TP: Réponse impulsionnelle de salle

Nicolas THIERRY & Adrien GUYOT

October 3, 2024

Contents

[1]:
```
import sounddevice as sd
import numpy as np
import matplotlib.pyplot as plt
import wavio
import scipy.io as sio
import datetime
from scipy import signal
from scipy.fft import fft, fftfreq
```

1 Simulation de l'effet de réverbération

1. Pour simuler le signal audio dans les différents environnements, on peut utiliser leur réponse impulsionnelle et la convoluer avec notre signal audio de chant enregistré dans une chambre anéchoïque. Ce type d'enregistrement permet de ne pas polluer notre signal d'entré avec la réponse impulsionnelle de la pièce d'enregistrement.

[2]:
```
#vitesse du son
c=340;

# signal émis
name_source = 'assets/singing.wav'

# RI
RI_nashville = 'assets/1st_baptist_nashville_far_wide.wav'
RI_university ='assets/auditorium-university-s1r2_0_1.wav'
RI_sportcentre ='assets/sportscentre_omni.wav';
```

```
# Lecture des données
fs, source = sio.wavfile.read(name_source)
fs_h_nashville, h_nashville = sio.wavfile.read(RI_nashville)
fs_h_university, h_university = sio.wavfile.read(RI_university)
fs_h_sportcentre, h_sportcentre = sio.wavfile.read(RI_sportcentre)

# Conversion en float
source = np.float64(source)
ho_nashville = np.float64(h_nashville)[:,0] # on garde 1 canal si plusieurs␣
  ↪canaux
ho_university = np.float64(h_university)[:,0] # on garde 1 canal si plusieurs␣
  ↪canaux
ho_sportcentre = np.float64(h_sportcentre)

# Réecchantillonage de la RI
# (les deux signaux n'ont pas la même fréquence d'échantillonnage)
h_nashville = signal.resample(ho_nashville, int(len(ho_nashville)*fs/
  ↪fs_h_nashville))
h_university = signal.resample(ho_university, int(len(ho_university)*fs/
  ↪fs_h_university))
h_sportcentre = signal.resample(ho_sportcentre, int(len(ho_sportcentre)*fs/
  ↪fs_h_sportcentre))

singing_nashville = signal.convolve(source, h_nashville, mode='full')
singing_university = signal.convolve(source, h_university, mode='full')
singing_sportcentre = signal.convolve(source, h_sportcentre, mode='full')
```

```
/tmp/ipykernel_52549/2241597930.py:13: WavFileWarning: Chunk (non-data) not
understood, skipping it.
  fs, source = sio.wavfile.read(name_source)
/tmp/ipykernel_52549/2241597930.py:14: WavFileWarning: Chunk (non-data) not
understood, skipping it.
  fs_h_nashville, h_nashville = sio.wavfile.read(RI_nashville)
/tmp/ipykernel_52549/2241597930.py:15: WavFileWarning: Chunk (non-data) not
understood, skipping it.
  fs_h_university, h_university = sio.wavfile.read(RI_university)
/tmp/ipykernel_52549/2241597930.py:16: WavFileWarning: Chunk (non-data) not
understood, skipping it.
  fs_h_sportcentre, h_sportcentre = sio.wavfile.read(RI_sportcentre)
```

2. Une fois les signaux simulés dans les environnements, on peut les écouter afin de comprendre comment l'environnement affecte le signal d'origine.

[3]:
```
# Restitution audio
print('Playing Singing in Nashville environnement')
sd.play(singing_nashville/max(singing_nashville),fs,blocking=True)
```

```
Playing Singing in Nashville environnement
```

[4]:
```python
# Restitution audio
print('Playing Singing in University environnement')
sd.play(singing_university/max(singing_university),fs,blocking=True)
```

```
Playing Singing in University environnement
```

[5]:
```python
# Restitution audio
print('Playing Singing in Sportcentre environnement')
sd.play(singing_sportcentre/max(singing_sportcentre),fs,blocking=True)
```

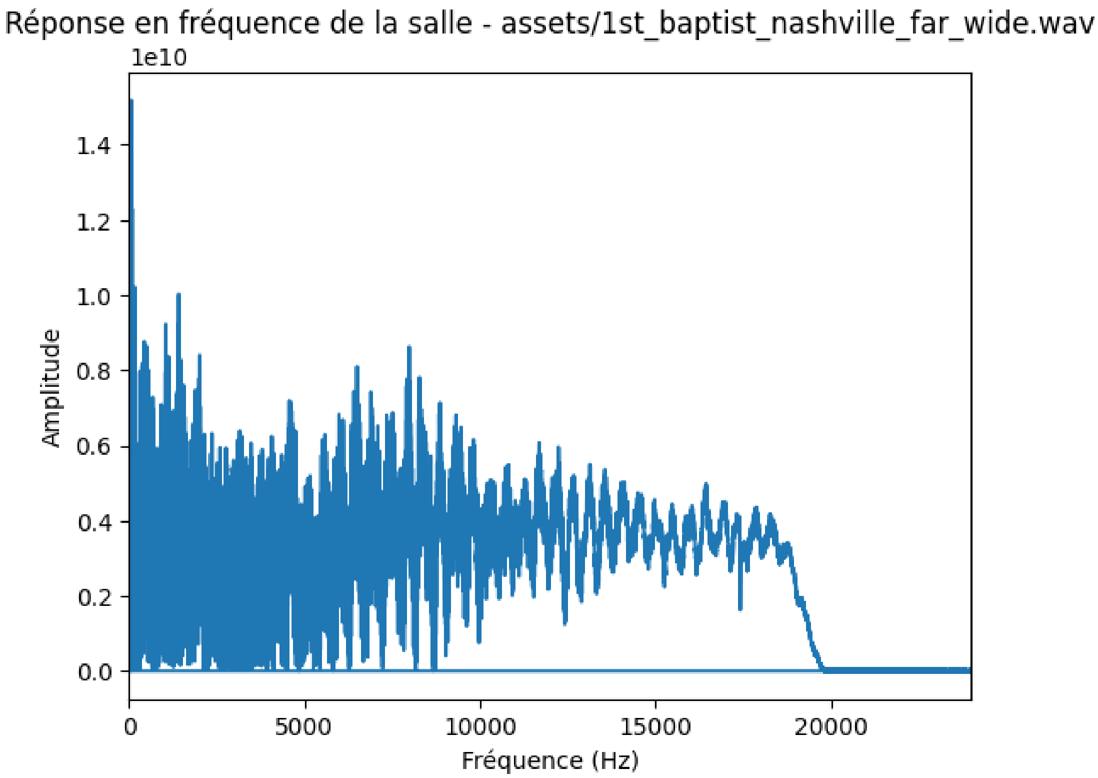
```
Playing Singing in Sportcentre environnement
```

On remarque qu'en fonction des différents environnements l'effet de réverbération est très différent. Plus la pièce est un espace vide plus l'effet est notable. Par exemple, c'est dans le gymnase qu'il est le plus fort et à l'inverse il est le moins prononcé dans l'auditorium d'université. On peut aussi remarquer qu'un effet de réverbération trop important limite la clarté et donc la compréhension du son par l'observeur.

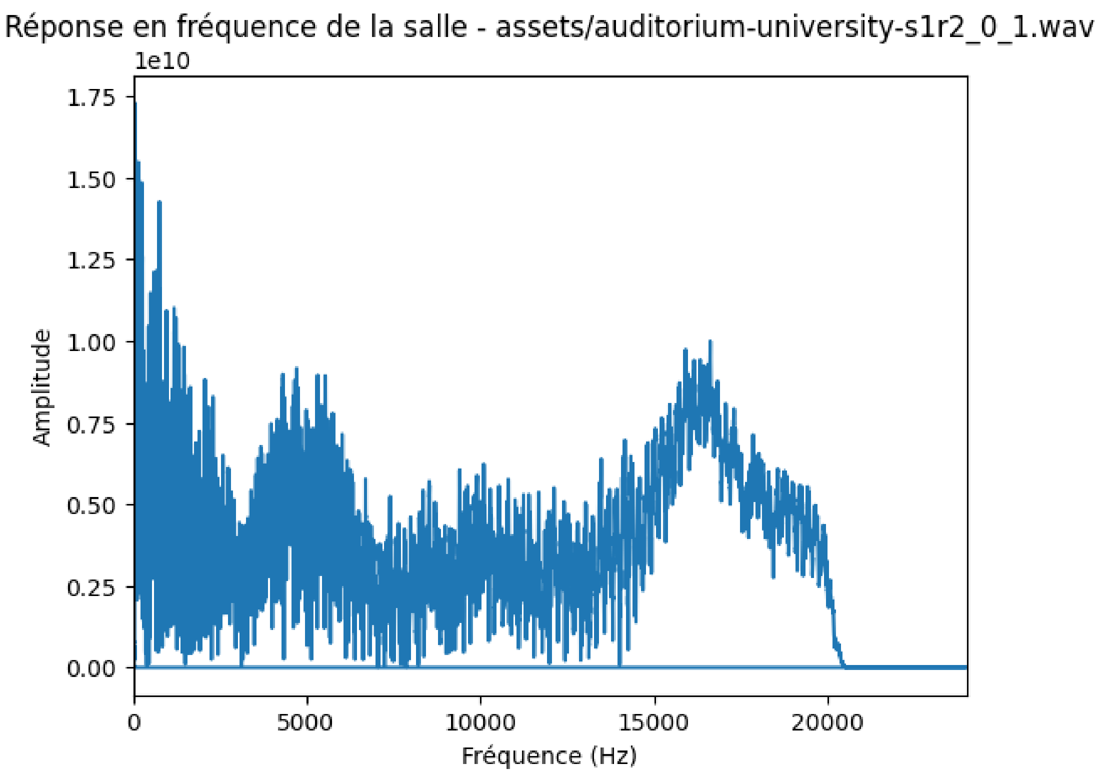
3. Pour expliquer ces différences, on peut analyser les réponses en fréquences des différents environnements.

[6]:
```python
def show_RI_fft(ri: np.ndarray, name: str):
    # FFT de la RI
    fft_result = fft(ri,len(ri)*10)
    fft_freqs = fftfreq(len(ri)*10, 1/fs)  # Fréquences correspondantes
    fft_amplitude = np.abs(fft_result)

    plt.figure()
    plt.plot(fft_freqs, fft_amplitude)
    plt.title("Réponse en fréquence de la salle - " + name)
    plt.xlabel("Fréquence (Hz)")
    plt.ylabel("Amplitude")
    plt.xlim(0, fs/2)
    plt.show()
```

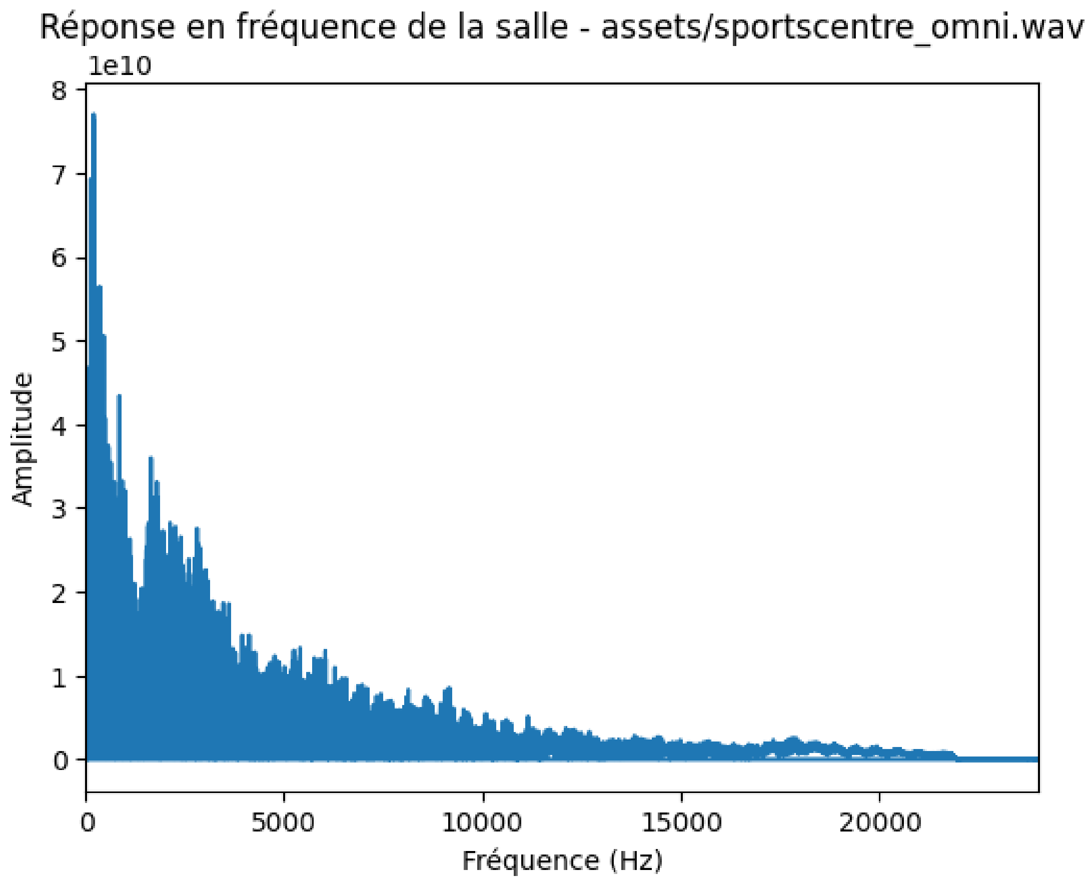
[7]:
```python
show_RI_fft(h_nashville, RI_nashville)
show_RI_fft(h_university, RI_university)
show_RI_fft(h_sportcentre, RI_sportcentre)
```



Réponse en fréquence de la salle - assets/1st_baptist_nashville_far_wide.wav
1e10
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Amplitude
0
5000
10000
15000
20000
Fréquence (Hz)



Réponse en fréquence de la salle - assets/auditorium-university-s1r2_0_1.wav
1e10
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Amplitude
0
5000
10000
15000
20000
Fréquence (Hz)



On peut voir que les réponses sont très différentes. En effets la salle de sport amplifie les basses fréquences et ne semble pas avoir d'objectif particulier. A l'inverse, dans l'église, on voit une amplification moins forte mais plus constante dans la plage de sensibilité de l'oreil humaine. En vu du lieu, cette réponse semble être voulu pour avoir une propagration agréable et audible d'un signal émis (un chant par exemple). Il en est de même pour l'auditorium d'université, même si on remarque que les amplifications sont centrés sur d'autres fréquences.

2 Validation par simulation

2.1 Simulation d'une mesure

1. D'abord, on commence par générer un signal de bruit blanc pour utiliser en signal émis.

[8]:
```
# Paramètres
T = 10  # secondes
fs = 44000  # Hz
c = 340  # vitesse du son
```

```
# signal source
N = int(T * fs)
noise = np.random.randn(N)
noise = noise/max(abs(noise)) # normalisation
```

2. Pour simuler le signal reçu dans un environnement réverbérant on peut convoluer le signal dans un réponse de salle utilisée dans la partie précédente. Puis l'écouter pour vérifier notre simulation.

[9]:
```
h_nashville = signal.resample(ho_nashville, int(len(ho_nashville)*fs/
 ↪fs_h_nashville))
noise_nashville = signal.convolve(noise, h_nashville, mode='full')
```

[10]:
```
# Restitution audio
print('Playing Noise in Nashville environnement')
sd.play(noise_nashville/max(noise_nashville),fs,blocking=True)
```

```
Playing Noise in Nashville environnement
```

2.2 Estimation de la réponse impulsionnelle de la salle

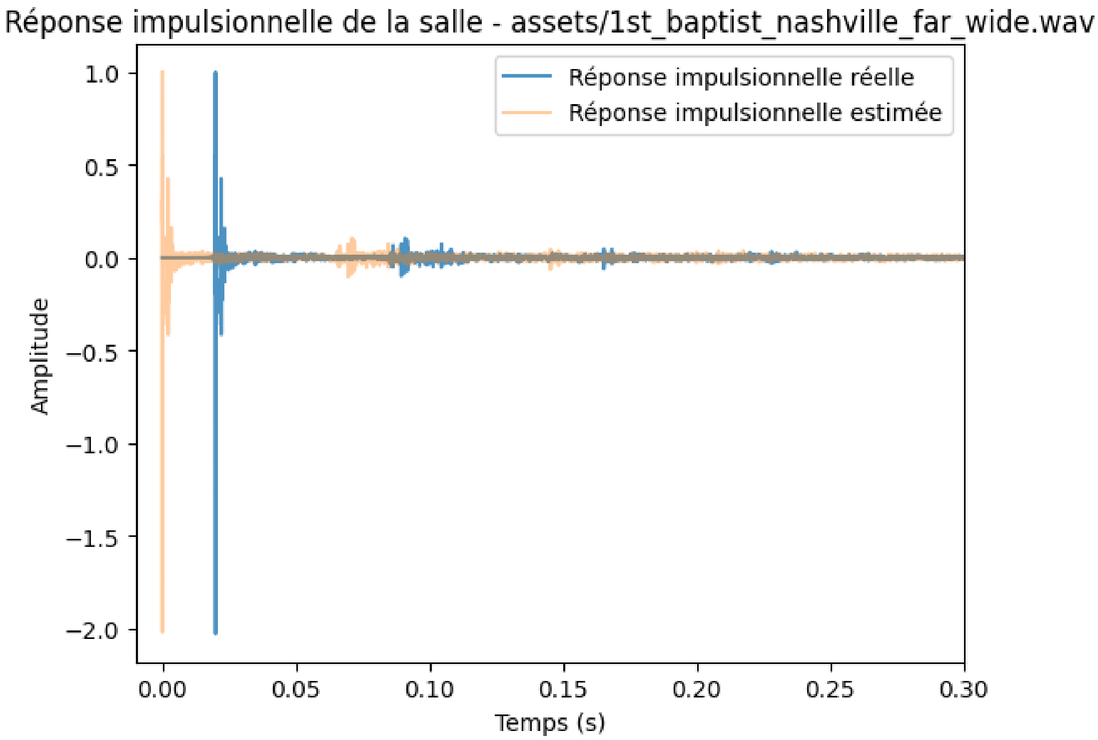
1. Pour estimer la réponse impulsionnelle, on peut émettre un bruit blanc dans la salle que l'on veut caractériser et lire le signal reçu. En calculant l'intercorrelation entre les deux signaux émis et reçu on trouvera la réponse impulsionnelle de la salle.

[11]:
```
estimation_og = signal.correlate(noise_nashville, noise, mode='full')
```

2.

[12]:
```
th = np.arange(0, len(h_nashville))/fs  # Fréquences correspondantes

first_max_value = np.argmax(estimation_og)
estimation = estimation_og[first_max_value:]
estimation = estimation[:len(h_nashville)]

plt.figure()
plt.plot(th, h_nashville/max(h_nashville), alpha=0.8, label="Réponse␣
 ↪impulsionnelle réelle")
plt.plot(th, estimation/max(estimation), alpha=0.4, label="Réponse␣
 ↪impulsionnelle estimée")
plt.title("Réponse impulsionnelle de la salle - " + RI_nashville)
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.legend()
plt.xlim(-0.01, 0.3)
```

[12]:
```
(-0.01, 0.3)
```



On voit que la réponse estimée est similaire à la réponse réelle enregistrée. Il y a un effet d'échelle, (les deux signaux n'ont pas la même amplitude) et de translation (le signal n'est pas situé au même temps). Pour autant, on retrouve les mêmes lobes au alentours de 0.1s et un peu avant 0.2s. C'est ces piques qui constituent les échos entendus. Ainsi, s'ils ont un emplacement et une amplitude similaire par rapport à la réponse impulsionnelle d'origine (en 0) alors le son émis aura la même sonorité. Ce n'est pas forcément important d'avoir la même amplitude absolue car elle est relative à la puissance du signal émis pour la mesure et des conditions d'observation.

3. Pour ajouter un bruit avec un RSB connu on peut calculer la puissance du bruit avec la formule suivante: $P_{noise} = \frac{P_{signal}}{RSB}$. Pour convertir le RSB exprimer en dB, il faut le linériser avec $RSB = 10^{\frac{RSB_{dB}}{10}}$. Une fois la puissance du bruit trouver on peut prendre le carré de cette valeur pour trouver l'écart-type de la distribution que l'on veut pour notre bruit de mesure.

[13]:
```
def add_measure_noise_RSB(signal: np.ndarray, RSB: int) -> np.ndarray:
    signal_power = np.var(signal)
    noise_power = signal_power / 10**(RSB/10)
    noise = np.random.normal(0, np.square(noise_power), len(signal))
    return signal + noise
```

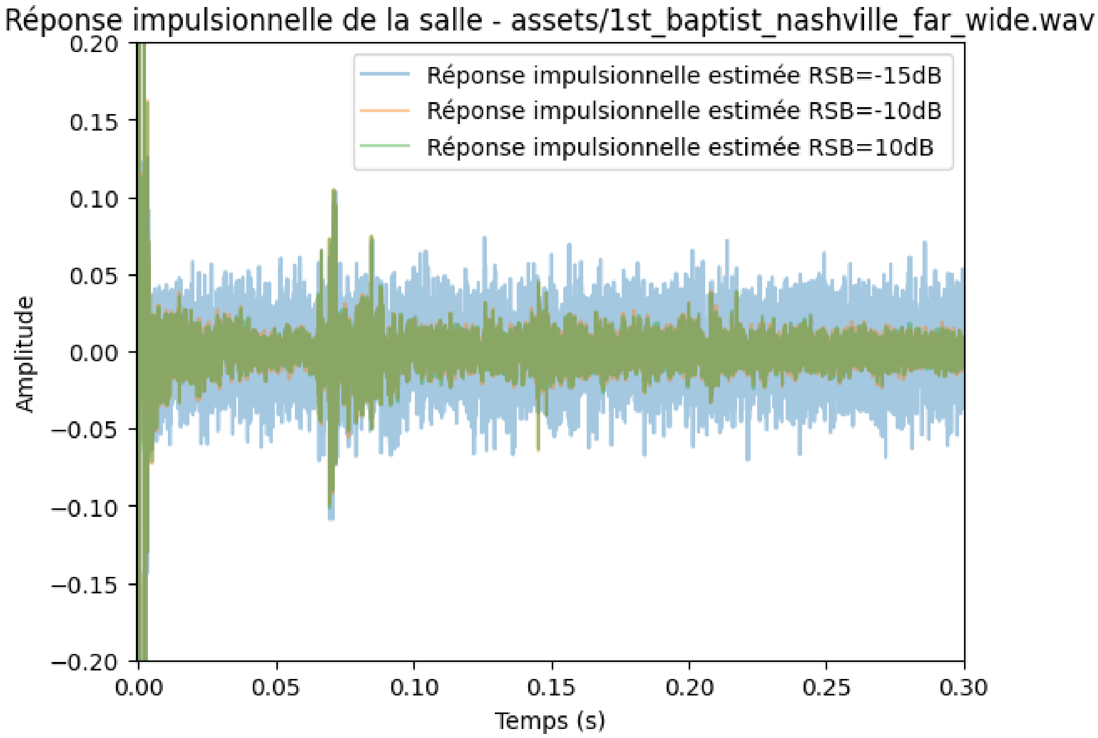
4. Nous allons maintenant étudier l'impact du RSB et de la durée d'émission du signal.

[14]:
```
RSBs = [-15, -10, 10]
noise_nashville_norm = noise_nashville/max(noise_nashville)
```

```
estimations = []
for RSB in RSBs:
    noise_nashville_measure_noise = add_measure_noise_RSB(noise_nashville_norm,␣
 ↪RSB)

    estimation = signal.correlate(noise_nashville_measure_noise, noise,␣
 ↪mode='full')
    first_max_value = np.argmax(estimation)
    estimation = estimation[first_max_value:]
    estimation = estimation[:len(h_nashville)-1]
    estimations.append(estimation)
```

[15]:
```
th = np.arange(0, len(estimations[0]))/fs  # Fréquences correspondantes

plt.figure()
for i, estimation in enumerate(estimations):
    plt.plot(th, estimation/max(estimation), alpha=0.4, label=f"Réponse␣
 ↪impulsionnelle estimée RSB={RSBs[i]}dB")
plt.title("Réponse impulsionnelle de la salle - " + RI_nashville)
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.legend()
plt.xlim(-0.001, 0.3)
plt.ylim(-0.2, 0.2)
```

[15]:
```
(-0.2, 0.2)
```



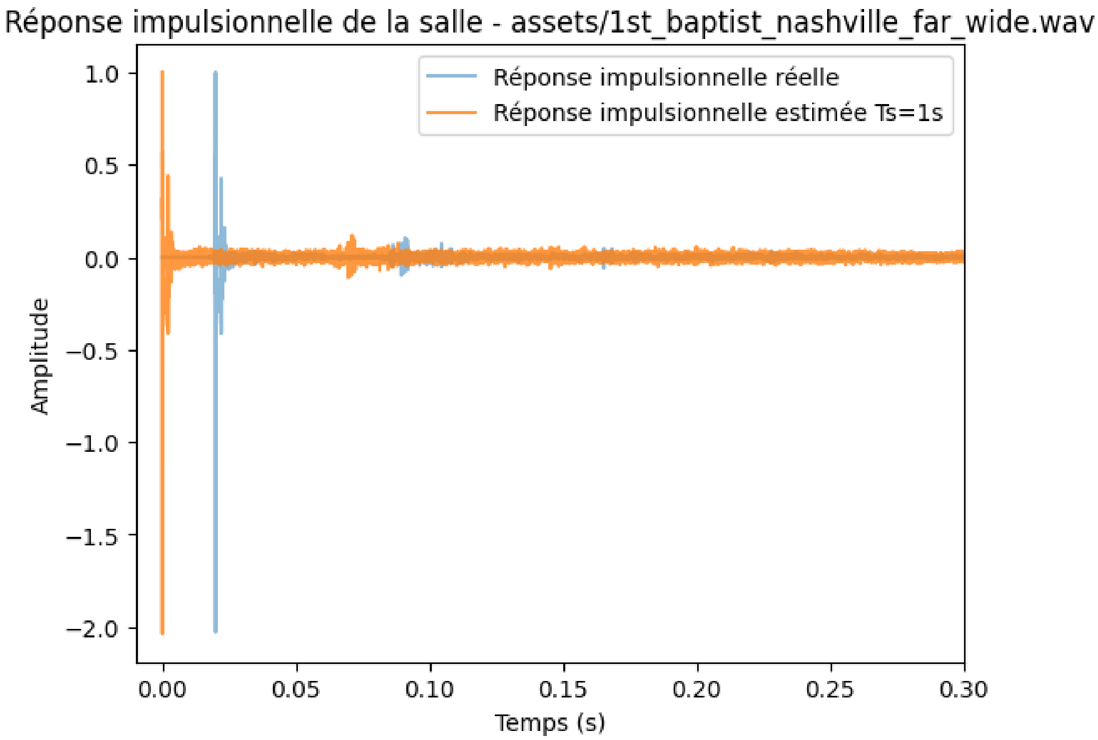
En augmentant la valeur RSB (Rapport Signal sur Bruit) on remarque que la précision de l'estimation de la réponse impulsionnelle de la salle est de plus en plus précise. En réalité la différence est plus visible pour les petits échos, qui, si le bruit est trop important ne seront pas pris en compte. Ainsi, au delà d'une certaine valeur de RSB la différence devient moins significative quant à la qualité de l'estimation.

Ce RSB peut être augmenter de deux manières, soit en réduisant le bruit dans l'environnement lors de la mesure et au niveau des intruments utilisés. Ou bien en augmentant la puissance du signal émis. C'est cette deuxième option que nous allons voir en jouant sur la durée d'enregistrement de notre bruit.

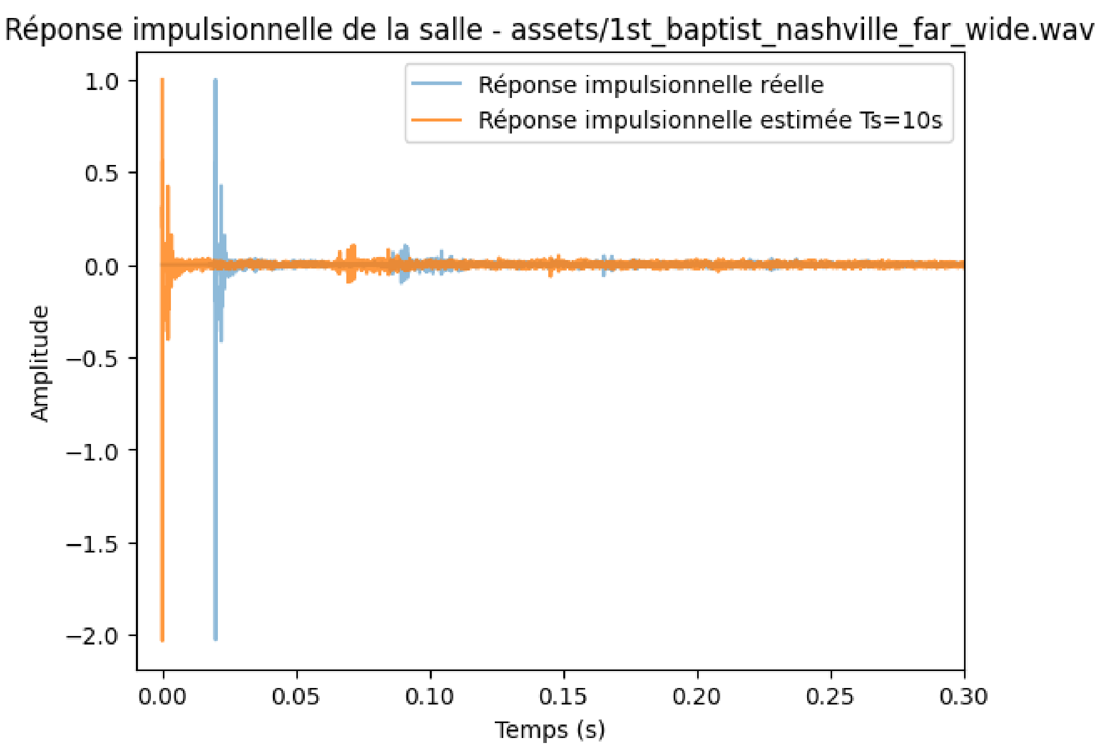
[16]:
```
# Paramètres
Ts = [1, 10, 20]  # secondes
fs = 44000  # Hz
c = 340  # vitesse du son

noises = []
for T in Ts:
    N = int(T * fs)
    noise = np.random.randn(N)
    noise = noise/max(abs(noise)) # normalisation
    noises.append(noise)
```

[17]:
```
estimations = []
RSB = -10

for noise in noises:
    noise_nashville = signal.convolve(noise, h_nashville, mode='full')
    noise_nashville_measure_noise = add_measure_noise_RSB(noise_nashville/
 ↪max(noise_nashville), RSB)

    estimation = signal.correlate(noise_nashville_measure_noise, noise,␣
 ↪mode='full')
    first_max_value = np.argmax(estimation)
    estimation = estimation[first_max_value:]
    estimation = estimation[:len(h_nashville)-1]
    estimations.append(estimation)
```

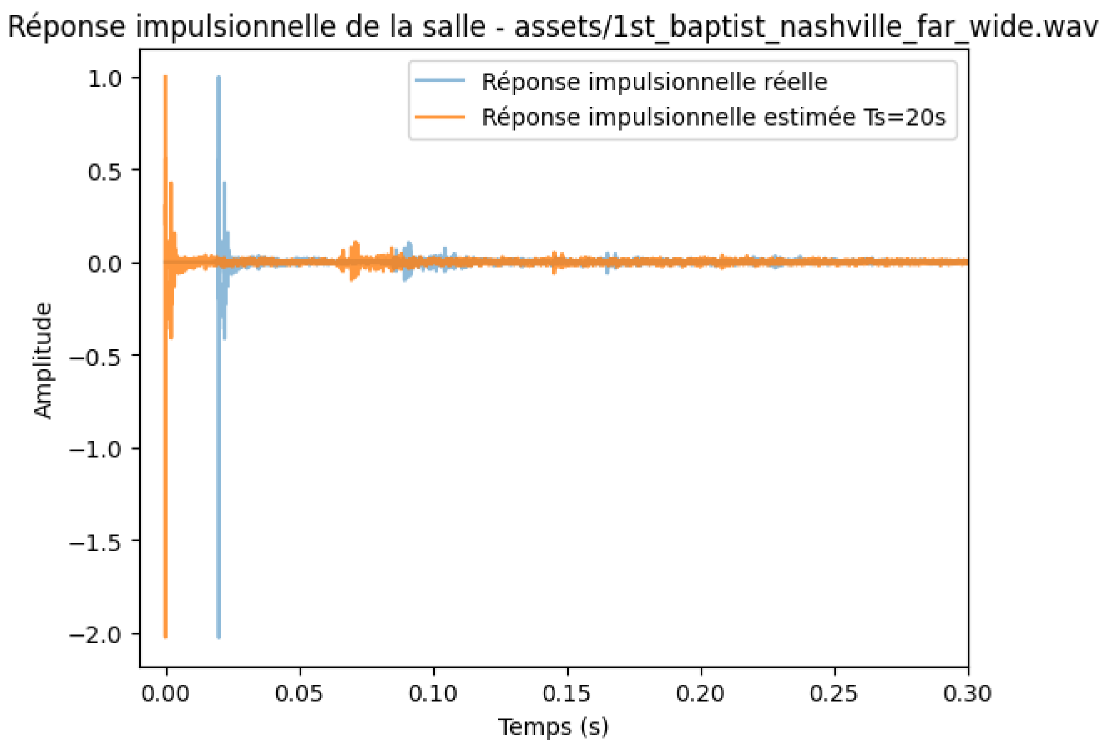
[18]:
```
th = np.arange(0, len(estimations[0]))/fs  # Fréquences correspondantes
th2 = np.arange(0, len(h_nashville))/fs  # Fréquences correspondantes

for i, estimation in enumerate(estimations):
    plt.figure()
    plt.plot(th2, h_nashville/max(h_nashville), alpha=0.5, label="Réponse␣
 ↪impulsionnelle réelle")
    plt.plot(th, estimation/max(estimation), alpha=0.8, label=f"Réponse␣
 ↪impulsionnelle estimée Ts={Ts[i]}s")
    plt.title("Réponse impulsionnelle de la salle - " + RI_nashville)
    plt.xlabel("Temps (s)")
    plt.ylabel("Amplitude")
    plt.legend()
    plt.xlim(-0.01, 0.3)
```



Réponse impulsionnelle de la salle - assets/1st_baptist_nashville_far_wide.wav
Réponse impulsionnelle réelle
Réponse impulsionnelle estimée Ts=1s
Amplitude
Temps (s)
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.00
0.05
0.10
0.15
0.20
0.25
0.30



Réponse impulsionnelle de la salle - assets/1st_baptist_nashville_far_wide.wav
Réponse impulsionnelle réelle
Réponse impulsionnelle estimée Ts=10s
Amplitude
Temps (s)
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.00
0.05
0.10
0.15
0.20
0.25
0.30



Commme attendu lors de l'étude de l'effet du rapport Signal-Bruit, l'augmentation du temps d'émission du bruit blanc pour l'estimation ajoute virtuellement de l'énergie à notre signal. Ce qui permet d'avoir un signal équivalent plus fort le rendant moins sensible au bruit. Cependant, l'augmentation de la durée d'enregistrement augmente également la difficuleté de traitement informatique car les données sont plus longues et donc plus volumineuses.

3 Validation expérimentale

3.1 Mesure de la réponse impulsionnelle de la salle

1. Grâce au script fourni, on peut faire une mesure réel d'un bruit et de son écoute dans un salle.

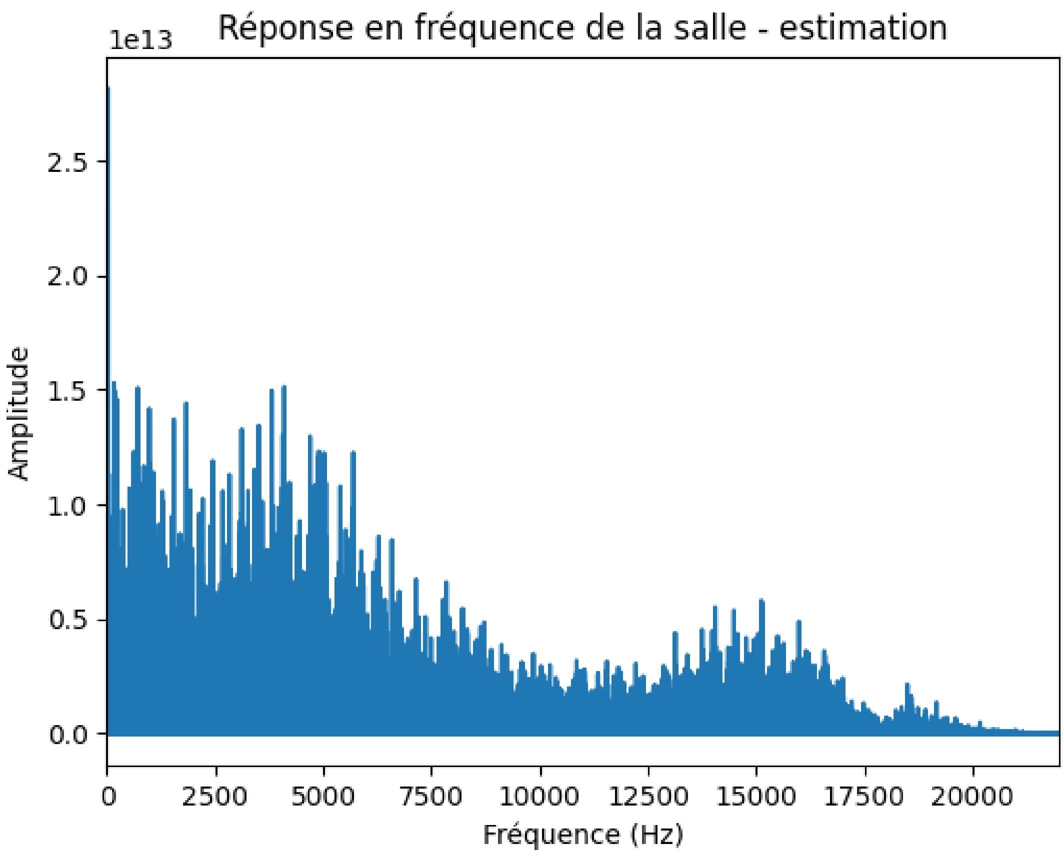
[19]:
```
# signal émis
name_source = 'assets/mesures/240927 11-54-source.wav'

# signal reçu
name_mesure = 'assets/mesures/240927 11-54-mesure.wav'

# Lecture des données
fs, source = sio.wavfile.read(name_source)
fs_mesure, mesure = sio.wavfile.read(name_mesure)
```

```python
# Conversion en float
source = np.float64(source)
mesure = np.float64(mesure)[:,0]

# Réecchantillonage
mesure = signal.resample(mesure, int(len(mesure)*fs/fs_mesure))
```

2. Pour estimer la réponse impulsionnelle de la salle, on peut utiliser notre fonction d'estimation vue précédemment.

[20]:
```python
estimation_og = signal.correlate(mesure, source, mode='full')
```

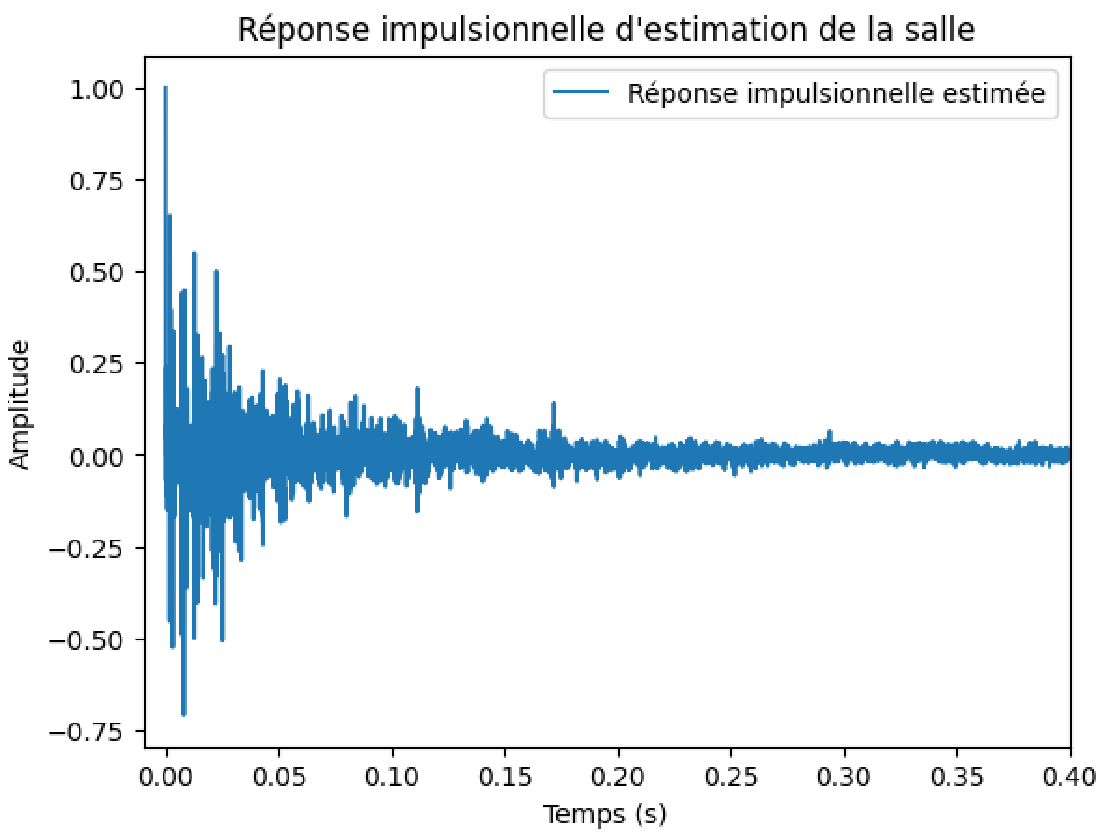
[21]:
```python
show_RI_fft(estimation_og, "estimation")
```



[22]:
```python
# Alignement de la réponse à zéro
first_max_value = np.argmax(estimation_og)
estimation = estimation_og[first_max_value:]
estimation = estimation[:len(h_nashville)-1]

# Temps correspondants
```

```
th = np.arange(0, len(estimation))/fs_mesure

plt.figure()
plt.plot(th, estimation/max(estimation), label=f"Réponse impulsionnelle␣
 ↪estimée")
plt.title("Réponse impulsionnelle d'estimation de la salle")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.legend()
plt.xlim(-0.01, 0.4)
```

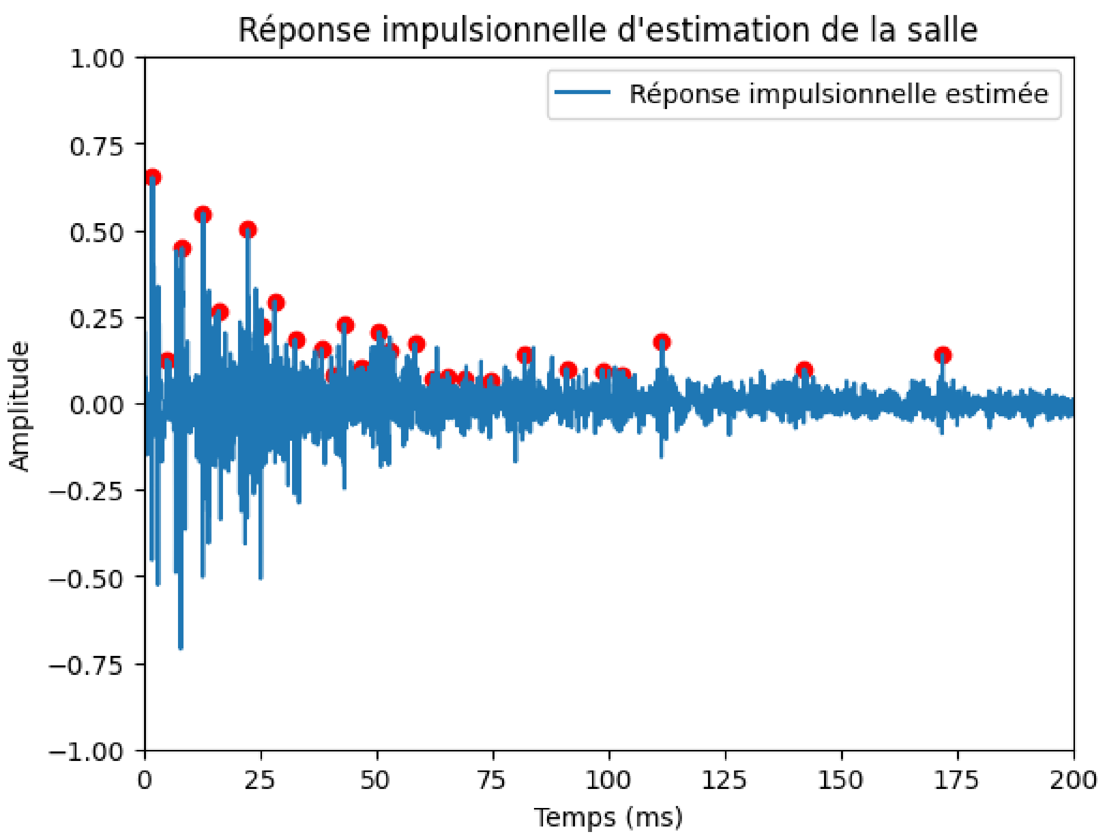
[22]: (-0.01, 0.4)



Quand on regarde la réponse en fréquence de la réponse impulsionnelle estimée, on remarque qu'elle se rapproche de la réponse impulsionnel de notre gymnase utilisé dans les premières parties du TP. Puis, sur la réponse impulsionelle, on peut remarquer que la réponse est bruité, en réalité il s'agit en partie des échos.

3.

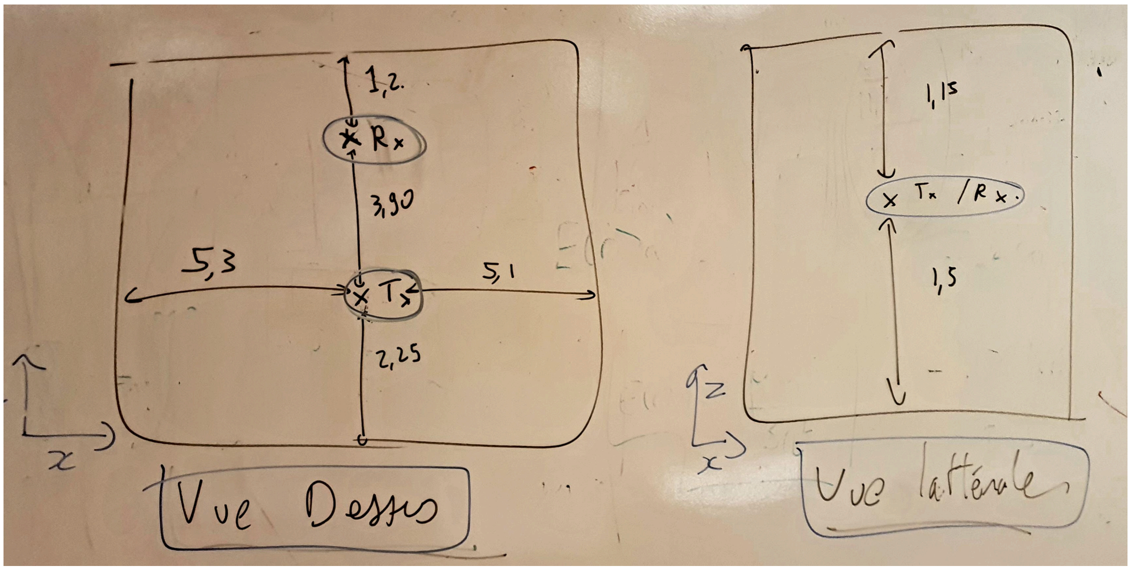
[23]:
```python
estimation_norm = estimation/max(estimation)
peaks, _ = signal.find_peaks(estimation_norm, threshold=0.02, distance=100)

plt.figure()
plt.plot(th*1000, estimation_norm, label=f"Réponse impulsionnelle estimée")
plt.scatter(th[peaks]*1000, estimation_norm[peaks], color="red")
plt.title("Réponse impulsionnelle d'estimation de la salle")
plt.xlabel("Temps (ms)")
plt.ylabel("Amplitude")
plt.legend()
plt.xlim(0, 200)
plt.ylim(-1, 1)
plt.show()
th[peaks]*1000
```



[23]:
```
array([ 1.93181818,   5.06818182,   8.31818182,  12.84090909,
       16.15909091,  19.90909091,  22.43181818,  25.40909091,
       28.25      ,  32.63636364,  36.        ,  38.36363636,
       40.68181818,  43.18181818,  47.11363636,  50.68181818,
       53.13636364,  55.59090909,  58.40909091,  62.15909091,
```

```
 65.59090909,  69.20454545,  72.22727273,  74.72727273,
 82.        ,  86.77272727,  91.06818182,  98.88636364,
103.02272727, 111.61363636, 142.22727273, 171.95454545])
```

Pour ces premiers échos on peut estimer à quel distance ils correspondent grâce aux mesures de la salle. Cela est possible car le premier pique observé sur la réponse impulsionnelle correspond à l'arrivé du signal en ligne directe, les autres piques correspondent à des signaux arrivés en décalage.



On calculera à l'aide de trigonométrie de base les différentes distances parcourues pour les échos rebondissant une fois: - Echo sur un des murs gauche ou droit : $d \approx 10.92m$ - Echo sur le plafond : $d \approx 4.53m$ - Echo sur le sol : $d \approx 4.92m$ - Echo sur le mur à l'arrière du micro : $d \approx 8.4m$ - Echo sur le mur à l'arrière du récepteur : $d \approx 6.3m$

D'où un décalage temporelle de $n = \frac{d_{micro_recepteur} - d_{parcourue}}{c}$: - Echo sur un des murs gauche ou droit : $\Delta t \approx 20.65ms$ - Echo sur le plafond : $\Delta t \approx 1.85ms$ - Echo sur le sol : $\Delta t \approx 3ms$ - Echo sur le mur à l'arrière du micro : $\Delta t \approx 13.24ms$ - Echo sur le mur à l'arrière du récepteur : $\Delta t \approx 7.06ms$

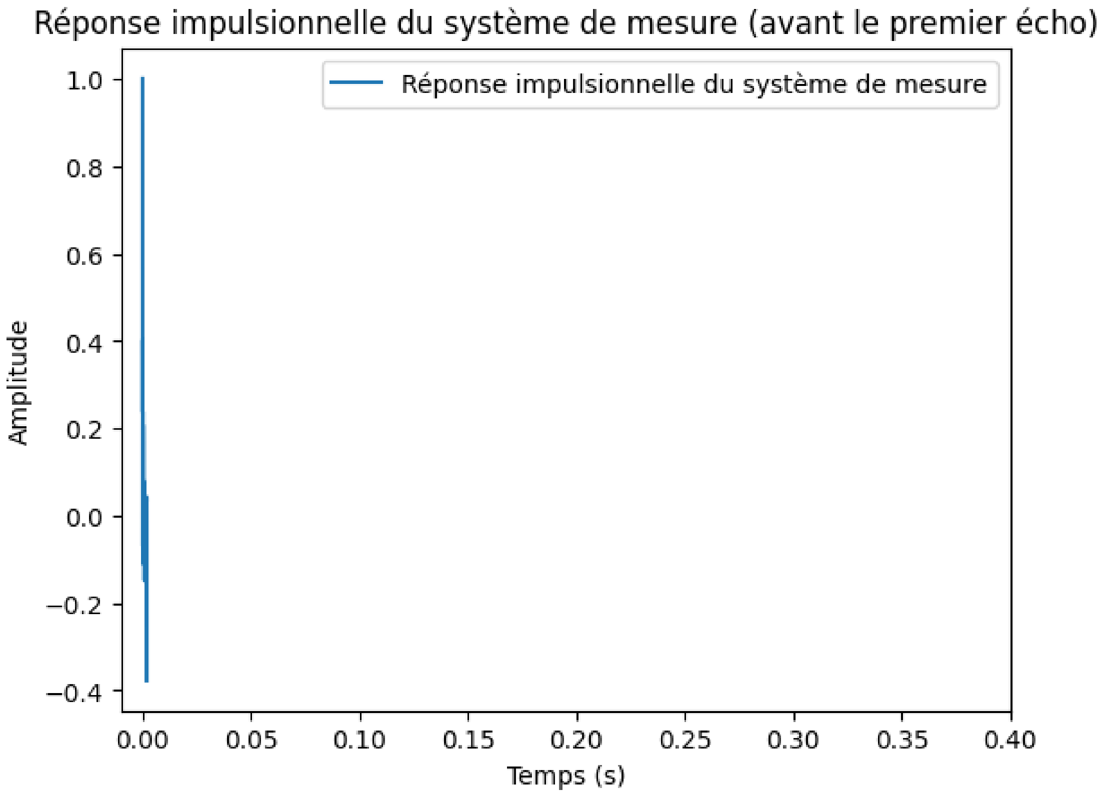
On remarque que le décalage temporelle est très petit et donc sensible au variation de mesure et d'estimation. Cependant, on retrouve certains écho dans notre tableau de piques. D'abord, l'écho sur le plafond qui est le premier à arriver après le signal en ligne directe. Puis, l'écho sur le sol est relativement différent de la mesure attendue, on peut aussi retrouver l'écho sur le mur à l'arrière du micro autour de 13ms et enfin sur le mur à l'arrière du récepteur vers 7/8ms. On trouvera ensuite toute les compositions et rebond multiples. On pourrait également prendre le problème inverse et à partir du Δt mesuré sur la réponse impulsionnelle de la salle puis trouver la distance totale parcourue par l'écho avec $d = \Delta t * c + 3.9$

3.2 Réponse du système audio

1. On cherche à isoler la réponse impulsionnelle du système de mesure en conservant la réponse impulsionnelle entre la réponse et le premier echo

[24]:

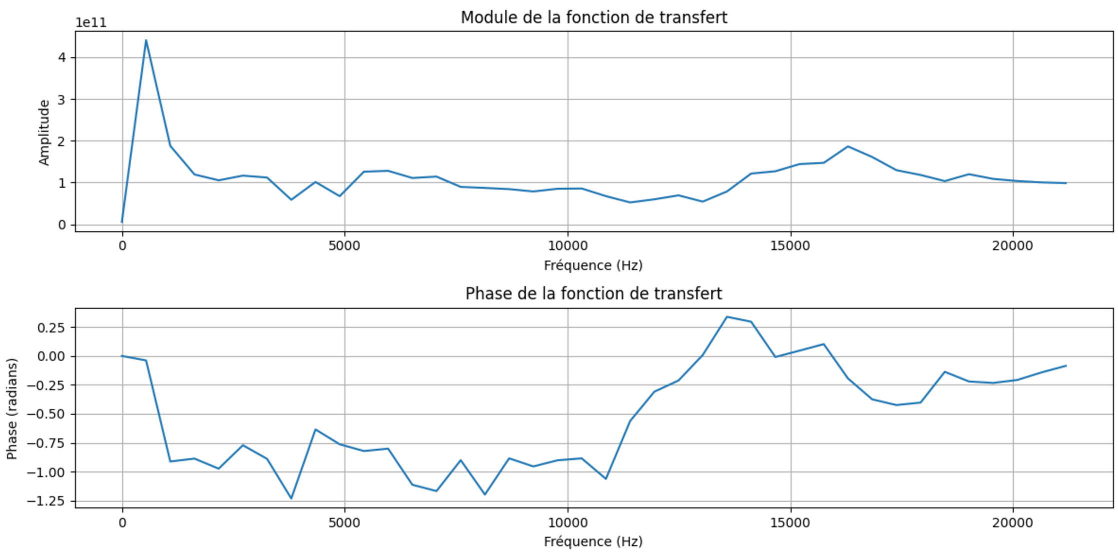
```python
# Temps du premier écho (en secondes)
t_echo1 = 1.85e-3  # 1.85 ms
n_echo1 = int(t_echo1 * fs_mesure)  # Nombre d'échantillons correspondant à␣
↪t_echo1

# Alignement de la réponse à zéro
first_max_value = np.argmax(estimation_og)
estimation = estimation_og[first_max_value:]

# Séparer la réponse impulsionnelle du système (avant t_echo1)
system_response = estimation[:n_echo1]

# Temps correspondants pour la réponse du système
th = np.arange(0, len(system_response)) / fs_mesure

# Affichage de la réponse impulsionnelle du système de mesure
plt.figure()
plt.plot(th, system_response / max(system_response), label=f"Réponse␣
↪impulsionnelle du système de mesure")
plt.title("Réponse impulsionnelle du système de mesure (avant le premier écho)")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.legend()
plt.xlim(-0.01, 0.4)
plt.show()
```



2. Puis on peut maitenant représenter sa fonction de transfert en calculant la transforméé de Fourier de la réponse impulsionnelle.

[25]:
```
# Calcul de la fonction de transfert (transformée de Fourier de la réponse␣
↪impulsionnelle)
H_system = fft(system_response)  # Transformée de Fourier
frequencies = fftfreq(len(system_response), 1/fs_mesure)  # Fréquences associées

# Magnitude (amplitude) et phase
magnitude = np.abs(H_system)
phase = np.angle(H_system)

# Affichage de la magnitude de la fonction de transfert
plt.figure(figsize=(12, 6))

# Affichage du module de la fonction de transfert (réponse en amplitude)
plt.subplot(2, 1, 1)
plt.plot(frequencies[:len(frequencies)//2], magnitude[:len(frequencies)//2])
plt.title("Module de la fonction de transfert")
plt.xlabel("Fréquence (Hz)")
plt.ylabel("Amplitude")
plt.grid(True)
```

```python
# Affichage de la phase de la fonction de transfert
plt.subplot(2, 1, 2)
plt.plot(frequencies[:len(frequencies)//2], phase[:len(frequencies)//2])
plt.title("Phase de la fonction de transfert")
plt.xlabel("Fréquence (Hz)")
plt.ylabel("Phase (radians)")
plt.grid(True)

plt.tight_layout()
plt.show()
```

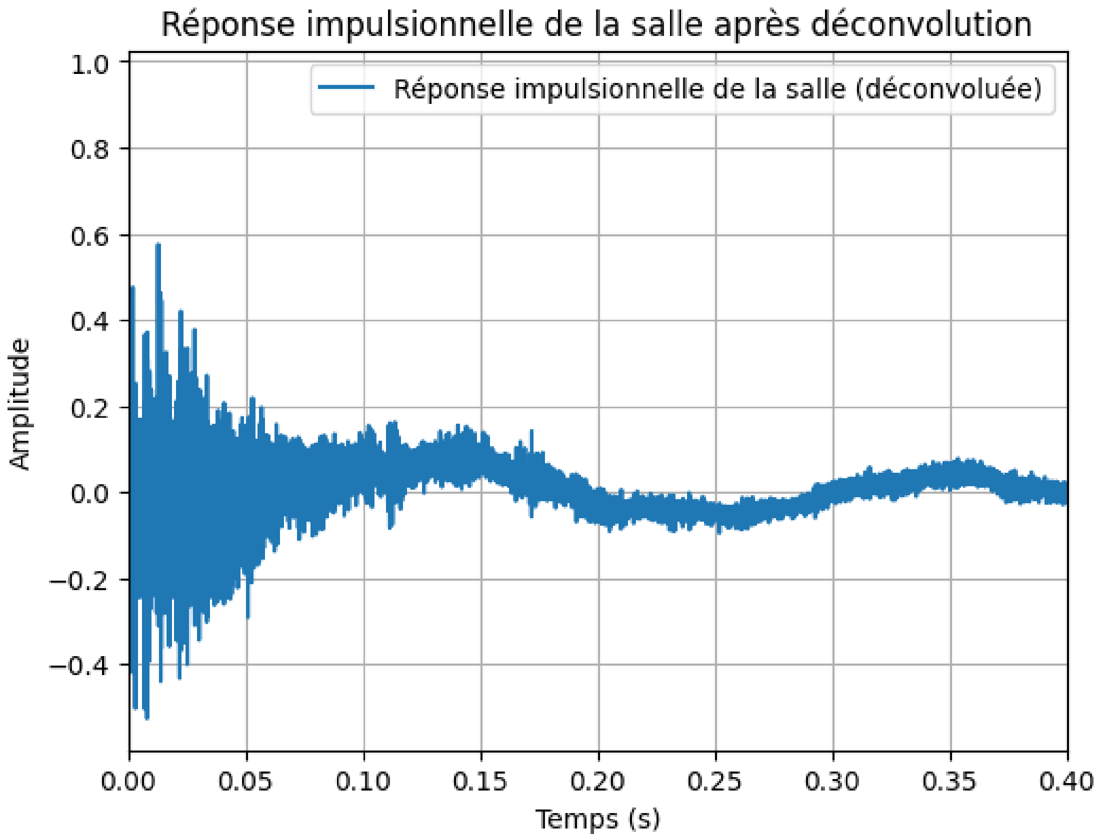


On observe que l'amplitude est amplifiée dans les basses fréquences et dans les fréquences supérieures à 15 kHz. Il y a une légère atténuation entre 10 et 15kHz. Les signaux sont presque déphasés de - /3 jusqu'à 10 kHz.

3. Pour déconvoluer le signal on passe à la transformer de Fourier : Y(f) = H_room(f)xH_system(f), on effectue une division dans le plan fréquentiel puis on calcule la TF inverse pour revenir à y(t).

[26]:
```python
from scipy.fft import ifft

# Transformée de Fourier du signal mesuré et de la réponse du système audio
Y = fft(estimation)  # Transformée de Fourier du signal mesuré

# Zero-padding de 'system_response' pour qu'elle ait la même longueur que 'estimation'
system_response_padded = np.pad(system_response, (0, len(estimation) - len(system_response)), 'constant')
```

```python
# Transformée de Fourier de la réponse du système audio
H_sys = fft(system_response_padded)  # Transformée de Fourier de la réponse du␣
 ↪système audio

# Éviter la division par zéro ou les très faibles valeurs dans H_sys
eps = 1e-10
H_sys = np.where(np.abs(H_sys) < eps, eps, H_sys)

# Déconvolution fréquentielle : Calcul de la réponse impulsionnelle de la salle
H_room = Y / H_sys  # Division dans le domaine fréquentiel

# Transformée inverse pour revenir dans le domaine temporel (déconvoluer)
h_room = np.real(ifft(H_room))  # Revenir à la réponse impulsionnelle dans le␣
 ↪domaine temporel

# Calcul des temps correspondants pour l'axe des x
th_room = np.arange(0, len(h_room)) / fs_mesure

# Affichage de la réponse impulsionnelle de la salle après déconvolution
plt.figure()
plt.plot(th_room, h_room, label="Réponse impulsionnelle de la salle␣
 ↪(déconvoluée)")
plt.title("Réponse impulsionnelle de la salle après déconvolution")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.legend()
plt.grid(True)
plt.xlim(0, 0.4)
plt.show()
```



Réponse impulsionnelle de la salle après déconvolution
Réponse impulsionnelle de la salle (déconvoluée)
Amplitude
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Temps (s)