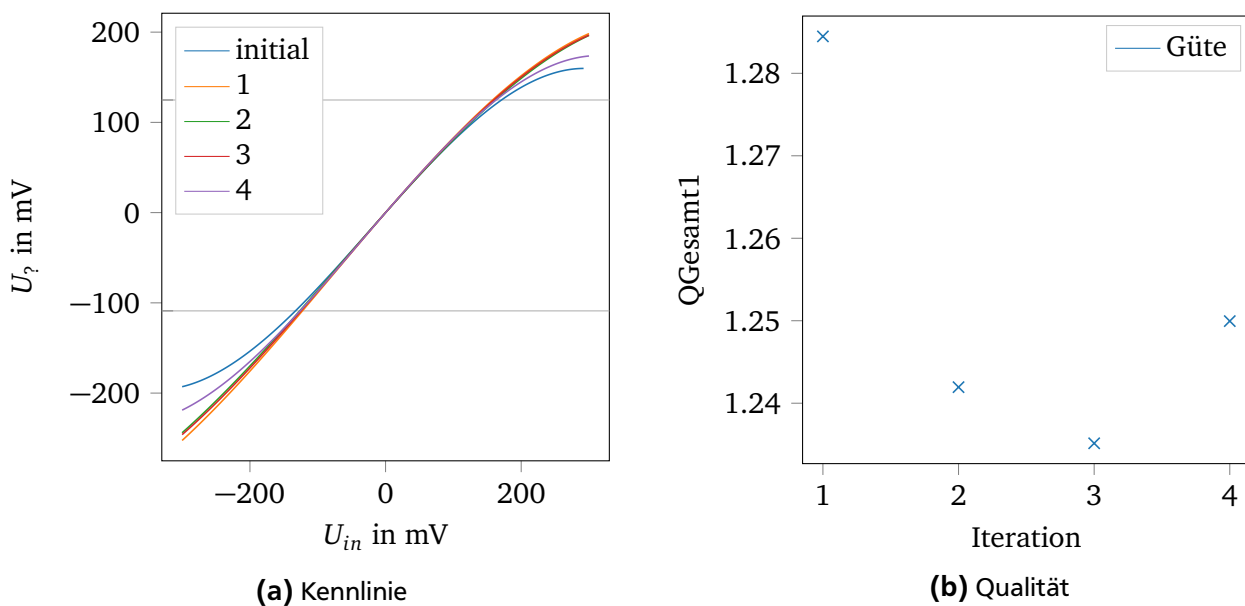


Abbildung 6.11: Zweite Ansatz zur Anpassung von K

7 Fazit

Das erweiterte Programm bietet die Möglichkeit, in einem Aufruf Kennlinie und Übertragungsfunktion des Hammerstein-Modells zu bestimmen, zu speichern und für eine weitere Auswertung vorzuhalten. Der Funktionsumfang wurde um Schleifen für die getrennte iterative Anpassung der beiden Blöcke ergänzt und ein erster Ansatz für diese Optimierung ausprobiert.

Dabei lässt sich feststellen, dass die Anpassung der Übertragungsfunktion theoretisch zielführend ist, jedoch für eine merkliche Verbesserung am realen System noch der Einfluss von Rauschen betrachtet werden muss. Auch muss ein Umgang mit den Einflüssen von Interpolation und Diskretisierung auf die Spektren der genutzten Signale gefunden werden. Die hierzu vorgenommenen Anpassungen in der Optimierung konnten zwar die Problematiken aufgreifen, stellen allerdings noch keine Lösung dar.

Die Optimierung der Kennlinie führt zur Feststellung konzeptioneller Probleme. Der Definitionsbereich des zur initialen Berechnung genutzten Signals und die Amplituden der für weitere Iterationen genutzten Signale haben entscheidenden Einfluss auf Erfolg und Qualität der Optimierung. Hier konnte die Problematik erkannt und beschrieben werden, ohne jedoch eine verlässliche Lösung anzubieten.

Es lässt sich anhand der vorgenommenen Implementierung feststellen, dass die verfolgten Ansätze das Modell nicht verschlechtern. Eine Aussage über eine merkliche Verbesserung ist mit den genutzten Algorithmen jedoch nicht abschließend möglich.

Bei erfolgreicher Durchführung einer iterativen Optimierung der Bausteine des Hammerstein-Modells ausgehend von initialen Werten könnte es möglich sein für ein beliebiges Ausgangssignal hinreichend gute Werte erhalten zu können. Wird die Optimierung weiterhin auf ein ideales Hammerstein-Modell angewendet, bietet sich die Möglichkeit zum Vergleich zwischen Messaufbau und mathematischem Modell. Auf dieser Basis kann der Algorithmus zur Beurteilung der Grenzen der vorgenommenen Modellierung für die Kavität genutzt werden. Dies ist auf Basis der im Rahmen dieser Arbeit vorliegenden Werte noch nicht möglich.

Im Entstehungsprozess der Implementierung konnte festgestellt werden, dass eine umfassende Anpassung der Code-Struktur notwendig war. Dadurch konnte verwirklicht werden, dass für weitere Arbeiten an der Implementierung eine aussagekräftige Dokumentation und ansprechende Struktur vorliegen, die einen leichten Einstieg und die einfache Erweiterbarkeit ermöglichen sollen.

Das vorliegende Design ist in modulare Blöcke aufgeteilt, die jeweils einzelne in sich geschlossene Funktionalitäten darstellen. Auch wurde ein Programmier-Konzept genutzt, bei dem bereits lange vor den Messungen an der Kavität die Funktionalität des Codes überprüft werden konnte. Die dafür elementaren Tests der modularen Blöcke können sicherstellen, dass neue Änderungen keine bereits erfolgreich implementierte Funktionalität beeinträchtigen. Das Konzept umfasst ebenso die Simulation eines Hammerstein-Modells, sodass die Optimierungsalgorithmen nicht erst am Messaufbau ausprobiert und weiterentwickelt werden mussten. Dabei ist festzustellen, dass der Prozess der Umstrukturierung und die Einbettung dieser neuen Umgebung sich ausnehmend aufwendig gestaltet hat. Fehlersuche, Übersicht und Erweiterbarkeit konnten unverkennbar verbessert werden. Dadurch konnte die Einbindung von neuen Funktionen um ein Vielfaches vereinfacht werden, wenn dies auch mit einem nennenswerten Maß an Disziplin beim Coden verbunden ist. Es lässt sich also konstatieren, dass die vorliegende Form der Optimierung ohne dieses Design nicht zu verwirklichen gewesen wäre.

Zusammenfassend stellt die vorliegende Projektarbeit eine Basis für nachfolgende inhaltliche Erweiterungen von Optimierungsansätzen am Barrier-Bucket-System dar.

8 Ausblick

8.1 Impressionen zum Code

In Bezug auf das Design des Codes verbleiben eine Reihe Anregungen und Gedanken, die nicht realisiert wurden, je nach Ausführung aber interessant zu bedenken sein könnten.

- Die Geräte AWG und DSO als Klassen einzubinden könnte den Vorteil bieten, alle SCPI Commands an einer Stelle zu bündeln und den Zugriff darauf allgemeingültig zu halten.
- Eine Einbindung des momentanen Programms in die RF-Tools des GSI-Standards könnte in zwei Teilen erfolgen. Die bestehende Funktionalität zur Berechnung von K und \underline{H} kann unabhängig der Optimierungsalgorithmen genutzt werden. Dabei ist insbesondere zu prüfen, ob oder wie die Aspekte im Code-Design beibehalten werden.
- Die momentane Version ist in erster Linie über eine Python-IDE ausführbar. Die Ausführung über die Kommandozeile wurde nicht fokussiert.

8.2 Ausblick: Optimierung

Auf den Ergebnissen aus Iterative Optimierung des Hammerstein-Modells aufbauend, verbleiben eine Reihe von offenen Fragestellungen:

1. Wie wirkt sich die Optimierung von K aufgrund ihrer Nichtlinearität auf die Übertragungsfunktion \underline{H} aus und muss dies in der Optimierung berücksichtigt werden, etwa in der Reihenfolge der Iterationsschritte?
2. Wie wird mit der Tatsache umgegangen, dass im Frequenzbereich unabhängig der Qualität der Messung nur etwa halb so viele Daten für die Anpassung zur Verfügung stehen, wie in \underline{H} selbst vorliegen?
3. Ist die getrennte Interpolation von Betrag und Phase der Signalspektren auf die Frequenzen der Übertragungsfunktion in der Optimierung von \underline{H} die beste Lösung?
4. In welcher Reihenfolge wird die Iteration durchgeführt? Wird zuerst \underline{H} in mehreren Durchgängen angepasst und danach K ? Oder im Wechsel je eine Iteration?
5. Wie wird die Auswahl der Schrittweiten σ_H und σ_K vorgenommen? Werden diese pauschal einmal festgesetzt zu Beginn des Algorithmus oder ist eine dynamische Anpassung, etwa durch die Qualität des letzten gemessenen Signals oder in Abhängigkeit des Iterationsschritts vorzuziehen?
6. Wie lässt sich der Einfluss von zufälligem Rauschen auf die Optimierung reduzieren? Insbesondere die Auflösung im höherfrequenten Betragsspektrum ist hier problematisch. Und im Falle des Ignorieren von Korrekturen an als fehlerhaft befundenen Frequenzen: Auf welchen Wert wird der Korrekturterm bei diesen Frequenzen gesetzt?
7. Wie lassen sich die im idealen Spektrum des Einzelsinus enthaltenen Nulldurchgänge in der Optimierung von \underline{H} berücksichtigen, um interpolationsbedingt große Fehlerterme zu vermeiden? Ist das einfache Ignorieren dieser Frequenzen für die Anpassung eine Möglichkeit? Und in welchem kleinen Frequenzbereich um einen Nulldurchgang müssten dann die Werte ignoriert werden?

-
8. Ist es sinnvoller, statt einer pauschalen Form der Anpassung der Korrekturterme zur Verminderung von Fehlern auf eine dynamische Version zu setzen, die spezieller auf die im jeweiligen Schritt vorliegenden Signale reagiert?
 9. Welches Qualitätsmerkmal des Ausgangssignal ist für die Optimierung entscheidend und wie wirkt sich dies auf den Algorithmus aus?
 10. Gibt es eine sinnvolle Abbruchbedingung, mit der die Iteration versehen werden sollte? Etwa, dass sich die Qualität im Vergleich zu den vorherigen Iterationen nicht mehr mit ähnlicher Rate verbessert hat? Der Trade-Off liegt zwischen Laufzeit und Signal-Qualität.
 11. Wie bestimmt man die Grenzen, in denen K genutzt werden kann? Wie kann man garantieren, dass K in den Bereichen, aus denen die Daten zur Berechnung von a_n benutzt werden, bijektiv ist?
 12. Gibt es eine Möglichkeit die Grenzen von K bei der Optimierung zu erweitern? Ein möglicher Indikator: Kann man die Rückgabe von `numpy.linalg.lstsq(LGS)` aus der Berechnung von a_n nutzen, um eine Aussage über die Abweichung der Werte zu treffen?

Literaturverzeichnis

- [1] Denys Bast, Armin Galetzka, *Projektseminar Beschleunigertechnik*, 2017.
- [2] Jens Harzheim *et al.*, *Input Signal Generation For Barrier Bucket RF Systems At GSI*, In Proceedings of IPAC2017 in Copenhagen, Dänemark. pp. 3948 - 3950 2017.
- [3] Jens Harzheim, *Idee iterative Optimierung der BB-Vorverzerrung* 2018.
- [4] Kerstin Gross *et al.*, *Test Setup For Automated Barrier Bucket Signal Generation*, In Proceedings of IPAC2017 in Copenhagen, Dänemark. pp. 3948 - 3950 2017.
- [5] Keysight Technologies, *Keysight Trueform Series Operating and Service Guide*, 2015.
- [6] Keysight Technologies, *33600A Series Trueform Waveform Generators - Data Sheet*, 2014.
- [7] C. Tröser, *Application Note: Top Ten SCPI Programming Tips for Signal Generators*, Rohde & Schwarz, 2013.
- [8] The SciPy community, <https://docs.scipy.org/doc/numpy/reference/routines.fft.html> - abgerufen am 11.08.2018.
- [9] James Cooley, John Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comput. 19, 1965.
- [10] Reinhard Lerch, *Elektrische Messtechnik, Analoge, digitale und computergestützte Verfahren* Springer, 2010.
- [11] Julius Smith, *Mathematics of the Discrete Fourier Transform (DFT), Second Edition* W3K Publishing, 2007.
- [12] I. Sommerville, *Software Engineering, 9th edition*, Pearson, 2012.
- [13] Wikipedia: The Free Encyclopedia, https://en.wikipedia.org/wiki/Helper_class - abgerufen am 11.08.2018.
- [14] Steve McConnell, *Code Complete 2*, Microsoft Press, 2004.
- [15] Testingexcellence, <https://www.testingexcellence.com/pros-cons-test-driven-development/> - abgerufen am 11.08.2018.
- [16] Wikipedia: The Free Encyclopedia, https://en.wikipedia.org/wiki/Mock_object - abgerufen am 11.08.2018.
- [17] Wikipedia: The Free Encyclopedia, https://en.wikipedia.org/wiki/System_testing - abgerufen am 11.08.2018.
- [18] Tools zur Berechnung der Verzerrungszahlen, Version: Rev. 0.0.3, Stand 15.06.2018.

9 Anhang

Name	Funktion	Input	Output
blocks			
adjust_a	Anpassung der Parameter für die Kennlinie	a_{old} U_{in} $U_{?,ideal}$ $U_{?,meas}$ σ_a	a_{new}
adjust_H	Anpassung der Übertragungsfunktion	H_{old} $U_{out,ideal}$ $U_{out,meas}$ σ_H	H_{new}
compute_a	Berechnung der Parameter für die Kennlinie	U_{in} $U_{?}$ N	a
compute_K	Aufstellen der Look-Up Tabelle für die Kennlinie	a	K
compute_Uin	Berechnung des nichtlinear vorverzerren Eingangssignals	$U_{?}$ K	U_{in}
compute_Uquest	Berechnung es linear vorverzerren Eingangssignals	U_{out} H	$U_{?}$
determine_a	Bestimmung der Parameter für die erste Kennlinie	H $U_{out,ideal}$ $f_{S,DSO}$	a_0
determine_H	Bestimmung der Übertragungsfunktion über ein Pseudorauschen nach [1]	load_csv save_csv	H_0
generate_BB_Signal	Erstellen eines idealen Barrier-Bucket Ausgangssignal	f_{rep} f_{BB} V_{PP} $f_{S,AWG}$ save_csv	$U_{out,ideal}$
get_H	Funktion um Übertragungsfunktion zu Messen	f_{max} V_{PP} bits	$[f]$ $[H]$ $[\arg(H)]$
loop_adust_a	Ein Iterationsschritt in der Anpassung von K	a_{old} K_0 $U_{out,ideal}$ Iterationen $f_{S,DSO}$	$U_{out,meas}$ $[Q_i]$ $[K_i]$
loop_adust_H	Ein Iterationsschritt in der Anpassung von H	H_0 K $U_{out,ideal}$ Iterationen $f_{S,DSO}$	$[H_i]$ $[Q_i]$ $U_{out,meas}$
measure_Uout	Messsignal aus DSO auslesen	a_{old} U_{in} $f_{S,DSO}$	U_{out}

Tabelle 9.1: Liste aller Funktionen in blocks

Name	Funktion	Input	Output
classes			
signal_class	Abstract Data Type für Signale	$[time]$ $[U]$	U
transfer_function	Abstract Data Type für Übertragungsfunktion	H_{old} $U_{out,ideal}$ $U_{out,meas}$ σ_H	H_{new}
helpers			
apply_transfer_function	Anwenden einer Übertragungsfunktion auf ein Signal	U_{out} H	$U_?$
csv_helper	Funktionen, um Werte einheitlich und einfach zu speichern und einzulesen		
MLBS	Pseudoräuschsignal bestimmen	bits	output seedRandom
overlay	Zwei Signale übereinanderlegen mit Kreuzkorrelation	U_1 U_2	$U_{1,shifted}$
plot_helper	Funktionen, um Datensätze einheitlich und einfach zu speichern		
read_from_DSO	Auslesen eines Messsignals aus dem DSO, der Parameter measure_H muss auf 1 gesetzt werden, damit das Signal für die Übertragungsfunktion richtig gemessen wird (diese Funktion sollte dringend nochmal überarbeitet werden)	$f_{S,DSO}$ $V_{PP,ch1}$ $V_{PP,out}$ f_{max} U measure_H	$[time]$ $[U_{in}]$ $[U_{out}]$
read_from_DSO_resolution	Verwendet die vorherige Funktion, um eine bessere Auflösung zu erreichen	$f_{S,DSO}$ $V_{PP,ch1}$ f_{max} U measure_H	$[time]$ $[U_{in}]$ $[U_{out}]$
signal_evaluate	Bestimmung der Güte eines Ausgangssignales mit dem RF-Tool [18]	Uout_file Result_file	$[Q]$
write_to_AWG	Signal an das AWG übergeben	U $V_{PP,AWG}$ f_{rep} $f_{S,AWG}$	

Tabelle 9.2: Liste aller Funktionen und Klassen aus classes und helpers

Erklärungen zu den Übergabeparametern

- a: Vektor mit Koeffizienten der Potenzreihe für die Kennlinie
- U: Signale werden als Objekte der Klasse signal_class übergeben
- H: Übertragungsfunktionen werden als Objekte der Klasse transfer_function_class übergeben
- K: Als Matrix mit $[U_{in}, U_?]$
- [...]: Vektoren mit Inhalt, der in den Klammern steht

Routine	Funktion
evaluate_adjust_a	Ist die Implementierung des ersten Optimierungsansatzes für K
evaluate_adjust_H	Ist die Implementierung des ersten Optimierungsansatzes für \underline{H}
evaluate_connect_Devices	Testet unterschiedliche Befehle aus der Geräte-kommunikation
evaluate_K	Testet mehrere Bereiche einer zuvor berechneten Kennlinie
evaluate_with_BBSignal	Berechnung des ersten nichtlinear vorverzerrten Eingangssignals

Tabelle 9.3: Liste aller verwendeten Routinen