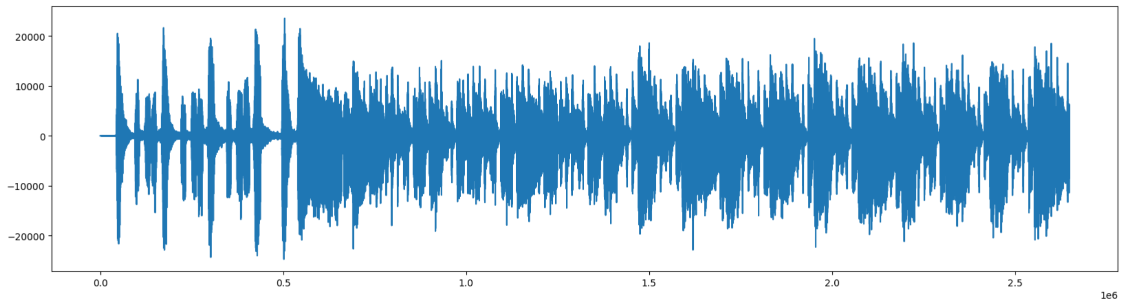
tp-vocoder

December 26, 2024

[1]:
```
import os, sys, wave, struct

import numpy as np
import pyaudio
import pandas as pd
import matplotlib.pyplot as plt

from copy import deepcopy
from math import ceil
from scipy.io.wavfile import write
from IPython.display import Audio
from scipy.signal import stft, istft
```

[54]:
```
# read wave file
def read_wave(filename):
    with wave.open(filename, 'rb') as wf:
        Fs = int(wf.getframerate())
        num_frames = wf.getnframes()
        num_channels = wf.getnchannels()
        sample_width = wf.getsampwidth()
        frames = wf.readframes(num_frames)
        fmt = '<{n}{t}'.format(n=num_frames * num_channels, t='h' if␣
 ↪sample_width == 2 else 'B')
        data = struct.unpack(fmt, frames)
        return Fs, np.array(data)


Fs, x = read_wave('salsa.wav')

fig, ax = plt.subplots(1, 1, figsize=(20, 5))
ax.plot(x)
plt.show()

start = np.argmax(x > 10000)
x = x[start:]
```



OLA

[2]:
```
def ola(w = None,hop = None,Nb = 10):
# function output = ola(w,hop,Nb)
# realise l'addition-recouvrement de la fenetre w,
# avec un décalage hop et un nombre Nb de fenetres.
# par defaut Nb = 10;

    w = w[:, np.newaxis]
    N = len(w)
    output = np.zeros(((Nb - 1) * hop + N,1)) # réserve l'espace memoire

    for k in np.arange(0,Nb).reshape(-1):
        deb = k* hop
        fin = deb + N
        output[np.arange(deb,fin)] = output[np.arange(deb,fin)] + w # OLA

    return output
```

TFCT

[9]:
```
N = x.shape[0] # % longueur du signal
Nw = 512
w = np.hanning(Nw) # définition de la fenetre d'analyse
ws = w.copy; # définition de la fenêtre de synthèse
R = Nw/4 # incrément sur les temps d'analyse, appelé hop size, t_a=uR
M = 512 # ordre de la tfd
L = M/2+1
affich = 0 ; # pour affichage du spectrogramme, 0 pour
             # pour faire analyse/modif/synthèse sans affichage
             # note: cf. spectrogram sous Matlab
Nt = np.rint((N - Nw) / R) # calcul du nombre de tfd à calculer
Nt = Nt.astype(int)
y = np.zeros((N,1)) # signal de synthèse

if affich:
```

```
    Xtilde = np.zeros((M,Nt),dtype=complex)
```

[10]:
```
for u in np.arange(0,Nt).reshape(-1): # boucle sur les trames
    deb = u * R + 1 # début de trame
    fin = deb + Nw # fin de trame
    tx = np.multiply(x[np.arange(deb.astype(int),fin.astype(int))],w) # calcul␣
 ↪de la trame
    X = np.fft.fft(tx,M) # tfd à l'instant b
    if affich:
        Xtilde[:,u] = X
    # opérations de transformation (sur la partie \nu > 0)
    # ....
    Y = X.copy
    # fin des opérations de transformation
    # resynthèse
    # overlap add

def extents(f):
  delta = f[1] - f[0]
  return [f[0] - delta/2, f[-1] + delta/2]
```

1 1 TFCT

1.1 1.1 Généralités

(a) It could be written as a convolution between $x(n)e^{-2\pi jfn}$ and $w(-n)$. If the window $w(n)$ is even, it is a Type 1 FIR filter. If the window $w(n)$ is odd, it is a Type 3 FIR filter.

(b) This step considers the window is fixed and moves the signal from right to left. $\tilde{X}_{loc}(b,f) = \tilde{X}_0(b,f)e^{2\pi jfb}$. `tfct.m` is the implementation of "convention passe-bande".

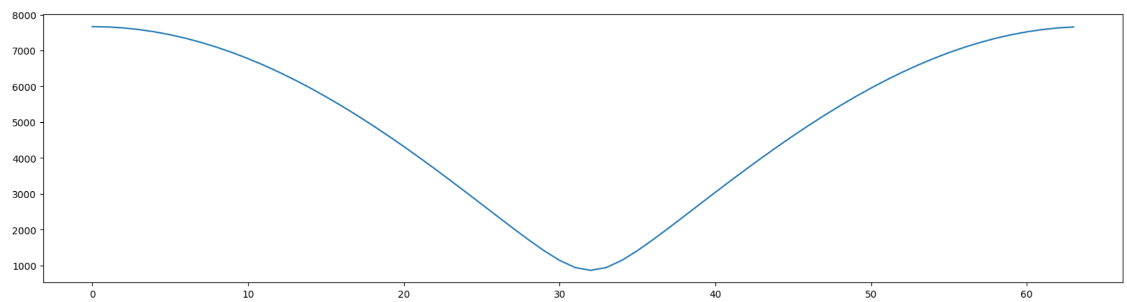
(c)

[55]:
```
def my_stft(x, w, hop_size=128, nfft=512):
    N = x.shape[0]
    Nw = w.shape[0]
    L = nfft // 2 + 1
    Nt = int(np.rint((N - Nw) / hop_size))
    X = np.zeros((L, Nt), dtype=complex)
    Xtilde = np.zeros((nfft,Nt),dtype=complex)

    Nt = np.rint((N - Nw) / hop_size) # calcul du nombre de tfd à calculer
    Nt = Nt.astype(int)
    for u in np.arange(0,Nt).reshape(-1): # boucle sur les trames
        deb = u * hop_size + 1 # début de trame
        fin = deb + Nw # fin de trame
        tx = np.multiply(x[np.arange(deb.astype(int),fin.astype(int))],w) #␣
 ↪calcul de la trame
```

```
        X = np.fft.fft(tx, nfft) # tfd à l'instant b
        Xtilde[:,u] = X

    times = np.arange(Nt) * hop_size
    freqs = np.fft.fftfreq(n=nfft)
    freqs = np.fft.fftshift(freqs)

    return times, freqs, Xtilde
```

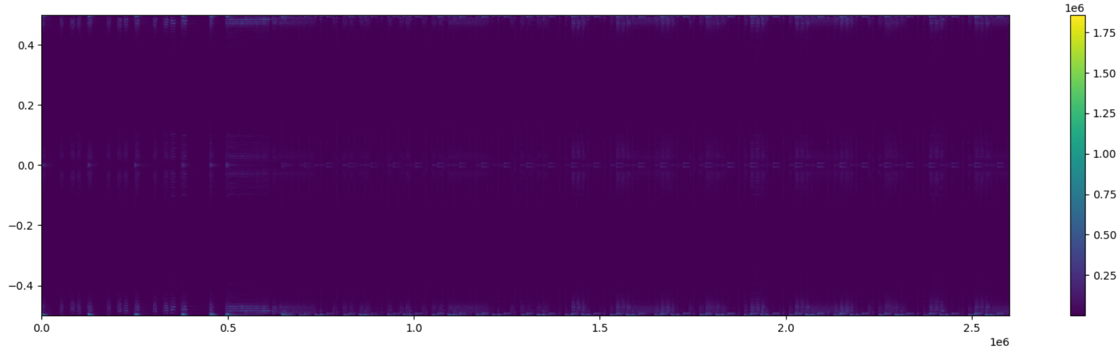
[56]:
```
N = 64
R = 1
w = np.hanning(R*4)
t, f, Xtilde = my_stft(x, w, R, N)
k = 12
x_k = Xtilde[:, k]
print(np.iscomplex(x_k))
fig, ax = plt.subplots(1, 1, figsize=(20, 5))
ax.plot(np.abs(x_k))
plt.show()
```

```
[False  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True False  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True]
```



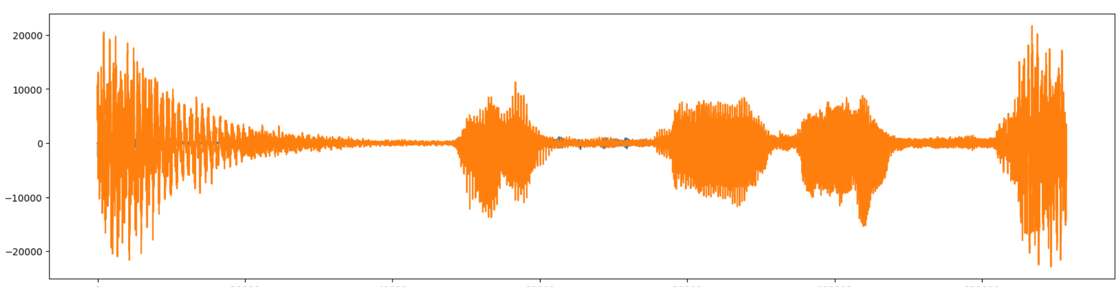
[62]:
```
N = 1024
R = 128
w = np.hanning(512)

t, f, Xtilde = my_stft(x, w, R, N)
fig, ax = plt.subplots(1, 1, figsize=(20, 5))
cax = ax.imshow(np.abs(Xtilde), aspect='auto', origin='lower', extent=[t[0],␣
 ↪t[-1], f[0], f[-1]])
fig.colorbar(cax, ax=ax)
```

```
plt.show()
```



1.2 1.2 Reconstruction

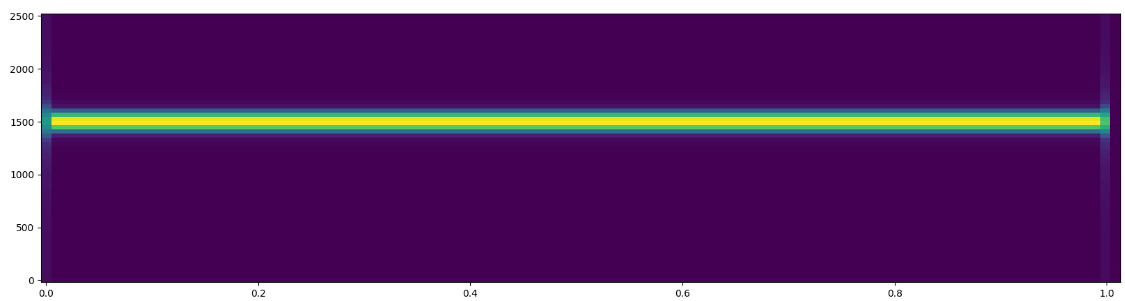
[65]:
```
def ola(X, w, ws, hop_size=128):
    Nw = w.shape[0]
    Nf = X.shape[0]
    nfft = X.shape[1]
    Nt = int(Nw + (Nf - 1) * hop_size)
    y = np.zeros(Nt, dtype=complex)
    ys = np.fft.ifft(X, n=nfft, axis=-1)[:, :Nw] * ws
    for p in range(Nf):
        start = p * hop_size
        end = start + Nw
        y[start: end] += ys[p]
    return y

W = np.fft.ifftshift(w)
y = ola(Xtilde, W, W, R)

fig, ax = plt.subplots(1, 1, figsize=(20, 5))
ax.plot(np.real(y))
ax.plot(x[:len(y)])
plt.show()
```



2 2 Étirement temporel

2.1 a

[ ]:
```
nfft = 128
win_len = 64
Fs = 5e3
f0 = 1e3
time_scale = np.arange(0, 1, 1/Fs)

x_f0 = np.exp(1j * 2 * np.pi * f0 * time_scale)

# STFT
f, t, Z = stft(np.real(x_f0), window='hann', fs=Fs, nperseg=win_len,␣
 ↪noverlap=win_len//4, nfft=nfft, return_onesided=True)

fig, ax = plt.subplots(1, 1, figsize=(20, 5))
ax.imshow(np.abs(Z), extent=extents(t) + extents(f), aspect='auto')
plt.show()

Ang_Z = np.angle(Z)

print(np.abs(np.max(Ang_Z) - np.min(Ang_Z)))
```



6.283185307179542

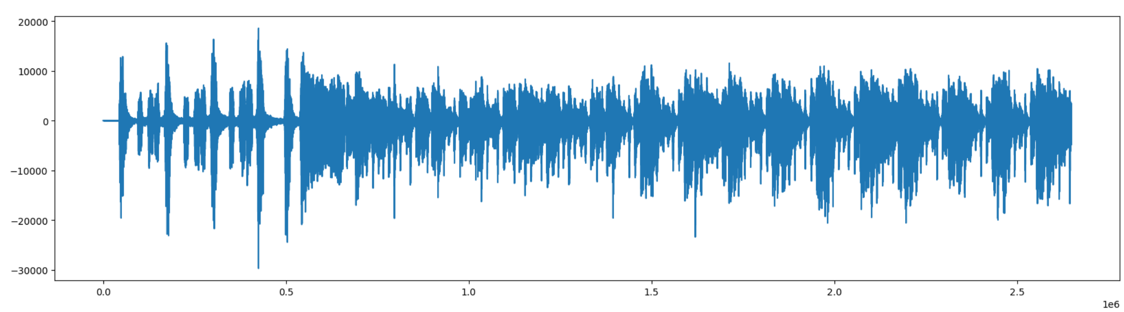
(b) $R < |\frac{1}{\Delta_f}|$

(c) $R < \frac{1}{2f_{max}}$ so that $R < \frac{1}{\max(\Delta_f, 2f_{max})}$

[93]:
```
def pitch_shift(signal, sample_rate, pitch_factor):

    win_length = 2048
    hop_length = 512
```

```
    f, t, Zxx = stft(signal, fs=sample_rate, nperseg=win_length,␣
↪noverlap=win_length-hop_length)

    num_bins = Zxx.shape[0]
    new_bins = int(num_bins / pitch_factor)
    Zxx_resampled = np.zeros((num_bins, Zxx.shape[1]), dtype=complex)

    for i in range(Zxx.shape[1]):
        resampled_magnitude = np.interp(
            np.linspace(0, new_bins, num_bins),
            np.arange(num_bins),
            np.abs(Zxx[:, i])
        )
        resampled_phase = np.interp(
            np.linspace(0, new_bins, num_bins),
            np.arange(num_bins),
            np.angle(Zxx[:, i])
        )
        Zxx_resampled[:, i] = resampled_magnitude * np.exp(1j * resampled_phase)

    _, output_signal = istft(Zxx_resampled, fs=sample_rate, nperseg=win_length,␣
↪noverlap=win_length-hop_length)

    return output_signal

sr, x = read_wave('salsa.wav')
y = pitch_shift(x, sr, 2)
```

[94]:
```
fig, ax = plt.subplots(1, 1, figsize=(20, 5))
ax.plot(y)
plt.show()
```



[95]:
```
Audio(y, rate=sr*2)
```

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

[96]:
```
x_hat = pitch_shift(y, sr*2, 1/2)
Audio(x_hat, rate=sr*2)
```

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