Rückgewinnung der Frequenzinformation aus PWM-moduliertem Licht zur Anwendung für die Visible Light Communication (VLC)

Abschlussarbeit

zur Erlangung des akademischen Grades
Bachelor of Engineering

eingereicht an der
Fachhochschule Brandenburg
Fachbereich Technik
Studiengang IT-Elektronik

von: Norman Schmidt
geb. am 28.08.1990, in Brandenburg an der Havel

Betreuer: Prof. Dr.-Ing. B. Hoier, FH Brandenburg
Dipl.-Ing. (FH) Sebastian Freidank, FH Brandenburg

Brandenburg an der Havel, den 28.08.2014

Fachhochschule
Brandenburg
University of
Applied Sciences
Fachbereich
Technik

Prüfungsausschuss

Aufgabenstellung zur Bachelorarbeit

Herr	Norman Schmidt
Matrikelnummer:	2010 4002
Studiengang:	IT-Elektronik

Thema:
Rückgewinnung der Frequenzinformation aus PWM-modulierten Licht zur Anwendung für die Visible Light Communication (VLC)

Schwerpunkte:

- Recherche zur digitalen Signalverarbeitung und zu Softwarewerkzeugen, die für VLC-Systeme erforderlich sind,
- Untersuchung zur Auswahl eines Satzes von 8 PWM-Frequenzen, die VLC geeignet sind (Optimierung von Frequenzabstand (absolut und relativ) und Dekodierbarkeit)
- Simulation eines digitalen PWM-Testsignals im Zeit- und Spektralbereich
- Auswertung des Spektrums zur Rückgewinnung der Frequenzinformation mit Hilfe der FFT und digitaler Filter unter Nutzung einer Simulationssoftware
- Test und Dokumentation der Arbeitsergebnisse

Betreuer:	Prof. Dr.-Ing. B. Hoier, FH Brandenburg
	Dipl.-Ing. (FH) Sebastian Freidank, FH Brandenburg
1. Prüfer:	Prof. Dr.-Ing. B. Hoier, FH Brandenburg
2. Prüfer	Dipl.-Ing. (FH) Sebastian Freidank, FH Brandenburg

Ausgabe der Aufgabenstellung:
Abgabe der Bachelorarbeit bis zum:

Prof. Dr. rer. nat. Thomas Kern

Stand: 04.06.2014

Kurzreferat

Schmidt, Norman: Rückgewinnung der Frequenzinformation aus PWM-moduliertem Licht zur Anwendung für die Visible Light Communication (VLC),Abschlussarbeit FH Brandenburg, Fachbereich Technik/IT-Elektronik 2014, 35 Seiten, 20 Abbildungen, 14 Tabellen, 42 Blätter im Anhang, CD (77 Dateien) im Anhang.

Referat

Gegenstand dieser Arbeit, ist die Rückgewinnung von Frequenzinformationen aus pulsweitenmoduliertem Licht. Einleitend wird auf das zu bearbeitende Thema eingegangen. Anschließend werden die Grundlagen erläutert, welche für das Verständnis der Arbeit notwendig sind. Dem folgt ein Entwurf mit mehreren Möglichkeiten zur Herleitung eines Satzes von PWM-Frequenzen. Danach wird auf Grundlage der PWM-Frequenzen, eine Simulation für die Frequenzrückgewinnung durchgeführt.

Abstract

Subject of this work is the recovery of frequencies from pulse-width-modulated light. In the beginning is the problem for this topic explained. After that, the bases which are necessary for understanding of this work will be explained. Followed by a number of design options for the PWM frequencies. On the basis of the PWM frequency, a simulation is performed. In the simulation, the recovery of the frequencies to be tested.

Inhaltsverzeichnis

Abkürzungsverzeichnis

A/D-Wandler ..... Analog-Digital Wandler
DFT ..... Diskrete Fouriertransformation
DSV ..... Digitale Signalverarbeitung
FIR ..... Finite Impulse Response
H ..... Harmonische Schwingung
IIR ..... Infinite Impulse Response
PIND ..... Positiv-Intrinsisch-Negativ Diode
PWM ..... Pulsweitenmodulation
TIA ..... Transimpedanz Amplifier
TP ..... Tiefpass

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Einleitung

Die vorliegende Arbeit behandelt die Untersuchung von pulsweitenmodulierten Lichtquellen im Frequenzbereich. Sie entstand im Rahmen eines Projekts an der Fachhochschule Brandenburg, welche sich mit der Visible Light Communication beschäftigt. In der heutigen Zeit wird die Pulseweitenmodulation in der Lichttechnik zur Dimmung des Lichtes der Leuchte benutzt.

Es soll untersucht werden, ob eine Frequenzrückgewinnung von mehreren pulsweitenmodulierten Lichtquellen möglich ist. Der Hintergrund dafür ist der Aufbau einer Kommunikationsstrecke. Die Frequenzen für die pulsweitenmodulierten Lichtquellen dienen als Informationsträger und sollen mittels Frequenzumtastung wechseln. Die Ermittlung der Frequenzen einer Lichtquelle stellt im Zeitbereich kein Problem dar. Das Problem ist die Überlagerung von zwei oder mehr Lichtquellen. Abbildung 1.1 stellt eine zeitliche Überlagerung von zwei pulsweitenmodulierten Signalen dar.

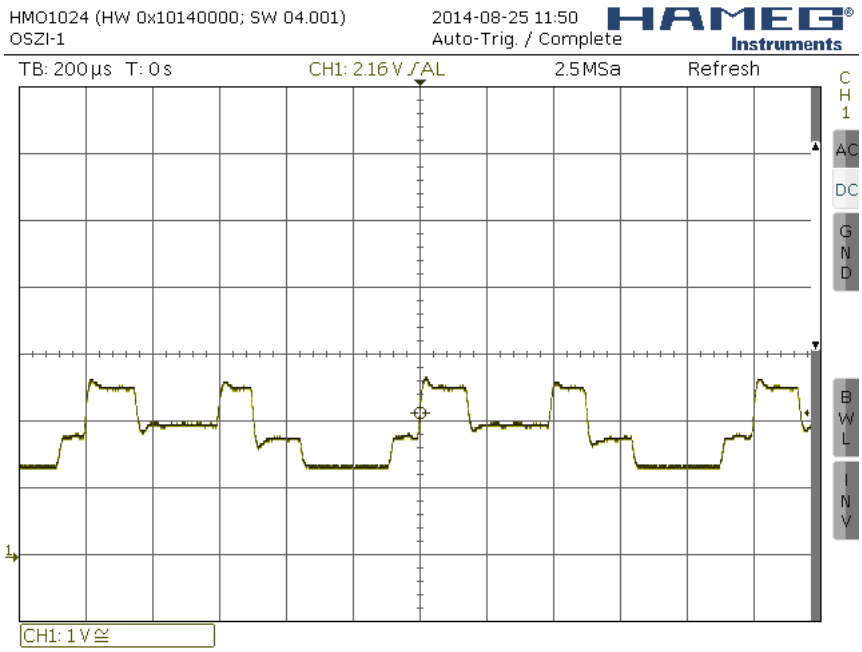

Abbildung 1.1.: Zeitliche Überlagerung von zwei PWM-Frequenzen

Ziel der Arbeit ist die Rückgewinnung der Informationstragen Frequenzen von den pulsweitenmodulierten Lichtquellen.

Es werden als erstes die Eigenschaften einer PWM untersucht um Verfahren für die Auswahl der PWM-Frequenzen zu entwickeln. Anschließend wird für eine Simulation ein Testsignal mit den ausgewählten Frequenzen entworfen. Unter Nutzung des Testsignals wird geprüft, ob oder mit welchen Signalverarbeitungsverfahren eine Frequenzrückgewinnung möglich ist. Auf die Eigenschaften von Licht, sowie auf eine weiterführende Datenverarbeitung wird in der Arbeit nicht eingegangen.

2. Grundlagen

In diesem Kapitel werden die Grundlagen für das Verständnis der Arbeit erläutert. Es wird dabei auf Komponenten der Signalverarbeitung und Signalanalyse eingegangen. Die Pulsweitenmodulation wird vertieft behandelt.

2.1. Analog-Digital Wandler

Ein Analog-Digital Wandler (A/D-Wandler) dient der Digitalisierung analoger Signale. Er hat die Funktion analoge Signale zeitlich abzutasten und diese zu quantisieren. Die zeitliche Abtastung erfolgt in einem festgelegten Intervall. Dieses darf das Nyquist-Shannon-Abtasttheorem[1] Gleichung 2.1 nicht verletzen. Da sonst eine Rekonstruktion des Signals nicht möglich ist. Die Auflösung des A/D-Wandlers ist an die Anzahl der Quantierungsstufen gebunden. Diese sind aufgrund der Digitaltechnik immer 2^N Werte. Um höherfrequente Signalanteile und somit Aliasing-Effekte vorzubeugen gibt es zwei Möglichkeiten. Bei der ersten Möglichkeit, wird das Eingangssignal vor dem A/D-Wandler mittels eines Tiefpasses gefiltert. Die Zweite Möglichkeit ist es, das Eingangsignal mit einer vielfach höheren Abtastrate zu erfassen.

$$f_{sam} > 2 \cdot f_{max} \tag{2.1}$$

Sollte die Abtastfrequenz f_{sam} kleiner oder gleich dem Zweifachen der maximalen Frequenz f_{max} sein, so ist eine zuverlässige Rekonstruktion des Signals nicht möglich. Diese Fälle sind in Abbildung 2.1 dargestellt.

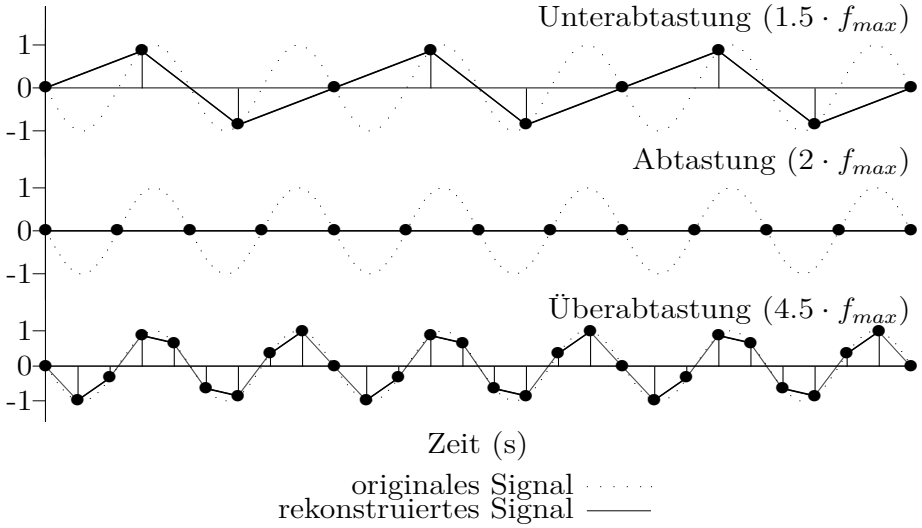

Abbildung 2.1.: Rekonstruktion eines abgetasteten Signals

[1] Benannt nach Physiker Harry Nyquist (*1889 - †1976) und Mathematiker Claude Elwood Shannon (*1916 - †2001).

2.2. Digitale Filter

Bei digitalen Filtern handelt es sich um Filter, welche auf mathematische Operationen beruhen und einen digitalen Datenstrom bearbeiten sollen. Anfangs wurden sie nur für die Simulation verwendet. Durch immer schnellere Recheneinheiten haben sie ihre praktische Anwendung in vielen Bereichen gefunden. Nicht nur als Ersatz für analoge Filter sondern auch für Realisierungen von Filtern, welche mit analogen Verfahren nur schwer oder gar nicht möglich sind. Sie weisen auch nicht die für die Analogtechnik typischen Probleme auf, welche analoge Filter für Massenproduktion aufwendig machen. Denn bei ihnen entfällt, weil keine Schwankungen der Filterparameter durch Bauteiltoleranzen, Temperaturschwankungen oder Alterserscheinungen entstehen, der manuelle Abgleich. Der Nachteil eines digitalen Filters ist die durch die Rechenoperationen bedingte Laufzeit des Filters, sowie die dadurch entstehende relativ niedrige Frequenzobergrenze. Digitale Filter können grundlegend in zwei Unterarten unterteilt werden, in nicht-rekursive und rekursive Filter.

2.2.1. Nicht-rekursive Filter

Nicht-rekursive Filter, führen das Ausgangssignal nicht zum Eingang zurück. Deshalb können diese Filter nicht schwingen und sind immer stabil. Durch ihre endliche Impulsantwort, werden sie im englischen Finite Impulse Response oder kurz FIR-Filter genannt. Sie haben einen höheren Implementierungsaufwand, aufgrund dessen sie für manche Filterungen eine höhere Filterordnung benötigen, als rekursive Filter. Die Systemlaufzeit eines FIR Filter ist proportional zur Filterordnung, welche gleich der Anzahl der Filterkoeffizienten ist. Sie werden oft für adaptive Systeme verwendet. Die Abbildung 2.2 zeigt einen digitalen Filter in kanonischer Normalform, welche mit Gleichung 2.2 mathematisch beschrieben wird. z^{-1} stellt ein Verzögerungsglied dar, welche die Weitergabe des aktuellen Arbeitswertes um einen Takt verzögert. Es dient somit als Buffer für die Eingangswerte $x[n]$, diese werden mit den Filterkoeffizienten b_m multipliziert und bilden eine Summe, die pro Takt einen neuen Ausgangswert $y[n]$ berechnet. Die allgemeine mathematische Form für FIR Filter wird mit Gleichung 2.3 beschrieben.

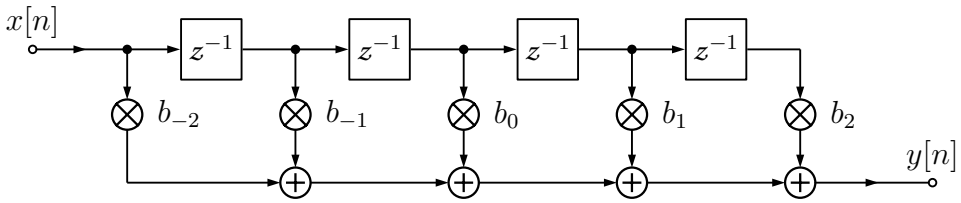

Abbildung 2.2.: Digitaler FIR Filter in der ersten kanonischen Normalform

$$y[n] = b_{-2} \cdot x[n+2] + b_{-1} \cdot x[n+1] + b_0 \cdot x[n] + b_2 \cdot x[n-1] + b_2 \cdot x[n-2] \quad (2.2)$$

$$y[n] = \sum_{m=-N}^{N} b_m \cdot x[n-m] \quad (2.3)$$

2.2.2. Rekursive Filter

Rekursive Filter, im engl. Infinite Impulse Response (IIR), sind digitale Filter, bei denen das Ausgangssignal zum Eingang zurückgeführt wird. IIR Filter sind sowohl für die Simulation von analogen Schaltungen, als auch für Anwendungen mit geringer Systemlaufzeit wichtig, da sie weniger Filterkoeffizienten als ein vergleichbarer FIR Filter benötigen. Aufgrund der Tatsache, dass ihr Phasengang meist nicht linear ist, werden sie da verwendet, wo Phaseninformationen nicht von Bedeutung sind. Aufgrund der Ausgangsrückführung können sie schwingen und instabil werden. Die kanonische Normalform eines IIR Filters ist in Abbildung 2.3 grafisch dargestellt und kann durch Gleichung 2.4 grundlegend mathematisch beschrieben werden. Durch die Rückführung entstehen zusätzliche Filterkoeffizienten a_m, welche mit den Ausgangswerten $y[n]$ multipliziert werden. Diese bilden pro Takt eine Summe, diese wird von der Summe der Filterkoeffizienten b_m mit den Eingangswerten $x[n]$ subtrahiert.

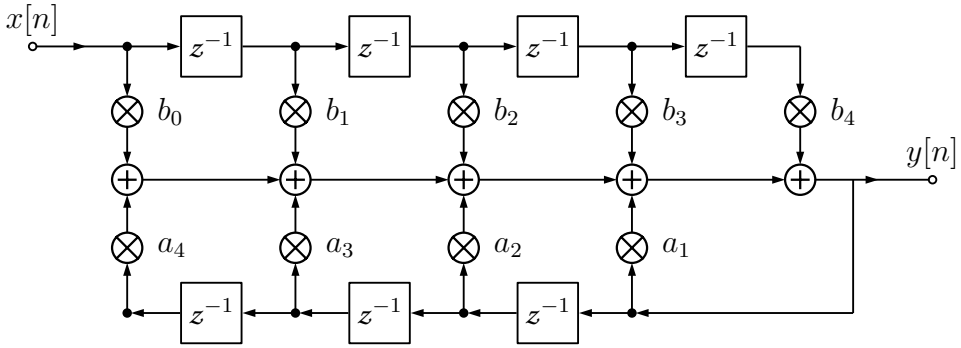

Abbildung 2.3.: Digitaler IIR Filter in der ersten kanonischen Normalform

$$y[n] = \sum_{m=-N}^{N} b_m \cdot x[n-m] - \sum_{m=1}^{N} a_m \cdot y[n-m] \quad (2.4)$$

2.3. Fourieranalyse

Fourieranalyse ist die Theorie der Fourierreihen und der Fouriertransformation. Sie wurde nach dem Mathematiker Jean Baptiste Joseph Fourier[1] benannt und findet bis heute in vielen Gebieten der Technik und Wissenschaft Anwendung. Im Folgenden wird auf die Fourierreihe und die Fouriertransformation, sowie für diskrete Signale auf die Diskrete Fouriertransformation eingegangen.

2.3.1. Fourierreihen

Nach Fourier kann jede mathematisch beschreibbare periodische Funktion als eine Überlagerung von Sinus- und Kosinusschwingungen betrachtet werden. Eine solche Funktion wird als Fourierreihe bezeichnet. Die allgemeine Fourierreihe lässt sich in drei Formen darstellen, in Trigonometrischer Form Gleichung 2.5, Amplituden-Phasen-Form Gleichung 2.6 und in der Exponential-Form Gleichung 2.7.

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cdot \cos(n\omega t) + b_n \cdot \sin(n\omega t)) \tag{2.5}$$

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(A_n \cdot \cos(n\omega t + \varphi_n)) \tag{2.6}$$

$$f(t) = \frac{a_0}{2} + \sum_{n=-\infty}^{\infty}(c_n \cdot e^{jn\omega t}) \tag{2.7}$$

Mit diesen Reihendarstellungen als Grundlage lassen sich periodische Funktionen, wie z. B. Sägezahnschwingungen oder Rechteckfunktionen beschreiben. Für letzteres wird die Reihe einer asymmetrischen Rechteckfunktion in Gleichung 2.8 und in Abbildung 2.4 gezeigt.

$$f(t) = A \cdot \frac{4}{\pi} \cdot \left(\sin(\omega t) + \frac{1}{3}\sin(3\omega t) + \frac{1}{5}\sin(5\omega t)....\right) \tag{2.8}$$

[1] Jean Baptiste Joseph Fourier (*1768 - † 1830) war ein französischer Mathematiker und Physiker

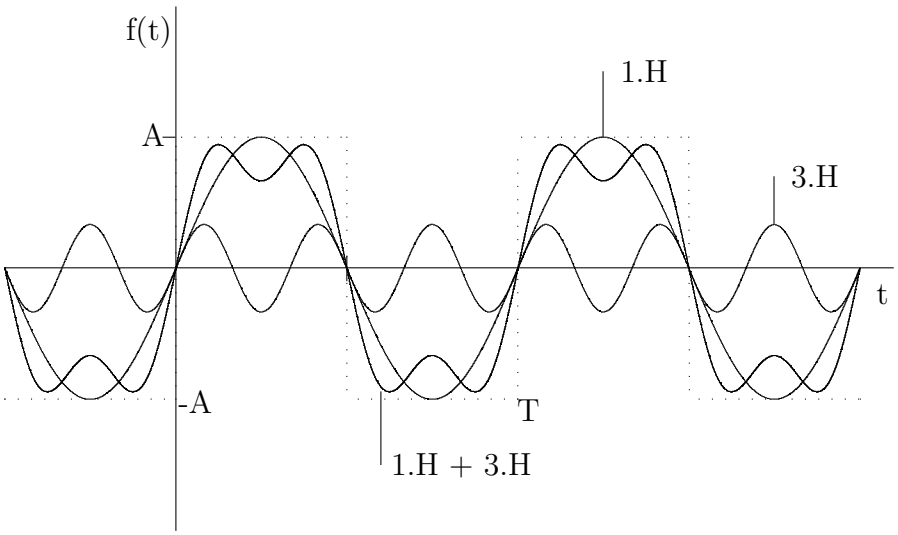

Abbildung 2.4.: Harmonische Schwingungen einer Rechteckfunktion

2.3.2. Fouriertransformation

Nicht alle Signale bzw. Signaleverläufe sind periodisch, somit auch nicht mittels Fourierreihen beschreibbar. Aperiodische Signale, wie Nachrichtensignale können mit Hilfe der Fouriertransformation in ein kontinuierliches Spektrum zerlegt werden. Das Signal wird, dadurch vom Zeitbereich in den Frequenzbereich transformiert. Die inverse Fouriertransformation macht die Transformation eines Spektrums vom Frequenzbereich in den Zeitbereich möglich. Die Fouriertransformation ist in Gleichung 2.9 und ihre inverse in Gleichung 2.10 dargestellt. Wobei $f(t)$ die Funktion der Gleichung im Zeitbereich und $F(f)$ den Frequenzbereich beschreibt.

$$F(f) = \int_{-\infty}^{\infty} f(t) \cdot e^{-j2\pi ft} \, \mathrm{d}t \tag{2.9}$$

$$f(t) = \int_{-\infty}^{\infty} F(f) \cdot e^{j2\pi ft} \, \mathrm{d}f, \tag{2.10}$$

2.3.3. Diskrete Fouriertransformation

Anders als bei der Fouriertransformation im Abschnitt 2.3.2, transformiert die diskrete Fouriertransformation (DFT) ein endliches zeitdiskretes Signal, welches sich als Zahlenfolge in Abhängigkeit des Abtastintervalls widerspiegelt. Die Gleichung für die DFT 2.11 und ihrer inversen 2.12 lassen sich, wie in [Loc97, S. 268ff.] aufgeführt, aus den Gleichungen in Abschnitt 2.3.2 mathematisch herleiten.

$$F_d(k) = \sum_{n=0}^{N-1} f(n) \cdot e^{-j2\pi kn/N} \; ; \; k, n \in \mathbb{N} \tag{2.11}$$

$$f_d(n) = \frac{1}{N} \sum_{k=0}^{N-1} F(k) \cdot e^{+j2\pi kn/N} \; ; \; k, n \in \mathbb{N} \tag{2.12}$$

Die Gleichung 2.13 stellt den Zusammenhang zwischen der Anzahl der Abtastwerte N, der Abtastfrequenz f_{sam} und der Auflösung des Spektrums f_{res} dar. Sie eignet sich zur Vorabprüfung, so lässt sich, beispielsweise bei einer festen Abtastfrequenz, prüfen wie viel Abtastwerte nötig sind, damit die Auflösung des Spektrums ausreicht. Aufgrund der diskreten Werte, kann keine eindeutige Aussage über die Frequenzen zwischen den Spektralkomponenten getroffen werden.

$$N = \frac{f_{sam}}{f_{res}} \tag{2.13}$$

2.3.3.1. Schnelle Fouriertransformation

Die schnelle Fouriertransformation (FFT) ist ein Algorithmus, der genutzt wird um die Berechnung der DFT effizienter und somit schneller durchführen zu können. Es existieren verschiedene Algorithmen zur FFT, welche sowohl softwaretechnisch, als auch als fertige Hardwareelemente verwendet werden. Letzteres hat den Vorteil der Hardwarebeschleunigung, diese begründet sich mit der Möglichkeit der parallelen Abarbeitung einzelner Schritte. Der wohl bekannteste Algorithmus ist von James Cooley und John W. Tukey[1] und wurde 1965 veröffentlicht. Die Grundidee des Algorithmus ist es die DFT in zwei N/2 Mengen, in gerade und ungerade Teilkomponenten, zu teilen. Nach der DFT-Berechnung der jeweiligen Menge, werden diese jeweils über Kreuz miteinander addiert. Dargestellt ist dies in Abbildung 2.5. Für die Berechnung einer FFT werden immer 2^N Werte benötigt, da sonst eine Berechnung nicht möglich ist. [Krü02, S. 40ff.] und [Kam+98, S. 232ff.] befassen sich mit dem Thema etwas ausführlicher, letztere greift vergleichend andere Algorithmen mit ihren Vor- und Nachteilen auf.

[1] James W. Cooley (*1926) US-amerikanischer Mathematiker
John W. Tukey (*1915 - † 2000) US-amerikanischer Statistiker

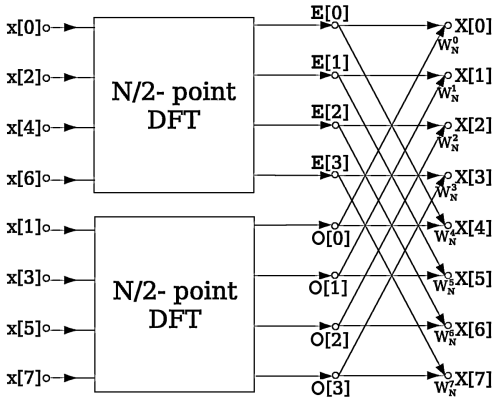

Abbildung 2.5.: Butterfly Diagramm radix-2[1]

2.4. Pulsweitenmodulation

Bei der Pulsweitenmodulation (PWM) handelt es sich um ein Verfahren, bei dem wie in Abbildung 2.6 dargestellt, die Impulsdauer von Rechteckimpulsen moduliert wird. Der Quotient von Impulsdauer τ und der Periodendauer T ist allgemein als Tastgrad, Gleichung 2.17, bezeichnet. Für die mathematische Beschreibung gibt es mehrere Möglichkeiten. Es empfiehlt sich aufgrund ihres periodischen Verhaltens und den enthaltenen Informationen im Zeit- und Frequenzbereich die Beschreibung als Fourierreihe Gleichung 2.14 [KS10, S. 287]. Die Gleichungen in 2.15 sind aus Gleichung 2.14 abgeleitet, mit ihnen lassen sich aussagen über das Amplituden- und Phasenspektrum sowie den Gleichanteil treffen.

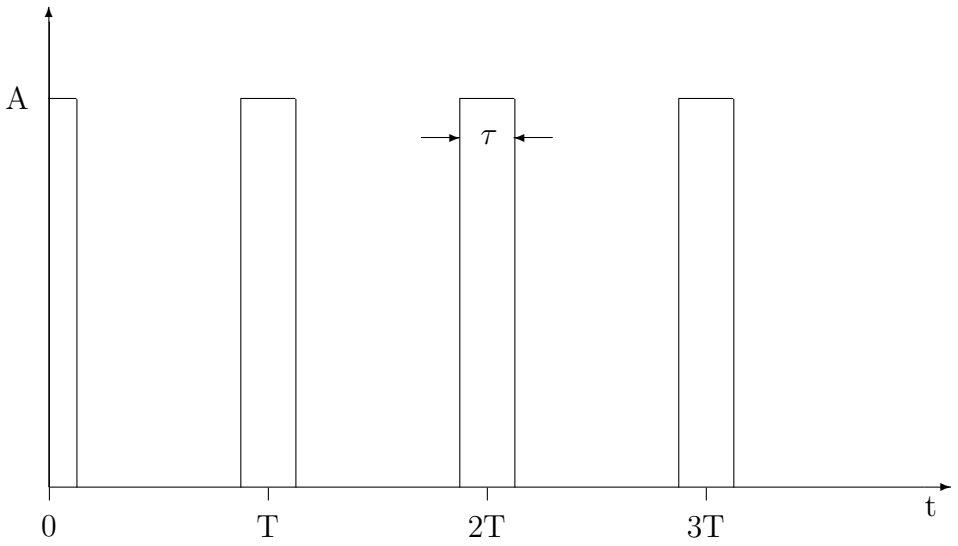

Abbildung 2.6.: Rechteckimpulse mit Tastgrad

[1] Quelle: http://en.wikipedia.org/wiki/Butterfly_diagram

$$f(t) = \underbrace{A \cdot \frac{\tau}{T}}_{\textbf{Gleichanteil}} + \overbrace{A \cdot \frac{2}{\pi} \cdot \sum_{x=1}^{\infty} \left(\frac{1}{x} \cdot \sin(x\pi\frac{\tau}{T}) \cdot \cos(x(\omega t \underbrace{+ \varphi_x}_{\textbf{Phasen}}))\right)}^{\textbf{Amplituden}} \quad ; \quad x \in \mathbb{N} \tag{2.14}$$

Gleichanteil: $$A_0 = A \cdot \frac{\tau}{T} \tag{2.15a}$$

Amplitudenspektrum: $$A_x = A \cdot \frac{2}{\pi x} \cdot \sin(x\pi\frac{\tau}{T}) \tag{2.15b}$$

Phasenspektrum: $$\varphi_x = x \cdot \varphi \tag{2.15c}$$

Kreisfrequenz: $$\omega_x = x \cdot \omega \quad ; \quad \omega = 2\pi \cdot f \tag{2.15d}$$

Gleichung 2.16 beschreibt das Verhalten der Frequenzen der harmonischen Schwingungen zur Grundfrequenz. Die Gleichung erhält man durch die Division der Gleichung 2.15d durch 2π.

$$f_x = x \cdot f \quad ; \quad x \in \mathbb{N} \tag{2.16}$$

2.4.1. Tastgrad

Im Zeitbereich ist der Tastgrad, Gleichung 2.17, der Faktor, der aus einem Rechtecksignal eine PWM macht. Betrachtet man ihn als einen variablen Wert so kann er relative Werte von 0 bis 100% annehmen. Aus den Gleichungen 2.15 ist zu entnehmen, dass sich der Tastgrad im Frequenzbereich auf den Gleichanteil sowie auf die Amplituden der Harmonischen Schwingungen auswirkt. Es existieren zwei Sonderfälle $\tau = T$ und $\tau = T/2$. Im Fall $\tau = T$ löschen sich alle Wechselanteile aus, übrig bleibt lediglich der Gleichanteil A_0. Beim Fall $\tau = T/2$ ergibt sich ein Rechtecksignal, bei der jede zweite harmonische Schwingung einer Amplitude von Null entspricht. Dies lässt sich damit begründen, dass der Inhalt der Sinusfunktion der Gleichung 2.15b für die Berechnung der Amplitude, bei diesen Harmonischen, immer ein vielfaches von π bildet, womit die Funktion zu Null wird. In Abbildung 2.7 sieht man den Amplitudenverlauf der ersten bis vierten, sowie der 20. Harmonischen Schwingung, in Abhängigkeit des Tastgrades. Die Anzahl der Halbwellen ist Äquivalent zur Ordnung des harmonischen Reihenglieds.

$$\text{Tastgrad} = \frac{\tau}{T} \tag{2.17}$$

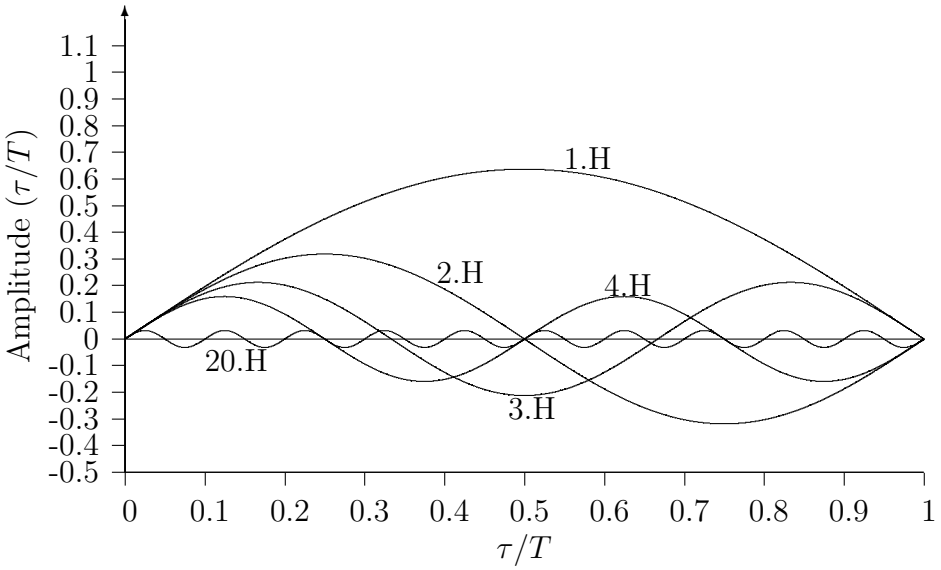

Abbildung 2.7.: Amplitudenverlauf harmonischer Reihenglieder (H) zum Tastgrad

Des Weiteren ist ersichtlich, dass die Amplituden der Harmonischen zueinander in einem Verhältnis stehen. Um dieses zu erhalten, substituiert man nun die Sinusfunktion mit dem Wert "1", womit man Gleichung 2.18 erhält .

$$\begin{aligned} A_x &= A \cdot \frac{2}{\pi x} \cdot \overbrace{\sin(x\pi\frac{\tau}{T})}^{=1} \\ &\Downarrow \\ A_x &= (A \cdot \frac{2}{\pi}) \cdot x^{-1} \end{aligned} \tag{2.18}$$

Gleichung 2.18 kommt zum Tragen, wenn das zur Übertragung genutzte Medium mehr als ein PWM-Signal beinhaltet. Gleichung 2.18 kann genutzt werden um das nötige Amplitudenverhältnis bei einer Überlagerung mit einem anderen Signal und einer Harmonischen des PWM-Signals zu bestimmen. Dabei ist es unwichtig, ob es sich bei dem anderen Signal um ein Sinussignal oder ein PWM-Signal handelt. Für die Erkennung des neuen Signals B_{neu}, muss dieses eine höhere Amplitude als die Harmonische des PWM-Signals haben. Denn bei gleicher und niedrigerer Amplitude, besteht die Gefahr der Auslöschung des Signals. Gleichung 2.19 stellt das nötige Verhältnis dar.

$$\begin{aligned} B_{neu} &> A_x \\ B_{neu} &> (A \cdot \frac{2}{\pi}) \cdot x^{-1} \end{aligned} \tag{2.19}$$

2.4.2. Abtastung eines PWM Signals

Bei der Abtastung eines PWM-Signals, kann es trotz scheinbar erfülltem Nyquist-Shannon-Abtasttheorem, Gleichung 2.1, zu einer Unterabtastung kommen. (Abbildung 2.8)

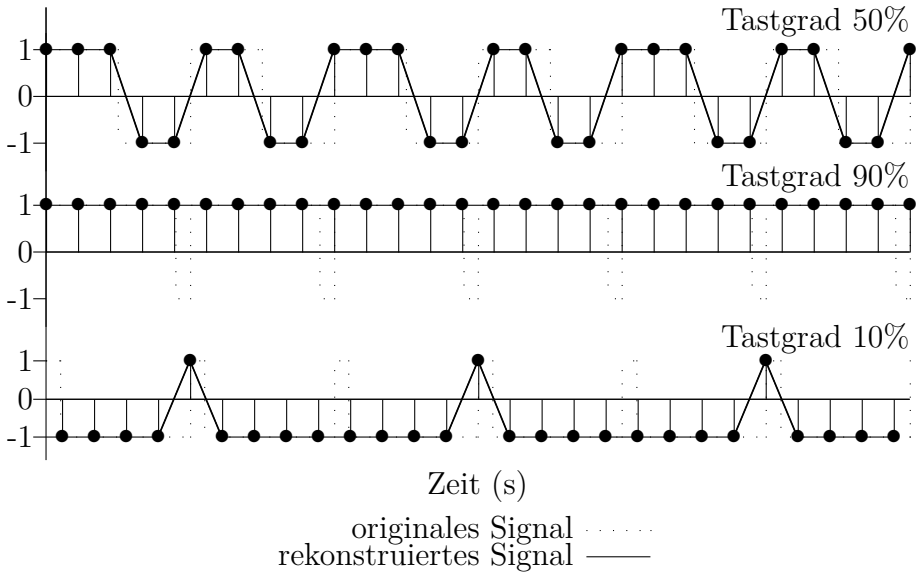

Abbildung 2.8.: Rekonstruktion eines abgetasteten PWM-Signals[1]

Der Grund dafür ist die Periodendauer der PWM die oft als PWM-Frequenz f_{pwm} bezeichnet wird. Sie beschreibt jedoch nur die erste harmonische Schwingung und damit dessen Frequenz, die Grundfrequenz der Reihe. Diese ist allerdings nicht ihre maximale Frequenz. Wenn die PWM über eine harmonische Reihe erstellt wurde, könnte man die maximale Frequenz über die Grundfrequenz und dem letzten Glied der harmonischen Reihe mit Gleichung 2.16 ermitteln. Praktisch ist das jedoch selten der Fall. Des Weiteren lässt sich die nötige Abtastfrequenz für eine Rekonstruktion über die Impulsbreite der PWM bestimmen. Dies geschieht in den Gleichungen 2.20 und 2.21. Die Abtastfrequenz wird hier nach dem existierenden Extremwert des Tastgrades und der PWM-Frequenz bestimmt. Dabei ist Gleichung 2.20 für alle Tastgrade als kleiner Gleich 50% und Gleichung 2.21 für alle Tastgrade größer als 50% definiert.

$$f_{sam} > \frac{1}{Tastgrad} \cdot f_{pwm} \qquad ; Tastgrad \leq 50\% \tag{2.20}$$

$$f_{sam} > \frac{1}{(1 - Tastgrad)} \cdot f_{pwm} \qquad ; Tastgrad \geq 50\% \tag{2.21}$$

[1] Abtastfrequenz beträgt das 4.5 fache der PWM-Frequenz

3. Theorie

Dieses Kapitel dient der Auswahl der PWM-Frequenzen. Dabei soll es möglich sein, die Grundfrequenz der PWM zu erkennen, welche den Informationsträger darstellt. Eine freie Wahl eignet sich aufgrund der Oberwellen[1] einer PWM nicht. Es handelt sich bei ihnen um vermeidbare Störquellen im System. Für die Vermeidung dieser, wird die Auswahl der PWM-Frequenzen, in Abhängigkeit von Reihenkonstrukten gesetzt und ihr Verhalten gegenüber ihrer Harmonischen betrachtet. Die daraus entstandenen Bedingungen und Gleichungen sind den folgenden Abschnitten zu entnehmen.

3.1. Geometrische Reihe

Mithilfe einer geometrischen Reihe soll eine Auswahl von PWM-Frequenzen getroffen werden. Diese benötigt, gemäß [Pap09, S. 16], einen Startpunkt und einen Multiplikator. Die Berechnung der Reihenglieder erfolgt mit Gleichung 3.1. Diese besteht aus einem Startpunkt, auch Startfrequenz f_0 genannt und dem Multiplikator k, dessen Potenz i die Nummer des Reihengliedes entspricht.

$$f_i = k^i \cdot f_0 \quad ; \quad i \in \mathbb{N} \tag{3.1}$$

Wie bereits aus Abschnitt 2.4 bekannt ist, besteht eine PWM im Frequenzbereich aus einer Reihe von harmonischer Schwingungen. Aus diesem Grund kann nicht jede beliebige geometrische Reihe für die Auswahl der Frequenzen angenommen werden. Die Reihe hängt von Randbedingungen ab, die den Zusammenhang der Reihenglieder zu ihren Harmonische definieren sollen. Für das Zusammenspiel der harmonischen Frequenzen der geometrischen Reihe kommen mehrere Szenarien in Frage, die im folgenden einzeln behandelt werden.

Wenn keine Oberwellen zwischen oder auf den Gliedern der geometrischen Reihe liegen sollen, gilt der im Folgenden beschriebene mathematische Zusammenhang. Es werden die Gleichungen 3.1 und 2.16 in ein Verhältnis gesetzt. Für letztere ist die Frequenz des ersten Glieds der geometrischen Reihe f als Frequenz der Grundwelle der Harmonischen f zu setzen ($f = f_0$). Mathematisch wird dies mit Gleichung 3.2 beschrieben, wobei i für das letzte Glied der Reihe $i = N - 1$ steht und $x = 2$ für die 2. harmonische Schwingung des ersten Gliedes der Reihe. Dies wird nach k umgeformt um Gleichung 3.3 zu erhalten.

[1] 1. Harmonische Schwingung = Grundwelle
2. Harmonische Schwingung = 1. Oberwelle
3. Harmonische Schwingung = 2. Oberwelle usw.

$$\begin{array}{c} f_i < f_x \\ \downarrow \quad \downarrow \\ k^i \cdot f_0 < x \cdot f_0 \end{array} \tag{3.2}$$

$$k < \sqrt[i]{x} \;\; ; \;\; i = N - 1 \; ; x = 2 \tag{3.3}$$

Im nächsten Fall könnte man die Harmonischen der Reihe zwischen oder auf den Reihengliedern der geometrischen Reihe fallen lassen. Beschränkt man dies zunächst nur auf die harmonische Reihe des ersten Gliedes f_0 der geometrischen Reihe entstehen zwei Bedingungen. Die erste Bedingung ist mit Gleichung 3.4 definiert und stellt das Verhalten der x-ten Harmonischen des ersten Gliedes dar. Diese soll zwischen dem i-ten Glied und seinem Vorgänger liegen. Durch umformen der Gleichung 3.4 nach k entsteht Gleichung 3.5.

$$\begin{array}{c} f_i > \quad f_x \quad > f_{i-1} \\ \downarrow \qquad \downarrow \qquad \downarrow \\ k^i \cdot f_0 > x \cdot f_0 > k^{(i-1)} \cdot f_0 \end{array} \tag{3.4}$$

$$\sqrt[i]{x} < k < \sqrt[i-1]{x} \tag{3.5}$$

Die nächste Bedingung ist der Fall, dass sich eine x-te Harmonische des ersten Gliedes mit einem Folgeglied i der geometrischen Reihe überlagert. Die Berechnung des notwendigen Multiplikators k für diese Bedingung, erfolgt mit Gleichung 3.6.

$$k = \sqrt[i]{x} \tag{3.6}$$

Nach Aufhebung der genannten Beschränkung erhält man den letzten Fall. Bei diesem können die Harmonischen der Reihenglieder zwischen oder auf die Folgeglieder liegen. Gleichung 3.7 besagt, dass die x-te Harmonische des m-ten Gliedes der geometrischen Reihe zwischen ein Folgeglied i und dessen Vorgänger $i - 1$ fällt. Nach der Umformung der Gleichung nach k, entsteht Gleichung 3.8.

$$\begin{array}{ccccc} f_i > & x \cdot f_m & > f_{i-1} & ; i, m \in \mathbb{N}; i > m \\ \downarrow & \downarrow & \downarrow & \\ k^i \cdot f_0 > & x \cdot (m f_0) & > k^{(i-1)} \cdot f_0 & \end{array} \tag{3.7}$$

$$\sqrt[i-m]{x} < k < \sqrt[i-m-1]{x} \quad ; i, m \in \mathbb{N}; i > m \tag{3.8}$$

Bei der nächsten Bedingung überlagern sich, die x-te Harmonische eines m-ten Gliedes mit einem folge Glied i der geometrische Reihe. Die Berechnung des notwendigen Multiplikators k für diese Bedingung, erfolgt mit Gleichung 3.9.

$$k = \sqrt[i-m]{x} \tag{3.9}$$

3.1.1. Reihenbeispiele: geometrische Reihe

Mithilfe der Gleichungen aus Abschnitt 3.1 wurden exemplarisch zwei normierte Reihen mit Startpunkt gleich Eins erstellt. Die Berechnung der notwendigen Multiplikatoren und der Reihe selbst erfolgte in Excel, die erstellte Datei ist im Anhang A.1 zu finden. Die Reihen wurden auf acht Glieder begrenzt. Dadurch wurde für die erste Reihe die mit Gleichung 3.3 und deren Bedingungen entstand, errechnet, dass der Multiplikator k kleiner als $1,104$ sein muss. Es wurde sich für den Multiplikator $k = 1,1091$ entschieden, dieser hat einen prozentualen Abstand zur 2. Harmonischen des ersten Gliedes der Reihe von 8%. Dies ist aus Tabelle 3.1, mit der dargestellten Reihe und ihren ersten Oberwellen ersichtlich.

Tabelle 3.1.: Normierte geometrische Reihe ohne Überschneidung

	Reihenglieder							
H.	0	1	2	3	4	5	6	7
1	1	1,091	1,190	1,299	1,417	1,546	1,686	1,840
2	2	2,182	2,381	2,597	2,834	3,091	3,373	3,680

Bei der zweiten Reihe soll sich die Reihe mit ihren Harmonischen überschneiden. Die Überschneidung soll zwischen den 4. und 5. Glied beginnen, dafür muss der Multiplikator nach Gleichung 3.5 zwischen 1,148 und 1,189 betragen. Die Tabelle 3.2 wurde mit dem Multiplikator $k = 1,165$ erstellt, dieser liegt im Rahmen der Parameter.

Tabelle 3.2.: Normierte geometrische Reihe mit Überschneidung

H.	Reihenglieder 0	1	2	3	4	5	6	7
1	1	1,165	1,357	1,581	1,842	2,146	2,500	2,913
2	2	2,33	2,714	3,162	3,684	4,292	5,000	5,825
3	3	3,495	4,072	4,744	5,526	6,438	7,500	8,738
4	4	4,66	5,429	6,325	7,368	8,584	10,000	11,650

3.2. Arithmetische Reihe

Die arithmetische Reihe soll wie die geometrische dem Zweck dienen eine Reihe von PWM-Frequenzen zu erzeugen. Die Differenz d zwischen zwei Folgeglieder der Reihe, wird aus dem Startpunkt –Startfrequenz– f_0 und einen Multiplikator k gemäß Gleichung 3.10 gebildet. Dies geschieht, damit die Lage der harmonischen der arithmetischen Reihe zu einander einfacher bestimmt werden kann. Das hat zu Folge, dass der relative Abstand der Reihenglieder bei Veränderung des Startpunktes und gleichbleibenden Multiplikator gleich ist. Der absolute Abstand, ist abhängig vom Startpunkt und Multiplikator. Die Reihenglieder werden, gemäß [Pap09, S. 16], mit Gleichung 3.11 berechnet. Dabei wird der Summand für die Addition aus der Differenz d und dem Reihenglied i gebildet.

$$d = k \cdot f_0 \tag{3.10}$$

$$f_i = (i \cdot d) + f_0 \tag{3.11}$$

Für den Zusammenhang zwischen der arithmetischen Reihe und ihren harmonischen kommen, wie in Abschnitt 3.1, verschiedene Szenarien in Frage. Diese werden im folgenden einzeln behandelt. Im ersten Szenario sollen sich keine Oberwellen zwischen oder auf den Gliedern der arithmetischen Reihe befinden. Dies wird erreicht, wenn man Gleichung 3.11 und Gleichung 2.16 in ein Verhältnis setzt. Für die letzte Gleichung wird $f = f_0$ gesetzt, wobei f_0 der Grundfrequenz des ersten Gliedes der arithmetischen Reihe entspricht. Mathematisch wird dieser Zusammenhang mit Gleichung 3.12 beschrieben. Dabei steht i für das letzte Glied der Reihe $i = N - 1$, wobei N die Anzahl der Glieder angibt. Die x-te Harmonische des ersten Gliedes, wird mit $x = 2$ für die 2. harmonische Schwingung angegeben. Nach Umformung nach k erhält man Gleichung 3.13.

$$\begin{aligned} f_i &< f_x \\ (i \cdot d) + f_0 &< x \cdot f_0 \\ (i(k \cdot f_0)) + f_0 &< x \cdot f_0 \end{aligned} \tag{3.12}$$

$$k < \frac{x-1}{i} \;\; ; \;\; i = N - 1 \; ; x = 2 \tag{3.13}$$

Im zweiten Szenario, wird das Verhalten der Harmonischen des ersten Gliedes f_0 der arithmetischen Reihe zu seinen Folgegliedern beschrieben. Diese sollen zwischen einem Folgeglied i und seinem Vorgänger $i-1$ liegen. Mathematisch wird dies mit Gleichung 3.14 beschrieben. Diese wird nach k umgeformt um Gleichung 3.15 zu erhalten.

$$\begin{array}{ccccc} f_i & > & f_x & > & f_{i-1} \\ \downarrow & & \downarrow & & \downarrow \\ (i(k \cdot f_0)) + f_0 & > & x \cdot f_0 & > & ((i-1)(k \cdot f_0)) + f_0 \end{array} \tag{3.14}$$

$$\frac{x-1}{i} < k < \frac{x-1}{i-1} \tag{3.15}$$

Im letzten Szenario, wird das Verhalten, der x-ten Harmonsichen eines m-ten Gliedes der arithmetischen Reihe zu einem nachfolgenden i-ten Glied und dessen Vorgänger $i-1$ der Reihe beschrieben. Diese lassen sich unterteilen in die Möglichkeiten, dass die Harmonischen zwischen den Reihengliedern fallen oder auf denen. Der erste Fall wird mathematisch mit Gleichung 3.16 beschrieben. Formt man diese nach k um, so erhält man Gleichung 3.17.

$$\begin{array}{ccccc} f_i & > & x \cdot f_m & > & f_{i-1} \quad ; i, m \in \mathbb{N}; i > m \\ \downarrow & & \downarrow & & \downarrow \\ (i(k \cdot f_0)) + f_0 & > & x \cdot (m f_0) & > & ((i-1)(k \cdot f_0)) + f_0 \end{array} \tag{3.16}$$

$$\frac{x-1}{i - xm} < k < \frac{x-1}{(i-1) - xm} \quad ; i, m \in \mathbb{N}; i > m \tag{3.17}$$

Im nächsten Fall überlagern sich, die x-te Harmonische eines m-ten Gliedes der Reihe mit einem folge Glied i der arithmetischen Reihe. Die Berechnung des notwendigen Multiplikators k für diesen Fall, erfolgt mit Gleichung 3.18.

$$k = \frac{x-1}{i-xm} \tag{3.18}$$

3.2.1. Reihenbeispiele: arithmetische Reihe

Mithilfe der Gleichungen aus Abschnitt 3.2 wurden exemplarisch zwei normierte Reihen mit Startpunkt gleich Eins erstellt. Die Berechnung der notwendigen Multiplikatoren und der Reihe selbst erfolgte in Excel, die erstellte Datei ist im Anhang A.1 zu finden. Die Reihen wurden auf acht Glieder begrenzt. Dadurch wurde für die erste Reihe die mit Gleichung 3.13 und deren Bedingungen entstand, errechnet, dass der Multiplikator k kleiner als $0,142$ sein muss. Es wurde sich für den Multiplikator $k = 0,12$ entschieden, dieser hat einen prozentualen Abstand zur 2. Harmonischen des ersten Gliedes der Reihe von 8%. Dies ist aus Tabelle 3.3, mit der dargestellten Reihe und ihren ersten Oberwellen ersichtlich.

Tabelle 3.3.: Normierte arithmetische Reihe ohne Überschneidung

H.	Reihenglieder 0	1	2	3	4	5	6	7
1	1	1,12	1,24	1,36	1,48	1,6	1,72	1,84
2	2	2,24	2,48	2,72	2,96	3,2	3,44	3,68

Bei der zweiten Reihe soll sich die Reihe mit ihren Harmonischen überschneiden. Die Überschneidung soll zwischen den 4. und 5. Glied beginnen, dafür muss der Multiplikator nach Gleichung 3.15 zwischen 0,2 und 0,25 betragen. Die Tabelle 3.4 wurde mit dem Multiplikator $k = 0,225$ erstellt, welcher zentral zu den Parametern liegt.

Tabelle 3.4.: Normierte arithmetische Reihe mit Überschneidung

H.	Reihenglieder 0	1	2	3	4	5	6	7
1	1	1,225	1,45	1,675	1,9	2,125	2,35	2,575
2	2	2,45	2,9	3,35	3,8	4,25	4,7	5,15
3	3	3,675	4,35	5,025	5,7	6,375	7,05	7,725
4	4	4,9	5,8	6,7	7,6	8,5	9,4	10,3

3.3. DFT-Reihe

Dieses Verfahren basiert auf dem Amplitudenspektrum, welches nach einer DFT erstellt werden kann. Es handelt sich hierbei um ein diskretes Spektrum, dessen Frequenzauflösung f_{res} mit Gleichung 2.13 bestimmt wird. D. h. , dass das Spektrum auch nur diese Frequenzen darstellen und zuordnen kann.(Gleichung 3.19) Des Weiteren bestimmt N zu gleich die maximale Anzahl der Spektralkomponenten. In Abbildung 3.1 ist ein Amplitudenspektrum einer PWM schematisch dargestellt.

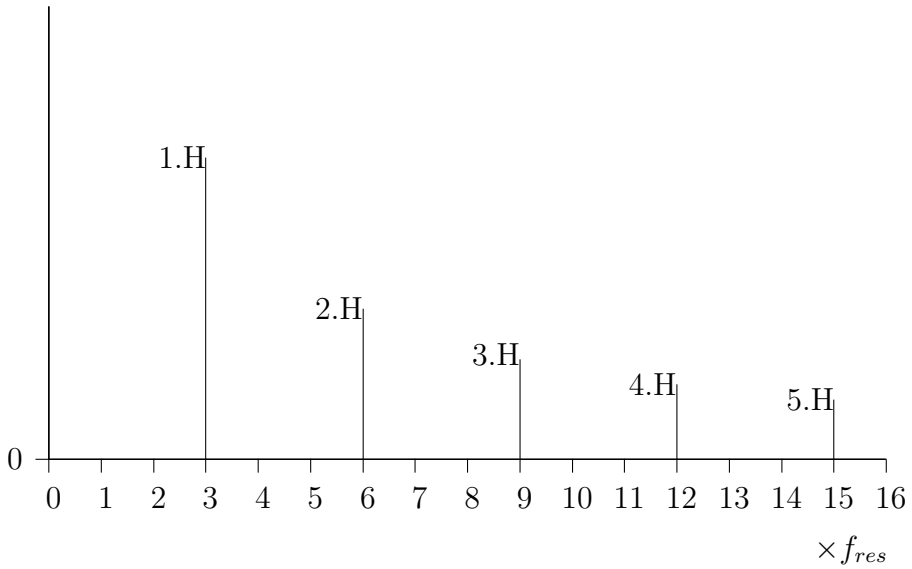

Abbildung 3.1.: Schematisch dargestelltes Spektrum einer PWM

$$f(k) = k \cdot f_{res} = k \cdot \frac{f_{sam}}{N} \tag{3.19}$$

Zu erkennen ist, dass die Frequenz der 1. harmonischen Schwingung bei drei und die 2. Harmonische bei sechs ist. Dies setzt sich bis zur 5. Harmonischen fort. Daraus lässt sich für die Bestimmung der Position der harmonischen Schwingungen in einem diskretem Spektrum die Gleichung 3.20 herleiten, wobei k_{H1} die Startposition der Harmonischen darstellt.

$$k_x = x \cdot k_{H1} \tag{3.20}$$

Aus der Startposition lässt sich die maximale Anzahl von aufeinander folgenden Frequenzen ableiten, die ohne Einfluss von harmonischen in einem System existieren können. Des Weiteren ist jede folgende Primzahl unbeeinflusst, da Primzahlen nur durch sich selbst oder Eins teilbar sind. Dadurch haben Harmonische Wellen von vorhergehenden Zahlen größer Eins keinen Einfluss auf Primzahlen. Dabei ist zu beachten, dass sich diese Aussagen auf die normierte Reihe bezieht. Das bedeutet,

dass das dargestellte Spektrum nicht mit der Frequenzauflösung multipliziert wird und somit nur natürliche Zahlen von 0 bis 16 darstellt.

3.3.1. Reihenbeispiele: DFT-Reihe

Folgend sind zwei normierte Reihen und ihre Harmonischen dargestellt. Beide Reihen haben jeweils acht Glieder, sie unterscheiden sich jedoch auf die Anzahl ihrer Folgeglieder. Die erste Reihe, welche Tabelle 3.5 dargestellt, ist fortlaufend ab $k = 8$, womit die 2. Harmonische des ersten Gliedes erst bei $k = 16$ anzutreffen ist und außerhalb der Reihe liegt.

Tabelle 3.5.: Normierte DFT Reihe

	Reihenglieder							
H.	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	18	20	22	24	26	28	30

Bei der zweiten Reihe, welche in Tabelle 3.6 dargestellt ist, handelt es sich um eine Reihenbildung der ersten acht Primzahlen und ihren Harmonischen. Mit dieser Reihe wird die Behauptung bestätigt, dass nachfolgende Primzahlen, nicht von Harmonischen vorhergehender Glieder beeinflusst werden. In Abschnitt 4.1.1 wird die Verwendung der normierten Reihen anhand eines Beispielsystems erläutert.

Tabelle 3.6.: Normierte DFT Reihe mit Primzahlen

	Reihenglieder							
H.	0	1	2	3	4	5	6	7
1	2	3	5	7	11	13	17	19
2	4	6	10	14	22	26	34	38
3	6	9	15	21	33	39	51	57
4	8	12	20	28	44	52	68	76

4. Simulation

Dieses Kapitel beschäftigt sich mit der Systemwahl und dessen Aufbau. Des Weiteren werden die Parameter für die Simulation festgelegt. Das im folgenden beschriebene System sowie die Simulation sind exemplarisch für diese Arbeit entstanden. Denn wie aus Anhang B.1 hervorgeht, würden sich auch andere Systeme simulieren lassen.

4.1. Systemwahl

Für die Simulation wird vom Systemaufbau in Abbildung 4.1 ausgegangen. Dieser stellt vier identische Sender **S** dar, die alle im gleichen Abstand **r** zu einen Empfänger **E** stehen. Dadurch trifft das Licht jedes Senders mit gleicher Intensität auf den Empfänger auf. Setzt man nun noch voraus, dass das Licht von jedem Sender ohne Störungen zu Empfangen ist, kann man auf eine Simulation des Übertragungsweges verzichten. Dadurch kann ein beliebiger Wert für die Intensität, der einzelnen Sender, am Empfänger definiert werden. Die Signale der Sender sind pulsweitenmoduliert. Der Tastgrad ist von 10% - 90% frei wählbar. Zudem werden jedem Sender zwei unterschiedliche PWM-Frequenzen zugeordnet. Es kann immer nur eine der Beiden anliegen. Somit muss der Empfänger acht verschiedene PWM-Frequenzen erkennen können. Gleichzeitig mischen sich allerdings nur vier PWM-Frequenzen zu einem Signal zusammen.

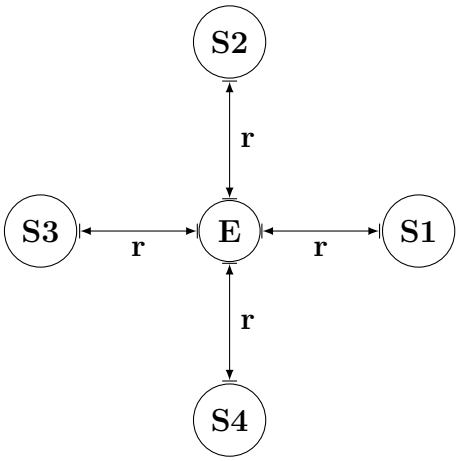

Abbildung 4.1.: Äußerer Systemaufbau

Abbildung 4.2 zeigt den schematischen Aufbau des Empfängers. Beginnend mit dem Sensor für die Lichtdetektion mittels Photodiode (PIND) und Transimpedanzwandler (TIA) bis zur Datenverarbeitung. Die Photodiode wandelt das Licht in einen elektrischen Strom um. Beim Transimpedanzwandler handelt es sich um einen Strom-Spannungswandler. Dieser wandelt den elektrischen Eingangsstrom proportional in eine Ausgangsspannung um. In der Praxis kann es zum Einen zu einer Übersteuerung des Sensors und zum Anderen zu einer bauteilbedingten Dämpfung des Signals kommen. Für die Simulation werden diese Effekte vernachlässigt und es wird von

einer proportionalen Wandlung von Licht in elektrischen Spannung angenommen. Dadurch kann der Umwandlungsprozess vernachlässigt werden. Die Simulation setzt somit nachdem Sensor an. Die weiterführende Datenverarbeitung wird nicht betrachtet, weil dies nicht Thema der Arbeit ist. Somit beschränkt sich die Simulation auf die Filterung, A/D-Wandlung und der Digitalen Signalverarbeitung (DSV). Die Filterung beschränkt den Frequenzbereich für den A/D-Wandler. Nach der Abtastung werden die Werte mittels DSV weiterverarbeitet.

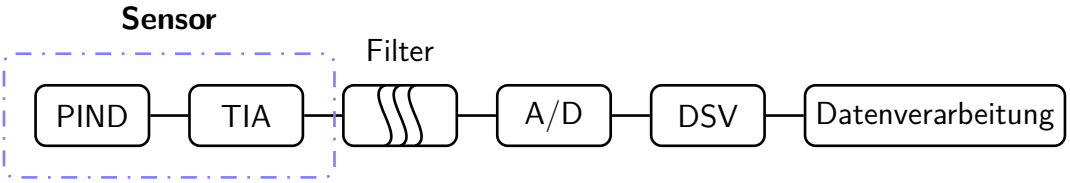

Abbildung 4.2.: Schematischer Aufbau Empfänger

Durch die rein numerische Simulation am PC, ist eine Abtastung des gefilterten analogen Signals mittels A/D-Wandler nicht möglich. Um das System trotzdem simulieren zu können, wurde ein Ersatzsystem erstellt, dieses ist in Abbildung 4.3 dargestellt.

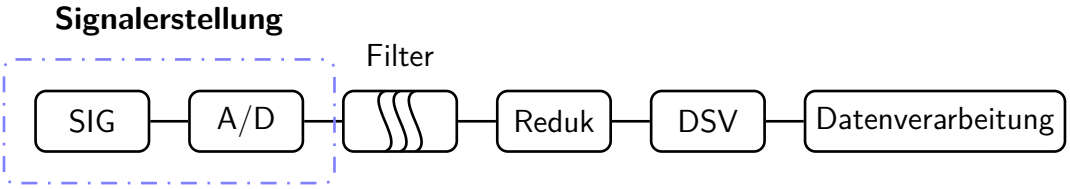

Abbildung 4.3.: Schematischer Aufbau Simulation

Es wird ein diskretes Zeitsignal erstellt, welches einem abgetasteten Signal gleicht. Bei diesem handelt es sich um ein gemischtes Signal, welches aus vier PWM-Signalen besteht. Die nötige Abtastfrequenz, für eine Rekonstruktion des Signals, kann über die Grenzfrequenz des nachfolgenden Filters oder über die höchste PWM-Frequenz und der Gleichung 2.20 bestimmt werden. Aus Sicht praktischer Anwendungen empfiehlt sich die Verwendung der Grenzfrequenz des Filters. Durch die nachfolgende Filterung soll das Frequenzspektrum begrenzt werden, womit die meisten Oberwellen des PWM-Signals heraus gefiltert werden.

Die Signalfolge die durch diese Filterung entsteht, kann durch Reduzierung der Zwischenschritte, und der Abtastfrequenz auf die Grenzfrequenz angepasst werden. Dies ist dadurch begründet, dass alle höheren Frequenzen heraus gefiltert wurden und somit im Signal nicht mehr vorhanden sind. Die Reduzierung geschieht im

Reduk-Block. Die resultierende Abtastfrequenz kann analog als Abtastfrequenz des A/D-Wandlers angesehen werden. Somit entspricht die resultierende Signalfolge den Abtastwerten des A/D-Wandlers. Zu beachten ist, dass die neue Abtastfrequenz das Nyquist-Shannon-Abtasttheorem (Gleichung 2.1) mit der oberen Grenzfrequenz des Filters nicht verletzen darf.

Der DSV-Block arbeitet mit der neuen Abtastfrequenz und der neuen Signalfolge. Diese Schritte waren, zur Simulation der Oberwellen der PWM-Signale, notwendig. Die daraus entstandene Signalfolge kann in Abtastfrequenz und Anzahl der Zwischenschritte reduziert werden.

4.1.1. Frequenzwahl

Die Frequenzen sind durch den verwendbaren Frequenzbereich eingeschränkt. Wenn kein Standard, für den Frequenzbereich definiert ist, bestimmt die Hardware und die Übertragung diesen. Im Folgenden werden die Frequenzen beispielhaft anhand von Hardwaregrenzen bestimmt. Sender und Empfänger sind hierbei so definiert, dass beide in einen Frequenzbereich von 30kHz bis 100kHz arbeiten. Somit entspricht die verfügbare Bandbreite 70kHz. Durch den Start ab 30kHz können evtl. Störungen durch andere nicht zum System gehörende Quellen vernachlässigt werden. Zu diesen Störquellen gehört z.B. die normale Zimmerbeleuchtung, der LED-Monitor oder der Fernseher. Diese Geräte arbeiten meistens mit Frequenzen unter 2kHz, wodurch sich evtl. Oberwellen als Rauschen definieren lassen. Der Übertragungsweg hat keinen Einfluss auf die Frequenzen.

Mit den definierten Bereich lassen sich nun die nötigen Frequenzen bestimmen. Zunächst werden die Grenzfrequenzen bestimmt in denen die Filterschaltung arbeiten soll. Aufgrund des niedrigen Frequenzbereich lässt sich die Filterschaltung als Tiefpassfilter darstellen. Damit liegt die obere Grenzfrequenz bei 100kHz. Eine untere Grenzfrequenz muss nicht definiert werden. Die PWM-Frequenzen basieren auf den in Kapitel 3 genannten Reihentypen und ihren Gleichungen. Es entstehen für jeden Reihentyp zwei Reihen von PWM-Frequenzen. Die bereits erstellten, normierten Reihen sind dafür die Grundlage. Die weitere Erklärung ist zunächst für die geometrische und arithmetische Reihe bestimmt.

Für die Anpassung der normierten Reihen gibt es zwei Möglichkeiten. Bei der ersten, nimmt man eine vorher definierte normierte Reihe und multipliziert diese mit der untersten Frequenz, die als Reihenglied verwendet werden soll. Ein Beispiel dafür, ist die normierte geometrische Reihe aus Tabelle 3.1, welche ohne Überschneidung definiert ist. Multipliziert man diese nun mit 30kHz so erhält man eine für den

Frequenzbereich passende Reihe. Diese ist in Tabelle 4.1 dargestellt und beschreibt eine geometrische Reihe ohne Überschneidung der Oberwellen.

Tabelle 4.1.: Geometrische Reihe ohne Überschneidung

H.	PWM Frequenzen 1	2	3	4	5	6	7	8
1	30000	32730	35708	38958	42503	46371	50591	55194
2	60000	65460	71417	77916	85006	92742	101181	110389

Die zweite Möglichkeit ist die Erstellung einer Reihe über die Gleichungen in dem man einen gewünschte Startfrequenz als Startpunkt definiert. Nimmt man z. B. die Parameter aus Tabelle 3.2 und ändert den Startpunkt auf 30kHz und lässt den Multiplikator gleich, so erhält man eine Reihe mit den selben Bedingungen und dennoch anderen Frequenzen. Die Reihe ist in Tabelle 4.2 dargestellt und beschreibt eine Reihe mit Überschneidung der Oberwellen.

Tabelle 4.2.: Geometrische Reihe mit Überschneidung

H.	PWM Frequenzen 1	2	3	4	5	6	7	8
1	30000	34950	40717	47435	55262	64380	75003	87378
2	60000	69900	81434	94870	110524	128760	150005	174756
3	90000	104850	122150	142305	165785	193140	225008	262134
4	120000	139800	162867	189740	221047	257520	300011	349513

Beide Varianten sind äquivalent zueinander und führen zum gleichen Ergebnis. Die normierten arithmetischen Reihen werden wie die geometrischen Reihen auf den Frequenzbereich angepasst. Dies ist notwendig, um die Reihentypen miteinander vergleichen zu können. Die normierten geometrischen und arithmetischen Reihen sind mit den gleichen Bedingungen erstellt worden. Die angepasste normierte Reihe aus Tabelle 3.3 ist in Tabelle 4.3 und beschreibt eine arithmetische Reihe ohne Überschneidung der Oberwellen die mit 30kHz multipliziert wurde.

Tabelle 4.3.: Arithmetische Reihe ohne Überschneidung

H.	PWM Frequenzen 1	2	3	4	5	6	7	8
1	30000	33600	37200	40800	44400	48000	51600	55200
2	60000	67200	74400	81600	88800	96000	103200	110400

Tabelle 4.4 ist mit Hilfe der Parameter, der normierten arithmetischen Reihe aus Tabelle 3.4 entstanden. Sie beschreibt eine Reihe mit Überschneidung der Oberwellen deren Startpunkt bei 30kHz liegt.

Tabelle 4.4.: Arithmetische Reihe mit Überschneidung

H.	PWM Frequenzen 1	2	3	4	5	6	7	8
1	30000	36750	43500	50250	57000	63750	70500	77250
2	60000	73500	87000	100500	114000	127500	141000	154500
3	90000	110250	130500	150750	171000	191250	211500	231750
4	120000	147000	174000	201000	228000	255000	282000	309000

Die normierten DFT-Reihen werden mit einer Frequenzauflösung in den entsprechenden Frequenzbereich geholt. Dies geschieht mit Gleichung 3.19. Dabei gilt es zu beachten, dass N groß genug ist, um die Reihe darzustellen. Damit die normierte DFT-Reihe aus Tabelle 3.5 in den genannten Frequenzbereich liegt wurde von einer Frequenzauflösung von 4kHz ausgegangen. Nach der Multiplikation der Reihe mit 4kHz entstand die resultierende Reihe in Tabelle 4.5. Diese Frequenzauflösung kann z. B. durch die Abtastfrequenz $f_{sam} = 1,024$ MHz und 256 Abtastwerten $N = 256$ entstehen.

Tabelle 4.5.: DFT Reihe fortlaufend

H.	PWM Frequenzen 1	2	3	4	5	6	7	8
1	32000	36000	40000	44000	48000	52000	56000	60000
2	64000	72000	80000	88000	96000	104000	112000	120000

Die normierte DFT-Reihe aus Tabelle 3.6 lässt sich in den genannten Frequenzbereich nicht vollständig darstellen. Deswegen wurde auf eine korrekte Darstellung verzichtet und die Frequenzauflösung wurde ebenfalls mit 4kHz festgelegt. Daraus resultiert die in Tabelle 4.6 stehende Reihe. Die relative Bandbreite, die für diese Reihe benötigt wird ist ersichtlich.

Tabelle 4.6.: DFT Reihe mit Primzahlen

H.	PWM Frequenzen 1	2	3	4	5	6	7	8
1	8000	12000	20000	28000	44000	52000	68000	76000
2	16000	24000	40000	56000	88000	104000	136000	152000
3	24000	36000	60000	84000	132000	156000	204000	228000
4	32000	48000	80000	112000	176000	208000	272000	304000

4.2. Durchführung

Die Simulation des bisher beschriebenen Systems erfolgt numerisch mit der freien Software GNU Octave. Es wurde die Version 3.8.1 mit dem Paket Signal Processing in Version 1.3.0 verwendet. Die Syntax ist weitestgehend mit der von Matlab identisch. Die Skripts sind unter Verwendung von [Eat+11] und [Beu08] sowie der Software und den Zusatzpakten entstanden. Im laufe der Arbeit entstanden Funktionen, welche die Durchführung der Simulationen erleichterten.

4.2.1. Signalerstellung für DSV

Für jede Reihe von PWM-Frequenzen wird ein Testsignal erstellt. Für das Testsignal werden zunächst zwei Einzelsignale erzeugt. Jedes besteht aus vier überlagerten PWM-Frequenzen. Das erste Signal (SIG1) wird aus den ungeraden PWM-Frequenzen erstellt. Das zweite Signal (SIG2) wurde aus der Überlagerung der geraden PWM-Frequenzen erstellt. Diese beiden Signale werden zu einem Testsignal zusammengesetzt. Abbildung 4.4 stellt das Testsignal schematisch dar. Die Wahl dieser Darstellungsform begründet sich damit, dass die PWM-Frequenzen für jede Reihe unterschiedlich sind. Bei einer Betrachtung des zeitlichen Verlaufs mittels eines Amplitudensignals würde dieses für jede Reihe anders aussehen.

Abbildung 4.4.: Schematische Darstellung Testsignal

Die Zeit in der das jeweilige Signal anliegt, wird über die Abtastfrequenz f_{sam} und der Anzahl der Abtastwerte N bestimmt. Für die Testsignale für die geometrische und arithmetische Reihe wurde mit Hilfe der Tiefpassgrenzfrequenz von 100kHz und Gleichung 2.20 eine Abtastfrequenz von $f_{sam} = 1,024\text{MHz}$ bestimmt. Die Abtastwerte wurden auf $N = 1024$ festgelegt, womit jede Millisekunde ein Wechsel der anliegenden Frequenzen von statten geht. Das Testsignal für die DFT-Reihe kann mit den selben Parametern erstellt werden, jedoch wird nur ein Viertel der Abtastwerte pro anliegenden Signal benötigt, um eine Frequenzauflösung von 4kHz darzustellen. Daraus folgt, dass die anliegenden Frequenzen alle 250µs wechseln können.

Nach der Erstellung des Testsignals wird dieses mit einen Butterworth Tiefpassfilter zweiter Ordnung und einer Grenzfrequenz von 100kHz gefiltert. Anschließend wird die Abtastfrequenz f_{sam} und die Anzahl der Zwischenschritte um den Faktor vier reduziert. Daraus resultiert eine Abtastfrequenz von 256kHz und 256 Abtastwerte pro anliegenden Signal. Die Gesamtzeit des Testsignals beträgt somit 7 ms und die Symbolrate 1 ms. Anschließend wird dieses Testsignal digital weiterverarbeitet.

4.2.2. Digitale Signalverarbeitung

Die Digitale Signalverarbeitung dient der Rückgewinnung der Frequenzinformationen. Die bisher genannten Schritte waren lediglich die Vorbereitung dafür. Für die Erkennung der Frequenzinformationen werden zwei Verfahren verwendet, zum einen die Digitale Filterung und zum anderen eine Spektralanalyse mittels FFT. Das erste Verfahren, die Digitale Filterung lässt sich auf alle Reihenverfahren anwenden. Für die Rückgewinnung der Frequenzinformationen werden Bandpassfilter für die PWM-Frequenzen erstellt. Jeder Filter ist so Konfiguriert, dass eine PWM-Frequenz in den Parametern liegt. Die umliegenden Oberwellen von diesen PWM-Frequenzen werden Störungen betrachtet und entsprechend gefiltert. Die Spektralanalyse mittels FFT hat, im Gegensatz zur Digitalen Filterung, den Nachteil, dass die darzustellenden Frequenzen über die Frequenzauflösung definiert sind. Das bedeutet für die geometrische und arithmetische Reihe an PWM-Frequenzen, dass die Frequenzen nicht eineindeutig dargestellt werden. Im schlimmsten Fall werden nahe beieinander liegende PWM-Frequenzen und Oberwellen als eine Frequenz dargestellt. Deswegen ist eine Anwendung, dieses Verfahrens außerhalb von Spezialfällen und speziell angepassten Reihen nicht sinnvoll.

4.2.2.1. Digitale Filterung

Die digitale Filterung wurde auf die geometrischen und arithmetischen Reihen angewendet. Jede Reihe wurde mit jeweils mit einem Vertreter der FIR und IIR-Filter simuliert. Beide Filter sind als Bandpassfilter realisiert, welche auf die jeweilige Reihe konfiguriert sind. Die Parameter für die Reihen ohne Überschneidung sind in Tabelle 4.7 dargestellt. Für die Reihen mit Überschneidung der Harmonischen, sind die Parameter in Tabelle 4.8 dargestellt.

Tabelle 4.7.: Reihen ohne Überschneidung

Simulation	Reihe	Bandpass		
		Filtertyp	Ordnung	Grenzfrequenz
1	geometrisch	FIR	184	± 1300
2	geometrisch	IIR	4	± 1300
3	arithmetisch	FIR	130	± 1700
4	arithmetisch	IIR	4	± 1700

Die Grenzfrequenzen für die Bandpassfilter, für Tabelle 4.7, wurden mithilfe des geringsten Abstandes zwischen zwei PWM-Frequenzen bestimmt. Für Tabelle 4.8

Tabelle 4.8.: Reihen mit Überschneidung

Simulation	Reihe	Bandpass		
		Filtertyp	Ordnung	Grenzfrequenz
1	geometrisch	FIR	92	± 2500
2	geometrisch	IIR	3	± 2500
3	arithmetisch	FIR	110	± 2000
4	arithmetisch	IIR	4	± 2000

war es notwendig die Oberwellen miteinzubeziehen, damit diese außerhalb der Filtergrenzen liegen. In den Tabellen ist lediglich der Wert gegeben, welche von der jeweiligen PWM-Frequenz abgezogen bzw. dazu addiert werden muss. Daraus folgt, dass für alle acht PWM-Frequenzen ein Bandpassfilter entsteht. Diese acht Filter arbeiten unabhängig von einander. Es ist jedoch notwendig, dass sie alle die gleiche Laufzeit besitzen. Die Laufzeit ist abhängig von der Filterordnung, diese wird je nach Bandbreite des Filters neu angepasst. Aus diesem Grund wäre eine Anpassung der Grenzfrequenzen für jede PWM-Frequenzen nicht sinnvoll, da sich die Filterordnung nachdem Filter mit der geringsten Bandbreite richtet. Zu beachten gibt es, dass die Filterordnung indirekt Einfluss auf die Zeit nimmt in der die zu filternde Frequenz anliegen muss. Für die Simulation war es aufgrund der relativ hohen Filterordnungen des FIR-Filters notwendig die Anzahl der Abtastwerte auf diese anzupassen. Was allerdings auch bedeutet, dass eine Symbolrate von der Filterordnung abhängt. Für eine praktische Realisierung sind IIR-Filter für diese Anwendung besser, da ihre Laufzeit geringer ist als die der FIR-Filter. Des Weiteren kann auf die Phaseninformation verzichtet werden, da nur die Frequenzinformation wichtig ist.

In Abbildung 4.5 ist der Simulationsablauf für die digitalen Bandpassfilter dargestellt. Ersichtlich ist, das in der Simulation die Bandpassfilterung als Schleife erfolgt und somit nacheinander. In einer praktischen Realisierung muss die Filterung für jede Frequenz zur selben Zeit geschehen.

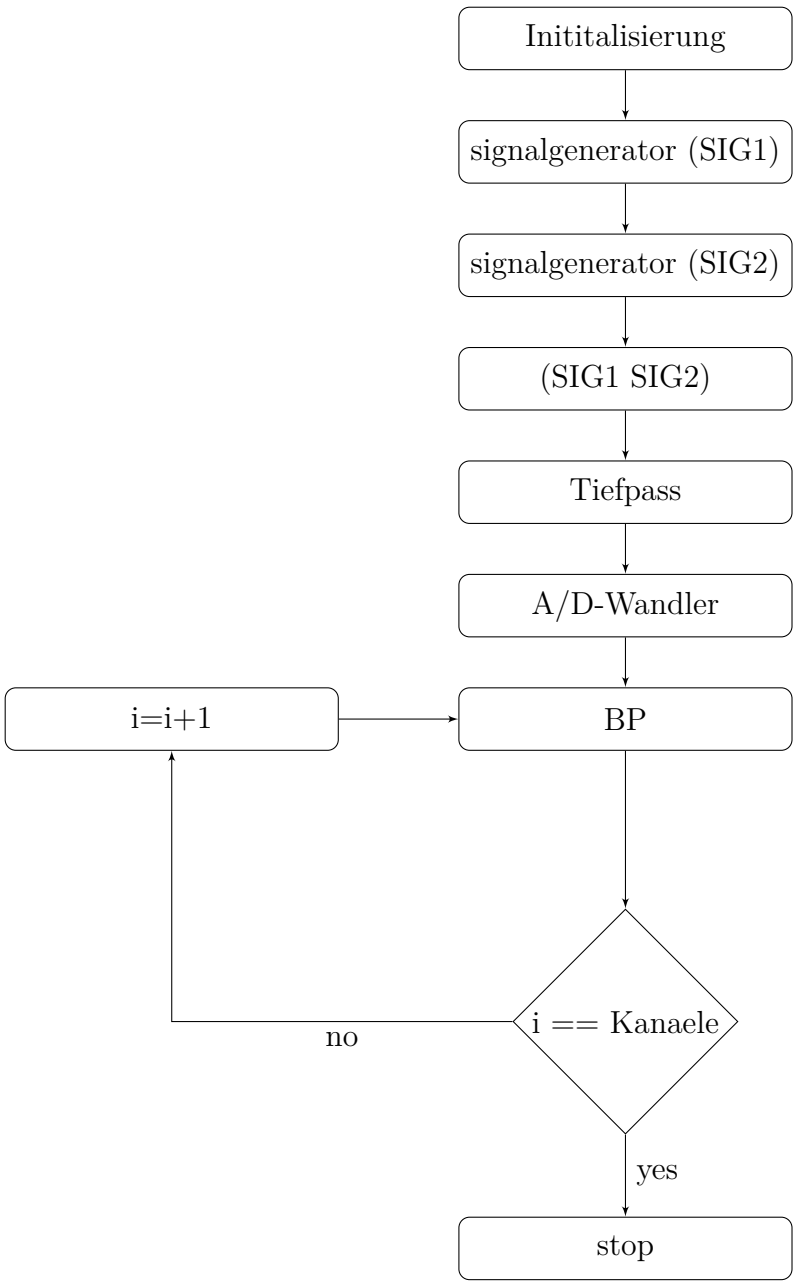

Abbildung 4.5.: Simulationsablaufplan digitale Filter

Abbildung 4.6 stellt das Testsignal nach der Bandpassfilterung des ersten Kanals dar, dieser ist auf die erste PWM-Frequenz ausgerichtet. Der Frequenzwechsel im Testsignal spiegelt sich deutlich im gefilterten Testsignal wider. Des Weiteren ist eine Ein- und Abschwingzeit zu beobachten, diese sind abhängig von der Filterordnung und Art des Filters. Durch den harten Frequenzwechsel der jede Millisekunde geschieht, ist ein Frequenzüberschwingen zwischen der vierten und fünften Millisekunde zu sehen.

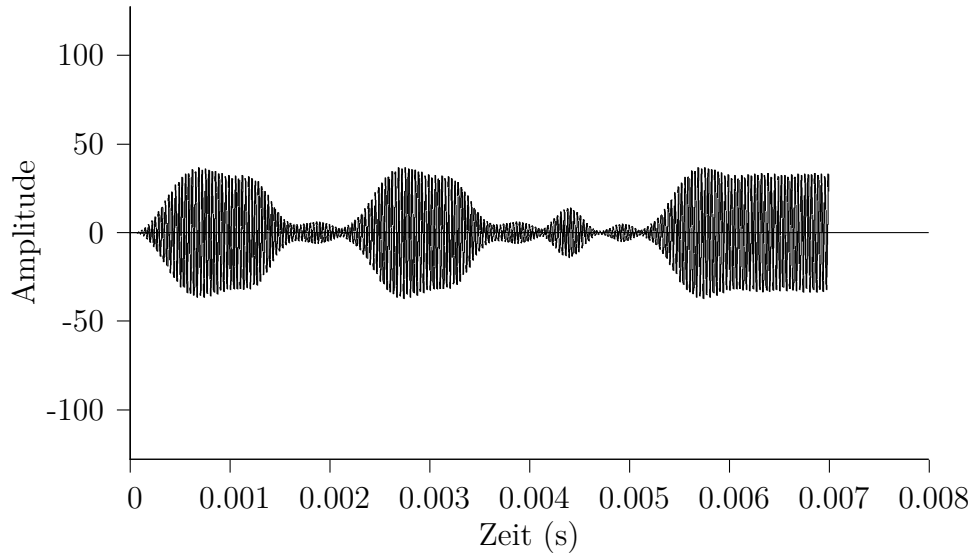

Abbildung 4.6.: Gefiltertes Signal des ersten Kanals

Die Auswertung des Kanal kann mit einen Hüllkurvendetektor durchgeführt werden. Anschließend muss ein Grenzwert definiert werden, damit eine Aussage über das Vorhandensein der Frequenz getroffen werden kann. Dasselbe gilt für den zweiten Kanal in Abbildung 4.7. Es wird hier die gefilterte zweite PWM-Frequenz zeitlich dargestellt. Diese beiden PWM-Frequenzen bilden ein Paar, somit kann eine eindeutig Aussage über das Momentan anliegende Symbol getroffen werden. Die Simulationen wurden mit den Skripts aus Anhang B.3 und B.2 durchgeführt.

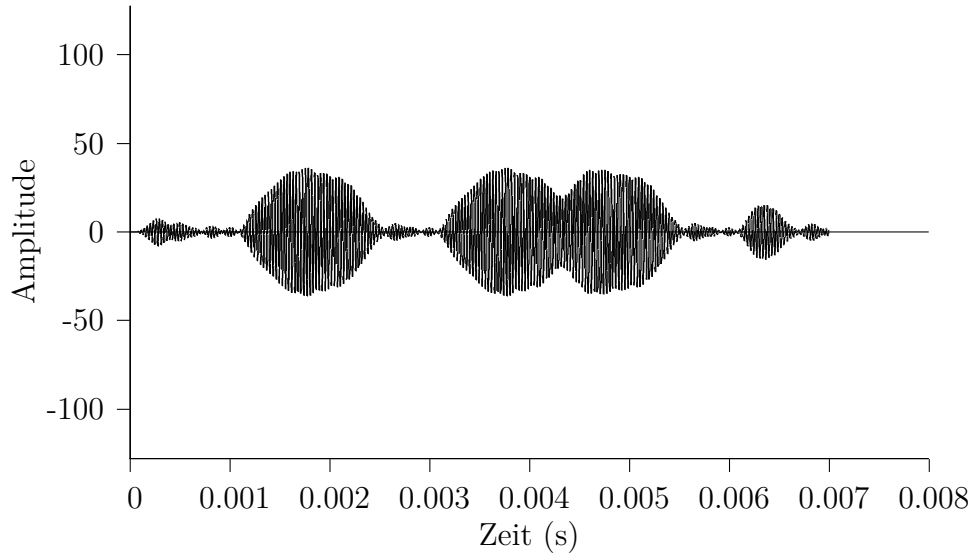

Abbildung 4.7.: Gefiltertes Signal des zweiten Kanals

4.2.2.2. Spektralanalyse

Die Spektralanalyse wird auf die DFT-Reihen angewendet. Bei diesen Verfahren wird eine bestimmte Anzahl von zeitlich abgetasteten Werten mittels FFT bzw. DFT in den Frequenzbereich transferiert. Für die Verwendung der DFT-Reihen aus Tabelle 4.5 und 4.6 benötigt man eine Frequenzauflösung von $f_{res} = 4\text{kHz}$. Um diese zu erhalten muss bei der Abtastfrequenz von 256kHz, die Anzahl der abzutastenden Werte 64 betragen. (Gleichung 2.13) Mit diesem Verhältnis ist zugleich die maximale Symbolrate des Systems beschrieben. Die maximale Symbolrate beträgt 4000 Baud.

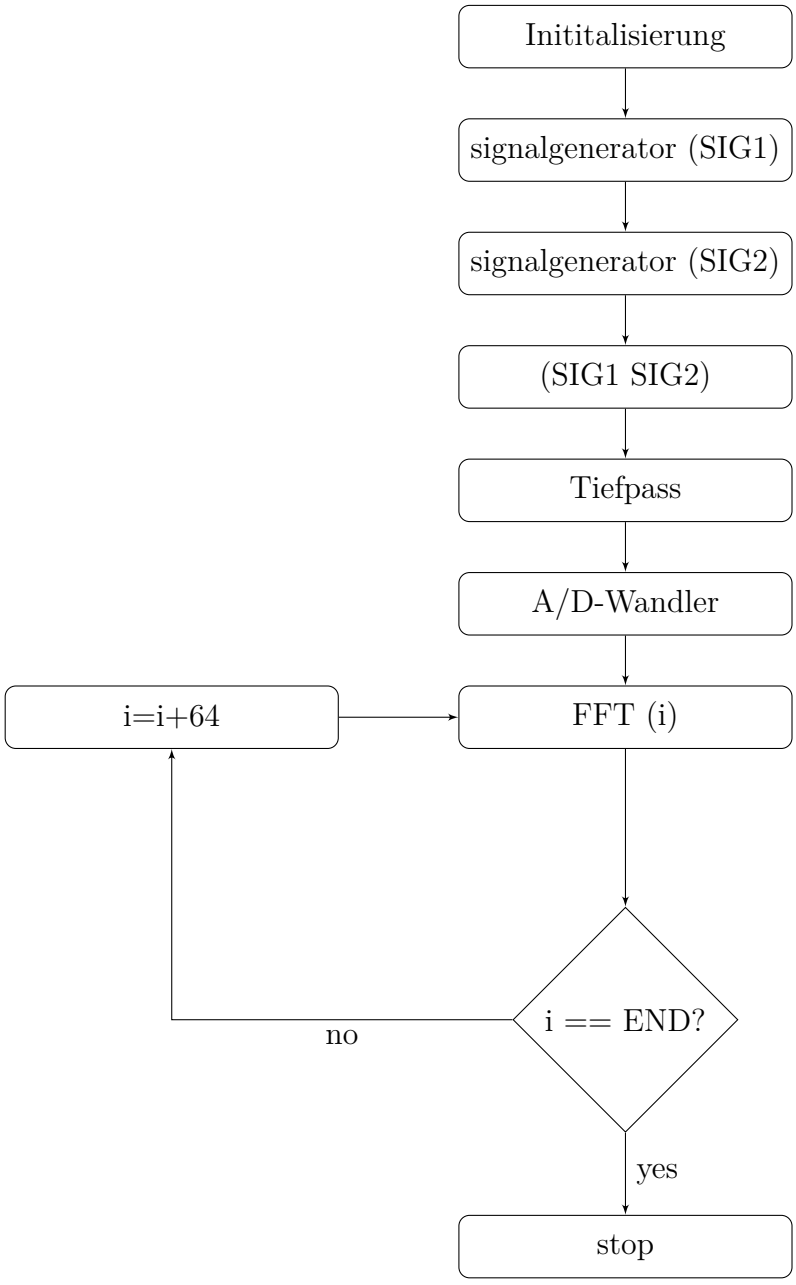

Abbildung 4.8.: Simulationsablaufplan Spektralanalyse: DFT-Reihe

Abbildung 4.8 stellt den Simulationsablaufplan für die Spektralanalyse dar. Aus diesem geht hervor, dass alle 64 Werte eine FFT gemacht wird, dies ist sowohl in der Simulation als auch in der Praxis nötig. Der einzige Unterschied besteht darin, dass das in der Simulation nur bis zum Ende des Testsignals geschieht und praktisch in einer Endlosschleife. Dies ist notwendig, damit eine zeitliche Aussage über die momentan anliegenden Frequenzen getroffen werden kann. Es empfiehlt sich für einen praktischen Aufbau die Abtastung und die FFT parallel durchzuführen. Das bedeutet, dass die FFT, für ihre Berechnung, einen vollen Zyklus von N-Abtastwerten Zeit hat.

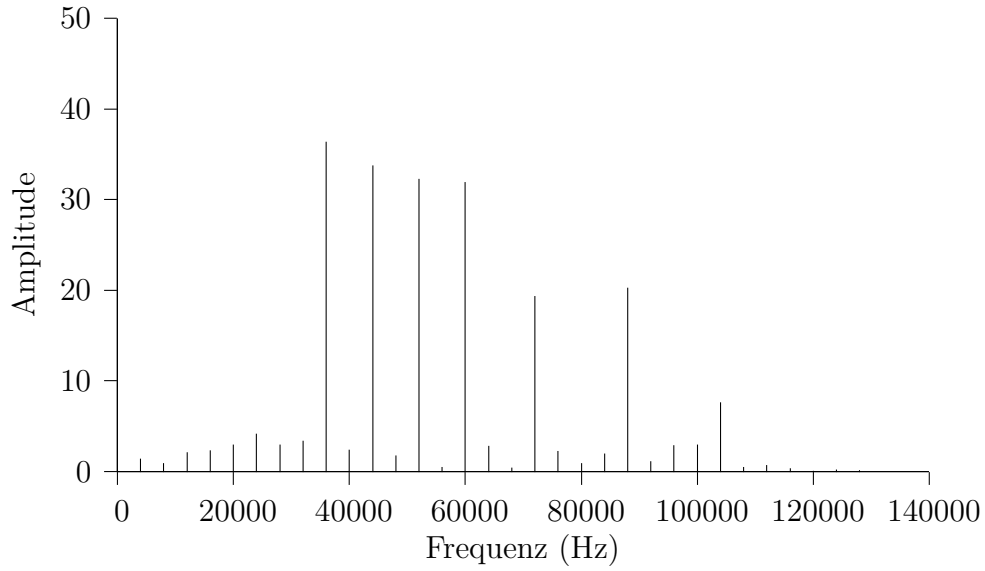

Abbildung 4.9.: Amplitudenspektrum: abgetastet SIG2

Abbildung 4.9 stellt ein Amplitudenspektrum von dem Testsignal dar, welches aus der DFT-Reihe aus Tabelle 4.5 entstand. Das Spektrum stellt einen Bereich dar, in dem die PWM-Frequenzen von SIG2 anliegen. Dabei gibt es zu beachten, dass nur die 8. bis 15. Spektralkomponente zu der DFT-Reihe gehört. Ersichtlich ist dies aufgrund der Tatsache, dass in den genannten Bereich nur vier hohe Amplituden dargestellt sind. Die hohen Amplituden über 60kHz gehören schon zu den Oberwellen der Reihe. Um das Vorhandensein der PWM-Frequenzen zu bestimmen, muss ein Schwellwert definiert werden. Damit eine Zuordnung des anliegenden Symbols von jedem Sender möglich ist, muss zusätzlich das PWM-Frequenz Paar jedes Senders bekannt sein.

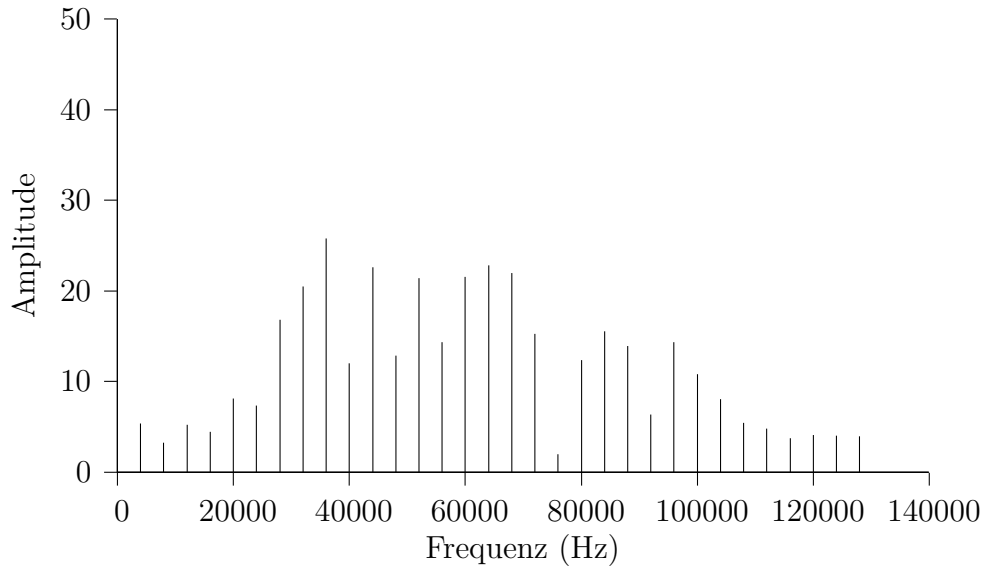

Abbildung 4.10.: Amplitudenspektrum: abgetastet zwischen SIG1 und SIG2

Abbildung 4.10 stellt einen abgetasteten Bereich des Testsignals dar, indem ein Signalwechsel von SIG1 zu SIG2 stattfindet. Damit findet begleitend ein Frequenzwechsel statt. Im Spektrum ist aufgrund dieses Wechsels zu beobachten, dass alle PWM-Frequenzen und ihre Oberwellen dargestellt sind. Dadurch ist kann keine eineindeutige Aussage über das anliegende Symbol getroffen werden, man befindet sich in einem nicht definierten Bereich. In diesem Fall sind durch den einheitlichen Wechsel der PWM-Frequenzen alle Sender gleichermaßen betroffen.

Der Umgang mit den Zwischenabtastungen hängt von der Symbolrate des Systems ab. Für den Fall, dass die Symbolrate gleich der Zeit eines Abtastzyklus von 64 Werten ist. Dürfen keine Zwischenabtastungen stattfinden. Das bedeutet, es müssen alle Sender zueinander synchronisiert sein. Damit die anliegenden PWM-Frequenzen ausgewertet werden können. Laufen Beispielsweise nur zwei synchron und die anderen beiden Phasenverschoben, so wäre es möglich den Empfänger auf die zwei Synchron laufenden zu synchronisieren. Allerdings ist sind die anderen beiden in diesem Fall nicht Auswertbar, weil außerhalb von einer längeren Anlegezeit des selben Symbols keine eindeutige Aussage über das anliegende Symbol getroffen werden kann. Für Symbolraten die ein vielfaches der Zeit eines Abtastzyklus von 64 Werten bilden, sollten Zwischenabtastungen kein Problem darstellen. Es muss lediglich bei der Auswertung der anliegenden PWM-Frequenzen definiert werden, dass wenn beide PWM-Frequenzen eines Senders vorhanden sind, das Letzte bekannte Symbol weiterhin anliegt. Diese Simulationen wurden mit den Skripts aus Anhang B.4 durchgeführt.

5. Fazit und Ausblick

Die Untersuchung hat gezeigt, dass ohne eine vernünftige Auswahl von PWM-Frequenzen eine Rückgewinnung der Frequenzen und somit der Information nicht möglich ist. Durch die jeweilige Methode zur Bestimmung dieser Frequenzen schränken sich die möglichen Verfahren und Herangehensweisen für eine Rückgewinnung der Frequenzen ein. Die Verfahren der digitalen Signalverarbeitung für die Frequenzrückgewinnung unterscheiden sich in ihrer Flexibilität. Die digitale Filterung ist durch ihre freie Wahl ihrer Frequenzgrenzen, an keine festen Frequenzspektren gebunden. Allerdings haben sie den Nachteil, dass sie für die Schmalbandfilterung meist hohe Filterordnungen benötigen, wodurch sich ihre Ein- und Abschwingzeit erhöht. Des Weiteren wird dadurch die mögliche Symbolrate begrenzt. Bei diesem Verfahren kann der Informationsträger nicht direkt sondern nur über zusätzliche Funktionen wie Hüllkurvendetektoren und definierten Schwellwerten ausgewertet werden. Die Spektralanalyse benötigt zwar für die Auswertung ebenfalls einen Schwellwert jedoch kann dieser direkt auf die transformierten Werte angewendet werden. Das Verfahren eignet sich gut für schmalbandige Systeme hat allerdings den Nachteil, dass je schmalbandiger das System sein soll desto geringer ist die Symbolrate. Die benötige Symbolrate kann nicht direkt mit der benötigen Symbolrate der digitalen Filter verglichen werden, da es für letztere verschiedene Konstellationen gibt. Es kann allerdings behauptet werden, dass mittels Spektralanalyse ein größeres System mit mehr Frequenzen und einen geringeren technischen Aufwand entstehen kann. Letzten Endes entscheidet die Hardware des Systems, welche Verfahren und Reihen man für die Rückgewinnung der Frequenzinformationen verwenden kann. Wenn die freie Wahl hat, empfiehlt sich das Verfahren der Spektralanalyse in Verbindung mit einer DFT-Reihe als Systemaufbau es hat die größte Erweiterbarkeit.

Die Fortführung der Arbeit wäre die Auswahl einer entsprechenden Hardware für einen Versuchsaufbau, der die Simulation überprüft. Anschließend müsste diese Hardware um die sendenden Lichtquellen und der Sensorik am Empfänger erweitert werden um ihr Verhalten zu einander zu bestätigen und auszuwerten. Der nächste Schritt wäre die Datenverarbeitung anhand der empfangenen Symbole und die damit einhergehenden Auswertungen der nötigen Verarbeitungsgeschwindigkeit und Fehlerrate. Anschließend würden Optimierungsversuche für die digitalen Filter stattfinden. Für die Spektralanalyse können verschiedene Berechnungsalgorithmen für eine schnellere Berechnung ausprobiert werden oder man versucht nur die Spektralkomponenten zu berechnen die belegt sind. Des Weiteren wären andere Systemkonstellationen mit unterschiedlichen Abständen und Anordnungen auf ihre Funktionalität zu prüfen.

6. Vom Autor verwendete Software

Im Folgenden werden die Programme vorgestellt, die der Autor zum Erstellen dieser Arbeit und der Simulation sowie den Berechnungen verwendet hat.

- **GNU Octave**
  Die Simulationen wurden mit GNU Octave, einer freien Software die numerische Lösungen mathematischer Probleme erstellt.
  Website: http://www.gnu.org/software/octave/
- **Gnuplot**
  Hier bei handelt es sich um ein Programm zur grafische Darstellung von Messdaten und mathematischen Funktionen. Es wurde verwendet, um die Graphen in dieser Arbeit zu erstellen.
  Website: http://www.gnuplot.info/
- **LaTeX**
  Diese Arbeit wurde mit LaTeX geschrieben. Als Distribution wurde TeX Live verwendet und als Editor TeXstudio.
  Websites: https://www.tug.org/texlive/,
  http://texstudio.sourceforge.net/
- **Microsoft Excel**
  Bei Microsoft Excel handelt es um ein weit verbreitetes Tabellenkalkulationsprogramm. Es ist ein Teil der Microsoft-Office-Suite.
  Website: http://www.microsoft.de/office/

Literaturverzeichnis

[Beu08] Ottmar Beucher. *MATLAB und Simulink - Grundlegende Einführung für Studenten und Ingenieure in der Praxis*. 4. aktualisierte Auflage. München: Pearson Deutschland GmbH, 2008. ISBN: 978-3-827-37340-3.

[Eat+11] John W. Eaton u. a. *GNU Octave*.
URL: https://www.gnu.org/software/octave/octave.pdf [Online; Stand 24. August 2014]. 2011.

[Kam+98] Karl Dirk Kammeyer u. a. *Digitale Signalverarbeitung. Filterung und Spektralanalyse mit MATLAB- Übungen.* Teubner Verlag, 1998. ISBN: 3519361221.

[Krü02] Klaus-Eberhard Krüger. *Transformationen: Grundlagen und Anwendungen in der Nachrichtentechnik (Studium Technik) (German Edition)*. Vieweg+Teubner Verlag, 2002. ISBN: 3528039086.

[KS10] Ralf Kories und Heinz Schmidt-Walter. *Taschenbuch der Elektrotechnik: Grundlagen und Elektronik*. 9., korrigierte Aufl. Frankfurt und M: Deutsch, 2010. ISBN: 9783817118588.

[Loc97] Dietmar Lochmann. *Digitale Nachrichtentechnik*. Verlag Technik /Huss Medi, 1997. ISBN: 3341011846.

[Pap09] Lothar Papula. *Mathematische Formelsammlung für Ingenieure und Naturwissenschaftler ; mit zahlreichen Rechenbeispielen und einer ausführlichen Integraltafel*. 2009. ISBN: 9783834807571.

Eidesstattliche Erklärung

Ich, Norman Schmidt, Matrikel-Nr. 20104002, versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema

> *Rückgewinnung der Frequenzinformation aus PWM-moduliertem Licht zur Anwendung für die Visible Light Communication (VLC)*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mir ist bekannt, dass ich meine Bachelorarbeit zusammen mit dieser Erklärung fristgemäß nach Vergabe des Themas in zweifacher Ausfertigung und gebunden im Prüfungsamt der Fachhochschule Brandenburg abzugeben oder spätestens mit dem Poststempel des Tages, an dem die Frist abläuft, zu senden habe.

Brandenburg an der Havel, den 28. August 2014

Norman Schmidt

A. Anhang

A.1. CD

CD mit folgendem Inhalt

- eine pdf-Version der vorliegenden Abschlussarbeit
- die Excel-Tabellen für die Berechnung der PWM-Frequenzen
- die Skriptdateien die im Rahmen der Abschlussarbeit für die Simulation erstellt wurden
- verwendete Internetquellen im PDF-Format

B. Simulationsskripts

Im folgenden sind die für die Simulation entstanden Skripts und Funktionen aufgezeigt.

B.1. Octave-Funkionen

Kenndaten Die Kenndaten, Abtastfrequenz, Anzahl der Abtastwerte und Frequenzauflösung, werden aus einer Reihe von Frequenzen erzeugt. Diese liegt als Matrix vor. Der Inhalt wird nach Größe sortiert. Die höchste Frequenz und der kleinste Abstand zwischen zwei aufeinander folgenden Frequenzen wird erfasst. Die Funktion hat mehrere Modi, so können die Kenndaten für Abtastfrequenz und Anzahl der Abtastwerte manuell vorgegeben oder mit Faktoren berechnet werden. Für letzteres sind die erfassten Daten aus der Frequenzmatrix wichtig.

Listing B.1: Funktion: `kenndaten`

```
1  %#########################################################################
2  % Autor: Norman Schmidt
3  % Date:  30.Juli.2014
4  %
5  % Decription:
6  %   Bestimmung der noetigen Aufloesung, damit eine unterscheidung der
7  %   Freqeuenzen zueinander noch moeglich ist. Die Abtastfrequenz wird
8  %   mittels der hoechsten Frequenz eingegebenen Frequenz bestimmt.
9  %#########################################################################
10
11 function [abtastfrequenz, N_abtast, resolution ] = kenndaten(frequenzen,
       abtastfaktor, aufloesungsfaktor, varargin)
12
13   if nargin < 3 || nargin > 10
14     print_usage;
15   end
16
17   stype=0;
18   Ntype=0;
19   afreq=0;
20
21   %% sort arglist, normalize any string
22   %% Copy from fir1.m
23   for i=1:length(varargin)
24     arg = varargin{i};
25     if ischar(arg), arg=lower(arg);end
26     if isempty(arg) continue; end % octave bug---can't switch on []
27     switch arg
28       case {'abt_dynamisch'},    stype = 0;
29       case {'abt_statisch'},     stype = 1;
```

```
30      case {'dyn'},          Ntype  = 0;
31      case {'stat'},         Ntype  = 1;
32      case {'1freq'},         afreq  = 1;
33      otherwise             window = arg;
34    end
35  end
36
37 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
38  if length(frequenzen)>1
39  %
40  % ABTASTFREQUENZ
41    f_sort=sort(frequenzen); % Nach groesse Sortieren
42
43    abtastfrequenz = f_sort(length(f_sort),:); % hoechste Position ==
          Abtastfrequenz
44
45    if stype == 0
46      abtastfrequenz = abtastfaktor*abtastfrequenz;
47    elseif stype == 1
48      abtastfrequenz = abtastfaktor;
49    end
50  %
51  % FREQUENZAUFLOESUNG
52    fres = 9999999999999999; % Gross, weil kleinst moeglich gesucht
53
54    for i=2:1:length(f_sort)
55      fres_temp=f_sort(i,:)-f_sort(i-1,:);
56      if (fres_temp<fres)
57        fres=fres_temp;
58      end
59    end
60
61  %
62  % ABTASTWERTE N^2
63  if Ntype == 0
64      fres = fres/aufloesungsfaktor;
65      resolution = fres;
66      N_tmp = abtastfrequenz/fres;
67      N_abtast = 2^nextpow2(N_tmp); % aufrunden auf 2^N wert
68  elseif Ntype > 0
69      N_abtast = aufloesungsfaktor;
70      % laenge durch Abtastwerte bestimmt
71      fres = abtastfrequenz/N_abtast;
72      resolution = fres;
73  end
74
75  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
76  elseif length(frequenzen)==1
```

```
77    %
78    % ABTASTFREQUENZ
79      abtastfrequenz = frequenzen;
80      if stype == 0
81        abtastfrequenz = abtastfaktor*abtastfrequenz;
82      elseif stype == 1
83        abtastfrequenz = abtastfaktor;
84      end
85    %
86    % ABTASTWERTE N^2
87      if Ntype == 0
88          %
89          % FREQUENZAUFLOEUNG
90          fres = frequenzen/5;
91          fres = fres/aufloesungsfaktor;
92          resolution = fres;
93          N_tmp=abtastfrequenz/fres;
94          N_abtast = 2^nextpow2(N_tmp); % aufrunden auf 2^N wert
95      elseif Ntype > 0
96          N_abtast=aufloesungsfaktor;
97          %
98          % FREQUENZAUFLOEUNG
99          % laenge durch Abtastwerte bestimmt
100         fres = abtastfrequenz/N_abtast;
101         resolution = fres;
102     end
103   end
104   %
105   % AUSGABE DER ERGEBNISSE
106     fprintf('Abtastfrequenz (Hz): %f \n',abtastfrequenz);
107     fprintf('Die Frequenzaufloesung ist (Hz): %f \n',fres);
108     fprintf('Anzahl der Abtastwerte betroegt: %f entspricht %f
          Spektralkomponenten\n',N_abtast,((N_abtast/2)+1));
109 end
```

Signalgenerator Der Signalgenerator erzeugt zwei Ausgangsmatrizen. Diese stellen in einen kartesischen Koordinatensystem den zeitlichen Amplitudenverlauf dar. Das heißt eine Matrix wird auf Grundlage der zweiten Erstellt. Für die Erstellung der Zeitmatrix werden Abtastfrequenz sowie die Anzahl der Abtastwerte, als Eingabewerte, benötigt. Die Matrix der Amplitudenwerte benötigt die Eingabe eines Amplitudenwertes und einer Frequenz, welche in Abhängigkeit der Zeitmatrix gesetzt werden. Für eine Überlagerung mehreren Frequenzen und Amplituden, können diese Werte als Matrizen eingespeist werden. Zusätzlich ist es möglich die Signalform zwischen Sinus, Kosinus und PWM zu wählen. Durch letztere müssen die Eingabewerte um den Tastgrad bzw. bei Überlagerung Tastgradmatrix erweitert werden.

Je nach Simulation oder nötigen Darstellung kann es vorteilhaft sein einen Signalverlauf zu haben der einen analogen Verlauf ähnlicher ist. Deswegen wurde die Funktion um einen Modi erweitert, dieser Multipliziert die Abtastfrequenz und die Anzahl der Abtastwerte um den Faktor 2^N.

Listing B.2: Funktion: `signalgenerator`

```
%#########################################################################
% Autor: Norman Schmidt
% Date:  31.Mai.2014
%
% Decription:
%   Signalgenerator optional mit rauschenden Ausgangssignal.
%   Mithilfe dieses Generators koennen beliebig viele Frequenzen mit
%   unterschiedlichen Amplituden ueber eine bestimmte Zeit(zeit vektor)
%   gemischt werden. Es kann zusaethlich ein Rauschen hinzugefuegt, um
%   reale Bedingungen zu simulieren.
%   Es es moeglich folgende Signalformen darzustellen: Sinus, Cosinus,
%                                                      Rechtecksignal,
%                                                      PWM-Signal
%
%   Hinweis: Rauschen ist auskommentiert.
%#########################################################################

function [ausgangssignal, zeit] = signalgenerator (wichtung, frequenz,
    abtastfrequenz, N,pulsbreite, amplitude, varargin)

  if nargin < 5 || nargin > 10
    print_usage;
  end

  atype=1;
  stype=0;
  dtype=0;

  %% sort arglist, normalize any string
  for i=1:length(varargin)
    arg = varargin{i};
    if ischar(arg), arg=lower(arg);end
    if isempty(arg) continue; end % octave bug---can't switch on []
    switch arg
      case {'sin'},          stype = 2;
      case {'cos'},          stype = 1;
      case {'square'},       stype = 0;
      case ('a2'),           atype = 2;
      case ('a4'),           atype = 4;
      case ('a2'),           atype = 2;
      case ('a8'),           atype = 8;
```

```
41      case ('a16'),           atype = 16;
42      otherwise             window = arg;
43    end
44  end
45 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
46 % Erstellung der Zeitmatrix
47 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
48 N=N*atype;
49 abtastfrequenz=abtastfrequenz*atype;
50 zeit=(0:N-1)*(1/abtastfrequenz);
51
52 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
53 % Wichtung der Einzelsignale bestimmen
54 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
55  gesamtgewicht =0;
56  for w=1:length(wichtung)
57    gesamtgewicht = gesamtgewicht+wichtung(w,:);
58  end
59
60  for w=1:length(wichtung)
61    wichtung(w,:)=((wichtung(w,:)/gesamtgewicht)*amplitude);
62  end
63
64 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
65 % Bildung der Kreisfrequenz
66 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
67  kreisfrequenz = ((2 * pi).*frequenz);
68
69 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
70 % Erstellung des Signales
71 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
72  if stype == 0
73    sigcharts = wichtung(1,:).*square(kreisfrequenz*zeit, pulsbreite(1,:));
74  elseif stype == 1
75    sigcharts = wichtung(1,:).*cos(kreisfrequenz*zeit);
76  elseif stype == 2
77    sigcharts = wichtung(1,:).*sin(kreisfrequenz*zeit);
78  else
79    disp('type are square, sin and cos')
80  end
81 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
82 % Erstellung des Rauschens
83 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
84 %rauschen = 6*randn(size(zeit));
85
86 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
87 % Mischen der einzelnen Signale mit dem Rauschen durch Addition dieser.
88 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
89 if (dtype==0)
90    ausgangssignal=0;
91    for i = 1:1:(length(wichtung))
92       ausgangssignal = (ausgangssignal + sigcharts(i,:));
93    end
94 %ausgangssignal = ausgangssignal + rauschen;
95 else
96    ausgangssignal = sigcharts1 + sigcharts2;
97 end
98
99 end
```

Signalfolge Diese Funktion kombiniert einzelne Signalverläufe zu einen einzigen. Damit diese nicht verfälscht werden müssen alle eingespeisten Verläufe mit dem selben Schrittintervall –Abtastfrequenz– erstellt worden sein. Sollten die Signale mit dem Signalgenerator und dessen Zusatzfunktion für analoge erstellt worden sein kann dieser Modi hier übernommen werden.

Listing B.3: Funktion: `signalfolge`

```
1  %##############################################################################
2  % Autor: Norman Schmidt
3  % Date:  30.Juli.2014
4  %
5  % Decription:
6  %   Vereint mehrere Signale mit gleicher Laenge und Abtastfrequenz
7  %   miteinander.
8  %##############################################################################
9
10 function [ausgangssignal, zeit] = signalfolge (abtastfrequenz, signale, varargin)
11
12
13   if nargin < 2 || nargin > 10
14     print_usage;
15   end
16
17   atype=1;
18
19   %% sort arglist, normalize any string
20   for i=1:length(varargin)
21     arg = varargin{i};
22     if ischar(arg), arg=lower(arg);end
23     if isempty(arg) continue; end % octave bug---can't switch on []
24     switch arg
25       case ('a2'),            atype  = 2;
26       case ('a4'),            atype  = 4;
27       case ('a2'),            atype  = 2;
28       case ('a8'),            atype  = 8;
```

```
29      case ('a16'),           atype = 16;
30      otherwise              window = arg;
31    end
32  end
33
34 %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
35 % Zusammenfuegen
36 %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
37
38 [zeilen, spalten]=size(signale);
39 gesamtlaenge=zeilen*spalten;
40 output=[];
41 for i=1:1:zeilen
42     output=[output signale(i,:)];
43 end
44
45 ausgangssignal=output;
46 %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
47 % Erstellung der Zeitmatrix
48 %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
49 N=gesamtlaenge;
50 abtastfrequenz=abtastfrequenz*atype;
51 zeit=(0:N-1)*(1/abtastfrequenz);
52
53 end
```

Reduzierung Diese Funktion Reduziert die Zwischenschritte und die Abtastfrequenz um einen beliebigen Faktor.

Listing B.4: Funktion: `reduzierung adw`

```
1  %########################################################################
2  %
3  % Author:     Norman Schmidt
4  % Date:       30.Juli.2014
5  %
6  % Description:
7  %   Matrix verkleinerung fuer Simulation
8  %
9  %########################################################################
10
11 function [ausgabe, zeit, f_sample] = reduzierung_adw (eingabewerte, e_zeit,
       abtastfrequenz, redfaktor)
12
13 h=1;
14 output=[];
15 outzeit=[];
16 for i=1:redfaktor:(length(eingabewerte))
```

```
% if rem(i,redfaktor)

% else
    output(h)=eingabewerte(i);
    outzeit(h)=e_zeit(i);
    h=h+1;
% end
end

f_sample=abtastfrequenz/redfaktor;
zeit=outzeit;
ausgabe=output;
```

Digitale Filter Diese Funktion dient der Vereinfachung, sie beinhaltet die für GNU Octave nötigen Schritte und Konfigurationen für eine digitale Filterung. Sie wird mit der Abtastfrequenz, Grenzfrequenzen, der Filterordnung sowie den zu filternden Werten gespeist. Sie hat drei Modi, welches erlaubt zwischen drei Filterformen zu wechseln. Der erste ist ein FIR Bandpassfilter. Die zweite und dritte Filterform ist ein Butterworth Design, welches zum einen als Tiefpassfilter und zum anderen als Bandpassfilter Verwendung findet. Der Butterworth Filter gehört zu den IIR Filtern.

Listing B.5: Funktion: `digfilter`

```
%#########################################################################
% Autor: Norman Schmidt
% Date:  30.Juli.2014
%
% Decription:
%   Erstellung digitaler Filter mit anschlieender Filterung der
%   Eingabewerte. Es kann zwischen dem optimalen FIR Filer remez und
% einem Butterworth filter gewaehlt werden.
%
%#########################################################################

function [ausgabe] = digfilter (ordnung, f_grenze ,f_sample ,eingabewerte,
    varargin)

  if nargin < 4 || nargin > 10
    print_usage;
  end

  atype=1;
  ftype=0;

  %% sort arglist, normalize any string
  for i=1:length(varargin)
    arg = varargin{i};
```

```
24    if ischar(arg), arg=lower(arg);end
25    if isempty(arg) continue; end % octave bug---can't switch on []
26    switch arg
27      case ('tiefpass'),           atype = 0;
28      case ('bandpass'),           atype = 1;
29      case ('fir'),          ftype = 0;
30      case ('iir'),          ftype = 1;
31      otherwise             window = arg;
32    end
33  end
34
35 if atype==1
36
37   if length(f_grenze)>2
38     disp('ERROR Filter obere und untere Grenzfrequenz benoetigt');
39   elseif length(f_grenze)>1
40     f_low=f_grenze(1);
41     f_high=f_grenze(2);
42   %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
43   % Normierung der Grenzen
44   %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
45   fl=(f_low/(f_sample*0.5));
46   fh=(f_high/(f_sample*0.5));
47
48   fll=((f_low-500)/(f_sample*0.5));
49   fhh=((f_high+500)/(f_sample*0.5));
50   %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
51   % Erstellen des Filters und Filtern der Eingabewerte
52   %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
53     if ftype==0
54       %b=fir1(ordnung,[fl fh],'pass');
55       b=remez(ordnung,[0, fl,(fl+((fh-fl)/4)), (fh-((fh-fl)/4)),fh, 1],
             [0,0,1,1,0,0], 'bandpass');
56       ausgabe=filter(b,1,eingabewerte);
57     elseif ftype>0
58       [b,a]=butter(ordnung,[fl fh]);
59       ausgabe=filter(b,a,eingabewerte);
60     else
61       disp('ERROR: Filtertype not declared!');
62     end
63
64   else
65     disp('ERROR: nicht genuegend Werte');
66   end
67
68 elseif atype==0
69   if length(f_grenze)>1
70     disp('ERROR: Es wird nur eine Obere Grenzfrequenz benoetigt!');
```

```
71   else
72   f_high =f_grenze;
73   %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
74   % Normierung der Grenzen
75   %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
76   fh=(f_high/(f_sample*0.5));
77   %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
78   % Erstellen des Filters
79   %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
80   %b=fir1(ordnung,fh);
81   [b,a]=butter(ordnung,fh);
82   %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
83   % Filtern der Eingabewerte
84   %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
85   ausgabe=filter(b,a,eingabewerte);
86   end
87 end
88 %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
89 % Anzeigen des genormten Frequenzganges
90 %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
91 figure(90)
92 freqz(b);
93
94 end
```

Spektralanalyse Es werden hiermit die nötigen Parameter für eine Darstellung des Amplituden- und Phasenspektrums erstellt. Die Funktion wird dafür mit der Abtastfrequenz sowie dem Signal, welcher mittels FFT in den Frequenzbereich transformiert werden soll gespeist. Da es nicht immer sinnvoll ist ein Spektrum über den gesamten Zeitverlauf zu erstellen, kann man Startpunkt und die Anzahl der nachfolgenden Werte wählen. Nur dieses Fenster wird transformiert, dass besonders bei sich zeitlich ändernden Signalen nützlich. Es wurde ein Modus eingebaut, der dieses separat darstellt.

Listing B.6: Funktion: `spektrum analyse`

```
1  %#########################################################################
2  % Autor: Norman Schmidt
3  % Date:  31.Juli.2014
4  %
5  % Decription:
6  %   Anzeige von Amplituden und Phasenspektrum eines diskreten Signals
7  %
8  %#########################################################################
9
10 function [betrag, phase, f] = spektrum_analyse (eingabewerte, abtastfrequenz,
       startpunkt, N, varargin )
```

```
11
12  if nargin < 4 || nargin > 10
13    print_usage;
14  end
15
16  atype=0;
17
18  %% sort arglist, normalize any string
19  for i=1:length(varargin)
20    arg = varargin{i};
21    if ischar(arg), arg=lower(arg);end
22    if isempty(arg) continue; end % octave bug---can't switch on []
23    switch arg
24      case ('darstellen'),          atype = 1;
25      otherwise             window = arg;
26    end
27  end
28
29 pos = startpunkt;
30 y= eingabewerte;
31
32 f = abtastfrequenz/2*linspace(0,1,N/2+1);
33 Y = fft(y(pos:(pos+(N-1))))/N;
34
35 betrag=2*abs(Y(1:N/2+1));
36 phase= angle(Y(1:N/2+1));
37
38
39 if atype >0
40   figure;
41   subplot(3,1,1);
42   plot((0:(length(eingabewerte)-1)),eingabewerte);
43   lim = get(gca,'YLim');
44   line([startpunkt startpunkt],lim,'LineStyle','--','LineWidth',2);
45   line([(startpunkt+(N-1)) (startpunkt+(N-1))],lim,'LineStyle','--','LineWidth'
       ,2);
46   ylabel(' Amplitude ');
47   xlabel(' Zeit (s)');
48   subplot(3,1,2);
49   % amplitudenspektrum
50   stem(f,betrag);
51   ylabel(' Amplitude ');
52   xlabel(' Frequenz (Hz)');
53   subplot(3,1,3);
54   % phasenspektrum
55   stem(f,phase);
56   ylabel(' Phase ');
57   xlabel(' Frequenz (Hz)');
```

```
58 end
59 end
```

B.2. Geometrische Reihe

B.2.1. Ohne Überschneidung

B.2.1.1. FIR-Filter

Listing B.7: Geometrische Reihe fortlaufend FIR gefiltert

```
1  %#########################################################################
2  %
3  % Author:     Norman Schmidt
4  % Date:       30.Juli.2014
5  %
6  % Description:
7  % Simulation geometrische Reihe ohne Ueberschneidung und FIR Filter
8  %
9  %#########################################################################
10 clear;
11 clc;
12 close all
13
14 %++++++++++++++++++++++++++++++++++++++++++++++++++
15 % Initialisierung
16 %++++++++++++++++++++++++++++++++++++++++++++++++++
17
18 freq=[30000; 32730; 35708; 38958; 42503; 46371; 50591; 55194];
19
20 tastgrad =0.3;
21 bp_order=184;
22 fg_tiefpass=100000;
23 bp_grenze=1300;
24
25 freq0 = [ freq(1); freq(3); freq(5); freq(7)];
26 ampl0 = [    1;     1;     1;     1];
27 puls0 = [    1;     1;     1;     1]*tastgrad;
28
29 freq1 = [ freq(2); freq(4); freq(6); freq(8)];
30 ampl1 = [    1;     1;     1;     1];
31 puls1 = [    1;     1;     1;     1]*tastgrad;
32
33 fu_bandpass=freq-bp_grenze;
34 fo_bandpass=freq+bp_grenze;
35
36
```

```
37 %++++++++++++++++++++++++++++++++++++++++++++++++++
38 % Kenndaten
39 %++++++++++++++++++++++++++++++++++++++++++++++++++
40
41 [f_sam, N_abt, fres] = kenndaten(freq, (1024*10^3), 1024, 'abt_statisch', 'stat')
       ;
42
43 %++++++++++++++++++++++++++++++++++++++++++++++++++
44 % Signalgenerator SIG1
45 %++++++++++++++++++++++++++++++++++++++++++++++++++
46
47 [y1, t1]=signalgenerator(ampl0, freq0, f_sam, N_abt, puls0, 128, 'square');
48 %figure 1;
49 %plot(t1,y1);
50
51 %++++++++++++++++++++++++++++++++++++++++++++++++++
52 % Signalgenerator SIG2
53 %++++++++++++++++++++++++++++++++++++++++++++++++++
54
55 [y2, t2]=signalgenerator(ampl1, freq1, f_sam, N_abt, puls1, 128, 'square');
56 %figure 2;
57 %plot(t2,y2);
58
59 %++++++++++++++++++++++++++++++++++++++++++++++++++
60 % Signalfolge
61 %++++++++++++++++++++++++++++++++++++++++++++++++++
62
63 [y, t] = signalfolge(f_sam,[y1;y2;y1;y2;y2;y1;y1]);
64
65 figure 1;
66 plot(t,y);
67 title('Zeitliches Signal');
68 xlabel('Zeit (s)');
69 ylabel('Amplitude');
70
71 %++++++++++++++++++++++++++++++++++++++++++++++++++
72 % Tiefpassfilterung
73 %++++++++++++++++++++++++++++++++++++++++++++++++++
74
75 y_tfilt=digfilter(20, fg_tiefpass, f_sam, y, 'tiefpass');
76 %y_tfilt=digfilter(150, [115000 125000], f_sam, y, 'bandpass');
77
78 figure 2;
79 plot(t,y_tfilt);
80 title('Tiefpass gefiltertes Signal');
81 xlabel('Zeit (s)');
82 ylabel('Amplitude');
83
```

```
84 [betrag, phase, f] = spektrum_analyse(y_tfilt,f_sam, 128, 256, 'darstellen');
85
86
87 %++++++++++++++++++++++++++++++++++++++++++++++++++
88 % Reduzierung / ADW
89 %++++++++++++++++++++++++++++++++++++++++++++++++++
90
91 figure;
92
93 [y_red, t_red, f_sam] = reduzierung_adw(y_tfilt, t, f_sam, 4);
94
95 %y_red=y_tfilt;
96 %t_red=t;
97
98 plot(t_red, y_red);
99 title('Signal nach Werte Reduzierung');
100 xlabel('Zeit (s)');
101 ylabel('Amplitude');
102
103
104 %++++++++++++++++++++++++++++++++++++++++++++++++++
105 % Bandpassfilterung
106 %++++++++++++++++++++++++++++++++++++++++++++++++++
107
108 [betrag, phase, f] = spektrum_analyse(y_red, f_sam, 128, 256, 'darstellen');
109
110 for i=1:1:8
111 y_bfilt(i,:)=digfilter(bp_order, [fu_bandpass(i) fo_bandpass(i)], f_sam, y_red, '
        bandpass', 'fir');
112 [betragf(i,:), phasef(i,:), fgf] = spektrum_analyse(y_bfilt(i,:), f_sam, 1,
        length(y_red));
113 end
114
115 pos = [1, 3, 5, 7, 2, 4, 6, 8];
116 figure;
117 for i=1:1:8
118   subplot(4,2,pos(i));
119   stem(fgf,betragf(i,:));
120   tstr = sprintf('KANAL %d', i);
121   title(tstr);
122   xlabel('f (Hz)');
123   ylabel('Amplitude');
124
125 end
126 figure;
127 for i=1:1:8
128   subplot(4,2,pos(i));
129   plot(t_red,y_bfilt(i,:));
```

```
  tstr = sprintf('KANAL %d', i);
  title(tstr);
  xlabel('Zeit (s)');
  ylabel('Amplitude');
end

%[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 1, 256, 'darstellen');
%[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 512, 256, 'darstellen');
%outtable([t_red;y_red], 'testtest.dat');
```

B.2.1.2. IIR-Filter

Listing B.8: Geometrische Reihe fortlaufend IIR gefiltert

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Author:     Norman Schmidt
% Date:       30.Juli.2014
%
% Description:
% Simulation geometrische Reihe ohne Ueberschneidung und IIR Filter
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
clc;
close all

%++++++++++++++++++++++++++++++++++++++++++++++++++
% Initialisierung
%++++++++++++++++++++++++++++++++++++++++++++++++++

%freq=[30000; 32730; 35708; 38958; 42503; 46371; 50591; 55194];
freq=[30000; 32730; 35708; 38958; 42503; 46371; 50591; 55194];

tastgrad =0.3;
bp_order=4;
fg_tiefpass=100000;
bp_grenze=1300;

freq0 = [ freq(1); freq(3); freq(5); freq(7)];
ampl0 = [    1;     1;     1;     1];
puls0 = [    1;     1;     1;     1]*tastgrad;

freq1 = [ freq(2); freq(4); freq(6); freq(8)];
ampl1 = [    1;     1;     1;     1];
puls1 = [    1;     1;     1;     1]*tastgrad;

```

```
34 fu_bandpass=freq-bp_grenze;
35 fo_bandpass=freq+bp_grenze;
36
37
38 %++++++++++++++++++++++++++++++++++++++++++++++++++
39 % Kenndaten
40 %++++++++++++++++++++++++++++++++++++++++++++++++++
41
42 [f_sam, N_abt, fres] = kenndaten(freq, (1024*10^3), 1024, 'abt_statisch', 'stat')
       ;
43
44 %++++++++++++++++++++++++++++++++++++++++++++++++++
45 % Signalgenerator SIG1
46 %++++++++++++++++++++++++++++++++++++++++++++++++++
47
48 [y1, t1]=signalgenerator(ampl0, freq0, f_sam, N_abt, puls0, 128, 'square');
49 %figure 1;
50 %plot(t1,y1);
51
52 %++++++++++++++++++++++++++++++++++++++++++++++++++
53 % Signalgenerator SIG2
54 %++++++++++++++++++++++++++++++++++++++++++++++++++
55
56 [y2, t2]=signalgenerator(ampl1, freq1, f_sam, N_abt, puls1, 128, 'square');
57 %figure 2;
58 %plot(t2,y2);
59
60 %++++++++++++++++++++++++++++++++++++++++++++++++++
61 % Signalfolge
62 %++++++++++++++++++++++++++++++++++++++++++++++++++
63
64 [y, t] = signalfolge(f_sam,[y1;y2;y1;y2;y2;y1;y1]);
65
66 figure 1;
67 plot(t,y);
68 title('Zeitliches Signal');
69 xlabel('Zeit (s)');
70 ylabel('Amplitude');
71
72 %++++++++++++++++++++++++++++++++++++++++++++++++++
73 % Tiefpassfilterung
74 %++++++++++++++++++++++++++++++++++++++++++++++++++
75
76 y_tfilt=digfilter(20, fg_tiefpass, f_sam, y, 'tiefpass');
77 %y_tfilt=digfilter(150, [115000 125000], f_sam, y, 'bandpass');
78
79 figure;
80 plot(t,y_tfilt);
```

```
81 title('Tiefpass gefiltertes Signal');
82 xlabel('Zeit (s)');
83 ylabel('Amplitude');
84
85 [betrag, phase, f] = spektrum_analyse(y_tfilt,f_sam, 128, 256, 'darstellen');
86
87
88 %++++++++++++++++++++++++++++++++++++++++++++++++++
89 % Reduzierung / ADW
90 %++++++++++++++++++++++++++++++++++++++++++++++++++
91
92 figure;
93
94 [y_red, t_red, f_sam] = reduzierung_adw(y_tfilt, t, f_sam, 4);
95
96 %y_red=y_tfilt;
97 %t_red=t;
98
99 plot(t_red, y_red);
100 title('Signal nach Werte Reduzierung');
101 xlabel('Zeit (s)');
102 ylabel('Amplitude');
103
104
105 %++++++++++++++++++++++++++++++++++++++++++++++++++
106 % Bandpassfilterung
107 %++++++++++++++++++++++++++++++++++++++++++++++++++
108
109 [betrag, phase, f] = spektrum_analyse(y_red, f_sam, 128, 256, 'darstellen');
110
111 for i=1:1:8
112 y_bfilt(i,:)=digfilter(bp_order, [fu_bandpass(i) fo_bandpass(i)], f_sam, y_red, '
        bandpass', 'iir');
113 [betragf(i,:), phasef(i,:), fgf] = spektrum_analyse(y_bfilt(i,:), f_sam, 1,
        length(y_red));
114 end
115
116 pos = [1, 3, 5, 7, 2, 4, 6, 8];
117 figure;
118 for i=1:1:8
119   subplot(4,2,pos(i));
120   stem(fgf,betragf(i,:));
121   tstr = sprintf('KANAL %d', i);
122   title(tstr);
123   xlabel('f (Hz)');
124   ylabel('Amplitude');
125 end
126 figure;
```

```
127 for i=1:1:8
128   subplot(4,2,pos(i));
129   plot(t_red,y_bfilt(i,:));
130   tstr = sprintf('KANAL %d', i);
131   title(tstr);
132   xlabel('Zeit (s)');
133   ylabel('Amplitude');
134 end
135 
136 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 1, 256, 'darstellen');
137 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 512, 256, 'darstellen');
138 %outtable([t_red;y_red], 'testtest.dat');
```

B.2.2. Mit Überschneidung

B.2.2.1. FIR-Filter

Listing B.9: Geometrische Reihe überschnitten FIR gefiltert

```
1  %#########################################################################
2  %
3  % Author:     Norman Schmidt
4  % Date:       30.Juli.2014
5  %
6  % Description:
7  % Simulation geometrische Reihe mit Ueberschneidung und FIR Filter
8  %
9  %#########################################################################
10 clear;
11 clc;
12 close all
13 
14 %++++++++++++++++++++++++++++++++++++++++++++++++++
15 % Initialisierung
16 %++++++++++++++++++++++++++++++++++++++++++++++++++
17 
18 freq=[30000; 34950; 40717; 47435; 55262; 64380; 75003; 87378];
19 
20 tastgrad =0.3;
21 bp_order=92;
22 fg_tiefpass=100000;
23 bp_grenze=2500;
24 
25 freq0 = [ freq(1); freq(3); freq(5); freq(7)];
26 ampl0 = [    1;     1;     1;     1];
27 puls0 = [    1;     1;     1;     1]*tastgrad;
```

```
28
29 freq1 = [ freq(2); freq(4); freq(6); freq(8)];
30 ampl1 = [    1;     1;     1;     1];
31 puls1 = [    1;     1;     1;     1]*tastgrad;
32
33
34 fu_bandpass=freq-bp_grenze;
35 fo_bandpass=freq+bp_grenze;
36
37 %++++++++++++++++++++++++++++++++++++++++++++++++++
38 % Kenndaten
39 %++++++++++++++++++++++++++++++++++++++++++++++++++
40
41 [f_sam, N_abt, fres] = kenndaten(freq, (1024*10^3), 1024, 'abt_statisch', 'stat')
       ;
42
43 %++++++++++++++++++++++++++++++++++++++++++++++++++
44 % Signalgenerator SIG1
45 %++++++++++++++++++++++++++++++++++++++++++++++++++
46
47 [y1, t1]=signalgenerator(ampl0, freq0, f_sam, N_abt, puls0, 128, 'square');
48 %figure 1;
49 %plot(t1,y1);
50
51 %++++++++++++++++++++++++++++++++++++++++++++++++++
52 % Signalgenerator SIG2
53 %++++++++++++++++++++++++++++++++++++++++++++++++++
54
55 [y2, t2]=signalgenerator(ampl1, freq1, f_sam, N_abt, puls1, 128, 'square');
56 %figure 2;
57 %plot(t2,y2);
58
59 %++++++++++++++++++++++++++++++++++++++++++++++++++
60 % Signalfolge
61 %++++++++++++++++++++++++++++++++++++++++++++++++++
62
63 [y, t] = signalfolge(f_sam,[y1;y2;y1;y2;y2;y1;y1]);
64
65 figure 1;
66 plot(t,y);
67 title('Zeitliches Signal');
68 xlabel('Zeit (s)');
69 ylabel('Amplitude');
70
71 %++++++++++++++++++++++++++++++++++++++++++++++++++
72 % Tiefpassfilterung
73 %++++++++++++++++++++++++++++++++++++++++++++++++++
74
```

```
75 y_tfilt=digfilter(20, fg_tiefpass, f_sam, y, 'tiefpass');
76 %y_tfilt=digfilter(150, [115000 125000], f_sam, y, 'bandpass');
77
78 figure 2;
79 plot(t,y_tfilt);
80 title('Tiefpass gefiltertes Signal');
81 xlabel('Zeit (s)');
82 ylabel('Amplitude');
83
84 [betrag, phase, f] = spektrum_analyse(y_tfilt,f_sam, 128, 256, 'darstellen');
85
86
87 %++++++++++++++++++++++++++++++++++++++++++++++++++
88 % Reduzierung / ADW
89 %++++++++++++++++++++++++++++++++++++++++++++++++++
90
91 figure;
92
93 [y_red, t_red, f_sam] = reduzierung_adw(y_tfilt, t, f_sam, 4);
94
95 %y_red=y_tfilt;
96 %t_red=t;
97
98 plot(t_red, y_red);
99 title('Signal nach Werte Reduzierung');
100 xlabel('Zeit (s)');
101 ylabel('Amplitude');
102
103
104 %++++++++++++++++++++++++++++++++++++++++++++++++++
105 % Bandpassfilterung
106 %++++++++++++++++++++++++++++++++++++++++++++++++++
107
108 [betrag, phase, f] = spektrum_analyse(y_red, f_sam, 128, 256, 'darstellen');
109
110 for i=1:1:8
111 y_bfilt(i,:)=digfilter(bp_order, [fu_bandpass(i) fo_bandpass(i)], f_sam, y_red, '
    bandpass', 'fir');
112 [betragf(i,:), phasef(i,:), fgf] = spektrum_analyse(y_bfilt(i,:), f_sam, 1,
    length(y_red));
113 end
114
115 pos = [1, 3, 5, 7, 2, 4, 6, 8];
116 figure;
117 for i=1:1:8
118   subplot(4,2,pos(i));
119   stem(fgf,betragf(i,:));
120   tstr = sprintf('KANAL %d', i);
```

```
121   title(tstr);
122   xlabel('f (Hz)');
123   ylabel('Amplitude');
124 
125 end
126 figure;
127 for i=1:1:8
128   subplot(4,2,pos(i));
129   plot(t_red,y_bfilt(i,:));
130   tstr = sprintf('KANAL %d', i);
131   title(tstr);
132   xlabel('Zeit (s)');
133   ylabel('Amplitude');
134 end
135 
136 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 1, 256, 'darstellen');
137 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 512, 256, 'darstellen');
138 %outtable([t_red;y_red], 'testtest.dat');
```

B.2.2.2. IIR-Filter

Listing B.10: Geometrische Reihe überschnitten IIR gefiltert

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %
3  % Author:     Norman Schmidt
4  % Date:       30.Juli.2014
5  %
6  % Description:
7  % Simulation geometrische Reihe mit Ueberschneidung und IIR Filter
8  %
9  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10 clear;
11 clc;
12 close all
13 
14 %++++++++++++++++++++++++++++++++++++++++++++++++++
15 % Initialisierung
16 %++++++++++++++++++++++++++++++++++++++++++++++++++
17 
18 freq=[30000; 34950; 40717; 47435; 55262; 64380; 75003; 87378];
19 
20 tastgrad =0.3;
21 bp_order=3;
22 fg_tiefpass=100000;
23 bp_grenze=2500;
24 
```

```
25 freq0 = [ freq(1); freq(3); freq(5); freq(7)];
26 ampl0 = [    1;     1;     1;     1];
27 puls0 = [    1;     1;     1;     1]*tastgrad;
28
29 freq1 = [ freq(2); freq(4); freq(6); freq(8)];
30 ampl1 = [    1;     1;     1;     1];
31 puls1 = [    1;     1;     1;     1]*tastgrad;
32
33
34 fu_bandpass=freq-bp_grenze;
35 fo_bandpass=freq+bp_grenze;
36
37
38 %++++++++++++++++++++++++++++++++++++++++++++++++++
39 % Kenndaten
40 %++++++++++++++++++++++++++++++++++++++++++++++++++
41
42 [f_sam, N_abt, fres] = kenndaten(freq, (1024*10^3), 1024, 'abt_statisch', 'stat')
       ;
43
44 %++++++++++++++++++++++++++++++++++++++++++++++++++
45 % Signalgenerator SIG1
46 %++++++++++++++++++++++++++++++++++++++++++++++++++
47
48 [y1, t1]=signalgenerator(ampl0, freq0, f_sam, N_abt, puls0, 128, 'square');
49 %figure 1;
50 %plot(t1,y1);
51
52 %++++++++++++++++++++++++++++++++++++++++++++++++++
53 % Signalgenerator SIG2
54 %++++++++++++++++++++++++++++++++++++++++++++++++++
55
56 [y2, t2]=signalgenerator(ampl1, freq1, f_sam, N_abt, puls1, 128, 'square');
57 %figure 2;
58 %plot(t2,y2);
59
60 %++++++++++++++++++++++++++++++++++++++++++++++++++
61 % Signalfolge
62 %++++++++++++++++++++++++++++++++++++++++++++++++++
63
64 [y, t] = signalfolge(f_sam,[y1;y2;y1;y2;y2;y1;y1]);
65
66 figure 1;
67 plot(t,y);
68 title('Zeitliches Signal');
69 xlabel('Zeit (s)');
70 ylabel('Amplitude');
71
```

```
%++++++++++++++++++++++++++++++++++++++++++++++++++
% Tiefpassfilterung
%++++++++++++++++++++++++++++++++++++++++++++++++++

y_tfilt=digfilter(20, fg_tiefpass, f_sam, y, 'tiefpass');
%y_tfilt=digfilter(150, [115000 125000], f_sam, y, 'bandpass');

figure 2;
plot(t,y_tfilt);
title('Tiefpass gefiltertes Signal');
xlabel('Zeit (s)');
ylabel('Amplitude');

[betrag, phase, f] = spektrum_analyse(y_tfilt,f_sam, 128, 256, 'darstellen');


%++++++++++++++++++++++++++++++++++++++++++++++++++
% Reduzierung / ADW
%++++++++++++++++++++++++++++++++++++++++++++++++++

figure;

[y_red, t_red, f_sam] = reduzierung_adw(y_tfilt, t, f_sam, 4);

%y_red=y_tfilt;
%t_red=t;

plot(t_red, y_red);
title('Signal nach Werte Reduzierung');
xlabel('Zeit (s)');
ylabel('Amplitude');


%++++++++++++++++++++++++++++++++++++++++++++++++++
% Bandpassfilterung
%++++++++++++++++++++++++++++++++++++++++++++++++++

[betrag, phase, f] = spektrum_analyse(y_red, f_sam, 128, 256, 'darstellen');

for i=1:1:8
y_bfilt(i,:)=digfilter(bp_order, [fu_bandpass(i) fo_bandpass(i)], f_sam, y_red, '
    bandpass', 'iir');
[betragf(i,:), phasef(i,:), fgf] = spektrum_analyse(y_bfilt(i,:), f_sam, 1,
    length(y_red));
end

pos = [1, 3, 5, 7, 2, 4, 6, 8];
figure;
```

```
for i=1:1:8
  subplot(4,2,pos(i));
  stem(fgf,betragf(i,:));
  tstr = sprintf('KANAL %d', i);
  title(tstr);
  xlabel('f (Hz)');
  ylabel('Amplitude');

end
figure;
for i=1:1:8
  subplot(4,2,pos(i));
  plot(t_red,y_bfilt(i,:));
  tstr = sprintf('KANAL %d', i);
  title(tstr);
  xlabel('Zeit (s)');
  ylabel('Amplitude');
end

%[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 1, 256, 'darstellen');
%[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 512, 256, 'darstellen');
%outtable([t_red;y_red], 'testtest.dat');
```

B.3. Arithmetrische Reihe

B.3.1. Ohne Überschneidung

B.3.1.1. FIR-Filter

Listing B.11: Arithmetrische Reihe fortlaufend FIR gefiltert

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Author:     Norman Schmidt
% Date:       30.Juli.2014
%
% Description:
% Simulation arithmetrische Reihe ohne Ueberschneidung und FIR Filter
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
clc;
close all

%++++++++++++++++++++++++++++++++++++++++++++++++++
% Initialisierung
%++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
17
18 freq=[30000; 33600; 37200; 40800; 44400; 48000; 51600; 55200];
19
20 tastgrad =0.3;
21 bp_order=130;
22 fg_tiefpass=100000;
23 bp_grenze=1700;
24
25 freq0 = [ freq(1); freq(3); freq(5); freq(7)];
26 ampl0 = [    1;     1;     1;     1];
27 puls0 = [    1;     1;     1;     1]*tastgrad;
28
29 freq1 = [ freq(2); freq(4); freq(6); freq(8)];
30 ampl1 = [    1;     1;     1;     1];
31 puls1 = [    1;     1;     1;     1]*tastgrad;
32
33 fu_bandpass=freq-bp_grenze;
34 fo_bandpass=freq+bp_grenze;
35
36 %++++++++++++++++++++++++++++++++++++++++++++++++++
37 % Kenndaten
38 %++++++++++++++++++++++++++++++++++++++++++++++++++
39
40 [f_sam, N_abt, fres] = kenndaten(freq, (1024*10^3), 1024, 'abt_statisch', 'stat')
       ;
41
42 %++++++++++++++++++++++++++++++++++++++++++++++++++
43 % Signalgenerator SIG1
44 %++++++++++++++++++++++++++++++++++++++++++++++++++
45
46 [y1, t1]=signalgenerator(ampl0, freq0, f_sam, N_abt, puls0, 128, 'square');
47 %figure 1;
48 %plot(t1,y1);
49
50 %++++++++++++++++++++++++++++++++++++++++++++++++++
51 % Signalgenerator SIG2
52 %++++++++++++++++++++++++++++++++++++++++++++++++++
53
54 [y2, t2]=signalgenerator(ampl1, freq1, f_sam, N_abt, puls1, 128, 'square');
55 %figure 2;
56 %plot(t2,y2);
57
58 %++++++++++++++++++++++++++++++++++++++++++++++++++
59 % Signalfolge
60 %++++++++++++++++++++++++++++++++++++++++++++++++++
61
62 [y, t] = signalfolge(f_sam,[y1;y2;y1;y2;y2;y1;y1]);
63
```

```
64 figure 1;
65 plot(t,y);
66 title('Zeitliches Signal');
67 xlabel('Zeit (s)');
68 ylabel('Amplitude');
69
70 %++++++++++++++++++++++++++++++++++++++++++++++++++
71 % Tiefpassfilterung
72 %++++++++++++++++++++++++++++++++++++++++++++++++++
73
74 y_tfilt=digfilter(20, fg_tiefpass, f_sam, y, 'tiefpass');
75 %y_tfilt=digfilter(150, [115000 125000], f_sam, y, 'bandpass');
76
77 figure 2;
78 plot(t,y_tfilt);
79 title('Tiefpass gefiltertes Signal');
80 xlabel('Zeit (s)');
81 ylabel('Amplitude');
82
83 [betrag, phase, f] = spektrum_analyse(y_tfilt,f_sam, 128, 256, 'darstellen');
84
85
86 %++++++++++++++++++++++++++++++++++++++++++++++++++
87 % Reduzierung / ADW
88 %++++++++++++++++++++++++++++++++++++++++++++++++++
89
90 figure;
91
92 [y_red, t_red, f_sam] = reduzierung_adw(y_tfilt, t, f_sam, 4);
93
94 %y_red=y_tfilt;
95 %t_red=t;
96
97 plot(t_red, y_red);
98 title('Signal nach Werte Reduzierung');
99 xlabel('Zeit (s)');
100 ylabel('Amplitude');
101
102
103 %++++++++++++++++++++++++++++++++++++++++++++++++++
104 % Bandpassfilterung
105 %++++++++++++++++++++++++++++++++++++++++++++++++++
106
107 [betrag, phase, f] = spektrum_analyse(y_red, f_sam, 128, 256, 'darstellen');
108
109 for i=1:1:8
110 y_bfilt(i,:)=digfilter(bp_order, [fu_bandpass(i) fo_bandpass(i)], f_sam, y_red, '
    bandpass', 'fir');
```

```
111 [betragf(i,:), phasef(i,:), fgf] = spektrum_analyse(y_bfilt(i,:), f_sam, 1,
        length(y_red));
112 end
113
114 pos = [1, 3, 5, 7, 2, 4, 6, 8];
115 figure;
116 for i=1:1:8
117   subplot(4,2,pos(i));
118   stem(fgf,betragf(i,:));
119   tstr = sprintf('KANAL %d', i);
120   title(tstr);
121   xlabel('f (Hz)');
122   ylabel('Amplitude');
123
124 end
125 figure;
126 for i=1:1:8
127   subplot(4,2,pos(i));
128   plot(t_red,y_bfilt(i,:));
129   tstr = sprintf('KANAL %d', i);
130   title(tstr);
131   xlabel('Zeit (s)');
132   ylabel('Amplitude');
133 end
134
135 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 1, 256, 'darstellen');
136 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 512, 256, 'darstellen');
137 %outtable([t_red;y_red], 'testtest.dat');
```

B.3.1.2. IIR-Filter

Listing B.12: Arithmetrische Reihe fortlaufend IIR gefiltert

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %
3  % Author:     Norman Schmidt
4  % Date:       30.Juli.2014
5  %
6  % Description:
7  % Simulation arithmetrische Reihe ohne Ueberschneidung und IIR Filter
8  %
9  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10 clear;
11 clc;
12 close all
13
14 %++++++++++++++++++++++++++++++++++++++++++++++++
```

```
15 % Initialisierung
16 %++++++++++++++++++++++++++++++++++++++++++++++++++
17
18 freq=[30000; 33600; 37200; 40800; 44400; 48000; 51600; 55200];
19
20 tastgrad =0.3;
21 bp_order=4;
22 fg_tiefpass=100000;
23 bp_grenze=1700;
24
25 freq0 = [ freq(1); freq(3); freq(5); freq(7)];
26 ampl0 = [    1;     1;     1;     1];
27 puls0 = [    1;     1;     1;     1]*tastgrad;
28
29 freq1 = [ freq(2); freq(4); freq(6); freq(8)];
30 ampl1 = [    1;     1;     1;     1];
31 puls1 = [    1;     1;     1;     1]*tastgrad;
32
33 fu_bandpass=freq-bp_grenze;
34 fo_bandpass=freq+bp_grenze;
35
36 %++++++++++++++++++++++++++++++++++++++++++++++++++
37 % Kenndaten
38 %++++++++++++++++++++++++++++++++++++++++++++++++++
39
40 [f_sam, N_abt, fres] = kenndaten(freq, (1024*10^3), 1024, 'abt_statisch', 'stat')
       ;
41
42 %++++++++++++++++++++++++++++++++++++++++++++++++++
43 % Signalgenerator SIG1
44 %++++++++++++++++++++++++++++++++++++++++++++++++++
45
46 [y1, t1]=signalgenerator(ampl0, freq0, f_sam, N_abt, puls0, 128, 'square');
47 %figure 1;
48 %plot(t1,y1);
49
50 %++++++++++++++++++++++++++++++++++++++++++++++++++
51 % Signalgenerator SIG2
52 %++++++++++++++++++++++++++++++++++++++++++++++++++
53
54 [y2, t2]=signalgenerator(ampl1, freq1, f_sam, N_abt, puls1, 128, 'square');
55 %figure 2;
56 %plot(t2,y2);
57
58 %++++++++++++++++++++++++++++++++++++++++++++++++++
59 % Signalfolge
60 %++++++++++++++++++++++++++++++++++++++++++++++++++
61
```

```
62 [y, t] = signalfolge(f_sam,[y1;y2;y1;y2;y2;y1;y1]);
63
64 figure 1;
65 plot(t,y);
66 title('Zeitliches Signal');
67 xlabel('Zeit (s)');
68 ylabel('Amplitude');
69
70 %++++++++++++++++++++++++++++++++++++++++++++++++++
71 % Tiefpassfilterung
72 %++++++++++++++++++++++++++++++++++++++++++++++++++
73
74 y_tfilt=digfilter(20, fg_tiefpass, f_sam, y, 'tiefpass');
75 %y_tfilt=digfilter(150, [115000 125000], f_sam, y, 'bandpass');
76
77 figure 2;
78 plot(t,y_tfilt);
79 title('Tiefpass gefiltertes Signal');
80 xlabel('Zeit (s)');
81 ylabel('Amplitude');
82
83 [betrag, phase, f] = spektrum_analyse(y_tfilt,f_sam, 128, 256, 'darstellen');
84
85
86 %++++++++++++++++++++++++++++++++++++++++++++++++++
87 % Reduzierung / ADW
88 %++++++++++++++++++++++++++++++++++++++++++++++++++
89
90 figure;
91
92 [y_red, t_red, f_sam] = reduzierung_adw(y_tfilt, t, f_sam, 4);
93
94 %y_red=y_tfilt;
95 %t_red=t;
96
97 plot(t_red, y_red);
98 title('Signal nach Werte Reduzierung');
99 xlabel('Zeit (s)');
100 ylabel('Amplitude');
101
102
103 %++++++++++++++++++++++++++++++++++++++++++++++++++
104 % Bandpassfilterung
105 %++++++++++++++++++++++++++++++++++++++++++++++++++
106
107 [betrag, phase, f] = spektrum_analyse(y_red, f_sam, 128, 256, 'darstellen');
108
109 for i=1:1:8
```

```
110 y_bfilt(i,:)=digfilter(bp_order, [fu_bandpass(i) fo_bandpass(i)], f_sam, y_red, '
        bandpass', 'iir');
111 [betragf(i,:), phasef(i,:), fgf] = spektrum_analyse(y_bfilt(i,:), f_sam, 1,
        length(y_red));
112 end
113
114 pos = [1, 3, 5, 7, 2, 4, 6, 8];
115 figure;
116 for i=1:1:8
117   subplot(4,2,pos(i));
118   stem(fgf,betragf(i,:));
119   tstr = sprintf('KANAL %d', i);
120   title(tstr);
121   xlabel('f (Hz)');
122   ylabel('Amplitude');
123
124 end
125 figure;
126 for i=1:1:8
127   subplot(4,2,pos(i));
128   plot(t_red,y_bfilt(i,:));
129   tstr = sprintf('KANAL %d', i);
130   title(tstr);
131   xlabel('Zeit (s)');
132   ylabel('Amplitude');
133 end
134
135 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 1, 256, 'darstellen');
136 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 512, 256, 'darstellen');
137 %outtable([t_red;y_red], 'testtest.dat');
```

B.3.2. Mit Überschneidung

B.3.2.1. FIR-Filter

Listing B.13: Arithmetrische Reihe überschnitten FIR gefiltert

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %
3 % Author:     Norman Schmidt
4 % Date:       30.Juli.2014
5 %
6 % Description:
7 % Simulation geometrische Reihe mit Ueberschneidung und FIR Filter
8 %
9 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
10 clear;
11 clc;
12 close all
13
14 %++++++++++++++++++++++++++++++++++++++++++++++++++
15 % Initialisierung
16 %++++++++++++++++++++++++++++++++++++++++++++++++++
17
18 freq=[30000; 36750; 43500; 50250; 57000; 63750; 70500; 77250];
19
20 tastgrad =0.3;
21 bp_order=110;
22 fg_tiefpass=100000;
23 bp_grenze=2000;
24
25
26
27 freq0 = [ freq(1); freq(3); freq(5); freq(7)];
28 ampl0 = [    1;     1;     1;     1];
29 puls0 = [    1;     1;     1;     1]*tastgrad;
30
31 freq1 = [ freq(2); freq(4); freq(6); freq(8)];
32 ampl1 = [    1;     1;     1;     1];
33 puls1 = [    1;     1;     1;     1]*tastgrad;
34
35
36 fu_bandpass=freq-bp_grenze;
37 fo_bandpass=freq+bp_grenze;
38
39 %++++++++++++++++++++++++++++++++++++++++++++++++++
40 % Kenndaten
41 %++++++++++++++++++++++++++++++++++++++++++++++++++
42
43 [f_sam, N_abt, fres] = kenndaten(freq, (1024*10^3), 1024, 'abt_statisch', 'stat')
       ;
44
45 %++++++++++++++++++++++++++++++++++++++++++++++++++
46 % Signalgenerator SIG1
47 %++++++++++++++++++++++++++++++++++++++++++++++++++
48
49 [y1, t1]=signalgenerator(ampl0, freq0, f_sam, N_abt, puls0, 128, 'square');
50 %figure 1;
51 %plot(t1,y1);
52
53 %++++++++++++++++++++++++++++++++++++++++++++++++++
54 % Signalgenerator SIG2
55 %++++++++++++++++++++++++++++++++++++++++++++++++++
56
```

```
57 [y2, t2]=signalgenerator(ampl1, freq1, f_sam, N_abt, puls1, 128, 'square');
58 %figure 2;
59 %plot(t2,y2);
60
61 %++++++++++++++++++++++++++++++++++++++++++++++++++
62 % Signalfolge
63 %++++++++++++++++++++++++++++++++++++++++++++++++++
64
65 [y, t] = signalfolge(f_sam,[y1;y2;y1;y2;y2;y1;y1]);
66
67 figure 1;
68 plot(t,y);
69 title('Zeitliches Signal');
70 xlabel('Zeit (s)');
71 ylabel('Amplitude');
72
73 %++++++++++++++++++++++++++++++++++++++++++++++++++
74 % Tiefpassfilterung
75 %++++++++++++++++++++++++++++++++++++++++++++++++++
76
77 y_tfilt=digfilter(20, fg_tiefpass, f_sam, y, 'tiefpass');
78 %y_tfilt=digfilter(150, [115000 125000], f_sam, y, 'bandpass');
79
80 figure 2;
81 plot(t,y_tfilt);
82 title('Tiefpass gefiltertes Signal');
83 xlabel('Zeit (s)');
84 ylabel('Amplitude');
85
86 [betrag, phase, f] = spektrum_analyse(y_tfilt,f_sam, 128, 256, 'darstellen');
87
88
89 %++++++++++++++++++++++++++++++++++++++++++++++++++
90 % Reduzierung / ADW
91 %++++++++++++++++++++++++++++++++++++++++++++++++++
92
93 figure;
94
95 [y_red, t_red, f_sam] = reduzierung_adw(y_tfilt, t, f_sam, 4);
96
97 %y_red=y_tfilt;
98 %t_red=t;
99
100 plot(t_red, y_red);
101 title('Signal nach Werte Reduzierung');
102 xlabel('Zeit (s)');
103 ylabel('Amplitude');
104
```

```
105
106 %++++++++++++++++++++++++++++++++++++++++++++++++
107 % Bandpassfilterung
108 %++++++++++++++++++++++++++++++++++++++++++++++++
109
110 [betrag, phase, f] = spektrum_analyse(y_red, f_sam, 128, 256, 'darstellen');
111
112 for i=1:1:8
113 y_bfilt(i,:)=digfilter(bp_order, [fu_bandpass(i) fo_bandpass(i)], f_sam, y_red, '
        bandpass', 'fir');
114 [betragf(i,:), phasef(i,:), fgf] = spektrum_analyse(y_bfilt(i,:), f_sam, 1,
        length(y_red));
115 end
116
117 pos = [1, 3, 5, 7, 2, 4, 6, 8];
118 figure;
119 for i=1:1:8
120   subplot(4,2,pos(i));
121   stem(fgf,betragf(i,:));
122   tstr = sprintf('KANAL %d', i);
123   title(tstr);
124   xlabel('f (Hz)');
125   ylabel('Amplitude');
126
127 end
128 figure;
129 for i=1:1:8
130   subplot(4,2,pos(i));
131   plot(t_red,y_bfilt(i,:));
132   tstr = sprintf('KANAL %d', i);
133   title(tstr);
134   xlabel('Zeit (s)');
135   ylabel('Amplitude');
136 end
137
138 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 1, 256, 'darstellen');
139 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 512, 256, 'darstellen');
140 %outtable([t_red;y_red], 'testtest.dat');
```

B.3.2.2. IIR-Filter

Listing B.14: Arithmetrische Reihe überschnitten IIR gefiltert

```
1 %#########################################################################
2 %
3 % Author:     Norman Schmidt
4 % Date:       30.Juli.2014
```

```
5  %
6  % Description:
7  % Simulation geometrische Reihe mit Ueberschneidung und IIR Filter
8  %
9  %#########################################################################
10 clear;
11 clc;
12 close all
13
14 %++++++++++++++++++++++++++++++++++++++++++++++++++
15 % Initialisierung
16 %++++++++++++++++++++++++++++++++++++++++++++++++++
17
18 freq=[30000; 36750; 43500; 50250; 57000; 63750; 70500; 77250];
19
20 tastgrad =0.3;
21 bp_order=4;
22 fg_tiefpass=100000;
23 bp_grenze=2000;
24
25
26
27 freq0 = [ freq(1); freq(3); freq(5); freq(7)];
28 ampl0 = [    1;     1;     1;     1];
29 puls0 = [    1;     1;     1;     1]*tastgrad;
30
31 freq1 = [ freq(2); freq(4); freq(6); freq(8)];
32 ampl1 = [    1;     1;     1;     1];
33 puls1 = [    1;     1;     1;     1]*tastgrad;
34
35 fu_bandpass=freq-bp_grenze;
36 fo_bandpass=freq+bp_grenze;
37
38
39 %++++++++++++++++++++++++++++++++++++++++++++++++++
40 % Kenndaten
41 %++++++++++++++++++++++++++++++++++++++++++++++++++
42
43 [f_sam, N_abt, fres] = kenndaten(freq, (1024*10^3), 1024, 'abt_statisch', 'stat')
       ;
44
45 %++++++++++++++++++++++++++++++++++++++++++++++++++
46 % Signalgenerator SIG1
47 %++++++++++++++++++++++++++++++++++++++++++++++++++
48
49 [y1, t1]=signalgenerator(ampl0, freq0, f_sam, N_abt, puls0, 128, 'square');
50 %figure 1;
51 %plot(t1,y1);
```

```
52
53 %++++++++++++++++++++++++++++++++++++++++++++++++++
54 % Signalgenerator SIG2
55 %++++++++++++++++++++++++++++++++++++++++++++++++++
56
57 [y2, t2]=signalgenerator(ampl1, freq1, f_sam, N_abt, puls1, 128, 'square');
58 %figure 2;
59 %plot(t2,y2);
60
61 %++++++++++++++++++++++++++++++++++++++++++++++++++
62 % Signalfolge
63 %++++++++++++++++++++++++++++++++++++++++++++++++++
64
65 [y, t] = signalfolge(f_sam,[y1;y2;y1;y2;y2;y1;y1]);
66
67 figure 1;
68 plot(t,y);
69 title('Zeitliches Signal');
70 xlabel('Zeit (s)');
71 ylabel('Amplitude');
72
73 %++++++++++++++++++++++++++++++++++++++++++++++++++
74 % Tiefpassfilterung
75 %++++++++++++++++++++++++++++++++++++++++++++++++++
76
77 y_tfilt=digfilter(20, fg_tiefpass, f_sam, y, 'tiefpass');
78 %y_tfilt=digfilter(150, [115000 125000], f_sam, y, 'bandpass');
79
80 figure 2;
81 plot(t,y_tfilt);
82 title('Tiefpass gefiltertes Signal');
83 xlabel('Zeit (s)');
84 ylabel('Amplitude');
85
86 [betrag, phase, f] = spektrum_analyse(y_tfilt,f_sam, 128, 256, 'darstellen');
87
88
89 %++++++++++++++++++++++++++++++++++++++++++++++++++
90 % Reduzierung / ADW
91 %++++++++++++++++++++++++++++++++++++++++++++++++++
92
93 figure;
94
95 [y_red, t_red, f_sam] = reduzierung_adw(y_tfilt, t, f_sam, 4);
96
97 %y_red=y_tfilt;
98 %t_red=t;
99
```

```
100 plot(t_red, y_red);
101 title('Signal nach Werte Reduzierung');
102 xlabel('Zeit (s)');
103 ylabel('Amplitude');
104
105
106 %++++++++++++++++++++++++++++++++++++++++++++++++
107 % Bandpassfilterung
108 %++++++++++++++++++++++++++++++++++++++++++++++++
109
110 [betrag, phase, f] = spektrum_analyse(y_red, f_sam, 128, 256, 'darstellen');
111
112 for i=1:1:8
113 y_bfilt(i,:)=digfilter(bp_order, [fu_bandpass(i) fo_bandpass(i)], f_sam, y_red, '
        bandpass', 'iir');
114 [betragf(i,:), phasef(i,:), fgf] = spektrum_analyse(y_bfilt(i,:), f_sam, 1,
        length(y_red));
115 end
116
117 pos = [1, 3, 5, 7, 2, 4, 6, 8];
118 figure;
119 for i=1:1:8
120   subplot(4,2,pos(i));
121   stem(fgf,betragf(i,:));
122   tstr = sprintf('KANAL %d', i);
123   title(tstr);
124   xlabel('f (Hz)');
125   ylabel('Amplitude');
126
127 end
128 figure;
129 for i=1:1:8
130   subplot(4,2,pos(i));
131   plot(t_red,y_bfilt(i,:));
132   tstr = sprintf('KANAL %d', i);
133   title(tstr);
134   xlabel('Zeit (s)');
135   ylabel('Amplitude');
136 end
137
138 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 1, 256, 'darstellen');
139 %[betrag, phase, f] = spektrum_analyse(y_bfilt,f_sam, 512, 256, 'darstellen');
140 %outtable([t_red;y_red], 'testtest.dat');
```

B.4. DFT-Reihe

B.4.1. Fortlaufende Reihe

Listing B.15: DFT-Reihe fortlaufend

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Author:     Norman Schmidt
% Date:       30.Juli.2014
%
% Description:
% Simulation DFT-Reihe fortlaufend
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
clc;
close all

%++++++++++++++++++++++++++++++++++++++++++++++++++
% Initialisierung
%++++++++++++++++++++++++++++++++++++++++++++++++++

freq = [ 32000; 36000; 40000; 44000; 48000; 52000; 56000; 60000]+200;

tastgrad =0.3;
fg_tiefpass=100000;

freq0 = [ freq(1); freq(3); freq(5); freq(7)];
ampl0 = [    1;     1;     1;     1];
puls0 = [    1;     1;     1;     1]*tastgrad;

freq1 = [ freq(2); freq(4); freq(6); freq(8)];
ampl1 = [    1;     1;     1;     1];
puls1 = [    1;     1;     1;     1]*tastgrad;

%++++++++++++++++++++++++++++++++++++++++++++++++++
% Kenndaten
%++++++++++++++++++++++++++++++++++++++++++++++++++

[f_sam, N_abt, fres] = kenndaten(freq, (1024*10^3), 256, 'abt_statisch', 'stat');

%++++++++++++++++++++++++++++++++++++++++++++++++++
% Signalgenerator SIG1
%++++++++++++++++++++++++++++++++++++++++++++++++++

[y1, t1]=signalgenerator(ampl0, freq0, f_sam, N_abt, puls0, 128, 'square');
%figure 1;
```

```
%plot(t1,y1);

%++++++++++++++++++++++++++++++++++++++++++++++++++
% Signalgenerator SIG2
%++++++++++++++++++++++++++++++++++++++++++++++++++

[y2, t2]=signalgenerator(ampl1, freq1, f_sam, N_abt, puls1, 128, 'square');
%figure 2;
%plot(t2,y2);

%++++++++++++++++++++++++++++++++++++++++++++++++++
% Signalfolge
%++++++++++++++++++++++++++++++++++++++++++++++++++

[y, t] = signalfolge(f_sam,[y1;y2;y1;y2;y1;y2;y1;y2;y2;y1;y1]);

figure 1;
plot(t,y);

%++++++++++++++++++++++++++++++++++++++++++++++++++
% Tiefpassfilterung
%++++++++++++++++++++++++++++++++++++++++++++++++++

y_tfilt=digfilter(20, fg_tiefpass, f_sam, y, 'tiefpass');
%y_tfilt=digfilter(150, [115000 125000], f_sam, y, 'bandpass');

figure 2;
plot(t,y_tfilt);
[betrag, phase, f] = spektrum_analyse(y_tfilt,f_sam, 128, 256, 'darstellen');

%++++++++++++++++++++++++++++++++++++++++++++++++++
% Reduzierung / ADW
%++++++++++++++++++++++++++++++++++++++++++++++++++

figure;
[y_red, t_red, f_sam] = reduzierung_adw(y_tfilt, t, f_sam, 4);

%y_red=y_tfilt;
%t_red=t;

plot(t_red, y_red);

%++++++++++++++++++++++++++++++++++++++++++++++++++
% Spektralanalyse und Datenauswertung
%++++++++++++++++++++++++++++++++++++++++++++++++++

h=1;
for i=1:64:length(y_red)
```

```
%disp(i); disp(i+64);
[betragt(h,:), phaset(h,:), f] = spektrum_analyse(y_red, f_sam, i, 64);
h=h+1;
end

pos = [1, 3, 5, 7, 2, 4, 6, 8];
op=[ 9, 10, 11, 12, 13, 14, 15, 16];
figure;
for i=1:1:8
  out = [];
  for u=1:1:11
    if betragt(u,op(i)) > 10
      hjk=ones(1,64);
    else
      hjk=zeros(1,64);
    end
    out=[out hjk];
  end
  subplot(4,2,pos(i));
  plot(t_red,out);
  tstr = sprintf('KANAL %d', i);
  title(tstr);
  axis([0 0.003 -0.1 1.2]);
end

%outtable([t_red;y_red], 'testtest.dat');
```

B.4.2. Primzahlen Reihe

Listing B.16: DFT-Reihe aus Primzahlen

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Author:     Norman Schmidt
% Date:       30.Juli.2014
%
% Description:
% Simulation DFT-Reihe mit Primzahlen
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
clc;
close all

%++++++++++++++++++++++++++++++++++++++++++++++++
% Initialisierung
```

```
16 %++++++++++++++++++++++++++++++++++++++++++++++++++
17
18 %freq = [ 2; 3; 5; 7; 11; 13; 17; 19]*4000;
19 freq = [ 8000, 12000, 20000, 28000, 44000, 52000, 68000, 76000];
20
21 tastgrad =0.3;
22 fg_tiefpass=100000;
23
24 freq0 = [ freq(1); freq(3); freq(5); freq(7)];
25 ampl0 = [    1;     1;     1;     1];
26 puls0 = [    1;     1;     1;     1]*tastgrad;
27
28 freq1 = [ freq(2); freq(4); freq(6); freq(8)];
29 ampl1 = [    1;     1;     1;     1];
30 puls1 = [    1;     1;     1;     1]*tastgrad;
31
32 %++++++++++++++++++++++++++++++++++++++++++++++++++
33 % Kenndaten
34 %++++++++++++++++++++++++++++++++++++++++++++++++++
35
36 [f_sam, N_abt, fres] = kenndaten(freq, (1024*10^3), 256, 'abt_statisch', 'stat');
37
38 %++++++++++++++++++++++++++++++++++++++++++++++++++
39 % Signalgenerator SIG1
40 %++++++++++++++++++++++++++++++++++++++++++++++++++
41
42 [y1, t1]=signalgenerator(ampl0, freq0, f_sam, N_abt, puls0, 128, 'square');
43 %figure 1;
44 %plot(t1,y1);
45
46 %++++++++++++++++++++++++++++++++++++++++++++++++++
47 % Signalgenerator SIG2
48 %++++++++++++++++++++++++++++++++++++++++++++++++++
49
50 [y2, t2]=signalgenerator(ampl1, freq1, f_sam, N_abt, puls1, 128, 'square');
51 %figure 2;
52 %plot(t2,y2);
53
54 %++++++++++++++++++++++++++++++++++++++++++++++++++
55 % Signalfolge
56 %++++++++++++++++++++++++++++++++++++++++++++++++++
57
58 [y, t] = signalfolge(f_sam,[y1;y2;y1;y2;y1;y2;y1;y2;y2;y1;y1]);
59
60 figure 1;
61 plot(t,y);
62
63 %++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
64 % Tiefpassfilterung
65 %++++++++++++++++++++++++++++++++++++++++++++++++++
66
67 y_tfilt=digfilter(20, fg_tiefpass, f_sam, y, 'tiefpass');
68 %y_tfilt=digfilter(150, [115000 125000], f_sam, y, 'bandpass');
69
70 figure 2;
71 plot(t,y_tfilt);
72 [betrag, phase, f] = spektrum_analyse(y_tfilt,f_sam, 128, 256, 'darstellen');
73
74 %++++++++++++++++++++++++++++++++++++++++++++++++++
75 % Reduzierung / ADW
76 %++++++++++++++++++++++++++++++++++++++++++++++++++
77
78 figure;
79 [y_red, t_red, f_sam] = reduzierung_adw(y_tfilt, t, f_sam, 4);
80
81 %y_red=y_tfilt;
82 %t_red=t;
83
84 plot(t_red, y_red);
85
86 %++++++++++++++++++++++++++++++++++++++++++++++++++
87 % Spektralanalyse und Datenauswertung
88 %++++++++++++++++++++++++++++++++++++++++++++++++++
89
90 h=1;
91 for i=1:64:length(y_red)
92 %disp(i); disp(i+64);
93 [betragt(h,:), phaset(h,:), f] = spektrum_analyse(y_red, f_sam, i, 64, '
       darstellen');
94 h=h+1;
95 end
96
97 pos = [1, 3, 5, 7, 2, 4, 6, 8];
98 op=[ 2; 3; 5; 7; 11; 13; 17; 19]+1;
99 figure;
100 for i=1:1:8
101   out = [];
102   for u=1:1:11
103     if betragt(u,op(i)) > 10
104       hjk=ones(1,64);
105     else
106       hjk=zeros(1,64);
107     end
108     out=[out hjk];
109   end
110   subplot(4,2,pos(i));
```

```
111   plot(t_red,out);
112   tstr = sprintf('KANAL %d', i);
113   title(tstr);
114   axis([0 0.003 -0.1 1.2]);
115 end
116
117 %outtable([t_red;y_red], 'testtest.dat');
```