Traitement de données audio en Python

Analyses avec le package librosa

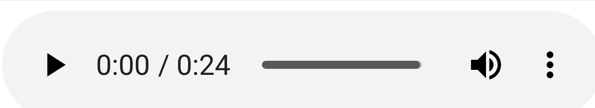
librosa est une bibliothèque d’analyse de données audio et plus particulièrement de musique. librosa repose sur `numpy` et `scipy`.

Lire un fichier audio

Pour lire un fichier audio avec `librosa` on fait appel à la classe `Audio` du module `display` de IPython

In [1]:
```
import librosa
import numpy as np
from IPython.display import Audio

# la fonction load (https://librosa.org/doc/latest/generated/librosa.load.ht
# avec deux variables :
# - y: les données
# - sr: sampling rate (taux d’échantillonnage)
y, sr = librosa.load("../audio/electric_cello.wav")
#y, sr = librosa.load("../audio/choir_chant.wav")
#y, sr = librosa.load(librosa.ex('choice'))
#y, sr = librosa.load(librosa.ex('vibeace'))
#y, sr = librosa.load(librosa.ex('fishin'))

Audio(data=y, rate=sr)
```

Out[1]:



Les données (les *samples*) sont des objets de type `ndarray` de numpy.
Ici un tableau de float à une dimension

In [2]:
```
print(f"Type de y : {type(y)}")
print(f"Type du 1er élément du tableau : {type(y[0])}")
```

```
Type de y : <class 'numpy.ndarray'>
Type du 1er élément du tableau : <class 'numpy.float32'>
```

Visualisations

Les visualisations se font à l’aide de la librairie `matplotlib`

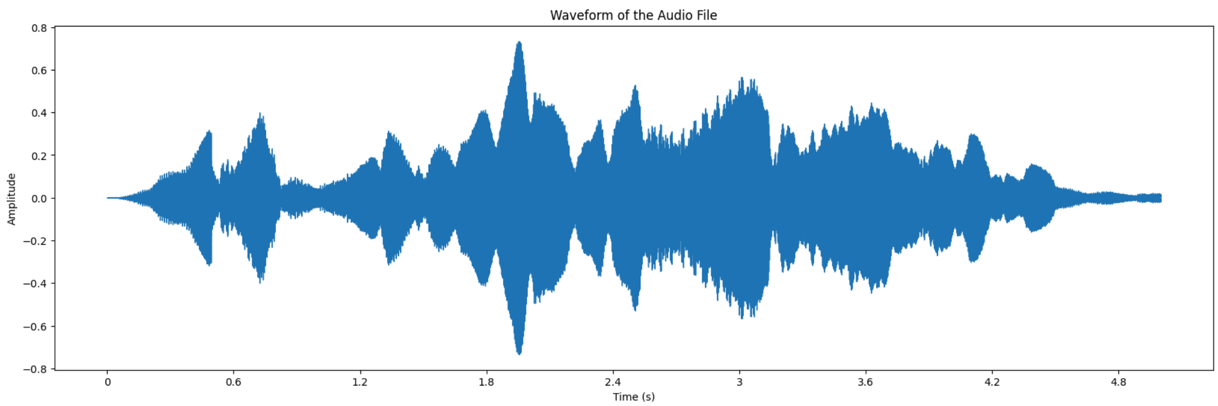
waveform

In [26]:
```
!pip install librosa --upgrade
```

```
Requirement already satisfied: librosa in /home/plancq/pro/dev/venvs/image/l
ib/python3.10/site-packages (0.10.2.post1)
Requirement already satisfied: audioread>=2.1.9 in /home/plancq/pro/dev/venv
s/image/lib/python3.10/site-packages (from librosa) (3.0.1)
Requirement already satisfied: numpy!=1.22.0,!=1.22.1,!=1.22.2,>=1.20.3 in /
home/plancq/pro/dev/venvs/image/lib/python3.10/site-packages (from librosa)
(1.25.2)
Requirement already satisfied: scipy>=1.2.0 in /home/plancq/pro/dev/venvs/im
age/lib/python3.10/site-packages (from librosa) (1.11.3)
Requirement already satisfied: scikit-learn>=0.20.0 in /home/plancq/pro/dev/
venvs/image/lib/python3.10/site-packages (from librosa) (1.3.1)
Requirement already satisfied: joblib>=0.14 in /home/plancq/pro/dev/venvs/im
age/lib/python3.10/site-packages (from librosa) (1.3.2)
Requirement already satisfied: decorator>=4.3.0 in /home/plancq/pro/dev/venv
s/image/lib/python3.10/site-packages (from librosa) (5.1.1)
Requirement already satisfied: numba>=0.51.0 in /home/plancq/pro/dev/venvs/i
mage/lib/python3.10/site-packages (from librosa) (0.58.0)
Requirement already satisfied: soundfile>=0.12.1 in /home/plancq/pro/dev/ven
vs/image/lib/python3.10/site-packages (from librosa) (0.12.1)
Requirement already satisfied: pooch>=1.1 in /home/plancq/pro/dev/venvs/imag
e/lib/python3.10/site-packages (from librosa) (1.7.0)
Requirement already satisfied: soxr>=0.3.2 in /home/plancq/pro/dev/venvs/ima
ge/lib/python3.10/site-packages (from librosa) (0.3.7)
Requirement already satisfied: typing-extensions>=4.1.1 in /home/plancq/pro/
dev/venvs/image/lib/python3.10/site-packages (from librosa) (4.8.0)
Requirement already satisfied: lazy-loader>=0.1 in /home/plancq/pro/dev/venv
s/image/lib/python3.10/site-packages (from librosa) (0.3)
Requirement already satisfied: msgpack>=1.0 in /home/plancq/pro/dev/venvs/im
age/lib/python3.10/site-packages (from librosa) (1.0.7)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /home/plancq/pr
o/dev/venvs/image/lib/python3.10/site-packages (from numba>=0.51.0->librosa)
(0.41.0)
Requirement already satisfied: platformdirs>=2.5.0 in /home/plancq/pro/dev/v
envs/image/lib/python3.10/site-packages (from pooch>=1.1->librosa) (3.11.0)
Requirement already satisfied: packaging>=20.0 in /home/plancq/pro/dev/venv
s/image/lib/python3.10/site-packages (from pooch>=1.1->librosa) (23.2)
Requirement already satisfied: requests>=2.19.0 in /home/plancq/pro/dev/venv
s/image/lib/python3.10/site-packages (from pooch>=1.1->librosa) (2.31.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/plancq/pro/dev/
venvs/image/lib/python3.10/site-packages (from scikit-learn>=0.20.0->libros
a) (3.2.0)
Requirement already satisfied: cffi>=1.0 in /home/plancq/pro/dev/venvs/imag
e/lib/python3.10/site-packages (from soundfile>=0.12.1->librosa) (1.16.0)
Requirement already satisfied: pycparser in /home/plancq/pro/dev/venvs/imag
e/lib/python3.10/site-packages (from cffi>=1.0->soundfile>=0.12.1->librosa)
(2.21)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/plancq/pro/
dev/venvs/image/lib/python3.10/site-packages (from requests>=2.19.0->pooch>=
1.1->librosa) (3.3.0)
Requirement already satisfied: idna<4,>=2.5 in /home/plancq/pro/dev/venvs/im
age/lib/python3.10/site-packages (from requests>=2.19.0->pooch>=1.1->libros
a) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /home/plancq/pro/dev/ve
nvs/image/lib/python3.10/site-packages (from requests>=2.19.0->pooch>=1.1->l
ibrosa) (2.0.6)
Requirement already satisfied: certifi>=2017.4.17 in /home/plancq/pro/dev/ve
```

```
nvs/image/lib/python3.10/site-packages (from requests>=2.19.0->pooch>=1.1->l
ibrosa) (2023.7.22)
```

In [27]:
```python
import matplotlib.pyplot as plt

y, sr = librosa.load("../audio/electric_cello.wav", duration=5)

plt.figure(figsize=(20, 6))
librosa.display.waveshow(y, sr=sr)
plt.title('Waveform of the Audio File')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()
```

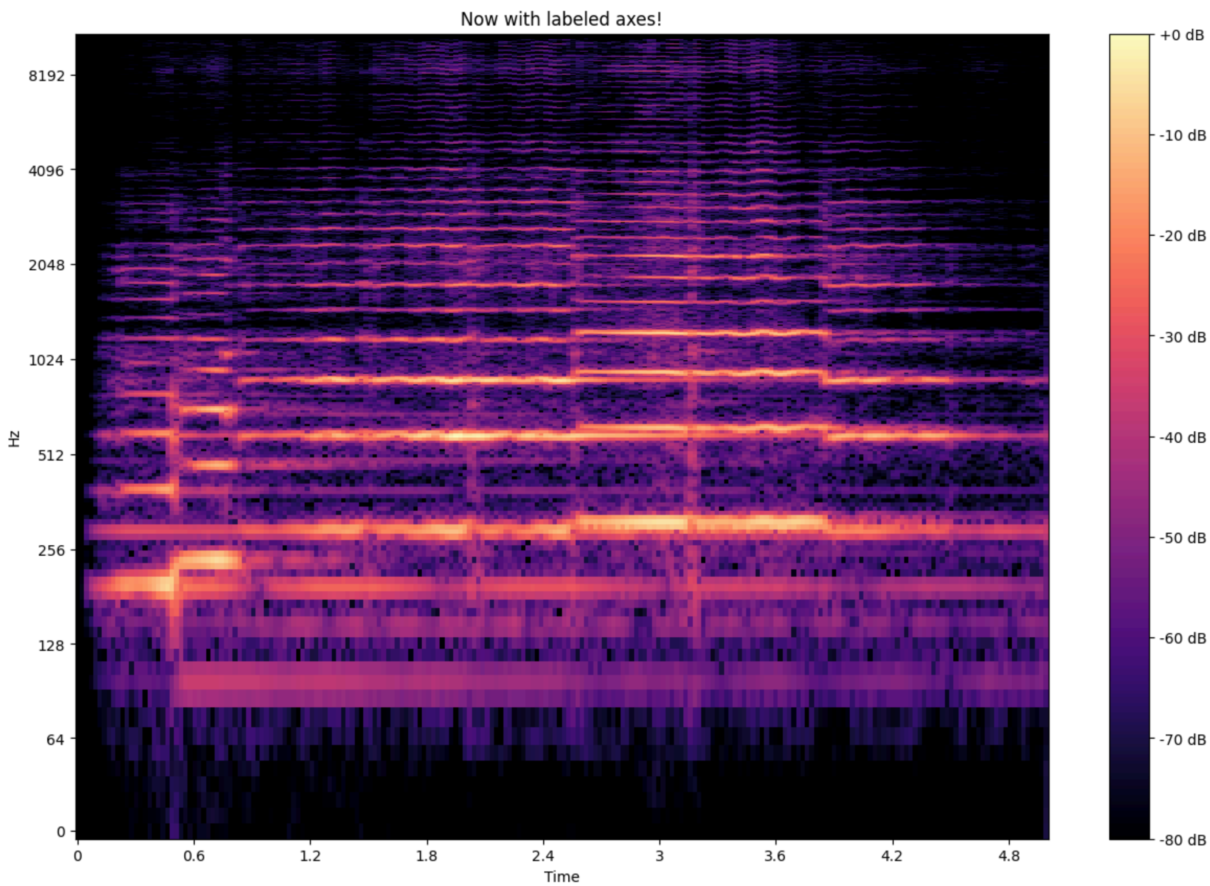


Spectrogramme

In [4]:
```python
# Génération du spectrogramme

D = librosa.stft(y)  # STFT (short-time Fourier Transform)
S_db = librosa.amplitude_to_db(np.abs(D), ref=np.max)
```

In [5]:
```python
# Visualisation

fig, ax = plt.subplots(figsize=(15,10))
img = librosa.display.specshow(S_db, x_axis='time', y_axis='log', ax=ax)
ax.set(title='Now with labeled axes!')
fig.colorbar(img, ax=ax, format="%+2.f dB")
```

Out[5]:
```
<matplotlib.colorbar.Colorbar at 0x7a7f34f95ab0>
```



Écouter à l’envers

In [6]:
```
y, sr = librosa.load("../audio/electric_cello.wav")

# Inverser le signal
y_inverted = y[::-1]

display(Audio(y_inverted, rate=sr))
```

0:00 / 0:24

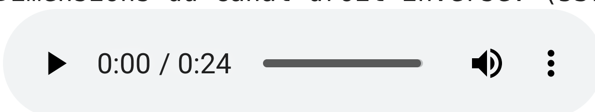
C’est un peu étrange. Encore plus étrange on peut créer un signal stéréo avec un canal à l’endroit et un canal à l’envers (à écouter au casque si possible)

In [7]:
```
# Charger l'audio stéréo
y, sr = librosa.load("../audio/electric_cello.wav", mono=False)
#y, sr = librosa.load("../audio/choir_chant.wav", mono=False)

# Séparer les canaux (gauche et droit)
y_left = y[0]  # Canal gauche (normal)
y_right = y[1]  # Canal droit (inversé)

# Vérifier les dimensions de y
print("Dimensions de y:", y.shape)  # Cela devrait être (2, n_samples) pour
```

```python
# Inverser le canal droit
y_right_inverted = y_right[::-1]

# Vérifier les dimensions des canaux séparés
print("Dimensions du canal gauche:", y_left.shape)
print("Dimensions du canal droit inversé:", y_right_inverted.shape)

stereo_signal = np.array([y_left, y_right_inverted])

# Écouter le signal stéréo dans le notebook
Audio(stereo_signal, rate=sr)
```

```
Dimensions de y: (2, 537668)
Dimensions du canal gauche: (537668,)
Dimensions du canal droit inversé: (537668,)
```

Out[7]:



Détection du tempo

C’est tout bête, il y a une fonction pour ça : librosa.feature.tempo

In [8]:

```python
y, sr = librosa.load("../audio/electric_cello.wav")
tempo = librosa.feature.tempo(y=y, sr=sr)
print(tempo)
```

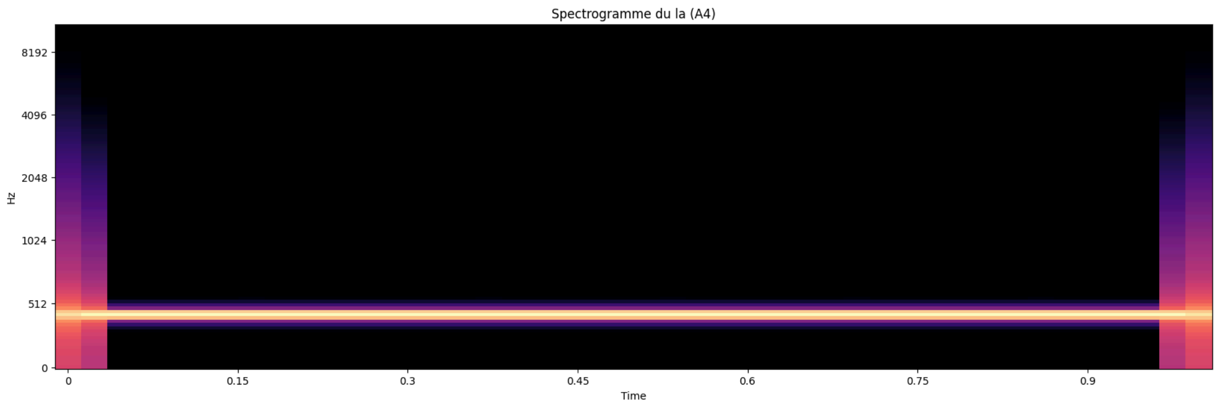
```
[117.45383523]
```

Reconnaissance des notes

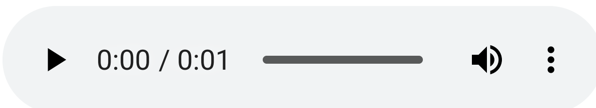
En musique chaque note a une hauteur (*pitch*) qui peut s’exprimer par une fréquence.
La note la (A4) a une fréquence de 440 Hz par exemple. Le la plus aigu d’un octave (A5) a une fréquence de 880 Hz. Il y a un rapport constant entre les différentes réalisations d’une même note.

In [9]:

```python
tone = librosa.tone(440, duration=1)
S = librosa.feature.melspectrogram(y=tone)
# Spectrogramme basé sur l’échelle des mels (https://fr.wikipedia.org/wiki/%
# Voir https://librosa.org/doc/latest/generated/librosa.feature.melspectrogr


plt.figure(figsize=(20, 6))
librosa.display.specshow(librosa.power_to_db(S, ref=np.max), x_axis='time',
plt.title('Spectrogramme du la (A4)')
plt.xlabel('Time')
plt.ylabel('Hz')
plt.show()
```



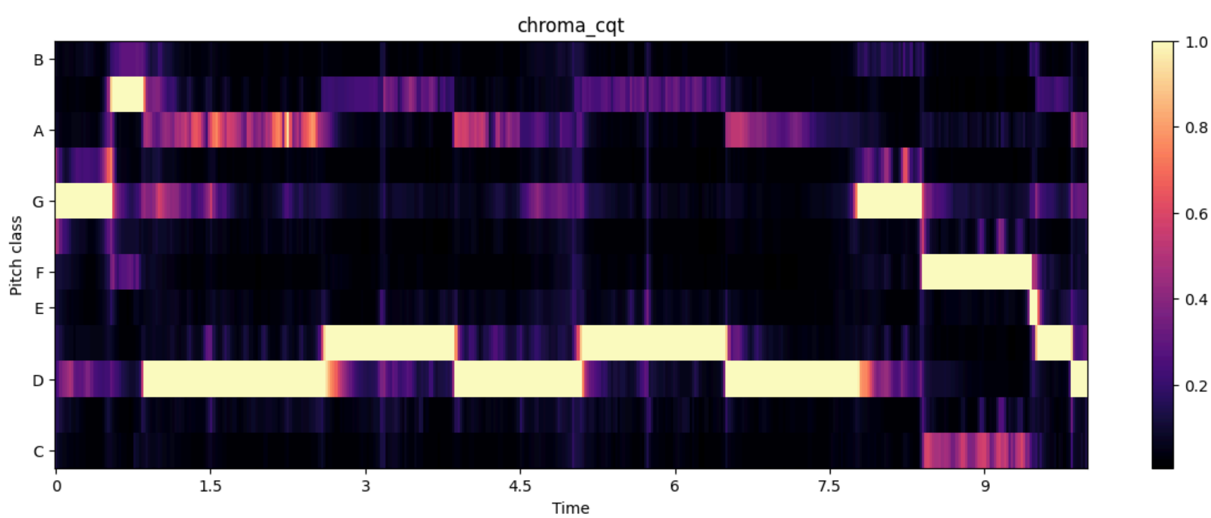
In [10]:
```
display(Audio(tone, rate=22050))
```



librosa permet une analyse chromatique d'un enregistrement. La gamme chromatique est une échelle musicale composée de douze degrés. L'analyse chromatique permet de déterminer quel est le degré d'un ensemble de *samples*.

In [11]:
```
y, sr = librosa.load("../audio/electric_cello.wav", duration=10)

chroma_cq = librosa.feature.chroma_cqt(y=y, sr=sr)

fig, ax = plt.subplots(sharex=True, figsize=(15,5))
img = librosa.display.specshow(chroma_cq, y_axis='chroma', x_axis='time', ax
ax.set(title='chroma_cqt')
fig.colorbar(img, ax=ax)
```

Out[11]: <matplotlib.colorbar.Colorbar at 0x7a7f2c43d480>



In [12]:
```
Audio(data=y, rate=sr)
```

Out[12]:

0:00 / 0:10

Séparation composants harmoniques et percussifs

À partir du spectrogramme librosa peut séparer harmonie et percussions.

In [13]:

```
y, sr = librosa.load(librosa.ex("nutcracker"), duration=20, offset=15)
D = librosa.stft(y)
D_harmonic, D_percussive = librosa.decompose.hpss(D)

Audio(data=y, rate=sr)
```

Out[13]:

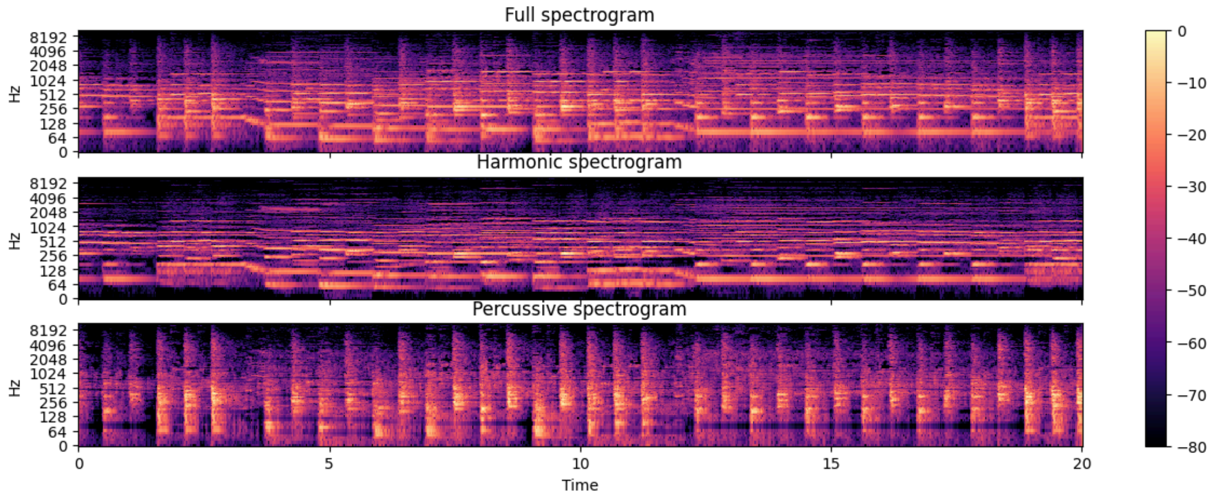
0:00 / 0:20

In [14]:

```
# Affichage des 3 spectrogrammes : complet, harmonie et percussions
rp = np.max(np.abs(D))

fig, ax = plt.subplots(nrows=3, sharex=True, sharey=True, figsize=(15,5))

img = librosa.display.specshow(librosa.amplitude_to_db(np.abs(D), ref=rp),
                               y_axis='log', x_axis='time', ax=ax[0])
ax[0].set(title='Full spectrogram')
ax[0].label_outer()

librosa.display.specshow(librosa.amplitude_to_db(np.abs(D_harmonic), ref=rp)
                         y_axis='log', x_axis='time', ax=ax[1])
ax[1].set(title='Harmonic spectrogram')
ax[1].label_outer()

librosa.display.specshow(librosa.amplitude_to_db(np.abs(D_percussive), ref=r
                         y_axis='log', x_axis='time', ax=ax[2])
ax[2].set(title='Percussive spectrogram')
fig.colorbar(img, ax=ax)
```

Out[14]:

```
<matplotlib.colorbar.Colorbar at 0x7a7f2c067d00>
```



Grâce à la transformée de Fourier inverse on peut même générer les données audio à partir du spectrogramme et écouter les deux composants isolés.

In [16]:
```
y_harmonic = librosa.istft(D_harmonic, length=len(y))
Audio(data=y_harmonic, rate=sr)
```

Out[16]: 0:00 / 0:20

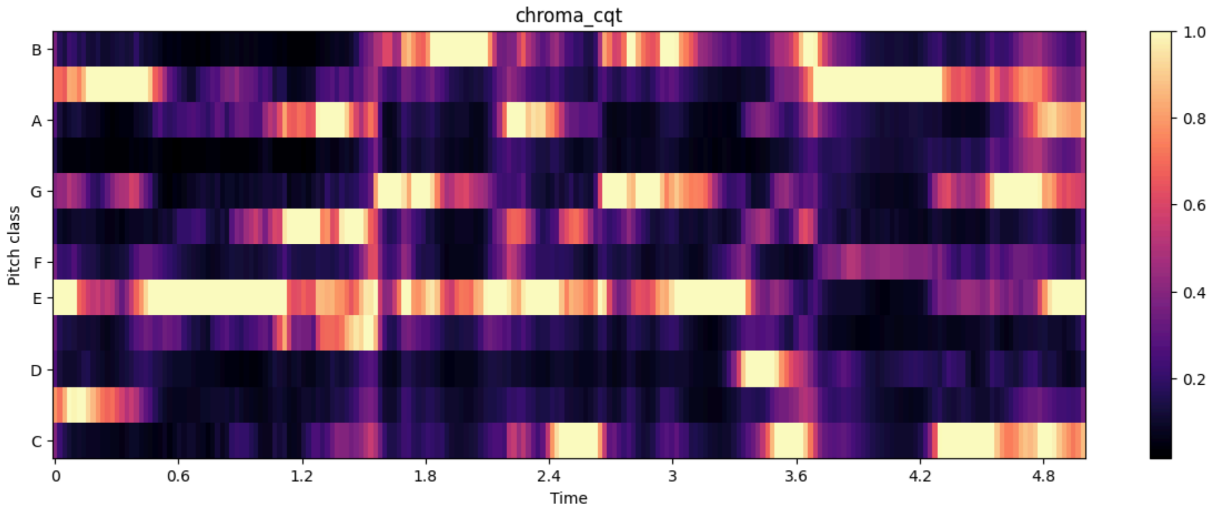
In [17]:
```
y_percussive = librosa.istft(D_percussive, length=len(y))
Audio(data=y_percussive, rate=sr)
```

Out[17]: 0:00 / 0:20

Afficher les notes sur l'échelle chromatique du composant harmonique

In [18]:
```
y, sr = librosa.load(librosa.ex("nutcracker"), duration=5, offset=15) # char
D = librosa.stft(y) # données signal vers spectrogramme (avec transformée de
D_harmonic, D_percussive = librosa.decompose.hpss(D) # décomposition harmoni
y_harmonic = librosa.istft(D_harmonic, length=len(y)) # spectrogramme -> don

chroma_cq = librosa.feature.chroma_cqt(y=y_harmonic, sr=sr) # analyse chroma

fig, ax = plt.subplots(sharex=True, figsize=(15,5))
img = librosa.display.specshow(chroma_cq, y_axis='chroma', x_axis='time', ax
ax.set(title='chroma_cqt')
fig.colorbar(img, ax=ax)
```

Out[18]:
```
<matplotlib.colorbar.Colorbar at 0x7a7f28b97100>
```



In [19]:
```
Audio(data=y_harmonic, rate=sr)
```

Out[19]:

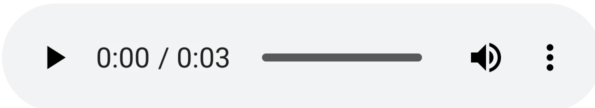
0:00 / 0:05

Générer des notes

In [32]:
```
import librosa
import numpy as np

def generate_note(freq, duration=0.5, sample_rate=22050):
    """Génère un son sinusoïdal pour une fréquence donnée"""
    t = np.linspace(0, duration, int(sample_rate * duration), False)
    return 0.5 * np.sin(2 * np.pi * freq * t)

def note_to_frequency(note):
    """Convertit un nom de note en fréquence"""
    # Dictionnaire des fréquences pour différentes notes
    note_freq = {
        'C4': 261.63, 'C#4': 277.18, 'D4': 293.66,
        'D#4': 311.13, 'E4': 329.63, 'F4': 349.23,
        'F#4': 369.99, 'G4': 392.00, 'G#4': 415.30,
        'A4': 440.00, 'A#4': 466.16, 'B4': 493.88,
        'Bb3': 233.08, 'F3': 174.61, 'G3': 196.00,
        'E3': 164.81, 'C5': 523.26
    }
    return note_freq.get(note, 440)  # Valeur par défaut : A4

def play_sequence(notes, durations=None, sample_rate=22050):
    """Génère une séquence de notes pour IPython.display"""
    if durations is None:
        durations = [0.5] * len(notes)

    # Générer la séquence audio
    sequence = []
    for note, duration in zip(notes, durations):
        freq = note_to_frequency(note)
```

```python
        note_sound = generate_note(freq, duration, sample_rate)
        sequence.extend(note_sound)

    # Retourne un objet Audio pour Jupyter
    return Audio(sequence, rate=sample_rate)
```

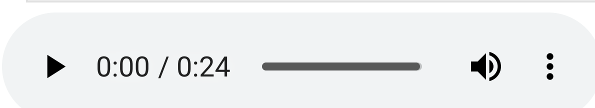
In [36]:
```python
# Génération de la séquence Bb - F - G - E
notes_sequence = ['C4', 'C4', 'C4', 'D4', 'E4', 'D4']
sequence = play_sequence(notes_sequence)
display(sequence)
```



Filtre ala French Touch

Nous allons commencer par définir un fonction `low_pass_filter` pour atténuer les fréquences supérieures à un certain seuil.
Ce filtre « coupe » les fréquences aigües en quelque sorte.

In [42]:
```python
from scipy.signal import butter, filtfilt, lfilter

# Define a low-pass filter
def low_pass_filter(data, cutoff, fs, order=4):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return filtfilt(b, a, data, padlen=1)

# Vous pouvez modifier le fichier son à filtrer
y, sr = librosa.load("../audio/electric_cello.wav")

# Apply low-pass filter
cutoff_freq = 200  # Modifiez le seuil ici
filtered_y = low_pass_filter(y, cutoff_freq, sr)

display(Audio(filtered_y, rate=sr))
```

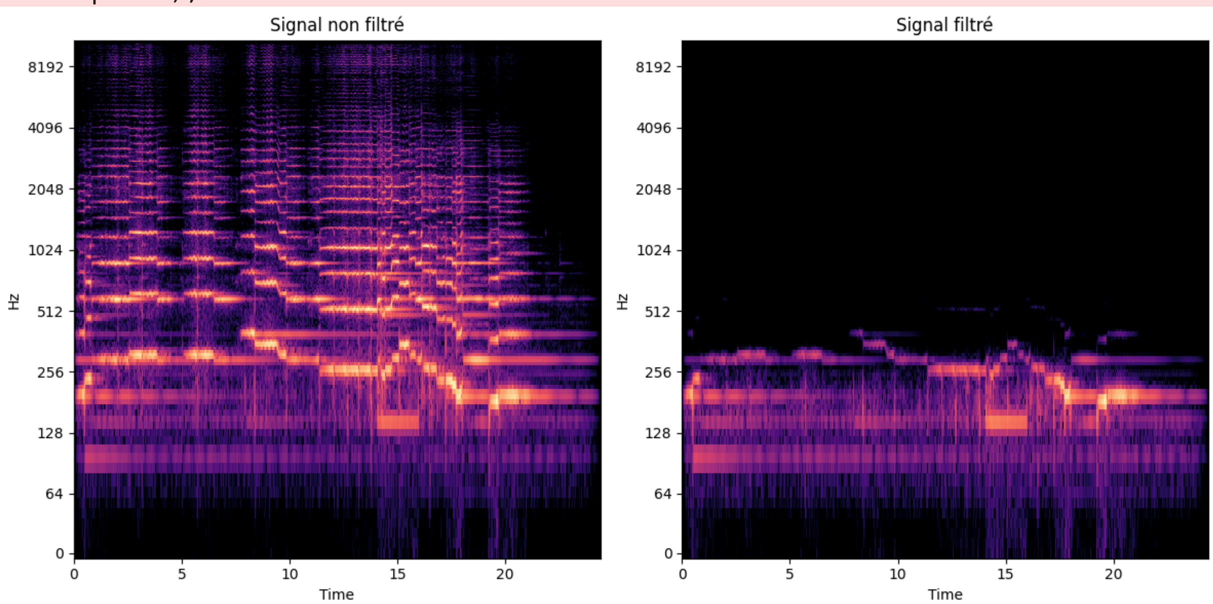


Avec une visualisation ça sera plus parlant

In [43]:
```python
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Plot the first spectrogram
librosa.display.specshow(librosa.amplitude_to_db(librosa.stft(y), ref=np.max
                         sr=sr, x_axis='time', y_axis='log', ax=axes[0])
```

```
axes[0].set_title("Signal non filtré")

# Plot the second spectrogram
librosa.display.specshow(librosa.amplitude_to_db(librosa.stft(filtered_y), r
                         sr=sr, x_axis='time', y_axis='log', ax=axes[1])
axes[1].set_title("Signal filtré")

# Display the plots
plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_47807/185004717.py:6: UserWarning: amplitude_to_db was called
on complex input so phase information will be discarded. To suppress this wa
rning, call amplitude_to_db(np.abs(S)) instead.
  librosa.display.specshow(librosa.amplitude_to_db(librosa.stft(y), ref=np.m
ax),
/tmp/ipykernel_47807/185004717.py:11: UserWarning: amplitude_to_db was calle
d on complex input so phase information will be discarded. To suppress this
warning, call amplitude_to_db(np.abs(S)) instead.
  librosa.display.specshow(librosa.amplitude_to_db(librosa.stft(filtered_y),
ref=np.max),
```



Une tentative de filtre dynamique ala French Touch : on déplace le filtre sur le signal en faisant varier le seuil.

In [46]:
```
from scipy.signal import butter, sosfilt

def dynamic_low_pass(data, sr, min_cutoff, max_cutoff, sweep_duration, order
    n_samples = len(data)
    sweep_samples = min(n_samples, int(sweep_duration * sr))

    # Create a linear sweep of cutoff frequencies
    cutoff_values = np.linspace(min_cutoff, max_cutoff, sweep_samples)
    nyquist = 0.5 * sr

    # Pre-allocate output
    output = np.zeros_like(data)
```

```python
    # Process in chunks
    start = 0
    for i in range(1, len(cutoff_values), sr // 10):  # Process in chunks ev
        cutoff = cutoff_values[i]
        normal_cutoff = cutoff / nyquist

        # Design low-pass filter
        sos = butter(order, normal_cutoff, btype='low', output='sos')

        # Apply filter in chunks
        end = min(i, n_samples)
        output[start:end] = sosfilt(sos, data[start:end])
        start = end

    return output

# Apply a filter sweep
min_cutoff = 100
max_cutoff = 5000
y, sr = librosa.load("../audio/choir_chant.wav", duration=10)
sweep_duration = len(y) / sr  # Full track length

swept_y = dynamic_low_pass(y, sr, min_cutoff, max_cutoff, sweep_duration)

display(Audio(swept_y, rate=sr))
```

Delay avec Librosa

Pour écrire cette fonction j'ai fait appel à Claude (https://claude.ai) en lui demandant :

- "Hi Claude. Can you help me code a simple delay effect with librosa library ?"
- "..."

In [47]:

```python
def apply_delay_effect(y, sr, delay_seconds=0.5, decay=0.5):
    """
    Apply a delay effect to an audio signal.

    Parameters:
    -----------
    y : numpy.ndarray
        Input audio time series
    sr : int
        Sample rate of the audio
    delay_seconds : float, optional
        Delay time in seconds (default is 0.5)
    decay : float, optional
        Decay factor for the delayed signal (default is 0.5)
```

```
    Returns:
    --------
    numpy.ndarray
        Delayed audio signal
    """
    # Calculate the number of samples for the delay
    delay_samples = int(delay_seconds * sr)

    # Create a copy of the original signal
    delayed_signal = y.copy()

    # Add the delayed and attenuated signal
    # Shift the signal and scale it down by the decay factor
    delayed_signal[delay_samples:] += decay * y[:-delay_samples]

    # Normalize to prevent clipping
    delayed_signal = librosa.util.normalize(delayed_signal)

    return delayed_signal
```

In [51]:

```
# Load a sample audio file
y, sr = librosa.load("../audio/electric_cello.wav")

# Apply different delay effects
original_audio = Audio(y, rate=sr)
delayed_audio_05 = Audio(apply_delay_effect(y, sr, delay_seconds=0.5), rate=
delayed_audio_1 = Audio(apply_delay_effect(y, sr, delay_seconds=1.0, decay=

print("Original Audio:")
display(original_audio)

print("\nDelayed Audio (0.5s delay):")
display(delayed_audio_05)

print("\nDelayed Audio (1.0s delay, lower decay):")
display(delayed_audio_1)
```

```
Original Audio:
```

0:00 / 0:24

```
Delayed Audio (0.5s delay):
```

0:00 / 0:24

```
Delayed Audio (1.0s delay, lower decay):
```

0:00 / 0:24