Graz University of Technology

Toningenieur Projekt

Digitale Audiotechnik Domonstrations-Software

durchgeführt am
Institut für *Signal Processing and Speech Communications*
Technische Universität Graz, Österreich

von
Marian Forster, BSc
1031275

Betreuer:
DI Dr. Gerhard Graber

Graz, März 2016

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am (Unterschrift)

Abstract

Digital technologies in audio engineering are pretty much standard nowadays. However, the steps involved in a digital signal chain are complex and often not easy to understand. In this thesis a MATLAB based simulation software was developed to show the topics quantization, frequency-aliasing, sample-jitter, granular error, dithering, sample-rate-conversion and many more. The GUI yields the opportunity to quantize different signal shapes, as well as audio files with a parameterizable analog-to-digital converter and simulate behaviors of a digital signal path. The results can be plotted with many different options and played back. The software will be used by students in the course *Nachrichtentechnik Labor* for a deeper understanding of digital (audio-) signal processing.

Zusammenfassung

Die digitale Audiotechnik hat mittlerweile in fast allen Belangen, analoge Medien abgelöst. Die Vorgänge in der Signalkette sind jedoch komplex und oft nicht sehr einfach zu verstehen. Im Zuge dieser Arbeit wurde eine auf MATLAB basierte Software entwickelt, welche die Themen Quantisierung, Frequenz-Aliasing, Sample-Jitter, Granularrauschen, Dithering, Sample-Rate-Conversion und viele mehr, veranschaulicht. Das Programm bietet die Möglichkeit, verschiedene Signalformen mittels eines Analaog/Digital Wandlers zu quantisieren und diverse Vorgänge im Signalweg zu simulieren. Es stehen umfangreiche Plot-Optionen sowie die Möglichkeit einer akustischen Ausgabe zur Verfügung. Die Software soll Studierenden helfen, ihr Verständnis der digitale (Audio-) Signalverarbeitung zu vertiefen.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Im Zuge einer vorangegangenen Projektarbeit ([1] Störverhalten, Franz Zotter 2002) wurde ein MATLAB basiertes Programm, zur Demonstration verschiedener Effekte der digitalen Audiotechnik erstellt. In dieser Software wurden die Auswirkungen von Aliasing, Quantisierung, Jitter und Dither anhand verschiedener Testsignale gezeigt. Zahlreiche Einstellmöglichkeiten boten dem Benutzer die Möglichkeit, digitale Audiotechnik durch visuelle und akustische Darstellung besser zu verstehen. Das Programm wurde über 10 Jahre in der Laborübung sowie in diversen Vorlesung zu Demonstrationszwecken verwendet.

Neuere MATLAB bzw. Betriebssystem Versionen erschweren allerdings seit einigen Jahren einen unkomplizierten Umgang mit der Software. Deshalb wurde im Zuge dieser Arbeit eine aktualisierte Version der Demonstrationssoftware programmiert. Der große Fortschritt in der Computertechnik sowie Neuerungen in MATLAB legen es nahe, das Programm um einige Funktionen zu erweitern. Im Folgenden werden die alten und neuen Spezifikationen der Software beschrieben.

1.2 Spezifikation

Es soll möglich sein verschiedene deterministische, als auch statistische Signale zu generieren. Diese Signale durchlaufen anschließend verschiedene Verarbeitungsstufen (Jitter, Dither, Sample Rate Converter, Analog-Digitalwandler) welche das Signal modifizieren. Zahlreiche Anzeigeoptionen (Zeitbereich, Frequenzbereich, Korrelation, Histogramm,...) ermöglichen es dem Anwender die Auswirkungen näher zu

untersuchen.

Die Bedienung soll intuitiv sein und Speicher sowie Rechenleistung des Computers sollen sparsam eingesetzt werden. Die Auswahl des bevorzugten Audiogerätes soll dem Benutzer möglich sein, außerdem soll die Software sowohl auf Windows, als auch auch Macintosh Geräten lauffähig sein.

1.2.1 Testsignale

Folgende Signale sollen zur Verfügung stehen:

- Sinus
- Sägezahn
- Rechteck
- Rauschen (weiß)
- Sinus Sweep (linear/exponentiell)
- Impulszug
- Audiodatei (Wave Format)

Die Amplitude (in dB), Frequenz (in Hz) sowie die Dauer (in Anzahl der Perioden oder Sekunden) soll frei wählbar sein.

1.2.2 Modifizieren der Testsignale

Die Testsignale sollen verschiedene, in der digitalen Signalverarbeitung vorkommende Prozesse durchlaufen, welche anschließende visuell und akustisch ausgegeben werden:

- **Aliasing**: Durch die Wahl einer zu hohen Frequenz soll es möglich sein, Alias-Frequenzen zu erzeugen.
- **Quantisierung**: Ein Analog-Digitalwandler soll das Signal auf eine beliebige Abtastrate sowie Auflösung quantisieren.
- **Jitter**: Das Hinzufügen von Sample-Jitter soll wahlweise möglich sein. Die Frequenz, Amplitude und Signalform soll einstellbar sein. Jitter soll auch auf beliebige Audiodateien anwendbar sein.
- **Dither**: Um die Korrelation des Quantisierungsrauschens zu reduzieren, soll einfaches Dithering möglich sein. Sowohl Amplitude, als auch das Spektrum des Rauschens (weiß oder gefiltert) soll wählbar sein.
- **Sample Rate Converter**: Eine Abtastratenkonversion auf eine beliebige Abtastrate soll mit und ohne anti-alias Filter auswählbar sein.

1.2.3 Signal Darstellungen

Nach Generierung der Testsignale, sollen zahlreiche Ausgabeoptionen (Plots) zur Verfügung stehen:

- **Zeitbereich**: Analog- und Digital-Signal, Quantisierungsrauschen.
- **Frequenzbereich**: Analog- und Digital-Signal, Dithersignal.
- **Leistungsdichte Spektrum**: Analog-Signal, Quantisierungsrauschen.
- **Korrelation**: Quantisierungsrauschen, Dithersignal.
- **Histogramm**: Analog-Signal.
- **Spektrogramm**: Analog- und Digital-Signal.
- **Spannung und Leistung**: Zeit- und Frequenzbereich.
- **Terzband Signal/Rauschabstand**: Analog-Signal und Quantisierungsrauschen im Frequenzbereich mit Terzbandfilter.

Die Anzeigen sollen über diverse Zoom- und Scroll-Funktionen verfügen, außerdem sollen die FFT Parameter (Fensterfunktion, Auflösung, lineare oder logarithmische Ausgabe) einstellbar sein.

1.2.4 Signal Wiedergabe

Ein einfacher Player soll die generierten Signale über ein Audiogerät abspielen. Der Benutzer soll zwischen Analog-, Digital-Signal und Quantisierungsrauschen (original Pegel oder normalisiert) auswählen können. Während der Wiedergabe soll eine Echtzeit-FFT berechnet und angezeigt werden. Wenn mehrere Audiogeräte installiert sind, soll durch den Benutzer eines ausgewählt werden.

1.2.5 Export von Diagrammen

Um die Erstellung von Laborprotokollen für die Studierenden zu vereinfachen, sollen sämtliche Diagramme in unterschiedlichen Grafikformaten (PDF, PNG, JPG, EPS) exportiert werden können. Alle Einstellungen sollen als Text auf der Grafik mitgespeichert werden.

1.2.6 Plattform Unabhängigkeit

Die Software soll sowohl auf Windows als auch auf Apple Rechnern problemlos laufen. Die Benutzeroberfläche muss so skaliert werden, damit alle Bedienelemente korrekt dargestellt werden. Auch die Verwendung von zusätzlichen MATLAB-Toolboxen soll vermieden werden.

2 Software

Die Software wurde so modular als möglich entwickelt, um einzelne Komponenten leicht testen, warten oder durch neu Versionen ersetzen zu können. Beispielsweise könnte der Signalgenerator um weitere Signalformen erweitert werden ohne die Programmstruktur verändern zu müssen. Im Folgenden werden die einzelnen Module (wie z.B. Signalgenerator, A/D-Wandler, Sample Rate Converter,..) im Detail beschrieben.

Die Architektur der Software folgt dem Blockdiagramm in Abb. 2.1. Es wurde ein Signalverlauf gewählt, der einer Hardware-Umsetzung nachempfunden ist. Zu Beginn der Signalkette steht wahlweise ein Signalgenerator oder eine Audiodatei im Wave-Format. Das Eingangssignal $x(t)$ wird anschließend einem Analog/Digital-Wandler (ADC) zugeführt, der es mit einstellbarer Auflösung und Abtastrate digitalisiert. An dieser Stelle kann wahlweise Dither und Jitter zum Signal hinzugefügt werden. Anschließend stehen zahlreiche Optionen zur Analyse und Wiedergabe des ADC-Ausgangs zur Verfügung.

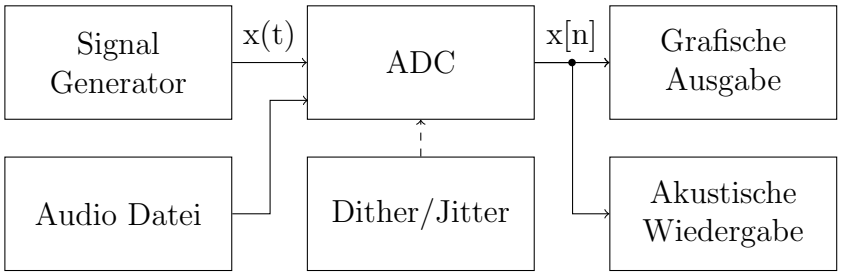


Abbildung 2.1: Softwarearchitektur

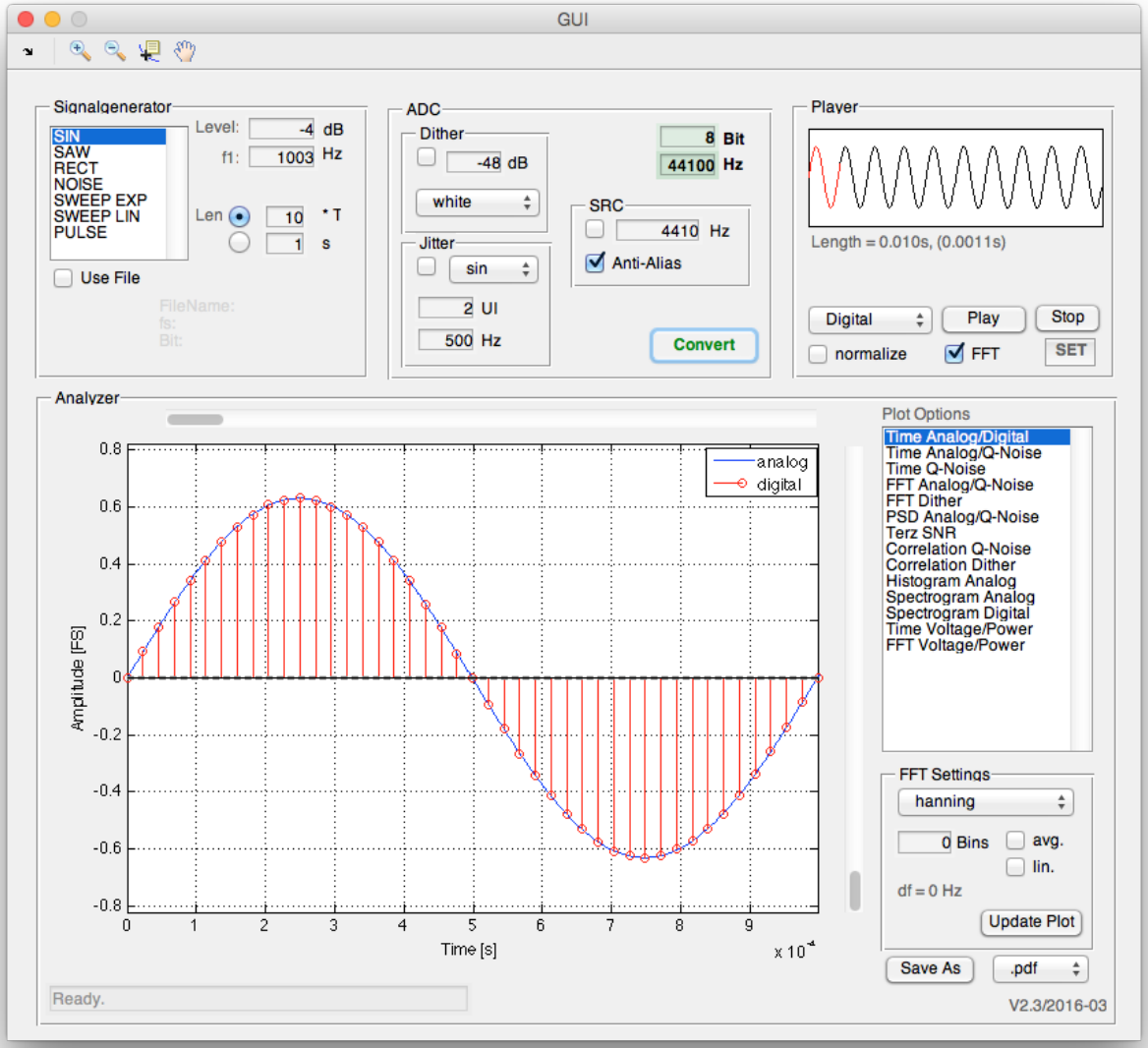


Abbildung 2.2: Digitale Audiotechnik Demonstrations Software

2.1 Aufbau eines MATLAB GUI

In MATLAB (2012b) besteht ein GUI (*graphical user interface*) grundsätzlich aus zwei Dateien. In der `GUI.fig` Datei sind sämtliche Informationen über die grafische Oberfläche und ihre Bedienelemente (Position, Farbe, Initialwert, Tag[1],..) abgelegt. In der `GUI.m` Datei befindet sich der eigentliche Programmcode, welcher bei einer Benutzerinteraktion ausgeführt wird. Zum Erstellen des GUI stellt MATLAB ein Tool bereit: `guide`. Nachdem MATLAB auf *Java* basiert, muss der Code nicht erst compiliert werden, sondern wird zur Laufzeit interpretiert. Dies beschleunigt die Entwicklung enorm, da der Programmcode bei laufendem GUI geändert werden kann.

Klickt der Benutzer einen Button, so wird die zugehörige Callback-Funktion aufgerufen. Der Button mit dem Tag `button_convert` besitzt beispielsweise die Funktion `button_convert_Callback(hObject, eventdata, handles)`. Die Argumente dieses Callbacks sind immer gleich und beinhalten folgendes:

- **`hObject`** - Handle zum aktuellen Object
- **`eventdata`** - Reserviert für zukünftige MATLAB Versionen
- **`handles`** - Eine Datenstruktur die alle Programmvariablen beinhaltet

Auf diese Weise verfügt jede Callback-Funktion über alle Programmdaten und kann auch auf diese schreiben. Der handle `hObject` kann dafür benuttz werden, um Werte der Bedienelemente auszulesen. Zum Beispiel die Position eines Schiebereglers, nachdem dieser mit der Maus bewegt wurde. Wichtig ist jedoch nach jeder Veränderung der Variable `handles` die Struktur mit folgendem Befehl zu aktualisieren: `guidata(hObject, handles)`. Um die große Anzahl an Variablen übersichtlich zu gruppieren, wurden weitere Datenstrukturen angelegt, welche im folgenden genauer beschrieben werden.

2.2 Datenstrukturen

Um im GUI bequem mit den Daten arbeiten zu können und Überblick zu behalten, wird das Objekt `userData` der Struktur `handles` verwendet. In dieser Variable werden die Daten sinnvoll zu Substrukturen gruppiert, welche ebenfalls Hardware-Komponenten nachempfunden sind. Es werden Einstellungen (zB Signaltyp, Frequenz, Signallänge,...) sowie Signale gespeichert. Befüllt werden diese Strukturen während der Laufzeit mit den vom Benutzer eingestellten Werten. Die Verwendung dieser Strukturen ist auch notwendig um Daten zwischen ausgelagerten Callback-Funktionen hin und her zu verschieben. Ansonsten würden die Variablen nur im Hauptprogramm und deren privaten Funktionen sichtbar sein.

[1] Der *Tag* eines Objekts ist der Name mit dem es eindeutig angesprochen werden kann.

userData.a_sig
Informationen über das Analogsignal:
Signaltyp, Frequenz, Amplitude,..

userData.d_sig
Informationen über das Digitalsignal:
Signaltyp, Frequenz, Amplitude,..

userData.adc
Einstellungen des ADC:
Auflösung, Abtastrate

userData.player
Player Daten:
Signal, Abtastfrequenz, Auflösung,..

userData.config
Globale Einstellungen:
Schriftgröße, Pfade, Audiogeräte

Abbildung 2.3: Datenstrukturen

2.2.1 Analogsignal `a_sig`

`fs`	Abtastfrequenz in Hz
`Ts`	Abtastperiode in Sekunden
`type`	Signaltyp (Sinus, Rechteck, Sägezahn,..)
`f1`	Signalfrequenz (bzw. Startfrequenz für Sweeps)
`f2`	Stopfrequenz (nur für Sweeps)
`T`	Periodendauer des Signals
`A_dB`	Signalamplitude in dB
`A`	Signalamplitude linear
`dur`	Signaldauer in Sekunden
`data`	Datenvektor
`dither`	Dithersignal
`t`	Zeitvektor
`Ts_hq`	Abtastperiode des über abgetasteten Signals
`data_hq`	Datenvektor des über abgetasteten Signals
`t_hq`	Zeitvektor des über abgetasteten Signals

Dies ist die wichtigste und größte Struktur in der Software und hält sämtliche Informationen über das Testsignal. Es kann wahlweise ein deterministisches (Sinus, Sägezahn, Rechteck..), ein zufälliges (Rauschen) oder ein Musiksignal in Form einer Wave-Datei sein. Der Signaltyp ist in der Variable `type` abgelegt. In Tabelle 2.1 sind alle Werte mit den zugehörigen Signalformen aufgelistet, die `type` annehmen kann.

a_sig.type	**Signalform**
0	Audiodatei
1	Sinus
2	Sägezahn
3	Rechteck
4	Rauschen
5	Sweep exponentiell
6	Sweep linear
7	Impulse

Tabelle 2.1: Zustände von `a_sig.type`

Die Abtastrate wird ebenfalls mit dem analogen Signal gespeichert, auch wenn das zunächst nicht intuitiv erscheint. Die Signalgenerator-Komponente erzeugt das Testsignal bereits mit der eingestellten Abtastrate. Die Werte sind mit 64 Bit (*double precision*) sehr fein aufgelöst und werden als wertkontinuierlich angesehen. Wählt der Benutzer eine Musikdatei als Testsignal, wird die Abtastrate aus der Datei übernommen. Die Frequenzen `f1` und `f2` (in Hz) beschreiben das Signal, wobei erstere die Grundfrequenz ist und `f2` nur für Sinus-Sweeps benötigt wird. Die Signalamplitude wird ebenfalls abgespeichert.

Um die Anzeige sehr hochfrequenter Signale zu verbessern, wird außer dem Analogsignal mit dem Abtastintervall `Ts` ein weiteres Signal mit `Ts_hq = Ts/20` generiert. Es wird allerdings ausschließlich zur Ausgabe im Zeitbereich verwendet um lange Rechenzeiten zu vermeiden.

2.2.2 Digitalsignal `d_sig`

`fs`	Abtastfrequenz in Hz
`data`	Datenvektor
`noise`	Quantisierungsrauschen
`R`	Dezimierungs-Faktor des Sample Rate Converters
`L`	Interpolations-Faktor des Sample Rate Converters

Die Struktur `d_sig` wird von der ADC-Komponente erzeugt und beinhaltet Zeitvektor, Daten sowie den Quantisierungsfehler `noise`, welcher die Differenz zum analogen Signal (`a_sig.data - d_sig.data`) beschreibt. Die Abtastrate kann vom Analogsignal abweichen, sollte der Benutzer die *Sample Rate Converter* (SRC) Komponente aktivieren. Dafür steht ein Eingabefeld für die gewünschte Ziel-Abtastfrequenz zur Verfügung.

2.2.3 Analog/Digital Wandler `adc`

`fs`	Abtastfrequenz in Hz
`res`	Auflösung in Bit
`q`	Quantisierungsstufe
`rounding`	Rundungsmethode (floor, ceil, rounding)

Hier sind die Parameter Abtastrate und Auflösung der ADCs gespeichert. Bei der Digitalwandlung wird das Analogsignal `a_sig.data`, welches bereits die eingestellte Abtastrate aufweist, Sample für Sample quantisiert. Sollte der Benutzer den SRC aktiviert haben, so wird zuvor die Abtastrate umgerechnet und erst danach quantisiert.

2.2.4 Audio Player `player`

`inID`	ID des Audio Eingangs
`outID`	ID des Audio Ausgangs
`inName`	Name des Audio Eingangs
`outName`	Name des Audio Ausgangs
`data`	Signalvektor
`t`	Zeitvektor
`fs`	Abtastfrequenz in Hz
`res`	Auflösung in Bit
`x_min`	Markierter Bereich in der Übersicht
`x_max`	Markierter Bereich in der Übersicht

In der `player`-Struktur stehen sämtliche Informationen, die zum Abspielen der Testsignale nötig sind. Zum einen beinhaltet die Variable Name und ID des Audiogerätes, zum anderen die Audiodaten mit zugehörigem Zeitvektor. Der Benutzer kann entweder das Analogsignal, das Digitalsignal oder das Quantisierungsrauschen (Differenz der beiden) abspielen. Um das sehr leise Rauschen gut hörbar zu machen, besteht die Möglichkeit, alle drei Signale vor dem Abspielen zu normalisieren (auf $0\,\mathrm{dB_{FS}}$ zu verstärken). Beim Start der Software sucht das Programm nach installierten Audiogeräten und fragt den Benutzer welches verwendet werden soll. Existiert nur ein Gerät, so wird dieses ohne Benutzerinteraktion ausgewählt.

2.2.5 Konfiguration `config`

FONTSIZE_WIN	Schriftgröße für Windows Rechner
FONTSIZE_MAC	Schriftgröße für Mac Rechner
MAX_HARMONICS	Anzahl der Obertöne (Rechteck und Sägezahn)
AUDIO_FILE_PATH	Dateipfad zu Beispiel Audiodateien
LAST_AUDIO_INPUT	Audio Eingang der letzten Sitzung
LAST_AUDIO_OUTPUT	Audio Ausgang der letzten Sitzung
SW_VERSION	Software Version
HQ_OVERSAMPLING	Oversampling Faktor für das Analogsignal
LAST_OS	Zuletzt verwendetes Betriebssystem
IMG_PATH	Zuletzt verwendeter Dateipfad für Diagramme

In dieser Struktur werden sämtliche Programmeinstellungen gespeichert. Die Einstellungen liegen in der `config.m` Datei, die durch den Benutzer verändert werden kann. Die Datei wird nur beim Starten gelesen, Änderungen erfordern also einen Neustart des Programms (MATLAB muss nicht geschlossen werden). Die Audiogeräte der letzten Sitzung werden bei jedem Start auf Verfügbarkeit geprüft. Sind sie verfügbar, so werden sie automatisch ausgewählt. Wenn nicht, wird der Benutzer gefragt, welches Gerät verwendet werden soll. Nach der Auswahl wird der Name des neuen Geräts zurück in die `config.m` Datei geschrieben.

2.3 Initialisierung

Die Software wird über ein kurzes Skript `start.m` aufgerufen. Hier werden zuerst alle Ordner mit Helferfunktionen zum MATLAB Pfad hinzugefügt und anschließend sämtliche Schriftgrößen angepasst. Die Darstellung von Schriften in einem GUI variiert sehr stark zwischen Windows und Macintosh Geräten. Die Funktion `changeGuiFontSize`, genauer beschrieben in Abschnitt 2.12, setzt alle Schriftgrößen korrekt. Um die Startzeit des Programms zu minimieren, werden die Schriftgrößen nur dann angepasst, wenn sich das Betriebssystem seit der letzten Sitzung geändert hat. Das zuletzt verwendete Betriebssystem ist im `config.m` in der Variable `LAST_OS` abgelegt.

Code 2.1: Startroutine.

```
1  addpath(genpath('functions')); % add folders with functions
2  run('config');    % load configuration file
3
4  % identify operating system
5  if ismac
6      OS = 'mac';
7  else
8      OS = 'win';
9  end
10
11 if (strcmp(LAST_OS,'win') && ~ismac) || (strcmp(LAST_OS,'mac') && ismac)
12 % OS is the same as last time, don't change GUI fontsizes
13
14 elseif (strcmp(LAST_OS,'win') && ismac) || (strcmp(LAST_OS,'mac') && ~ismac)
```

```
15 % OS has changed since the last session
16
17     % Rezise all fonts for mac/windows
18     changeGuiFontSize('functions/GUI.fig',str2double(FONTSIZE_WIN),'default',
           str2double(FONTSIZE_MAC),'default');
19 end
20
21 updateConfigFile('config.m','LAST_OS', OS);
```

Nach Aufruf der Funktion `GUI` werden zunächst alle Textfelder mit sinnvollen Initialwerten beschrieben. Obwohl das Programm bei jeder Benutzerinteraktion sämtliche Werte von Textfeldern, Checkboxes und Slider ausließt, ist es von Vorteil, wenn diese zu Beginn des Codes gesetzt werden. Dadurch sind schnelle Änderungen auch ohne dem MATLAB-Tool `guide` möglich. Um dem Benutzer schnelle Ergebnisse zu liefern, werden die Einstellungen auf repräsentative Werte gesetzt. Damit alle Datenstrukturen gesetzt und die Diagramme nicht leer sind, wird hier ebenfalls die Callback-Funktion des *Convert* Buttons ausgeführt. Unten ist eine exemplarische Initialisierung gezeigt. Der `set`-Befehl erwartet den Objekt-Handle, den zu ändernden Parameter des Blocks, sowie den neuen Wert. Hier wird die Startfrequenz des Signalgenerators auf 1003 Hz gesetzt. Es ist zu beachten, dass die Textfelder keine Zahlen sondern Strings beinhalten. Beim Auslesen müssen die Zeichenketten zuerst in Zahlen umgewandelt werden.

```
set(handles.edit_f1,'String','1003');
```

2.4 Signalgenerator

Der Signalgenerator kann diverse deterministische Signale (z.B. Sinus, Sägezahn, Rechteck, Sweep, etc.), als auch zufällige Signale wie Rauschen erzeugen. Einstellbar sind hierbei Amplitude, Frequenz und Länge des Signals. Zurück geliefert wird ein Vektor im Format `double` (64 Bit floating point), der im weiteren Programmverlauf als quasi-analoges (wert-kontinuierliches) Eingangssignal gedeutet wird. Die Zeitachse ist bereits an dieser Stelle auf die globale Abtastperiode T_s skaliert. Der Benutzer kann wahlweise die Anzahl der zu generierenden Perioden, oder die Signallänge in Sekunden eingeben. Die Eingabe von wenigen Perioden macht die Anzeige im Zeitbereich wesentlich übersichtlicher und auch die Zoomfunktionen sind einfacher zu bedienen. Die Frequenzdarstellung wird allerdings bei längeren Signalen besser aufgelöst. Außerdem wird die Eingabe der Länge in Sekunden für Sweeps benötigt.

Die Funktion `genSignal` erwartet die oben genannten Parameter sowie den Signaltyp (Tab. 2.1). Auf diese Weise kann die Funktion einfach um neue Signalformen erweitert werden. Für die Signalformen *Sinus* und *Rauschen* wurden die MATLAB Funktionen `sin` und `rand` verwendet. Auf die Generierung von Rechteck, Sägezahn und Sweep soll nun genauer eingegangen werden.

2.4.1 Generierung eines Rechtecksignals

Ein Rechtecksignal kann auf zwei Arten generiert werden. Die einfachere ist, einen Vektor mit $n \cdot A$ (A sei die Amplitude), gefolgt von $n \cdot (-A)$ zu füllen und diesen beliebig oft zu wiederholen (z.B. mit dem Befehl `repmat`). Da es sich hier um ein zeitdiskretes Signal handelt, kann die Periodendauer nur ganzzahlige Vielfache der Abtastperiode T_s sein. Dies ist eine Einschränkung, da so nicht jede beliebige Frequenz erzeugt werden kann. So ist beispielsweise bei einer Abtastfrequenz von $f_s = 44.1\,\text{kHz}$ nur ein minimaler Frequenzsprung von $8.73\,\text{Hz}$ möglich.

Die genauere, aber auch sehr viel rechenaufwändigere Methode ist, das Rechteck über eine Fourierreihe zu synthetisieren. Die theoretisch unendliche Reihe kann wegen der geforderten Bandbegrenzung (Shannon'shes Abtasttheorem) bei $f_s/2$ abgebrochen werden.

$$x(t) = \frac{4A}{\pi} \sum_{n=1}^{\infty} \frac{\sin((2n-1)\omega t)}{(2n-1)} \tag{2.1}$$

Für ein Rechtecksignal werden also nur alle ungeradzahligen Oberwellen benötigt. Die Summe wird entweder bei der halben Abtastfrequenz oder bei einer maximalen Anzahl von `MAX_HARMONICS` (einstellbar über die `config.m` Datei, Standardwert is 60) Oberwellen abgebrochen. Auf diese Weise kann jede beliebige Frequenz generiert werden. Der Programmcode dafür ist im Codeblock 2.2 skizziert.

Code 2.2: Generierung eines Rechtecksignals.

```
1  t = (0:Ts:dur_sig-Ts);
2
3  numHarm = round(min(fs/2/f_sig,MAX_HARMONICS));   % number of harmonics
4
5  SIN = zeros(numHarm,length(t));          % array of harmonics
6
7  for i = 1:2:numHarm
8      coeff = 1/i;                         % harmonic weight
9      SIN(i,:) = coeff * sin(2*pi*i*f_sig*t);
10 end
11
12 a_sig = 4*A_sig/pi * sum(SIN);
```

Es wird eine Matrix mit `numHarm` (Anzahl der Oberwellen) Zeilen und `length(t)` (Anzahl der Elemente im Zeitvektor) Spalten generiert, welche in einer Schleife mit den Oberwellen gefüllt wird (Zeile 9). In der ersten Zeile steht die Grundschwingung, in der zweiten die erste Harmonische usw. In Zeile 12 werden diese Signale spaltenweise aufsummiert und ergeben so das finale Rechtecksignal. Um den ADC nicht zu übersteuern, wird das Maximum des Signals auf die eingestellte Amplitude gebracht (in Abb. 2.4 auf 0.5).

2.4.2 Generierung eines Sägezahnsignals

Das Sägezahnsignal wird analog zum Rechtecksignal generiert. Der einzige Unterschied liegt in der Fourierreihe. Hier werden alle ganzzahligen Oberwellen benötigt,

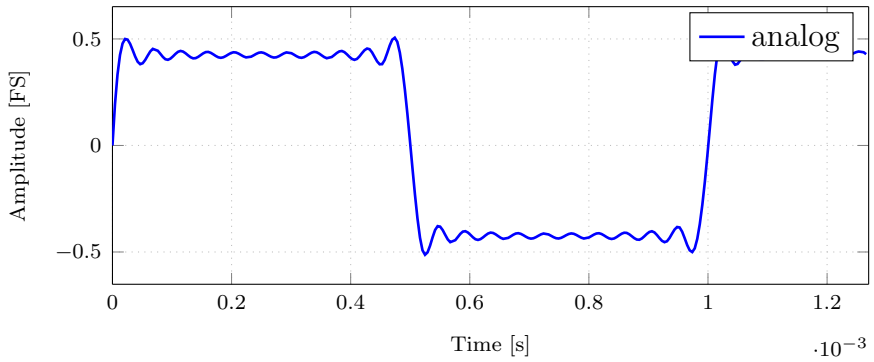


Abbildung 2.4: Synthetisiertes Rechtecksignal $f = 1000\,\text{Hz}$

die Koeffizienten sind wieder $1/n$:

$$x(t) = \sum_{n=1}^{\infty} \frac{\sin(n\omega t)}{n} \tag{2.2}$$

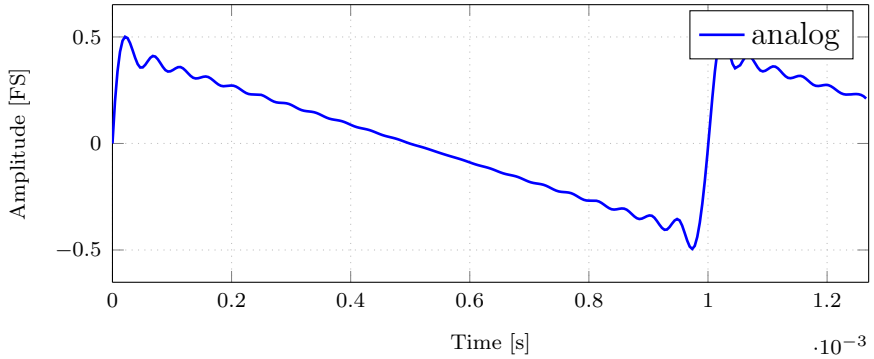


Abbildung 2.5: Synthetisiertes Sägezahnsignal $f = 1000\,\text{Hz}$

2.4.3 Generierung eines Sinus-Sweeps

Der Zeitvektor `t` gibt die Abtastfrequenz und die Länge des Signals vor. Es muss nun für jeden Abtastwert eine Momentanfrequenz berechnet werden. Dazu wird in einer Hilfsvariable `k` die Steigung der Frequenz zwischen zwei Abtastwerten berechnet. Diese multipliziert mit einem Zählvektor (Ganzzahlige Werte von 0 bis zur Anzahl der Werte in `t`) und addiert zur Startfrequenz, ergibt den Frequenzvektor. Als Argument der Sinus-Funktion steht dann `(2*pi*f.*t)`, Zeit und Frequenzvektor werden Elementweise multipliziert (`.*`).

Code 2.3: Generierung eines linearen Sinus-Sweeps.

```
1 t = (0:Ts:dur_sig-Ts);                    % time vector
2
3 k = (f_stop-f_sig)/(length(t)-1)/2;       % frequency slope
4 f = 0:length(t)-1;
5
6 f = f_sig + f*k;                          % frequency vector
7 a_sig = A_sig*sin(2*pi*f.*t);
```

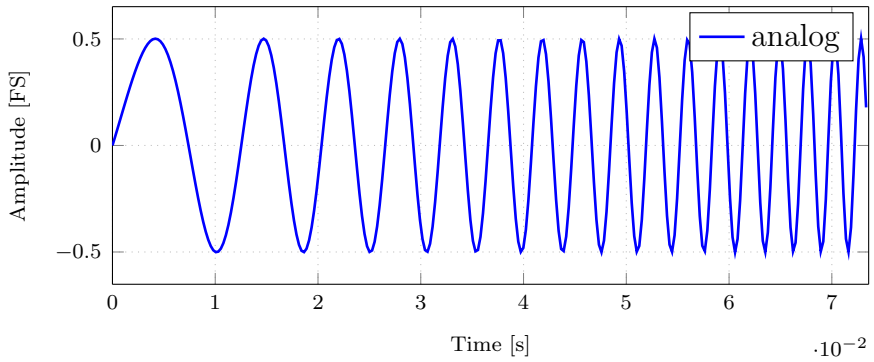


Abbildung 2.6: Linearer Sinus-Sweep.

Zur Generierung eines exponentiellen Sweeps wurde eine Funktion von Piotr Majdak ([2]*Acoustics Research Institute - Austrian Academy of Sciences*), vorgestellt in der Lehrveranstaltung *Algorithmen in Akustik und Computermusik UE* verwendet. Diese Funktion generiert den Sweep wie oben gezeigt, allerdings mit einem exponentiell ansteigenden Frequenzvektor.

2.5 Dither Generator

Um die Korrelation des Quantisierungsrauschens zu reduzieren, besteht die Möglichkeit des Ditherings. Hierbei wird vor der Quantisierung ein sehr leises Rauschen zum Analogsignal addiert. Die Funktion `genDither` erzeugt ein Rauschsignal gleicher Länge wie das analoge Signal. Es kann zwischen verschiedenen Verteilungsfunktionen des Rauschens gewählt werden (Gleichverteilt und Hochpass gefiltert). In Abb. 2.7 sind die Spektren zweier Dithersignale mit einer gesamten Rauschleistung von je $-48\,\mathrm{dB}$ abgebildet. Zur Filterung wurde ein IIR Filter 2. Ordnung mit einer Grenzfrequenz von $f_s/2$ verwendet. Die verlorene Energie wird durch einen Verstärkungsfaktor wieder kompensiert, sodass beide Signale die selbe Leistung aufweisen.

2.6 Analog/Digital Wandler

Das Eingangssignal (wahlweise mit oder ohne Dither) wird anschließend dem ADC zugeführt. An dieser Stelle kann sowohl die Auflösung als auch die Abtastrate eingestellt werden. Um den Signalfluss logisch zu halten, wird die Abtastrate am ADC und nicht am Signalgenerator eingestellt. Tatsächlich liefert der Signalgenerator bereits ein zeitdiskretes, quasi wert-kontinuierliches Signal vom Datentyp `double` (64 Bit floating point).

Zunächst wird eine Quantisierungsstufe (`q`) berechnet indem das Interval -1 bis 1 durch die Anzahl der Stufen (2^k) dividiert wird. Dies ist der Wertebereich den ein voll ausgesteuertes Signal annehmen kann. Ein Hilfsvektor `steps` wird angelegt, der die Quantisierungskennlinie beinhaltet. Am Beispiel eines 2 Bit ADC ist `q` =

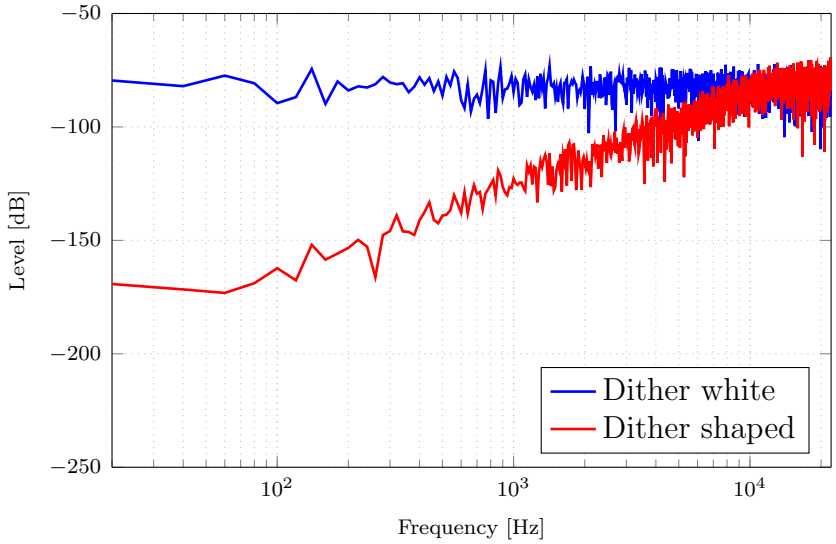


Abbildung 2.7: Dithersignale mit einer Rauschleistung von je −48 dB.

`0.6667` und `steps = [0; 0.6667; 1.3333; 2.0000]`. Der Algorithmus in der Schleife berechnet nun den Index der nächsten Quantisierungsstufe in `steps` für jeden Abtastwert. Nachdem keine negativen Indizes verwendet werden dürfen, wird das Signal vorher um +1 in den positiven Wertebereich verschoben.

Analoge Signale, die das Intervall ±1 verlassen, werden durch die beiden `if`-Bedingungen gesättigt. Durch die Rundungsoperation wird kaufmännisch gerundet und es entsteht ein gleichanteilfreies, quantisiertes Signal. Hier könnten auch andere Rundungsmethoden verwendet werden. In Abbildung 2.8 ist ein Sinus vor und nach der Quantisierung mit einem 2 Bit ADC dargestellt.

Code 2.4: Analog/Digitalwandlung.

```
1  q = 2/(2^res);        % LSB
2  steps = (0:q:2-q)';   % conversion curve
3
4  len = length(a_sig);
5  d_sig = zeros(len,1);    % digital signal
6
7  for i=1:len
8
9      ind = round((a_sig(i)+1)/q)+1;     % determine quantization step index
10
11     if (ind < 1), ind = 1; end                          % overflow
12     if(ind > numel(steps)), ind = numel(steps); end     % underflow
13
14     d_sig(i) = steps(ind)-1;           % remove offset and assign value
15
16 end
```

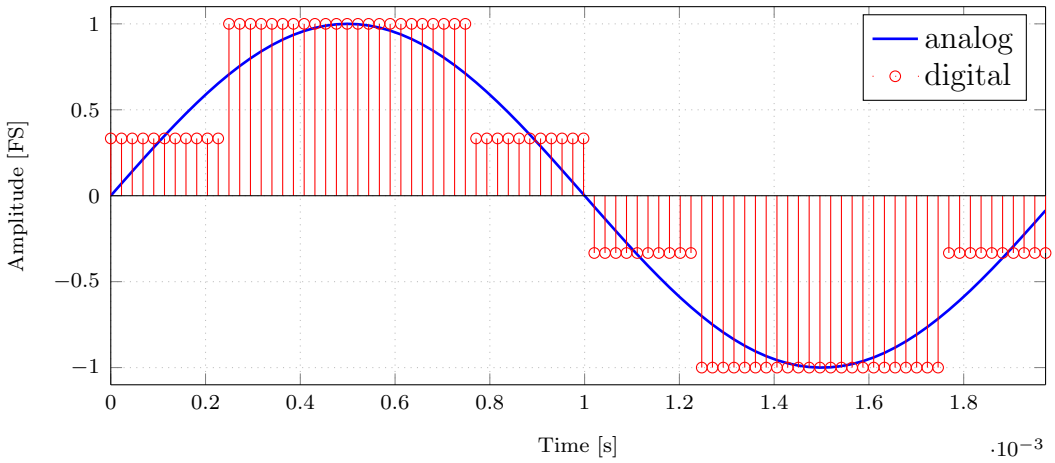


Abbildung 2.8: Sinus Signal (500 Hz) vor und nach ADC ($f_s = 44.1$ kHz, 2 Bit)

2.7 Sample Rate Converter

Um die Effekte von Unter- bzw. Überabtastung genauer untersuchen zu können, wurde ein *Sample Rate Converter* (SRC) implementiert. Dieser erlaubt dem Benutzer die Abtastrate des digitalen Signals beliebig zu verändern. Dabei gibt es die Option die Bandbegrenzung (anti-aliasing Filter) zu deaktivieren um Aliasing-Effekte zu simulieren.

Die Änderung der Abtastrate wird durch Interpolation (um den Faktor L) gefolgt von Dezimation (um den Faktor R) erzielt. So kann jede rationale Abtastratenänderung erreicht werden. Die Berechnung der Faktoren L und R erfolgt mit dem Befehl `[L,R] = rat(fs_src/fs)`. Die neue Abtastfrequenz berechnet sich dann $f_{s,neu} = f_s \cdot L/R$.

SRC ohne Bandbegrenzung

Wird der anti-aliasing Filter deaktiviert, so verwendet der Algorithmus die beiden Funktionen `interp` und `downsample`. Interpolation bedeutet in diesem Fall nur Einfügen von L Nullen nach jedem Sample und Dezimation verwirft jedes R-te Sample. Es wird also nicht sicher gestellt, dass keine Signalanteile $> f_{s,neu}/2$ enthalten sind und es kann zu Aliasing-Effekten kommen.

Code 2.5: SRC ohne Bandbegrenzung.

```
1 d_re = interp(data_d,L);
2 d_re = downsample(d_re,R);
3 fs_re = fs*L/R;
```

SRC mit Bandbegrenzung

Um eine saubere Abtastratenkonversion durchzuführen muss die Bandbreite des Signals zunächst auf $f_{s,neu}/2$ begrenzt werden. Die MATLAB Funktion `resample` wendet einen Poly-Phasen Filter zur Änderung der Abtastrate an, welcher in einem Schritt Tiefpass filtert und die Abtastrate reduziert. Durch den Poly-Phasen Algorithmus wird der Rechenaufwand stark vermindert, weil Multiplikationen mit

Nullen (eingefügt durch die Interpolation) eingespart werden.

Code 2.6: SRC mit Bandbegrenzung.

```
1 d_re = resample(data_d,L,R);
2 fs_re = fs*L/R;
```

2.8 Jitter Generator

Idealer Weise liefert ein ADC genau zu allen Vielfachen der Abtastperiode T_s einen Abtastwert. Praktisch kann es aber zu leichten Abweichungen kommen, die den Abtast-Takt leicht zittern lassen. Diese, meist harmonische Ungenauigkeit wird als *Jitter* bezeichnet. Der Effekt auf das Audiosignal ist eine Frequenzmodulation mit der Jitter-Frequenz. Die Software bietet die Funktion Jitter zu simulieren und diesen auf das Testsignal anzuwenden. Auch eigene Audiodateien können damit manipuliert werden.

Die Funktion `genJitter` erwartet das Signal, Jitter-Amplidude und Frequenz, sowie die Signalform des Jitters (Sinus, Sägezahn,..) und liefert die manipulierten Daten zurück. Es wird ein neuer Zeitvektor berechnet, der mit der gegebenen Form von einer Geraden abweicht. Zur Berechnung der neuen Abtastwerte, wird eine kubische Interpolation (es wird ein Polynom 3. Ordnung durch 4 benachbarte Samples gelegt) verwendet. Diese lässt sich auch bei längeren Sequenzen sehr schnell berechnen und liefert gute Ergebnisse. Die Jitter Amplitude wird in *Unit Intervals* (UI) angegeben. Diese Zeiteinheit wird vom AES-EBU Format abgeleitet und entspricht der Periode eines übertragenen Halb-Bits. Ein Frame hat eine Länge von 64 Bit, also 128 Halbbit. Ein UI ist demnach $T_s/128$.

Code 2.7: Jitter Berechnung.

```
1 t = 0:Ts:len*Ts-Ts;                          % original time vector
2 A = A_UI * Ts/128;
3 offset = A * sin(2*pi*f*t);
4 t_jit = t + offset;                          % jittery time vector
5 signal_jit = spline(t,signal(:,1),t_jit);    % new, jittery signal
```

In Abb. 2.9 ist ein sehr stark übertriebener Jitter von 64 UI eingestellt, um den Effekt auch im Zeitbereich erkennen zu können. Es ist deutlich zu sehen, dass bei 0.5 ms das rote Signal voreilt und bei 1.5 ms nachhinkt. In der Praxis sind die Amplituden sehr viel kleiner, allerdings werden die Seitenbänder der Frequenzmodulation bereits ab etwa 2 UI hörbar. In Abb. 2.10 sind diese im Quantisierungsrauschen deutlich zu erkennen.

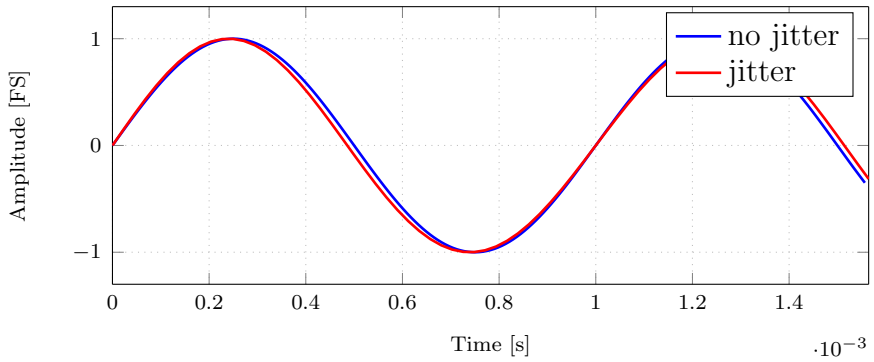


Abbildung 2.9: Sinus Signal (0 dB @ 1000 Hz) mit Jitter (64 UI @ 500 Hz).

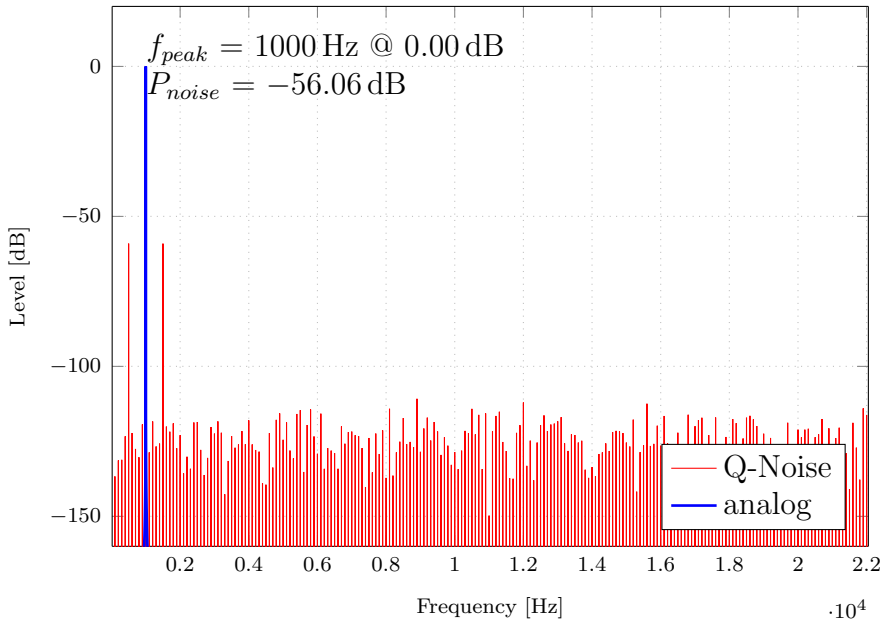


Abbildung 2.10: Sinus Signal (0 dB @ 1000 Hz) mit Jitter (2 UI @ 500 Hz).

2.9 Anzeigeoptionen

Um die Effekte der Quantisierung mit verschiedenen, simulierten Zusätzen (Dither, Jitter, SRC, ...) genauer studieren zu können, steht eine Vielzahl an Analyse-Funktionen zur Verfügung. Nach drücken des *Convert* Buttons, erscheint das neue Signal automatisch im Analysebereich (Abb. 2.11). Auf der rechten Seite werden die verschiedenen Anzeigeoptionen aufgelistet. Der Plot wird bei jedem Klick automatisch aktualisiert. Die beiden Scrollbalken über und rechts neben dem Plot dienen zur Vergrößerung und Navigation durch das Testsignal. Der Zoom-Slider vergrößert den Ausschnitt nicht linear, sondern über eine Exponentialfunktion. Das bedeutet, je kleiner der Ausschnitt, desto feiner ist die Auflösung des Balkens. Das erleichtert die Navigation durch lange Signale erheblich.

In den folgenden Abschnitten werden nun die wichtigsten Optionen und deren Implementierung beschrieben.

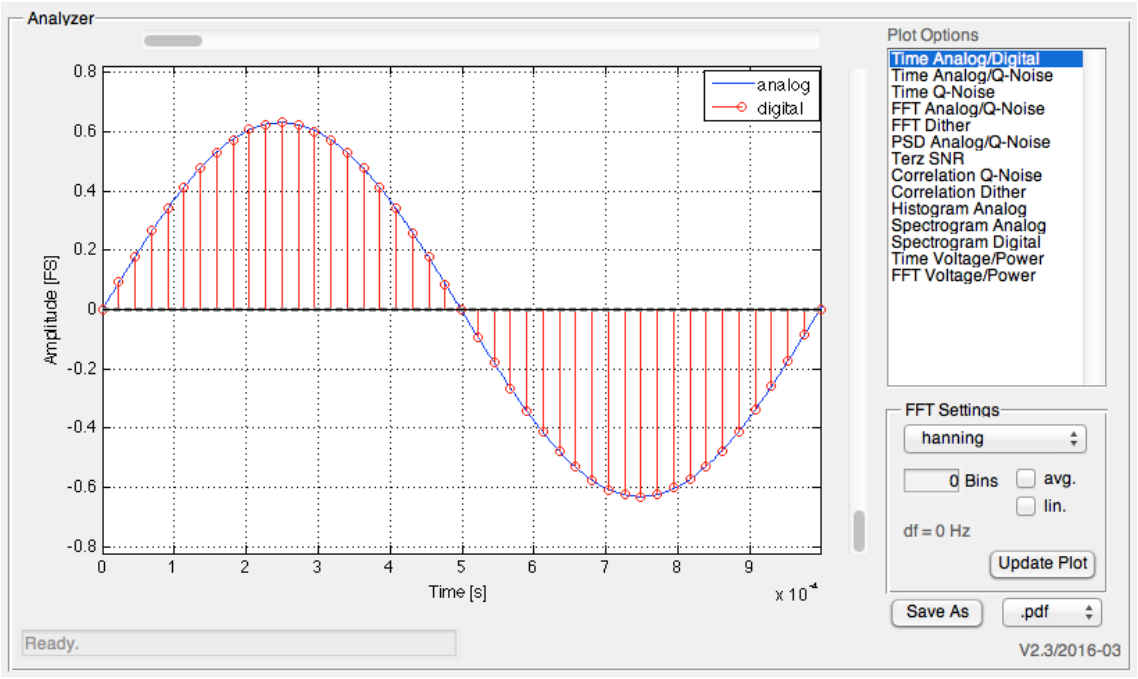


Abbildung 2.11: Analysebereich mit Plot-Optionen.

2.9.1 Analog/Digital im Zeitbereich

Die erste Option zeigt das Testsignal vor und nach der Digitalwandlung. Für das Digitalsignal wurde die *Lollipop*-Darstellung gewählt, da diese der mathematischen Beschreibung eines abgetasteten Signals besser entspricht als eine Stufendarstellung. Das Analogsignal wird als durchgezogene, blaue Linie dargestellt.

Hier kommt das überabgetastete Analogsignal zum Einsatz (`a_sig.data_hq`). Nachdem es 20 Mal so viele Samples aufweist, wird sogar bei sehr hohen Frequenzen (auch über $f_s/2$) eine glatte Linie angezeigt. Der Slider über dem Plot navigiert durch das Signal (der angezeigte Bereich wird im Player Plot rot markiert) und der vertikale Slider vergrößert, bzw. verkleinert den angezeigten Zeitbereich.

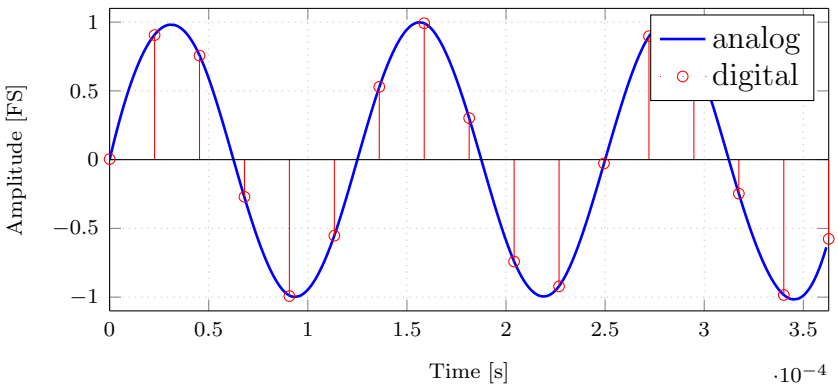


Abbildung 2.12: Sinus Signal (0 dB @ 8 kHz) vor und nach ADC.

2.9.2 Quantisierungsrauschen im Zeitbereich

Das Quantisierungsrauschen wird als Differenz von Analog- und Digitalsignal berechnet. Wieder bewirken die Scrollbalken eine Vergrößerung bzw. ein Verschieben der Zeitachse. Die Y-Achse wird automatisch so skaliert, dass eine übersichtliche Anzeige gewährleistet ist. Die strichlierten Linien zeigen $\pm Q/2$ der aktuellen Auflösung an.

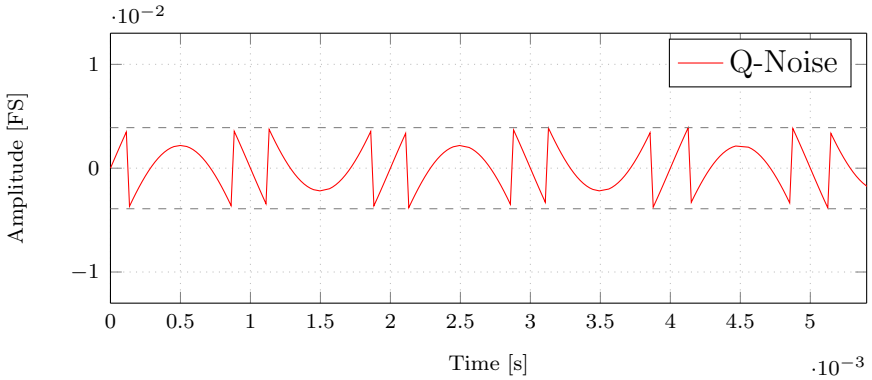


Abbildung 2.13: Quantisierungsrauschen eines Sinus-Signals (500 Hz @ −40 dB, 8 Bit).

2.9.3 Analog/Digital Frequenzbereich

Das Spektrum des Analogsignals und das Quantisierungsrauschens wird hier in einem Plot dargestellt. Für die Berechnung wird Code 2.8 verwendet. Es besteht die Möglichkeit zwischen verschiedenen Fensterfunktionen zu wählen (z.B. Rechteck, Hanning). Um korrekte Pegel zu erhalten, muss die verlorene Energie berücksichtigt werden. Die Hanning Funktion entfernt die Hälfte der Signalenergie, deshalb wird das Fenster mit 2 multipliziert (Zeile 1). In den Zeilen 9 und 10 wird die Signalenergie auf die Länge normiert und die negativen Frequenzen des hermitisch symmetrischen Spektrums berücksichtigt. Die FFT Ordnung ist standardmäßig gleich der Länge des Testsignals, kann aber durch den Anwender reduziert werden. Es besteht weiters die Möglichkeit einer Mittelwertbildung. Ist diese ausgewählt, so wird das Testsignal in gleiche Teile zerlegt und die Spektren werden gemittelt. Die Anzeige kann mit linearer oder logarithmischer Frequenzachse erfolgen. All diese Einstellungen befinden sich rechts unten im Analyse Fenster (Abb. 2.11).

Code 2.8: FFT Berechnung.

```
1  w = 2 * hann(order);          % Window function with lost energy
2  sig = sig(:);                 % Make column vector
3  sig = sig(1:order) .* w;
4
5  df = fs/order;                % Frequency resolution
6  f = 0:df:df*order-df;         % Frequency vector for plots
7
8  dft = fft(sig,order);
9  dft = 1/order.*dft;           % Scale by order
10 dft(2:end-1) = 2*dft(2:end-1); % Add imaginary part
11 fft_log = 20*log10(abs(dft));
```

Der höchste Punkt im Spektrum wird gesucht und dessen Werte (Pegel und Frequenz) werden im Diagramm ausgegeben. Außerdem wird die gesamte Rauschleistung sowie die Frequenzauflösung bestimmt und ebenfalls ausgegeben. Die Berechnung der Leistung eines Signals wird an mehreren Stellen in der Software benötigt, deshalb wurde eine Helferfunktion implementiert. Der Algorithmus dafür steht im Codesegment 2.9 unten. Nachdem nur Leistungen addiert werden dürfen, wird in Zeile 10 das Amplitudenspektrum Elementweise quadriert, aufsummiert und der Zehner-Logarithmus berechnet. Das Ergebnis ist die Signalleistung in dB.

Code 2.9: Berechnung der Signalleistung.

```
1  order = length(sig);
2
3  dft = fft(sig,order);            % calculate dft
4  dft = 1/order.*dft;              % normalize by signal length
5  dft(2:end-1) = 2*dft(2:end-1);   % add negative frequency content
6  dft_abs = abs(dft);              % calculate magnitude
7
8  start = 1;
9  stop = floor(order/2);
10 power = 10*log10(sum(dft_abs(start:stop).^2));
```

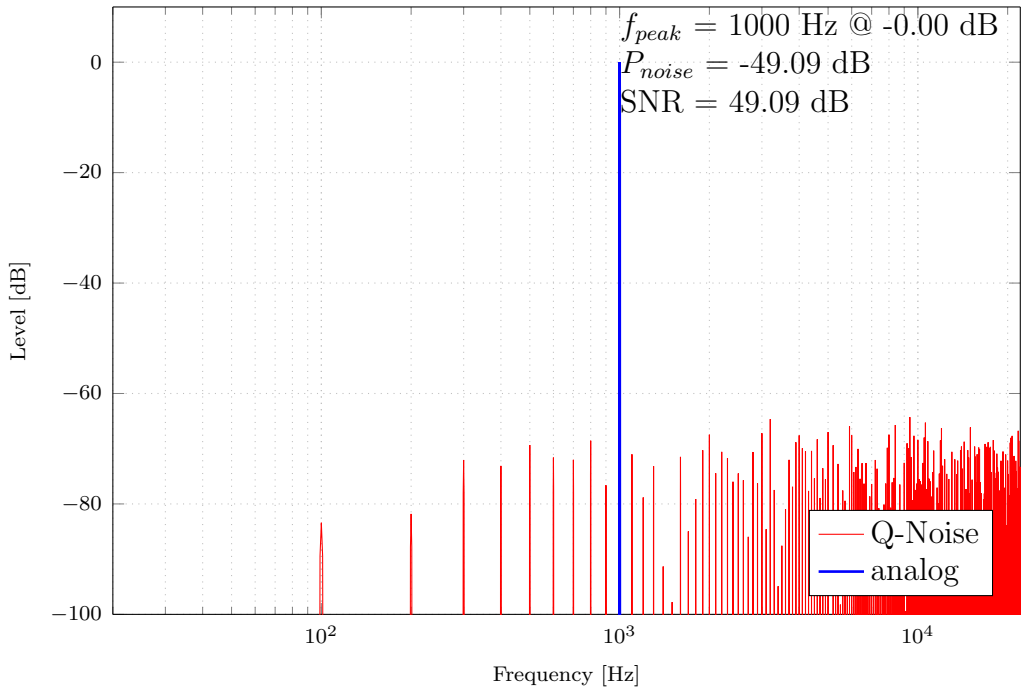


Abbildung 2.14: Ausgabe im Frequenzbereich. (1000 Hz @ 0 dB, 8 Bit).

2.9.4 Leistungsdichtespektrum

Das Leistungsdichtespektrum ist die Leistung pro Herz [3]. Die Berechnung und Ausgabe der Leistungsdichtespektren erfolgt also analog zu Abschnitt 2.9.3, mit

dem Unterschied, dass die FFT durch die Frequenzauflösung dividiert wird.

Die FFT und das Leistungsdichtespektrum sind dann gleich, wenn die Auflösung der FFT $df = 1\,\text{Hz}$ ist, also das Signal gleich viele Samples wie die Abtastfrequenz hat (Beispiel: $f_s = 48\,000\,\text{Hz}$, Signallänge ist $1\,\text{s} = 48000$ Samples. Die Auflösung der FFT ist dann genau 1 Hz). Hat das Signal nur halb so viele Samples, ergibt sich durch die Division durch die Frequenzauflösung ($df = 2\,\text{Hz}$) im Leistungsdichtesprektrum der halbe Pegel ($-3\,\text{dB}$).

2.9.5 Terzband Signal-Rauschabstand

In Abb. 2.15 ist das Spektrum eines Sinussignals sowie das Quantisierungsrauschen in Terzbänder zusammengefasst dargestellt. Wieder wird der Spitzenwert beschriftet und die gesamte Rauschleistung (P_{noise}) berechnet. Zusätzlich wird hier noch der Signal-Rauschabstand im aktuellen Terzband (SNR_{terz})angezeigt.

Diese Darstellung soll verdeutlichen, dass die Rauschleistung in einer Terz kleiner ist als die gesamte Rauschleistung und mit der Frequenz steigt. Wird von weißem Quantisierungsrauschen ausgegangen, so beinhaltet jede Terz um 1 dB mehr Rauschleistung als die vorhergehende, aufgrund der steigenden Bandbreite. Das Codesegment 2.10 zeigt den Algorithmus:

Code 2.10: Berechnung der Signalleistung.

```
1  Fc_tz = [12.5 16 20 25 31.5 40 50 63 80 100 125 160 200 250...
2           315 400 500 630 800 1000 1250 1600 2000 2500 3125 4000 5000...
3           6300 8000 10000 12500 16000 20000];
4
5  for i=1:length(Fc_tz)
6
7      fu = Fc_tz(i) / 2^(1/6);     % lower terz limit
8      fo = Fc_tz(i) * 2^(1/6);     % upper terz limit
9
10     idx_u = max(1,round(fu/df));                    % lower index
11     idx_o = min(length(fft_noise),round(fo/df));    % upper index
12
13     if (idx_o-idx_u) == 0
14         disp('Not enough fft resolution to calculate terz band noise!')
15     end
16
17     % save frequencies and power in a matrix
18     snr(i,1) = fu;
19     snr(i,2) = fo;
20     snr(i,3) = 10*log10(sum(fft_noise(idx_u:idx_o).^2));
21
22 end
```

In einem Vektor sind die Mittenfrequenzen der Terzbänder abgelegt. In einer Schleife wird zunächst die obere und untere Grenzfrequenz berechnet ($f_u = f_c/\sqrt[6]{2}$, $f_o = f_c \cdot \sqrt[6]{2}$). Diese müssen nun in einen Frequenzindex umgerechnet werden. Das geschieht in den Zeilen 10 und 11. In der Variable `df` ist die Frequenzauflösung der FFT abgelegt, die Grenzfrequenzen dividiert durch die Frequenzauflösung ergeben

den Index. Dabei ist darauf zu achten, dass dieser nicht kleiner 1 oder größer der FFT-Ordnung wird. Sollten pro Terzband weniger als ein Frequenzbin zur Verfügung stehen, so wird eine Fehlermeldung ausgegeben.

Die Leistung im Terzband wird in Zeile 19 ermittelt. Die Leistung, sowie untere und obere Grenzfrequenz werden in der Matrix `snr` abgelegt. Der Code unten zeigt wie das Ergebnis geplottet wird. Eine Schleife durchläuft die Matrix und zeichnet Linienelemente von der unteren zur oberen Grenzfrequenz.

Code 2.11: Berechnung der Signalleistung.

```
1 semilogx(f,fft_a_sig,'r')
2 hold on
3 grid on
4 for i=1:length(Fc_tz)
5    semilogx([snr(i,1) snr(i,2)],[snr(i,3) snr(i,3)])
6    hold on
7    grid on
8 end
```

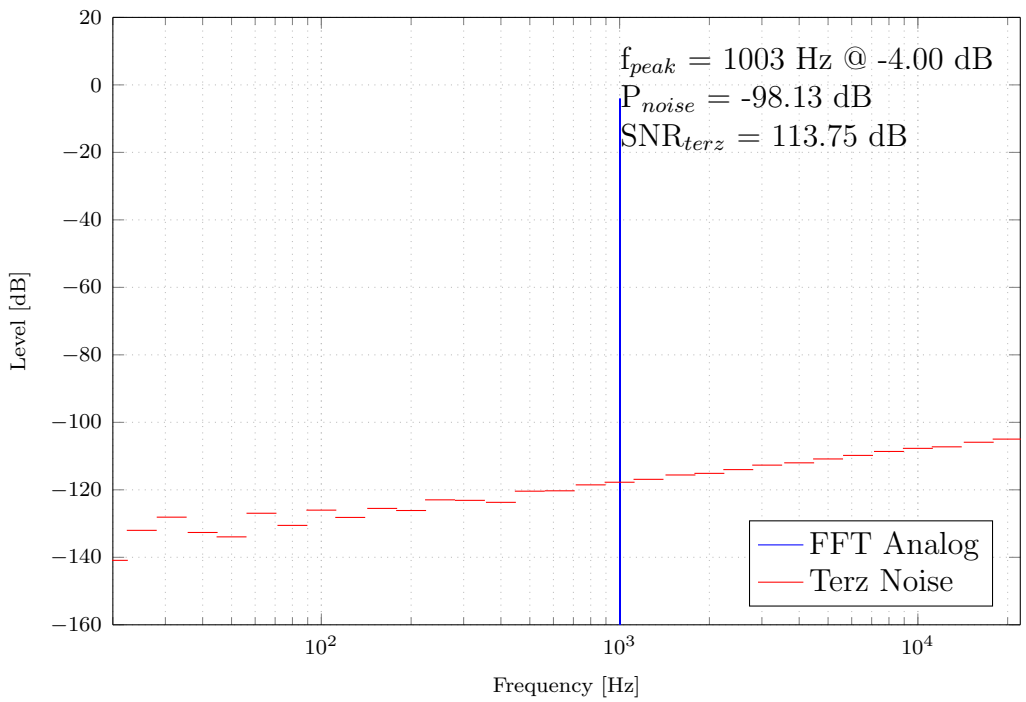


Abbildung 2.15: Sinussignal (16 Bit ADC) und Terzband-Rauschleistung.

2.9.6 Autokorrelation

Es besteht die Möglichkeit die Autokorrelation des Quantisierungsrauschens sowie des Dithersignals berechnen und anzeigen zu lassen. Hier wählt der vertikale Scrollbalken den Bereich im Signal aus, über den die Korrelation berechnet werden soll. Die Signalenergie (repräsentiert durch den Wert an der Stelle 0) wird auf 1 normiert, damit Signale besser vergleichbar sind. Zur Berechnung wurde die MATLAB

Funktion `xcorr` verwendet, `x_min` und `x_max` sind die Indizes des ausgewählten Bereichs:

Code 2.12: Berechnung der Autokorrelation.

```
1 corr_noise = xcorr(noise(x_min:x_max),noise(x_min:x_max));
2 corr_noise = corr_noise/max(corr);  % normalize to 1
```

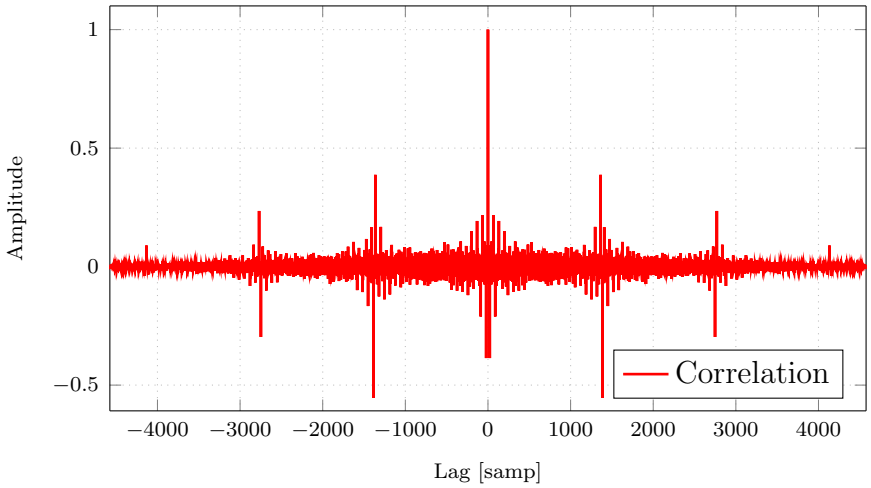


Abbildung 2.16: Autokorrelation Quantisierungsrauschen.

2.9.7 Histogramm

Eine weitere, neue Funktion ist die Ausgabe eines Histogramms (Amplitudenverteilung). Auf der X-Achse sind die Amplituden aufgetragen und auf der Y-Achse die statistische Häufigkeit. Die X-Achse verläuft immer von -1 bis +1, um die Signale besser vergleichen zu können. Das Beispiel in Abb. 2.17 zeigt die charakteristische Badewannen-Verteilung eines Sinussignals bei einer Amplitude von −4 dB.

Im Codebeispiel 2.13 ist die Berechnung der statistischer Momente Mittelwert, Varianz, Standardabweichung, Crest- und Scheitelwertfaktor skizziert. Die Funktionen `var`, `mean`, `std` und `peak2rms` sind in MATLAB enthalten.

Code 2.13: Berechnung der statistischen Momente.

```
1 a_var = var(a_data);
2 a_mean = mean(a_data);
3 a_std = std(a_data);
4 a_crest = peak2rms(a_data);
5 a_swf = 20*log10(sqrt(3)/a_crest);
```

2.9.8 Spektrogramm

Ein Spektrogramm ist eine Darstellung der FFT (Y-Achse) über die Zeit (X-Achse). Die Werte der FFT Bins werden in Farben codiert. MATLAB bietet die Funktion `spectrogram`. Die Übergabeparameter sind Fensterlänge (1024), Überlappungsfaktor (50 %), FFT Ordnung (1024) und Abtastrate.

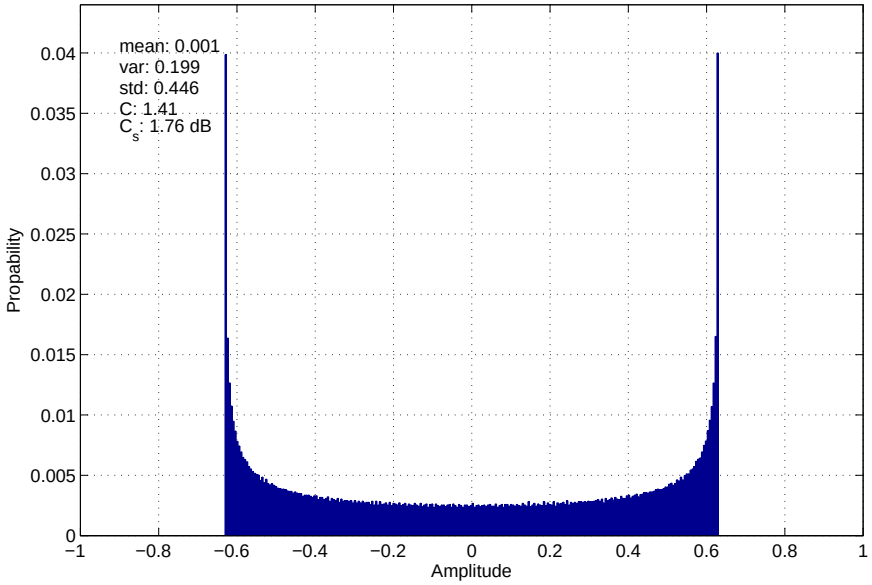


Abbildung 2.17: Histogramm eines Sinussignals (500 Hz @ −4 dB).

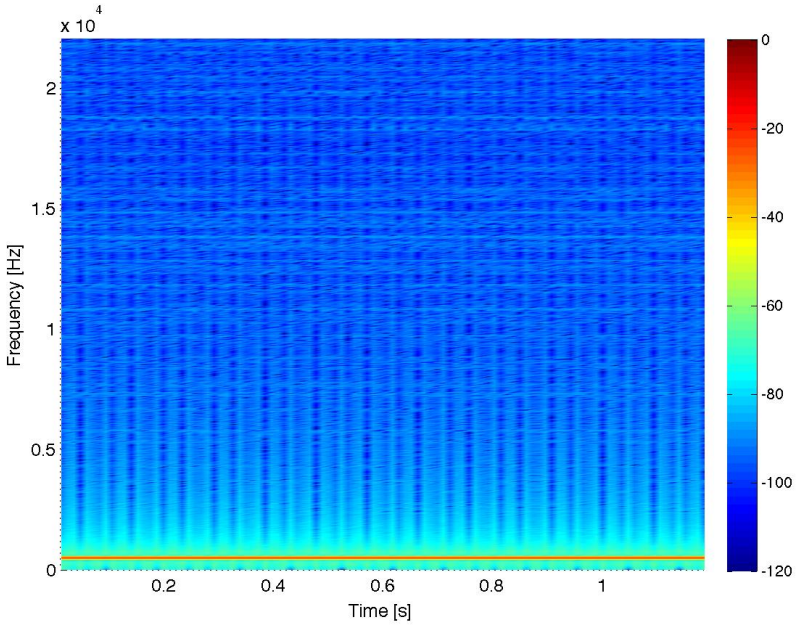


Abbildung 2.18: Spektrogramm eines digitalen Sinussignals (500 Hz @ 8 Bit ADC).

2.9.9 Spannung und Leistung

Die letzten beiden Anzeigeoptionen beschäftigen sich mit dem Zusammenhang zwischen Spannung und Leistung. Es soll verdeutlicht werden, dass die Definition der Leistung aus der Nachrichtentechnik (Gleichung 2.3) eine Mittelung über die Zeit darstellt. Hingegen bei der FFT die Leistung aus der Spannung abgeleitet wird. Es wird die gemittelte Leistung dem Frequenzbin der Spannung zugewiesen, wobei die Leistung eigentlich die doppelte Frequenz und einen Gleichanteil aufweist (wegen

u^2). Dies wird in den Zeit- und Frequenzplots von Spannung und Leistung deutlich.

$$P = \int_0^T \frac{u^2(t)}{R}\, dt \tag{2.3}$$

$$P_{log} = 10 \cdot \log\left(\frac{U^2}{R}\right) = 20 \cdot \log(U) \quad \text{mit R} = 1 \text{ Ohm} \tag{2.4}$$

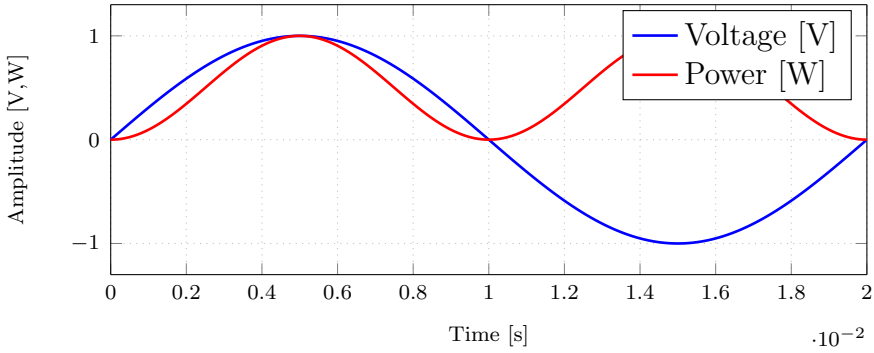


Abbildung 2.19: Spannung (50 Hz @ 1 V) und resultierende Leistung an 1 Ohm.

In Abb. 2.20 ist das Spektrum von Spannung und Leistung an einer 1 Ω Last im Frequenzbereich aufgetragen. Es ist zu erkennen, dass die tatsächlich Leistung auf zwei Frequenzbins aufgeteilt ist. Nämlich auf 100 Hz und 0 Hz jeweils mit −3 dB. Diese Leistung ist eine Größe, die in der Praxis hauptsächlich als Durchschnittswert interessant ist.

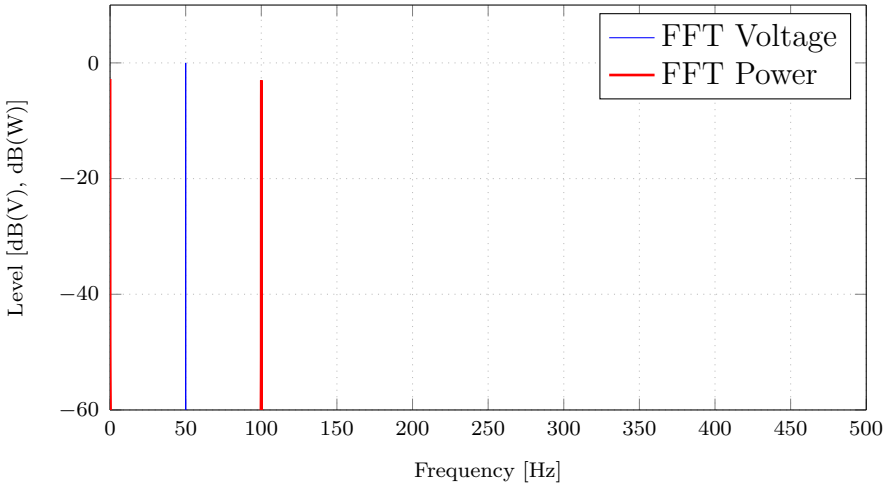


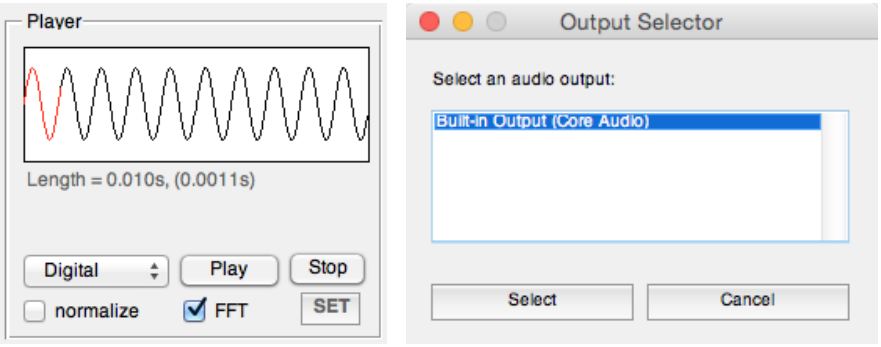
Abbildung 2.20: FFT von Spannung (50 Hz) und resultierende Leistung an 1 Ohm.

2.10 Audio Player

Rechts oben im Fenster befindet sich der Audio-Player. Dieser zeigt das gesamte Testsignal in einem kleinen Diagramm, wobei der gerade im Haupt-Plot angezeigte

Bereich rot markiert ist. Wird der *Play* Button gedrückt, startet die Wiedergabe des gesamten Testsignals. Je nach Auswahl wird dann das Analogsignal, Digitalsignal oder Quantisierungsrauschen abgespielt. Die Leertaste kann alternativ zum Starten bzw. Stoppen der Wiedergabe verwendet werden. Ist die `normalize` Box aktiviert, so wird das Testsignal auf $0\,\mathrm{dB_{FS}}$ gebracht. Diese Option ist für sehr leise Signale bzw. das Quantisierungsrauschen gedacht.

Zur Wiedergabe von Audiodateien wird die Funktion `audioplayer` verwendet. Diese erzeugt ein Objekt für den übergebenen Signalvektor, mit einer floating-point Auflösung von 24 Bit und einer beliebigen Abtastfrequenz. Außerdem wird noch die Audiogerät ID benötigt, welche beim Start oder über den *SET* Button gesetzt wird. Das Fenster in Abb. 2.21(b) erscheint, wenn das Audiogerät der letzten Sitzung nicht verfügbar ist, oder der *SET* Button gedrückt wird.



(a) Player Bereich. (b) Audiogerät Auswahl.

Abbildung 2.21: Player und Audiogerät Auswhlfenster.

2.10.1 Echtzeit Spektrum

Während der Wiedergabe ändert sich der Hauptplot zu einem Spektrumanalyzer. Der Hintergrund wird hellgrau und das aktuelle Spektrum wird ausgegeben. Nachdem das Playback gestartet wurde, durchläuft das Skript eine Schleife mit etwa 15 Aktualisierungen pro Sekunde. In jedem Durchlauf wird ein Puffer der Länge 2048 herausgelöst und das Amplitudenspektrum berechnet. Der Startindex des Puffers wird aus der verstrichenen Zeit berechnet. Auf diese Weise wird garantiert, dass die Anzeige mit der Wiedergabe synchron läuft.

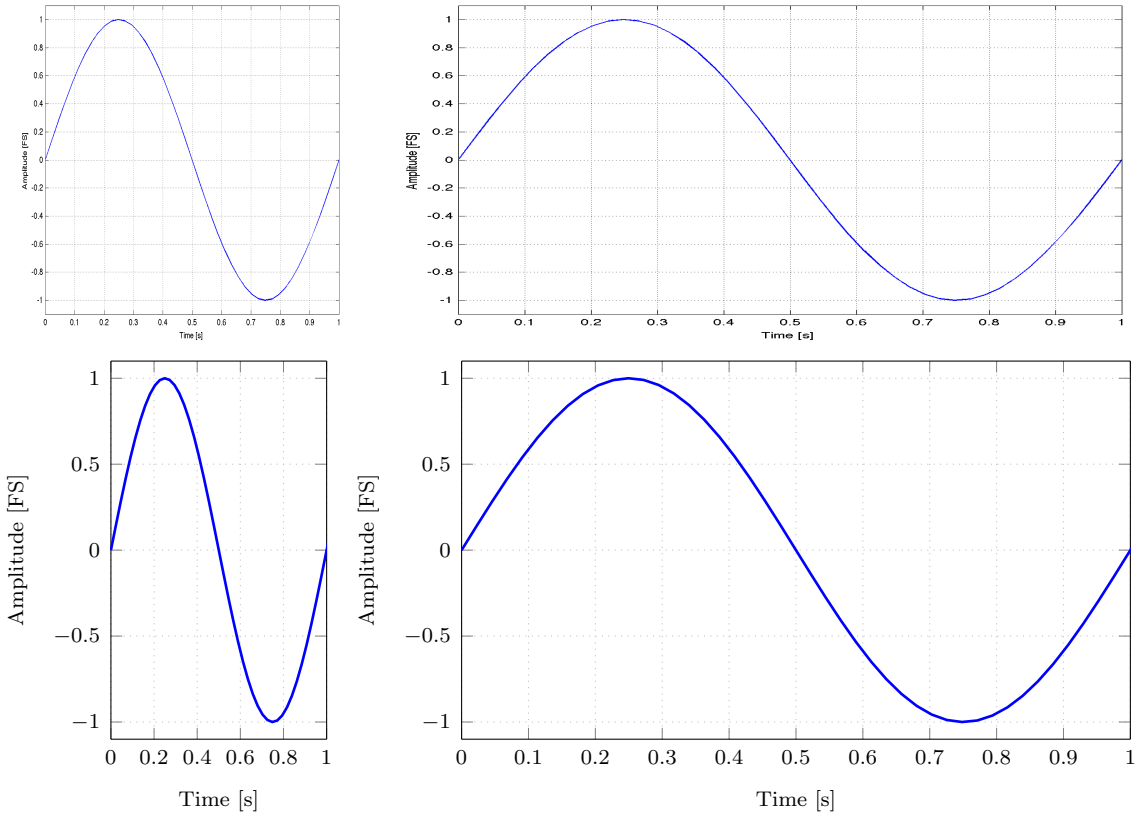
Code 2.14: Berechnung der Signalleistung.

```
1 play(p);     % start playback
2
3 tic;
4
5 while toc < dur
6
7     ti = toc;    % [s] elapsed time
8
```

```
9       start = max(1,floor(ti/Ts));     % buffer start index
10      stop = start+buff-1;             % buffer stop index
11
12      % prevent overrun
13      if(stop > len), stop = len; end
14
15      % fill the buffer with the current frame and calculate
16      % the fft for this frame
17      buffer = sig(start:stop);
18      [fft_abs, dmy, f] = stdFFT(buffer,fs,buff,window);
19
20      axes(axes_fft)    % needed that the plot does not jump to different axes
21      plot(f,fft_abs,'r')
22      set(axes_fft,'Color',[0.8 0.8 0.8]);
23
24      axis([20 fmax -100 10])
25      grid on
26      hold off
27
28      xlabel('Frequenz [Hz]')
29      ylabel('Amplitude [dB]')
30
31      pause(1/update);
32
33      userData = get(handles.figure1,'UserData');
34
35      if(userData.abort == 1)
36          pause(p);
37          userData.abort = 0;
38          set(handles.figure1,'UserData',userData);
39          %updatePlayerPlot(handles,0)
40
41          break;
42      end
43  end
```

2.11 Plot Export

Um die Erstellung eines Laborprotokolls zu erleichtern, besteht die Möglichkeit, alle Diagramme mit sämtlichen Einstellungen zu exportieren. Es stehen zahlreiche Grafikformate zur Verfügung: PDF, PNG, JPG, EPS oder TIKZ. Das Format TIKZ ist ein Vektorformat, welches von LaTeX importiert werden kann. Ein großer Vorteil ist, dass bei späterer Skalierung im Dokument alle Achsen dynamische Beschriftungen haben und Schriftart sowie Schriftgröße immer mit der im Dokument eingestellten übereinstimmt. Die ersten beiden Diagramme verwenden die selbe PNG-Datei, die zweite Reihe verwendet die selbe TIKZ-Datei:



Die TIKZ-Datei hat nur 2 KB und kann beliebig skaliert werden. Wie man oben erkennt, wurde die Skalierung der X-Achse automatisch angepasst. Bei der PNG Datei werden die Beschriftungen und Linienstärken verzerrt. Ein exportierter Plot mit allen GUI-Einstellungen als Text ist in Abb. 2.22 dargestellt.

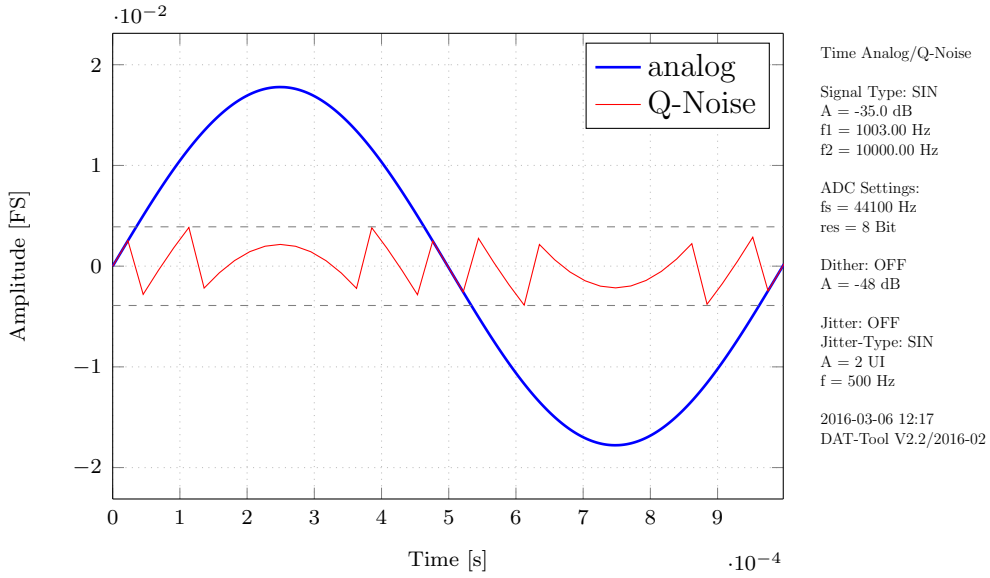


Abbildung 2.22: Automatisch generierte Grafik für Protokolle.

2.12 Plattform Unabhängigkeit

Die Verwendung unterschiedlicher Plattformen bereitet oft Schwierigkeiten. Im Folgenden werden die notwendigen Einstellungen beschrieben, um ein MATLAB GUI sowohl unter Windows, als auch unter Mac OSX betreiben zu können.

Es beginnt bei der Erstellung der Oberfläche mit dem MATLAB Tool `guide`. Sämtliche Steuerelemente müssen auf die Einheit (`Units`) Pixel gestellt werden. Die Standardeinstellung ist Character, allerdings ergeben sich dabei Verzerrungen der Oberfläche. Da dies ein sehr zeitaufwändiger Arbeitsschritt ist, wurde hierzu ein Skript verwendet.

Ein Problem, welches nicht durch eine Einstellung gelöst werden kann, taucht jedoch noch auf. Die Schriftgrößen erscheinen unter Windows wesentlich größer und sprengen zum Teil die Textfelder. Als Lösung wird ein Skript verwendet, das vor jedem Start des Programms die Plattform erkennt, alle Texte in der .fig Datei sucht und die Schriftgröße dementsprechend anpasst. Die Datei wird gespeichert und erst dann aufgerufen. Der Kern dieses Skripts ist in Codeblock 2.15 gezeigt.

Code 2.15: Änderung aller Schriften.

```
1  if ismac
2      FontSize = macSize;
3      FontName = macFont;
4  else
5      FontSize = winSize;
6      FontName = winFont;
7  end
8
9  try
10     H = hgload(FileName);
11     if not(ishandle(H))
12         uiwait(errordlg('Invalid handle. Looks like not a valid figure file?','
               Bad Input', 'modal'));
13     end
14 catch me
15     uiwait(errordlg('Error calling hgload() on this file. Is this a valid fig
           file?','Bad Input', 'modal'));
16 end
17
18 set(findall(H, '-property', 'FontUnits'),'FontName', FontName);
19 set(findall(H, '-property', 'FontUnits'), 'FontSize', FontSize, 'FontUnits', '
       points');
20
21 hgsave(H, FileName);
22 hgclose(H);
```

2.13 Sonstige Funktionen

An dieser Stelle werden noch kleinere Funktionen in der Software kurz beschrieben:

- **Statusleiste für Benutzerinformationen**
  Unten links im Programmfenster befindet sich eine Statusleiste, die Infor-

mationen über die aktuellen Berechnungsschritte liefert. Diese ist besonders bei längeren Testsignalen nützlich, da gewisse Operationen mehrere Sekunden dauern können.

- **Warnhinweise als Pop-Up**
  In Situation, die die Ausführung des Programms verhindern, wird der Benutzer mit einem Pop-Up gewarnt. Zum Beispiel wenn das *Use File* Häkchen gesetzt ist, aber keine Audiodatei geladen wurde.

- **Fortschrittsbalken bei zu langen Testsignalen**
  Überschreitet das Testsignal eine gewisse Länge, so zeigt eine Ladebalken während der Konversion den Fortschritt an.

- **Update der Konfigurationsdatei**
  Diverse Einstellungen (z.B. Audiogeräte, Pfade,...) werden automatisch in einer Datei (`config.m`) gespeichert. Beim nächsten Programmstart werden sie wieder geladen und es ist keine Benutzerinteraktion notwendig.

- **Werkzeuge im Plotfenster**
  Über die Scrollbalken lässt sich die Anzeige sehr einfach und schnell einstellen. Trotzdem stehen alle üblichen Werkzeuge (z.B. Zoom, Pan, Data Cursor) für genauere Analyse zur Verfügung.

- **Dynamische Skalierung der Pegelachsen**
  Je nach verwendeter ADC Auflösung, wird bei Frequenzdarstellungen die Pegelachse so skaliert, dass das Quantisierungsrauschen optimal dargestellt wird.

3 Laborunterlagen

3.1 Zeitliche Diskretisierung

Die Abtastfunktion $\Delta(t)$ ist eine periodisch Folge von äquidistanten Diracimpulsen, die analytisch in folgender Form geschrieben werden kann. Die Periodendauer T ist durch die Abtastfrequenz $f_s = 1/T$ gegeben. Der Dirac-Impuls ist eine Verteilungsfunktion und ist definiert in Gleichung 3.2.

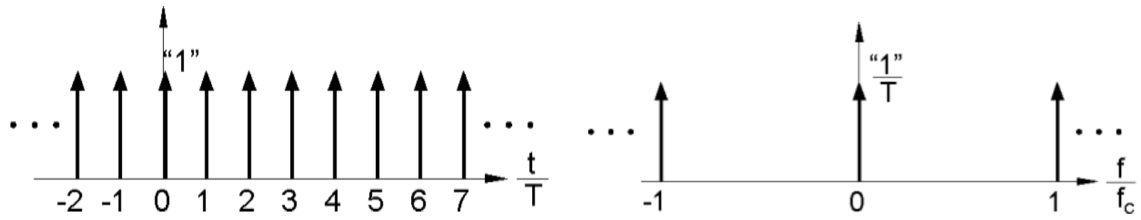
$$\Delta(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{3.1}$$

$$\delta(t) = \begin{cases} \infty & t = 0 \\ 0 & t \neq 0 \end{cases} \tag{3.2}$$

Jede periodische Funktion lässt sich über die Foueierreihenentwicklung darstellen und führt zu einem diskreten Linienspektrum, welches die Grundfrequenz und die ganzzahligen Vielfachen (=Oberwellen oder Harmonischen) enthält. Als Spektrum der Abtastfunktion ergibt sich also eine Folge von Diracimpulsen im Frequenzbereich:

$$\Delta(f) = \frac{1}{T} \sum_{n=-\infty}^{\infty} \delta(f - nf_s) \tag{3.3}$$

Durch Multiplikation der Abtastfunktion $\Delta(t)$ mit einem analogen Signal bleiben nur die Werte an den Abtastzeitpunkten als gewichtete Diracimpulse erhalten, der



(a) Abtastfunktion im Zeitbereich. (b) Abtastfunktion im Frequenzbereich.

Abbildung 3.1: Abtastfunktion, Folge von Diracimpulsen.

Rest des Signales wird ausgeblendet. Gleichung 3.4 zeigt die Zeitdiskretisierung mathematisch, wobei $x(t)$ einem analogen Eingangssignal entspricht.

$$x_d(nT) = x(t) \cdot \Delta(t) = x(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{3.4}$$

Die Multiplikation im Zeitbereich entspricht einer Faltung im Frequenzbereich (und umgekehrt). Daher kann die Abtastung auch durch die Faltung der Spektren beschrieben werden.

$$\begin{aligned} X_d(f) &= X(f) * \Delta(f) \\ &= \int_{-\infty}^{\infty} X(f-\lambda) \frac{1}{T} \sum_{n=-\infty}^{\infty} \delta(f - nf_s)\,\mathrm{d}\lambda \\ &= \frac{1}{T} \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} X(f-\lambda) \cdot \delta(f - nf_s)\,\mathrm{d}\lambda \\ &= \frac{1}{T} \sum_{n=-\infty}^{\infty} X(f - nf_s) \end{aligned} \tag{3.5}$$

Die Auflösung des Faltungsintegrals in Gl. 3.5 zeigt eine Reproduktion des Spektrums $X(f)$ bei allen ganzzahligen Vielfachen $n \cdot f_s$ und eine Gewichtung mit $1/T$. Dies wird in Abb. 3.2 anhand der Funktion $h(t)$ gezeigt.

3.2 Frequenz Aliasing

Ist die größte im Analogsignal enthaltene Frequenz größer als die halbe Abtastfrequenz f_s, kommt es zu Überlappungen der Spiegelfrequenzbänder mit dem Basisfrequenzband des Signals. Dadurch ist eine eindeutige Rekonstruktion nicht mehr

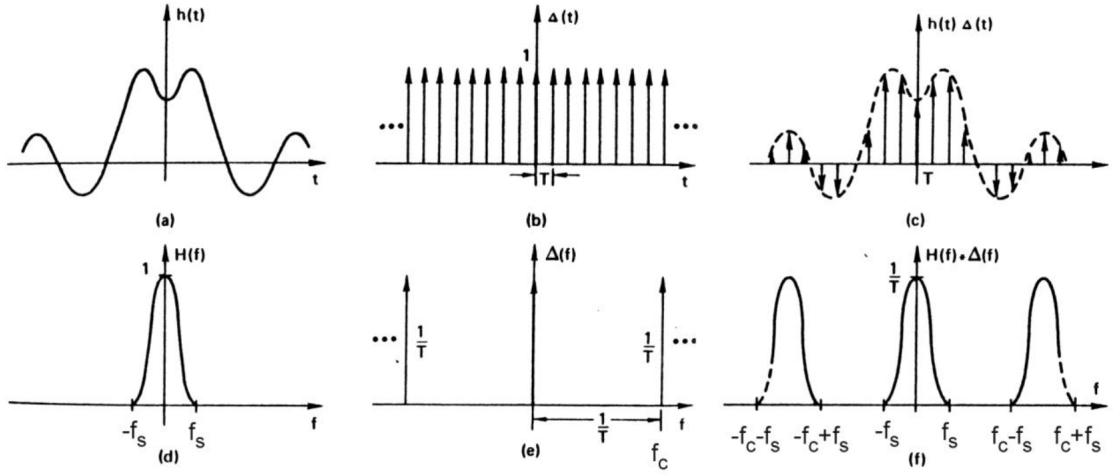


Abbildung 3.2: **(a)** Analoges Signal $h(t)$, **(b)** Abtastfunktion $\Delta(t)$, **(c)** Multiplikation im Zeitbereich, **(d)** Spektrum $H(f)$, **(e)** Spektrum $\Delta(f)$, **(f)** Gefaltete Spektren.

möglich. Dieser Effekt wird Aliasing genannt, es ist ein Fehler, der bei Zeitdiskretisierung entsteht, wenn das Shannon'sche Abtasttheorem ($f_{max} < f_s/2$) nicht eingehalten wird. Vor der Abtastung muss das Signal also auf die halbe Abtastfrequenz bandbegrenzt werden. In Abb. 3.3 ist der Aliasing Effekt dargestellt.

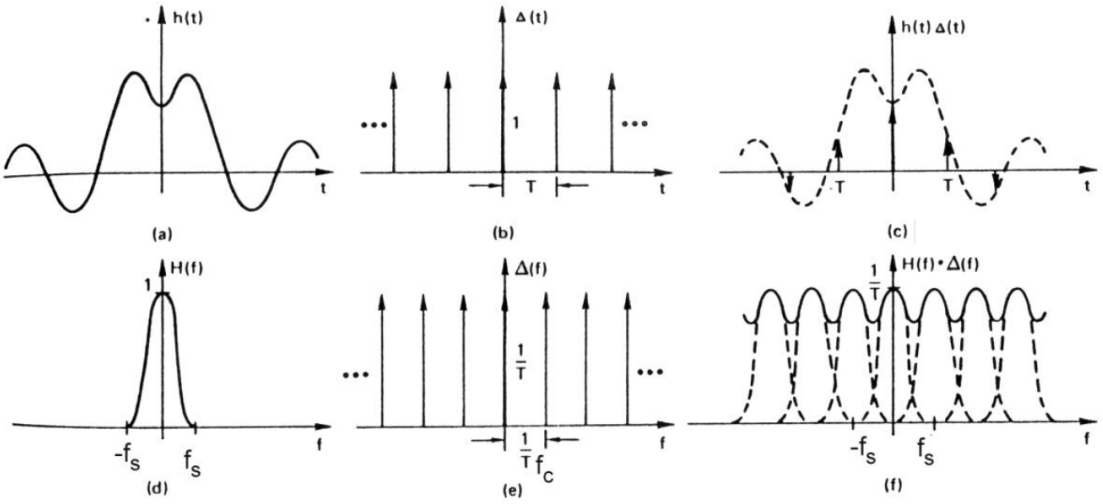


Abbildung 3.3: Durch zu kleine Abtastfrequenz entsteht Überlappung der Frequenzbänder. Das Originalspektrum lässt sich deshalb nicht mehr eindeutig von den gespiegelten Frequenzen trennen.

3.3 Quantisierung

Um ein zeitdiskretes und wertkontinuierliches Signal digital bearbeiten zu können, muss es wertdiskretisiert werden. Die Umsetzerkennlinie ist nichtlinear sondern stu-

fenförmig. Der analoge Eingangswertebereich wird in 2^k gleich große Intervalle der Stufenhöhe Q unterteilt (k entspricht der Wortbreite). Jedem Intervall wird ein Binärcode im Zweierkomplement (bei PCM-Codierung) zugeordnet.

3.3.1 Quantisierungsfehler und Signal/Rauschabstand

Unter folgenden Voraussetzungen darf der Quantisierungsfehler als gleichverteilte Zufallsgröße angesehen werden:

1. Wenn der Quantisierer gut ausgesteuert ist
2. Wenn der Quantisierungsfehler e nicht mit dem Signal korreliert.

Wie aus der Vorlesung *Nachrichtentechnik* bekannt ist, berechnen sich die Rauschleistung und Signal/Rauschabstand (SNR) eines Quantisierers mit 2^k Quantisierungsstufen der Höhe Q:

$$P_Q = \frac{Q^2}{12} \tag{3.6}$$

$$\mathrm{SNR} = 20 \cdot \log(2^k) = 20 \cdot k \cdot \log(2) = 6.02 \cdot k \tag{3.7}$$

Der Signal/Rauschabstand ist daher direkt von der Auflösung des ADC bzw. der Wortbreite des Systems abhängig.

3.3.2 Spektrale Verteilung des Quantisierungsrauschens

Wenn die Voraussetzungen:

1. Das Signal korreliert nicht mit dem Quantisierungsrauschen
2. Der Quantisierungsfehler zweier aufeinander folgenden Werte korreliert nicht

erfüllt sind, dann ist das Quantisierungsrauschen bis $f_s/2$ näherungsweise weißes Rauschen.

Der Wert für die Rauschleistung P_Q ist das Integral über die Leistungsdichteverteilung $G_Q(f)$ über den Frequenzbereich 0 bis $f_s/2$. Nachdem durch oben genannter Annahme die Leistungsdichteverteilung als konstant angenommen werden kann, wird nun G_Q abgeleitet:

$$P_Q = G_Q(f_2 - f_1)$$

$$G_Q = \frac{P_Q}{(f_2 - f_1)} \tag{3.8}$$

$$G_Q \Big|_{dB} = P_Q \Big|_{dB} - 10 \cdot \log(f_2 - f_1) \tag{3.9}$$

Zur Veranschaulichung an dieser Stelle zwei Beispiele:

Beispiel 1: Nutzfrequenzbereich
ADC: 16 Bit bei $f_s = 44.1\,\text{kHz}$
$P_Q = -96\,\text{dB}$ im Frequenzbereich bis $f_s/2 = 22\,050\,\text{Hz}$
$G_Q = -96\,\text{dB} - 43\,\text{dB} = -139\,\text{dB}$

Beispiel 2: Rauschleistung Terzband
Terz-Mittenfrequenz: 1000 Hz, Terzband: 250 Hz
$P_Q = -139\,\text{dB} + 10 \cdot \log(250) = -115\,\text{dB}$

Bei schmalbandigen Messungen zB. mit Terzbandfilter ist es in einem 16Bit-System möglich etwa bei der Linearitätsmessung Signale bis $-115\,\text{dB}$ zu messen.

3.4 Digital/Analog Wandlung

Die Rekonstruktion des Analogsignales geschieht durch Abtrennen der höheren Spektralanteile, die beim Abtastvorgang zum Originalspektrum dazugekommen sind. Dieser Vorgang entspricht einer Multiplikation des Spektrums des Digitalsignals mit einem Rechteckfenster von $-f_s/2$ bis $+f_s/2$ mit der Höhe T ($T = 1/f_s$).

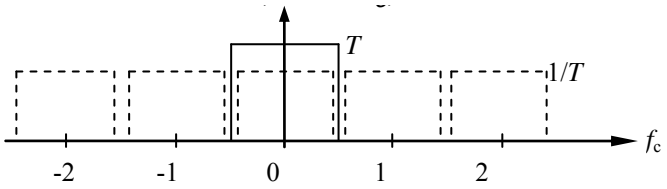


Abbildung 3.4: Rekonstruktionsfilter.

Dieser ideale Tiefpass hat folgende, inverse Fouriertransformation:

$$h(t) = \int_{-\infty}^{\infty} H(f) \cdot e^{j\omega t}\,\mathrm{d}f = \int_{-f_s/2}^{+f_s/2} T \cdot e^{j\omega t}\,\mathrm{d}f =$$

$$= \frac{T}{j2\pi t} \cdot e^{j2\pi f t}\Bigg|_{-f_s/2}^{+f_s/2} = \frac{T}{\pi t}\left(\frac{e^{j\pi f_s t} - e^{-j\pi f_s t}}{2j}\right) = \frac{\sin(\pi f_s t)}{\pi f_s t} \tag{3.10}$$

Mit $\text{sinc}(x) = \sin(\pi x)/\pi x$ ergibt sich für die Impulsantwort des Rechteckfilters $h(t) = \text{sinc}(f_s t)$. Dem Rechteck im Frequenzbereich entspricht eine sinc-Funktion im Zeitbereich. Durch Faltung des Digitalsignals mit der sinc-Funktion im Zeitbereich,

kann das Analogsignal bei Bekanntsein genügend vieler Abtastwerte vollständig wiederhergestellt werden.

Durch Aufsummieren aller gewichteten sinc-Funktionen kann somit die Momentanauslenkung zu jedem beliebigen Zeitpunkt (auch zwischen den Abtastzeitpunkten) berechnet werden.

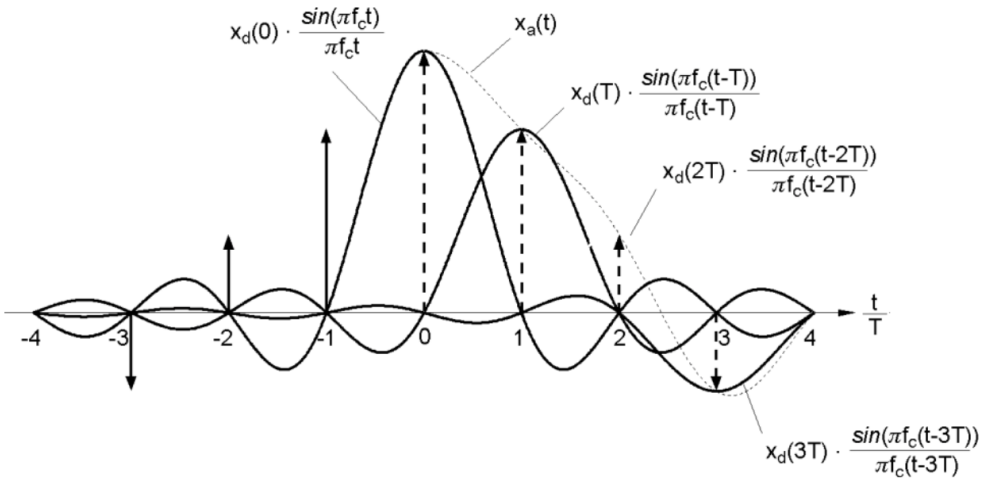


Abbildung 3.5: Rekonstruktion mittels Sinc-Funktion.

3.5 Sample Jitter

Jitter ist ein Fehler von digitalen Signalen, wenn die Abtastzeitpunkte nicht mehr äquidistant sind. Dadurch werden die Amplituden von „falschen“ Zeitpunkten quantisiert. Als Folge davon treten Verzerrungen (=zusätzliche nichtharmonische Frequenzkomponenten) auf.

Häufige Ursachen für Jitter sind asynchrone Datenformate (self clocking), bei welchen das Taktsignal über eine PLL (*phase locked loop*) Schaltung aus dem Datenstrom regeneriert wird. Die PLL hat die Tendenz um die Sollfrequenz zu schwingen. Der so entstandene Jitter kann als periodische Schwingung angenommen werden mit einer Frequenz von einigen hundert Hz bis in den kHz Bereich. Bei Abtasten mit sinusförmig verjittertem Takt ähnelt das Spektrum einer Frequenzmodulation.

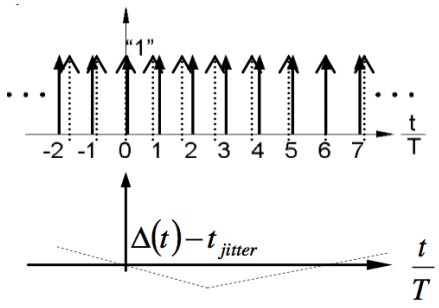
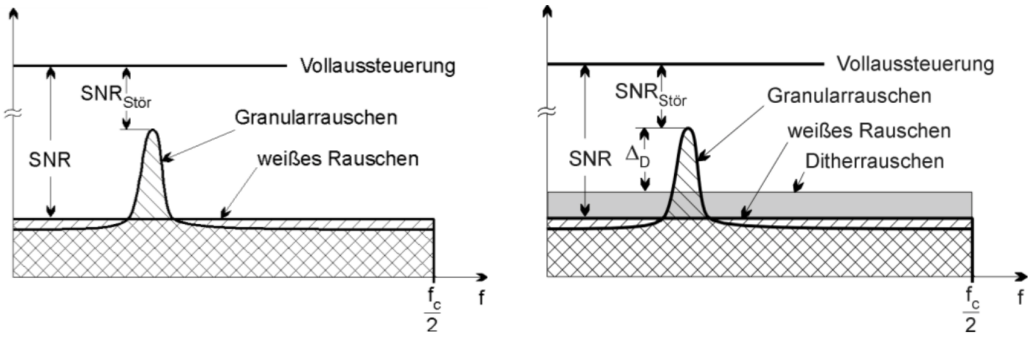


Abbildung 3.6: Jitter - Fehler im Taktsignal

3.5.1 Granularrauschen und Dithering

Granularrauschen ist ein Effekt, der bei sehr kleinen Aussteuerungen und tiefen Frequenzen auftritt. Durch die langsame Signalveränderungen werden wenige Quantisierungsstufen durchlaufen, wobei sich gleiche Quantisierungsstufen sehr oft wiederholen. Dadurch korrelieren Quantisierungsfehler mehrerer aufeinanderfolgender Abtastzeitpunkte. Die Voraussetzungen für weißes Quantisierungsrauschen sind nicht erfüllt und es kommt zu einer schmalbandigen Rauschmodulation mit wesentlich höherem Pegel als das weiße Rauschen. Die schmalbandigen Rauschbänder stehen in nichtharmonischen Zusammenhang mit dem Signal.

Abhilfe kann durch *Dithering* geschaffen werden. Dithering nennt man das Hinzufügen von breitbandigem, niederpegeligem Rauschen zu einem Signal. Dadurch wir die Voraussetzung, dass benachbarte Quantisierungsfehler nicht miteinander korrelieren wiederhergestellt und Rauschmodulation verhindert. Das weiße Grundrauschen wird dabei angehoben, dafür fallen aber schmalbandige Störspitzen weg.



(a) Granularrauschen, schematisch. (b) Dithering bei Granularrauschen.

Abbildung 3.7: Granularrauschen und Dithering.

Redithering spielt bei Wortbreitenreduktion in Digitalsystemen eine wichtige Rolle. Bei der Signalverarbeitung digitaler Signale werden zur Vermeidung des Rechnerrauschens intern große Wortbreiten verwendet (zB. 32 Bit). Für die DA-Umsetzung

(16 bis 20 Bit) muss das Signal wortbreitenreduziert werden. Um hier Rauschmodulationen zu vermeiden wird eben- falls ein Dither hinzugefügt.
Dithersignale können verschiedene Wahrscheinlichkeitsverteilungen haben, zB: Dither mit rechtecksförmiger, dreiecksförmiger und Gauß'scher Wahrscheinlichkeitsverteilung können verwendet werden. Zur Berechnung des Signal-Rauschabstandes darf die Rauschleistung des Dithersignales mit jener des Quantisierungsrauschens addiert werden, weil beide Signale nicht korreliert sind:

$$P_{Q,D} = P_Q + P_D$$

$$\mathrm{SNR}_D = 10 \cdot \log\left(\frac{P_S}{P_Q + P_D}\frac{P_Q}{P_Q}\right) = \mathrm{SNR} - 10 \cdot \log\left(\frac{P_Q + P_D}{P_Q}\right) \tag{3.11}$$

Die Leistung typischer Dithersingale liegt im Bereich des Quantisierungsrauschens $P_D = Q^2/12$. Der Signal/Rauschabstand mit Dithering liegt daher bei etwa:

$$\mathrm{SNR}_D = 10 \cdot \log\left(\frac{2P_Q}{P_Q}\right) = \mathrm{SNR} - 10 \cdot \log(2) = \mathrm{SNR} - 3\,\mathrm{dB} \tag{3.12}$$

Bei 16 Bit würde sich der Signal/Rauschabstand von 96 dB auf 93 dB reduzieren.

3.6 Spektrale Darstellung von Signalen

3.6.1 Fourierreihe

Mit der Fourierreihe können periodische Signal der Periodendauer T_0 dargestellt werden. Durch die Periode T_0 wird die Grundfrequenz $f_0 = 1/T_0$ festgelegt. Im Spektrum der Fourierreihe kommen nur diskrete Frequenzen $n \cdot f_0$ und ein Gleichanteil ($n = 0$) vor. Die Grundfrequenz f_0 kann im diskreten Spektrum als Schrittweite gesehen werden. Periodische Signale haben also ein diskretes (Linien)-Spektrum.

$$y(t) = \sum_{n=-\infty}^{\infty} \alpha_n e^{j2\pi n f_0 t} \tag{3.13}$$

$$\alpha_n = \frac{1}{T_0} \int_{\frac{-T_0}{2}}^{\frac{T_0}{2}} y(t) e^{-j2\pi n f_0 t}\,\mathrm{d}t \tag{3.14}$$

3.6.2 Fourierintegral

Zur spektralen Darstellung von nichtperiodischen Signalen wird das Fourierintegral verwendet. Nichtperiodische Signale haben ein kontinuierliches Spektrum.

$$Y(f) = \int_{-\infty}^{\infty} y(t) \cdot e^{-j2\pi f t}\,\mathrm{d}t \tag{3.15}$$

Die Rücktransformation (Synthesegleichung) lautet:

$$y(t) = \int_{-\infty}^{\infty} Y(t) \cdot e^{j2\pi ft} \, \mathrm{d}f \tag{3.16}$$

Ein wichtiger Zusammenhang, der aus der Fouriertransformation abgeleitet werden kann wurde bereits zuvor angewendet: eine Multiplikation im Frequenzbereich entspricht einer Faltung (Convolution) im Zeitbereich und umgekehrt.

3.6.3 Diskrete- und Fast-Fourier-Transformation

Die diskrete Fouriertransformation (DFT) und die Fast Fourier Transformation (FFT) sind die technische Anwendung der Fourier-Reihe. Einer durch ein Zeitfenster der Dauer T_0 beschränkten Folge von Abtastwerten wird eine Folge von Frequenzwerten im Frequenzbereich zugeordnet. Dazu muss das Eingangssignal zeitlich diskretisiert und mit einem Zeitfenster multipliziert werden. Beide Folgen besitzen die gleiche Länge (N). Die diskreten Frequenzen $k \cdot f_0$ im Spektrum sind durch die Fenstergröße $T_0 = 1/f_0$ vorgegeben.

$$N \cdot T = T_0 \quad \text{bzw.} \qquad N = T_0 \cdot f_s \quad \text{Fensterbreite} \tag{3.17}$$

$$y[k] \rightleftharpoons Y[k] \qquad k = \{0, 1, 2, \ldots, N-1\} \tag{3.18}$$

$$t = k \cdot T = k\frac{1}{f_s} \qquad \text{zeitliche Diskretisierung} \tag{3.19}$$

$$f_{min} = \frac{f_s}{N} \qquad \text{kleinste darstellbare Frequenz} \tag{3.20}$$

Die Fast Fourier Transformation (FFT) ist ein sehr effizienter und weit verbreiteter Algorithmus zur Berechnung der diskreten Fouriertransformation, der auf geringe Rechenzeit optimiert ist. Unterstützte Fensterbreiten bei der FFT sind ganzzahlige Potenzen von 2 ($N = 2^m$; $m = 1, 2, 3, \ldots$).

3.6.4 Lattenzaun- oder Leck-Effekt

Der Lattenzaun- oder Leck-Effekt tritt bei der diskreten Fouriertransformation auf. Der Grund dafür ist das Zeitfenster, welches die darstellbaren Frequenzen $k \cdot f_0$ festlegt. Ist die gemessene Frequenz kein ganzzahliges Vielfaches von f_0 ($=1/T_0$), wird im Spektrum eine zu kleine Amplitude (bzw. Leistung) angezeigt. Die dargestellte Amplitude/Leistung wird auf die benachbarten Spektrallinien (Frequenzbins) verteilt. Die größte Abweichung ergibt sich für Frequenzen $f = \frac{2n+1}{2} \cdot f_0$ mit ca. 36% der ursprünglichen Amplitude. Der Fehler im Spannungspegel liegt somit bei $-3.9\,\mathrm{dB}$. Der Lattenzauneffekt kann durch spezielle Gewichtung des Zeitfensters oder durch

ein genügend großes Zeitfenster vermindert werden.

Beispiel:
Berechnung der kleinsten, auflösbaren Frequenz bei einer Fensterbreite von 2^10 Punkten und einer Abtastfrequenz von 44 100 Hz:

$$f_0 = \frac{f_s}{N} = \frac{44100}{1024} = 43.06\,\text{Hz}$$

Berechnung von 2 Frequenzen, eine davon soll genau aufgelöst werden können, die andere genau im Bereich zwischen 2 Spektrallinien liegen:

$$k = 100: \qquad f_1 = k \cdot f_0 = 4306.6406\,\text{Hz}$$

$$k = 100.5: \qquad f_2 = k \cdot f_0 = 4328.1738\,\text{Hz}$$

Für exakte Messungen, bei welchen die Auswertung über die FFT läuft, sind daher Messfrequenzen von $k \cdot f_0$ zu wählen mit ganzzahligem k.

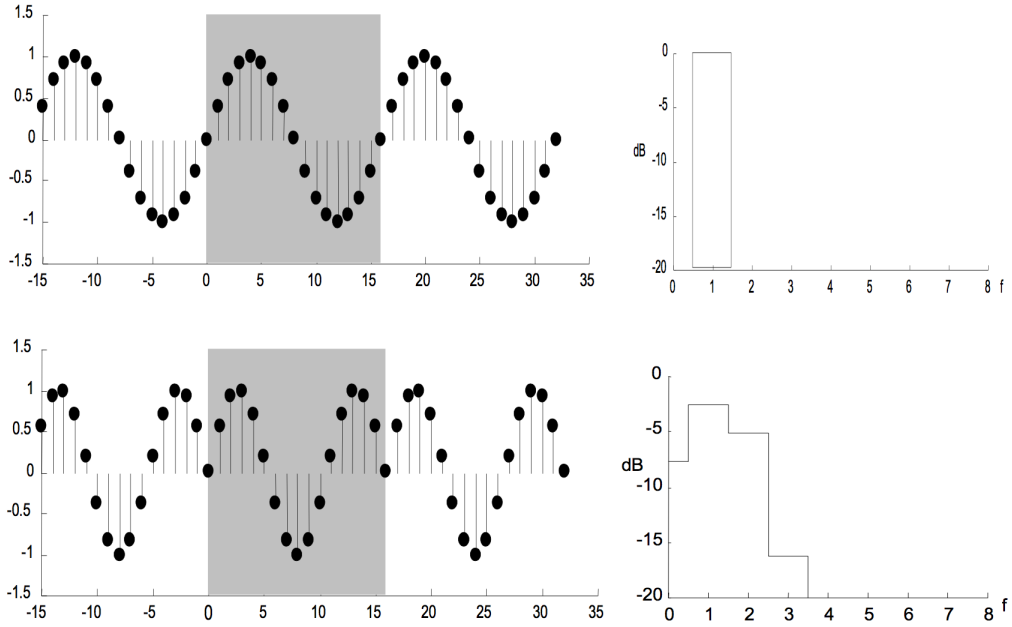


Abbildung 3.8: Lattenzauneffekt bei $N = 16$, $f_s = 16\,\text{Hz}$, $f_1 = 1\,\text{Hz}$, $f_2 = 1.5\,\text{Hz}$.

3.6.5 Amplitudenverteilung – Leistungsdichteverteilung

Die FFT liefert eine Amplitudenverteilung. Jeder Wert der Amplitudenverteilung (Bin) steht näherungsweise für die Amplituden aller Frequenzen innerhalb des umliegenden Frequenzbandes f_0. Um von der Amplitudenverteilung auf eine Leistungsdichteverteilung zu kommen, ist es daher notwendig f_0 zu kennen. Wie zuvor beschrieben, hängt die Breite der einzelnen Frequenzbänder unmittelbar mit der Fensterbreite zusammen. Um auf die Leistungsdichteverteilung zu kommen, nimmt man

innerhalb eines Frequenzbandes ($k \cdot f_0 - \frac{f_0}{2} \leq f \leq k \cdot f_0 + \frac{f_0}{2}$) eine konstante spektrale Verteilung an, das Quadrat des Amplitudenwertes steht für die Leistung (an $1\,\Omega$) innerhalb des Frequenzbandes:

$$P[k] = A^2[k] \qquad \text{für alle } k$$

$$\frac{P[k]}{f_0} = G[k]$$

$$G[k] \Big|_{dB} = P[k] \Big|_{dB} - 10 \cdot \log(f_0) \tag{3.21}$$

Beispiel:
Fensterbreite 2^{10} Punkte, Abtastfrequenz 44 100 Hz. $f_0 = 43\,\text{Hz}$, wie oben berechnet. Der Spannungspegel bei 4306.6406 Hz sei $-30\,\text{dB}$. Wo liegt der Wert für die Leistungsdichteverteilung in diesem Frequenzband?

$$G_{4306\,\text{Hz}} = -30\,\text{dB} - \log(43) = -46\,\text{dB}$$

3.7 Aufgabenstellungen

3.7.1 Aufgabe 1 - Aliasing

1. Wählen Sie Frequenz und Abtastrate so, dass eine Alias-Frequenz von 440 Hz auftritt. Überprüfen Sie das Resultat graphisch und akustisch. Überprüfen Sie das Ergebnis mit dem ebenfalls mit dem Klavier.

2. Wählen Sie Anfangs- und Endfrequenzen eines linearen Frequenzsweeps so, dass Aliasing auftritt. Der Sweep soll zu Beginn korrekt ausgegeben werden (kein Aliasing) und am Ende um $3 \cdot f_s$ gespiegelt sein. Überprüfen Sie das Resultat graphisch (Echtzeit FFT während der Wiedergabe) und akustisch.

3. Wählen Sie die Checkbox *use File* an und laden sie die Audiodatei `floeten`. Durch Eingabe einer neuen Samplingfrequenz im Sample Rate Converter (SRC) Feld (z.B. 4000 Hz, Die Checkbox SCR muss ausgewählt sein) kann das eingelesene Audiosignal unterabgetastet werden. Es kann mit und ohne Bandbegrenzung (Anti-Alias Filter) verarbeitet werden. Schätzen Sie die Auswirkungen ab und probieren Sie beide Einstellungen mit mehreren Abtastraten aus (Tipp: Eingabe von Ausdrücken wie '44100/4' ist möglich). Überprüfen Sie akustisch die Auswirkungen und beschreiben Sie diese im Protokoll.

3.7.2 Aufgabe 2 - Leistungsdichte Spektrum

1. Wie sieht der Quantisierungsfehler bei Vollaussteuerung und 8 Bit Wortbreite aus? In welchem Bereich liegt die Amplitude des Quantisierungsfehlers? Sind die Voraussetzungen für $SNR = 6.02 \cdot k$ erfüllt? [Signal?]

2. Berechnen Sie den erwarteten Sinal-Rauschabstand (SNR) und die Quantisierungsrauschleitungsdichte für die gewählte Auflösung. Welchen Einfluss hat die FFT-Fensterbreite auf das Amplitudenspektrum bzw. die Leistungsdichteverteilung des Quantisierungsrauschens? Überprüfen Sie den berechneten SNR und Quantisierungsrauschleistungsdichteverteilung bei verschiedenen FFT Fensterbreiten mit dem Simulationsprogramm. Finden Sie eine Frequenz bei der der maximale Leck-Effekt auftritt und protokollieren Sie die Amplitudendifferenz. Tipp: Stellen Sie sicher, dass das ein rechteckiges FFT-Fenster verwendet wird.

3. Berechnen Sie die Rauschleistung in einem Terzband und überprüfen Sie das Ergebnis mit dem Simulationsprogramm. Hinweis: Terzband: $f_{oben} = \sqrt[6]{2} \cdot f_{sig}$, $f_{unten} = f_{sig}/\sqrt[6]{2}$. Im Plot Menü unter `Terz SNR` wird die gesamte Rauschleistung und die im aktuellen Terzband angezeigt.

3.7.3 Aufgabe 3 - Jitter

1. Wählen Sie eine Frequenz im Grundtonbereich ($< 3000\,\text{Hz}$), eine Jitterfrequenz (100 bis $2000\,\text{Hz}$) und eine Jitteramplitude (0 bis $10\,\text{UI}$). Schätzen Sie die Auswirkungen bei Abtastung mit sägezahnförmigem und sinusförmigem Jitter ab.

2. Überprüfen Sie Ihre Annahmen mittels Spektralanalyse und Hörprobe. Stellen Sie die Auswirkungen auf das Signalspektrum fest. Ab welcher Jitter-Amplitude können Sie eine Veränderung hören?

3.7.4 Aufgabe 4 - Dither

1. Schätzen Sie die Auswirkungen des Abtastvorganges für eine tiefe Frequenz ($80\,\text{Hz}$) mit niedriger Amplitude ($-80\,\text{dB}$) und $16\,\text{Bit}$ Auflösung mit und ohne Dither ab. Sind die Voraussetzungen für $SNR = 6.02 \cdot k$ erfüllt? Überprüfen Sie Ihre Annahmen mit dem Simulationsprogramm. Wählen Sie ein Dithersignal mit einer Amplitude von $2\,\text{dB}$ über dem Quantisierungsrauschen. Führen Sie Hörproben des analogen sowie digitalen Signals durch (Hinweis: Nutzen Sie die Möglichkeit der Normalisierung, achten Sie auf die Lautstärke).

2. Berechnen Sie der Autokorrelation des Quantisierungsfehlers mit und ohne Dither. (Hinweis: Plot Menü `Correltion Q-Noise`, verwenden Sie den vertikalen Slider um den Signalausschnitt zu verlängern).

3. Laden Sie die Audiodatei `kontrabass.wav`. Dämpfen Sie die Signalamplitude um $-96.32\,\text{dB}$ (Textfeld unter Button) und führen Sie Hörproben mit und ohne Dithersignal durch.

4. Wählen Sie ein Testsignal ohne Dither mit so kleiner Amplitude, dass Sie gerade keine Quantisierungsstufen mehr sehen und somit ein digitales Nullsi-

gnal erhalten (alle Abtastwerte sind 0). Wiederholen Sie die Abtastung mit Dithering und führen Sie Hörproben durch.

Literaturverzeichnis

[1] F. Zotter, "Erstellen von Laborunterlagen zu Messung des Störverhaltens in nachrichtentechnischen Systemen und Implementierung eines Simulationsprogrammes in MATLAB."

[2] P. Majdak, "Algorithmen in Akustik und Computermusik," 2007. Vorlesungsunterlagen.

[3] G. Graber, "Digitale Audiotechnik 1," 2009.

A Anhang

A.1 Funktionsverzeichnis

```
[d_sig, q] = adc(sig, res, rounding)
```

Quantisiert ein Signal auf eine gegebene Auflösung `res` und verwendet dazu die Rundungsmethode `round`.

```
[power] = calcPower(sig)
```

Berechnet die Leistung eines Signals in dB.

```
changeGuiFontSize(FileName, winSize, winFont, macSize, macFont)
```

Erkennt das laufende Betriebssystem und passt die Schriftgrößen in der Figure Datei `FileName` dem entsprechend an.

```
[out_str] = checkInput(in_str, min_val, max_val, def_val)
```

Erwartet den String eines Textfeldes und prüft den Wertebereich, ersetzt , durch . und setzt den Wert bei falscher Eingabe auf `def_val`.

```
[sw,isw]=expsweep(fstart,fend,T,fs,phi)
```

Berechnet einen Sinus-Sweep, sowie den korrespondierenden inversen Sweep von `fstart` zu `fstop` mit der Länge `T`, der Abtastrate `fs` und einer optionalen Phasenverschiebung `phi`.

```
[dither] = genDither(A_dB,dur,fs,type)
```

Generiert ein Dither-Signal mit der Amplitude `A_dB`, der Länge `dur`, der Abtastrate `fs` und einer Verteilung `type`.

```
[signal_jit] = genJitter(sig,fs,A,f,type)
```

Berechnet Jitter für ein gegebenes Signal `sig` der Abtastrate `fs`. Die Verzerrung der Zeitachse hat die Amplitude `A`, die Frequenz `f` und die Signalform `type`.

```
[a_sig, t] = genSignal(sigType, A_sig, f_sig, f_stop, dur_sig, Ts, numHarm)
```

Generiert ein Signal vom Typ `sigType` mit der Amplitude `A_sig`, der Frequenz `f_sig`, der Stop-Frequenz `f_stop` (nur für Sweeps), der Länge `dur_sig`, der Abtastperiode `Ts` und der Anzahl an Oberwellen `numHarm` (nur für Rechteck und Sägezahn).

```
fig = genTightFigure(width,height)
```

Erzeugt eine Figure mit den Dimensionen `width` und `height` in Pixel. Dies wird für das Exportieren von PDF-Dateien benötigt.

```
genTikz(FileName,dx_min,digits)
```

Erzeugt eine TIKZ-Datei mit dem Namen `FileName`, der X-Auflösung `dx_min` und der Anzahl an Dezimalstellen pro Punkt `digits`.

```
[audioDev] = getAudioDevID(lastInput,lastOutput)
```

Erzeugt eine Struktur `audioDev` die ID und Namen von Ein- und Ausgangsgeräten beinhaltet. Die Argumente `lastInput` und `lastOutput` werden bei der Wahl bevorzugt. Sollten sie nicht zur Verfügung stehen, so wird der Benutzer gebeten ein Audiogerät auszuwählen.

```
[sig] = normalize(sig)
```

Normalisiert ein Signal, sodass Maximalwert zu +1 wird, bzw. Minimalwert zu -1 wird.

```
playerFFT(handles)
```

Erwartet den GUI-Handle `handle` und spielt das ausgewählte Signal über das gewählte Audiogerät ab. Zusätzlich wird eine Echtzeit-FFT berechnet und im Hauptplot angezeigt.

```
plotAD(handles)
```

Erwartet den GUI-Handle `handle` und stellt das Analog- und Digital-Signal im Zeitbereich dar.

```
plotAQNoise(handles)
```

Erwartet den GUI-Handle `handle` und stellt das Analog-Signal sowie das Quantisierungsrauschen im Zeitbereich dar.

```
plotCorrDither(handles)
```

Erwartet den GUI-Handle `handle` und stellt die Korrelation des Dither-Signals dar.

```
plotCorrQNoise(handles)
```

Erwartet den GUI-Handle `handle` und stellt die Korrelation des Quantisierungsrauschens dar.

```
plotFftAQNoise(handles)
```

Erwartet den GUI-Handle `handle` und stellt die FFT des Analogsignals, sowie des Quantisierungsrauschens dar.

```
plotFftDither(handles)
```

Erwartet den GUI-Handle `handle` und stellt die FFT des Dither-Signals dar.

```
plotFftPower(handles)
```

Erwartet den GUI-Handle `handle` und stellt die FFT von Spannung und Leistung dar.

```
plotHistA(handles)
```

Erwartet den GUI-Handle `handle` und stellt das Histogramm des Analog-Signals dar.

```
plotPsdAQNoise(handles)
```

Erwartet den GUI-Handle `handle` und stellt das Leistungsdichtespektrum von Analog-Signal und Quantisierungsrauschen dar.

```
plotQNoise(handles)
```

Erwartet den GUI-Handle `handle` stellt das Quantisierungsrauschen in Zeitbereich dar.

```
plotSpecA(handles)
```

Erwartet den GUI-Handle `handle` und stellt ein Spektrogramm des Analog-Signals dar.

```
plotSpecD(handles)
```

Erwartet den GUI-Handle `handle` und stellt ein Spektrogramm des Digital-Signals dar.

```
plotTerzSnr(handles)
```

Erwartet den GUI-Handle `handle` und stellt die FFT des Analog-Signals sowie Terzband gefiltertes Quantisierungsrauschen dar.

```
plotVoltPower(handles)
```

Erwartet den GUI-Handle `handle` und stellt Spannung und Leistung im Zeitbereich dar.

```
[FileStr] = readTextFile(filePath,waitbar)
```

Liest die ASCII-Datei `filePath` als String ein und zeigt einen Fortschrittsbalken, wenn `waitbar` gleich 1 ist.

```
[fft_log df f] = stdFFT(sig, fs, order, window)
```

Berechnet $20 \cdot \log(|\mathrm{FFT}(sig)|)$ eines Signals `sig` mit der Abtastrate `fs` und der Fensterfunktion `window`. Zurück geliefert werden außer dem Amplitudenspektrum `fft_log` noch die Frequenzauflösung `df` und der Frequenzvektor `f`.

```
error = updateConfigFile(FilePath, param, value)
```

Aktualisiert den Eintrag `param` mit dem Wert `value` in der Config-Datei `FilePath`.

```
updatePlayerPlot(handles)
```

Erwartet den GUI-Handle `handle` und aktualisiert den Übersichts-Plot im Playerbereich, zum Beispiel nach Bewegen eines Scroll-Balkens.

```
updatePlotLimits(handles)
```

Erwartet den GUI-Handle `handle` und berechnet die aktuellen Plot-Grenzen `x_min`, `x_max`, `y_min` und `y_max`. Diese werden im GUI-Handle gespeichert.