Labor 1 Dokumentation

Inhalt

[Importierte Bibliotheken 2](#_Toc168772196)

[Aufgabe 1 3](#_Toc168772197)

[Aufgabe 2 5](#_Toc168772198)

[Aufgabe 3 10](#_Toc168772199)

[Aufgabe 4 13](#_Toc168772200)

[Aufgabe 5 14](#_Toc168772201)

[Aufgabe 6 16](#_Toc168772202)

## 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.

Ploten erklären oder son

## 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.

* 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.

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 Sehen. Ergänzen

## 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.

**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

Ergänzen mit Beobachtungen

**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**)**

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.

## 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**)**

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**)**

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**

ergänzen

## 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**)**

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**)**

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

Automatisch generierte Beschreibung**

Abbildung 12 Rechteckfunktion und Fourier Approximation

**Analyse**

Zusammenhang der Funktionen?

## 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: 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**)**

ergänzen

**Analyse**

ergänzen