Fourier-Transformation - Signale & Systeme

Martikelnummer: 7531990

In [1]:
```
import librosa
import os
import numpy as np
from cmath import sqrt
import ipywidgets as widgets
from ipywidgets import interact, interactive, IntSlider, FloatSlider, FloatLogSlider, R
import IPython.display as ipd
import matplotlib.pyplot as plt
```

Initialisierung

In [2]:
```
DIR = os.path.abspath('')
multi400 = os.path.join(DIR, "sounds\\410And420And430.wav")
multi456 = os.path.join(DIR, "sounds\\410AND510AND610.wav")
square100 = os.path.join(DIR, "sounds\\square-100Hz.wav")
lowFreq = os.path.join(DIR, "sounds\\lowFreq.wav")
highFreq = os.path.join(DIR, "sounds\\highFreq.wav")
highAmp = os.path.join(DIR, "sounds\\highAmp.wav")
lowAmp = os.path.join(DIR, "sounds\\lowAmp.wav")
```

In [3]:
```
def loadSound(mainSound, sampleRate=22050):
    global mainArray
    global sr   # sample rate
    mainArray, sr = librosa.load(mainSound, sr=sampleRate)
```

In [4]:
```
def drawSinPlot(zoomFaktor):
    duration = 2
    x = np.linspace(0, duration, len(mainArray))

    plt.figure(figsize=(18,7))
    plt.ylabel("amplitude", fontsize=15)
    plt.xlabel("time", fontsize=15)
    plt.title(name, fontsize=15)
    plt.plot(x[:zoomFaktor], mainArray[:zoomFaktor])
    plt.show()
```

In [5]:
```
def drawIftPlot(zoomFaktor):
    duration = 2
    x = np.linspace(0, duration, len(mainArray))

    plt.figure(figsize=(18,7))
    plt.ylabel("amplitude", fontsize=15)
    plt.xlabel("time", fontsize=15)
    plt.title(name, fontsize=15)
    plt.plot(x[:zoomFaktor], mainArray[:zoomFaktor] , "-b", label="Original Signal")
    plt.plot(x[:zoomFaktor], iftSound.real[:zoomFaktor], "-r" ,label='Inverse Fourier-T
```

```
    plt.legend()
    plt.show()
```

In [6]:
```
def drawFtPlot(upperAdjustment, lowerAdjustment):
    ftXAxis= np.linspace(0, sr, len(np.abs(ftSound)))

    plt.figure(figsize=(18,7))
    plt.xlabel("frequency", fontsize=15)
    plt.title(name + " Fourier Transformed", fontsize=15)
    plt.plot(ftXAxis[lowerAdjustment:upperAdjustment], np.abs(ftSound)[lowerAdjustment:
    plt.show()
```

In [7]:
```
def bandpassFilter(freqMin, freqMax):
    """
    inspired from: https://github.com/alessandro-gentilini/opencv_exercises-butterworth
    """
    tempFft = ftSound.copy()

    frequencies = np.fft.fftfreq(mainArray.shape[0], d=1.0/sr)

    # Indizes des unteren endes finden
    boundLow = (np.abs(frequencies - freqMin)).argmin()
    # indizes des oberen endes finden
    boundHigh = (np.abs(frequencies - freqMax)).argmin()

    tempFft[:boundLow] = 0
    tempFft[-boundLow:] = 0
    tempFft[boundHigh:-boundHigh] = 0

    transformIfft(tempFft)
```

Einführung

Eine Frequenz beschreibt die Zyklen pro Sekunde für ein beliebiges Signal. Frequenzen werden in Hertz angegeben.

$$Hz = 1s^{-1}$$

Das menschliche Gehör kann das Spektrum von 20Hz - 20.000Hz noch erfassen.
Wobei sich jedoch die menschliche Sprache nur von 85Hz - 100Hz strecken kann.

Im Folgenden werden kurz die zwei Hauptmerkmale von Tönen beschrieben.

1) Frequenzen: Je höher die Frequenz ist, desto höher ist der Ton. Somit werden mit tieferen Frequenzen tiefere Töne erreicht.

In [8]:
```
#Ton mit 30 Hz:
lowFreqSound = ipd.Audio(lowFreq)
#Ton mit 800 Hz:
highFreqSound = ipd.Audio(highFreq)
print("Achten Sie auf die Lautstärke! Beginnen Sie mit einer geringen Lautstärke und st
display(lowFreqSound)
display(highFreqSound)
```

```
Achten Sie auf die Lautstärke! Beginnen Sie mit einer geringen Lautstärke und stellen Si
e die Lautstärke von dort aus ein
```

0:00 / 0:02

0:00 / 0:02

2) Amplitude: Je höher die Amplitude eines Tons ist, desto lauter ist er. Im Gegenschluss ist der Ton leiser je kleiner die Amplitude.

In [9]:
```
#Ton mit kleiner amplitude:
lowAmpSound = ipd.Audio(lowAmp)
#Ton mit großer amplitude:
highAmpSound = ipd.Audio(highAmp)
print("Achten Sie auf die Lautstärke! Beginnen Sie mit einer geringen Lautstärke und st
display(lowAmpSound)
display(highAmpSound)
```

```
Achten Sie auf die Lautstärke! Beginnen Sie mit einer geringen Lautstärke und stellen Si
e die Lautstärke von dort aus ein
```

0:00 / 0:01

0:00 / 0:01

Menü Erzeugung und Signal auswahl

In [10]:
```
sound = widgets.Dropdown(
       options=["410Hz+420Hz+430Hz", "410Hz+510Hz+610Hz", "Square Signal 100Hz"],
       value="410Hz+510Hz+610Hz",
       description="Sound:")
sound
```

In [11]:
```
if sound.value == "410Hz+420Hz+430Hz":
    mainSound = multi400
    global name
    name = "Sinus 410Hz + 420Hz + 430Hz"
elif sound.value == "410Hz+510Hz+610Hz":
    mainSound = multi456
    name = "Sinus 410Hz + 510Hz + 610Hz"
elif sound.value == "Square Signal 100Hz":
    mainSound = square100
    name = "Square Signal 100Hz"
else:
    raise NameError("Chosen sound doesnt match any sounds")
```

```
loadSound(mainSound)

print("\nJetzt können Sie sich den Ton anhören, auf dem wir die Fourier-Transformation
print("Achten Sie auf die Lautstärke! Beginnen Sie mit einer geringen Lautstärke und st
ipd.Audio(mainSound)
```

```
Jetzt können Sie sich den Ton anhören, auf dem wir die Fourier-Transformation ausführen
werden
Achten Sie auf die Lautstärke! Beginnen Sie mit einer geringen Lautstärke und stellen Si
e die Lautstärke von dort aus ein
```

Out[11]:

0:00 / 0:02

Wie man anhand dieses Tones möglicherweise erhören kann handelt es sich hierbei um ein Signal, dass sich aus mehreren Signalen mit verschiedenen Frequenzen zusammensetzt.

Dies basiert auf dem Prinzip der Interferenz. Hierbei wird die Auslenkung der einzelnen Signale zusammenaddiert, wodurch sich die Amplitude im zusammengesetzten Signal vergrößert oder verkleinert.
Durch die unterschiedliche Amplitudengröße erhält man somit den Ton, der lauter und leiser wird.

Dies ist in folgendem Bild verdeutlicht.

Interferenzen Beispiel

Quelle

Visualierung der original Daten

In [12]:

```
slider = interact(drawSinPlot, zoomFaktor=widgets.IntSlider(min=0, max=sr/2, step= 10,
display(slider)
```

```
<function __main__.drawSinPlot(zoomFaktor)>
```

Ausführung der Fourier-Transformation

Mithilfe der Fourier-Transformation kann ein periodisches zeitdiskretes Signal aus dem Zeitspektrum in das Frequenzspektrum überführt werden.

Diese Verfahren wird in der Signalverarbeitung oft verwendet, um die einzelnen Frequenzen in einem zusammengesetzten Signal zu identifizieren oder nicht periodische Funktionen durch die Linearkombination aus Sinus und Cosinus-Funktionen (welche verschiedene Frequenzen haben) darzustellen.

Da im praktischen meistens eine Menge diskreter Werte vorliegt wird eine Diskrete Fourier-Transformation durchgeführt.
Es sind verschiedene Variationen der Fourier-Transformation möglich, die sich im Vorzeichen des

Exponenten oder der Normalisierung bemerkbar machen.
Im Folgenden wird diese Varianten der Fourier-Transformation verwendet.

$$X[k] = \sum_{n=0}^{N-1} x[n] * e^{-2\pi i \frac{mk}{N}} \qquad k = 0, . , n-1$$

In [13]:
```
global ftSound
ftSound = np.fft.fft(mainArray)
```

In [14]:
```
print("Aufgrund der Symmetrie wird hier nur die Hälfte der Fourier Transformation gezei
slider = interact(drawFtPlot, upperAdjustment=widgets.IntSlider(min=0, max=int(len(ftSo
display(slider)
```

```
Aufgrund der Symmetrie wird hier nur die Hälfte der Fourier Transformation gezeigt
```

```
<function __main__.drawFtPlot(upperAdjustment, lowerAdjustment)>
```

Information: Die nächsten Zellen müssen ausgeführt werden, jedoch werden die Funktionen zum Schluss in einem interaktiven Plot verwendet. Aus Übersichtlichkeitsgründen wurde die Reihenfolge so gewählt.

Ausführung der inversen Fourier-Transformation

Durch die inverse Fourier-Transformation wird die Rücktransformation vom Frequenzspektrum in das Zeitsprektrum erfüllt.

In dieser Implementierung wird die Normalisierung hier durchgeführt. (zu sehen an $\frac{1}{N}$).

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] * e^{2\pi i \frac{mk}{N}} \qquad m = 0, . , n-1$$

In [15]:
```
def transformIfft(fftData):
    global iftSound
    iftSound = np.fft.ifft(fftData)
    print("    RMSE :" + str(calcRMSE()))
    slider = interact(drawIftPlot, zoomFaktor=widgets.IntSlider(min=0, max=44100, step=
    display(slider)
```

Grad der Abweichung

Zur Bestimmung der Abweichung von dem original zum invers Fourier-Transformierten Signal wird der Root Mean Square Error (RMSE) verwendet.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2}$$

$$N = Anzahl\ an\ verglichenen\ Werten$$
$$y_i = Tatsächliche\ Daten$$
$$\hat{y}_i = Berechnete\ Daten (Vorhergesagte\ Daten\ häufig)$$

Mithilfe dieses Verfahrens kann eine Beurteilung über die Übereinstimmung eines generierten Signals zu einem vorliegendem Signal getätigt werden.
Häufige Verwendung des RMSE ist bei der Vorhersage von Daten. Hierbei werden vorhergesagte Daten (Corona Daten, Wetter Daten, etc..) mit tatsächlichen (eingetretenen) Daten verglichen, um die Genauigkeit eines Algorithmus zu beurteilen.

> Je größer der RMSE ist, desto weiter entfernt sind die verglichenen Werte und somit ist der benutzte Algorithmus schlechter.
> Je kleiner der RMSE wird, desto besser wird der Algorithmus. Ideal wäre ein Wert von 0, der aber in der Realität fast nie erreicht wird.

In [16]:
```
def calcRMSE():
    RMSE = sqrt(np.square(np.subtract(iftSound.real, mainArray)).mean())
    return RMSE.real
```

Filterung des Signals

Mithilfe eines Bandpasses wird im folgenden ein Frequenzbereich ausgewählt, der mithilfe der Fourier-Transformation in das Zeitspektrum zurücktransformiert wird.
Bei der Rücktransformation werden nur Frequenzen berücksichtigt, die sich in dem angegebenen Frequenzband befinden (zwischen freqMin und freqMax).
Frequenzen, die nicht in dem angegebenen Frequenzband liegen, werden nicht berücksichtigt, wodurch Abweichungen zum originalen Signal eintreten können.

> Anzumerken ist, dass im unteren interaktiven Plot die Grenzen nicht inklusive sind. Dies bedeutet, dass, falls das Signal eine Frequenz von 410 Hz hat und freqMin auf 410 Hz eingestellt ist diese Frequenz nicht berücksichtigt wird und somit der RMSE steigt. Bei einem freqMin von 400 Hz und einem frqMax > als 410 Hz würde die Frequenz im Bandpass liegen. Falls diese die einzige Frequenz im Signal ist, würde der RMSE sehr klein werden.

In [17]:
```
# Because using the slider with 20000 as max Value we just set it when we want to analy
# In this range you can see all of the different forms of the rect signal
# The value is set to 99 and doesnt need to be changed
if name == "Square Signal 100Hz":
    maxValue = 20000
    value = 99
else:
    maxValue = 1000
    value = 410
```

```
sliderFilter = interact(bandpassFilter, freqMin=widgets.IntSlider(min=0, max=maxValue,
display(sliderFilter)
```

```
<function __main__.bandpassFilter(freqMin, freqMax)>
```

In [ ]: