Labor 1 Dokumentation

Inhalt

[Importierte Bibliotheken 2](#_Toc168845824)

[Plotten 3](#_Toc168845825)

[Aufgabe 1 4](#_Toc168845826)

[Aufgabe 2 6](#_Toc168845827)

[Aufgabe 3 11](#_Toc168845828)

[Aufgabe 4 14](#_Toc168845829)

[Aufgabe 5 16](#_Toc168845830)

[Aufgabe 6 19](#_Toc168845831)

## Importierte Bibliotheken

1. NumPy ist eine Bibliothek für wissenschaftliches Rechnen in Python. Sie bietet Unterstützung für große, mehrdimensionale Arrays und Matrizen sowie eine große Sammlung mathematischer Funktionen, um an diesen Arrays Operationen durchzuführen.

NumPy wird verwendet, um Audiodaten als Arrays zu laden und zu manipulieren z.B. das Berechnen von Fourier-Transformationen

1. SciPy ist eine Bibliothek, die häufig für wissenschaftliche und technische Berechnungen verwendet wird. Der Untermodul `scipy.io.wavfile` ermöglicht das Lesen und Schreiben von WAV-Dateien. WAV-Dateien sind ein Format für Audiodateien, das Roh-Audiodaten speichert. Um zu verarbeiten und zu analysieren. Die Funktion `wavfile.read` lädt eine WAV-Datei und gibt die Abtastrate und die Audiodaten zurück.
2. `pyplot` ist ein Modul in der Matplotlib Bibliothek, das eine MATLAB-ähnliche Schnittstelle bietet. Es wird verwendet, um 2D-Grafiken zu erstellen, die in einer Vielzahl von Formaten und interaktiven Umgebungen eingebettet werden können. In diesem Fall wird `pyplot` verwendet, um die Audiodaten zu visualisieren, indem wir unsere Grafiken zu Plotten was einen Überblick über die Amplitudenänderungen im Laufe der Zeit gibt.
3. IPython.display bietet eine reichhaltige Palette von Funktionen, um die Darstellung von Ergebnissen im Jupyter-Notebook zu verbessern. In diesem Fall wird `IPython.display` verwendet, um Audiodaten direkt im Notebook abzuspielen.

## Plotten

Zur Visualisierung unserer Signale und Töne verwenden wir das `pyplot` Modul mit der Abkürzung plt.

Hier sind einmal die Befähle die wir im Laufe des Projektes zur Grafik Erstellung verwendet haben und deren Erklärungen:

* `plt.figure(figsize=(width, height)) `: Erstellt eine neue Abbildung mit einer Breite und einer Höhe in Zoll. Beispiel : ` plt.figure(figsize=(10, 4)) `eine neue Abbildung mit einer Breite von 10 Zoll und einer Höhe von 4 Zoll.
* `plt.plot(x, y)`: Erstellt eine einfache 2D-Linie. `x` und `y` sind Listen oder Arrays von Koordinaten. Beispiel: `plt.plot([1, 2, 3], [4, 5, 6])` in unseren Fall verwenden wir den Zeitvektor und das erstellte Signal mit seinen Amptitudenwerte auf die Zeit
* `plt.xlabel('Text')` und `plt.ylabel('Text')`: Beschriftet die x- und y-Achse. Beispiel: `plt.xlabel('X-Achse')` und `plt.ylabel('Y-Achse')`
* `plt.title('Text')`: Fügt einen Titel zum Diagramm hinzu. Beispiel: `plt.title('Mein Diagramm')`
* `plt.xlim(left, right)`: wird verwendet, um die Grenzen der x-Achse des Diagramms festzulegen. Beispiel: plt.xlim(0,0.1) diesem Beispiel wird die x-Achse auf den Bereich von 0 bis 0.01 beschränkt.
* `plt.legend()`: Fügt eine Legende hinzu. Wird oft zusammen mit dem `label`-Argument in Plot-Befehlen verwendet. Beispiel: plt.plot(… , label='Linie 1') plt.legend()
* `plt.grid()`: Fügt ein Gitternetz zum Diagramm hinzu. Beispiel: `plt.grid(True)`
* `plt.show()`: Zeigt das erstellte Diagramm an. Dies ist der letzte Befehl, den man ausführt, nachdem alle anderen Befehle ausgeführt wurden. Beispiel: `plt.show()`

## Aufgabe 1

*Erzeugen Sie eine Audio-Datei auf ihrem PC mit folgendem Inhalt: "Dies ist eine Suchmaschine". Verwenden Sie als Abtastrate 𝑓= 16𝑘𝐻𝑧. Erstellen Sie anschließend ein Jupyter-Notebook das die erstellte Audio-Datei lädt, abspielt und das Sprachsignal als Funktion der Zeit plottet. Können Sie im geplotteten Sprachsignal Teile ihres Satzes wieder erkennen? Analysieren Sie ihr Sprachsignal schrittweise und beschreiben Sie ihr Ergebnis textuell.*

WAV-Dateien enthalten Roh-Audiodaten, die oft in PCM (Pulse Code Modulation) kodiert sind. Mithilfe von:

wavfile.read(audio\_path)

wird es uns ermöglicht die Audiodatei einzulesen und gibt die Abtastrate aus. Die Abtastrate gibt an, wie oft das analoge Signal pro Sekunde abgetastet wurde. In unseren Fall sind es 16kHz.

Danach erzeugen wir den Zeitvektor:

t = np.arange(0, len(signal)) / rate

der die Zeitpunkte für jeden Abtastwert des Audiosignals enthält.

* `np.arange(0, len(signal)) `: Erzeugt ein Array von 0 bis zur Länge des Signals.
* `/ rate`: ermöglicht die Skalierung des Arrays durch die Abtastrate, um die tatsächlichen Zeitwerte in Sekunden zu erhalten.

**Abtasttheorem**

Das Abtasttheorem (Nyquist-Shannon-Abtasttheorem) besagt, dass ein kontinuierliches Signal vollständig rekonstruiert werden kann, wenn es mit einer Rate abgetastet wird, die mindestens doppelt so hoch ist wie die höchste Frequenzkomponente des Signals. Die Formel dafür ist:

wobei die Abtastrate und die höchste Frequenz im Signal ist.

**Zeitvektor**

Der Zeitvektor wird berechnet, indem die Anzahl der Abtastwerte durch die Abtastrate geteilt wird:

wobei die Abtastwerte und die Abtastrate ist.

**Visualisierung (**Plotten der Audiodatei**)**

Der Plot zeigt das Audiosignal im Zeitbereich, was einen Überblick über die Amplitudenänderungen im Laufe der Zeit bietet. Hierfür wurde das `pyplot` Modul mit der Abkürzung plt verwendet. Erklärungen hierzu finden Sie unter der Überschrift “Plotten “

Ein Bild, das Diagramm, Reihe, Text enthält.

Automatisch generierte Beschreibung

Abbildung 1 Das Audiosignal im Zeitbereich

**Abspielen der Audio-Datei**

Hierfür wird `IPython.display` verwendet mit der Abkürzung ipd.

ipd.Audio(audio\_path)

Diese Funktion lädt die Audiodatei und stellt eine Audioausgabe im Jupyter-Notebook bereit, die durch Klicken auf den Abspielknopf abgespielt werden kann.

**Analyse**

Anhand der Ausschläge der Grafik kann man die Einzelnen Wörter und Pausen des Satzes erkennen und sogar in unseren Fall das Hintergrundgeräusch am Anfang der Aufnahme. Man erkennt im ersten Großen Block das Wort "Dies" ganz deutlich und auch die Lautstärke der Stimme, die im Laufe des Satzes abnimmt erkennt man dadurch das die Ausschläge nicht so groß sind wie am Anfang. Ebenfalls kann man die einzelnen Silben des Satzes erkennen.

## Aufgabe 2

*Generieren Sie jetzt ein harmonisches Signal bestehend aus dem Kammerton (𝑓=440Hz) und seiner 2. und 3. Harmonischen. Die Amplitude A und die Zeitdauer 𝑡 seien gegeben durch: A= 1.0 und 𝑡 =1s. Tasten Sie das erstellte Signal mit einer Abtastfrequenz f = 20 ∗ f ab und speichern Sie es in einer wave-Audiodatei ab. Plotten Sie das resultierende Oszillogramm und den Kammerton. Lesen Sie die Audio-Datei ein und geben Sie den Klang aus.*

**Parameterdefinition**

* `a` (Amplitude): Dies ist die maximale Auslenkung der Schwingung eines Signals. Eine Amplitude von 1.0 bedeutet, dass die Schwingung zwischen -1.0 und 1.0 oszilliert.
* `td` (Zeitdauer): Die Dauer des generierten Signals in Sekunden.
* `f0` (Grundfrequenz): Die Grundfrequenz des Tons, die in Hertz (Hz) angegeben wird. Sie bestimmt die Tonhöhe des erzeugten Signals. Ein Wert von 440 Hz entspricht dem Kammerton A.
* `fa` (Abtastfrequenz): Die Frequenz, mit der das Signal abgetastet wird. Sie ist hier 20-mal die Grundfrequenz, um eine hohe Abtastrate und damit eine genaue Darstellung des Signals zu gewährleisten. Abtastfrequenz muss gemäß dem Nyquist-Shannon-Abtasttheorem mindestens doppelt so hoch wie die höchste Frequenzkomponente des Signals sein, um Aliasing zu vermeiden.

**Zeitvektor**

t = np.linspace(0, td, int(td \* fa), endpoint=False)

Der Zeitvektor wird mit `np.linspace` erstellt, um sicherzustellen, dass die Abtastpunkte gleichmäßig über die gewünschte Dauer verteilt sind. Es erzeugt einen gleichmäßig verteilten Vektor von Zeitpunkten von 0 bis `td` Sekunden. Die Anzahl der Punkte ist das Produkt aus `td` und `fa`, was sicherstellt, dass die Abtastrate `fa` eingehalten wird. `endpoint=False` bedeutet, dass der Endwert `td` nicht eingeschlossen ist.

**Generierung der Harmonischen**

def harmonic\_generator(k , t):

    return a \* np.sin(2 \* np.pi \* k \* f0 \* t)

Diese Funktion generiert die k-te harmonische Schwingung basierend auf der Grundfrequenz `f0`.

* k: Der Harmonische-Index. `k=1` bedeutet die Grundfrequenz, `k=2` die erste Obertonfrequenz usw.
* t: Der Zeitvektor.
* Rückgabewert: Ein Array, das die Amplitudenwerte der k-ten harmonischen Schwingung für jeden Zeitpunkt `t` enthält. Also unseren Fertigen Ton

Eine harmonische Schwingung wird mit der Sinusfunktion generiert. Die Formel zur Generierung der harmonischen Schwingung lautet:

wobei: die Amplitude ist, der Harmonische-Index ist, die Grundfrequenz ist und die Zeit ist. Der Faktor

stellt sicher, dass die Frequenz der Schwingung ein ganzzahliges Vielfaches der Grundfrequenz ist, was die Definition von Harmonischen erfüllt.

**Generierung des Kammertons und der Harmonischen**

`chambertone`: Dies ist das generierte Signal des Kammertons, das nur die Grundfrequenz enthält. Hier wird die Funktion `harmonic\_generator` mit dem Harmonischen-Index `k=1` und dem Zeitvektor `t` aufgerufen, um die Grundfrequenz zu erzeugen.

wobei die Grundfrequenz (440 Hz) und die Amplitude (1.0) ist.

`secound\_harmonic `: Ist die generierte zweite Harmonische des Grundtons mit 440Hz. Hier wird die Funktion `harmonic\_generator` mit dem Harmonischen-Index `k=2` und dem Zeitvektor `t` aufgerufen, um die Grundfrequenz zu erzeugen.

`third\_harmonic `: Ist die generierte dritte Harmonische des Grundtons mit 440Hz. Hier wird die Funktion `harmonic\_generator` mit dem Harmonischen-Index `k=3` und dem Zeitvektor `t` aufgerufen, um die Grundfrequenz zu erzeugen.

**Visualisierung(**Plotten**)**

Hierfür wurde das `pyplot` Modul mit der Abkürzung plt verwendet. Dies ermöglicht eine visuelle Inspektion der Schwingungen der erzeugten Töne. Erklärungen hierzu finden Sie unter der Überschrift “Plotten “

**Ein Bild, das Text, Reihe, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung**

Abbildung 2 Oszillogramm des Kammertons im Zeitbereich

**Ein Bild, das Text, Reihe, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung**

Abbildung 3 Oszillogramm der 2. Harmonischen

Ein Bild, das Text, Reihe, Schrift, Diagramm enthält.

Automatisch generierte Beschreibung

Abbildung 4 Oszillogramm der 3. Harmonischen

Hier kann man deutlich die Veränderung der Kurven sehen. Alle Harmonischen sind periodisch mit der Grundfrequenz von 440 Hz. Das bedeutet, dass sich ihre Schwingungen nach einem Vielfachen der Periode des Grundtons wiederholen. Der Grundton (440 Hz) hat eine einfache Sinuswelle. Die 1. Harmonische (880 Hz) eine Sinuswelle mit der doppelten Frequenz und die 2. Harmonische (1320 Hz) eine Sinuswelle mit der dreifachen Frequenz. Harmonische sind das Vielfache der Grundfrequenz. Das bedeutet, dass sie bei ganzzahligen Vielfachen der Grundfrequenz liegen, was man hier deutlich erkennen kann.

Doch auch beim Hören gibt es Unterschiede. Wenn man den Kammerton allein hört, hört man eine gleichmäßige Sinuswelle, die sich klare und konstant anhört. Wenn man die 2.Harmonische allein hört, klingt es eine Oktave höher als der Grundton. Der Klang ist ebenfalls klar, aber aufgrund der höheren Frequenz wirkt er heller und schärfer. Die 3.Harmonische klingt anderthalb Oktaven höher als der Grundton. Es wird noch heller und schriller wahrgenommen jedoch etwas leiser und dumpfer.

**Abspielen der Töne**

Hierfür wird `IPython.display` verwendet mit der Abkürzung ipd.

print("Klang des Kammertons")

ipd.display(ipd.Audio(chambertone, rate=fa))

Hierbei dient der `Print()` Befehl einer Textmeldung als Ausgabe, die anzeigt, um welchen Ton es sich handelt und dient nur der Verschönerung der Ausgabe.

ipd.display(ipd.Audio())Diese Funktion lädt das generierte Audioarray und stellt eine Audioausgabe im Jupyter-Notebook bereit. Der Parameter `rate=fa` gibt die Abtastrate an, die beim Abspielen verwendet wird. Diese Methode wird statt nur dem `ipd.Audio()`verwendet da wir somit keine vorherige Speicherung des Tons benötigen und wir somit viele unnötige Dateien verhindern.

In derselben Art werden auch die anderen Dateien ausgegeben.

**Summierung der harmonischen Signale**

`combined\_signal`: Dies ist das resultierende Signal, das durch die Summierung des Kammertons (Grundfrequenz) und der zweiten und dritten Harmonischen entsteht. Die Addition der Signale erfolgt durch die punktweise Addition der Amplitudenwerte:

combined\_signal = chambertone + second\_harmonic + third\_harmonic

Dies erzeugt ein komplexes Signal, das die Frequenzkomponenten der Grundfrequenz sowie der zweiten und dritten Harmonischen enthält.

**Speichern des Signals in einer Wave-Datei**

wavfile.write('harmonisches\_signal.wav', fa, combined\_signal.astype(np.float32))

Die Funktion speichert das kombinierte Signal in einer WAV-Datei. Die Parameter sind:

* `'harmonisches\_signal.wav'`: Der Dateiname der zu speichernden WAV-Datei
* `fa`: Die Abtastrate, die beim Speichern verwendet wird
* `combined\_signal.astype(np.float32)`: Das kombinierte Signal wird in das `float32`-Format konvertiert, um die erforderliche Präzision für Audiodaten zu gewährleisten.

**Visualisierung(**Plotten**)**

Hierfür wurde das `pyplot` Modul mit der Abkürzung plt verwendet. Erklärungen hierzu finden Sie unter der Überschrift “Plotten “

Der Plot zeigt das Oszillogramm des kombinierten Signals zusammen mit den einzelnen harmonischen Komponenten. Dies ermöglicht eine visuelle Inspektion, wie die Harmonischen zum Gesamtton beitragen.

Ein Bild, das Text, Reihe, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 5 Oszillogramm des kombinierten Signals mit dem Kammerton und den Harmonischen

**Abspielen des Tons**

print("Klang des combinierten Signals")

ipd.Audio('harmonisches\_signal.wav')

hier wird `ipd.Audio()`verwendet da die Datei laut Aufgabenstellung gespeichert werden soll.

**Analyse**

Wenn man die Sinuswellen addiert, ergibt sich eine komplexere Wellenform. Diese Wellenform zeigt Peaks und Täler, die nicht in den einzelnen Sinuswellen zu sehen sind. Die Form wird weniger regelmäßig und enthält zusätzliche Details, die durch die Interferenzen der einzelnen Frequenzen entstehen. Die resultierende Wellenform bleibt periodisch, aber die Periodizität ist komplexer, da sie die gemeinsamen Eigenschaften aller drei Frequenzen integriert. Es entstehen Muster, die eine höhere Frequenz als der Grundton, aber keine einfache Sinuswelle mehr darstellen. Der summierte Klang hat eine reichere und vollere Klangfarbe im Vergleich zum reinen Grundton.

“Prinzip der Überlagerung“

Interferenz: Wenn zwei oder mehr Wellen sich überlagern, addieren sich ihre Amplituden zu einer neuen Wellenform. Dies nennt man Interferenz. Bei konstruktiver Interferenz (wenn die Wellen in Phase sind) verstärken sich die Amplituden. Bei destruktiver Interferenz (wenn die Wellen außer Phase sind) schwächen sich die Amplituden ab. Diese komplexe Wellenform enthält Komponenten aller beteiligten Frequenzen und kann als Summe der einzelnen Sinuswellen beschrieben werden.

## Aufgabe 3

Verändern Sie jetzt die Phase der drei Einzeltöne mit einer Zufallsfunktion und plotten Sie den Klang als Zeitfunktion und geben das Audiosignal aus. Verändert sich der Klang sichtbar und hörbar?

**Zufällige Phasenverschiebung für jede Schwingung generieren**

phase\_shifts = np.random.uniform(0, 2\*np.pi, 3)

Ein Array, das drei zufällige Phasenverschiebungen zwischen 0 und enthält. Diese Phasenverschiebungen werden unabhängig für jede harmonische Schwingung generiert und zufällig gewählt.

* phase\_shifts: Ein Array, das drei zufällige Phasenverschiebungen zwischen 0 und enthält. Diese Phasenverschiebungen werden unabhängig für jede harmonische Schwingung generiert.
* np.random.uniform(0, 2\*np.pi, 3): Erzeugt drei Zufallszahlen, die gleichmäßig zwischen 0 und verteilt sind.

Eine Phasenverschiebung verändert den Zeitpunkt, zu dem die Sinuswelle beginnt. Die Formel zur Generierung der harmonischen Schwingung mit Phasenverschiebung lautet:

wobei die zufällige Phasenverschiebung ist.

chambertone\_random\_phase = harmonic\_generator(1,  t+ phase\_shifts[0])

second\_harmonic\_random\_phase = harmonic\_generator(2,  t + phase\_shifts[1])

third\_harmonic\_random\_phase = harmonic\_generator(3, t + phase\_shifts[2])

Die Funktion `harmonic\_generator` wird aufgerufen, wobei der Zeitvektor `t` um die jeweilige zufällige Phasenverschiebung verschoben wird. Dies führt zu einem Phasenverschobenen Signal. Dadurch wird das Signal wie folgt berechnet:

wobei die zufällige Phasenverschiebung ist.

Dies führt dazu, dass das Signal entlang der Zeitachse verschoben wird, was zu einer veränderten Wellenform und Klangfarbe führt.

**Visualisierung(**Plotten**)**

Hierfür wurde das `pyplot` Modul mit der Abkürzung plt verwendet.

Erklärungen hierzu finden Sie unter der Überschrift “Plotten “

Ein Bild, das Text, Reihe, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 6 Oszillogramm des Kammertons und des Kammertons mit zufälliger Phasenverschiebung

Ein Bild, das Text, Reihe, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 7 Oszillogramm der 2.Harmonischen und der 2.Harmonischen mit zufälliger Phasenverschiebung

Ein Bild, das Text, Reihe, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 8 Oszillogramm der 3.Harmonischen und der 3.Harmonischen mit zufälliger Phasenverschiebung

**Kombiniertes Signal mit zufälliger Phasenverschiebung erstellen**

combined\_signal\_random\_phase = chambertone\_random\_phase + second\_harmonic\_random\_phase + third\_harmonic\_random\_phase

Dies ist das resultierende Signal, das durch die Summierung des Kammertons mit zufälliger Phasenverschiebung und der zweiten und dritten Harmonischen ebenfalls mit zufälliger Phasenverschiebung entsteht.

Alle weiteren Schritte erfolgen gleich der vorherigen Summierung und Speicherung der Datei.

**Visualisierung(**Plotten**)**

Hierfür wurde das `pyplot` Modul mit der Abkürzung plt verwendet. Erklärungen hierzu finden Sie unter der Überschrift “Plotten “

Ein Bild, das Text, Reihe, Schrift, Diagramm enthält.

Automatisch generierte Beschreibung

Abbildung 9 Oszillogramm des kombinierten Signals und des kombinierten Signals mit zufälliger Phasenverschiebung

**Analyse**

Wenn die Phasen der drei Sinustöne zufällig verändert werden, beeinflusst das sowohl die sichtbare Wellenform als auch den hörbaren Klang.

Die Wellenform verändert sich im Zeitbereich, weil die Phasenverschiebungen bewirken, dass sich die Wellen zu unterschiedlichen Zeiten verstärken und abschwächen. Die summierte Wellenform ist weiterhin komplex, aber ihre Form sieht anders aus als bei der Überlagerung ohne Phasenverschiebungen. Durch die zufälligen Phasenverschiebungen können die konstruktiven und destruktiven Interferenzen an verschiedenen Stellen auftreten, was zu einer anderen Verteilung von Peaks und Tälern führt.

Der Grundton bleibt gleich, da die Frequenzen dieselben bleiben. Die Klangfarbe kann sich jedoch ändern, da die harmonischen Komponenten unterschiedlich kombiniert werden. Der Klang wird weniger „rein“ und möglicherweise weniger harmonisch klingen. Das kann dazu führen, dass der Klang rauer oder weniger klar wahrgenommen wird, insbesondere wenn die Phasenverschiebungen zu destruktiver Interferenz führt.

## Aufgabe 4

Mit welchem Schallsignal können sie den Kammerton komplett auslöschen. Generieren Sie in ihrem Programm das resultierende Signal und plotten und spielen Sie dieses ab.

**Generierung des Signals zur Auslöschung des Kammertons**

phase\_shift = np.pi  # Phasenverschiebung um pi

canceling\_signal = a \* np.sin(2 \* np.pi \* f0 \* t + phase\_shift)

* phase\_shift: Eine Phasenverschiebung um π (180 Grad). Dies führt zu einem Signal, das genau entgegengesetzt zur ursprünglichen Welle ist.
* canceling\_signal: Das erzeugte Signal zur Auslöschung des Kammertons. Es hat die gleiche Frequenz und Amplitude wie der Kammerton, jedoch eine Phasenverschiebung von π.

Eine Phasenverschiebung um π (180 Grad) bedeutet, dass das Signal um eine halbe Periode verschoben wird. Dadurch wird jede positive Amplitude in eine negative umgewandelt und umgekehrt.

Das ursprüngliche Signal und das Auslöschungssignal werden wie folgt kombiniert:

Da gilt ergibt sich:

Die destruktive Interferenz tritt auf, wenn zwei Wellen gleicher Frequenz und Amplitude, aber entgegengesetzter Phase sich überlagern. Die resultierende Welle hat eine Amplitude von null, was zur Auslöschung des Signals führt.

**Visualisierung(**Plotten**)**

Hierfür wurde das `pyplot` Modul mit der Abkürzung plt verwendet.

Erklärungen hierzu finden Sie unter der Überschrift “Plotten “

Ein Bild, das Text, Reihe, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 10 Oszillogramm des Signals zur Auslöschung und des Kammertons

Ein Bild, das Text, Reihe, Diagramm, Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 11 Oszillogramm des Auslöschten Kammertons

## Aufgabe 5

*Erstellen Sie ein Programm zur Modellierung einer periodische Rechteckfunktion (f =1Hz, Breite=0.5s, f =8kHz) durch Überlagerung aus seinen ersten 9 harmonischen Komponenten. Plotten Sie die so erzeugte Rechteckfunktion und zusätzlich die 1. Harmonische und die 9. Harmonische. Erklären und beschreiben Sie den Zusammenhang zwischen den 3 Funktionen.*

**Parameterdefinition**

* f0 (Grundfrequenz): Die Grundfrequenz der Rechteckfunktion in Hertz.
* T0 (Periodendauer): Die Dauer einer Periode der Rechteckfunktion.
* breite: Die Breite des Rechteckimpulses in Sekunden.
* fa (Abtastfrequenz): Die Abtastrate in Hertz.
* t: Zeitvektor über zwei Perioden der Rechteckfunktion, mit einer Abtastrate von `fa`.

**Rechteckfunktion erzeugen**

rect = np.zeros\_like(t)

rect[(t % T0) < breite] = 1

* rect: Ein Array zur Speicherung der Rechteckfunktion. Das Array ist zu Beginn mit Nullen gefüllt.
* rect[(t % T0) < breite] = 1: Setzt die Werte des Arrays auf 1, wenn die Bedingung `(t % T0) < breite` erfüllt ist. Dies erzeugt die Rechteckfunktion.

**Fourier-Koeffizienten und Überlagerung der harmonischen Komponenten**

def fourier\_rechteck(t, f0, N):

result = 0.5  # DC-Komponente

    for k in range(1, N+1):

        if k % 2 != 0:  # Nur ungerade harmonische Komponenten

            result += (2 / (k \* np.pi)) \* np.sin(2 \* np.pi \* k \* f0 \* t)

    return result

* fourier\_rechteck(t, f0, N): Diese Funktion berechnet die Fourier-Reihenentwicklung der Rechteckfunktion.
* result = 0.5: Initialisiert das Ergebnis mit der DC-Komponente.
* for-Schleife: Addiert die ungeraden harmonischen Komponenten zur Rechteckfunktion.
* (2 / (k \* np.pi)) \\* np.sin(2 \\* np.pi \\* k \\* f0 \\* t): Berechnet die k-te ungerade harmonische Komponente.

Die Funktion startet mit der DC-Komponente (`result = 0.5`). Die Schleife durchläuft alle ungeraden harmonischen Komponenten (1, 3, 5, ..., N) und addiert diese zur Approximation der Rechteckfunktion durch eine endliche Summe von Sinuswellen, die harmonischen Komponenten. Jede dieser Komponenten trägt zur Form der Rechteckfunktion bei. Die Fourier-Reihenentwicklung ist eine Methode, um periodische Signale wie die Rechteckfunktion in ihre Grundfrequenz und Obertöne zu zerlegen.

Eine Rechteckfunktion kann als unendliche Summe ihrer ungeraden harmonischen Sinuswellen dargestellt werden. Die Fourier-Reihe der Rechteckfunktion lautet:

rect\_approx = fourier\_rechteck(t, f0, N)

` rect\_approx`: ist die generierte Rechtecksfunktion basierend auf die `fourier\_rechteck()`Funktion.

**1. und 9. harmonische Komponente**

def harmonische\_komponente(t, f0, k):

    return (2 / (k \* np.pi)) \* np.sin(2 \* np.pi \* k \* f0 \* t)

Die erste und neunte Harmonische werden erzeug durch:

mit dem Harmonischen-Index `k=1 bzw. 9` und dem Zeitvektor `t` aufgerufen, um die Grundfrequenz zu erzeugen.

**Visualisierung (**Plotten**)**

Hierfür wurde das `pyplot` Modul mit der Abkürzung plt verwendet.

Erklärungen hierzu finden Sie unter der Überschrift “Plotten “

**Ein Bild, das Text, Diagramm, Reihe, Schrift enthält.

Automatisch generierte Beschreibung**

Abbildung 12 Rechteckfunktion und Fourier Approximation

**Analyse**

Eine Rechteckfunktion kann durch die Summe ihrer Fourier-Komponenten (Sinus- und Kosinuswellen) dargestellt werden. Da die Rechteckfunktion eine ungerade Funktion ist (Symmetrie um den Ursprung), enthält sie nur Sinuskomponenten und keine Kosinuskomponenten. Die resultierende Wellenform wird durch die Summe der ersten 9 ungeradzahligen Harmonischen erstellt. Die Amplituden der harmonischen Komponenten sind invers proportional zu ihrer Frequenz. Die n-te Harmonische hat eine Amplitude von . Dies bedeutet, dass höhere Harmonische niedrigere Amplituden haben, was zur Form und Charakteristik der Rechteckwelle beiträgt. Die approximierte Rechteckwelle zeigt deutliche Anzeichen der Rechteckform, obwohl sie nicht perfekt ist. Je mehr harmonische Komponenten hinzugefügt werden, desto genauer wird die Annäherung an eine ideale Rechteckwelle. In diesem Fall mit den ersten 9 harmonischen Komponenten sind die Übergänge zwischen den Hoch- und Tiefpunkten deutlich erkennbar, aber noch nicht ganz so scharf wie in einer perfekten Rechteckwelle.

Die approximierte Rechteckwelle enthält die Grundfrequenz und die ungeradzahligen Harmonischen, was zu einem reicheren und komplexeren Klang führt als ein reiner Sinuston. Der Klang einer Rechteckwelle ist deutlich heller und schärfer als der eines Sinustons, da die zusätzlichen Harmonischen eine komplexere Klangfarbe erzeugen.

## Aufgabe 6

*Wie klingt eine periodische Rechteckfunktion mit 𝑓=440Hz? Vergleichen Sie den Klang mit einem reinen sinus-Ton 𝑓=440Hz?*

**Parameterdefinition**

* f0 (Grundfrequenz): Die Frequenz des erzeugten Tons in Hertz. Ein Wert von 440 Hz entspricht dem Kammerton A.
* duration (Dauer): Die Dauer des Tons in Sekunden.
* sampling\_rate (Abtastrate): Die Anzahl der Abtastwerte pro Sekunde. Eine Abtastrate von 44100 Hz ist Standard für Audio-CDs und sorgt für eine hohe Klangqualität.

**Zeitvektor**

t = np.linspace(0, duration, int(sampling\_rate \* duration), endpoint=False)

t: Ein Vektor, der gleichmäßig verteilte Zeitpunkte von 0 bis zur angegebenen Dauer enthält, basierend auf der Abtastrate. `endpoint=False` bedeutet, dass der Endwert nicht eingeschlossen ist. erstellt, um sicherzustellen, dass die Abtastpunkte gleichmäßig über die gewünschte Dauer verteilt sind.

**Sinusfunktion**

sin\_wave =  np.sin(2 \* np.pi \* f0 \* t)

sin\_wave: Das erzeugte Sinussignal mit einer Frequenz von 440 Hz. Die Sinusfunktion wird mit der Formel berechnet.

**Rechteckfunktion**

square\_wave = np.sign(np.sin(2 \* np.pi \* f0 \* t))

square\_wave: Das erzeugte Rechtecksignal mit einer Frequenz von 440 Hz. Die Rechteckfunktion wird durch Anwenden der Signum-Funktion auf die Sinusfunktion berechnet, was die Werte auf -1 oder 1 beschränkt.

**Visualisierung (**Plotten**)**

Hierfür wurde das `pyplot` Modul mit der Abkürzung plt verwendet.

Erklärungen hierzu finden Sie unter der Überschrift “Plotten “

Ein Bild, das Reihe, Diagramm, Text, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 13 Periodische Rechteck und Sinus Funktion mit f\_0 = 440$ Hz

**Analyse**

Ein reiner Sinuston klingt glatt, gleichmäßig und klar. Es ist der einfachste und reinste Ton, den man erzeugen kann, da er nur eine einzelne Frequenz enthält. Eine Rechteckfunktion klingt viel schärfer und brillanter als ein Sinuston. Dies liegt daran, dass sie eine Reihe von Obertönen enthält.