Università degli Studi di Perugia

Dipartimento di Matematica e Informatica

Report di Laboratorio – Tecniche di Acquisizione Dati

Garage Band

Studenti

Leonardo Nicoletta

Filippo Notari

Nicolò Tittarelli

Anno Accademico 2023-2024

Sommario

Introduzione

In questa esercitazione vengono eseguite operazioni di elaborazione audio, tra cui lettura, filtraggio, analisi in frequenza e visualizzazione di segnali audio. Utilizzeremo librerie come ‘soundfile’, ‘matplotlib’, ‘scipy’, ‘numpy’, ‘librosa’ e ‘soundcard’. Nel primo capitolo verranno raccolti i dati del file audio ('diapason.wav') e calcolata la durata. Successivamente, saranno estratti i canali sinistro e destro del segnale audio, calcolando la Trasformata di Fourier (FFT) per entrambi. Nel secondo capitolo mostreremo come identificare i picchi nello spettro di potenza e convertire le frequenze di picco in note musicali. Nel terzo capitolo, viene definita una classe Filter che implementa un filtro passa-banda basato su frequenze specifiche. Il filtraggio dei segnali viene eseguito creando e applicando filtri passa-banda ai segnali FFT. I segnali filtrati vengono ricostruiti, normalizzati e salvati in un nuovo file audio ('filter.wav').

Capitolo 1. Raccolta dati e trasformata

1.1 Importazione File audio

Specificando il percorso del file, selezionando il file audio desiderato ed aprendolo successivamente con la funzione read() del modulo SoundFile riusciamo ad ottenere la variabile data, che contiene le informazioni relative ai canali sinistro e destro del suono, e fs ovvero la frequenza di campionamento.
La durata dell'audio, quindi, sarà calcolata dividendo il numero totale di campioni per la frequenza di campionamento. Per Semplicità e chiarezza verranno mostrati solamente i risultati relativi al canale sinistro dell'audio "diapason.wav", nonostante anche gli altri file siano stati processati allo stesso modo.

```
#read file and get duration
path_to_audio = ['diapason.wav','pulita_semplice.wav','distorta.wav']
selected_audio = path_to_audio[0]
data, fs = sf.read(selected_audio)
f = sf.SoundFile(selected_audio)
duration = f.frames/f.samplerate
```

1.2 Scrittura copia File audio

Usando la funzione write() del modulo SoundFile verrà scritto su disco un nuovo file dato il nome, i dati del suono e la frequenza di campionamento.

```
#write new audio file
sf.write('new_diapason.wav',data,fs)
```

1.3 Calcolo trasformata di Fourier a partire dal File audio

La variabile 'Data' sarà diviso quindi nei canali sinistro e destro, in seguito tramite la funzione fft.rfftfreq() e la funzione np.linspace() vengono calcolati rispettivamente il dominio della frequenza e il dominio del tempo.

Infine, la funzione fft.rfft() restituisce in 'fft_sx' e 'fft_dx' le trasformate di Fourier del canale sinistro e del canale destro, che per comodità sono concatenate in una lista.

```
#select channel, get fft and plot all
channel_sx = data[:,0]
channel_dx = data[:,1]
freq = fft.rfftfreq(len(channel_sx),1./fs)
t = np.linspace(0,duration,len(channel_sx))
fft_sx = fft.rfft(channel_sx,norm='forward')
fft_dx = fft.rfft(channel_dx,norm='forward')
ffts = [fft_sx,fft_dx]
```

I dati ottenuti fin ora permettono di tracciare i seguenti grafici:

Fig 1.1 suono nel dominio del tempo

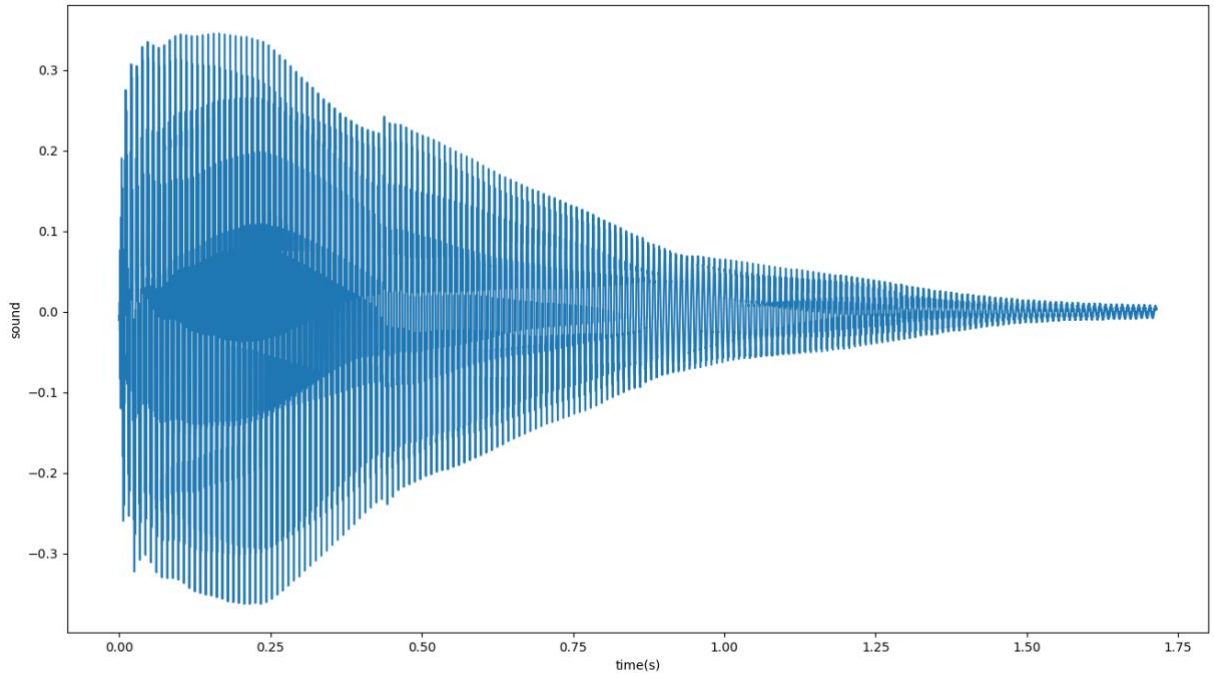


Fig 1.2 potenza del segnale nel dominio della frequenza

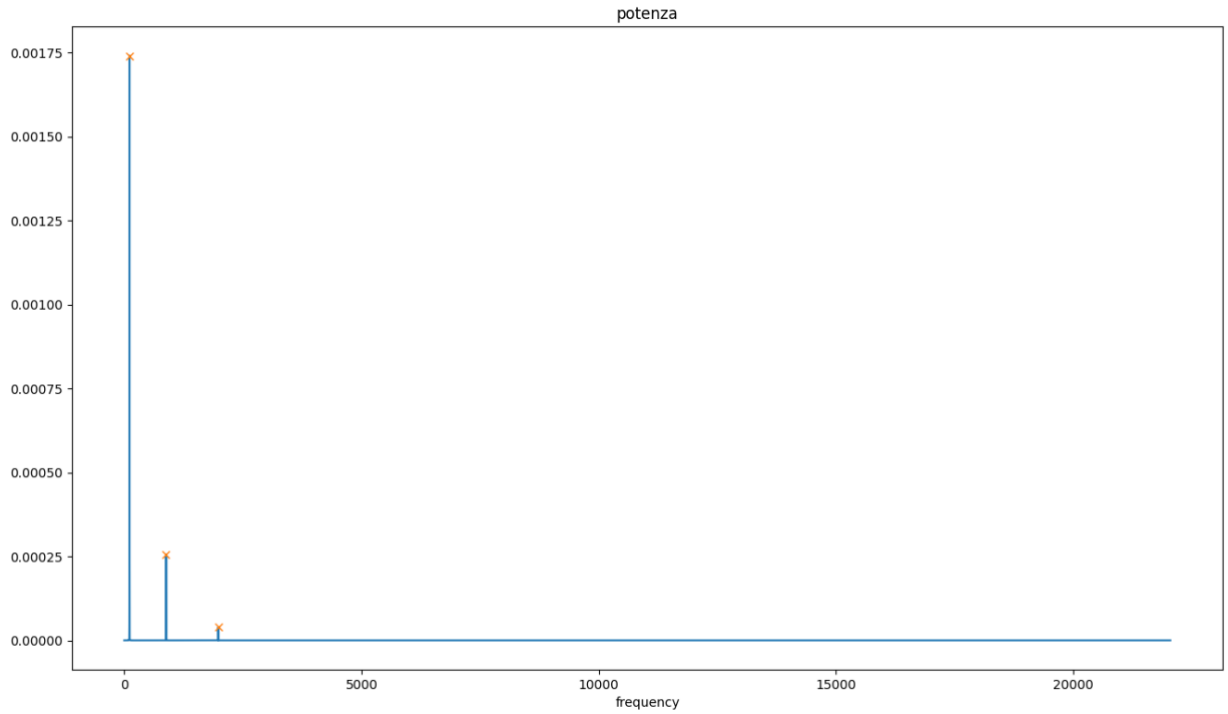


Fig 1.3 parte reale del segnale nel dominio della frequenza

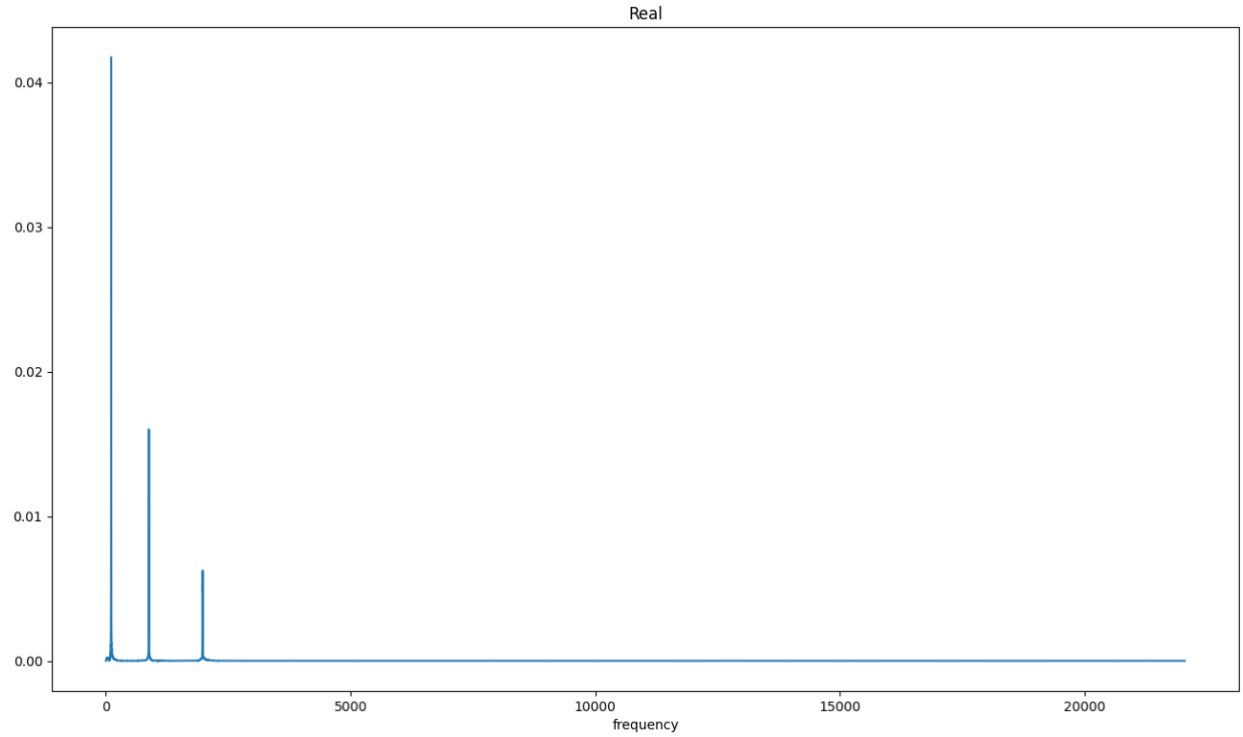
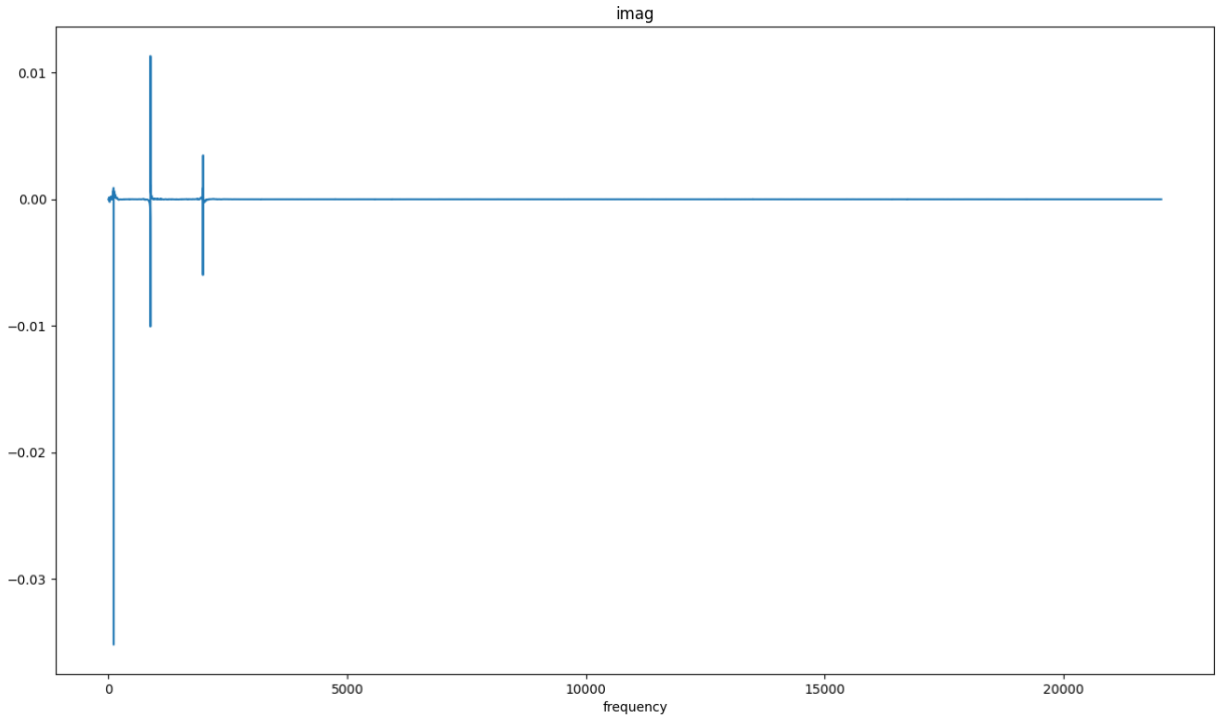


Fig 1.4 parte immaginaria del segnale nel dominio della frequenza



Capitolo 2. Selezione delle frequenze

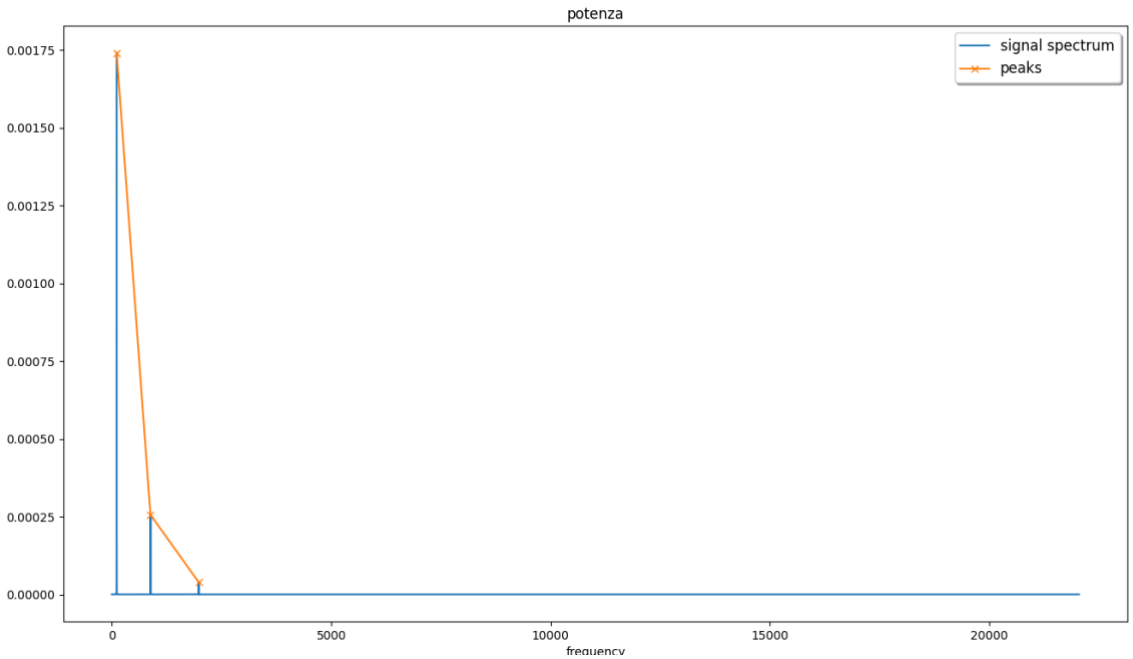
2.1 Trovare i picchi nello spettro di potenza

```
#find peak and peaks width

peaks = [signal.find_peaks(abs(i)**2, height = 2.8e-5, rel_height = .99)[0] for i in ffts]
peak_heights = [signal.find_peaks(abs(i)**2, height = 2.8e-5, rel_height = .99)[1]['peak_heights'] for i in ffts]
peaks_widths = [signal.peak_widths(abs(i)**2,j, rel_height = .99)[0] for i,j in zip(ffts, peaks)]
left_peaks , right_peaks = [signal.peak_widths(abs(i)**2,j, rel_height = .99)[2] for i,j in zip(ffts, peaks)],\
[signal.peak_widths(abs(i)**2,j, rel_height = .99)[3] for i,j in zip(ffts, peaks)]
```

La funzione signal.find_peaks() del modulo scipy.signal restituisce gli indici della lista in cui ci sono dei picchi, i parametri con cui lavora sono: la potenza del segnale, height, che sarebbe un filtro minimo sull' altezza dei picchi e rel_height che permette di selezionare a quale altezza del picco in percentuale verrà poi calcolata la larghezza. La stessa funzione restituisce anche l'altezza dei picchi. Nelle righe rimanenti viene utilizzata la funzione signal.peak_widths del modulo scipy.signal per analizzare la larghezza dei picchi nelle trasformate di Fourier. Il codice è strutturato in modo da calcolare le larghezze dei picchi, nonché le posizioni dei limiti sinistro e destro di ciascun picco che faciliteranno la costruzione di un filtro passa/elimina banda.

Fig 2.1



In figura sono rappresentati in arancione i picchi rilevati dalla funzione signal.find_peaks() e in blu lo spettro di potenza del segnale

2.2 Conversione dei picchi in note musicali

In Python grazie al modulo librosa risulta molto semplice la traduzione da frequenze a note musicali:

```
notes = [librosa.hz_to_note(freq[i]) for i in peaks]
```

Note corrispondenti ai picchi: ['A2', 'A5', 'B6']

2.3 Selezionare banda di frequenze dato il picco con maggiore potenza

Prima di tutto tramite np.argmax() si trova l'indice per cui peak_heights assume il valore massimo, isolando di fatto il picco principale. Inserendo l'indice appena calcolato nelle altre liste si ottengono la larghezza, i limiti inferiore e superiore della frequenza del picco principale.

```
#find main peak
target_freq_index = [np.argmax(i) for i in peak_heights]
target_left_delim =[]
target_right_delim =[]

for i in range (2):
target_width = peak_heights[i][target_freq_index[i]]
target_left_delim.append(left_peaks[i][target_freq_index[i]])
target_right_delim.append(right_peaks[i][target_freq_index[i]])
```

Capitolo 3. Applicazione del filtro

3.1 Costruzione del filtro

Nel seguente codice viene definita la classe Filter che è caratterizzato dai seguenti attributi:

- Freq: il dominio della frequenza, che il filtro condivide con il segnale che si vuole filtrare.
- Lb: il limite sinistro del filtro.
- Rb: il limite destro del filtro.
- Mode: 1 se si vuole costruire un filtro passa banda, 0 per un filtro elimina banda.
- Signal: il vero e proprio filtro che sarà applicato al segnale da filtrare.

```
class Filter():

def __init__(self, freq, left_bound, right_bound, mode) -> None:
self.freq = freq
self.lb = left_bound
self.rb = right_bound
self.mode = mode #mode 1 se filtro passa banda, 0 se filtro elimina banda
self.signal = [1 if ((i> abs(freq[int(self.lb)])) and i < abs(freq[int(self.rb)])) else 0 for i in self.freq] if self.mode
else\
[0 if ((i> abs(freq[int(self.lb)])) and i < abs(freq[int(self.rb)])) else 1 for i in self.freq]

filters = [Filter(freq,i,j,1) for i,j in zip(target_left_delim,target_right_delim)]
```

Quindi, nella variabile ‘filter’, più precisamente nel suo attributo ‘signal’, si trova un segnale che vale 1 (0), tra il limite sinistro e il limite destro, e 0 (1) nel resto del dominio, che corrisponde ad un filtro passa (elimina) banda.

3.2 Applicazione del filtro

Quando si hanno segnali nel dominio della frequenza l'applicazione di un filtro diventa una semplice moltiplicazione perché la convoluzione di due segnali nel dominio nel tempo corrisponde al prodotto delle trasformate nel dominio della frequenza.

```
fft_filtered_signal = [i*j.signal for i,j in zip(ffts, filters)]
```

3.3 Ricostruzione File audio filtrato

```
filtered_signal = [fft.irfft(i).real for i in fft_filtered_signal]
max_abs_values = [np.max(np.abs(channel)) for channel in filtered_signal]
normalized_channels = [channel / max_abs_value for channel, max_abs_value in zip(filtered_signal, max_abs_values)]
audio_data = np.column_stack(normalized_channels)
scaled_data = np.int16(audio_data * 32767)
sf.write('filter.wav',scaled_data, fs)
```

Per ricostruire un nuovo file audio sulla base del segnale filtrato occorre prima di tutto calcolare la trasformata inversa del segnale nel dominio della frequenza in modo da riportarlo nel dominio del tempo. In seguito, ci sono le operazioni di normalizzazione e moltiplicazione per uno scalare dei due canali di dati che permettono di avere dei valori in un intervallo prestabilito, in questo caso 32767 che corrisponde al massimo valore rappresentabile da un intero a 16-bit. Infine, con la funzione sf.write() specificando percorso del file, i dati e la frequenza di campionamento otteniamo il file filtrato “filter.wav”.

Conclusioni

In questo report abbiamo illustrato un processo completo di acquisizione, elaborazione e analisi di segnali audio utilizzando Python. Il codice sviluppato permette di leggere file audio, analizzare il contenuto spettrale attraverso la Trasformata di Fourier, individuare le frequenze di picco e applicare filtri passa-banda per isolare specifiche componenti del segnale.