### Dokumentation des Codes

#### Importierte Bibliotheken

1. \*\*NumPy\*\*:

import numpy as np

NumPy ist eine wesentliche 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.

2. \*\*SciPy\*\*:

import scipy.io.wavfile as wavfile

SciPy ist eine Open-Source-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 gängiges Format für Audiodateien, das Roh-Audiodaten speichert.

3. \*\*Matplotlib\*\*:

import matplotlib.pyplot as plt

Matplotlib ist eine Plotting-Bibliothek für die Programmiersprache Python und seine numerische Mathematikerweiterung NumPy. `pyplot` ist ein Modul in Matplotlib, 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.

4. \*\*IPython.display\*\*:

import IPython.display as ipd

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.

#### Erklärung der Bibliotheken und ihrer Anwendung in der Sprachverarbeitung

1. \*\*NumPy in der Sprachverarbeitung\*\*:

NumPy wird verwendet, um Audiodaten als Arrays zu laden und zu manipulieren. Dies ist wichtig, da Sprachsignale oft als Zeitreihen von Amplitudenwerten dargestellt werden, die numerische Operationen erfordern, wie z.B. das Berechnen von Fourier-Transformationen, Filtern oder anderen Signalverarbeitungsoperationen.

2. \*\*SciPy in der Sprachverarbeitung\*\*:

Mit `scipy.io.wavfile` können wir Audiodaten aus WAV-Dateien laden und speichern. Dies ist nützlich, um Sprachaufnahmen zu verarbeiten und zu analysieren. Die Funktion `wavfile.read` lädt eine WAV-Datei und gibt die Abtastrate und die Audiodaten zurück.

3. \*\*Matplotlib in der Sprachverarbeitung\*\*:

Matplotlib wird verwendet, um die Audiodaten zu visualisieren. Dies kann durch das Plotten des Audiosignals im Zeitbereich erfolgen, was einen Überblick über die Amplitudenänderungen im Laufe der Zeit gibt.

4. \*\*IPython.display in der Sprachverarbeitung\*\*:

Mit `IPython.display` können wir Audiodaten im Jupyter-Notebook abspielen. Dies ist nützlich, um das Audiosignal direkt nach der Verarbeitung zu hören.

Die Kombination dieser Werkzeuge bietet eine solide Grundlage für die Durchführung fortgeschrittener Sprachverarbeitungsaufgaben.

### Dokumentation des Codes

Laden, Visualisieren und Abspielen einer Audiodatei. Dies ermöglicht, ein Verständnis für die Struktur und Eigenschaften des Sprachsignals zu gewinnen.

1. \*\*Laden der Audiodatei\*\*:

WAV-Dateien enthalten Roh-Audiodaten, die oft in PCM (Pulse Code Modulation) kodiert sind. Die Abtastrate gibt an, wie oft das analoge Signal pro Sekunde abgetastet wurde.

2. \*\*Zeitvektor erstellen\*\*:

- \*\*t\*\*: Dies ist der Zeitvektor, 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\*\*: Skalierung des Arrays durch die Abtastrate, um die tatsächlichen Zeitwerte in Sekunden zu erhalten.

3. \*\*Plotten der Audiodatei\*\*:

- \*\*plt.figure(figsize=(10,4))\*\*: Erstellt eine neue Figur mit den angegebenen Abmessungen.

- \*\*plt.plot(t, signal)\*\*: Plottet das Audiosignal als Funktion der Zeit.

- \*\*plt.xlabel("Zeit (s)")\*\*: Beschriftung der x-Achse mit "Zeit (s)".

- \*\*plt.ylabel("Amplitude")\*\*: Beschriftung der y-Achse mit "Amplitude".

- \*\*plt.title("Dies ist eine Suchmaschine")\*\*: Titel des Plots.

- \*\*plt.grid(True)\*\*: Aktiviert das Gitter für bessere Lesbarkeit des Plots.

- \*\*plt.show()\*\*: Zeigt den Plot an.

Der Plot zeigt das Audiosignal im Zeitbereich, was einen Überblick über die Amplitudenänderungen im Laufe der Zeit bietet.

4. \*\*Abspielen der Audiodatei\*\*:

- \*\*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.

#### Erklärung und Formeln

1. \*\*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:

f\_s größer = 2f\_max

wobei \( f\_s \) die Abtastrate und \( f\_{max} \) die höchste Frequenz im Signal ist.

2. \*\*Zeitvektor\*\*:

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

t = n / f\_s

wobei \( n \) die Abtastwerte und \( f\_s \) die Abtastrate ist.

### Dokumentation des Codes

#### Einleitung

In diesem Abschnitt werden Parameter definiert und ein Zeitvektor erstellt, um harmonische Schwingungen zu generieren. Diese harmonischen Schwingungen werden für die Erzeugung von Tönen verwendet, die oft in der digitalen Sprachverarbeitung und Musikproduktion Anwendung finden.

#### Code-Abschnitt

1. \*\*Parameterdefinition\*\*:

- \*\*a (Amplitude)\*\*: Dies ist die maximale Auslenkung der Schwingung. 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. 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.

2. \*\*Zeitvektor\*\*:

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

3. \*\*Generierung der Harmonischen\*\*:

- \*\*harmonic\_generator(k, t)\*\*: Diese Funktion generiert die k-te harmonische Schwingung basierend auf der Grundfrequenz `f0`.

- \*\*Parameter\*\*:

- \*\*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.

Die Formel zur Generierung der harmonischen Schwingung lautet:

y(t) = a\* sin(2 \*PI\*k\*f\_0\*t)

wobei:

- \( a \) die Amplitude ist,

- \( k \) der Harmonische-Index ist,

- \( f\_0 \) die Grundfrequenz ist,

- \( t \) die Zeit ist.

#### Erklärung und Formeln

1. \*\*Amplitude\*\*:

Die Amplitude ist die maximale Auslenkung eines Signals. In diesem Fall ist sie auf 1.0 gesetzt.

2. \*\*Grundfrequenz (f0)\*\*:

Die Grundfrequenz ist die Frequenz des Grundtons. Sie bestimmt die Tonhöhe des erzeugten Signals. Eine Grundfrequenz von 440 Hz ist die Frequenz des A-Tons, der als Kammerton dient.

3. \*\*Abtastfrequenz (fa)\*\*:

Die Abtastfrequenz muss gemäß dem Nyquist-Shannon-Abtasttheorem mindestens doppelt so hoch wie die höchste Frequenzkomponente des Signals sein, um Aliasing zu vermeiden. Hier wurde sie sehr hoch gewählt (20 mal die Grundfrequenz), um eine genaue Darstellung zu ermöglichen.

4. \*\*Zeitvektor (t)\*\*:

Der Zeitvektor wird mit `np.linspace` erstellt, um sicherzustellen, dass die Abtastpunkte gleichmäßig über die gewünschte Dauer verteilt sind.

5. \*\*Harmonische Schwingung\*\*:

Eine harmonische Schwingung wird mit der Sinusfunktion generiert. Der Faktor \(2 \pi k f\_0 t\) stellt sicher, dass die Frequenz der Schwingung ein ganzzahliges Vielfaches der Grundfrequenz ist, was die Definition von Harmonischen erfüllt.

### Dokumentation des Codes

#### Einleitung

In diesem Abschnitt wird der Kammerton (A4) generiert, visualisiert und abgespielt. Der Kammerton ist ein wichtiger Referenzton in der Musik und hat eine Grundfrequenz von 440 Hz.

#### Code-Abschnitt

1. \*\*Generierung des Kammertons\*\*:

- \*\*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` aufgerufen, um die Grundfrequenz zu erzeugen.

2. \*\*Plot des Kammertons (nur die Grundfrequenz)\*\*:

- \*\*plt.figure(figsize=(12, 4))\*\*: Erstellt eine neue Figur mit den angegebenen Abmessungen.

- \*\*plt.plot(t, chambertone)\*\*: Plottet das Audiosignal des Kammertons als Funktion der Zeit.

- \*\*plt.xlim(0, 0.01)\*\*: Begrenzt die x-Achse auf die ersten 10 Millisekunden des Signals, um eine detaillierte Ansicht der Schwingungen zu bieten.

- \*\*plt.title('Oszillogramm des Kammertons (Grundfrequenz)')\*\*: Setzt den Titel des Plots.

- \*\*plt.xlabel('Zeit (s)')\*\*: Beschriftet die x-Achse mit "Zeit (s)".

- \*\*plt.ylabel('Amplitude')\*\*: Beschriftet die y-Achse mit "Amplitude".

- \*\*plt.grid(True)\*\*: Aktiviert das Gitter für bessere Lesbarkeit des Plots.

- \*\*plt.show()\*\*: Zeigt den Plot an.

Der Plot zeigt das Oszillogramm des Kammertons im Zeitbereich. Dies ermöglicht eine visuelle Inspektion der Schwingungen des erzeugten Tons.

3. \*\*Abspielen des Kammertons\*\*:

- \*\*print("Klang des Kammertons")\*\*: Gibt eine Textmeldung aus, die anzeigt, dass der Kammerton abgespielt wird.

- \*\*ipd.display(ipd.Audio(chambertone, rate=fa))\*\*: 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.

#### Erklärung und Formeln

1. \*\*Generierung des Kammertons\*\*:

Der Kammerton wird mit der Funktion `harmonic\_generator` erzeugt. Diese Funktion berechnet die harmonische Schwingung mit der Formel:

y(t) = a\* sin(2 \*PI\*k\*f\_0\*t)

wobei \( f\_0 \) die Grundfrequenz (440 Hz) und \( a \) die Amplitude (1.0) ist.

2. \*\*Visualisierung des Kammertons\*\*:

Das Oszillogramm zeigt die zeitliche Darstellung des Kammertons. Durch das Begrenzen der x-Achse auf die ersten 10 Millisekunden wird die Periodizität und die Schwingungsform deutlich sichtbar.

3. \*\*Abspielen des Kammertons\*\*:

Das Abspielen des erzeugten Tons ermöglicht eine auditive Überprüfung des Signals. Die Funktion `ipd.Audio` verwendet die generierten Amplitudenwerte und die Abtastrate, um das Signal korrekt wiederzugeben.

### Dokumentation des Codes

#### Einleitung

In diesem Abschnitt wird ein kombiniertes Audiosignal aus dem Kammerton und seinen ersten beiden Harmonischen generiert, visualisiert und abgespielt. Dieses kombinierte Signal wird anschließend in einer WAV-Datei gespeichert. Dies ist eine typische Vorgehensweise in der digitalen Signalverarbeitung, um komplexe Töne zu erzeugen und zu analysieren.

#### Code-Abschnitt

1. \*\*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 punktweise.

2. \*\*Speichern des Signals in einer Wave-Datei\*\*:

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

3. \*\*Plot des Oszillogramms des kombinierten Signals\*\*:

- \*\*plt.figure(figsize=(12, 4))\*\*: Erstellt eine neue Figur mit den angegebenen Abmessungen.

- \*\*plt.plot(t, combined\_signal, label="combined")\*\*: Plottet das kombinierte Audiosignal als Funktion der Zeit.

- \*\*plt.plot(t, third\_harmonic, label="3rd Harmonic")\*\*: Plottet die dritte Harmonische als Funktion der Zeit.

- \*\*plt.plot(t, second\_harmonic, label="2nd Harmonic")\*\*: Plottet die zweite Harmonische als Funktion der Zeit.

- \*\*plt.plot(t, chambertone, label="1st Harmonic")\*\*: Plottet den Kammerton (Grundfrequenz) als Funktion der Zeit.

- \*\*plt.xlim(0, 0.01)\*\*: Begrenzt die x-Achse auf die ersten 10 Millisekunden des Signals, um eine detaillierte Ansicht der Schwingungen zu bieten.

- \*\*plt.title('Oszillogramm des kombinierten Signals mit dem Kammerton und den Harmonischen')\*\*: Setzt den Titel des Plots.

- \*\*plt.xlabel('Zeit (s)')\*\*: Beschriftet die x-Achse mit "Zeit (s)".

- \*\*plt.ylabel('Amplitude')\*\*: Beschriftet die y-Achse mit "Amplitude".

- \*\*plt.legend()\*\*: Zeigt eine Legende an, die die verschiedenen Signalkomponenten beschreibt.

- \*\*plt.grid(True)\*\*: Aktiviert das Gitter für bessere Lesbarkeit des Plots.

- \*\*plt.show()\*\*: Zeigt den Plot an.

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.

4. \*\*Audio-Datei einlesen und abspielen\*\*:

- \*\*print("Klang des kombinierten Signals")\*\*: Gibt eine Textmeldung aus, die anzeigt, dass das kombinierte Signal abgespielt wird.

- \*\*ipd.Audio('harmonisches\_signal.wav')\*\*: Diese Funktion lädt die gespeicherte WAV-Datei und stellt eine Audioausgabe im Jupyter-Notebook bereit, die durch Klicken auf den Abspielknopf abgespielt werden kann.

#### Erklärung und Formeln

1. \*\*Summierung der harmonischen Signale\*\*:

Die Addition der harmonischen Signale erfolgt durch die punktweise Addition der Amplitudenwerte:

\[

\text{combined\\_signal}(t) = \text{chambertone}(t) + \text{second\\_harmonic}(t) + \text{third\\_harmonic}(t)

\]

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

2. \*\*Speichern des Signals in einer Wave-Datei\*\*:

Das Signal wird in einer WAV-Datei gespeichert, um es später wiedergeben zu können. Die Konvertierung zu `float32` stellt sicher, dass die Audiodaten in einem geeigneten Format gespeichert werden.

3. \*\*Visualisierung des kombinierten Signals\*\*:

Das Oszillogramm zeigt die zeitliche Darstellung des kombinierten Signals sowie der einzelnen harmonischen Komponenten. Dies hilft, die Struktur und das Zusammenspiel der Harmonischen zu verstehen.

4. \*\*Abspielen der Audio-Datei\*\*:

Das Abspielen der gespeicherten WAV-Datei ermöglicht eine auditive Überprüfung des kombinierten Signals. Dies ist besonders nützlich, um die Wirkung der hinzugefügten Harmonischen auf den Klang zu hören.

#### Zusammenfassung

In diesem Abschnitt des Codes wird ein kombiniertes Audiosignal aus dem Kammerton und seinen ersten beiden Harmonischen generiert, visualisiert und abgespielt. Das kombinierte Signal wird in einer WAV-Datei gespeichert, um es später wiedergeben zu können. Die Visualisierung zeigt die Zusammensetzung des Signals, während das Abspielen eine auditive Überprüfung ermöglicht.

### Dokumentation des Codes

#### Einleitung

In diesem Abschnitt werden zufällige Phasenverschiebungen für jede harmonische Schwingung generiert und auf die harmonischen Signale angewendet. Dies führt zu einer Veränderung der Wellenform und Klangfarbe des kombinierten Signals.

#### Code-Abschnitt

1. \*\*Zufällige Phasenverschiebung für jede Schwingung generieren\*\*:

- \*\*phase\_shifts\*\*: Ein Array, das drei zufällige Phasenverschiebungen zwischen 0 und \(2\pi\) 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 \(2\pi\) verteilt sind.

2. \*\*Signal mit zufälliger Phasenverschiebung erzeugen\*\*:

- \*\*chambertone\_random\_phase\*\*: Das Kammerton-Signal (Grundfrequenz) mit einer zufälligen Phasenverschiebung.

- \*\*second\_harmonic\_random\_phase\*\*: Das Signal der zweiten Harmonischen mit einer zufälligen Phasenverschiebung.

- \*\*third\_harmonic\_random\_phase\*\*: Das Signal der dritten Harmonischen mit einer zufälligen Phasenverschiebung.

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

#### Erklärung und Formeln

1. \*\*Zufällige Phasenverschiebung\*\*:

Die Phasenverschiebung wird zufällig gewählt und liegt im Bereich von 0 bis \(2\pi\). Eine Phasenverschiebung verändert den Zeitpunkt, zu dem die Sinuswelle beginnt. Die Formel zur Generierung der harmonischen Schwingung mit Phasenverschiebung lautet:

\[

y(t) = a \* sin(2\* PI \* k \* f\_0(t + zufällige Phasenverschiebung ))

\]

2. \*\*Signal mit zufälliger Phasenverschiebung\*\*:

Durch das Hinzufügen der zufälligen Phasenverschiebung zum Zeitvektor wird das Signal wie folgt berechnet:

\[

y(t + zufällige Phasenverschiebung ) = a \* sin(2\* PI \* k \* f\_0(t + zufällige Phasenverschiebung ))

\]

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

#### Zusammenfassung

In diesem Abschnitt des Codes werden zufällige Phasenverschiebungen für jede harmonische Schwingung generiert und auf die harmonischen Signale angewendet. Dies verändert die zeitliche Lage der Schwingungen und führt zu einer Modifikation der resultierenden Wellenform und des Klangs.

### Dokumentation des Codes

#### Einleitung

In diesem Abschnitt wird ein kombiniertes Audiosignal aus dem Kammerton und seinen ersten beiden Harmonischen mit zufälliger Phasenverschiebung generiert, visualisiert und abgespielt. Das kombinierte Signal wird anschließend in einer WAV-Datei gespeichert. Der Vergleich zwischen dem ursprünglichen und dem phasenverschobenen Signal wird ebenfalls visualisiert und auditiert.

#### Code-Abschnitt

1. \*\*Kombiniertes Signal mit zufälliger Phasenverschiebung erstellen\*\*:

- \*\*combined\_signal\_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. Die Addition der Signale erfolgt punktweise.

2. \*\*Speichern des Signals mit zufälliger Phasenverschiebung in einer Wave-Datei\*\*:

- \*\*wavfile.write('harmonisches\_signal\_random\_phase.wav', fa, combined\_signal\_random\_phase.astype(np.float32))\*\*: Diese Funktion speichert das kombinierte Signal mit zufälliger Phasenverschiebung in einer WAV-Datei. Die Parameter sind:

- `'harmonisches\_signal\_random\_phase.wav'`: Der Dateiname der zu speichernden WAV-Datei.

- `fa`: Die Abtastrate, die beim Speichern verwendet wird.

- `combined\_signal\_random\_phase.astype(np.float32)`: Das kombinierte Signal wird in das `float32`-Format konvertiert, um die erforderliche Präzision für Audiodaten zu gewährleisten.

3. \*\*Plot des Oszillogramms des kombinierten Signals mit zufälliger Phasenverschiebung\*\*:

- \*\*plt.figure(figsize=(12, 4))\*\*: Erstellt eine neue Figur mit den angegebenen Abmessungen.

- \*\*plt.plot(t, combined\_signal\_random\_phase, label="combined signal random phase")\*\*: Plottet das kombinierte Signal mit zufälliger Phasenverschiebung als Funktion der Zeit.

- \*\*plt.plot(t, combined\_signal, label="combined signal")\*\*: Plottet das ursprüngliche kombinierte Signal als Funktion der Zeit.

- \*\*plt.xlim(0, 0.01)\*\*: Begrenzt die x-Achse auf die ersten 10 Millisekunden des Signals, um eine detaillierte Ansicht der Schwingungen zu bieten.

- \*\*plt.title('Oszillogramm des kombinierten Signals und des kombinierten Signals mit zufälliger Phasenverschiebung')\*\*: Setzt den Titel des Plots.

- \*\*plt.xlabel('Zeit (s)')\*\*: Beschriftet die x-Achse mit "Zeit (s)".

- \*\*plt.ylabel('Amplitude')\*\*: Beschriftet die y-Achse mit "Amplitude".

- \*\*plt.grid(True)\*\*: Aktiviert das Gitter für bessere Lesbarkeit des Plots.

- \*\*plt.legend()\*\*: Zeigt eine Legende an, die die verschiedenen Signalkomponenten beschreibt.

- \*\*plt.show()\*\*: Zeigt den Plot an.

Der Plot zeigt das Oszillogramm des kombinierten Signals und des kombinierten Signals mit zufälliger Phasenverschiebung im Zeitbereich. Dies ermöglicht eine visuelle Inspektion der Auswirkungen der Phasenverschiebung.

4. \*\*Abspielen des kombinierten Signals mit zufälliger Phasenverschiebung\*\*:

- \*\*print("Klang des kombinierten Signals mit zufälliger Phasenverschiebung")\*\*: Gibt eine Textmeldung aus, die anzeigt, dass das kombinierte Signal mit zufälliger Phasenverschiebung abgespielt wird.

- \*\*ipd.display(ipd.Audio(combined\_signal\_random\_phase, rate=fa))\*\*: Diese Funktion lädt das generierte Audioarray mit zufälliger Phasenverschiebung und stellt eine Audioausgabe im Jupyter-Notebook bereit. Der Parameter `rate=fa` gibt die Abtastrate an, die beim Abspielen verwendet wird.

5. \*\*Abspielen des kombinierten Signals\*\*:

- \*\*print("Klang des kombinierten Signals")\*\*: Gibt eine Textmeldung aus, die anzeigt, dass das ursprüngliche kombinierte Signal abgespielt wird.

- \*\*ipd.display(ipd.Audio(combined\_signal, rate=fa))\*\*: Diese Funktion lädt das originale kombinierte Audioarray und stellt eine Audioausgabe im Jupyter-Notebook bereit. Der Parameter `rate=fa` gibt die Abtastrate an, die beim Abspielen verwendet wird.

#### Erklärung und Formeln

1. \*\*Summierung der harmonischen Signale mit zufälliger Phasenverschiebung\*\*:

Die Addition der phasenverschobenen harmonischen Signale erfolgt durch die punktweise Addition der Amplitudenwerte:

\[

\text{combined\\_signal\\_random\\_phase}(t) = \text{chambertone\\_random\\_phase}(t) + \text{second\\_harmonic\\_random\\_phase}(t) + \text{third\\_harmonic\\_random\\_phase}(t)

\]

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

2. \*\*Speichern des Signals in einer Wave-Datei\*\*:

Das Signal wird in einer WAV-Datei gespeichert, um es später wiedergeben zu können. Die Konvertierung zu `float32` stellt sicher, dass die Audiodaten in einem geeigneten Format gespeichert werden.

3. \*\*Visualisierung des kombinierten Signals\*\*:

Das Oszillogramm zeigt die zeitliche Darstellung des kombinierten Signals mit und ohne zufällige Phasenverschiebung. Dies hilft, die Struktur und das Zusammenspiel der Harmonischen zu verstehen und die Auswirkungen der Phasenverschiebung zu überprüfen.

4. \*\*Auditive Überprüfung\*\*:

Das Abspielen der Signale ermöglicht eine auditive Überprüfung der Auswirkungen der Phasenverschiebung. Dies ist besonders nützlich, um die subjektiven Unterschiede im Klang zu bewerten.

#### Zusammenfassung

In diesem Abschnitt des Codes wird ein kombiniertes Audiosignal aus dem Kammerton und seinen ersten beiden Harmonischen mit zufälliger Phasenverschiebung generiert, visualisiert und abgespielt. Das kombinierte Signal wird in einer WAV-Datei gespeichert, um es später wiedergeben zu können. Die Visualisierung zeigt die Zusammensetzung des Signals mit und ohne Phasenverschiebung, während das Abspielen eine auditive Überprüfung ermöglicht.

### Dokumentation des Codes

#### Einleitung

In diesem Abschnitt wird ein Signal zur Auslöschung des Kammertons (Grundfrequenz) generiert. Dies erfolgt durch eine Phasenverschiebung um π (180 Grad), was zu destruktiver Interferenz führt. Das resultierende Signal und das Signal zur Auslöschung werden visualisiert und abgespielt, um die Auswirkungen zu demonstrieren.

#### Code-Abschnitt

1. \*\*Generierung des Signals zur Auslöschung des Kammertons\*\*:

- \*\*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.

2. \*\*Plot des Oszillogramms des Signals zur Auslöschung des Kammertons\*\*:

- \*\*plt.plot(t, canceling\_signal, label="Canceling Signal")\*\*: Plottet das Signal zur Auslöschung des Kammertons.

- \*\*plt.plot(t, chambertone, label="Chambertone")\*\*: Plottet den ursprünglichen Kammerton.

- \*\*plt.xlim(0, 0.01)\*\*: Begrenzt die x-Achse auf die ersten 10 Millisekunden des Signals.

- \*\*plt.title('Oszillogramm des Signals zur Auslöschung des Kammertons')\*\*: Setzt den Titel des Plots.

- \*\*plt.xlabel('Zeit (s)')\*\*: Beschriftet die x-Achse mit "Zeit (s)".

- \*\*plt.ylabel('Amplitude')\*\*: Beschriftet die y-Achse mit "Amplitude".

- \*\*plt.grid(True)\*\*: Aktiviert das Gitter.

- \*\*plt.legend()\*\*: Zeigt eine Legende an, um die beiden Signale zu unterscheiden.

- \*\*plt.show()\*\*: Zeigt den Plot an.

Der Plot zeigt das Oszillogramm des ursprünglichen Kammertons und des Signals zur Auslöschung. Die entgegengesetzte Phase des Auslöschungssignals ist deutlich sichtbar.

3. \*\*Kombination des Kammertons und des Auslöschungssignals\*\*:

4. \*\*Speichern des Signals in einer Wave-Datei\*\*:

5. \*\*Plot des Oszillogramms des kombinierten Signals zur Auslöschung des Kammertons\*\*:

- \*\*plt.plot(t, chambertone\_canceling\_signal)\*\*: Plottet das kombinierte Signal aus dem Kammerton und dem Auslöschungssignal.

- \*\*plt.xlim(0, 0.001)\*\*: Begrenzt die x-Achse auf die ersten 1 Millisekunde des Signals, um die Auslöschung deutlich zu zeigen.

6. \*\*Abspielen der Audiosignale\*\*:

Diese Abschnitte geben die ursprünglichen und modifizierten Audiosignale aus, um die Auswirkungen der Phasenverschiebung zu demonstrieren.

#### Erklärung und Formeln

1. \*\*Phasenverschiebung\*\*:

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.

2. \*\*Destruktive Interferenz\*\*:

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.

3. \*\*Mathematische Darstellung\*\*:

Das ursprüngliche Signal und das Auslöschungssignal werden wie folgt kombiniert:

\[

y\_{\text{combined}}(t) = y(t) + y(t + \pi)

\]

Da \( y(t + \pi) = -y(t) \), ergibt sich:

\[

y\_{\text{combined}}(t) = y(t) - y(t) = 0

\]

#### Zusammenfassung

In diesem Abschnitt des Codes wird ein Signal zur Auslöschung des Kammertons durch Phasenverschiebung um π (180 Grad) generiert, visualisiert und abgespielt. Durch die destruktive Interferenz wird der Kammerton effektiv ausgelöscht, was durch die Visualisierung und das Abspielen des Signals demonstriert wird.

### Dokumentation des Codes

#### Einleitung

In diesem Abschnitt wird eine Rechteckfunktion unter Verwendung der Fourier-Reihenentwicklung generiert, visualisiert und abgespielt. Die Rechteckfunktion wird aus ihren ungeraden harmonischen Komponenten zusammengesetzt. Zusätzlich werden die erste und die neunte Harmonische separat visualisiert und abgespielt.

#### Code-Abschnitt

1. \*\*Parameterdefinition\*\*:

- \*\*f0 (Grundfrequenz)\*\*: Die Grundfrequenz der Rechteckfunktion in Hertz.

- \*\*width (Breite)\*\*: Die Breite der Rechteckfunktion in Sekunden.

- \*\*fa (Abtastfrequenz)\*\*: Die Abtastrate in Hertz.

- \*\*num\_harmonics (Anzahl der Harmonischen)\*\*: Die Anzahl der zu verwendenden harmonischen Komponenten. Hier werden die ersten neun ungeraden Harmonischen verwendet.

2. \*\*Zeitvektor\*\*:

- \*\*t\*\*: Ein Vektor, der gleichmäßig verteilte Zeitpunkte von 0 bis zur angegebenen Breite enthält, basierend auf der Abtastrate. `endpoint=False` bedeutet, dass der Endwert nicht eingeschlossen ist.

3. \*\*Fourier-Reihe der Rechteckfunktion mit ungeraden Harmonischen\*\*:

- \*\*rectangular\_wave\*\*: Ein Array zur Speicherung der Rechteckfunktion.

- \*\*for-Schleife\*\*: Diese Schleife addiert die ungeraden harmonischen Komponenten zur Rechteckfunktion. Jede harmonische Komponente hat die Form:

\[

Komponente\_n(t) = 1/n sin(2\* PI \* n \* f\_0 \* t)

\]

wobei \( n \) die n-te ungerade Harmonische ist (1, 3, 5, ..., 9).

4. \*\*Generierung der ersten und neunten Harmonischen\*\*:

5. \*\*Plot der Rechteckfunktion und der Harmonischen\*\*:

- \*\*plt.plot(t, rectangular\_wave, label='Rechteckfunktion')\*\*: Plottet die Rechteckfunktion.

- \*\*plt.plot(t, first\_harmonic, label='1. Harmonische')\*\*: Plottet die erste Harmonische.

- \*\*plt.plot(t, ninth\_harmonic, label='9. Harmonische')\*\*: Plottet die neunte Harmonische.

- \*\*plt.legend()\*\*: Zeigt eine Legende an, um die Signale zu unterscheiden.

- \*\*plt.grid(True)\*\*: Aktiviert das Gitter für bessere Lesbarkeit des Plots.

6. \*\*Abspielen der Rechteckfunktion und der Harmonischen\*\*:

- \*\*ipd.display(ipd.Audio(rectangular\_wave, rate=fa))\*\*: Spielt die Rechteckfunktion ab.

- \*\*ipd.display(ipd.Audio(first\_harmonic, rate=fa))\*\*: Spielt die erste Harmonische ab.

- \*\*ipd.display(ipd.Audio(ninth\_harmonic, rate=fa))\*\*: Spielt die neunte Harmonische ab.

#### Erklärung und Formeln

1. \*\*Fourier-Reihenentwicklung der Rechteckfunktion\*\*:

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

\[

Rechteck(t) = Summenzeichen oben unendlich unten n = 1,3,5,… 1/n sin(2\* PI \* n \* f\_0 \* t )

\]

2. \*\*Harmonische\*\*:

- \*\*Erste Harmonische (Grundfrequenz)\*\*:

\[

1. Harmonische(t) = sin(2 \* pi\* f\_0\* t)

\]

- \*\*Neunte Harmonische\*\*:

\[

9. Harmonische(t) = sin(2 \* pi\* 9 \* f\_0\* t)

\]

3. \*\*Zeitvektor\*\*:

Der Zeitvektor `t` wird mit `np.linspace` erstellt, um sicherzustellen, dass die Abtastpunkte gleichmäßig über die gewünschte Dauer verteilt sind.

#### Zusammenfassung

In diesem Abschnitt des Codes wird eine Rechteckfunktion unter Verwendung der Fourier-Reihenentwicklung generiert, die ersten neun ungeraden harmonischen Komponenten werden summiert. Die Rechteckfunktion und die ersten und neunten Harmonischen werden visualisiert und abgespielt, um die harmonischen Komponenten und ihre Beiträge zur Rechteckfunktion zu demonstrieren.

‚‘‘‘######### neuer recht eck

### Dokumentation des Codes

#### Einleitung

In diesem Abschnitt wird eine Rechteckfunktion erzeugt und mit der Fourier-Reihenentwicklung approximiert. Dabei werden die ersten neun harmonischen Komponenten zur Approximation verwendet. Zusätzlich werden die erste und neunte Harmonische separat visualisiert.

#### Code-Abschnitt

1. \*\*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`.

2. \*\*Rechteckfunktion erzeugen\*\*:

- \*\*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.

3. \*\*Fourier-Koeffizienten und Überlagerung der harmonischen Komponenten\*\*:

- \*\*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.

4. \*\*1. und 9. harmonische Komponente\*\*:

5. \*\*Generierung der Rechteckfunktion durch Überlagerung der ersten 9 harmonischen Komponenten\*\*:

6. \*\*Plotten der Ergebnisse\*\*:

- \*\*plt.plot(t, rect, label='Originale Rechteckfunktion', linestyle='--')\*\*: Plottet die originale Rechteckfunktion.

- \*\*plt.plot(t, rect\_approx, label='Fourier Approximation mit 9 Harmonischen')\*\*: Plottet die Fourier-Approximation der Rechteckfunktion mit den ersten neun harmonischen Komponenten.

- \*\*plt.plot(t, harmonic\_1, label='1. Harmonische')\*\*: Plottet die erste Harmonische.

- \*\*plt.plot(t, harmonic\_9, label='9. Harmonische')\*\*: Plottet die neunte Harmonische.

- \*\*plt.legend()\*\*: Zeigt eine Legende an, um die Signale zu unterscheiden.

- \*\*plt.grid()\*\*: Aktiviert das Gitter für bessere Lesbarkeit des Plots.

#### Erklärung und Formeln

1. \*\*Fourier-Reihenentwicklung der Rechteckfunktion\*\*:

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

\[

Rechteck(t) = 0.5 + sum oben N unten k = 1,3,5,… 2/ k\* PI sin(2\* PI \* k \* f\_0 \* t )

\]

2. \*\*Harmonische\*\*:

- \*\*Erste Harmonische (Grundfrequenz)\*\*:

\[

1. Harmonische(t) = 2/ PI sin(2 \* pi \* f\_0\* t)

\]

- \*\*Neunte Harmonische\*\*:

\[

9. Harmonische(t) = 2/ PI sin(2 \* pi\* 9 \* f\_0\* t)

\]

#### Zusammenfassung

In diesem Abschnitt des Codes wird eine Rechteckfunktion durch die Fourier-Reihenentwicklung mit den ersten neun ungeraden harmonischen Komponenten approximiert. Die Rechteckfunktion, die Fourier-Approximation und die erste sowie die neunte Harmonische werden visualisiert, um die Beiträge der harmonischen Komponenten zur Rechteckfunktion zu demonstrieren.

### Zusammenhang zwischen den Funktionen

In diesem Abschnitt wird der Zusammenhang zwischen den drei Funktionen `fourier\_rechteck`, `harmonische\_komponente` und der Hauptfunktion, die die Rechteckfunktion erzeugt und visualisiert, erläutert. Diese Funktionen arbeiten zusammen, um eine Rechteckfunktion mittels Fourier-Reihenentwicklung zu approximieren und die einzelnen harmonischen Komponenten zu analysieren.

#### 1. Funktion: `fourier\_rechteck`

```python

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

```

##### Erklärung:

- \*\*Zweck\*\*: Diese Funktion berechnet die Fourier-Approximation einer Rechteckfunktion.

- \*\*Parameter\*\*:

- `t`: Zeitvektor.

- `f0`: Grundfrequenz der Rechteckfunktion.

- `N`: Anzahl der zu verwendenden harmonischen Komponenten.

- \*\*Funktionsweise\*\*:

- Die Funktion startet mit der DC-Komponente (`result = 0.5`).

- Eine Schleife durchläuft alle ungeraden harmonischen Komponenten (1, 3, 5, ..., N) und addiert diese zur Approximation der Rechteckfunktion.

- Die Formel zur Berechnung jeder harmonischen Komponente ist \(\frac{2}{k \pi} \sin(2 \pi k f\_0 t)\).

#### 2. Funktion: `harmonische\_komponente`

```python

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

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

```

##### Erklärung:

- \*\*Zweck\*\*: Diese Funktion berechnet eine einzelne harmonische Komponente der Fourier-Reihe.

- \*\*Parameter\*\*:

- `t`: Zeitvektor.

- `f0`: Grundfrequenz.

- `k`: Index der harmonischen Komponente.

- \*\*Funktionsweise\*\*:

- Die Funktion berechnet die k-te harmonische Komponente der Fourier-Reihe mit der Formel \(\frac{2}{k \pi} \sin(2 \pi k f\_0 t)\).

- Dies ermöglicht eine separate Betrachtung und Analyse jeder einzelnen harmonischen Komponente.

#### 3. Hauptfunktion zur Erzeugung und Visualisierung der Rechteckfunktion

Generierung der Rechteckfunktion durch Überlagerung der ersten 9 harmonischen

##### Erklärung:

- \*\*Zweck\*\*: Diese Abschnitte des Codes erzeugen die Rechteckfunktion und visualisieren sowohl die Originalfunktion als auch die Fourier-Approximation und einzelne harmonische Komponenten.

- \*\*Ablauf\*\*:

- \*\*Generierung der Rechteckfunktion\*\*:

- `rect\_approx = fourier\_rechteck(t, f0, 9)`: Erzeugt die Fourier-Approximation der Rechteckfunktion mit den ersten neun harmonischen Komponenten.

- `harmonic\_1 = harmonische\_komponente(t, f0, 1)`: Berechnet die erste Harmonische.

- `harmonic\_9 = harmonische\_komponente(t, f0, 9)`: Berechnet die neunte Harmonische.

- \*\*Visualisierung\*\*:

- Ein Plot wird erstellt, der die originale Rechteckfunktion, die Fourier-Approximation und die ersten und neunten harmonischen Komponenten zeigt.

- Die Legende, Achsenbeschriftungen und das Gitter helfen bei der besseren Lesbarkeit und Interpretation des Plots.

### Zusammenhang und Interaktion

1. \*\*Fourier-Reihe und Rechteckfunktion\*\*:

- Die Funktion `fourier\_rechteck` approximiert die 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.

2. \*\*Harmonische Komponenten\*\*:

- Die Funktion `harmonische\_komponente` erlaubt es, einzelne harmonische Beiträge separat zu berechnen und darzustellen. Dies ist nützlich, um zu verstehen, wie jede Harmonische zur Gesamtform des Signals beiträgt. Beispielsweise hat die erste Harmonische die größte Amplitude und definiert die Grundform des Signals, während höhere Harmonische feinere Details hinzufügen.

3. \*\*Visualisierung und Interpretation\*\*:

- Die Hauptfunktion nutzt beide anderen Funktionen (`fourier\_rechteck` und `harmonische\_komponente`), um eine Rechteckfunktion zu erzeugen und zu analysieren. Durch das Plotten der Rechteckfunktion, der Fourier-Approximation und der einzelnen harmonischen Komponenten wird deutlich, wie die Summe der Harmonischen zur Gesamtform des Rechtecksignals führt.

### Fazit

Die drei Funktionen arbeiten zusammen, um die Rechteckfunktion durch Fourier-Reihenentwicklung zu erzeugen und zu analysieren. `fourier\_rechteck` erzeugt die gesamte approximierte Rechteckfunktion, `harmonische\_komponente` ermöglicht die Berechnung einzelner Harmonischer, und die Hauptfunktion visualisiert die Ergebnisse, um die Beiträge der harmonischen Komponenten zur Rechteckfunktion zu verdeutlichen. Diese Visualisierungen helfen dabei, das Konzept der Fourier-Reihe und ihre Anwendung zur Signalverarbeitung zu verstehen.

### Dokumentation des Codes

#### Einleitung

In diesem Abschnitt wird ein Sinuston und eine Rechteckfunktion mit einer Grundfrequenz von 440 Hz erzeugt und visualisiert. Die beiden Signale werden über eine Dauer von 2 Sekunden bei einer Abtastrate von 44100 Hz erstellt. Diese Schritte umfassen die Generierung der Signale, ihre Visualisierung und das Abspielen der resultierenden Audiodateien.

#### Code-Abschnitt

1. \*\*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.

2. \*\*Zeitvektor\*\*:

- \*\*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.

3. \*\*Sinusfunktion\*\*:

```python

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 \(\sin(2 \pi f\_0 t)\) berechnet.

4. \*\*Rechteckfunktion\*\*:

```python

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 (\(\text{sign}\)) auf die Sinusfunktion berechnet, was die Werte auf -1 oder 1 beschränkt.

5. \*\*Plot der Rechteck- und Sinusfunktion\*\*:

- \*\*plt.plot(t, square\_wave, label='Rechteckfunktion')\*\*: Plottet die Rechteckfunktion als Funktion der Zeit.

- \*\*plt.plot(t, sin\_wave, label='Sinusfunktion')\*\*: Plottet die Sinusfunktion als Funktion der Zeit.

- \*\*plt.xlim(0, 0.01)\*\*: Begrenzt die x-Achse auf die ersten 10 Millisekunden des Signals, um eine detaillierte Ansicht der Schwingungen zu bieten.

- \*\*plt.title('Periodische Rechteck- und Sinusfunktion mit $f\_0 = 440$ Hz')\*\*: Setzt den Titel des Plots.

- \*\*plt.xlabel('Zeit (s)')\*\*: Beschriftet die x-Achse mit "Zeit (s)".

- \*\*plt.ylabel('Amplitude')\*\*: Beschriftet die y-Achse mit "Amplitude".

- \*\*plt.grid(True)\*\*: Aktiviert das Gitter für bessere Lesbarkeit des Plots.

- \*\*plt.legend()\*\*: Zeigt eine Legende an, um die Signale zu unterscheiden.

- \*\*plt.show()\*\*: Zeigt den Plot an.

6. \*\*Abspielen der periodischen Rechteckfunktion\*\*:

7. \*\*Abspielen des reinen Sinustons\*\*:

#### Erklärung und Formeln

1. \*\*Sinusfunktion\*\*:

Die Sinusfunktion für einen Ton mit Frequenz \( f\_0 \) wird berechnet mit:

\[

y(t) = \sin(2 \pi f\_0 t)

\]

Dies erzeugt eine periodische Schwingung, die gleichmäßig zwischen -1 und 1 oszilliert.

2. \*\*Rechteckfunktion\*\*:

Die Rechteckfunktion kann durch Anwenden der Signum-Funktion auf die Sinusfunktion erzeugt werden:

\[

y(t) = \text{sign}(\sin(2 \pi f\_0 t))

\]

Dies ergibt ein Signal, das zwischen -1 und 1 wechselt, wobei der Übergang abrupt stattfindet.

3. \*\*Zeitvektor\*\*:

Der Zeitvektor `t` wird mit `np.linspace` erstellt, um sicherzustellen, dass die Abtastpunkte gleichmäßig über die gewünschte Dauer verteilt sind.

#### Zusammenfassung

In diesem Abschnitt des Codes wird ein Sinuston und eine Rechteckfunktion mit einer Grundfrequenz von 440 Hz erzeugt, visualisiert und abgespielt. Die Visualisierung zeigt die Wellenformen der Signale, und das Abspielen ermöglicht eine auditive Überprüfung der erzeugten Töne.

### Vergleich des Klangs einer Rechteckfunktion und eines reinen Sinustons bei 440 Hz

#### Einleitung

In diesem Abschnitt werden der Klang einer Rechteckfunktion und eines reinen Sinustons bei einer Grundfrequenz von 440 Hz verglichen. Beide Signale haben die gleiche Grundfrequenz, aber ihre klanglichen Eigenschaften unterscheiden sich aufgrund ihrer unterschiedlichen Wellenformen und harmonischen Inhalte.

#### Klang eines reinen Sinustons (440 Hz)

Ein reiner Sinuston besteht aus einer einzigen Frequenzkomponente, die als Grundfrequenz bezeichnet wird. Bei einem Sinuston mit 440 Hz ist die Wellenform eine glatte, kontinuierliche Schwingung. Der Klang eines reinen Sinustons ist als "reiner" oder "weicher" bekannt und enthält keine zusätzlichen Obertöne oder Harmonischen.

```python

# Generierung des reinen Sinustons

f0 = 440 # Grundfrequenz in Hz

duration = 2 # Dauer des Tons in Sekunden

sampling\_rate = 44100 # Abtastrate in Hz

# Zeitvektor

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

# Sinusfunktion

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

# Abspielen des reinen Sinustons

print("Klang des reinen Sinustons:")

ipd.display(ipd.Audio(sin\_wave, rate=sampling\_rate))

```

#### Klang einer Rechteckfunktion (440 Hz)

Eine Rechteckfunktion enthält neben der Grundfrequenz auch eine Reihe von Obertönen, die ungeraden Harmonischen (3. Harmonische, 5. Harmonische, etc.). Diese Obertöne verleihen der Rechteckfunktion einen "rauhen" oder "hellen" Klang. Die abrupten Übergänge der Rechteckwellenform erzeugen zusätzliche Frequenzkomponenten, die im Klang wahrnehmbar sind.

```python

# Generierung der Rechteckfunktion

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

# Abspielen der periodischen Rechteckfunktion

print("Klang der periodischen Rechteckfunktion:")

ipd.display(ipd.Audio(square\_wave, rate=sampling\_rate))

```

#### Vergleich der Klangeigenschaften

1. \*\*Klangfarbe\*\*:

- \*\*Sinuston\*\*: Der Sinuston klingt rein und weich, da er nur die Grundfrequenz (440 Hz) enthält. Er hat keine Obertöne oder zusätzliche Frequenzkomponenten.

- \*\*Rechteckfunktion\*\*: Der Klang der Rechteckfunktion ist rauher und heller im Vergleich zum Sinuston. Dies liegt an den zusätzlichen ungeraden Harmonischen (880 Hz, 1320 Hz, etc.), die im Klang präsent sind.

2. \*\*Harmonische Inhalte\*\*:

- \*\*Sinuston\*\*: Besteht ausschließlich aus der Grundfrequenz von 440 Hz.

- \*\*Rechteckfunktion\*\*: Enthält neben der Grundfrequenz auch die ungeraden Harmonischen, was zu einem komplexeren Klang führt.

3. \*\*Visuelle Unterschiede in der Wellenform\*\*:

- \*\*Sinuston\*\*: Die Wellenform ist eine glatte, periodische Kurve, die kontinuierlich zwischen -1 und 1 oszilliert.

- \*\*Rechteckfunktion\*\*: Die Wellenform wechselt abrupt zwischen -1 und 1, was zu scharfen Kanten und einem blockartigen Erscheinungsbild führt.

#### Fazit

Der reine Sinuston mit 440 Hz hat einen klaren und reinen Klang, während die Rechteckfunktion mit derselben Grundfrequenz aufgrund der zusätzlichen ungeraden Harmonischen einen raueren und komplexeren Klang hat. Diese Unterschiede sind sowohl visuell in der Wellenform als auch auditiv im Klang hörbar.

Beide Signale können im Jupyter-Notebook abgespielt werden, um die Unterschiede im Klang zu hören:

```python

# Abspielen des reinen Sinustons

print("Klang des reinen Sinustons:")

ipd.display(ipd.Audio(sin\_wave, rate=sampling\_rate))

# Abspielen der periodischen Rechteckfunktion

print("Klang der periodischen Rechteckfunktion:")

ipd.display(ipd.Audio(square\_wave, rate=sampling\_rate))

```

Durch das Anhören der beiden Signale kann man die beschriebenen Unterschiede in der Klangfarbe und den harmonischen Inhalten deutlich wahrnehmen.