Transformee Fourier

September 28, 2022

1 Transformée de Fourier

Discret Fourier Transform - DFT (algorithm : Fast Fourier Transform = FFT)

N points x_i

$$\tilde{x}_k = \sum_i x_i e^{-2j\pi k \frac{i}{N}}$$

Signal de durée T échantillonée avec dt : $T = Ndt$

Fréquence minimum : $1/T$

Fréquence maximum : $\frac{1}{2dt}$ (Nyquist)

Le point $\tilde{x}_k$ a une fréquence k/T

Attention : la DFT suppose implicitement que la fonction est périodique de période T.

Utilisation : filtre, convolution.

FFT : complexité en $N \log(N)$

Librairie numpy : * `np.fft.fft`; `np.fft.ifft` * `np.fft.rfft`; `np.fft.irfft` (signaux réél -> fréquences >0). Evite les nb complexe * `np.fft.fftfreq`; `np.fft.fftshift`

1.1 Exemple : bruit d'une machine à laver le linge (essorage)

Déterminer la fréquence de rotation du tambour * En visualisant la fréquence sur une transformée de Fourier * En filtrant le signat à l'aide d'une transformée de Fourier

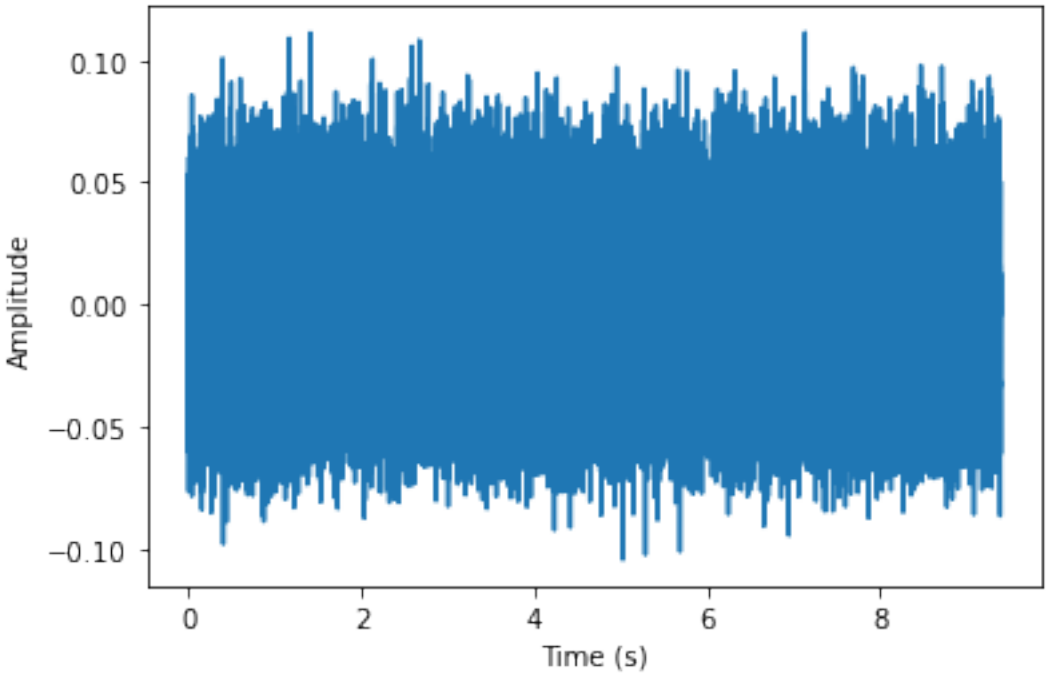
[2]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import read
```

1.2 Plot dans le domaine temporel

[3]:
```
samplerate, amplitude = read('machine_a_laver.wav')

t = np.arange(len(amplitude))/samplerate
plt.plot(t, amplitude)
plt.xlabel('Time (s)')
```

```
plt.ylabel('Amplitude')
```

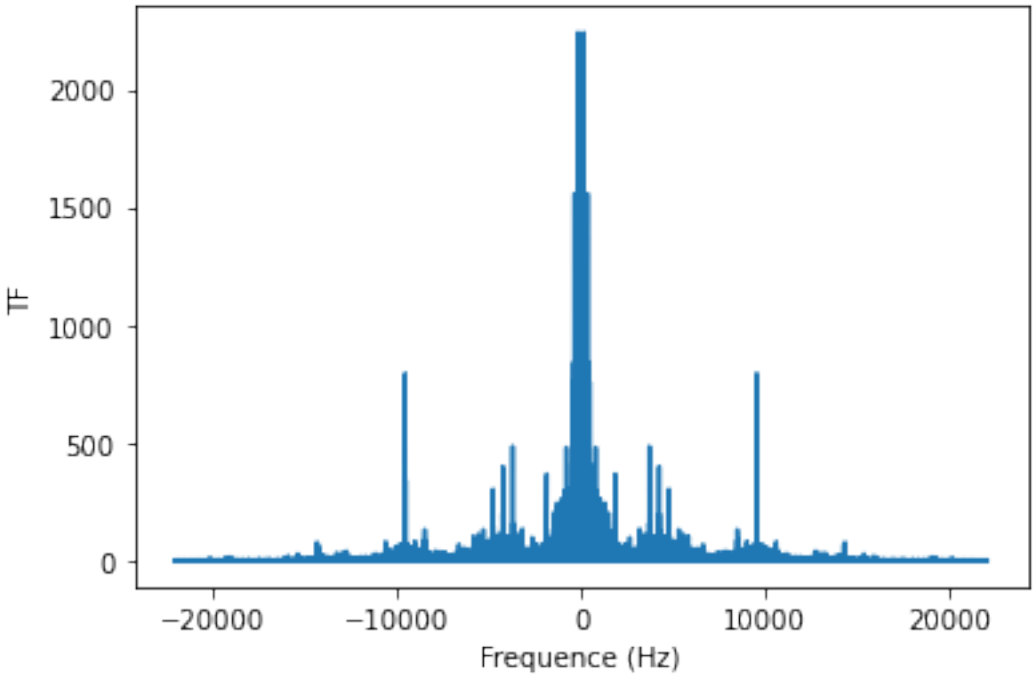
[3]: Text(0, 0.5, 'Amplitude')



1.3 Transformée de Fourier

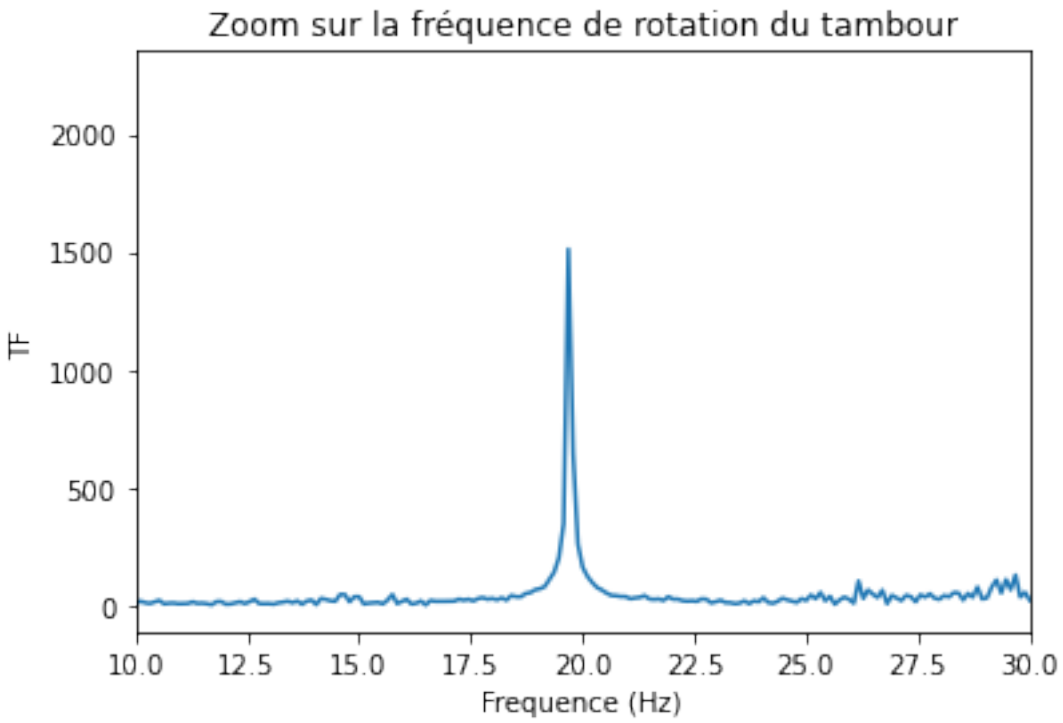
[4]:
```
amplitude_tilde = np.fft.fft(amplitude)
freq = np.fft.fftfreq(len(amplitude), d=1/samplerate)

plt.plot(np.fft.fftshift(freq), np.fft.fftshift(np.abs(amplitude_tilde)), '-')
plt.xlabel('Frequence (Hz)')
plt.ylabel('TF')
```

[4]: Text(0, 0.5, 'TF')



[5]:
```
plt.plot(np.fft.fftshift(freq), np.fft.fftshift(np.abs(amplitude_tilde)), '-')
plt.xlim(10, 30)
plt.xlabel('Frequence (Hz)')
plt.ylabel('TF')
plt.title('Zoom sur la fréquence de rotation du tambour')
```

[5]:
```
Text(0.5, 1.0, 'Zoom sur la fréquence de rotation du tambour')
```

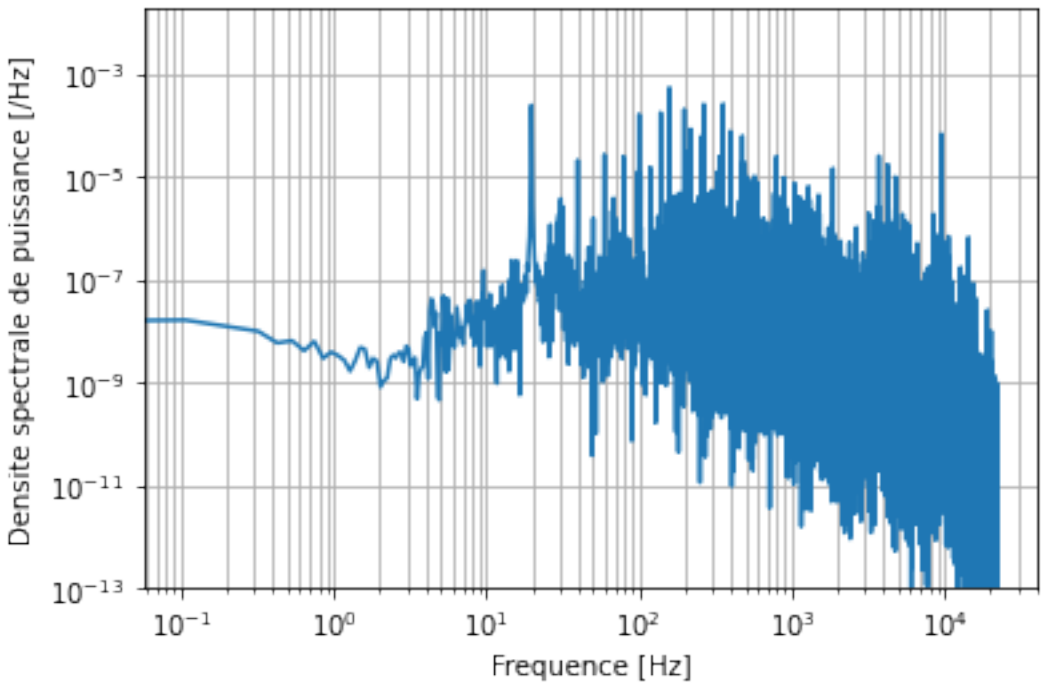


1.4 Densité spectrale de puissance

[6]:
```
from scipy.signal import periodogram

freq, psd = periodogram(amplitude, samplerate)
```

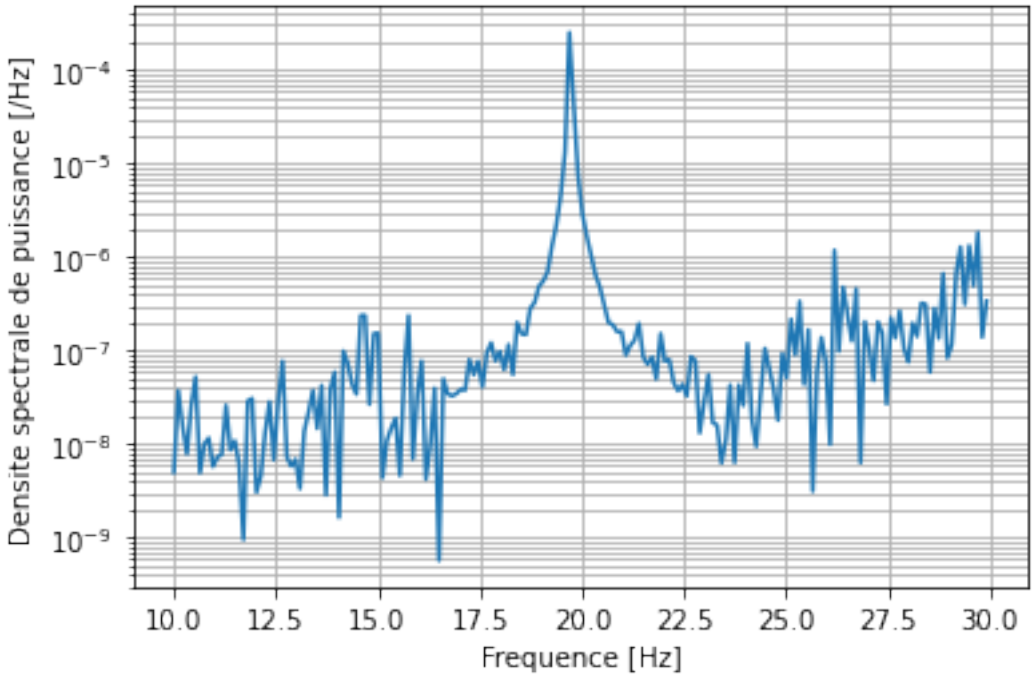
[7]:
```
#mask = (freq>=10) & (freq<50)
plt.loglog(freq, psd)
plt.grid(which='both')

plt.ylim(1E-13, None)

plt.xlabel('Frequence [Hz]')
plt.ylabel('Densite spectrale de puissance [/Hz]');
```

[7]:
```
Text(0, 0.5, 'Densite spectrale de puissance [/Hz]')
```



[8]:
```
mask = (freq>=10) & (freq<30)
plt.semilogy(freq[mask], psd[mask])
plt.grid(which='both')

plt.xlabel('Frequence [Hz]')
plt.ylabel('Densite spectrale de puissance [/Hz]');
```

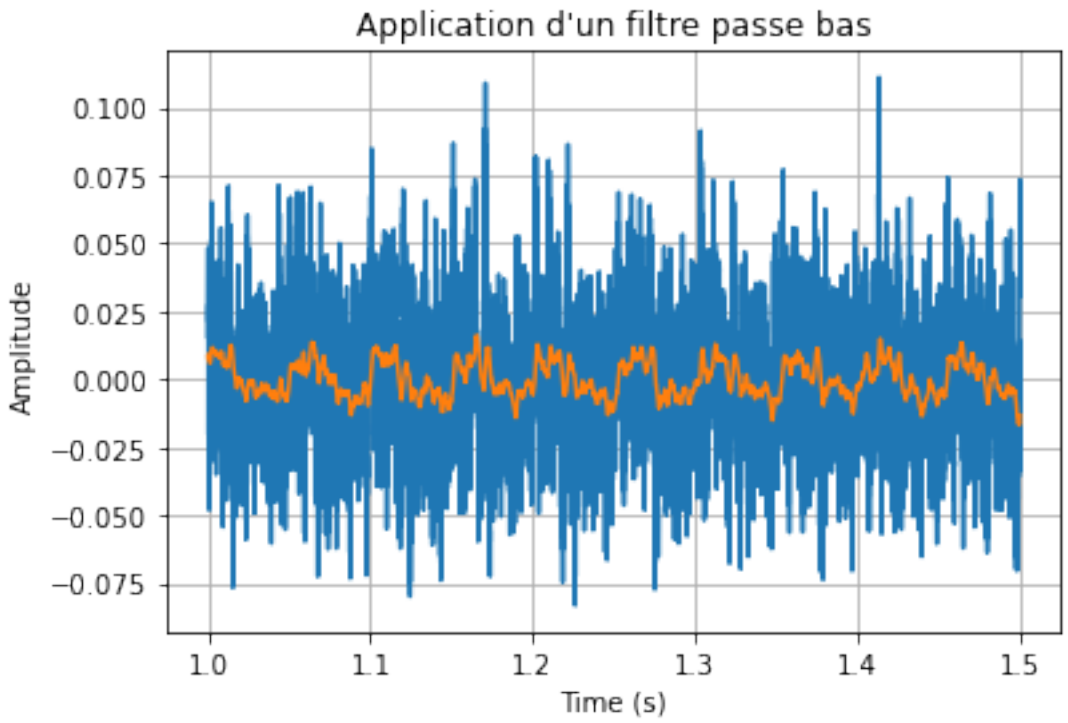
[8]:
```
Text(0, 0.5, 'Densite spectrale de puissance [/Hz]')
```



1.5 Filtrage

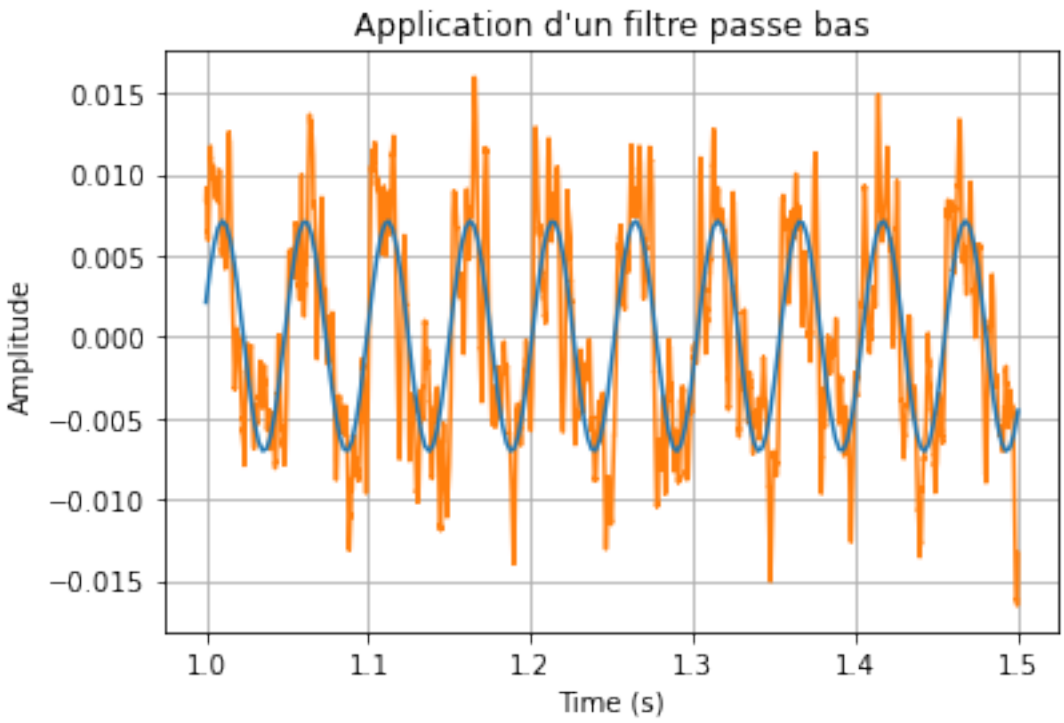
[29]:
```
# Utilisation de la FFT pour faire un filtre :
# On utiliser rfft et irfft car signaux réels
def passe_bas(signal, f_c, samplerate=44100):
    signal_tilde = np.fft.rfft(signal)
    freqs = np.fft.rfftfreq(len(signal), 1/samplerate)
    H = 1/(1+1J*(freqs/f_c))
    signal_2 = np.fft.irfft(signal_tilde*H)
    return signal_2
```

[43]:
```
mask = (t>1) & (t<1.5)

plt.plot(t[mask], amplitude[mask])
plt.plot(t[mask], passe_bas(amplitude, f_c=30)[mask])
plt.title("Application d'un filtre passe bas")
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid()
```



[48]:
```
# Pour s'amuser

from scipy.optimize import curve_fit

def modele(t, amplitude, offset, frequence, phase):
    return offset + amplitude*np.sin(2*np.pi*t*frequence + phase)

plt.plot(t[mask], passe_bas(amplitude, f_c=30)[mask], 'C1')
t_plot = np.linspace(1, 1.5, 201)

p0 = (0.01, 0, 20, 0)
popt, pcov = curve_fit(modele, t[mask], passe_bas(amplitude, f_c=30)[mask], p0)
plt.plot(t_plot, modele(t_plot, *popt))

plt.title("Application d'un filtre passe bas")
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid()

print(f'Fréquence : {popt[2]:.2f} Hz')
```

```
Fréquence : 19.68 Hz
```


Application d'un filtre passe bas
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
Amplitude
1.0
1.1
1.2
1.3
1.4
1.5
Time (s)

[ ]: