Artem Moskalew
Jonas Christ
Maximilian Nolte

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhaltsverzeichnis

1 Abstract

— dieses chapter führt eine kurze Vorstellung unseres Projektseminars aus. Beinhaltet einen kurz-Überblick über wichtigste Ergebnisse und Erkenntnisse, sollte (m. E.) wenig auf den Ausblick eingehen —

2 Einführung

Mit den Barrier Bucket (BB) RF Systemen können am neu entstehenden Synchrotron SIS100 oder im Experimentier Speicherring (ESR) am GSI Helmholtzzentrum viele longitudinale Manipulationen am Teilchenstrahl vorgenommen werden. Der dazu notwendige Spannungspuls hat die Form wie in Abb. (2.1) dargestellt. Wenn die Wiederholfrequenz des Spannungspulses gleich der Umlauffrequenz ist, so wird im Phasenraum eine stationäre Potential Barriere erstellt. Wenn die Wiederholfrequenz nicht gleich der Umlauffrequenz ist verschiebt sich die Potential Barriere im Phasenraum und es entstehen Bunches mit unterschiedlicher Länge.
Der Anspruch an diese Systeme liegt darin eine hohe Qualität des Impulses am Gap der Kavität zu erzeugen, damit sogenannte Microbunches unerwünscht entstehen, deshalb müssen die Nachschwinger nach dem Einzelsinus kleiner als 2,5% von $\hat{U}_{BB}$ sein.

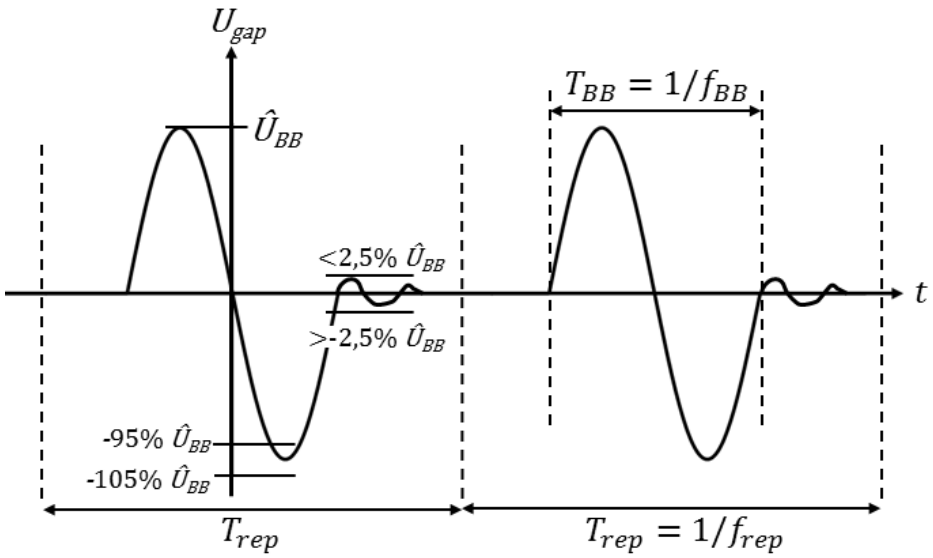


Abbildung 2.1: Ausgangssignal

Dabei benutzen wir im Folgenden f_{rep} für die Wiederholfrequenz des Spannungssignals und die Barrier Bucket Frequenz f_{BB} bestimmt die Breite der Potential Barriere. Für unsere Messungen haben wir $f_{rep} = 900\text{kHz}$ 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 AWG), einem Verstärker (AR1000A225) und der Breitband Ringkern Kavität. Der Versuchsaufbau ist in Abb. (2.2) gezeigt.
Dabei kann angenommen werden, dass sich das System bis $\hat{U}_{BB} = 550V$ 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.

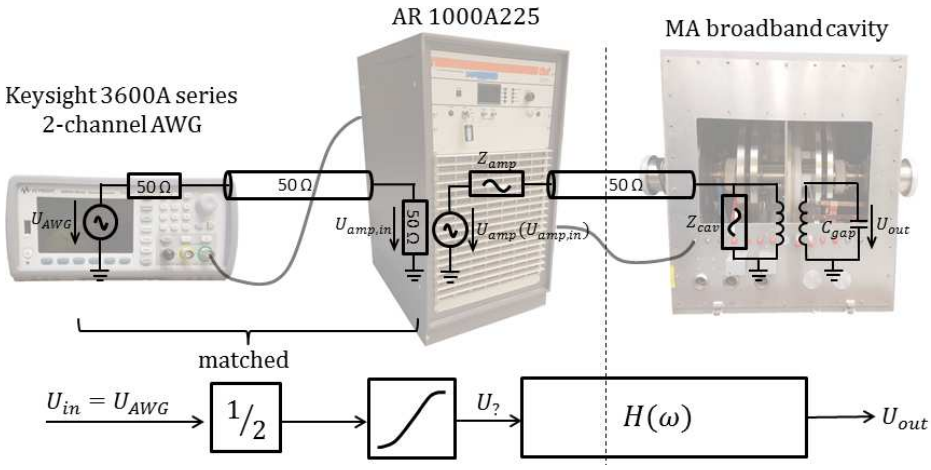


Abbildung 2.2: Versuchsaufbau und Modell

2.1.1 Hammerstein Modell

In Abb. (2.3) sind alle von uns verwendeten Größen dargestellt. U_{in} ist die Eingangsspannung vom Funktionsgenerator. Der erste Block in Abb. (2.3) stellt die nichtlineare Modellierung dar. Die Spannung $U_?$ ist eine rein virtuelle Größe und kann wie in Glg. (2.1) über die Koeffizienten a_n berechnet werden. Bei der Potenzreihe hatte $N = 3$ in der MATLAB Implementierung schon gute Ergebnisse geliefert und wurde von uns auch weiterhin so verwendet. Die nichtlinearen Kennlinie im Folgenden nur noch als K bezeichnet wird als Look-Up Tabelle in unserem Programm hinterlegt. Die Spannung U_{out} ist die Gap Spannung in der Kavität.

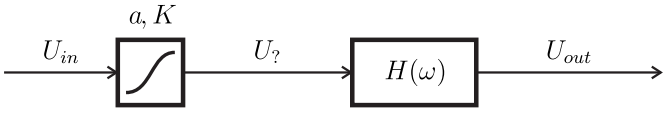


Abbildung 2.3: Hammerstein Modell

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

2.2 Motivation

Dieses Modell wurde bis auf die Berechnung von K schon erfolgreich in Python implementiert, dabei sei auf das Projektseminar [2] verwiesen. Deshalb lag unsere Motivation darin dieses Modell in Python zu vervollständigen, um dann einen möglichen Optimierungsansatz aufzustellen.

2.3 Aufgabenstellung

Im Rahmen unseres Projektseminar 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. Dabei wird die Methodik vorgestellt mit der der vorhandenen Weg fortgesetzt wurde im Hinblick darauf, dass andere Personen daran anschließen können.

3.1 Vorhandene Implementierung

Das Projekt setzt an 2 Punkten an. Zum einen das Projektseminar [2], in dem der Grundstein in Python gelegt wurde und zum anderen eine funktionierende Implementierung des nichtlinearen Bestandteils des Modells aus Abb. (2.3) nach der Vorstellung aus [3].

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 vom Oszilloskop zu messen.
- aus gegebenen Koeffizienten a_n eine Kennlinie K als Look-Up Tabelle zu berechnen.

3.1.2 MATLAB

Als zweiten Ansatzpunkt hatten wir funktionierende Methoden aus der MATLAB Implementierung des nichtlinearen Teils mit folgenden Funktionalitäten:

- $\underline{H}$ auf das Spektrum von $U_{out,ideal}$ anzuwenden.
- a_n für die nichtlineare Kennlinie zu berechnen.
- K als Look-Up Tabelle aufzustellen.
- ein vorverzerrtes Eingangssignal über K zu berechnen.

3.2 Funktionalität reproduzieren

Als ersten wurde versucht die vorhandene Implementierung zu überblicken 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 auch 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.

3.2.1 Neues Design

Das in Abb. (2.3) 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 in Kap. 4 genauer erklärt ist und dabei möglichst selbsterklärend dieses Modell abbildet. Dieses Design wurde auch in Hinblick auf die Team Arbeit gewählt, damit an mehreren logisch getrennten Abschnitten gleichzeitig gearbeitet werden kann.
Dafür wurde eine Github Repository erstellt, um eine Versionen Kontrolle vorzunehmen.

3.2.2 Tests

Aus den erstellten Daten der beiden 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.

3.2.3 Fehlersuche

Die Möglichkeiten der Fehlersuche soll an zwei Beispielen erklärt werden. In der Python Implementierung aus dem Projektseminar [2] war schon ein erster Entwurf zur Berechnung der Koeffizienten a_n gegeben, der aber noch keine guten 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 (5.2). 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. 5. Der Ansatz wurde mit zwei Routinen getestet und mit dem RF-Tool zur Güteberechnung des Ausgangssignals evaluiert.
Das Tool befand sich zu diesem Zeitpunkt noch in der Entwicklung und Testphase, sodass erst einmal herausgefunden werden musste, wie das Tool die Daten verarbeiten kann.
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 mehr Iterationsschritten noch optimiert werden sollte. Dazu werden im folgenden Abschnitt Ansätze vorgestellt.

3.4 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 IEE-488 Standard, sodass unabhängig vom konkreten Gerät oder der Art der Verbindung eine Reihe von Befehlen existiert, die gemäß IEE-488.2 gebräuchliche Standards bieten [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.

Laufzeitoptimierung

Im vorliegenden Messaufbau ist es notwendig, beim Konfigurieren des AWG 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. Einen Eindruck der zeitlichen Größenordnung bietet Tab. (3.1).

(a) Knotenliste

$P_\#$	x	y
1	$\cos(72°)$	$\sin(72°)$
2	1	0
3	$\cos(72°)$	$-\sin(72°)$
4	$-\cos(36°)$	$-\sin(36°)$
5	$-\cos(36°)$	$\sin(36°)$
6	0	0

Tabelle 3.1: Laufzeit unterschiedlicher Befehle am Keysight 33622A

In der zu Beginn vorliegenden Implementierung lagen pauschale Wartezeiten nach einigen Befehlen sowohl in der Ansteuerung des AWG als auch des Oszilloskops vor.

Die Wartezeiten für das AWG wurde letztendlich mit ——— angepasst. Die Laufzeit der Routine zum Schreiben eines Arbiträrsignals in der speziellen hier geforderten Form wurde damit von etwa 25 s auf ungefähr — reduziert.

Erfolglos blieben Versuche mit einigen anderen Standardbefehlen. `*WAI` und `*BUSY?` sind keine für das AWG definierten Befehle. `*OPC?` zwingt Python zum Warten auf Beenden aller Befehle im AWG und führt zu einem `TimeoutError`, da die Ausführung zu lange für die internen Routinen von Python braucht, vergleiche hierfür [7, S. 9]. 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.

Es wurde festgestellt, das die Verarbeitungsgeschwindigkeit des Oszilloskops bei den hier benötigten Abfragen und Befehlen kein Hindernis darstellt und die Ausführung der Befehle in Python ohne Zeitverzögerung möglich ist. Durch diese Feststellung konnte die Laufzeit je Lese-Aufruf bereits um ungefähr 15 s reduziert werden. Bei den zur Sicherheit implementierten Warte-Befehlen handelt es sich um Status-Abfragen ähnlich den oben für das AWG erläuterten.

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.

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.

3.5 Kritische Punkte

Infolge der Auseinandersetzung mit dem anfangs vorliegenden Code sowie im Zuge der Ausarbeitung sind einige Punkte aufgetreten, an deren Bedeutung die implementierte Funktionalität teilweise essentiell hing, die jedoch im Zuge der Implementierung für Probleme sorgten. Diese seien aufgrund ihrer unterschiedlichen Natur hier stichpunktartig erläutert:

- 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 = time[end] - time[0] + delta_time`, womit gelten sollte `T = N * delta_time`. Dies wirkt sich insbesondere auch auf die FFT aus, wenn die zugehörige Frequenzachse berechnet werden muss.

4 Code-Design

Die Aufgabe des Projektseminars war ein bestehendes Tool weiter zu entwickeln. Es wurde nach einer Methode gesucht an die vorhandene Funktionalität anzuschließen und fortzusetzen. Gleiches wird wahrscheinlich mit dieser Projektarbeit passieren. Deshalb wird in diesem Kapitel die Namenskonvention und die allgemeine Struktur unseres Design vorgestellt.

4.1 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 angestrebt, damit Bestandteile ohne die Logik anderer Funktionen verstanden und bearbeitet werden können. Dies verbessert zum einen die Arbeitsaufteilung im Team zum anderen auch die Verständlichkeit des gesamten Programms.
Ohne ein modulares Design ist eine Optimierung der einzelnen Bestandteile nur schwer umzusetzen.

4.2 Aufbau

Der Aufbau des Programms orientiert sich an dem vorgegebenen Modell aus Abb. (2.3).

4.2.1 Namenskonvention

In der nachfolgenden Tab. (4.1) 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 [2] übernommen für get_H
`loop`	Eine Iterierungsschleife im Optimierungsalgorithmus
`measure`	Messung einer Größe

Tabelle 4.1: Erklärung der Präfixe von Methoden

4.2.2 Ordnerstruktur

In Tab. (4.2) sind die Inhalte der verwendeten Ordner augelistet.

Ordner	Inhalt
`blocks`	Implementierung der abstrakten Schritte im Modell
`classes`	ADTs siehe 4.2.3
`data`	Alle Ergebnisse
`helpers`	Hilfsfunktionen nach [9]
`tests`	Alle Klassen mit Unit Tests
`tools`	RF-Tools

Tabelle 4.2: Namen der Ordner

4.2.3 Abstact Data Types

Bestimmte Datensätze wurden als eigene abstrakte Datentypen (ADTs) implementiert, siehe [10, Kap. 6.1]. Daraus ergeben sich folgende Vorteile:

- Allgemeine Funktionalität wird in der Klasse implementiert
- Typüberprüfung wird ermöglicht
- Implementierung generischer Klassen 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 Orginalsignal ist immer gespeichert
  - dadurch Vermeidung der Fortpflanzung von Interpolationsfehlern durch Resampling

4.3 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.

4.3.1 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 Abschnitt erzwingt einen modularen Aufbau
- Tests dokumentieren Funktionalität
- Refactoring wird ermöglicht
- Einfachere Fehlersuche auch ohne Versuchsaufbau möglich
- Reproduktion von Messvorgängen, die Fehler aufgedeckt haben
- Arbeitsteilung an logisch getrennten Abschnitten ermöglicht

- Vergleich zwischen der Implementierung aus MATLAB und der aus 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

4.3.2 Mock-System

Das implementierte Mock System stellt ein typtisches Beispiel für ein mock object nach [12] dar und simuliert das getestete reale System nach einem bestimmten Modell. In diesem Fall wird zur Simulation ein ideales Hammerstein Modell als Grundlage verwenden, 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 das Ausgangssignals über die Übertragungsfunktion $\underline{H}$
- Das Ausgangssignal wird wieder an die eigentliche Methode übergeben

Unter der Annahme, dass der Versuchsaufbau sich hinreichend genau als Hammerstein Modell beschreiben lässt, wurde die Konvergenz der Optimierungsideen in der Simulation schon erreicht. Ein weiterer Vorteil dieser Methode war, dass man durch den Aufbau der Simulation das System besser verstehen konnte. Dadurch können die Aufgaben Refactoring, Debugging und Optimierung des Codes einer Person zugeteilt werden, die selten bis gar keinen Zugriff zu den Geräten hat.

5 Iterative Optimierung des Hammerstein-Modells

—- In diesem Kapitel werden die Aspekte der durchgeführten Optimierungs-Algorithmen erläutert — Ziel der Optimierung von Übertragungsfunktion $\underline{H}$ und Kennlinie K mit ihren Parametern a ist die Minimierung des Fehlers zwischen idealem und gemessenem Ausgangssignal, $U_{out,\mathrm{id}}$ und $U_{out,\mathrm{meas}}$. Die Minimierung des relativen Fehlers ist also gegeben durch

$$\min f(t) = \min\left(\frac{U_{out,\mathrm{meas}} - U_{out,\mathrm{id}}}{U_{out,\mathrm{id}}}\right) = \min\left(\frac{U_{out,\mathrm{meas}}}{U_{out,\mathrm{id}}} - 1\right). \tag{5.1}$$

Für das verwendete Hammerstein-Modell liegt die in [?] vorgeschlagene getrennte, iterative Optimierung von $\underline{H}$ und K nahe. Die Auswertung der Qualität des Einzelsinus erfolgt dabei durch das RF-Tool von — Zitat RF-Tool — mit Entwicklungsstand vom — — unter Verwendung des als `QGesamt1` geführten Qualitätswerts [1].

5.1 Optimierung der linearen Übertragungsfunktion H

—insbesondere gemessene Daten, ohne jedwede Anpassung /Limitierung der Faktoren —

Die Optimierung von $\underline{H}(\omega)$ beruht auf der Annahme, dass sich Glg. (5.1) auf die Betragsspektren des berechneten und des gemessenen Ausgangssignals, $\underline{U}_{out,\mathrm{id}}(\omega)$ und $\underline{U}_{out,\mathrm{meas}}(\omega)$ fortsetzen lässt mit

$$f_{\mathrm{abs}}(\omega) := \frac{\mathrm{abs}\left(\underline{U}_{out,\mathrm{meas}}(\omega)\right)}{\mathrm{abs}\left(\underline{U}_{out,\mathrm{id}}(\omega)\right)} - 1\,. \tag{5.2}$$

Ist im Betragsspektrum des gemessenen Signals eine Frequenz mit halbem Betrag verglichen mit dem idealen Signal vertreten, wird dies entsprechend der Linearität der Übertragunsfunktion dahingehend gedeutet, dass die Verstärkung von $\underline{H}$ bei dieser Frequenz um einen Faktor 2 zu gering ist. Iterativ mit einer Schrittweite σ_H ausgeführt, folgt für den i-ten Schritt

$$\mathrm{abs}\left(\underline{H}^{i+1}\right) = \mathrm{abs}\left(\underline{H}^{i}\right) \cdot \left(1 - \sigma_H^i \, f_{\mathrm{abs}}^i\right) \tag{5.3}$$

für $\sigma_H^i \in [0,1]$ und $\underline{U}_{out,\mathrm{meas}}^i$ in f_{abs}^i als gemessenem Ausgangssignal für das mit $\underline{H}^i$ berechnete Eingangssignal [2]. Würde allerdings Glg. (5.3) mit komplexen Zahlen und nicht allein den Beträgen ausgeführt, würde auch die Phase der -1 beachtet und folglich die durch σ_H skalierte komplexe Zahl wesentlich verändert. Also muss für das Phasenspektrum eine andere Optimierung erfolgen. Eine Möglichkeit hierfür wäre die simple Anpassung der Phase $\arg\left(\underline{H}\right) = \varphi_H$ mit

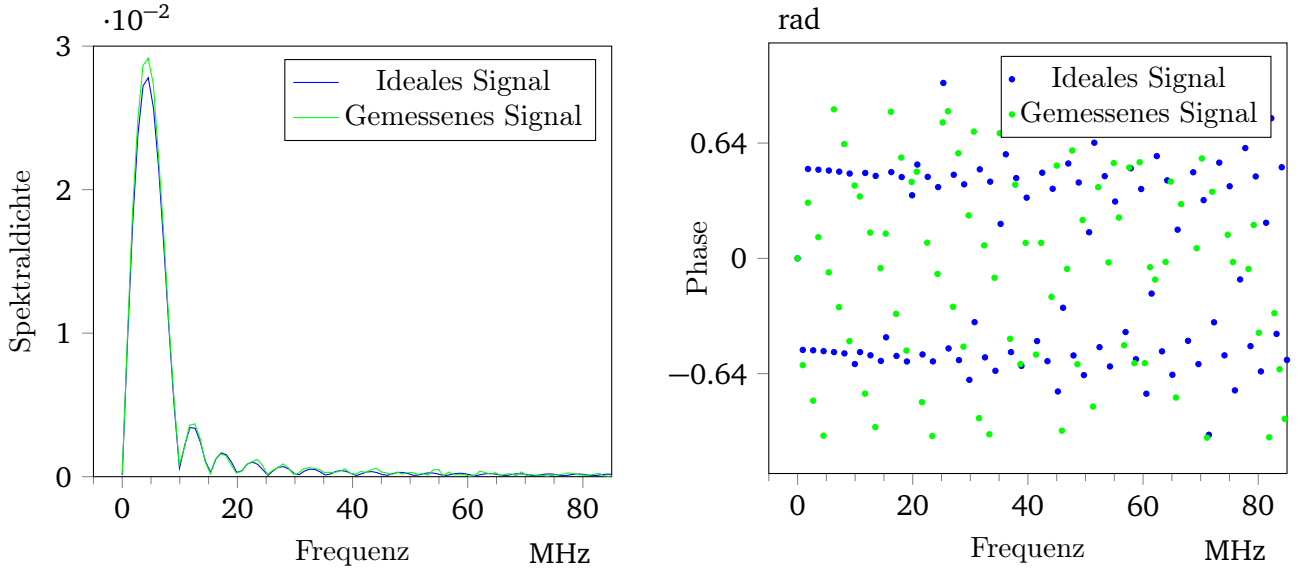
$$\varphi_H^{i+1} = \varphi_H^i - \sigma_\varphi^i \left(\arg\left(\underline{U}_{out,\mathrm{meas}}\right) - \arg\left(\underline{U}_{out,\mathrm{id}}\right) \right) \tag{5.4}$$

mit $\sigma_\varphi^i \in [0,1]$. Diese Anpassung der Phase wurde jedoch nur kurzen Tests unterzogen und anschließend nicht weiter verfolgt. Es hat sich die Signalform des Ausgangssignals unproportional stärker verändert,

[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_{abs} statt dem länglichen Bruch genutzt

als dies nur im Falle der Betrags-Anpassung der Fall war. Vermutlich liegt dies an dem aus dem Ausgangssignal gewonnenen Phasengang, der in wesentlich größerem Maße vom idealen Phasengang abweicht als im Betragsspektrum. In Abb. (5.1) sind Betrag und Phase der durch FFT erhaltenen Spektren für gemessenes und ideales Ausgangssignal vor Durchführung einer Optimierung dargestellt. Insbesondere illustriert Abb. (??) die bei gemessenem Signal auftretende Streuung der Phase.



(a) Betragsspektren

(b) Phasenspektren

Abbildung 5.1: Spektrum des Einzelsinus-Signals, berechnet und gemessen mit je 109 Punkten

Eine Aufstellung der rein auf Glg. (5.3) beruhenden Anpassung der Übertragungsfunktion über mehrere Iterationsschritte findet sich in — Abb —. Neben den für kontinuierliche Funktionen problemlos definierbaren iterativen Zuweisungen ergeben sich in Messung und diskreter Ausführung jedoch Fehlerquellen. Problematisch sind insbesondere solche, die in Glg. (5.3) 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: Weißes Rauschen macht sich in allen Frequenzen bemerkbar mit kritischem Einfluss bei geringer Spektraldichte des Signals.
- 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_{abs} an den Frequenzen von $\underline{H}$ kann insbesondere den Einfluss oben genannter Punkte verstärken.

Weiterhin zeigt sich auch in der geringen Stützstellenzahl in Abb. (5.1) bereits eine erste konzeptuelle Problematik des Vorgehens. Der Frequenzabstand zwischen zwei Werten der FFT ist stets mit der Wiederholfrequenz f_{rep} gegeben und lässt sich somit nicht durch eine höhere Auflösung der Messgeräte verbessern, der betrachtete Bereich bis 80 MHz nicht besser auflösen. Folglich setzt sich diese Ungenauigkeit auch auf die Optimierung der Kennlinie fort. Insbesondere relevant wird dies, da die Kennlinie mit nahezu der doppelten Anzahl an Werten erstellt wird und somit der Interpolationsfehler ungleich größer wird als bei ähnlicher Anzahl Stützstellen.

Ignorieren kleiner Beträge im Spektrum

Um Rauscheinflüsse und Probleme durch Nulldurchgänge zu dämpfen, wurde einer erster intuitiver Ansatz vorgenommen: Bei den Betragsspektren der in f_{abs} eingehenden Signale, des gemessenen und idealisierten Spannungssignals, wurden 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 $f_{\text{abs}} = 0$ und damit keiner Änderung von $\underline{H}$. Dies bedeutet also, dass alle Einträge des Betragsspektrums von $U_{out,\text{ideal}}$ mit weniger als zum Beispiel $5\,‰$ der maximalen Amplitude auf den Default-Wert gesetzt werden. Insbesondere werden auch die Einträge an den Frequenzen zurückgesetzt, die im Spektrum von $U_{out,\text{meas}}$ klein gegen das zugehörige Maximum sind.
Zu beachten bei letzterem Punkt ist die notwendige Rundung, wenn die Einträge der FFT an unterschiedlichen Frequenzen vorliegen.

Mit Beschränkung auf $5\,‰$ und dem globalen Minimum beider Spektren als Default-Wert ergeben sich die angepassten Betragsspektren wie in —- ABB — zu sehen, der Übersichtlichkeit halber mit kleinem vertikalen Ausschnitt. Mit diesem Schritt ergibt sich über drei Iterationen ein Verlauf des Korrekturterms f_{abs} und der Übertragungsfunktion für eine Schrittweite $\sigma_H = 1/2$ wie in —- ABB — dargestellt.

Ignorieren großer Korrektur-Terme

Ein zweiter, sehr grober Ansatz liegt in der Beschränkung von f_{abs} auf Werte unterhalb einer vorgegebenen Schwelle. Zugrunde liegt die Annahme, dass die gerade an Nulldurchgängen des Spektrums sowie 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 Ungenauickgeit behaftet. Dies kann zu den beschriebenen, großen Korrektur-Termen in f_{abs} führen.

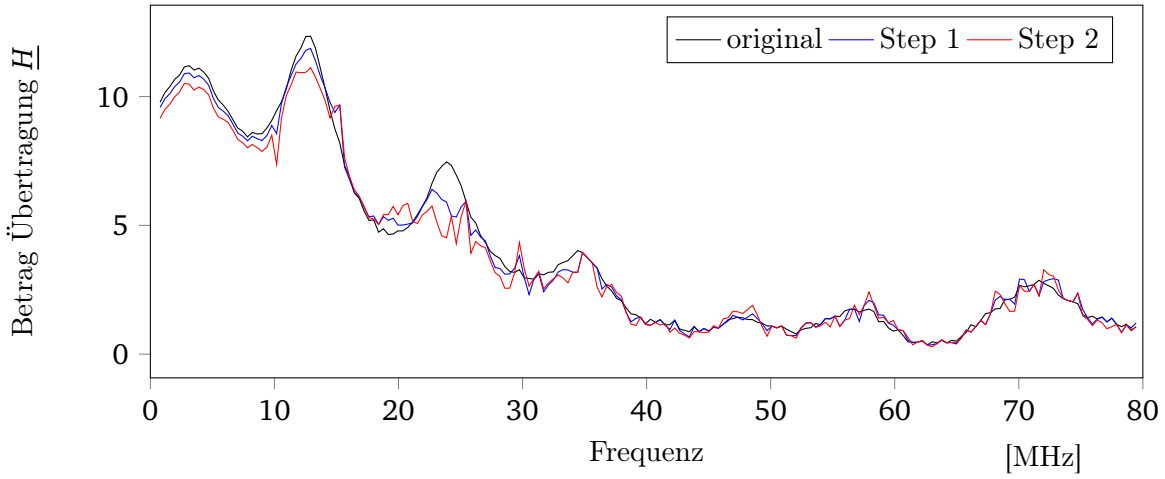
Listing 5.1: Pseudocode zur Veranschaulichung der Anpassung des Korrekturterms

```
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
```

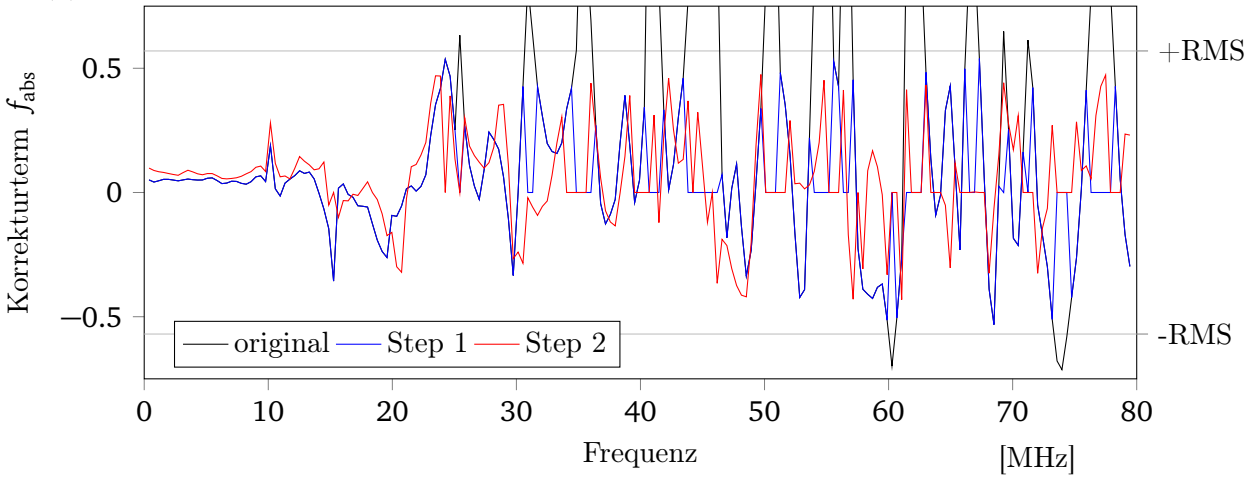
Vereinfacht bedeutet der verfolgte Ansatz, ausnehmend große Werte von f_{abs} als unrealistisch abzutun. Eine Pseudo-Implementierung findet sich in 5.1, um die nachfolgende Erläuterung zu illustrieren. In der vorgenommenen Implementierung wurde f_{abs} an den ausgewählten Frequenzen auf 0 gesetzt. Als Grenze genutzt wurde ein modifizierter Effektivwert, nachfolgend mit RMS (Root Mean Square) bezeichnet. Der reine RMS von f_{abs} unterliegt der Problematik, eine unproportional große Gewichtung von kleinen Einträgen zu enthalten.
Idealerweise enthält f_{abs} mit jeder Iteration kleinere Einträge als zuvor. Es würden also bei Nutzung des reinen RMS unter Umständen mit zunehmender Schrittzahl zunehmend mehr Werte in f_{abs} 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 Anpassung berücksichtig werden sollen, den reinen RMS-Wert beeinflussen. Folglich muss der RMS modifiziert werden. Hier wurden zur Berechnung des modifizierten RMS nur die Werte einbezogen, die mehr als beliebig gewählte $2\,\%$ des reinen RMS betragen. Es handelt sich also bei der vorgenommenen

Anpassung um eine sehr grobe und größtenteils willkürliche Wahl der Parameter, die zu Zwecken der Illustration jedoch brauchbare Ergebnisse liefert.

In Abb. (5.2) ist die Entwicklung von Übertragungsfunktion und f_{abs} über mehrere Iterationen aufgetragen. Der Einfluss des RMS-Cutters macht sich dabei verglichen mit —- ABB oben, rein iteriert — bemerkbar, es treten weniger starke Änderungen auf. Gleichzeitig zeigt sich, dass nicht in jedem Schritt an exakt den gleichen Stellen am jeweiligen RMS geschnitten werden muss. Dies belegt die Zufälligkeit des Fehlers und erläutert die prinzipielle Berechtigung der Methodik. Es zeigt sich, dass die vorgenommene Anpassung keinen großartigen Einfluss auf die Qualität des Signals hat. Dies ist insofern beachtenswert, als dass die Übertragungsfunktion auch an einigen Stellen mit massiver Verstärkung stark angepasst wird, vergleiche hierzu Abb. (5.2a) bei etwa 25 MHz. Die Qualität des Signals bewegt sich zwischen einem Wert von —- —- und —- —- und zeigt vor allem rauschbedingte Schwankungen.



(a) Entwicklung des Betrags der Übertragungsfunktion über mehrere Iterationen und im Anfangszustand



(b) Entwicklung des Korrekturterms in angepasster Form über mehrere Iterationen und in initialer, nicht angepasster Form - RMS aus Step 1 zum Vergleich

Abbildung 5.2: Entwicklung von Übertragungsfunktion und Korrekturterm bei Beschränkung von f_{abs} mit angepasstem RMS-Wert und Schrittweite $\sigma_H = \frac{1}{2}$

5.2 Optimierung der nichtlinearen Kennlinie K

Der Unterschied zur Optimierung von $\underline{H}$ ist, dass diese Optimierung im Zeitbereich statt findet. Deshalb kann Glg. (5.1) zu

$$\Delta U_? (t) = U_{?,\text{meas}} (t) - U_{?,\text{ideal}} (t) \qquad U_{?,\text{meas}} (t) = \mathscr{F}^{-1} \left\{ \underline{H}^{-1} (\omega) \cdot \underline{U}_{out,\text{meas}} (\omega) \right\} \tag{5.5}$$

geändert werden. Bei den Funktionen $U_{?,\text{meas}} (t)$ und $U_{?,\text{ideal}} (t)$ handelt es sich um Polynome gleichen Grades deshalb lässt sich die Differenz ebenfalls als ein Polynom mit Grad N darstellen

$$\Delta U_? (t) = \sum_{n=1}^{N} \tilde{a}_n \left[U_{in}(t) \right]^n \tag{5.6}$$

Die Berechnung der Koeffizienten $\tilde{a}_n$ stellt ebenso ein lineares Optimierungsproblem dar wie schon die Berechnung der Koeffizienten a_n in Glg. (2.1) siehe [3]. Dabei werden M Samples von $\Delta U_{?,i} = \Delta U_?(i \cdot \Delta t)$ mit zugehörigen Samples des Eingangssignals $U_{in,i} = U_{in}(i \cdot \Delta t)$ verglichen. Mit der Potenzreihe aus Glg. (5.5) ergibt sich folgendes Gleichungssystem

$$\begin{pmatrix} U_{in,1} & U_{in,1}^2 & \dots & U_{in,1}^N \\ U_{in,2} & U_{in,2}^2 & \dots & U_{in,2}^N \\ \vdots & \vdots & \ddots & \vdots \\ U_{in,M} & U_{in,M}^2 & \dots & U_{in,M}^N \end{pmatrix} \cdot \begin{pmatrix} \tilde{a}_1 \\ \tilde{a}_2 \\ \vdots \\ \tilde{a}_N \end{pmatrix} = \begin{pmatrix} \Delta U_{?,1} \\ \Delta U_{?,2} \\ \vdots \\ \Delta U_{?,M} \end{pmatrix} \tag{5.7}$$

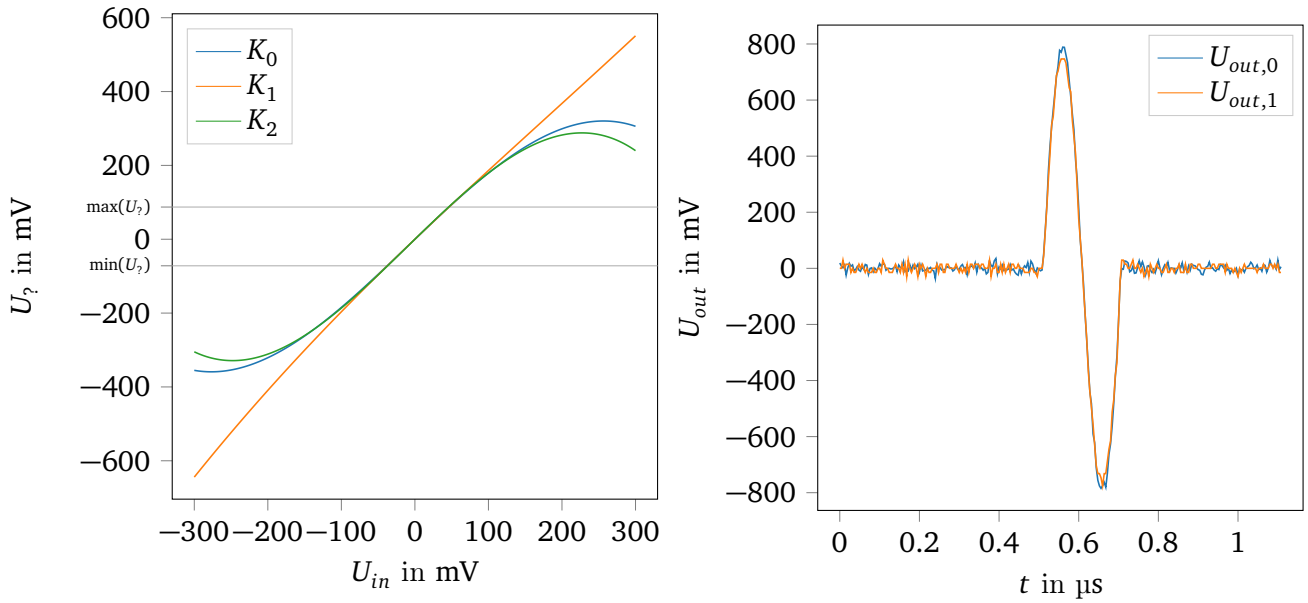
Dieses Gleichungssystem ist mit normalerweise $M > N$ überbestimmt und wird mit der Methode der kleinsten Quadrate gelöst. Die Koeffizienten $\tilde{a}_n$ werden nun wie folgt zur Anpassung der Koeffizienten a_n verwendet

$$a_n^{i+1} = a_n^i + \sigma_a^i \tilde{a}_n^i \tag{5.8}$$

Für die Schrittweite gilt $\sigma_a^i \in [0, 1]$.

Erste Ergebnisse

Für die Berechnung der ersten Kennlinie K_0 wurde das ideal Ausgangssignal $U_{out,\text{ideal}}$ über $\underline{H}^{-1}$ zurückgerechnet und als Eingangssignal verwendet $U_{in,\text{initial}} = U_{?,\text{ideal}}$. Dabei wurde $V_{PP} = 600\,\text{mV}$ gesetzt, um K_0 in einen größeren Bereich berechnen zu können.
Wenn man jetzt $U_{?,\text{ideal}}$ mit $V_{PP} = 600\,\text{mV}$ über K_0 zurückrechnet, um das erste nichtlinear vorverzerrte Eingangssignal zu erhalten, so stellt man fest, dass die Grenzen, in denen K_0 invertiert werden kann, zu klein sind. Wenn also $U_{?,\text{ideal}}$ über die Grenzen von K_0 geht, so wäre ein möglicher Ansatz V_{PP} auf den maximal von K_0 zulässigen Wert zu setzen.
Als andere Möglichkeit die Kennlinie anzupassen könnte man V_{PP} von $U_{?,\text{ideal}}$ verkleinern siehe Abb. (5.3a)



(a) Kennlinie

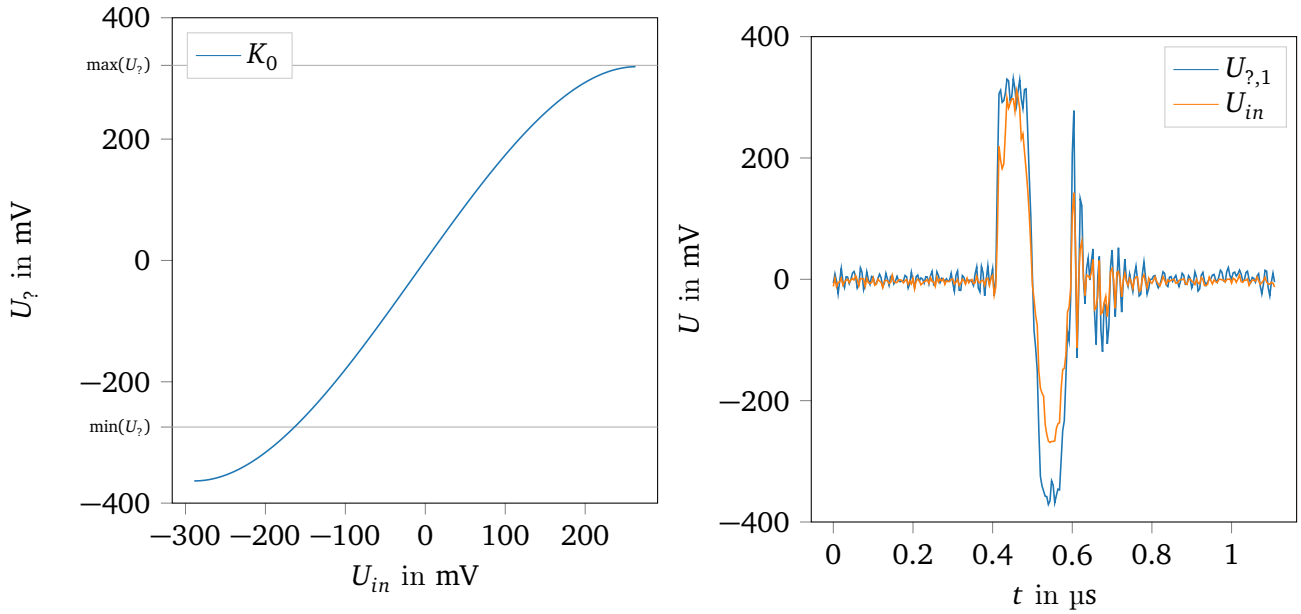
(b) Ausgangsspannung

Abbildung 5.3: Anpassung von K

In Abb. (5.3a) ist K_0 die initial Kennlinie und K_1 ist die Kennlinie nach der ersten Anpassung und K_2 nach der 2. Anpassung. Dabei wurde $U_{?,\text{ideal}}$ so berechnet, dass für $U_{out,\text{ideal}}$ gilt $V_{PP} = 1.5\,\text{V}$. Die Grenzen sind in Abb. (5.3a) angezeigt.

Grenzen der initial Kennlinie

In Abb. (5.4b) sind die Spannungen eingezeichnet mit denen K_0 berechnet wurde. Dabei gilt für $U_{in} = U_{?,\text{ideal}}$, was sich durch $U_{out,\text{ideal}}$ mit $V_{PP} = 3\,\text{V}$ berechnen lässt. Die Spannung $U_{?,1}$ wurde aus dem gemessenen $U_{out,\text{meas}}$ berechnet.
Das eingezeichnete Maximum von $U_?$ in Abb. (5.4a) liegt über der zulässigen Grenze von K_0, deshalb kann damit kein vorverzerrtes Signal berechnet werden. -Erläuterung des Problems mit der least square method: Grenzen bekommen nicht genug Gewichtung, um genau berechnet zu werden. Und damit nicht garantiert im bijektiven Bereich liegen-



(a) Kennlinie

(b) Berechnung von K

Abbildung 5.4: Nichtlineare Vorverzerrung

6 Fazit

—- In diesem Kapitel wird eine kurze Evaluierung vorgenommen. Welche Aspekte der Problemstellung wurden erfüllt (welche nicht), welche Hindernisse genommen? Einordnung der eigenen Arbeit in Kontext der in der Einleitung geführten Rahmenbedingung? Welche Erkenntnisse sind besonders erwähnenswert? Hier können Erfahrungen mit den gedachten Vorteilen (siehe ??) des Codes oder eine Bewertung der Sinnhaftigkeit der Optimierung nochmals geführt werden.—
Erfahrung: ähnlich viel Zeitaufwand in Aufräumen des gegebenen Codes wie notwendig war, um (komplexe) Funktionalität zu erweitern. -> Erweiterbarkeit viel leichter, weitere Möglichkeiten im Idealfall schneller möglich einzubetten und zu testen.

Auf Basis der im Rahmen dieser Arbeit vorliegenden Werte lässt sich keine Aussage darüber treffen, ob das verwendete Modell für jedes Signal (Signalform und Amplitude) gleichermaßen genau das Verhalten des Messaufbaus simuliert. Deshalb könnte es ausgehend von initialen Werten für die Bausteine des Hammerstein-Modells möglich sein, über eine iterative Optimierung eine Anpassung an $\underline{H}$ und K derart vorzunehmen, dass sie für das gewünschte $U_{out,\text{ideal}}$ hinreichend gute Werte liefert. Dies könnte etwa zur Kalibrierung der Kavität auf Grundlage der Charakteristik des Vortages genutzt werden. Vorteilhaft ist, dass die iterative Optimierung unabhängig der zur Berechnung der initialen Charakteristik genutzten Signale arbeiten könnte und damit etwa eine Verbesserung der Kennlinie für die momentan gewünschte Amplitude der Gapspannung möglich wäre. Es ließen sich mit einer solchen Optimierung beim Versuch, hohe Amplituden zu erreichen, auch Aussagen über die - falls vorhandenen - Grenzen des Hammerstein-Modells für den Messaufbau treffen.
Bewertung der Zielführung der Optimierung

7 Ausblick

—- In diesem Kapitel wird auf offene Fragen / neue Probleme / Anstöße für weitere Arbeiten eingegangen. Dabei sollte es um eher inhaltliche Aspekte gehen (u. U. wenig to dos für Code-Design) gegebenenfalls darf hier bei vielem auf die Erfahrungen aus den vorigen Kapiteln verwiesen werden und damit einen Übersichts-Charakter haben (erleichtert nachfolgenden Projekten die Arbeit) —

7.1 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.

7.2 Ausblick: Optimierung

- (offene Punkte K-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. In welcher Reihenfolge wird die Iteration durchgeführt? Wird zuerst $\underline{H}$ in mehreren Durchgängen angepasst und danach K? Oder im Wechsel je eine Iteration?
4. Wie wird die Auswahl der Schrittweiten σ_H und σ_a 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?
5. Wie lässt sich der Einfluss von zufälligem Rauschen auf die Optimierung reduzieren?
6. 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?

7. Wie - insofern überhaupt - ist eine Optimierung der Phase von $\underline{H}$ zu gestalten?

8. Nach welchem Qualitätsmerkmal wird das Signal bewertet und wie wirkt sich dies auf den Algorithmus aus?

9. 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.

10. 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 benutzten werden, bijektiv ist?

11. 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.

7.3 — Gerätekomm—

— in dieser Section werden weitere Punkte der Geräte-Komm aufgegriffen, etwa - die (geringe) Auflösung des AWG im Kontext der Optimierung (ggf. in 7.2 besser?) , die Einbindung des neuen Oszis oder die Idee der Klassen-Implementierung

Literaturverzeichnis

[1] Leslie Lamport, LaTeX: a document preparation system, Addison Wesley, Massachusetts, 2nd edition, 1994.

[2] Denys Bast, Armin Galetzka, Projektseminar Beschleunigertechnik, 2017.

[3] 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.

[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] I. Sommerville, Software Engineering, 9th edition

[9] https://en.wikipedia.org/wiki/Helper_class

[10] Code Complete 2 by Steve McConnell

[11] https://www.testingexcellence.com/pros-cons-test-driven-development/

[12] https://en.wikipedia.org/wiki/Mock_object

[13] https://en.wikipedia.org/wiki/System_testing

8 — Anhang —