iNFO

IUT GRAND OUEST NORMANDIE

R 5.A.06

2025 – 2026

Sensibilisation à la programmation multimédia

TP n° 1
Programmation du son

ANNE Jean-François

Sensibilisation à la programmation multimédia

Programmation du son

Le but de ce TP est de se familiariser avec la programmation Python en audio.

A. Exercices de base :

1°) Exercice 1

Un fichier audio WAV (PCM non compressé) possède les propriétés suivantes : stéréo, 16 bits, 44100 Hz, taille du fichier 97896 octets

1) Quelle est la taille de l'en-tête (en octets) ?

   Pour déterminer la taille de l'en-tête, il faut connaître le format du fichier WAV utilisé. Dans la plupart des cas, le format utilisé est le format standard PCM (Pulse Code Modulation), qui utilise un en-tête de 44 octets. Cet en-tête est composé de plusieurs champs qui contiennent des informations sur le format audio, telles que le nombre de canaux, le nombre de bits par échantillon, le taux d'échantillonnage, etc.

   Donc, dans notre cas, la taille de l'en-tête est de 44 octets.

2) Quelle est la taille des données (en octets) ?

   La taille des données peut être calculée en soustrayant la taille de l'en-tête (44 octets) de la taille totale du fichier (97896 octets). Donc, la taille des données est de 97896 - 44 = 97852 octets.

3) Quelle est la durée du son de ce fichier (en ms) ?

   Calculons la durée du son :

   Durée (ms) = (97852 octets / (44100 Hz * (16 bits / 8) * 2 canaux)) * 1000

   Durée (ms) = (97852 / (44100 * (2) * (2))) * 1000

   Durée (ms) ≈ 0,5547165532879819 * 1000

   Durée (ms) ≈ 554,7165 ms

   La durée du son du fichier audio WAV, en tenant compte du format stéréo 16 bits à 44100 Hz, est d'environ 553,2879 millisecondes.

2°) Exercice 2

Afficher en python les caractéristiques d'un fichier son fourni

```python
import wave

# Remplacez le chemin par le chemin absolu vers votre fichier audio WAV
file_path = "./Audios/Piano La3.wav"

# Fonction pour afficher les caractéristiques du fichier audio WAV


def display_wav_info(file_path):
    with wave.open(file_path, "rb") as wav_file:
        num_channels = wav_file.getnchannels()
        sample_width = wav_file.getsampwidth()
        frame_rate = wav_file.getframerate()
```

```
        num_frames = wav_file.getnframes()
        duration = num_frames / frame_rate

        print("Informations sur le fichier audio WAV:")
        print(f"Nombre de canaux: {num_channels}")
        print(f"Largeur de l'échantillon en octets: {sample_width}")
        print(f"Taux d'échantillonnage (fréquence): {frame_rate} Hz")
        print(f"Nombre total de frames: {num_frames}")
        print(f"Durée du fichier audio: {duration:.2f} secondes")

display_wav_info(file_path)
```

3°) ***Exercice 3***

Créer un programme python qui joue les notes sur une octave

Version 1 en utilisant Pygame en version midi

Voici un exemple de programme Python 3 qui utilise la bibliothèque pygame pour jouer des notes sur une octave :

Python

```
import pygame.midi

# Initialisation de la bibliothèque Pygame MIDI
pygame.midi.init()

# Ouvrir un port MIDI pour jouer les notes
port = pygame.midi.Output(0)

# Définir la durée de chaque note en millisecondes
duration = 500

# Définir les notes de l'octave à jouer
notes = [60, 62, 64, 65, 67, 69, 71, 72]

# Jouer chaque note pendant la durée spécifiée
for note in notes:
    port.note_on(note, 127)
    pygame.time.delay(duration)
    port.note_off(note, 127)

# Fermer le port MIDI et terminer l'utilisation de la bibliothèque Pygame MIDI
port.close()
pygame.midi.quit()
```

Ce programme utilise la fonction note_on() de l'objet Output de pygame.midi pour jouer chaque note de l'octave, et la fonction note_off() pour arrêter la note après la durée spécifiée. La fonction pygame.time.delay() est utilisée pour attendre la durée spécifiée entre chaque note. Notez que le code ci-dessus joue les notes sur le port MIDI 0, qui est généralement le port par défaut sur la plupart des ordinateurs. Si vous avez plusieurs ports MIDI disponibles, vous devrez peut-être ajuster ce numéro pour jouer les notes sur le port MIDI souhaité.

Version 2 en utilisant les fréquences

Bien sûr, voici un exemple de programme Python 3 qui utilise la bibliothèque numpy pour générer des fréquences sinusoidales et la bibliothèque sounddevice pour jouer les notes sur une octave:

Python

```
import numpy as np
import sounddevice as sd

# Définir les paramètres audio
sample_rate = 44100
duration = 0.5

# Définir les fréquences de chaque note de l'octave
notes = [261.63, 293.66, 329.63, 349.23, 392.00, 440.00, 493.88, 523.25]

# Générer une onde sinusoïdale pour chaque note de l'octave
samples = []
for note in notes:
    t = np.linspace(0, duration, int(duration * sample_rate), False)
    note_sample = np.sin(2*np.pi*note*t)
    samples.append(note_sample)

# Concaténer les échantillons pour chaque note en un seul tableau
d'échantillons
audio_data = np.concatenate(samples)

# Jouer les notes de l'octave
sd.play(audio_data, sample_rate)
sd.wait()
```

Ce programme utilise la fonction np.linspace() de numpy pour générer un échantillonnage temporel pour chaque note de l'octave, et la fonction np.sin() pour générer une onde sinusoïdale pour chaque note à la fréquence correspondante. Les échantillons sont ensuite concaténés en un seul tableau d'échantillons à jouer, et la fonction sd.play() de sounddevice est utilisée pour jouer le tableau d'échantillons à la fréquence d'échantillonnage spécifiée. La fonction sd.wait() est utilisée pour attendre la fin de la lecture du tableau d'échantillons avant de terminer le programme. Notez que les fréquences de chaque note sont exprimées en Hertz et peuvent être ajustées pour jouer d'autres notes ou octaves.

4°) ***Exercice 4 : Bruit blanc***

Ecrire un programme python qui génère un fichier .wav qui contient 10s de bruit blanc, qui affiche son courbe en fonction du temps et qui affiche son spectre.

Version 1

Voici un exemple de programme Python qui génère un fichier WAV contenant 10 secondes de bruit blanc :

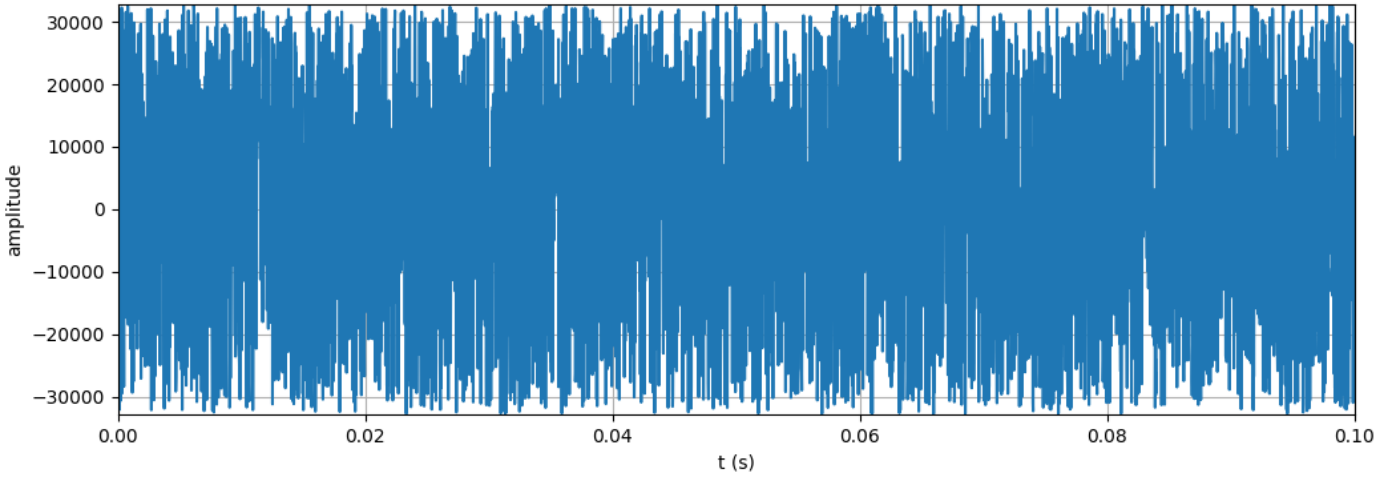
Python

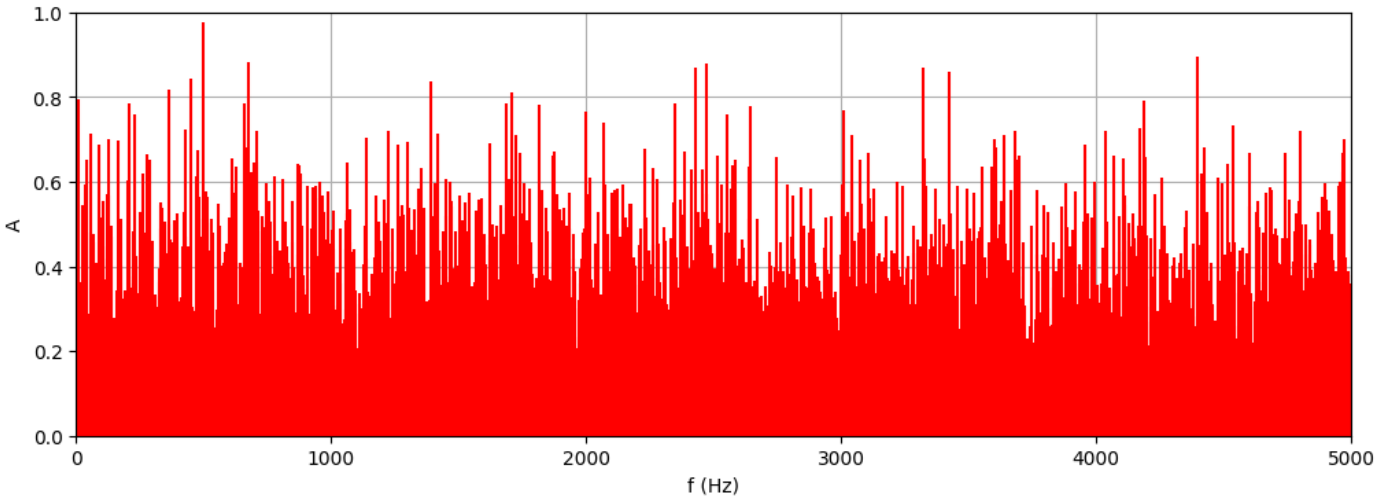
```
# Importation des modules:
import wave
import struct
import random
import scipy.io.wavfile as wave_scipy
```

```
from numpy.fft import fft
from matplotlib.pyplot import figure, plot, xlabel, ylabel, axis, grid, vlines

# Paramètres du fichier WAV
freq = 44100
nchannels = 1
sampwidth = 2

# Durée du signal
duration = 10  # en secondes

# Création du fichier WAV
wav_file = wave.open('bruit_blanc.wav', 'wb')
wav_file.setparams((nchannels, sampwidth, freq, 0, 'NONE', 'not compressed'))

# Génération du signal
for i in range(int(freq*duration)):
    # Génère une valeur aléatoire entre -32768 et 32767
    value = random.randint(-32768, 32767)
    # Convertit la valeur en binaire sur 16 bits
    data = struct.pack('<h', value)
    # Écrit les données dans le fichier WAV
    wav_file.writeframesraw(data)

wav_file.close()

# Lecture du fichier WAV:
rate, data = wave_scipy.read('bruit_blanc.wav')
n = data.size
duree = 1.0*n/rate

# Affichage de la Fréquence d'échantillonnage (en Hz) et durée de l'enregistrement:
print("La fréquence d'échantillonnage est de ", rate)
print("La durée de l'enregistrement est de ", duree)

# Tracé du signal:
te = 1.0/rate
t = np.arange(n)*te
figure(figsize=(12, 4))
plot(t, data)
xlabel("t (s)")
ylabel("amplitude")
axis([0, 0.1, data.min(), data.max()])
grid()

# Tracé du spectre
# Calcul du spectre avec la transformée de Fourier discrète, pour une fenêtre dont
on précise le début et la durée(en secondes):


def tracerSpectre(data, rate, debut, duree):
    start = int(debut*rate)
    stop = int((debut+duree)*rate)
```

```
spectre = np.absolute(fft(data[start:stop]))
spectre = spectre/spectre.max()
n = spectre.size
freq = np.arange(n)*1.0/n*rate
vlines(freq, [0], spectre, 'r')
xlabel('f (Hz)')
ylabel('A')
axis([0, 0.5*rate, 0, 1])
grid()
```

```
figure(figsize=(12, 4))
tracerSpectre(data, rate, 0.0, 0.5)
axis([0, 5000, 0, 1])
```



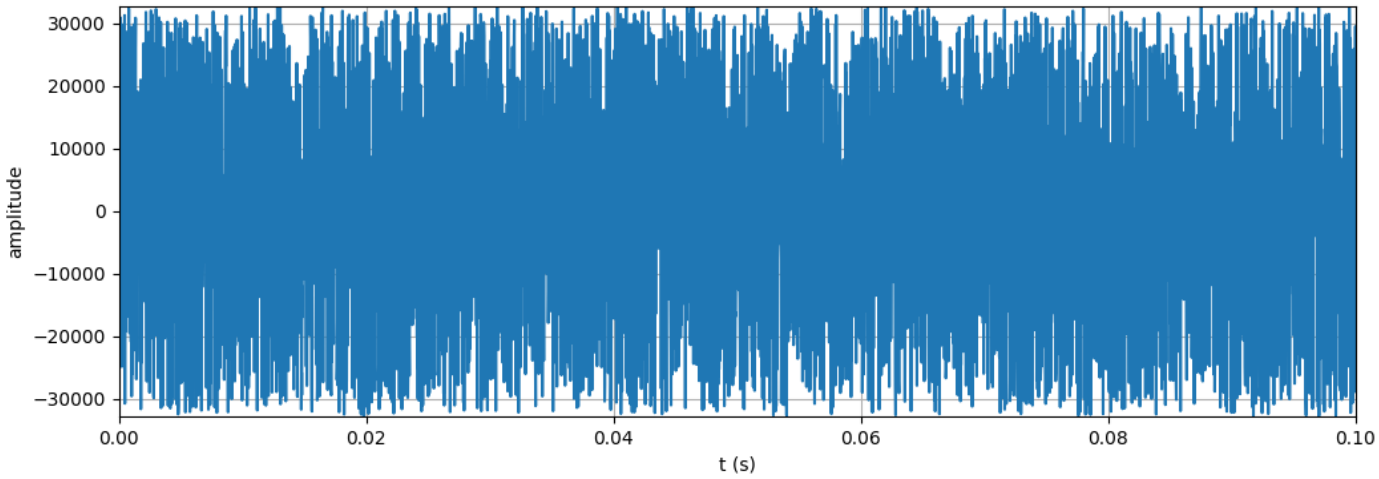


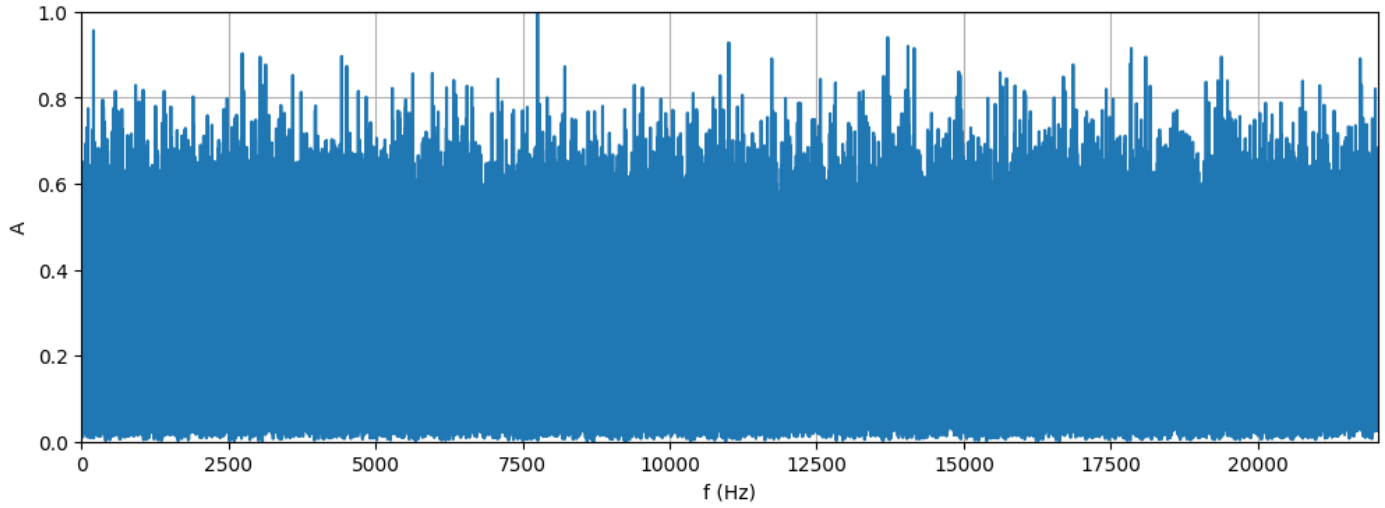
Ce programme utilise la bibliothèque wave pour créer un fichier WAV. La fréquence d'échantillonnage est définie à 44100 Hz, le nombre de canaux est 1 (mono) et la taille de chaque échantillon est de 2 octets (16 bits). Le signal est généré en bouclant 10 secondes (le nombre d'échantillons est égal à la fréquence multipliée par la durée), et en générant une valeur aléatoire pour chaque échantillon. La valeur est convertie en binaire sur 16 bits et écrite dans le fichier WAV en utilisant la méthode writeframesraw.

Version 2

```
import numpy as np
import scipy.io.wavfile as wavfile
import matplotlib.pyplot as plt
```

```
# Paramètres du fichier WAV
freq = 44100
nchannels = 1
sampwidth = 2
duration = 10  # en secondes

# Création du fichier WAV
data = np.random.randint(-32768, 32767, size=freq*duration)
wavfile.write('bruit_blanc.wav', freq, data.astype(np.int16))

# Lecture du fichier WAV
rate, data = wavfile.read('bruit_blanc.wav')
n = len(data)
duree = n/rate

# Affichage de la Fréquence d'échantillonnage (en Hz) et durée de l'enregistrement :
print("La fréquence d'échantillonnage est de ", rate)
print("La durée de l'enregistrement est de ", duree)

# Tracé du signal:
t = np.arange(n)/rate
plt.figure(figsize=(12, 4))
plt.plot(t, data)
plt.xlabel("t (s)")
plt.ylabel("amplitude")
plt.axis([0, 0.1, data.min(), data.max()])
plt.grid()

# Tracé du spectre
spectre = np.abs(np.fft.fft(data))/n
spectre = spectre/spectre.max()
freq = np.arange(n)*1.0/n*rate
plt.figure(figsize=(12, 4))
plt.plot(freq, spectre)
plt.xlabel('f (Hz)')
plt.ylabel('A')
plt.axis([0, 0.5*rate, 0, 1])
plt.grid()

plt.show()
```





Version simplifiée

5°) ***Encodeur de Code DTMF (dual-tone multi-frequency)***

Ecrire un programme python qui génère et sauvegarde un fichier .wav qui contient les tonalité DTMF d'un numéro de téléphone à 10 chiffres, et qui affiche sa courbe et son spectre.

Version 1

Voici un exemple de programme Python qui génère et sauvegarde un fichier WAV contenant les tonalités DTMF d'un numéro de téléphone à 10 chiffres :

Python

```python
import wave
import math
import scipy.io.wavfile as wave_scipy

# Fréquences des tonalités DTMF
frequencies = {
    '1': (697, 1209),
    '2': (697, 1336),
    '3': (697, 1477),
    '4': (770, 1209),
    '5': (770, 1336),
    '6': (770, 1477),
    '7': (852, 1209),
    '8': (852, 1336),
    '9': (852, 1477),
    '*': (941, 1209),
    '0': (941, 1336),
    '#': (941, 1477)
}

# Durée de chaque tonalité en secondes
tone_duration = 0.5

# Durée de silence entre chaque tonalité en secondes
silence_duration = 0.1

# Numéro de téléphone
```

```
phone_number = '0123456789'

# Fréquence d'échantillonnage
sampling_frequency = 44100

# Nombre de canaux
num_channels = 1

# Taille de chaque échantillon en octets
sample_width = 2

# Calcul de la durée totale du signal
total_duration = (len(phone_number) * tone_duration) + \
    ((len(phone_number) - 1) * silence_duration)

# Création du fichier WAV
wav_file = wave.open('dtmf.wav', 'wb')
wav_file.setparams((num_channels, sample_width,
                    sampling_frequency, 0, 'NONE', 'not compressed'))

# Génération du signal
for digit in phone_number:
    # Récupération des fréquences de la tonalité
    freq1, freq2 = frequencies[digit]

    # Nombre d'échantillons pour chaque tonalité
    num_samples = int(tone_duration * sampling_frequency)

    # Génération de la tonalité
    for i in range(num_samples):
        t = float(i) / float(sampling_frequency)
        value = int(32767 * (math.sin(2.0 * math.pi * freq1 * t) +
                    math.sin(2.0 * math.pi * freq2 * t)))
        data = wave.struct.pack('<i', value)
        wav_file.writeframesraw(data)

    # Ajout d'un silence entre chaque tonalité
    num_samples = int(silence_duration * sampling_frequency)
    for i in range(num_samples):
        data = wave.struct.pack('<h', 0)
        wav_file.writeframesraw(data)

# Fermeture du fichier WAV
wav_file.close()
# Lecture du fichier WAV:
rate, data = wave_scipy.read('dtmf.wav')
n = data.size
duree = 1.0*n/rate

# Affichage de la Fréquence d'échantillonnage (en Hz) et durée de l'enregistrement:
print("La fréquence d'échantillonnage est de ", rate)
print("La durée de l'enregistrement est de ", duree)
```

```
# Tracé du signal:
te = 1.0/rate
t = np.arange(n)*te
figure(figsize=(12, 4))
plot(t, data)
xlabel("t (s)")
ylabel("amplitude")
axis([0, 11, data.min(), data.max()])
grid()

# Tracé du spectre
# Calcul du spectre avec la transformée de Fourier discrète, pour une fenêtre dont
on précise le début et la durée(en secondes):


def tracerSpectre(data, rate, debut, duree):
    start = int(debut*rate)
    stop = int((debut+duree)*rate)
    spectre = np.absolute(fft(data[start:stop]))
    spectre = spectre/spectre.max()
    n = spectre.size
    freq = np.arange(n)*1.0/n*rate
    vlines(freq, [0], spectre, 'r')
    xlabel('f (Hz)')
    ylabel('A')
    axis([0, 0.5*rate, 0, 1])
    grid()


figure(figsize=(12, 4))
tracerSpectre(data, rate, 0.0, 10)
axis([0, 10000, 0, 1])
```

Ce programme utilise les fréquences standard des tonalités DTMF et les durées de tonalité et de silence spécifiées. Il génère un signal pour chaque chiffre du numéro de téléphone en utilisant la méthode de sommation de deux sinusoïdes pour chaque tonalité. Le signal est ensuite écrit dans le fichier WAV en utilisant la méthode writeframesraw. Le fichier WAV est créé avec une fréquence d'échantillonnage de 44100 Hz, un canal mono et une taille d'échantillon de 2 octets (16 bits).

Version 2 :

```
from pydub import AudioSegment
import struct
from math import sin, pi
from datetime import timedelta

DTMF_TONES = {
    '0': (941, 1336),
    '1': (697, 1209),
    '2': (697, 1336),
    '3': (697, 1477),
    '4': (770, 1209),
    '5': (770, 1336),
    '6': (770, 1477),
    '7': (852, 1209),
```

```
    '8': (852, 1336),
    '9': (852, 1477),
}


def generate_sine_wave(duration, frequency):
    # Fréquence d'échantillonnage (nombre d'échantillons par seconde)
    sample_rate = 44100

    # Nombre total d'échantillons dans la durée donnée
    num_samples = int(sample_rate * duration / 1000)

    # Créer une liste de valeurs d'échantillon pour l'onde sinusoïdale
    samples = [int(32767 * float(sin(2 * pi * frequency * t / sample_rate)))
               for t in range(num_samples)]

    # Convertir la liste en un objet bytes
    raw_data = struct.pack('h' * num_samples, *samples)

    # Créer un objet AudioSegment à partir des données brutes
    audio_segment = AudioSegment(
        raw_data, frame_rate=sample_rate, sample_width=2, channels=1)

    return audio_segment


def generate_dtmf_tones(phone_number):
    dtmf_sequence = []
    for digit in phone_number:
        dtmf_sequence.append(DTMF_TONES[digit])

    audio = AudioSegment.empty()
    for dtmf_tone in dtmf_sequence:
        duration = 500  # ms
        frequency_1, frequency_2 = dtmf_tone

        # Créer une tonalité DTMF avec une durée de 500 ms
        tone = AudioSegment.silent(duration=duration)

        # Ajouter la première fréquence de la tonalité DTMF
        sine_wave_1 = generate_sine_wave(duration, frequency_1)
        tone = tone.overlay(sine_wave_1)

        # Ajouter la deuxième fréquence de la tonalité DTMF
        sine_wave_2 = generate_sine_wave(duration, frequency_2)
        tone = tone.overlay(sine_wave_2)

        # Ajouter un silence de 100 ms après la tonalité DTMF
        silence = AudioSegment.silent(duration=100)
        audio += tone + silence

    return audio
```

```
phone_number = '0231555234'
audio = generate_dtmf_tones(phone_number)
audio.export('dtmf1.wav', format='wav')

# Tracé temporel
samples = audio.get_array_of_samples()
duration = len(samples) / audio.frame_rate
time = np.linspace(0, duration, len(samples))
plt.plot(time, samples)
plt.xlabel('Temps (s)')
plt.ylabel('Amplitude')
plt.title('Signal audio en fonction du temps')
plt.show()

# Tracé en fréquence
N = len(samples)
Y = np.fft.fft(samples)
freqs = np.fft.fftfreq(N, d=1/audio.frame_rate)
plt.plot(freqs[:N//2], 20*np.log10(np.abs(Y[:N//2])), color="red")
plt.xlabel('Fréquence (Hz)')
plt.ylabel('Magnitude (dB)')
plt.title('Spectre de fréquence')
plt.show()
```

Version 3 :

```
import numpy as np
import scipy.io.wavfile as wavfile
import matplotlib.pyplot as plt

# Fonction pour générer le signal audio DTMF pour un chiffre donné


def generate_dtmf_tone(digit, duration, sample_rate):
    dtmf_tones = {
        "1": (697, 1209),
        "2": (697, 1336),
        "3": (697, 1477),
        "4": (770, 1209),
        "5": (770, 1336),
        "6": (770, 1477),
        "7": (852, 1209),
        "8": (852, 1336),
        "9": (852, 1477),
        "0": (941, 1336)
    }

    t = np.linspace(0, duration, int(sample_rate * duration), endpoint=False)
    frequency1, frequency2 = dtmf_tones[digit]
    tone = 0.5 * np.sin(2 * np.pi * frequency1 * t) + \
        0.5 * np.sin(2 * np.pi * frequency2 * t)
    return tone

# Fonction pour générer le signal audio DTMF pour un numéro de téléphone donné
```

```
def generate_phone_tones(phone_number, duration, sample_rate):
    tones = []
    for digit in phone_number:
        tones.append(generate_dtmf_tone(digit, duration, sample_rate))
    return np.concatenate(tones)


# Paramètres
phone_number = "1234567890"  # Numéro de téléphone à 10 chiffres
duration = 0.3  # Durée de chaque tonalité en secondes
sample_rate = 44100  # Taux d'échantillonnage en Hz

# Génération du signal audio DTMF
dtmf_signal = generate_phone_tones(phone_number, duration, sample_rate)

# Sauvegarde du fichier WAV
wavfile.write("dtmf_tones.wav", sample_rate, np.int16(dtmf_signal * 32767))

# Affichage individuel des courbes d'amplitude et des spectres de chacun des
chiffres
for i, digit in enumerate(phone_number):
    digit_signal = generate_dtmf_tone(digit, duration, sample_rate)

    # Affichage de la courbe d'amplitude
    time = np.linspace(0, duration, len(digit_signal))
    plt.figure(figsize=(8, 4))
    plt.plot(time, digit_signal)
    plt.title(f"Courbe d'amplitude pour le chiffre {digit}")
    plt.xlabel("Temps (secondes)")
    plt.ylabel("Amplitude")
    plt.xlim( 0 , 0.010 )
    plt.grid(True)
    plt.show()

    # Affichage du spectre fréquentiel
    plt.figure(figsize=(8, 4))
    plt.magnitude_spectrum(digit_signal, Fs=sample_rate, scale='dB')
    plt.title(f"Spectre fréquentiel pour le chiffre {digit}")
    plt.xlabel("Fréquence (Hz)")
    plt.xlim( 500 , 2000 )
    plt.ylabel("Amplitude (dB)")
    plt.grid(True)
    plt.show()
```

6°) ***Echo acoustique***

Ecrire un programme python qui lit un fichier .wav, ajoute de l'écho, le joue sur le pc et le sauvegarde dans un fichier .wav.

Pour ajouter un écho à un fichier audio .wav, on peut utiliser la fonction scipy.signal.convolve pour convoluer le signal avec une réponse impulsionnelle représentant l'écho. Voici un exemple de

programme Python qui lit un fichier .wav, ajoute de l'écho et enregistre le signal échoé dans un nouveau fichier .wav :

Python

```
import sounddevice as sd
import numpy as np
import scipy.io.wavfile as wav
from scipy import signal

# Lire le fichier .wav
fs, signal_in = wav.read("Audios/Cri_wilhelm.wav")

# Définir la réponse impulsionnelle pour l'écho
delay = int(0.5 * fs)  # retard de 0,5 seconde
gain = 0.5  # gain de l'écho
numtaps = delay + 1
taps = np.zeros(numtaps)
taps[0] = 1
echo = signal.lfilter(taps, [1.0] + [0.0]*(numtaps-1), signal_in)*gain

# Ajouter l'écho au signal d'entrée
signal_out = signal_in + echo

# Jouer le signal sur le pc
sd.play(signal_out, fs)
sd.wait()

# Sauvegarder le signal dans un fichier .wav
wav.write("Audios/Echo_Cri_wilhelm.wav", fs, signal_out.astype(np.int16))
```

Notez que ce code utilise la bibliothèque sounddevice pour jouer le signal sur le PC, il peut donc être nécessaire de l'installer au préalable avec pip install sounddevice.

Version 2 :

```
import wave
import numpy as np
import sounddevice as sd

# Fonction pour ajouter de l'écho au signal audio


def add_echo(signal, sample_rate, delay_sec=0.5, decay=0.5):
    num_samples_delay = int(sample_rate * delay_sec)
    echo = np.zeros_like(signal)
    echo[num_samples_delay:] = signal[:-num_samples_delay] * decay
    return signal + echo


# Charger le fichier .wav
# Remplacez par le chemin vers votre fichier .wav
file_path = "Audios/Piano La3.wav"
with wave.open(file_path, 'rb') as wav_file:
    num_channels = wav_file.getnchannels()
    sample_width = wav_file.getsampwidth()
    frame_rate = wav_file.getframerate()
```

```
    num_frames = wav_file.getnframes()
    audio_data = np.frombuffer(wav_file.readframes(num_frames), dtype=np.int16)

# Ajouter de l'écho au signal audio

for i in range(4):
    audio_with_echo = add_echo(audio_data, frame_rate, 0.5, 0.8)
    audio_data = audio_with_echo

# Jouer le signal audio avec écho sur le PC
print("Lecture du signal audio avec écho...")
sd.play(audio_with_echo, samplerate=frame_rate)
sd.wait()

# Sauvegarder le signal audio avec écho dans un fichier .wav
# Remplacez par le chemin pour le nouveau fichier .wav
output_file_path = "echo piano la3.wav"
with wave.open(output_file_path, 'wb') as output_wav_file:
    output_wav_file.setnchannels(num_channels)
    output_wav_file.setsampwidth(sample_width)
    output_wav_file.setframerate(frame_rate)
    output_wav_file.writeframes(audio_with_echo.tobytes())

print("Fichier .wav sauvegardé avec écho.")
```

B. Battements

1°) “Battements”.

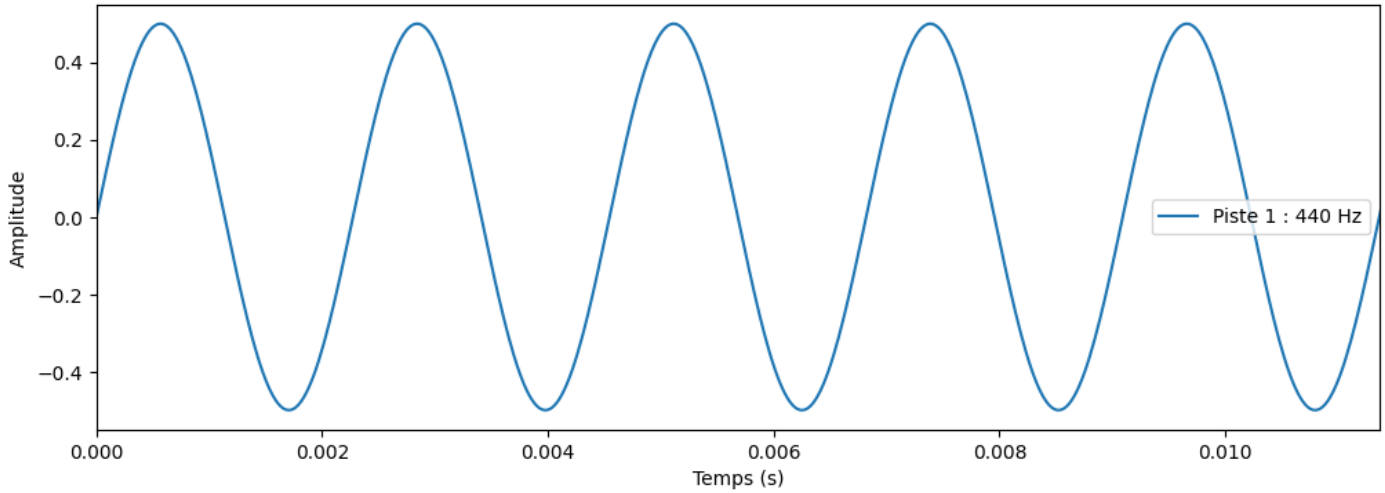
Créer un programme python qui sur deux pistes différentes un signal de fréquence f = 440 Hz et f2 = 444 Hz. Et faire la somme sur une nouvelle piste. On obtient la superposition de deux signaux sinusoïdaux de fréquence f = 440 Hz et f2 = 444 Hz (on verra que c’est une différence de 1/6 de demi-ton, à la limite du perceptible.

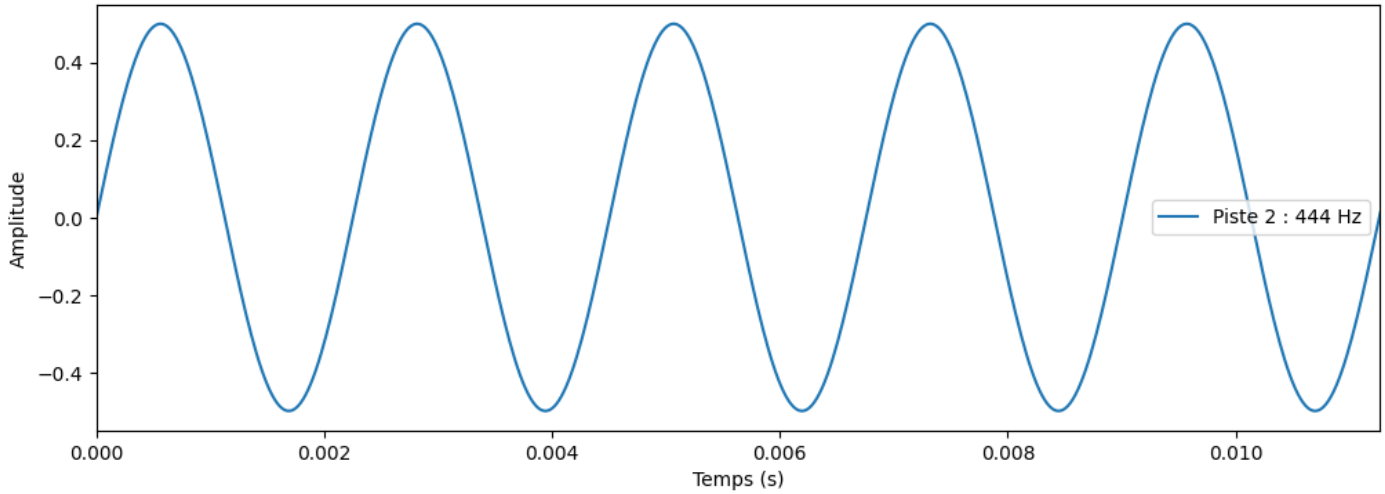
Voici un programme qui génère un signal de fréquence 440 Hz sur la piste 1 et un signal de fréquence 444 Hz sur la piste 2, les combine en sommant les deux signaux et les affiche :

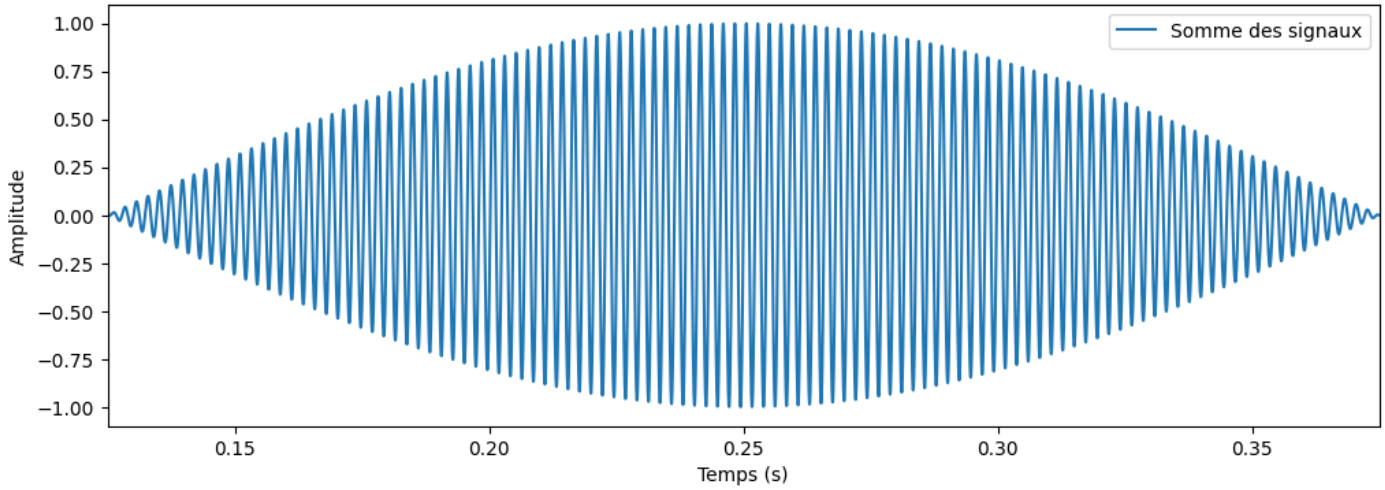
Python

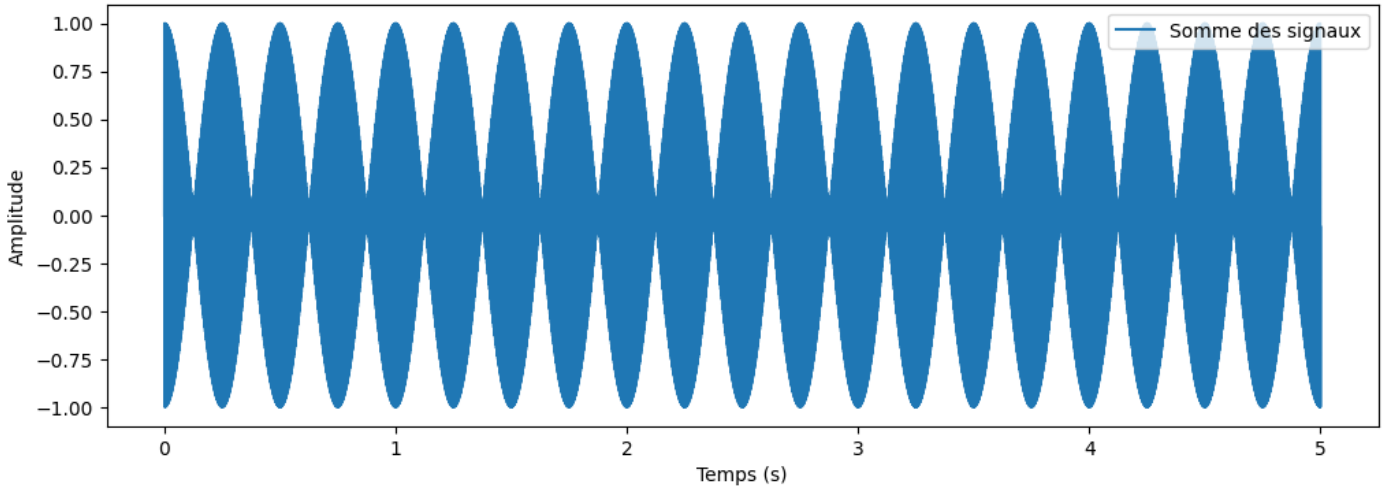
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import write
import sounddevice as sd

# Fréquence d'échantillonnage
fs = 44100

# Durée du signal en secondes
duration = 5

# Générer le signal de fréquence 440 Hz sur la piste 1
t = np.linspace(0, duration, int(fs * duration), endpoint=False)
freq = 440
amplitude = 0.5
```

```
signal1 = amplitude * np.sin(2 * np.pi * freq * t)

# Générer le signal de fréquence 444 Hz sur la piste 2
freq2 = 444
signal2 = amplitude * np.sin(2 * np.pi * freq2 * t)

# Somme des signaux des deux pistes
signal_out = signal1 + signal2

# Tracé des signaux
plt.figure(figsize=(12, 4))
plt.plot(t, signal1, label="Piste 1 : 440 Hz")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.xlim(0,5/freq)
plt.legend()
plt.show()
plt.figure(figsize=(12, 4))
plt.plot(t, signal2, label="Piste 2 : 444 Hz")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.xlim(0, 5/freq2)
plt.legend()
plt.show()
plt.figure(figsize=(12, 4))
plt.plot(t, signal_out, label="Somme des signaux")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.xlim(0.125, 0.375)
plt.legend()
plt.show()
plt.figure(figsize=(12, 4))
plt.plot(t, signal_out, label="Somme des signaux")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.legend()
plt.show()

# Jouer le signal sur le PC
sd.play(signal_out, fs)
sd.wait()

# Enregistrer le signal dans un fichier WAV
write("signal.wav", fs, signal_out)
```


Amplitude
Temps (s)
Piste 1 : 440 Hz


Amplitude
Temps (s)
Piste 2 : 444 Hz


Amplitude
Temps (s)
Somme des signaux



Ce programme utilise la bibliothèque NumPy pour générer les signaux sinusoïdaux, Matplotlib pour tracer les signaux, Sounddevice pour jouer le signal sur le PC, et Scipy pour enregistrer le signal dans un fichier WAV.

On entend des battements dont la fréquence est :

fbatt = |f2 − f1| = 4 Hz (2.1.2)

(i.e. on entends 4 battements par seconde).

3. Donner une explication simple de cette formule ?

3. Explication simple de la formule de battements (2.1.2) (explication plus formelle ci-dessous) : Si à la date t = 0 les deux signaux “sont en phase” c’est à dire que les maximas coïncident alors ensuite ils vont se décaler puis à nouveau se retrouver en phase à la date t lorsque le signal de plus haute fréquence f2 = 1

T2 aura pris une avance d’une période, c’est à dire faisant N + 1 oscillations, lorsque le signal de fréquence f1 = 1 T2 fait N oscillations. Cela donne Tbat. = (N + 1) T2 = N T1, donnant N = T2 T1−T2 et fbat = 1

Tbat = 1 T2 – 1 T1 = f2 − f1.

Essayer en stéréo : générer les signaux u1 (t) = 0.2 cos (2πf t) et u2 (t) = 0.2 cos ((2π) (f + ε) t) sur deux canaux différents. Entendez-vous des battements ?

Solution 2.1.22. Programme battements.py. Fichiers battement.wav et battement_stereo.wav.

Version stéréo :

Python

```
from scipy.io.wavfile import write
import numpy as np
import sounddevice as sd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib

# Fréquence d'échantillonnage
fs = 44100

# Durée du signal
duration = 5  # en secondes
```

```
# Fréquences des signaux à générer
f1 = 440  # Hz
f2 = 444  # Hz

# Nombre total d'échantillons
# total_samples = sampling_freq * signal_duration
t = np.linspace(0, duration, int(fs * duration), endpoint=False)

# Générer les signaux pour chaque voie
signal_left = np.sin(2 * np.pi * f1 * t)
signal_right = np.sin(2 * np.pi * f2 * t)

# Empiler les signaux pour créer un signal stéréo
signal_stereo = np.column_stack((signal_left, signal_right))

# Jouer le signal stéréo sur le PC
sd.play(signal_stereo, fs)
sd.wait()

# Sauvegarder le signal stéréo dans un fichier .wav
write("signal_stereo.wav", fs, signal_stereo)

# Afficher les courbes
plt.figure(figsize=(12, 4))
plt.xlim(0, 5/f1)
plt.plot(t, signal_left, label="Signal 440 Hz (voie gauche)")
plt.title("Signal 440 Hz (voie gauche)")
plt.show()

plt.figure(figsize=(12, 4))
plt.xlim(0, 5/f2)
plt.plot(t, signal_right, label="Signal 444 Hz (voie droite)")
plt.title("Signal 444 Hz (voie droite)")
plt.show()

plt.figure(figsize=(12, 4))
plt.xlim(0, 0.025)
plt.plot(t, signal_stereo, label="Signal 444 Hz (voie droite)")
plt.title("Signal stéréo")
plt.tight_layout()
plt.show()
```

Explication de la formule de battements (2.1.2) : On a besoin de la formule de trigonométrie cos a + cos b = 2 cos a + b 2 cos a – b 2

qui se démontre à partir de la formule de Euler eiθ = cos θ + i sin θ. On l'applique à (2.1.3) avec a = 2πf t et b = (2π) (f + ε) t, avec ε = 0.01, donnant a+b 2 = 2π

C. MP3 :

1°) Exercice 1 :

Ecrire un programme python qui lit un fichier .mp3, qui affiche les caractéristiques du fichier MP3 :

Titre, Artiste, Album, Durée, Taille du fichier, Bitrate, Echantillonnage, Canaux audios, Format audio, Taux de compression.

Vous pourrez après modifier votre programme pour qu'il modifie quelques tags ID de votre MP3.

```
import eyed3

# Chemin vers le fichier MP3
file_path = "./Nomyn-Fragments.mp3"  # Remplacez par le chemin vers votre fichier
.mp3

# Lecture des informations du fichier MP3
audiofile = eyed3.load(file_path)


if not audiofile.tag:
    audiofile.initTag()

# Vérification si le fichier MP3 a des tags ID3
if audiofile.tag is None:
    print("Le fichier MP3 ne contient pas de tags ID3.")
else:
    print("Informations du fichier MP3:")
    print("Titre:", audiofile.tag.title)
    print("Artiste:", audiofile.tag.artist)
    print("Album:", audiofile.tag.album)

    # Vérifier le numéro de piste (track number)
    if audiofile.tag.track_num:
        print("Numéro de piste:", audiofile.tag.track_num[0])
    else:
        print("Numéro de piste: Non disponible")

    # Vérifier si l'année est présente
    if audiofile.tag.release_date:
        print("Année:", audiofile.tag.release_date.year)
    else:
        print("Année: Non disponible")

    # Vérifier le genre (avec gestion d'erreur)
    try:
        genre_name = audiofile.tag.genre.name
        print("Genre:", genre_name)
    except AttributeError:
        print("Genre: Non disponible")
```

```
        print("Durée (en secondes):", audiofile.info.time_secs)
        print("Bitrate (kbps):", audiofile.info.bit_rate[1])
        print("Taux d'échantillonnage (Hz):", audiofile.info.sample_freq)

        # Vérifier le nombre de canaux audio (mono ou stéréo)
        print("audiofile.info.mode:", audiofile.info.mode)

    audiofile.tag.artist = u"Damien : Nomyn"
    audiofile.tag.album = u"1er Album"
    audiofile.tag.title = u"Fragments"
    audiofile.tag.save()


    Informations du fichier MP3:
    Titre: Fragments
    Artiste: Damien : Nomyn
    Album: 1er Album
    Numéro de piste: None
    Année: Non disponible
    Genre: Non disponible
    Durée (en secondes): 227.34
    Bitrate (kbps): 128
    Taux d'échantillonnage (Hz): 44100
    audiofile.info.mode: Joint stereo
```

2°) ***Exercice 2 :***

Ecrire un programme python qui prend un fichier .mp3, qui demande à shazam les caractéristiques du MP3, les affiche :

Titre, Artiste, Album, Durée, Taille du fichier, Bitrate, Echantillonnage, Canaux audios, Format audio, Taux de compression.

Et qui met à jour les informations dans les TAGs ID de votre MP3.

```
import asyncio
from shazamio import Shazam, Serialize, HTTPClient
from mutagen.id3 import TIT2
from mutagen.id3 import TPE1

async def get_track_info(shazam, mpsong):
    new_version_path = await shazam.recognize(mpsong)
    track_info = new_version_path["track"]
    print("titre :", track_info["title"])
    print("artiste :", track_info["subtitle"])
    # Récupération de la durée du fichier (vous devrez peut-être ajuster cette partie
en fonction de votre méthode pour obtenir la durée)
    # Ici, on simule la récupération de la durée avec un appel à une fonction fictive
get_file_duration
    duration = get_file_duration(mpsong)  # Remplacez par votre fonction réelle

    return {
        "titre": track_info["title"],
        "artiste": track_info["subtitle"],
    }
```

```
def MP3TagID(album_name, song_file):
    # Créer un objet MP3 pour modifier les tags
        audio = MP3(song_file)
        # Écrire les tags
        # Créer un objet Frame pour le titre
        title_frame = TIT2(encoding=3, text=album_name["titre"])
        audio.tags.add(title_frame)

        # Créer un objet Frame pour l'artiste
        artist_frame = TPE1(encoding=3, text=album_name["artiste"])
        audio.tags.add(artist_frame)

        # Enregistrer les modifications
        audio.save()
        print(f"Tags mis à jour pour {song_file}")

shazam = Shazam(
        http_client=HTTPClient(
            retry_options=ExponentialRetry(
                attempts=12, max_timeout=204.8, statuses={500, 502, 503, 504, 429}
            )
        )
    )

# Exemple d'utilisation
song_file = "LeonardRichter-CrimeTime.mp3"
#album_name = asyncio.run(get_album_name(song_file))
album_name = await get_track_info(shazam, mpsong)
if album_name:
    print("L'album est :", album_name)
    MP3TagID(album_name, song_file)
else:
    print("Impossible de trouver l'album.")
```

3°) ***Exercice 3 : Détermination la perte due à la compression MP3 d'un fichier WAV***

Vous prendrez un fichier wav de la qualité d'un CD audio que vous compresserez en MP3 avec un débit inférieur ou égal à 128k/s. Puis vous afficherez le spectrogramme du fichier wav, du fichier MP3, et de la différence entre le fichier wav et MP3.

Vous pourrez réaffichez le spectrogramme de différence pour plusieurs débits de MP3.

```
import librosa
import librosa.display
import matplotlib.pyplot as plt

# Chemin vers le fichier MP3
# Remplacez par le chemin vers votre fichier .mp3
file_path1 = "./LeonardRichter-CrimeTime.wav"
file_path2 = "./LeonardRichter-CrimeTime.mp3"
# Chargement du fichier audio
y1, sr1 = librosa.load(file_path1)
```

```
y2, sr2 = librosa.load(file_path2)
# Calcul du spectrogramme
spectrogram1 = librosa.feature.melspectrogram(y=y1, sr=sr1)
spectrogram2 = librosa.feature.melspectrogram(y=y2, sr=sr2)

# Calculer le spectre de différence
spectre_difference = spectrogram2 - spectrogram1

# Conversion en décibels
spectrogram_db1 = librosa.power_to_db(spectrogram1, ref=np.max)
spectrogram_db2 = librosa.power_to_db(spectrogram2, ref=np.max)
spectrogram_db3 = librosa.power_to_db(spectre_difference, ref=np.max)
# Affichage du spectrogramme
plt.figure(figsize=(10, 6))
librosa.display.specshow(spectrogram_db1, sr=sr1, x_axis='time', y_axis='mel')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogramme du fichier WAV')
plt.xlabel('Temps (secondes)')
plt.ylabel('Fréquence (Hz)')
plt.tight_layout()
plt.show()
# Affichage du spectrogramme 2
plt.figure(figsize=(10, 6))
librosa.display.specshow(spectrogram_db2, sr=sr2, x_axis='time', y_axis='mel')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogramme du fichier MP3')
plt.xlabel('Temps (secondes)')
plt.ylabel('Fréquence (Hz)')
plt.tight_layout()
plt.show()
# Affichage du spectrogramme 2
plt.figure(figsize=(10, 6))
librosa.display.specshow(spectrogram_db3, sr=sr1, x_axis='time', y_axis='mel')
plt.colorbar(format='%+2.0f dB')
plt.title('Différence de Spectre')
plt.xlabel('Temps (secondes)')
plt.ylabel('Fréquence (Hz)')
plt.tight_layout()
plt.show()
```

D. Son Multicanal

1°) Exercice 1 :

Ecrire un programme python qui crée un fichier .wav, qui contient un son multicanal (7.1) et qui le joue sur le PC.

```
from pydub.generators import Sine
from pydub import AudioSegment
import numpy as np
from scipy.io import wavfile
import soundfile as sf
import pyaudio
```

```
# Paramètres du son
duree_son = 5  # Durée du son en secondes
frequence = 440  # Fréquence du son en Hz
amplitude = 0.5  # Amplitude du son (valeur entre 0 et 1)

# Création du son en multicanal 7.1
channels = 8  # Nombre de canaux audio pour le son multicanal 7.1
# Nombre d'échantillons pour la durée spécifiée
samples = int(duree_son * 1000)
# Créer un tableau de temps pour les échantillons
temps = np.linspace(0, duree_son, samples, False)
# Initialiser un tableau vide pour les canaux
son_multicanal = np.zeros((samples, channels))

# Générer le son sinusoïdal pour chaque canal
for channel in range(channels):
    son_multicanal[:, channel] = amplitude * \
        np.sin(2 * np.pi * frequence * temps)

# Convertir les échantillons en entiers 16 bits (pour le format WAV)
son_multicanal = np.int16(son_multicanal * 32767)

# Enregistrer le son en format WAV
filename = "son_multicanal_7.1.wav"
wavfile.write(filename, int(samples / duree_son), son_multicanal)

print(f"Le son multicanal 7.1 a été enregistré dans le fichier : {filename}")

# Chemin vers le fichier WAV multicanal 7.1
# Remplacez par le chemin vers votre fichier WAV
file_path = "son_multicanal_7.1.wav"

# Lecture du fichier WAV multicanal 7.1
data, sample_rate = sf.read(file_path)

# Vérification du nombre de canaux (doit être 8 pour le multicanal 7.1)
num_channels = data.shape[1]
if num_channels != 8:
    raise ValueError("Le fichier WAV doit être en multicanal 7.1 (8 canaux).")
else :
    print ("Le fichier WAV est en multicanal 7.1 (8 canaux).")

# Lecture du fichier audio en multicanal 7.1 avec pyaudio
p = pyaudio.PyAudio()
stream = p.open(format=pyaudio.paInt16,
                channels=num_channels,
                rate=sample_rate,
                output=True)

# Jouer le fichier audio
stream.write(data.tobytes())
```

```
# Arrêter la lecture et fermer le flux audio
stream.stop_stream()
stream.close()
p.terminate()
```

2°) ***Exercice 2 :***

Créer un programme python qui prend 8 fichiers wav : flute, clarinette, piano, guitare, trompette, violon, harpe et qui les diffuse individuellement sur chacun des 7 canaux du 7.1 et le fichier batterie sur le canal LFE .1

- *https://universal-soundbank.com/*

```
from pydub import AudioSegment

# Chemins vers les fichiers audio
flute_file = "./Fichiers/flute.wav"
clarinette_file = "./Fichiers/clarinette.wav"
piano_file = "./Fichiers/piano.wav"
guitare_file = "./Fichiers/guitare.wav"
trompette_file = "./Fichiers/trompette.wav"
violon_file = "./Fichiers/violon.wav"
harpe_file = "./Fichiers/Harpe.wav"
batterie_file = "./Fichiers/basse.wav"

# Charger les fichiers audio
flute = AudioSegment.from_wav(flute_file)
clarinette = AudioSegment.from_wav(clarinette_file)
piano = AudioSegment.from_wav(piano_file)
guitare = AudioSegment.from_wav(guitare_file)
trompette = AudioSegment.from_wav(trompette_file)
violon = AudioSegment.from_wav(violon_file)
harpe = AudioSegment.from_wav(harpe_file)
batterie = AudioSegment.from_wav(batterie_file)

# Fréquence d'échantillonnage cible
frequence_cible = 44100

# Ajuster la longueur de tous les fichiers audio à la longueur maximale
longueur_max = max(len(flute), len(clarinette), len(piano), len(
    guitare), len(trompette), len(violon), len(harpe), len(batterie))

flute = flute[:longueur_max].set_frame_rate(frequence_cible)
clarinette = clarinette[:longueur_max].set_frame_rate(frequence_cible)
piano = piano[:longueur_max].set_frame_rate(frequence_cible)
guitare = guitare[:longueur_max].set_frame_rate(frequence_cible)
trompette = trompette[:longueur_max].set_frame_rate(frequence_cible)
violon = violon[:longueur_max].set_frame_rate(frequence_cible)
harpe = harpe[:longueur_max].set_frame_rate(frequence_cible)
batterie = batterie[:longueur_max].set_frame_rate(frequence_cible)

# Diffuser les fichiers individuellement sur les canaux du 7.1
channels = 8  # Nombre de canaux audio pour le son multicanal 7.1
```

```
# Combiner les canaux pour former le son multicanal 7.1
son_multicanal = flute.overlay(clarinette, position=0)
son_multicanal = son_multicanal.overlay(piano, position=1)
son_multicanal = son_multicanal.overlay(guitare, position=2)
son_multicanal = son_multicanal.overlay(trompette, position=3)
son_multicanal = son_multicanal.overlay(violon, position=4)
son_multicanal = son_multicanal.overlay(harpe, position=5)
son_multicanal = son_multicanal.overlay(batterie, position=6)

# Enregistrer le son en format WAV multicanal 7.1
son_multicanal.export("son_multicanal_7.1.wav", format="wav")

print("Le son multicanal 7.1 a été enregistré dans le fichier :
son_multicanal_7.1.wav")
```

E. Analyse :

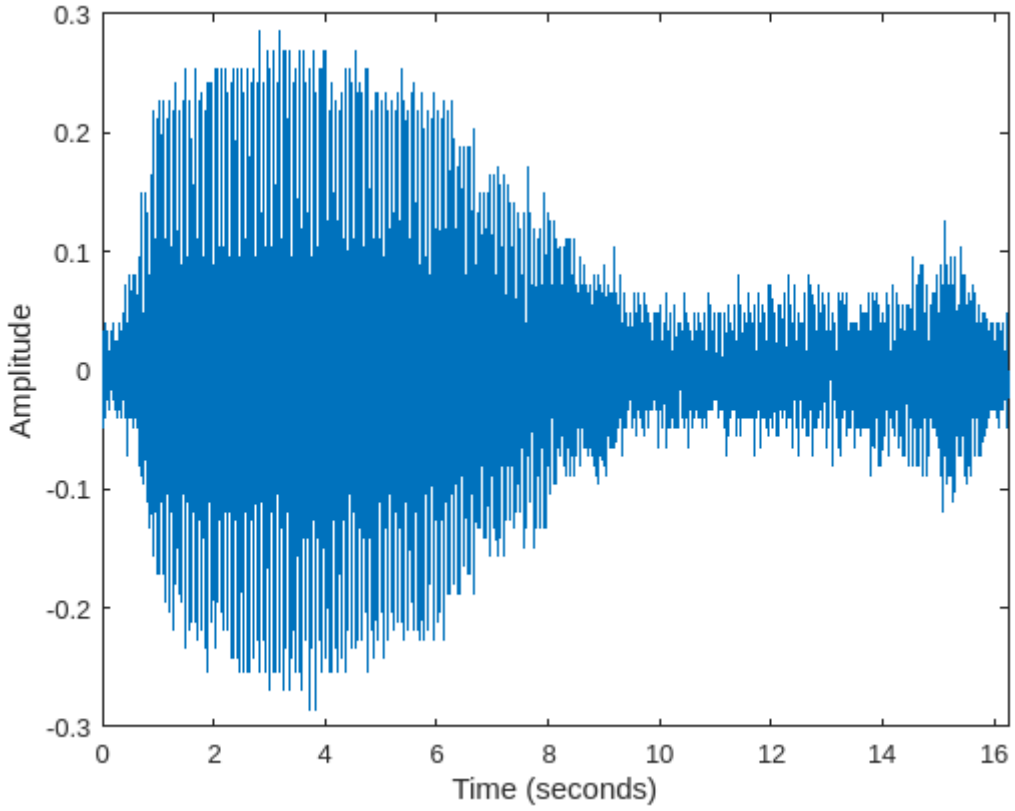
Objectifs

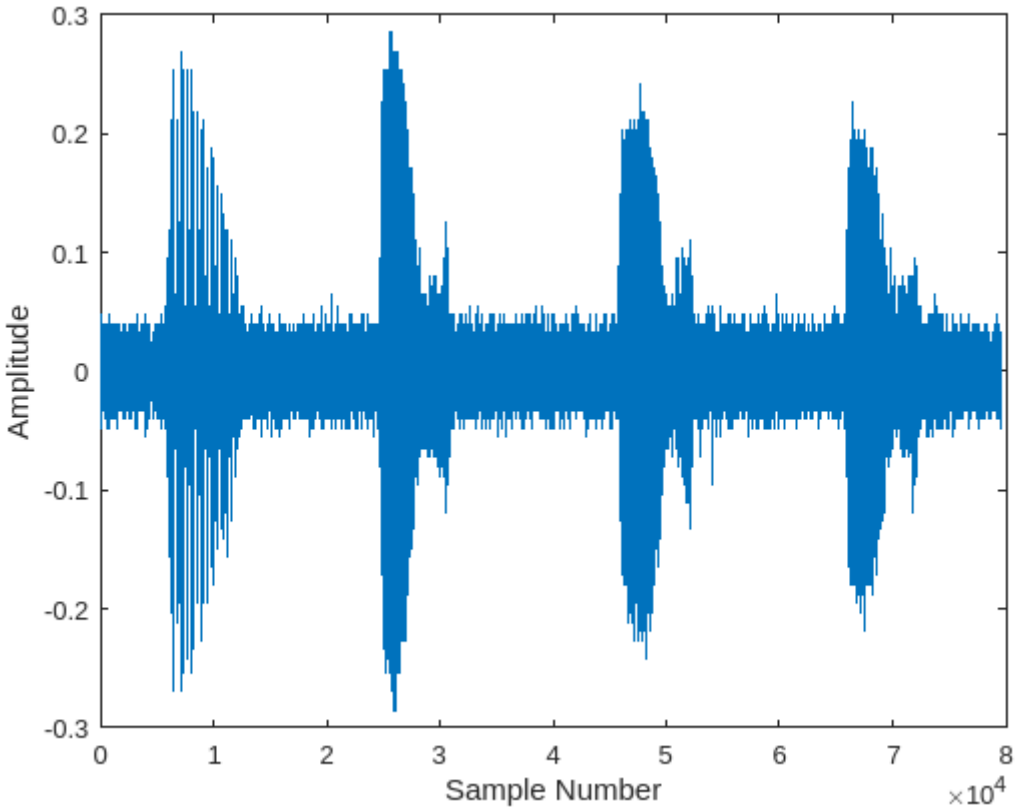
- ❖ Analyser le chant d'une baleine bleue dans le domaine spectral
- ❖ Analyser le chant d'une cigale dans le domaine spectral
- ❖ Déterminer un numéro de téléphone par analyse spectrale
- ❖ Supprimer par filtrage un harmonique parasite dans un morceau de musique

1°) Analyse spectrale du chant d'une baleine bleue :

Ecrire un programme python qui :

- ➢ Charge l'enregistrement du chant d'une baleine bleue « bluewhalesong.wav » :
- ➢ Permet d'écouter le chant de la baleine.
- ➢ Affiche la fréquence d'échantillonnage.
- ➢ Visualisez les 20 secondes enregistrées
- ➢ Affichez le spectre complet du chant de baleine.





On rappelle que la bande de fréquences audibles par l'oreille humaine s'étale de 20 Hz à 20 kHz. Lorsque la fréquence est faible, le son est grave (de 20 à 200 Hz). On parle de son médium pour une fréquence comprise entre 200 et 1000 Hz et de son aigu lorsque la fréquence est comprise entre 1000 et 15000 Hz.

- Relevez la plage de fréquences et en déduire le type de son produit par le chant d'une baleine.

Il est possible de restreindre l'analyse d'un signal sur une durée d'observation donnée.

- Réalisez l'analyse du signal sur la fenêtre de 0 à 5s permettant de se focaliser sur le premier morceau du chant. Observez les harmoniques principales de ce premier son émis par la baleine.
- Faites glisser la fenêtre pour observer les caractéristiques des 3 autres sons émis par la baleine.

```
from playsound import playsound
import matplotlib.pyplot as plt
import numpy as np
from scipy.io import wavfile
from scipy.signal import spectrogram

filename = "Audios/bluewhalesong.wav"

# Charger l'enregistrement de la baleine
sampling_rate, audio = wavfile.read(filename)

# Ecouter le chant de la baleine
# playsound(filename)

# Relever la fréquence d'échantillonnage
print("Fréquence d'échantillonnage :", sampling_rate, "Hz")

# Visualiser les 20 premières secondes du signal
time = np.arange(0, len(audio)) / sampling_rate
plt.figure(figsize=(15, 4))
plt.plot(time, audio)
plt.title("Signal audio")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.xlim(0, 20)
plt.show()

# Afficher le spectre du signal
freq, time, spec = spectrogram(audio, fs=sampling_rate)
plt.figure(figsize=(15, 4))
plt.pcolormesh(time, freq, np.log(spec))
plt.title("Spectre du signal audio")
plt.xlabel("Temps (s)")
plt.ylabel("Fréquence (Hz)")
plt.show()

# Relever la plage de fréquences du chant de baleine
print("Plage de fréquences :", freq.min(), "-", freq.max(), "Hz")
print("Type de son produit : chant")

# Analyse du premier son émis par la baleine (fenêtre de 1 à 3.5s)
start_time = 1
end_time = 3.5
start_index = int(start_time * sampling_rate)
```

```
end_index = int(end_time * sampling_rate)
window = audio[start_index:end_index]
time_window = np.arange(0, len(window)) / sampling_rate

# Afficher le spectre de la fenêtre
freq, time, spec = spectrogram(window, fs=sampling_rate)
plt.figure(figsize=(15, 4))
plt.pcolormesh(time, freq, np.log(spec))
plt.title("Spectre du premier son émis par la baleine")
plt.xlabel("Temps (s)")
plt.ylabel("Fréquence (Hz)")
plt.show()

# Trouver les harmoniques principales du premier son émis par la baleine
freqs, amps = np.fft.fftfreq(len(window)) * \
    sampling_rate, np.abs(np.fft.fft(window))
mask = (freqs >= 0) & (freqs < sampling_rate / 2)
freqs, amps = freqs[mask], amps[mask]
plt.figure(figsize=(15, 4))
plt.plot(freqs, amps)
plt.title("Transformée de Fourier du premier son émis par la baleine")
plt.xlabel("Fréquence (Hz)")
plt.ylabel("Amplitude")
plt.xlim(0, 500)
plt.show()

# Analyse des 3 autres sons émis par la baleine (fenêtre de 5 à 20s)
start_time = 5
end_time = 20
start_index = int(start_time * sampling_rate)
end_index = int(end_time * sampling_rate)
window = audio[start_index:end_index]
time_window = np.arange(0, len(window)) / sampling_rate

# Afficher le spectre de la fenêtre
freq, time, spec = spectrogram(window, fs=sampling_rate)
plt.figure(figsize=(15, 4))
plt.pcolormesh(time, freq, np.log(spec))
plt.title("Spectre des 3 autres sons émis par la baleine")
plt.xlabel("Fréquence (Hz)")
plt.ylabel("Amplitude")
plt.xlim(0, 1500)
plt.show()

# Trouver les harmoniques principales des 3 autres sons émis par la baleine
freqs, amps = np.fft.fftfreq(len(window)) * \
    sampling_rate, np.abs(np.fft.fft(window))
mask = (freqs >= 0) & (freqs < sampling_rate / 2)
freqs, amps = freqs[mask], amps[mask]
plt.figure(figsize=(15, 4))
plt.plot(freqs, amps)
plt.title("Transformée de Fourier des 3 autres sons émis par la baleine")
plt.xlabel("Fréquence (Hz)")
```

```
plt.ylabel("Amplitude")
plt.xlim(0, 500)
plt.show()

# Faire glisser la fenêtre pour observer les caractéristiques des 3 autres sons
for i in range(3):
    start_time = 30 + i*10
    end_time = start_time + 5
    start_index = int(start_time * sampling_rate)
    end_index = int(end_time * sampling_rate)
    signal_window = audio[start_index:end_index]
    window_fft = np.fft.fft(signal_window)
    plt.figure(figsize=(16, 4))
    plt.plot(signal_window, np.abs(window_fft[mask]))
    plt.xlabel('Fréquence (Hz)')
    plt.ylabel('Amplitude')
    plt.title('Harmoniques principales de la fenêtre de {} à {}s'.format(start_time,
end_time))
    plt.show()
```

2°) ***Analyse spectrale du chant d'une cigale :***

- Chargez l'enregistrement du chant d'une cigale en saisissant la commande :
- Relevez la fréquence d'échantillonnage.
- Visualisez les 2 secondes enregistrées et affichez le spectre du chant de la cigale.
- Relevez la plage de fréquences et en déduire le type de son produit par le chant d'une cigale.
- Réalisez l'analyse du signal sur la fenêtre de 0 à 0.5s.
- Faites glisser la fenêtre pour observer les caractéristiques des 3 autres sons émis par la cigale et relevez le caractère périodique du son.

3°) ***Détermination d'un numéro de téléphone par analyse spectrale***

Un code DTMF (dual-tone multi-frequency) ou FV (Fréquences Vocales) est une combinaison de fréquences utilisée pour la téléphonie filaire traditionnelle utilisée sur le Réseau téléphonique commuté (RTC) (sauf voix sur IP). Ces codes sont émis lors de l'appui sur une touche du clavier téléphonique, et sont utilisés pour la composition des numéros de téléphones et de fax.

Techniquement, chaque touche d'un téléphone correspond à un couple de deux fréquences audibles qui sont émises simultanément. On s'intéresse ici au code DTMF utilisé aux Etats-Unis qui utilise sept fréquences bien distinctes qui permettent de coder les douze touches du clavier téléphonique représenté sur la figure 3.1. Ces sept fréquences sont visibles sur le bord gauche et sur le bas de la figure 3.1.

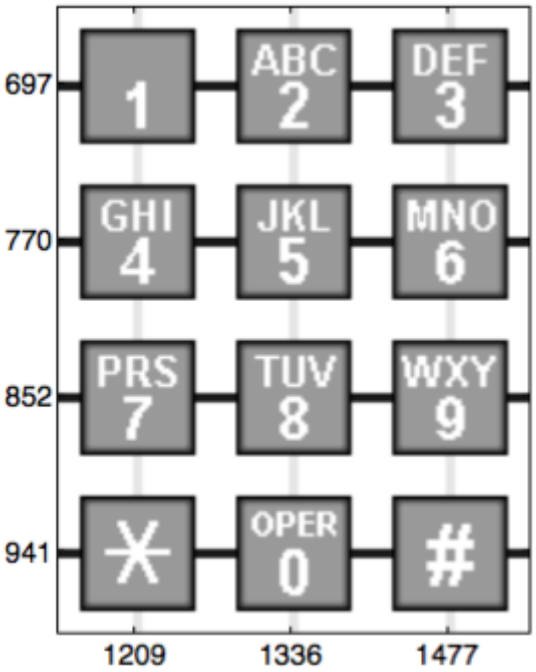


Figure 3.1 - Diapason

Elles sont également précisées dans le tableau ci-dessous.

697 Hz	770 Hz	852 Hz	941 Hz	1209 Hz	1336 Hz	1477 Hz

Le son généré par l'appui d'une touche résulte de la somme de deux sinusoïdes aux fréquences associées. Ainsi l'appui de la touche 1 produira le signal suivant :

$$y1 = \frac{1}{2}\left(sin(2\pi \times 697 \times t) + sin(2\pi \times 1209 \times t)\right);$$

La figure 3.2 représente le signal et son spectre d'amplitude lorsqu'on appuie sur la touche "1" du clavier. Les deux fréquences peuvent être clairement identifiées à partir du spectre.

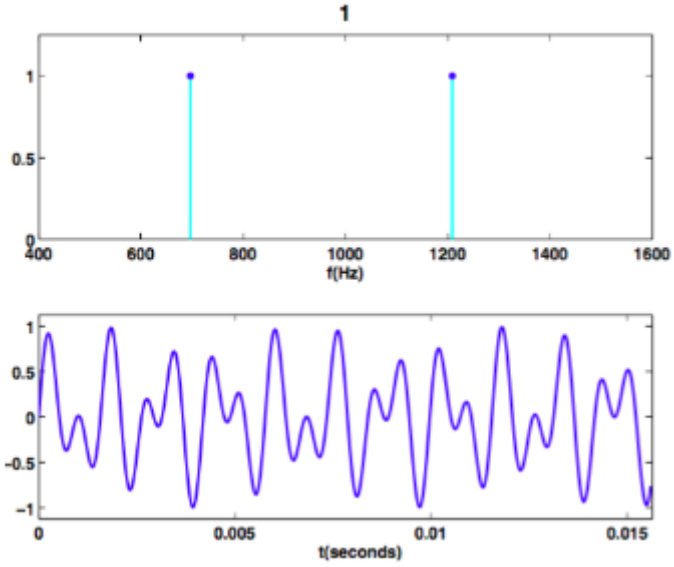


Figure 3.2 – Evolution temporelle et spectre du signal résultant de l'appui de la touche 1 du clavier

Un fichier numero_tel.mat contenant un numéro de téléphone américain est disponible sur le site web du cours.

Dans ce fichier se trouvent les vecteurs colonne y et t contenant les échantillons du numéro composé et les instants d'enregistrement ainsi que la fréquence d'échantillonnage fe.

- Créez un nouveau fichier td3_1.m.
- Chargez le numéro de téléphone en saisissant la commande :

load numero_tel;

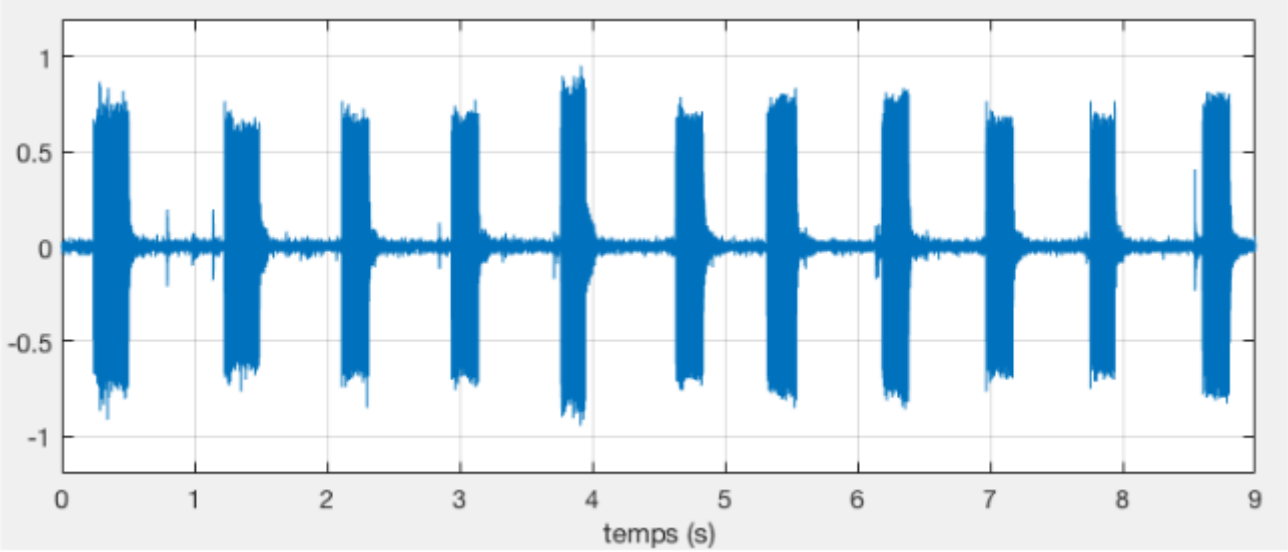


Figure 3.3 – Evolution temporelle du numéro de téléphone composé

- Tracez l'évolution temporelle du numéro de téléphone sur les 9 s d'enregistrement disponibles à l'aide de la commande suivante :

plot(t,y),shg

- Vérifiez que vous obtenez un tracé similaire à celui représenté sur la figure 3.3.
- En zoomant, observez la forme de chaque son. Que constatez-vous ?
- De combien de chiffres est composé le numéro de téléphone ?
- Est-il possible de déterminer le numéro de téléphone d'après l'évolution temporelle du signal composé ?
- Ecoutez le numéro composé en saisissant la commande :

sound(y,fe);

- On peut tracer la représentation temps-fréquence du numéro composé en saisissant la commande :

spectrogram(y,kaiser(128,18),120,128,fe,'yaxis');

La figure permet de visualiser le contenu fréquentiel du signal au cours du temps mais la précision sur l'axe des fréquences n'est pas suffisante pour identifier les deux fréquences.

- Utilisez l'outil graphique d'analyse de signaux `SignalAnalyzer` pour déterminer le numéro de téléphone.
- A l'aide d'Internet, déterminez le nom de la société à qui appartient le numéro. Trouvez son adresse aux Etats-Unis et utilisez Google Maps pour visualiser la porte d'entrée principale de l'entreprise américaine. Un indice est fourni sur la figure 3.4.

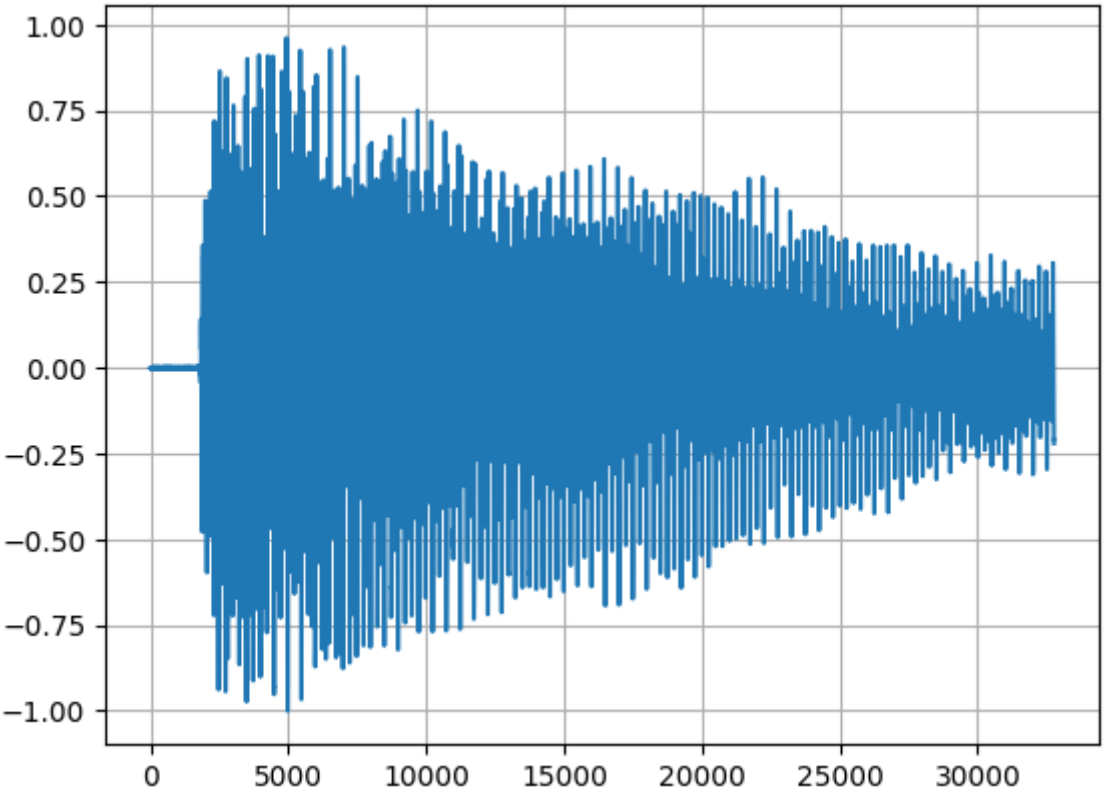
En pratique, la détection de tonalités DTMF peut être effectuée en utilisant différents algorithmes. Pour en savoir plus, parcourez l'article publié en 1996 qui présente l'analyse comparative des performances de trois algorithmes :

- Relevez le nom des trois algorithmes comparés.
- Trouvez en surfant sur Internet le code DTMF ou combinaison de fréquences utilisée pour la téléphonie filaire traditionnelle utilisé en France. Générer le numéro de l'accueil de l'IUT.
- Tracez son évolution temporelle et son spectrogramme à l'aide de l'outil graphique d'analyse de signaux de Matlab.
- Faites valider vos résultats par l'enseignant.

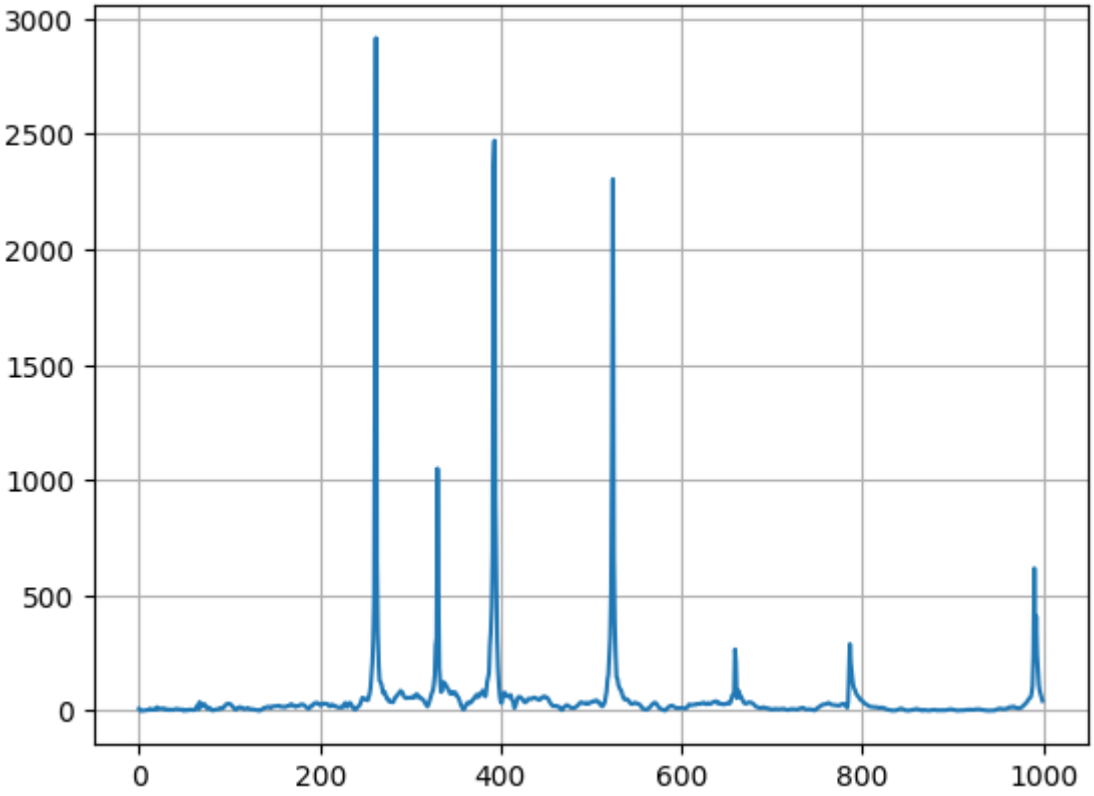
Figure 3.4 – Indice photographique pour trouver à qui appartient le numéro à identifier

4°) Traitement d'un morceau de piano

- Chargez le fichier `domisol.wav` :
- Quelle est la fréquence d'échantillonnage utilisée pour enregistrer le signal ?
- Visualiser le signal en fonction du temps :



- Observez le spectre d'amplitude du signal.



- Déterminez les 3 notes qui sont jouées.

  - Le pic le plus élevé se situe à environ 262 Hz. Il s'agit de la note DO 3.

  - Le second pic se situe à environ 392 Hz. C'est un SOL 3.

  - Nous trouvons ensuite un pic à environ 524 = 2 x 262 Hz. Il s'agit d'un DO 4, une octave au-dessus du premier DO : c'est une _harmonique_.

- Où est le MI 3 ? Cherchez la fréquence 330 Hz. Et regardez aussi ses multiples, 660 (MI 4), 990 (MI 5) !

5°) ***Traitement d'un extrait musical parasité***

Lors de l'enregistrement d'un signal musical, un signal parasite haute fréquence est venu perturber l'enregistrement. L'objectif est donc d'identifier par analyse spectrale du signal enregistré la perturbation puis de la supprimer au travers d'un filtrage afin de retrouver le signal d'origine.

- Créez un nouveau fichier `td3_2`.mlx.
- Chargez le fichier `extrait_musique.wav`, à l'aide de la fonction Matlab `audioread` :

[y,fe] = audioread('extrait_musique.wav');

- Quelle est la fréquence d'échantillonnage utilisée pour enregistrer le signal ?
- Utilisez la fonction `sound` pour écouter l'extrait musical bruité (attention au volume car cela peut s'avérer désagréable à entendre) :

sound(y,fe);

- Observez le spectre d'amplitude du signal. Vous pouvez utiliser l'outil `SignalAnalyzer` pour visualiser le spectre. Déterminez la fréquence de l'harmonique parasite.
- On propose de supprimer la perturbation par application d'un filtrage passe-bas de Butterworth en étudiant l'influence du choix de l'ordre. Matlab comporte une fonction `butter` (help `butter`) qui fournit les paramètres de la fonction de transfert du filtre à partir de la spécification de l'ordre, de la fréquence de coupure du filtre normalisée et du type de filtre (low, high,...). On pourra s'inspirer de la solution suivante en spécifiant la fréquence coupure souhaitée :

```
fc=????; % fréquence de coupure du filtre
fcn=fc/(fe/2); % fréquence de coupure du filtre normalisée
par rapport à fe/2
n=1; % ordre du filtre
[b,a] = butter(n,fcn,'low'); % Calcul du numérateur et du
dénominateur du filtre
freqz(b,a,512,fe) % tracé de la réponse fréquentielle du
filtre
yf=filter(b,a,y); % produit de convolution du filtre et du
signal bruité
sound(yf,fe); % pour écouter l'extrait filtré
```

- Renouvelez la procédure en augmentant l'ordre du filtre jusqu'à déterminer l'ordre qui permet de supprimer totalement l'harmonique perturbateur et ainsi écouter le signal musical d'origine.

Matlab possède la fonction `buttord` qui fournit l'ordre du filtre de Butterworth qui permet de respecter l'atténuation en dB souhaitée. Utilisez cette fonction pour déterminer l'ordre minimal qui permet de ne plus entendre la perturbation dans l'extrait musical.

- Faites valider votre filtrage par l'enseignant.

F. Pour aller plus loin :

G. 8.2.3. Lab week 3: Fourier series, transformation and discrete signals

1°) 8.2.3.1. Assignment 1.

Sympy est une belle bibliothèque pour la manipulation de symboles en Python et a une certaine ressemblance avec Mathematica. Malheureusement, il contient quelques erreurs, mais l'affectation suivante est possible.

```
from sympy.plotting import plot
from sympy import symbols

x = Symbols('x')
plot(x, x**2, x**3, (x, -5, 5))
from sympy import *
x = Symbol('x')
solve(x**2 - 3, x)
```

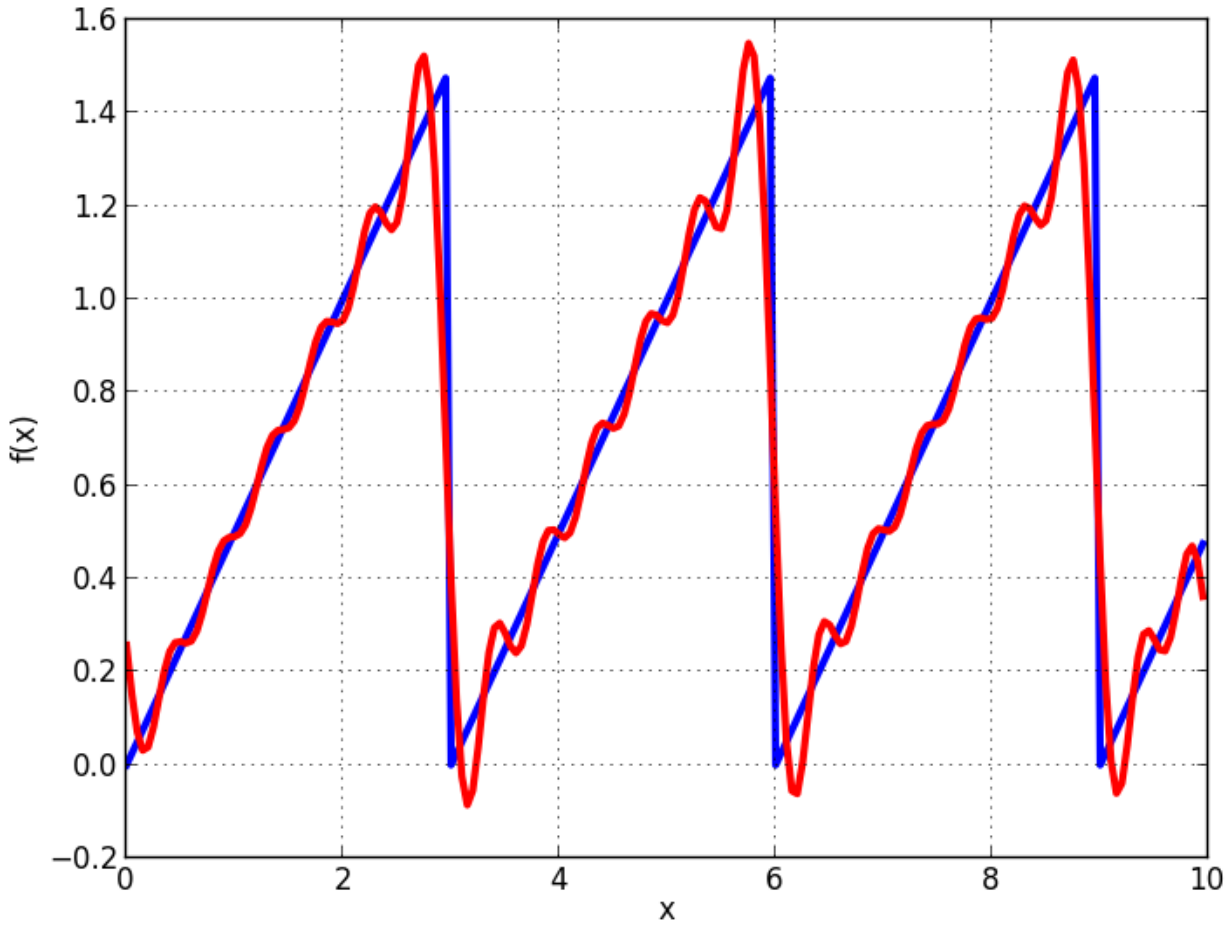
Veuillez noter qu'il y a une différence entre sympy.symbols et sympy. Symbole

Résolvez les problèmes suivants avec sympy:

$$\int_{-\infty}^{+\infty} sin(x2)dx$$

2°) 8.2.3.2. Assignment 2.

Et maintenant un travail un peu plus sérieux avec la série de Fourier. Dans les systèmes de contrôle, il est intéressant de voir comment un système se comporte lorsque nous appliquons un certain signal d'entrée. Les signaux d'entrée les plus utilisés sont le pas (Heaviside), la rampe (en dents de scie) et l'impulsion (dirac, celui-ci peut entraîner des problèmes de sympie). Déterminez la réponse de Fourier pour ces 3 signaux en utilisant la sympie et tracez quelques termes de la série pour un court intervalle. Tracer le signal d'entrée ET de sortie. Le résultat devrait ressembler à :



3°) 8.2.3.3. Assignment 3.

Écrivez une fonction Python qui crée un sinus de 100 Hz et retourne les exemples avec minutage. Recréez le signal du domaine temporel à l'aide d'un Sinc et tracez les deux fonctions. Expérimentez avec différents échantillonssarate au-dessus et au-dessous de la fréquence de Shannon et montrez ce qui se passe.

4°) 8.2.3.4. Assigment 4.

Faites de même pour la fonction suivante :

$$x(t)=sin(\omega t)+5sin(3\omega t)+3cos(7\omega t+10)$$

Montrer ce qui se passe avec différentes fréquences d'échantillonnage.

5°) 8.2.3.5. Assignment 5.

Le fichier suivant muziek_48kHz.wav contient un court morceau de musique au format wav. En utilisant la bibliothèque 'wave' de Python, vous pouvez extraire plusieurs champs intéressants de ce fichier:

```
wav_header_dtype = np.dtype([
 ("chunk_id", (str, 4)),
 ("chunk_size", "<u4"),
 ("format", "S4"),          # 4-byte string
 ("fmt_id", "S4"),
 ("fmt_size", "<u4"),
 ("audio_fmt", "<u2"),
 ("num_channels", "<u2"),
 ("sample_rate", "<u4"),
```

```
("byte_rate", "<u4"),
("block_align", "<u2"),
("bits_per_sample", "<u2"),
("data_id", ("S1", (2, 2))),
("data_size", "u4"),
])
```

Chargez la musique en utilisant des 'readframes' et effectuez un 'downsampling' en utilisant seulement 1 échantillon de chaque 6 échantillons et créez un nouveau fichier wav. Lisez le fichier wav. Pas très agréable, comme vous le remarquerez. Que s'est-il passé? La bonne façon est d'utiliser une fonction de rééchantillonnage. Sous Linux, la voie facile consiste à utiliser le programme 'sox'. Faites un rééchantillonnage à 8 kHz et comparez le fichier avec l'original.

6°) 8.2.3.6. Assignment 6.

Téléchargez le fichier suivant contenant les données de mesure EEG : :d ownload: données EEG Vous verrez que c'est assez bruyant. Le fichier contient des échantillons de 16 échantillons et a été échantillonné avec 500 Hz. Nous ne nous intéressons qu'au canal 1. Tracez ce canal et veillez à avoir des valeurs correctes sur l'axe. Nous pouvons filtrer ce signal en utilisant la convolution de la bibliothèque numpy en utilisant une fenêtre de 'ones':

```
fenêtre = np. ones(int(window_size))/float(window_size)
```

Tracez le signal filtré avec une couleur différente et expérimentez avec la taille de la fenêtre. Commencez par 10 par exemple.

H. 8.2.4. Lab week4. Implementation of Shazam music recognition

Shazam est une société avec des droits de brevet sur un algorithme pour identifier la musique. L'idée principale est de créer des empreintes acoustiques basées sur des spectrogrammes et de les stocker dans une base de données. Lorsque la base de données a été remplie, une nouvelle empreinte audio permet de rechercher des correspondances dans la base de données. La publication de « code shazam » est souvent interdite par l'entreprise avec des fils juridiques. Vous êtes donc prévenus.

La mission consiste à implémenter une implémentation commentée de l'algorithme Shazam. Lisez pour cela les articles suivants :

```
Wang, An Industrial-Strength Audio Search Algorithm

Guo, A Music Retrieval System Using Melody and Lyric

Yao, Digital Song Analysis Using Frequency Analysis
```

Pour ceux qui veulent améliorer plusieurs parties de leur programme, le lien suivant peut aider à trouver des idées: http://www.ismir.net/proceedings

Il est recommandé d'utiliser d'abord des chansons musicales « normales » avec par exemple 2 versions ou une partie différente de la même chanson. Expérimentez plus tard avec des versions plus difficiles et étendez votre base de données avec des empreintes digitales. Tous les exemples doivent présenter un spectrogramme de la musique et des informations d'empreintes digitales facultatives (graphiques ou textuelles). Prenez soin des informations correctes sur les axes. Plus difficile est la reconnaissance de sons qui ne sont pas de la musique. Pour ceux qui ont mis en œuvre la reconnaissance musicale, nous avons différents sons d'oiseaux qui se trouvent être assez difficiles à stocker parfois comme empreintes digitales. Il serait intéressant que certains d'entre vous puissent montrer si la reconnaissance vocale est également possible en utilisant cette méthode. L'idée est que vous donniez une courte présentation de vos résultats.

I. 8.2.5. Lab week 5. Digital filters

1°) 8.2.5.1. Assignment 1

L'ordre d'un filtre est souvent déterminé par la fréquence de coupure et une atténuation requise en dB à une certaine fréquence. Supposons que nous voulions utiliser un filtre Butterworth avec la caractéristique suivante:

$$H(\omega)=G_{20}/1+(\omega\omega_c)_{2n}$$

La fréquence de coupure est de 250 Hz et nous voulons une atténuation de -30 dB à 500 Hz. Quel est l'ordre du filtre ? Donnez le résultat de votre calcul dans un document pdf. Affichez les réponses d'amplitude de ce filtre et vérifiez si le résultat calculé est correct. Vous pouvez utiliser la fonction beurre de scipy pour cela, mais notez que cette fonction utilise une fréquence normalisée (comme dans l'équation ci-dessus). Assurez-vous que l'axe est correct et vérifiez la pente du filtre. Quelle est la pente en dB/octave (rappelez-vous que pour un doublement de fréquence la sortie du filtre tombe de 6 dB pour un filtre de premier ordre)

2°) 8.2.5.2. Assignment 2

La fonction scipy est une implémentation IIR. Créez une implémentation FIR avec les mêmes caractéristiques. Vérifiez le nombre de « taps » dont vous avez besoin (les taps sont les blocs typiques contenant les paramètres b entre les délais de transformation z avec additionneurs). Créez une réponse d'amplitude (prenez soin du type d'axe correct, etc. et d'un tracé pôle-zéro)

3°) 8.2.5.3. Assignment 3

Un « problème » des filtres FIR est la fenêtre que nous utilisions normalement. La réponse impulsionnelle avec l'utilisation d'une fenêtre rectangulaire conduit à un dépassement et à une ondulation de la réponse en fréquence, ce qui peut être problématique (c'est ce qu'on appelle le « phénomène de Gibb »): le dépassement de la réponse d'amplitude ne diminue pas si nous commençons à utiliser plus de robinets. Nous utilisons donc souvent une fenêtre différente : le hamming est courant, mais il y en a plus. Affichez la différence entre les deux fenêtres d'un tracé.

4°) 8.2.5.4. Assignment 4

Créez un filtre IIR du type Chebyshev « 1 » avec à nouveau les mêmes caractéristiques. Voyez-vous la différence entre la première et la deuxième implémentation ? Vérifiez l'ordre du filtre que vous utilisez. Créez à nouveau un tracé d'amplitude et de pôle zéro.

5°) 8.2.5.5. Assignment 5

Vous avez vu lorsque vous utilisez le filtre de type Tchebychev que ce type de filtre a une ondulation sur la réponse d'amplitude, mais une caractéristique typique est que ce filtre a une pente très raide pour l'atténuation. Lorsque ce filtre est utilisé, on veut normalement avoir un certain contrôle sur l'ondulation sur la bande passante. Expérimentez avec la fonction scipy.signal.cheb1ord qui peut être utilisée pour contrôler l'ondulation avec différents ordres en utilisant cette caractéristique de filtre.

6°) 8.2.5.6. Assignment 6

Analysez le comportement du filtre suivant :

$$y[n]=1.85y[n-1]-0.95y[n-2]+x[n]-1.9x[n-1]+x[n-2]$$

Voir aussi le livre de recettes de filtre de Rein en ipython:

7°) 8.2.5.7. Assignment 7

Faites de même pour ce filtre notch :

$$y[n]=1.5y[n-1]-0.85y[n-2]+x[n]$$

J. Webographie

- *Documentation sur le module wave*
- *WAVEform audio file format*
- *PCM (Pulse Code Modulation)*
- *Site officiel du logiciel libre et gratuit Audacity*
- *Lecteur multimédia*
- *Editeur hexadécimal*
- *Code DTMF*
- *Module pygame.mixer (documentation)*
- ---
- *http://fsincere.free.fr/isn/python/download/audio/dtmf.wav]*
- *http://fsincere.free.fr/isn/python/download/audio/balla.wav]*
- *http://fsincere.free.fr/isn/python/download/audio/resultat.wav]*
- *http://nsinfo.yo.fr/cours_python/python_15_son.html*
- *http://nsinfo.yo.fr/cours_python/python_10_images.html*
- *http://pcpagnol.free.fr/CRIADO/acoustique/gamme/note.htm*
- *https://phet.colorado.edu/sims/html/waves-intro/latest/waves-intro_fr.html*
- *https://f2school.com/wp-content/uploads/2019/10/Traitement-du-signal-exercices-03.pdf*
- *https://dept-info.labri.fr/~hanna/ImageSon/*
- *https://www.cimat.mx/~moralesjh/html/ballena1.html*
- *Virtual Lab: Sound Analysis Principles – K. Lisa Yang Center for Conservation Bioacoustics (cornell.edu)*
- *https://mpsib-camille-guerin.pagesperso-orange.fr/Python/FFT/FFT.pdf*
- *https://perso.esiee.fr/~bercherj/Python/DistribsTP/distrib_ri/Corrig%C3%A9_RI_TF_Filtrage.html*
- *https://www-fourier.ujf-grenoble.fr/~faure/enseignement/musique/cours_acoustique_musicale.pdf*
- *https://jckantor.github.io/CBE30338/05.03-Creating-Bode-Plots.html*
- *https://staff.fnwi.uva.nl/r.vandenboomgaard/SP20162017/Python/Audio/rta_signaldisplay.html*
- *https://staff.fnwi.uva.nl/r.vandenboomgaard/SP20162017/20162017/LabExercises/wk3_fourier_transform.html*

- *https://music-classification.github.io/tutorial/part3_supervised/data-augmentation.html*
- *https://universal-soundbank.com/cymbales5.htm*
- *https://rapidapi.com/blog/shazam-api-java-python-php-ruby-javascript-examples/*
-