separation_de_sources

January 1, 2022

1 Séparation de sources

- VO Van Nghia
- PHAM Tuan Kiet
- 4MA-A

[ ]:
```
from __future__ import division
from IPython.display import Audio
import numpy as np
import scipy as scp
import pylab as pyl
from matplotlib import cm
import matplotlib.pyplot as plt
import pywt
import scipy.io as sio
from scipy import fftpack
from matplotlib.pyplot import imshow as imageplot
from mpl_toolkits.mplot3d import Axes3D
import wave
import warnings
from scipy.io.wavfile import read
from scipy.signal import find_peaks

warnings.filterwarnings("ignore")
```

Le but de ce TP est de réaliser un programme python qui effectue une séparation de sources. Plus précisément un programme qui prend en entrée un son stéréophonique, mélange instantané de plusieurs sources sonores (ici des instruments de musique) et qui renvoie plusieurs sons stéréophoniques. On utilisera la transformée de Fourier à court terme (TFCT), ou transformée de Fourier à fenêtres, vue lors du TP précédent. Le procédé général de ce programme de séparation de sources est intégralement décrit dans le poly de cours. Je vous invite à vous y reporter pour de plus amples explications. En bref, on va utiliser le fait, que la plupart du temps, deux instrusments ne jouent pas la même note au même moment et on va segmenter les spectrogrammes des deux voix de manière à attribuer chaque atome temps-fréquence à un instrument, en fonction du rapport d'intensité entre les deux voix, qui est supposé caractérisé chaque instrument. Le TP de décompose en trois parties.

1. On considère un mélange de seulement deux instruments et on fait l'hypothèse que l'on connait les coefficients du mélange.

2. On considère un mélange de trois instruments et on fait l'hypothèse que l'on connait les coefficients du mélange.
3. On considère le même mélange de trois instruments mais cette fois ci on estime les coefficients de mélange.

Ce TP s'appuiera sur les codes réalisés lors du précédent, plus précisément TFCourtTerme et RecSon.

[ ]:
```
import io
import os
import urllib

from urllib.request import urlopen


def fetch_data(
    filename,
    force_online=False,
    prefix_url="https://plmlab.math.cnrs.fr/dossal/optimisationpourlimage/raw/
 ↪master/",
):
    if os.path.exists(filename) and not force_online:
        return filename
    path = urllib.parse.urljoin(prefix_url, filename)
    data = io.BytesIO(urlopen(path).read())
    if not force_online:
        dir = os.path.dirname(filename)
        if dir:
            os.makedirs(os.path.dirname(filename), exist_ok=True)
        with open(filename, "wb") as f:
            f.write(data.read())
        return filename
    else:
        return data
```

[ ]:
```
def TFCourtTerme(Son, N=1024, rec=8):
    H = np.hanning(N)
    NS = len(Son)
    D = N // rec
    Nf = NS * rec // N - rec + 1
    TF = np.empty((N, Nf), dtype=complex)
    for i in range(Nf):
        idx = i * D
        TF[:, i] = fftpack.fft(H * Son[idx : idx + N])
    return TF
```

[ ]:
```
def RecSon(TF, rec=8):
    N = np.shape(TF)[0]
    Nf = np.shape(TF)[1]
    H = np.hanning(N)
    D = N // rec
    NS = D * Nf + D * (rec - 1)
    Son = np.zeros(NS, dtype=complex)
    for i in range(Nf):
        idx = i * D
        Son[idx : idx + N] += H * fftpack.ifft(TF[:, i])
    Son = np.real(Son)
    return Son / 3
```

[ ]:
```
def plot_spec(spec, label, figsize=(20, 20), aspect=None, colorbar=False):
    fig = plt.figure(figsize=figsize)
    ax = fig.gca()
    im = ax.imshow(spec)
    if colorbar:
        fig.colorbar(im)
    if aspect:
        ax.set_aspect(aspect)
    ax.set_xlabel("Time")
    ax.set_ylabel("Frequency")
    ax.set_title(f"Spectrogram of {label}")
```

1.1 Séparation de deux instruments

A l'aide la fonction read, charger dans deux vecteurs différents $V1$ et $V2$ les sons Mix11.wav et Mix21.wav et afficher les tranformées de Fourier à fenêtres de chacune des deux voix (on utilisera TFCourtTerme). On pourra afficher les basses fréquences pour mieux voir. On fait l'hypothèse que les voix 1 et 2 (dites V1 et V2) est obtenue à partir des deux sources S1 et S2 de la manière suivante :

$$V_1 = \frac{1}{3}S_1 + \frac{2}{3}S_2 \text{ et } V_2 = \frac{2}{3}S_1 + \frac{1}{3}S_2$$

Vous devriez voir les traces des deux instruments qui diffèrent par leur structure.

[ ]:
```
fe11, mix11 = read(fetch_data("img/Mix11.wav"))
fe21, mix21 = read(fetch_data("img/Mix21.wav"))
assert fe11 == fe21
fe = fe11
```

[ ]:
```
tf11 = TFCourtTerme(mix11)
tf21 = TFCourtTerme(mix21)
```

[ ]:
```
Audio(mix11, rate=fe)
```

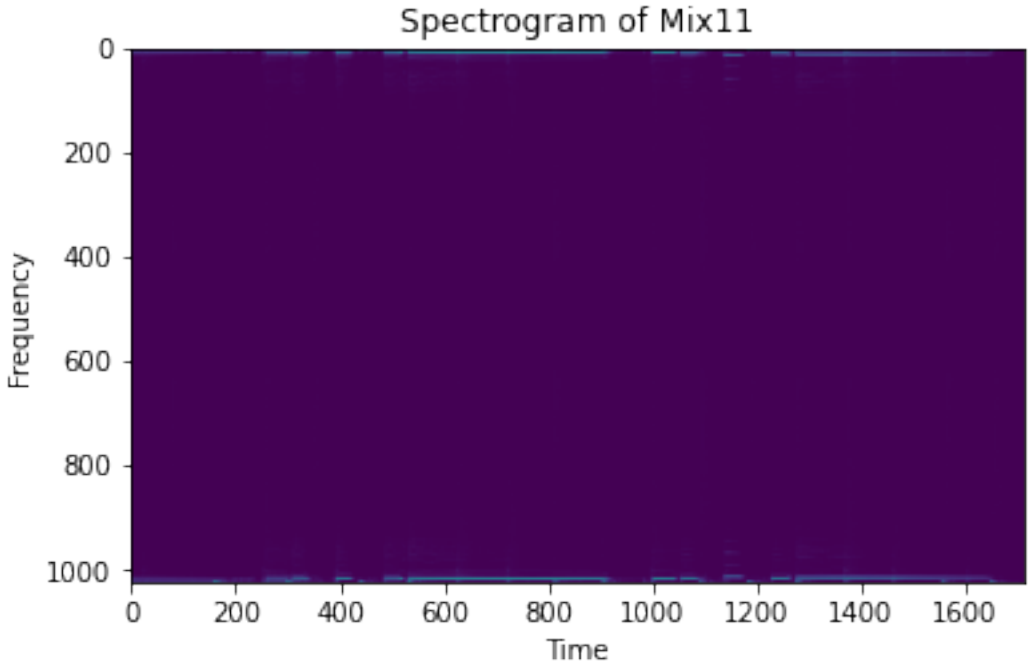
[ ]:
```
<IPython.lib.display.Audio object>
```

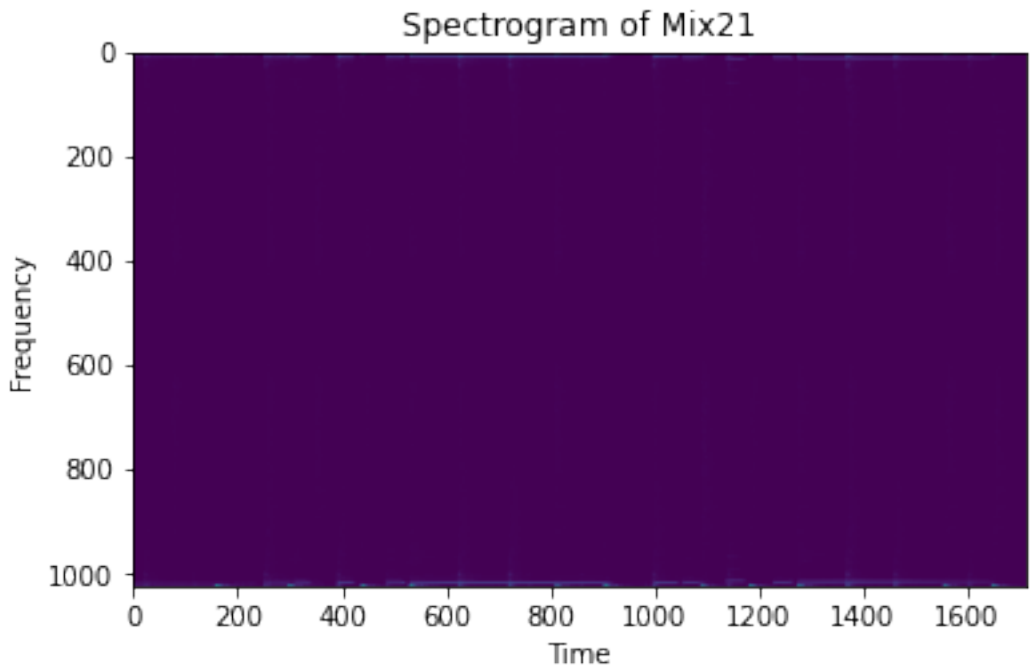
[ ]:
```
Audio(mix21, rate=fe)
```

[ ]:
```
<IPython.lib.display.Audio object>
```

[ ]:
```
spec11 = np.abs(tf11)
spec21 = np.abs(tf21)
```

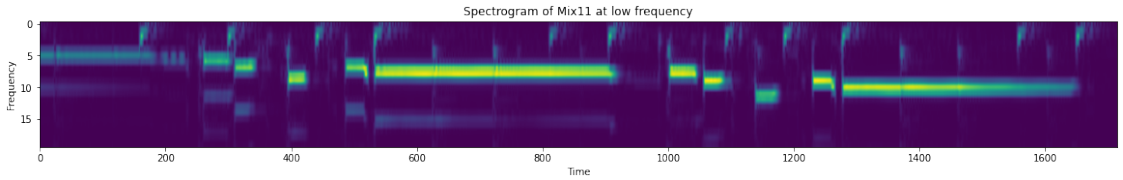
[ ]:
```
plot_spec(spec11, "Mix11", (6, 6))
plot_spec(spec21, "Mix21", (6, 6))
```

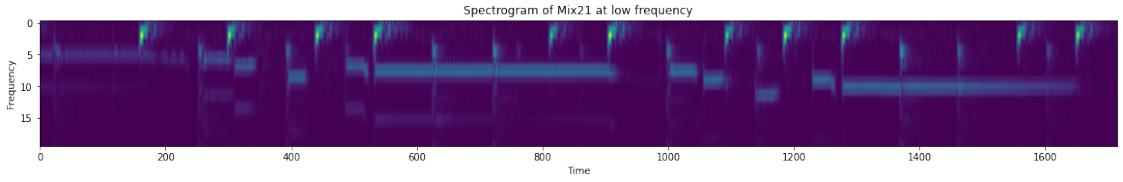




On voir qu'il y a des points verts autour des basses fréquences, donc il faut faire le spectrogramme de basse fréquence.

```
[ ]: plot_spec(spec11[:20], "Mix11 at low frequency", aspect=10)
     plot_spec(spec21[:20], "Mix21 at low frequency", aspect=10)
```





La partie plus lumineuse dans la première spectrogramme est de la guitare et celle dans la deuxième spectrogramme est de la batterie.

Pour la première fois, on peut estimer le bon résultat en résolvant les deux équations :

$$\begin{cases} V_1 = \frac{1}{3}S_1 + \frac{2}{3}S_2 \\ V_2 = \frac{2}{3}S_1 + \frac{1}{3}S_2 \end{cases} \implies \begin{cases} S_1 = 2V_2 - V_1 \\ S_2 = 2V_1 - V_2 \end{cases}$$

Ce n'est pas notre objectif puisque on veut séparer les sources sans connaître les coefficients mais cela nous donne une idée de ce que on doit obtenir au final.

[ ]:
```
tf1_true = 2 * tf21 - tf11
s1_true = RecSon(tf1_true)
Audio(s1_true, rate=fe)
```

[ ]: <IPython.lib.display.Audio object>

[ ]:
```
tf2_true = 2 * tf11 - tf21
s2_true = RecSon(tf2_true)
Audio(s2_true, rate=fe)
```

[ ]: <IPython.lib.display.Audio object>

On entend parfaitement deux sons séparés sans bruit.

Proposer un programme Separation2Instru qui prend en entrée deux Transformées de Fourier à court terme (TFCT) et un seuil T et qui renvoie 4 TFCT, chacune associée à une source et à un instrument. Le détail de la procédure est décrite dans le poly et a été expliqué en cours. On rappelle, qu'on effectue la séparation uniquement à partir du module de la TFCT. On reconstruit ensuite les TFCT en utilisant les phases des voix 1 et 2.

[ ]:
```
def separation_2(tfv1, tfv2, t):
    ratio = np.abs(tfv1) / np.abs(tfv2)
    is_s1 = ratio < t
    is_s2 = ~is_s1

    tfv1s1 = tfv1 * is_s1
    tfv1s2 = tfv1 * is_s2
    tfv2s1 = tfv2 * is_s1
    tfv2s2 = tfv2 * is_s2

    return tfv1s1, tfv1s2, tfv2s1, tfv2s2
```

Quel seuil peut-on proposer ici ? Proposer une formule générale en fonction des coefficients du mélange.

Si $\begin{cases} S_1 = \alpha_1 V_1 + \beta_1 V_2 \\ S_2 = \alpha_2 V_1 + \beta_2 V_2 \end{cases}$, on pose $R_1 = \frac{\alpha_1}{\beta_1}$ et $R_2 = \frac{\alpha_2}{\beta_2}$ les coefficients du mélange.

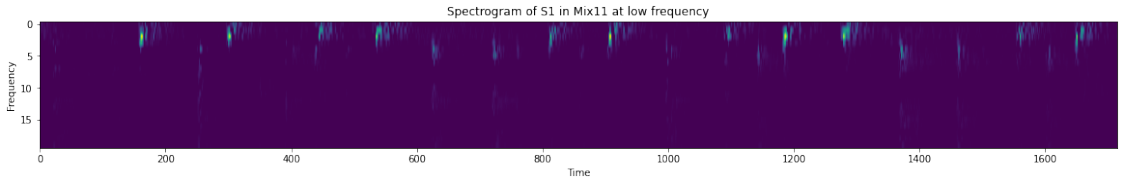
Il y a beaucoup de méthode pour choisir T de R_1 et R_2. Ici, on choisit $T = \sqrt{R_1 R_2}$.

Reconstruire ensuite à l'aide du programme RecSon, chacune des voix et écouter chacun des 4 sons produits. Commentez le résultat.

[ ]:
```
t = np.sqrt(((1 / 3) / (2 / 3)) / ((2 / 3) / (1 / 3)))
tf11s1, tf11s2, tf21s1, tf21s2 = separation_2(tf11, tf21, t)
```

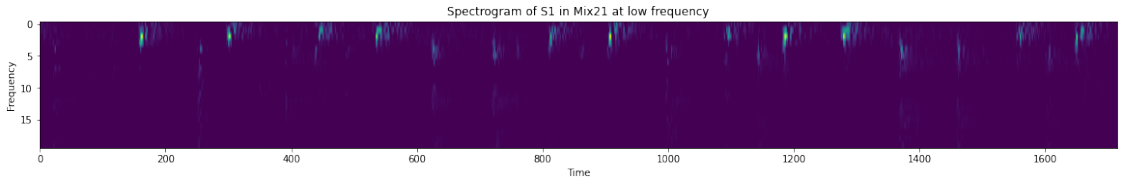
[ ]:
```
s1_11 = RecSon(tf11s1)
plot_spec(np.abs(tf11s1)[:20], "S1 in Mix11 at low frequency", aspect=10)
Audio(s1_11, rate=fe)
```

[ ]: <IPython.lib.display.Audio object>



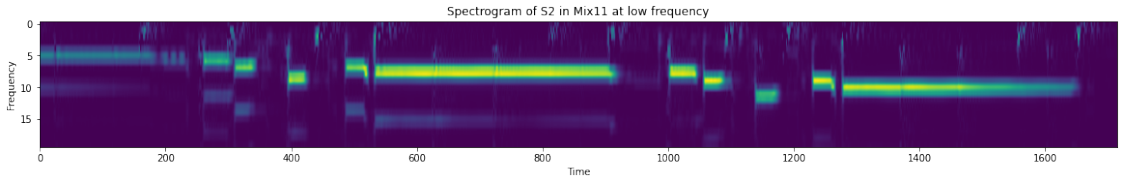
[ ]:
```
s1_21 = RecSon(tf21s1)
plot_spec(np.abs(tf21s1)[:20], "S1 in Mix21 at low frequency", aspect=10)
Audio(s1_21, rate=fe)
```

[ ]: <IPython.lib.display.Audio object>



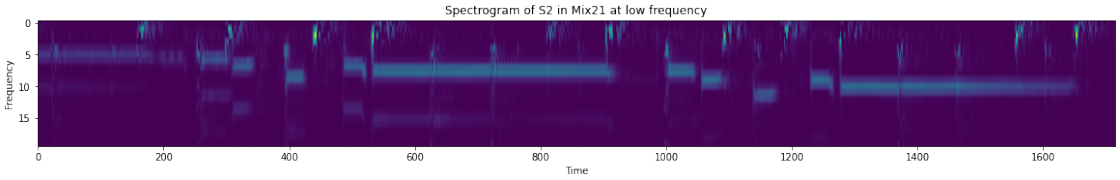
[ ]:
```
s2_11 = RecSon(tf11s2)
plot_spec(np.abs(tf11s2)[:20], "S2 in Mix11 at low frequency", aspect=10)
Audio(s2_11, rate=fe)
```

[ ]: <IPython.lib.display.Audio object>



```
[ ]: s2_21 = RecSon(tf21s2)
     plot_spec(np.abs(tf21s2)[:20], "S2 in Mix21 at low frequency", aspect=10)
     Audio(s2_21, rate=fe)
```

```
[ ]: <IPython.lib.display.Audio object>
```



Cette méthode permet de reconstituer un enregistrement parfait de chaque instrument. Cependant, il faut connaître les rapports du mélange ce qui n'est généralement pas le cas dans la réalité.

1.2 Séparation de trois instruments

Charger les nouvelles voix 1 et 2 à partir des fichiers Mix12.wav et Mix22.wav. On fait l'hypothèse que chacune des voix est maintenant un mélange instantané de trois instruments

$$SV_1 = \frac{2}{5}S_1 + \frac{1}{5}S_2 + \frac{4}{5}S_3 \text{ et } V_2 = \frac{3}{5}S_1 + \frac{4}{5}S_2 + \frac{1}{5}S_3$$

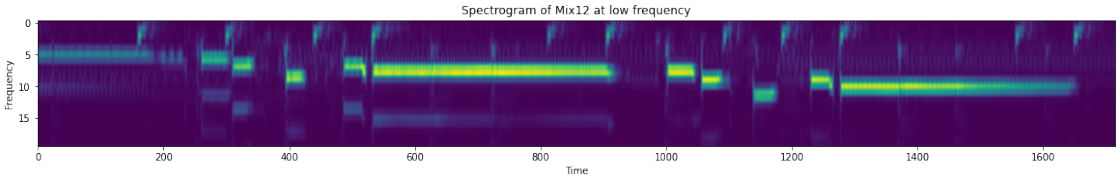
Afficher les spectrogrammes des deux voix et visualiser les traces des trois instruments.

```
[ ]: fe12, mix12 = read(fetch_data("img/Mix12.wav"))
     fe22, mix22 = read(fetch_data("img/Mix22.wav"))
     assert fe11 == fe21 == fe
```

```
[ ]: tf12 = TFCourtTerme(mix12)
     tf22 = TFCourtTerme(mix22)
```

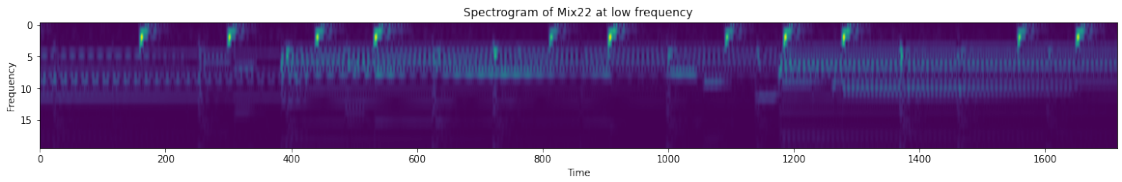
```
[ ]: plot_spec(np.abs(tf12)[:20], "Mix12 at low frequency", aspect=10)
     Audio(mix12, rate=fe)
```

```
[ ]: <IPython.lib.display.Audio object>
```



[ ]:
```
plot_spec(np.abs(tf22)[:20], "Mix22 at low frequency", aspect=10)
Audio(mix22, rate=fe)
```

[ ]:
```
<IPython.lib.display.Audio object>
```



Proposer un programme de séparation Separation3Instru qui prend en entrée deux TFCT, un vecteur T seuil à 2 composantes et qui renvoie 6 TFCT associées aux deux voix de chacun des trois instruments.

[ ]:
```
def separation_3(tfv1, tfv2, ts):
    ratio = np.abs(tfv1) / np.abs(tfv2)
    is_s1 = ratio < ts[0]
    is_s2 = (ts[0] <= ratio) & (ratio < ts[1])
    is_s3 = ts[1] <= ratio

    tfv1s1 = tfv1 * is_s1
    tfv1s2 = tfv1 * is_s2
    tfv1s3 = tfv1 * is_s3
    tfv2s1 = tfv2 * is_s1
    tfv2s2 = tfv2 * is_s2
    tfv2s3 = tfv1 * is_s3

    return tfv1s1, tfv1s2, tfv1s3, tfv2s1, tfv2s2, tfv2s3
```

Ecrire un programme CalculSeuil qui prend en entrée 3 rapports d'intensité et qui renvoie deux seuils associés.

[ ]:
```
def calcul_seuil(rs):
    index = np.argsort(rs)
    ts = np.array(
        [np.sqrt(rs[index[0]] * rs[index[1]]), np.sqrt(rs[index[1]] *␣
 ↪rs[index[2]])]
    )
    return ts
```

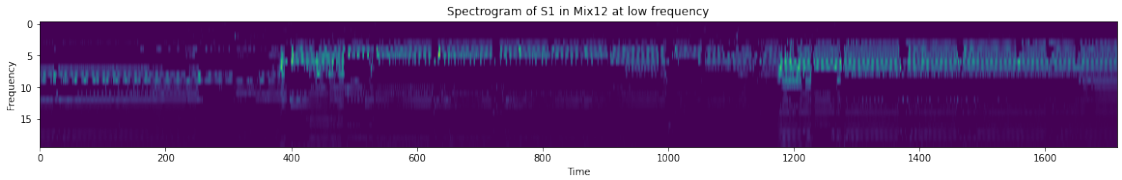
Tester le programme en utilisant les rapports d'intensités calculés à partir des coefficients de mélange, et commenter.

[ ]:
```
rs = np.array([2 / 3, 1 / 4, 4 / 1])
ts = calcul_seuil(rs)
```

[ ]:
```
tf12s1, tf12s2, tf12s3, tf22s1, tf22s2, tf22s3 = separation_3(tf12, tf22, ts)
```

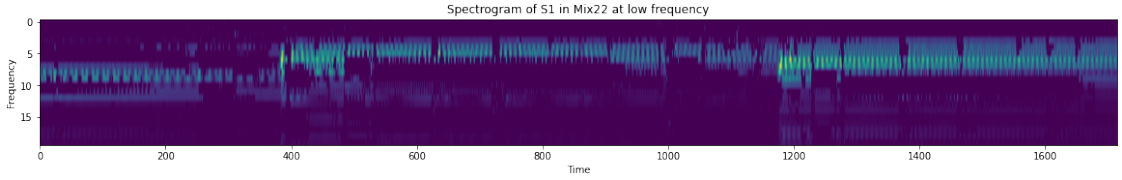
[ ]:
```
s1_12 = RecSon(tf12s1)
plot_spec(np.abs(tf12s1)[:20], "S1 in Mix12 at low frequency", aspect=10)
Audio(s1_12, rate=fe)
```

[ ]:
```
<IPython.lib.display.Audio object>
```



[ ]:
```
s1_22 = RecSon(tf22s1)
plot_spec(np.abs(tf22s1)[:20], "S1 in Mix22 at low frequency", aspect=10)
Audio(s1_22, rate=fe)
```

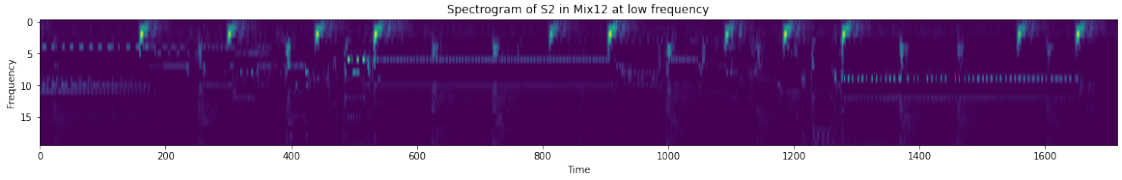
[ ]:
```
<IPython.lib.display.Audio object>
```



Le séparateur peut distinguer le son du piano mais il y a encore des defaux audibles et des discontinuation dans le son.

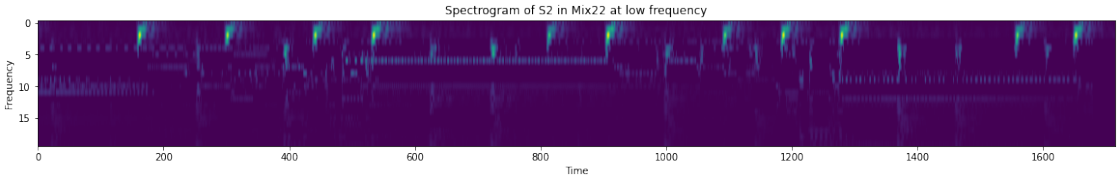
[ ]:
```
s2_12 = RecSon(tf12s2)
plot_spec(np.abs(tf12s2)[:20], "S2 in Mix12 at low frequency", aspect=10)
Audio(s2_12, rate=fe)
```

[ ]:
```
<IPython.lib.display.Audio object>
```



[ ]:
```
s2_22 = RecSon(tf22s2)
plot_spec(np.abs(tf22s2)[:20], "S2 in Mix22 at low frequency", aspect=10)
Audio(s2_22, rate=fe)
```

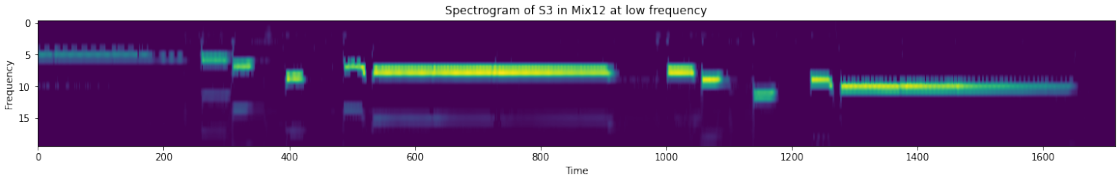
[ ]: <IPython.lib.display.Audio object>



La même résultat pour le batterie, la séparation peut distinguer la son du batterie mais il y a encore des defaux audibles.

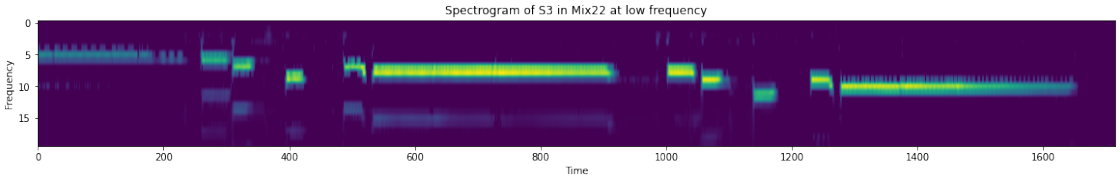
[ ]:
```
s3_12 = RecSon(tf12s3)
plot_spec(np.abs(tf12s3)[:20], "S3 in Mix12 at low frequency", aspect=10)
Audio(s3_12, rate=fe)
```

[ ]: <IPython.lib.display.Audio object>



[ ]:
```
s3_22 = RecSon(tf22s3)
plot_spec(np.abs(tf22s3)[:20], "S3 in Mix22 at low frequency", aspect=10)
Audio(s3_22, rate=fe)
```

[ ]: <IPython.lib.display.Audio object>



Le son de guitare est le son le plus correct. On voit que lorsque plus il y a d'instruments, moins le séparateur est robuste. C'est parce qu'il y a plus de chances que 2 instruments ou plus jouent la

même note en même temps.

1.3 Estimation des coeffients de mélange

On suppose dans cette partie qu'on veut effectuer la séparation précédente, sans connaitre les coefficienst de mélange. On parle de séparation aveugle. On va donc chercher à estimer ces coefficients de mélange, ou plus précisément des rapports d'intensité des deux voix pour chacun des instruments. L'idée générale est simple : on va calculer un histogramme des rapports d'intensité des modules des TFCT et sélectionner les 3 valeurs les plus réprésentées dans l'histogramme, c'est à dire les troix maxima locaux de l'histogramme. Dans le détail on procéder de la manière suivante :

1. On construit un tableau ModuleCarre qui va contenir la somme des carrés des modules des TFCT des deux voix.
2. On seuille ce tableau de manière à ne conserver que les N= 10% de coefficients les plus importants du tableau.
3. On construit une matrice $2 \times N$ qui contient les pairs d'intensité associées des TFCT des deux voix.
4. On affiche sur un diagramme 2D les points obtnus (on ne les relie pas, on affiche par exemple des croix).

Les points obtenus doivent se répartir approximativement selon des nuages dirigés le long de droites dont les coefficients directeurs sont les rapports d'intensité.

5. On calcule un vecteur R de rapports d'intensité (de longueur N).
6. On affiche un histogramme de R et un histogramme de $U = \frac{R}{1+R}$. Vous devriez observer que les maxima de l'histogramme de U sont plus marqués.
7. Ecrire un programme qui estime les 3 valeurs des maximas de l'histogramme de U et donc les rapports R_1, R_2 et R_3 associés aux trois isntruments.
8. A l'aide du programme précédent, calculer les seuils optimaux et effectuer la séparation.
9. Commenter.

1. On construit un tableau ModuleCarre qui va contenir la somme des carrés des modules des TFCT des deux voix

[ ]:
```
module_carre = np.abs(tf12) ** 2 + np.abs(tf22) ** 2
```

2. On seuille ce tableau de manière à ne conserver que les N= 10% de coefficients les plus importants du tableau.

[ ]:
```
N = int(np.size(module_carre) / 10)
module_10_args = np.argpartition(module_carre, -N, None)[-N:]
```

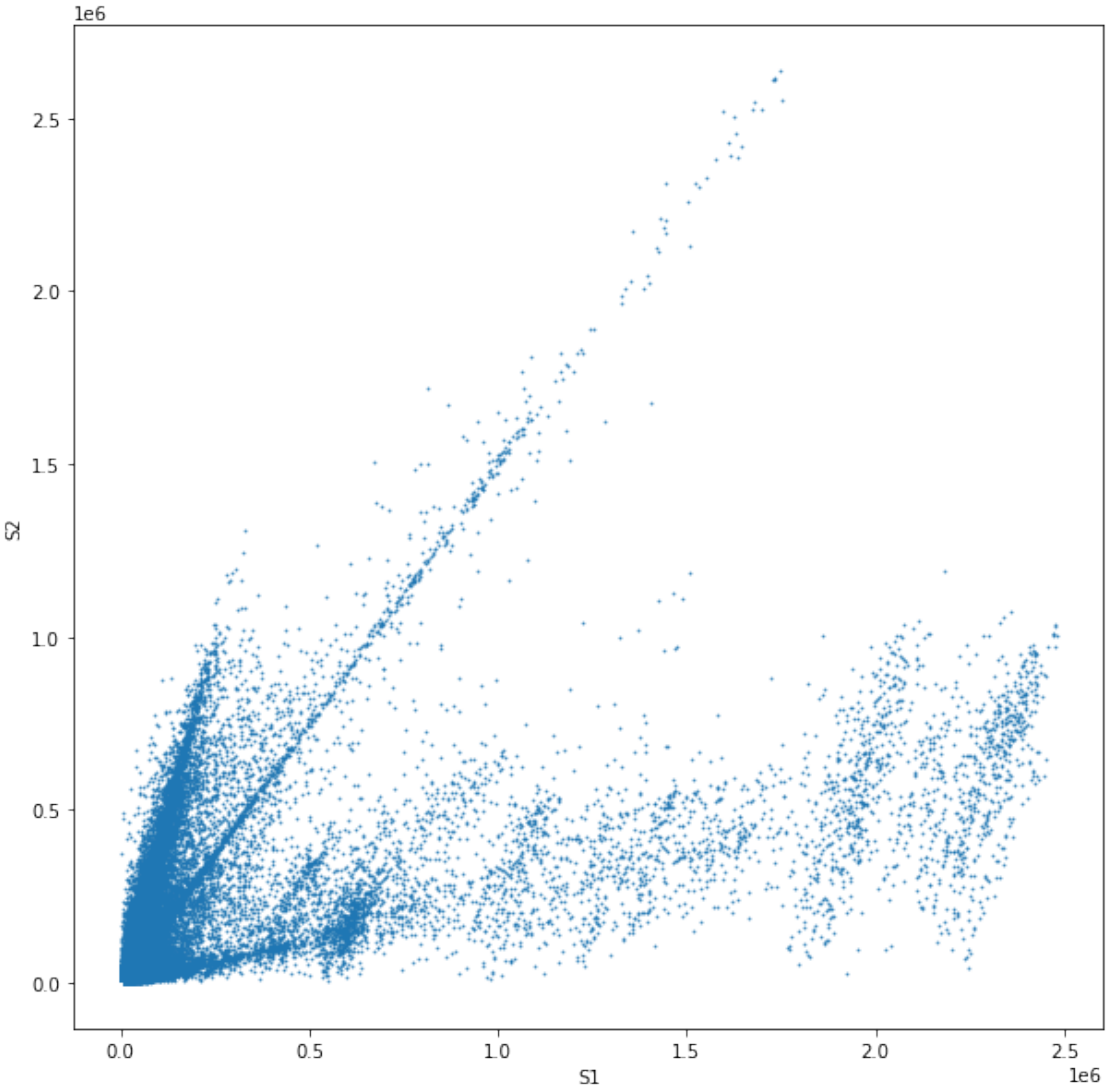
3. On construit une matrice $2 \times N$ qui contient les pairs d'intensité associées des TFCT des deux voix.

[ ]:
```
tab_10_12 = np.abs(np.take_along_axis(tf12, module_10_args, None))
tab_10_22 = np.abs(np.take_along_axis(tf22, module_10_args, None))
```

```
tab_10 = np.vstack((tab_10_12, tab_10_22))
```

4. On affiche sur un diagramme 2D les points obtnus (on ne les relie pas, on affiche par exemple des croix).

[ ]:
```
plt.figure(figsize=(10, 10))
plt.scatter(tab_10_12, tab_10_22, s=0.5, alpha=0.5)
plt.xlabel("S1")
plt.ylabel("S2")
plt.show()
```



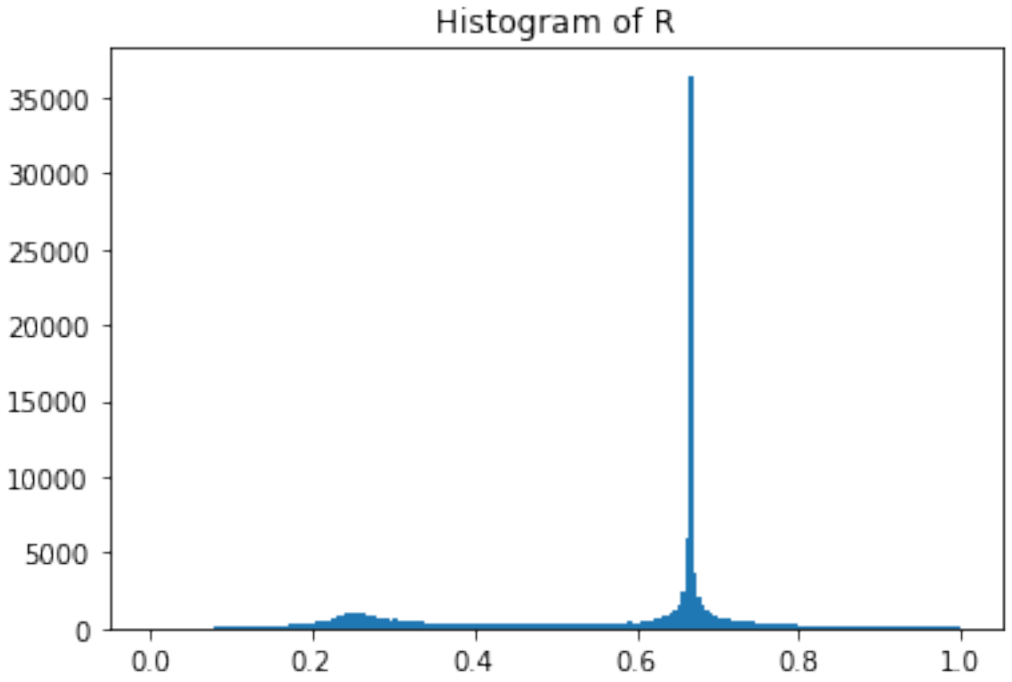
On observe clairement qu'il y a 3 tendances dans le graphique ci-dessus.

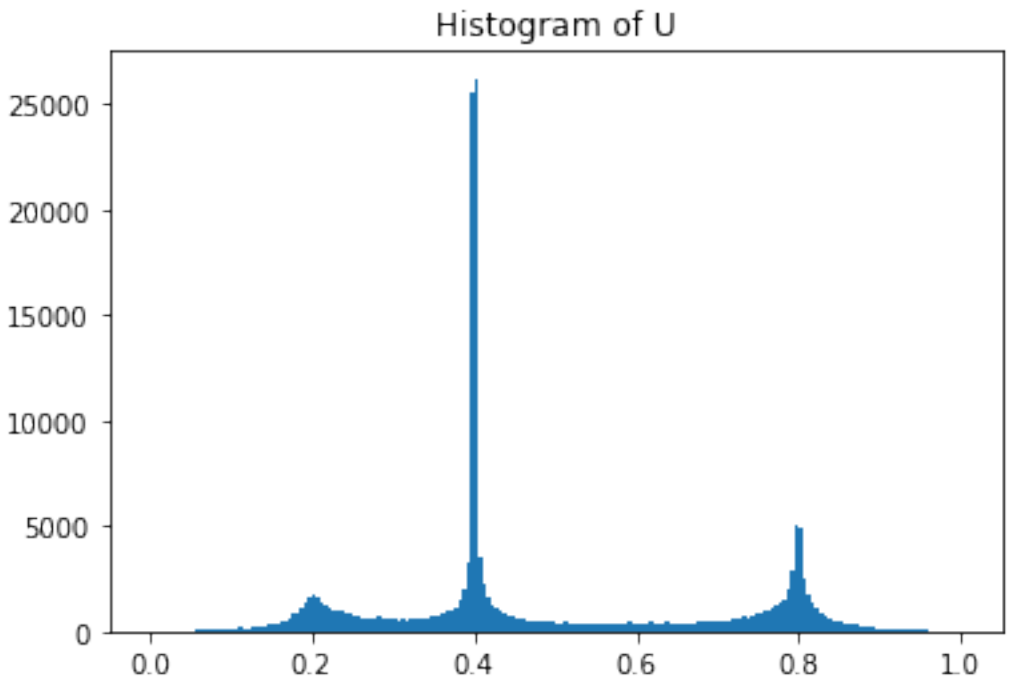
5. On calcule un vecteur R de rapports d'intensité (de longueur N).

```
[ ]: R = tab_10_12 / tab_10_22
```

6. On affiche un histogramme de R et un histogramme de $U = \frac{R}{1+R}$. Vous devriez observer que les maxima de l'histogramme de U sont plus marqués.

```
[ ]: U = R / (1 + R)
plt.hist(R, bins=200, range=(0, 1))
plt.title("Histogram of R")
plt.show()
u_hist = plt.hist(U, bins=200, range=(0, 1))
plt.title("Histogram of U")
plt.show()
```

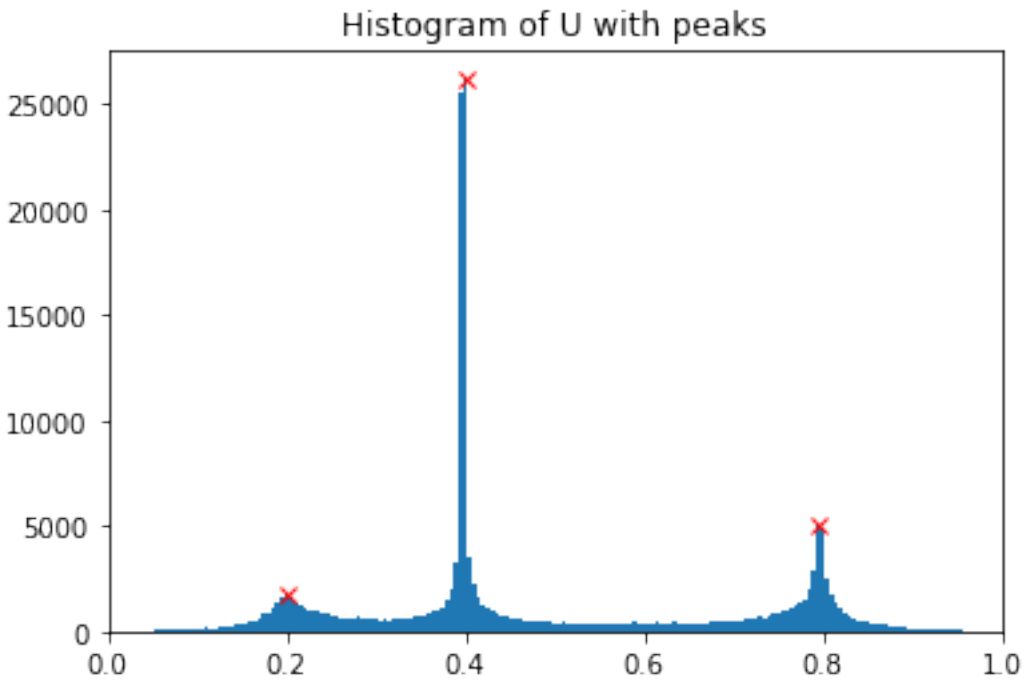




7. Ecrire un programme qui estime les 3 valeurs des maximas de l'histogramme de U et donc les rapports R_1, R_2 et R_3 associés aux trois isntruments.

[ ]:
```
def calculate_r(hist, label, **kwargs):
    peaks_idx, _ = find_peaks(hist[0], **kwargs)
    H, bins = hist[0], hist[1]
    x_peaks = bins[peaks_idx]
    y_peaks = H[peaks_idx]
    plt.bar(bins[:-1], H, width=1 / np.size(H), align="center")
    plt.plot(x_peaks, y_peaks, "x", color="red")
    plt.xlim((0, 1))
    plt.title(f"Histogram of {label} with peaks")
    plt.show()
    return x_peaks / (1 - x_peaks)
```

[ ]:
```
rs = calculate_r(u_hist, "U", height=(1000, None))
print(rs)
```




```
[0.25       0.66666667 3.87804878]
```

Les valeurs estimées sont toutes proches des valeurs théoriques des R_1, R_2 et R_3.

8. A l'aide du programme précédent, calculer les seuils optimaux et effectuer la séparation.

[ ]:
```
ts = calcul_seuil(rs)
print(ts)
```

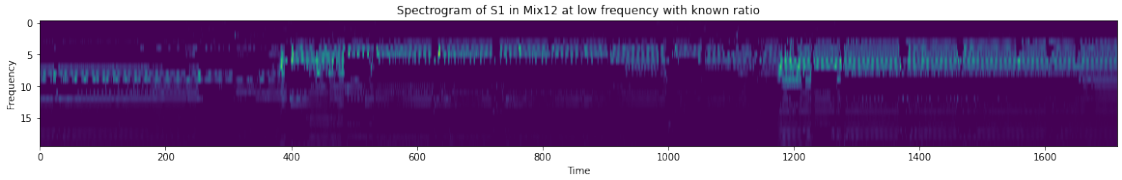
```
[0.40824829 1.60790729]
```

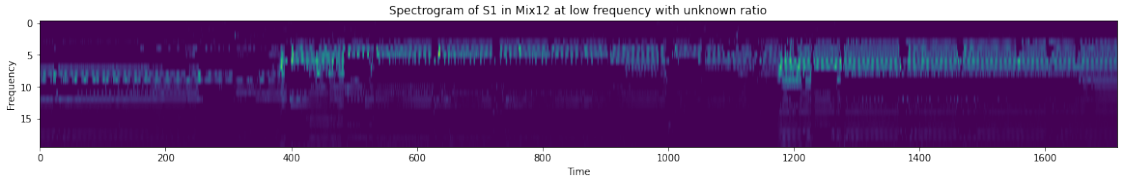
[ ]:
```
tf12s1u, tf12s2u, tf12s3u, tf22s1u, tf22s2u, tf22s3u = separation_3(tf12, tf22,␣
↪ts)
```

9. Commenter.

[ ]:
```
s1_12u = RecSon(tf12s1u)
plot_spec(
    np.abs(tf12s1)[:20], "S1 in Mix12 at low frequency with known ratio",␣
↪aspect=10
)
plot_spec(
    np.abs(tf12s1u)[:20], "S1 in Mix12 at low frequency with unknown ratio",␣
↪aspect=10
)
Audio(s1_12u, rate=fe)
```

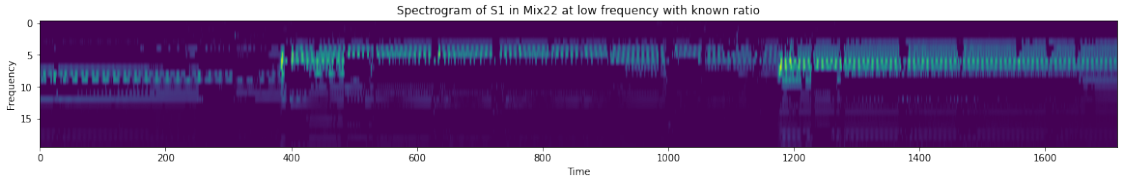
[ ]: <IPython.lib.display.Audio object>

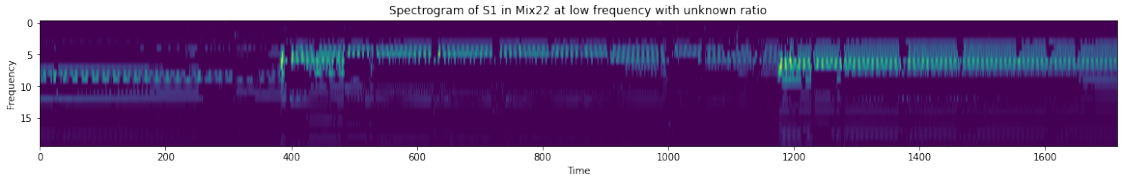




[ ]:
```
s1_22u = RecSon(tf22s1u)
plot_spec(
    np.abs(tf22s1)[:20], "S1 in Mix22 at low frequency with known ratio",␣
 ↪aspect=10
)
plot_spec(
    np.abs(tf22s1u)[:20], "S1 in Mix22 at low frequency with unknown ratio",␣
 ↪aspect=10
)
Audio(s1_22u, rate=fe)
```

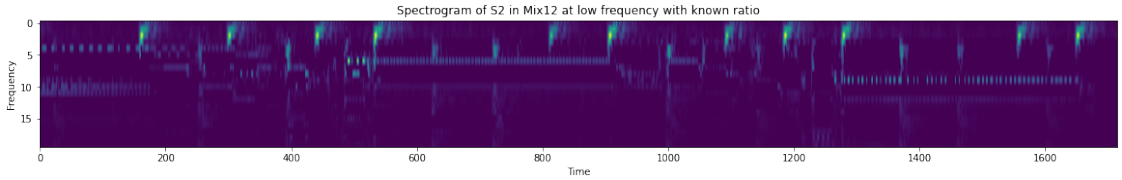
[ ]: <IPython.lib.display.Audio object>

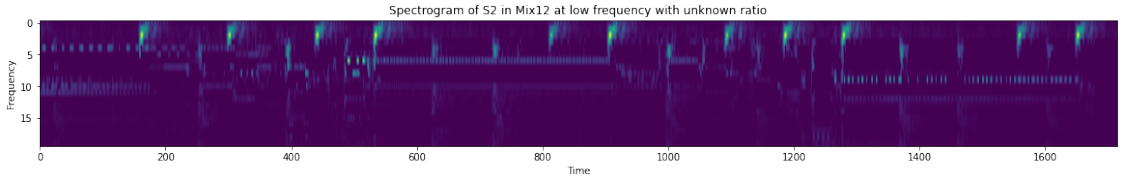




[ ]:
```
s2_12u = RecSon(tf12s2u)
plot_spec(
    np.abs(tf12s2)[:20], "S2 in Mix12 at low frequency with known ratio",␣
 ↪aspect=10
)
plot_spec(
    np.abs(tf12s2u)[:20], "S2 in Mix12 at low frequency with unknown ratio",␣
 ↪aspect=10
)
Audio(s2_12u, rate=fe)
```

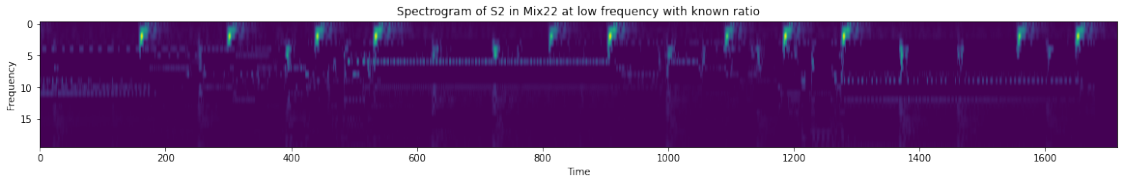
[ ]: <IPython.lib.display.Audio object>

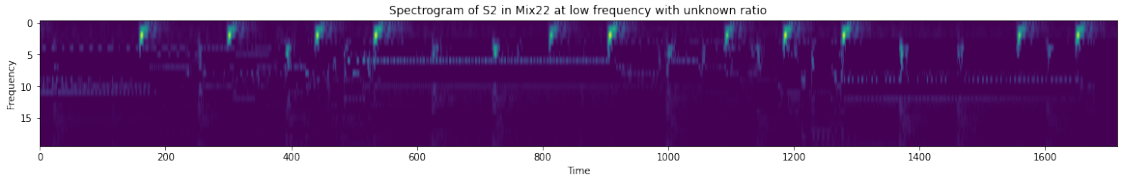




[ ]:
```
s2_22u = RecSon(tf22s2u)
plot_spec(
    np.abs(tf22s2)[:20], "S2 in Mix22 at low frequency with known ratio",␣
 ↪aspect=10
)
plot_spec(
    np.abs(tf22s2u)[:20], "S2 in Mix22 at low frequency with unknown ratio",␣
 ↪aspect=10
)
Audio(s2_22u, rate=fe)
```

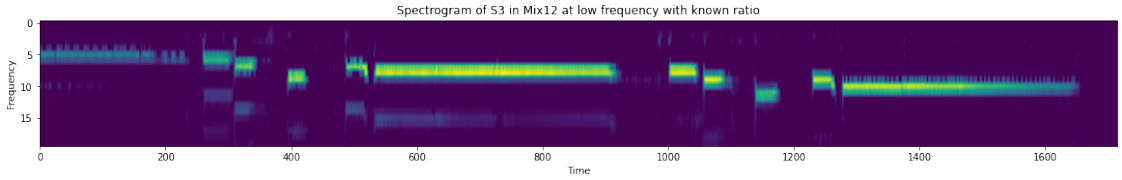
[ ]: <IPython.lib.display.Audio object>

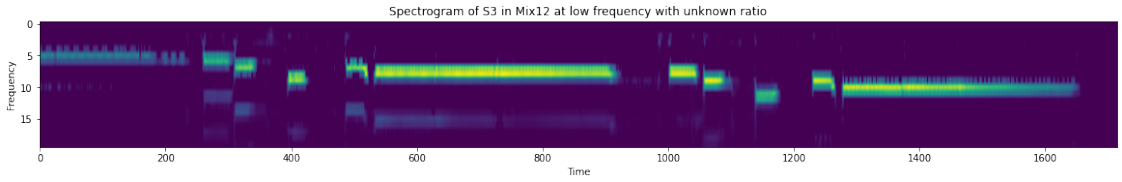




[ ]:
```
s3_12u = RecSon(tf12s3u)
plot_spec(
    np.abs(tf12s3)[:20], "S3 in Mix12 at low frequency with known ratio",␣
 ↪aspect=10
)
plot_spec(
    np.abs(tf12s3u)[:20], "S3 in Mix12 at low frequency with unknown ratio",␣
 ↪aspect=10
)
Audio(s3_12u, rate=fe)
```

[ ]:
```
<IPython.lib.display.Audio object>
```

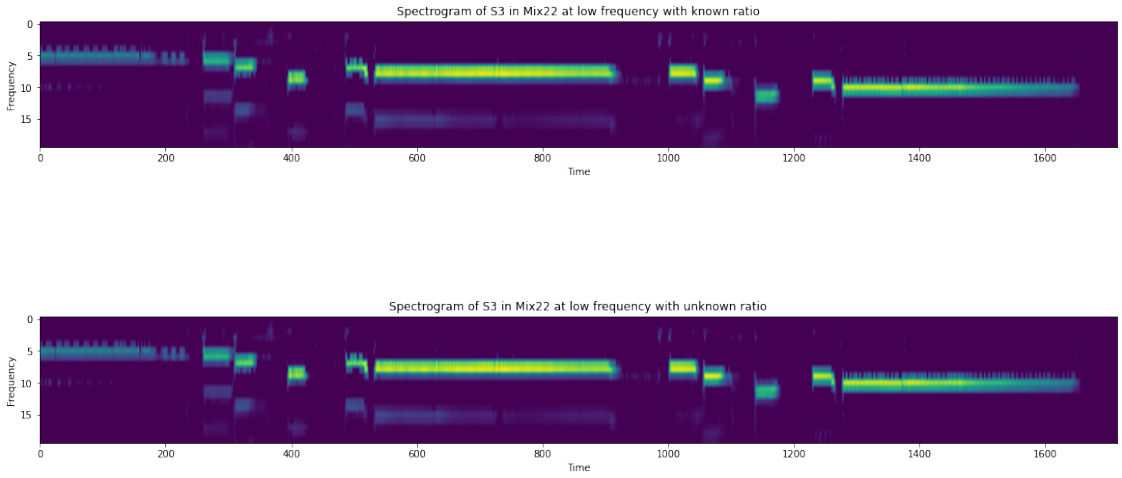




[ ]:
```
s3_22u = RecSon(tf22s3u)
plot_spec(
    np.abs(tf22s3)[:20], "S3 in Mix22 at low frequency with known ratio",␣
 ↪aspect=10
)
plot_spec(
```

```
    np.abs(tf22s3u)[:20], "S3 in Mix22 at low frequency with unknown ratio",␣
 ↪aspect=10
)
Audio(s3_22u, rate=fe)
```

[ ]: <IPython.lib.display.Audio object>



Les mêmes résultats peuvent être obtenus quelle que soit la méthode. Les résultats ne sont évidemment pas parfaits mais acceptables.