project

November 11, 2025

Progetto: Tentativo di rimuovere eventuali rumori da tracce registrate mediante un giradischi

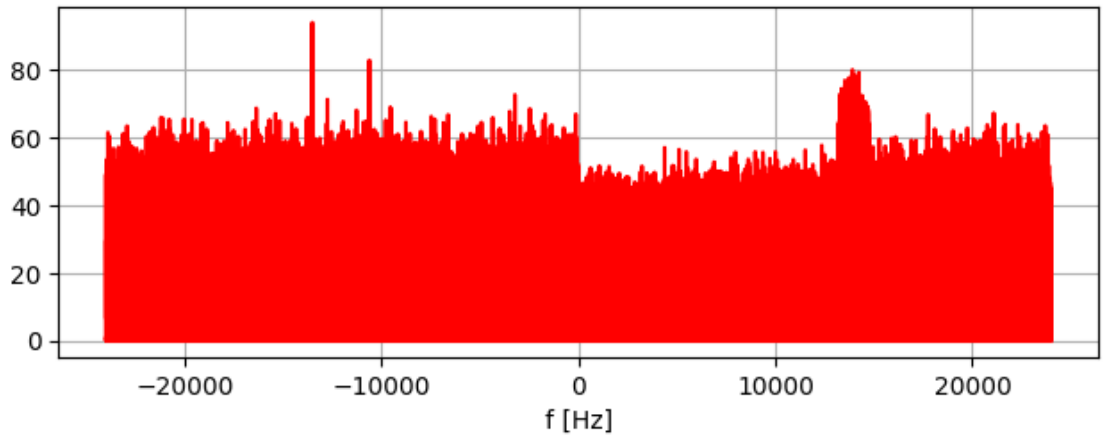
[107]:
```
# Import
import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile as wav
import IPython
import warnings

import librosa as lb
import soundfile as sf
```

[108]:
```
# Costanti
FS = 48000 # frequenza di campionamento dei file del dataSet
NOISE_SECTION = 4 # Ultimi dieci secondi del brano = rumore puro
N_FFT = 2048 # Dimensione della finestra della STFT
HL = 512 # Overlap-Add, dopo aver analizzato i primi 2048 campioni non passo ai␣
 ↪2048 successivi
        # ma ai 2048/4 successivi (512), necessario per non creare artefatti␣
 ↪(sovrapposizione)

track_wav="dataSet/Everything_In_Its_Right_Place-Radiohead.wav"
noise_wav="dataSet/noise.wav"
OUTPUT="dataSet/output.wav"
```

Partiamo analizzando il fenomeno rumoroso in se, ottenuto registrando l'output dell'amplificatore senza alcun brano in riproduzione

[109]:
```
# Analizziamo il rumore
nfs, noise = wav.read(noise_wav) # creo l'array

N = len(noise) # Lunghezza dell'array noise (dei campioni)

# Proviamo a calcolare la trasformata di Fourier con la funzione np.fft.fft di␣
 ↪numpy
nf = nfs/N * np.arange(N) # asse delle frequenze
FFT_noise = np.fft.fft(noise)/N # DFT dell'intero array
noise_freq = np.fft.fftfreq(N, 1/nfs) # Vettore delle frequenze
```

```
plt.subplot(2,1,2)
plt.plot(noise_freq,np.abs(FFT_noise),'-r',label="$X_f(f)$")
plt.xlabel('f [Hz]')
plt.tight_layout()
plt.grid()
```

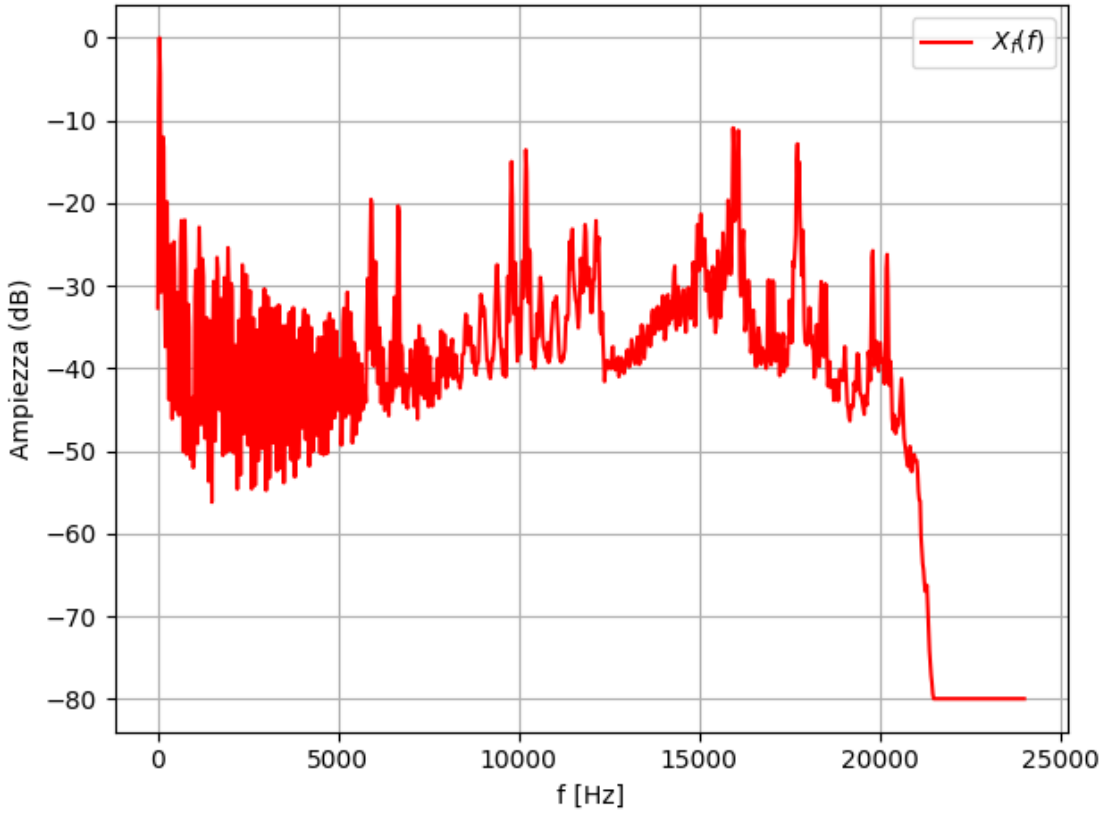


L'approccio con una singola Trasformata di Fourier (FFT) sull'intero brano è inadatto, perché tratta il segnale come statico. Così facendo, si ottiene uno spettro che ha un'altissima risoluzione in frequenza, ma perdiamo completamente i riferimenti temporali. Si perde tutta l'informazione su quando una frequenza appare.

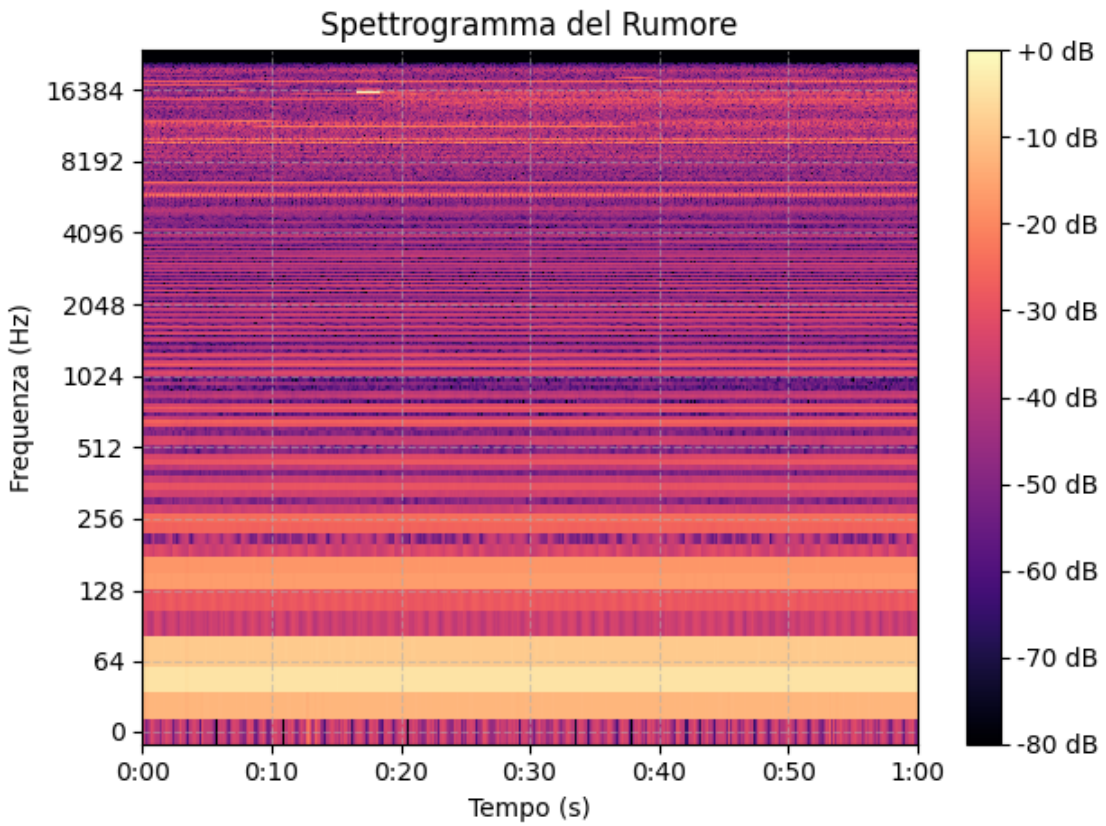
Dato che la musica è un segnale non-stazionario (che evolve continuamente), dobbiamo passare a una Short-Time Fourier Transform (STFT). Analizzando il segnale a fette cosi' mantenendo il legame frequenza-tempo

[110]:
```
noise, sr = lb.load(noise_wav, sr=FS) # come per wav.read, ma crea un array di␣
↪Float piuttosto che di Interi con Frequenza di Campionamento pari a FS =␣
↪48000Hz
STFT_noise = lb.stft(noise, n_fft = N_FFT, hop_length=HL) # eseguo la␣
↪trasformata di Fourier (STFT) con le finestre prescelte

noise_mod = np.abs(STFT_noise) # considero il modulo della trasformata (freq,␣
↪time)
noise_profile = np.mean(noise_mod, axis=1) # esegue la media di tutti i moduli␣
↪delle stft generate, con axis=1 specifico che considero le frequenze

f = lb.fft_frequencies(sr=sr, n_fft=N_FFT) # asse delle frequenze
noise_profile_db = lb.amplitude_to_db(noise_profile, ref=np.max) # converte i␣
↪float delle frequenze dell'array noise in decibel
```

```
plt.plot(f, noise_profile_db,'-r',label="$X_f(f)$")
plt.xlabel('f [Hz]')
plt.ylabel('Ampiezza (dB)')
plt.legend()
plt.tight_layout()
plt.grid()
```



[111]:
```
# Spettrogramma del Rumore:
noise_mod_db = lb.amplitude_to_db(noise_mod, ref=np.max)
lb.display.specshow(noise_mod_db, sr=sr, hop_length=HL, x_axis='time',␣
 ↪y_axis='log')

plt.colorbar(format='%+2.0f dB')
plt.title('Spettrogramma del Rumore')
plt.xlabel('Tempo (s)')
plt.ylabel('Frequenza (Hz)')
plt.tight_layout()
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```



Notiamo dallo spettrogramma come i rumori si annidano anche nelle frequenze piu' basse.

Ora procediamo con l'analisi del brano rumoroso in se

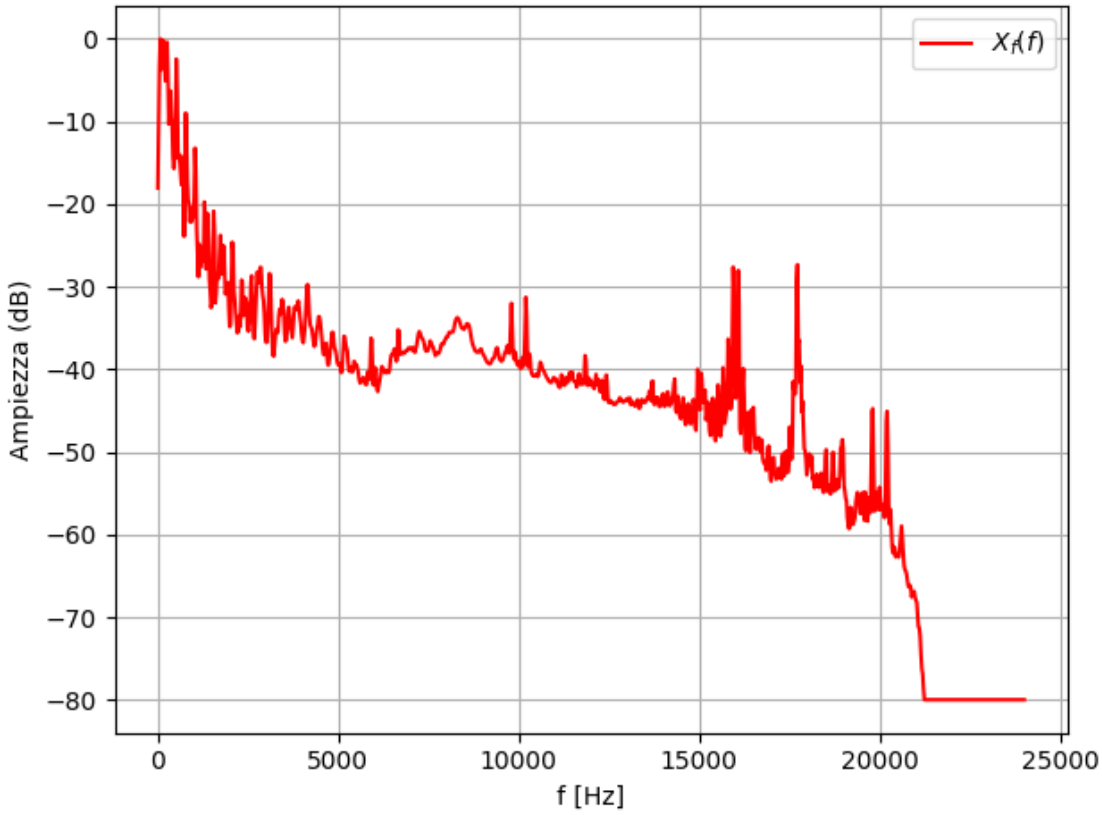
[112]:
```
# Riproduciamo il brano di partenza contenente il rumore
IPython.display.Audio(track_wav)
```

[112]:
```
<IPython.lib.display.Audio object>
```

[113]:
```
# Analizziamo il Brano
track, sr = lb.load(track_wav, sr=FS) # Questo comando carica il segnale e␣
 ↪restituisce : frequenza_campionamento, segnale (array float)
STFT_track = lb.stft(track, n_fft = N_FFT, hop_length=HL) # eseguo la␣
 ↪trasformata di Fourier (STFT) con le finestre prescelte

track_mod = np.abs(STFT_track) # considero il modulo della trasformata (freq,␣
 ↪time)
track_profile = np.mean(track_mod, axis=1) # esegue la media di tutti i moduli␣
 ↪delle stft generate, con axis=1 specifico che considero le frequenze

f = lb.fft_frequencies(sr=sr, n_fft=N_FFT) # asse delle frequenze
```

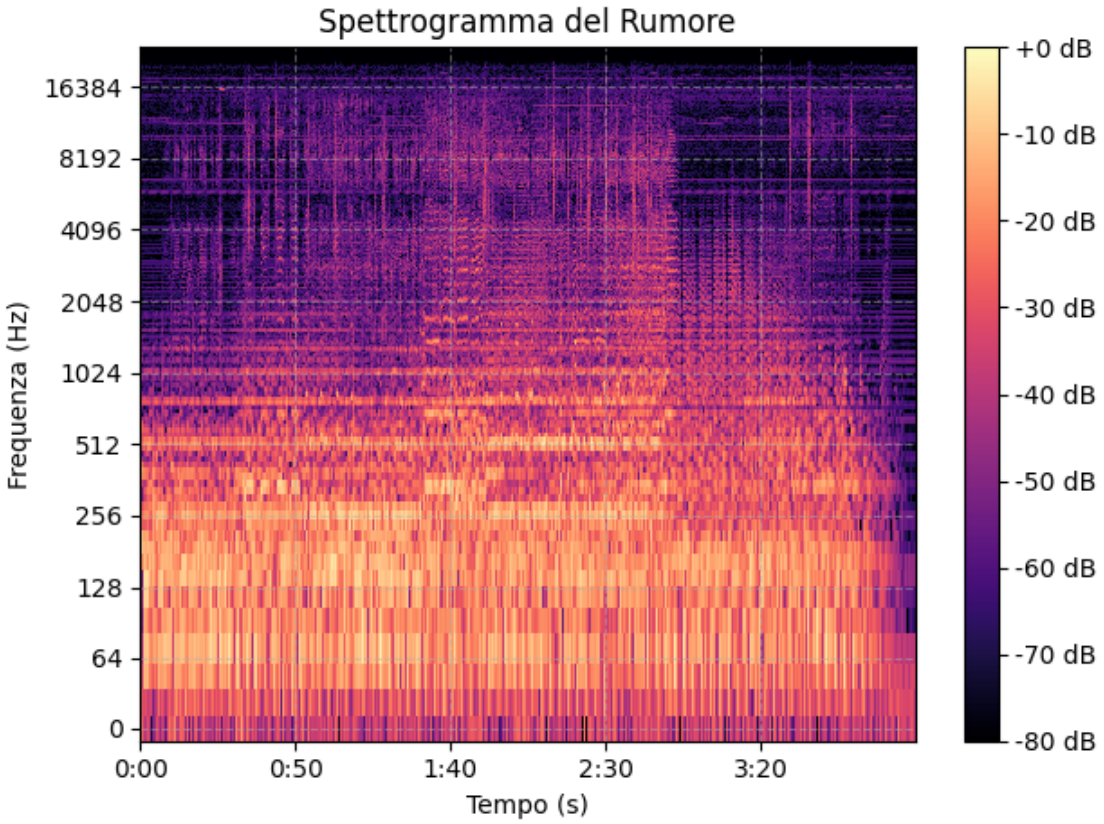
```
track_profile_db = lb.amplitude_to_db(track_profile, ref=np.max) # converte i␣
 ↪float delle frequenze dell'array noise in decibel
```

[114]:
```
plt.plot(f, track_profile_db,'-r',label="$X_f(f)$")
plt.xlabel('f [Hz]')
plt.ylabel('Ampiezza (dB)')
plt.legend()
plt.tight_layout()
plt.grid()
```



[115]:
```
# Spettrogramma del Brano:
track_mod_db = lb.amplitude_to_db(track_mod, ref=np.max)
lb.display.specshow(track_mod_db, sr=sr, hop_length=HL, x_axis='time',␣
 ↪y_axis='log')

plt.colorbar(format='%+2.0f dB')
plt.title('Spettrogramma del Rumore')
plt.xlabel('Tempo (s)')
plt.ylabel('Frequenza (Hz)')
plt.tight_layout()
```

```
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```

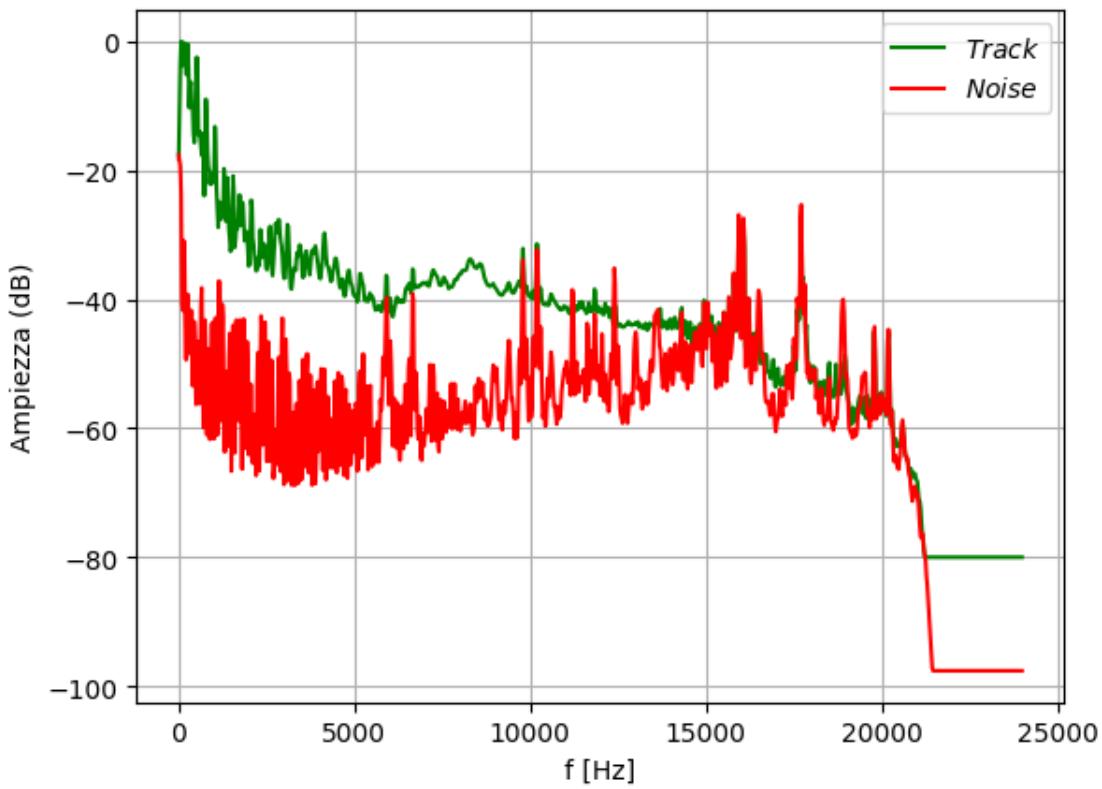


Abbiamo due Approcci, o utilizziamo il rumore in se come riferimento, rischiando di avere registrazioni effettuate in condizioni diverse oppure quindi per assicurarci che Rumore e Brano siano normalizzati fra di essi aggiungiamo la condizione che i file in input terminino con dei secondi di rumore puro cosi' saremo in grado di estrapolare tale sezione assicurandoci cosi' di avere gli stessi parametri dell'amplificatore (volume)

[116]:
```
# Estraiamo il rumore dagli ultimi NOISE_SECTION secondi del brano
noise_samples = int(NOISE_SECTION * sr)

# Dividiamo il brano in due: Musica e Rumore Finale
noise_part = track[-noise_samples:]
music_part = track[:-noise_samples]

# Eseguiamo STFT del brano e del Rumore
STFT_track = lb.stft(music_part, n_fft=N_FFT, hop_length=HL)
STFT_noise = lb.stft(noise_part, n_fft = N_FFT, hop_length=HL)
```

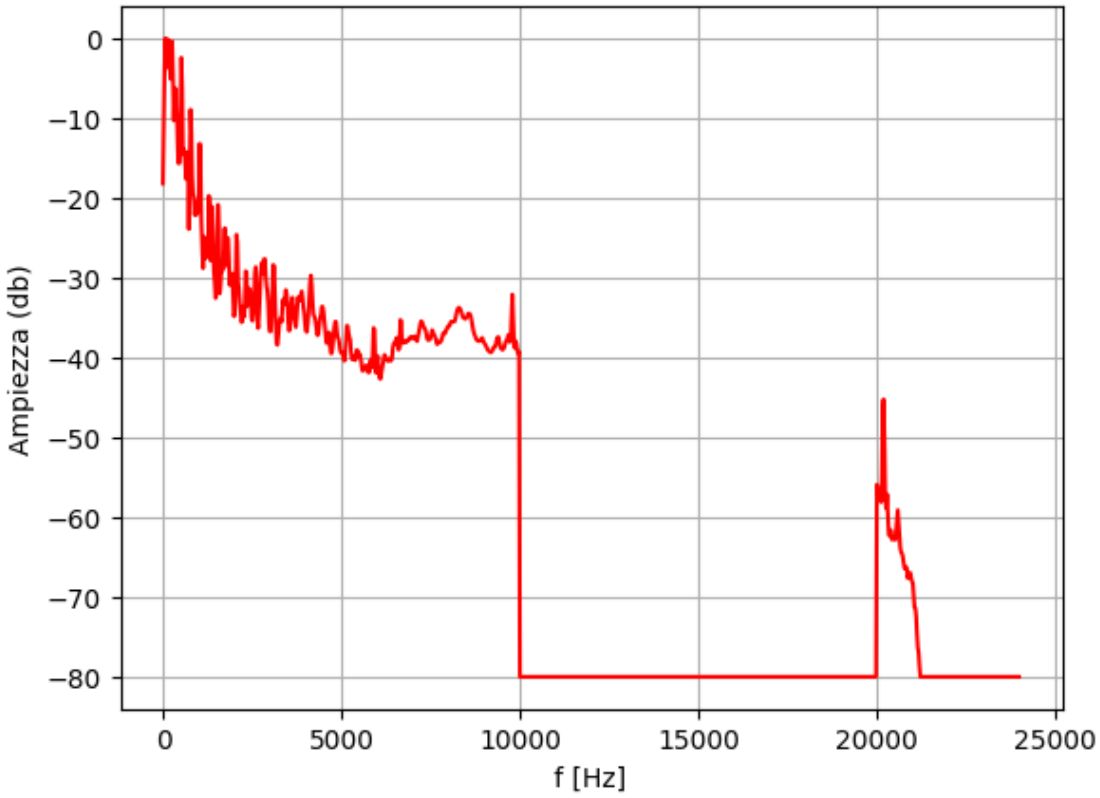
[117]:
```
track_mod = np.abs(STFT_track) # calcoliamo il modulo del brano
track_profile = np.mean(track_mod, axis=1) # esegue la media di tutti i moduli␣
 ↪delle stft generate, con axis=1 specifico che considero le frequenze
noise_mod = np.abs(STFT_noise)
noise_profile = np.mean(noise_mod, axis=1)

f = lb.fft_frequencies(sr=sr, n_fft=N_FFT) # asse delle frequenze
ref_value = max(np.max(track_profile), np.max(noise_profile)) # valore per␣
 ↪normalizzare le due tracce

noise_profile_db = lb.amplitude_to_db(noise_profile, ref=ref_value) # converte␣
 ↪le ampiezze dell'array in decibel
track_profile_db = lb.amplitude_to_db(track_profile, ref=ref_value)
```

[118]:
```
plt.plot(f, track_profile_db,'-g', label="$Track$")
plt.plot(f, noise_profile_db,'-r', label="$Noise$")

plt.xlabel('f [Hz]')
plt.ylabel('Ampiezza (dB)')

plt.legend(loc="upper right")
plt.grid()
```



con questa vista possiamo notare un problema fondamentale, il rumore vive nello stesso range di frequenze del brano dunque l'applicazione di un semplice filtro di sogliatura rimuoverebbe gran parte dei dettagli del brano rendendolo irriconoscibile.

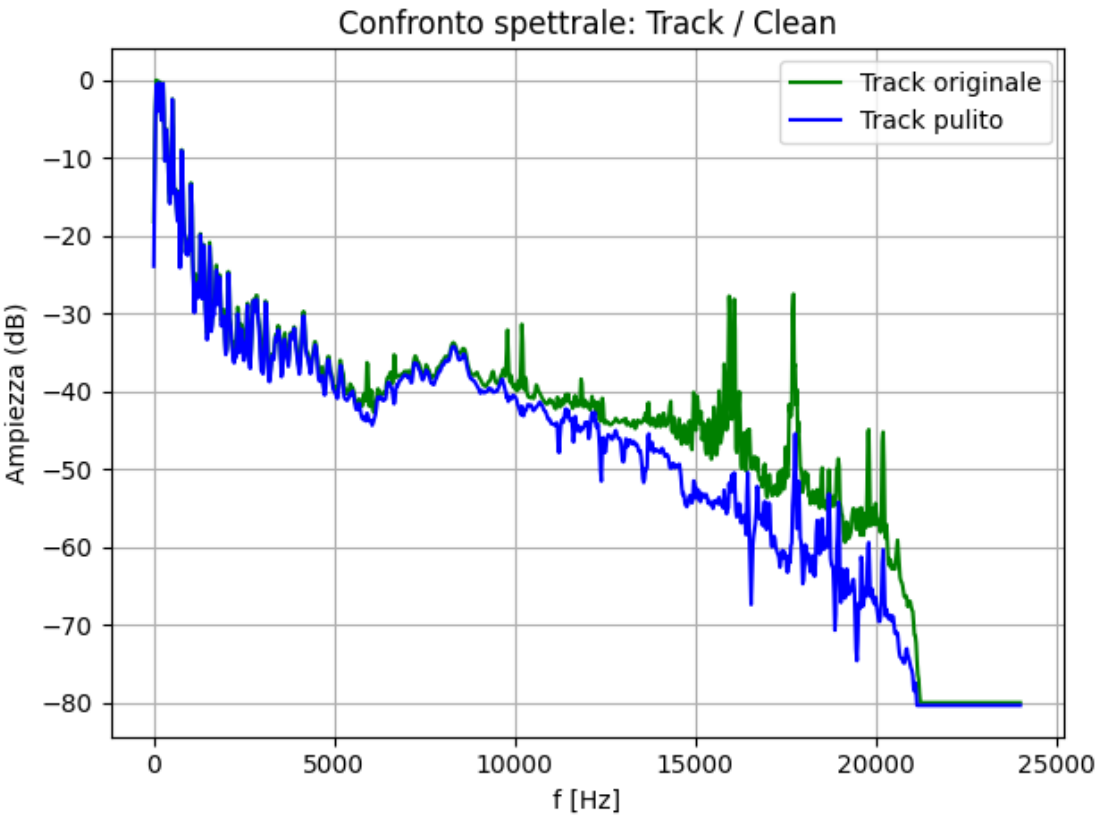
[119]:
```
# Esempio di Sogliatura
thresh_h = 20000
thresh_l = 10000

# find all the indices corresponding to frequencies greather than a threshold
idx = np.argwhere((np.abs(f) < thresh_h) & (np.abs(f) > thresh_l))
STFT_cut=np.copy(STFT_track)
STFT_cut[idx, :] = 0 # imposta l'ampiezza dei valori a zero

track_filt = np.mean(np.abs(STFT_cut), axis=1)
track_filt_db = lb.amplitude_to_db(track_filt, ref=np.max)

plt.plot(f, track_filt_db, '-r', label=f"Spettro Filtrato (- ({thresh_l} <␣
 ↪{thresh_h}) Hz)")
plt.xlabel('f [Hz]')
plt.ylabel('Ampiezza (db)')

plt.grid()
```



Evidentemente quindi non e' possibile eseguire una semplice sogliatura per rimuovere la parte rumorosa

[120]:
```
# Procediamo con la rimozione del Rumore
profile_expanded = noise_profile[:, np.newaxis] # Passiamo da un vettore 1D␣
 ↪(medie frequenze rumorose) ad una versione 2D
# Sottrazione Spettrale
mod_clean = np.maximum(track_mod - profile_expanded, 0) # Dove era presente␣
 ↪noise > track il rumore viene azzerato

track_phase = np.angle(STFT_track) # Otteniamo la fase (parte immaginaria) del␣
 ↪brano
clean_STFT = mod_clean * np.exp(1j * track_phase)

track_clean = lb.istft(clean_STFT, hop_length=HL) # Torniamo nel dominio del␣
 ↪tempo

# Output
sf.write(OUTPUT, track_clean, sr, subtype='PCM_24')
```

[121]:
```
# Analisi spettrale del segnale pulito
STFT_clean = lb.stft(track_clean, n_fft=N_FFT, hop_length=HL)
clean_mod = np.abs(STFT_clean)
clean_profile = np.mean(clean_mod, axis=1)

f = lb.fft_frequencies(sr=sr, n_fft=N_FFT)
ref_value = max(np.max(track_profile), np.max(clean_profile)) # valore per␣
 ↪normalizzare le due tracce

clean_profile_db = lb.amplitude_to_db(clean_profile, ref=ref_value)

# Plot confronto spettrale
plt.figure()
plt.plot(f, track_profile_db, '-g', label='Track originale')
plt.plot(f, clean_profile_db, '-b', label='Track pulito')
plt.xlabel('f [Hz]')
plt.ylabel('Ampiezza (dB)')
plt.title('Confronto spettrale: Track / Clean')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```


Confronto spettrale: Track / Clean
Track originale
Track pulito
Ampiezza (dB)
0
−10
−20
−30
−40
−50
−60
−70
−80
0
5000
10000
15000
20000
25000
f [Hz]