Compte rendu de traveaux pratiques : Projet MATLAB

Tudor OPRAN

7 décembre 2023

Table des matières

1 Introduction

Dans ce rapport, nous abordons le domaine complexe et en constante évolution du traitement audio et des effets numériques. Notre objectif est de démystifier les principes fondamentaux du traitement du signal audio, de l'analyse fréquentielle aux techniques avancées d'effets audio tels que la réverbération et le délai. À travers une combinaison de théorie et d'application pratique, ce rapport vise à fournir une compréhension approfondie de la manière dont les effets numériques peuvent être appliqués et manipulés pour améliorer la qualité sonore. L'intégralité du projet se trouve à l'adresse : Code GitHub.

2 Echantillonnage

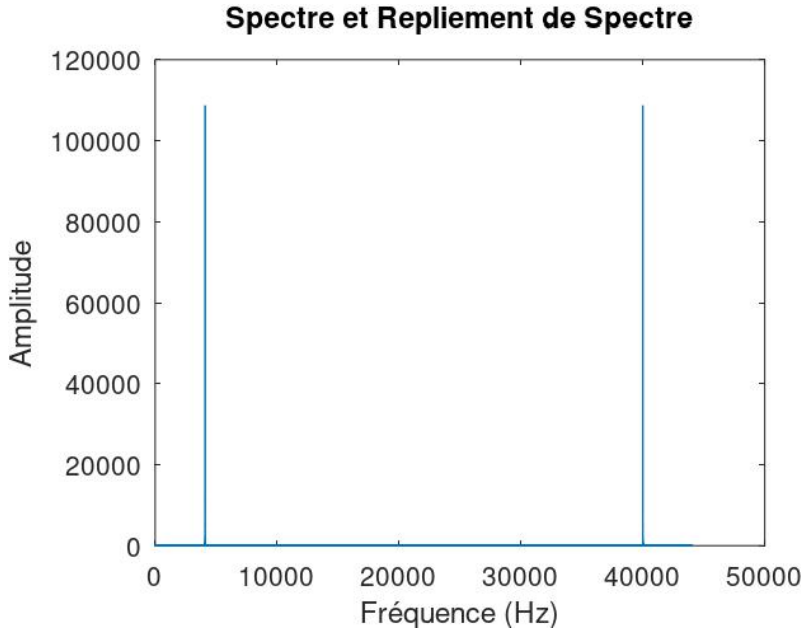
Manipulation 7.2.1: Ce programme nous fait entendre un son de fréquence 44100 Hz sous-échantillonné à la fréquence 40000 Hz. D'après le critère de Nyquist-Shannon, nous avons un repliement de spectre donc la fréquence du son émis devient :

$$f_{\text{émission}} = \frac{f_{\text{échantillonnage}}}{2} - \left| f_{\text{signal}} - \frac{f_{\text{échantillonnage}}}{2} \right|$$

or $f_{\text{signal}} \geqslant \frac{1}{2} f_{\text{échantillonnage}}$ donc $f_{\text{émission}} = |f_{\text{signal}} - f_{\text{échantillonnage}}|$. On obtient alors une fréquence d'émission audible égale à 4100 Hz.

Programme 7.2.1: Le programme implémentant la manipulation est le suivant :

```
fc = 40000;
fe = 44100;
t = [0 : 1 / fe : 5];
x = sin(2 * pi * t * fc);
soundsc(x, fe);
```



Question 7.2.1: Le son généré par le programme peut être entendu par l'humain ca sa fréquence se situe dans le domaine des fréquences audibles 4100Hz $\in [0, 20000]$ Hz.

Manipulation 7.2.2: Pour charger le son contenu dans le fichier, il faut exécuter le programme ci-joint.

Programme 7.2.2: Ce programme charge le son dans Matlab et affiche la fréquence d'encodage :

```
[signal, fe] = audioread('mehldau.wav');
fe
```

Question 7.2.2: En executant le programme, nous retrouvons une fréquence d'encodage de 44100 Hz.

Manipulation 7.2.3: En executant les trois sous-échantillonages de facteurs 2, 4 et 8, nous observons que dans chaque cas, puisque la fréquence du signal (44100 Hz) est largement supérieure à la moitié de la fréquence d'échantillonnage, le repliement se produit autour de la fréquence de Nyquist ($\frac{fe}{2}$).

Programme 7.2.3: Ce programme sous-échantillonne le signal original d'un facteur 2, 4 et 8 :

```
[signal, fe] = audioread('mehldau.wav');

% Sous-echantillonnage par un facteur de 2
signal_sous_echantillonne2 = signal(1:2:end);
audiowrite('signal_sous_echantillonne2.wav',

% Sous-echantillonnage par un facteur de 4
signal_sous_echantillonne4 = signal(1:4:end);
audiowrite('signal_sous_echantillonne4.wav',

% Sous-echantillonnage par un facteur de 8
signal_sous_echantillonne8 = signal(1:8:end);
audiowrite('signal_sous_echantillonne8.wav',
```

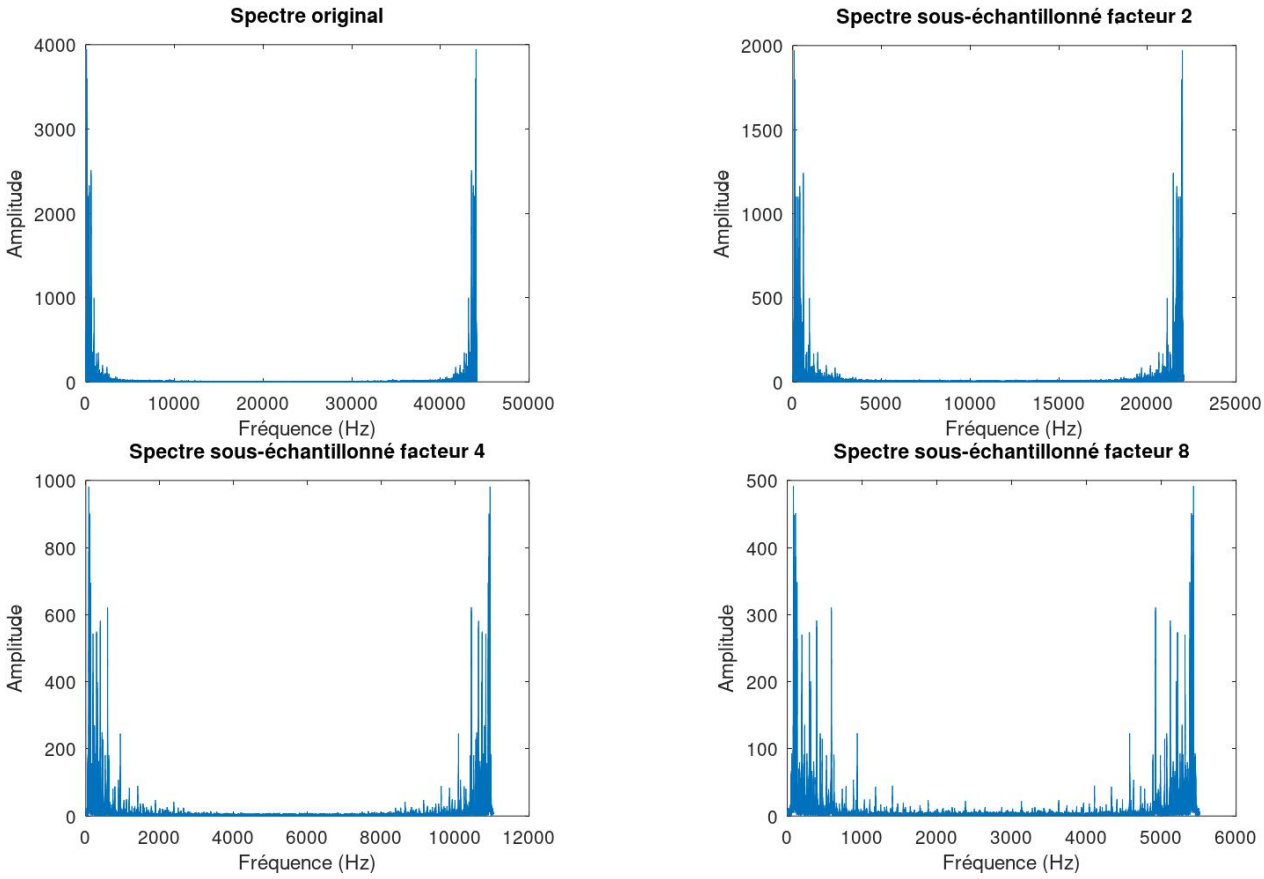


FIGURE 1 – Comparaison des spectres pour différents facteurs de sous-échantillonnage

Question 7.2.3: Après execution du programme et mesure de la taille des fichiers sous-échantillonés, nous obtenons les données suivantes :

Type de fichier	Taille
Fichier original	832 Ko (852 646 octets)
Sous-échantillonnage facteur 2	416 Ko (426 346 octets)
Sous-échantillonnage facteur 4	208 Ko (213 196 octets)
Sous-échantillonnage facteur 8	104 Ko (106 620 octets)

TABLE 1 – Tailles des fichiers après sous-échantillonnage

Question 7.2.4: Plus le facteur de sous-échantillonnage est important, plus la qualité sonore diminue. Ceci est dû à l'effacement de certaines nuances sonores par repliement de spectre. De plus, on observe que les fichiers audio sous-échantillonnés ressemblent au fichier original entendu de loin ou à travers des murs. Ceci peut aussi témoigner d'un effet de distorsion induit par le sous-échantillonnage.

3 Analyse fréquentielle

Programme 7.3.1: Ce programme charge le fichier audio, affiche sa représentation temporelle, son spectre d'amplitude en décibels, et permet de l'écouter pour analyser son contenu sonore.

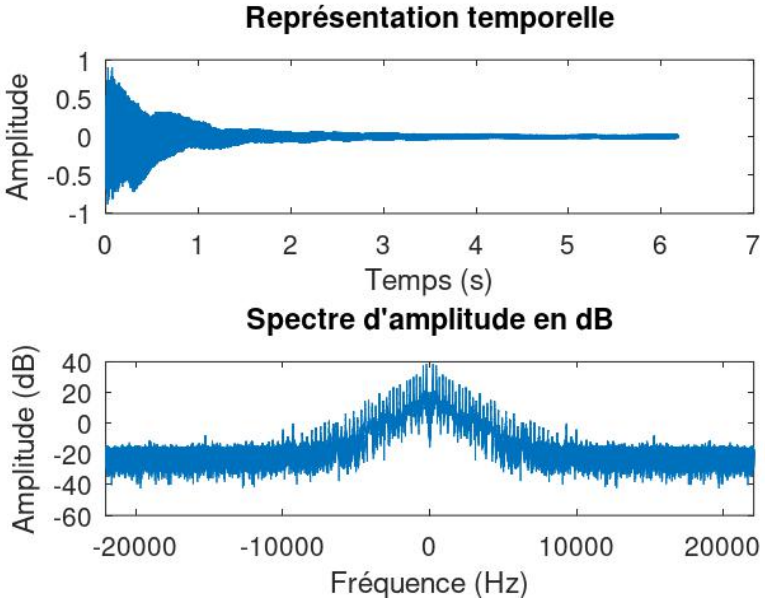
```
% Charger le son piano1.wav
[y, fs] = audioread('input/piano1.wav');

% Visualiser la representation temporelle
subplot(2, 1, 1);
t = (0:length(y) - 1) / fs;
plot(t, y);
xlabel('Temps (s)');
ylabel('Amplitude');
title('Representation temporelle');

% Visualiser le spectre d'amplitude en decibels
subplot(2, 1, 2);
N = length(y);
f = (-N/2:N/2-1) * fs / N;
Y = fftshift(fft(y));
amplitude_dB = 10 * log10(abs(Y));
plot(f, amplitude_dB);
xlabel('Frequence (Hz)');
ylabel('Amplitude (dB)');
title('Spectre d'amplitude en dB');
xlim([-fs/2, fs/2]);

% Ecouter la note du piano
sound(y, fs);
```

Question 7.3.1: La représentation temporelle du signal révèle l'existence de possibles harmoniques à travers les oscillations en amplitude observés entre 0 et 1.5 secondes. De plus la representation temporelle permet une mesure de la durée de la note quand cette métrique est bien définie.



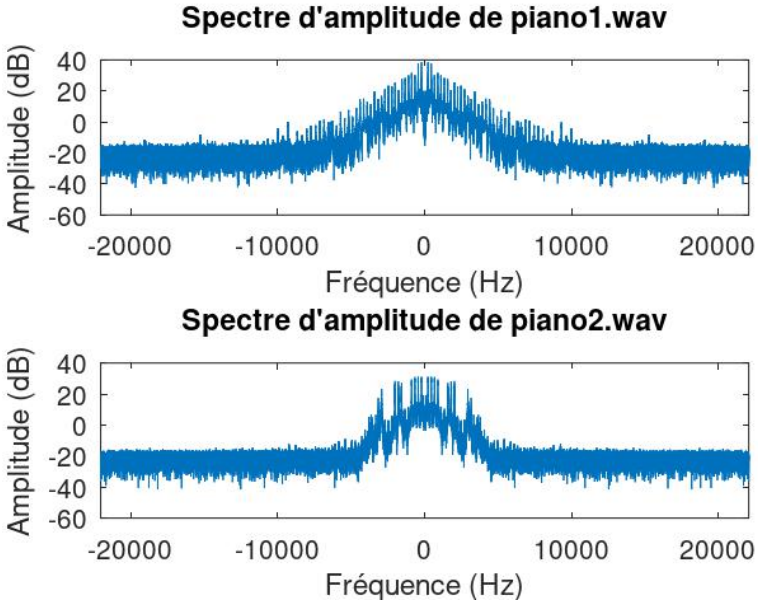
Question 7.3.2: La fréquence fondamentale f_1 du signal se défini par la fréquence du premier pic en amplitude après celui de la valeur éfficace. Nous mesurons $f_1 = 220.374$ Hz.

Question 7.3.3: On en déduit le tableau suivant :

Fréquence (Hz)	f_1	f_2	f_3	f_4	f_5	f_6	f_7
Théoriques	220	440	660	880	1100	1320	1540
Mesurées	220.374	442.041	663.224	885.538	1108.34	1331.46	1556.2
Inharmonicité	2.9406	8.0120	8.4362	10.8608	13.0764	14.9654	18.1166

TABLE 2 – Degré d'inharmonicité par fréquence

Question 7.3.4: Plus les harmoniques sont bien définis et espacés, plus le son est harmonieux. Donc le second piano est plus harmonieux que le premier même si le son est plus simple et plus raide.



4 Effets audio-numériques

4.1 Prédétermination : mesure de réponse impulsionnelle

Question 7.4.1: En partant des définitions suivantes :

$$R_{yx} = \sum_{k\in\mathbb{Z}} y(k)X^*(k-u) \qquad y(k) = \sum_{n\in\mathbb{Z}} h(n)X(k-n)$$

on obtient :

$$\begin{aligned}
R_{yx}(u) &= \sum_{k\in\mathbb{Z}}\sum_{n\in\mathbb{Z}} h(n)X(k-n)X^*(k-u) \\
&= \sum_{n\in\mathbb{Z}}\sum_{k\in\mathbb{Z}} h(n)X(k-n)X^*(k-u) \\
&= \sum_{n\in\mathbb{Z}} h(n)\sum_{k\in\mathbb{Z}} X(k-n)X^*(k-u) \\
&= \sum_{n\in\mathbb{Z}} h(n)\sum_{j\in\mathbb{Z}} X(j)X^*(j-(n+u)) && \text{changement de variable} j = k-n \\
&= \sum_{n\in\mathbb{Z}} h(n)R_{xx}(n+u) && \text{par symétrie de } R \\
&= (h \star R_{xx})(u)
\end{aligned}$$

Question 7.4.2: Supposons que $R_{xx}(u) \approx d(u)$, où $d(u)$ est la fonction impulsion unité. Dans ce cas, nous avons :

$$R_{yx}(u) = d(u) \star h(u)$$

Pour estimer la réponse impulsionnelle $h(k)$, nous pouvons effectuer la transformation de Fourier inverse (FT^{-1}) de $R_{yx}(u)$ pour revenir au domaine temporel. Cela revient à :

$$h(k) = \mathrm{FT}^{-1}[R_{yx}(u)]$$

En effectuant cette opération, nous obtenons une estimation de la réponse impulsionnelle $h(k)$ de la pièce, qui inclut également la réponse de la chaîne de génération-captation.

4.2 Prédétermination : étude théorique de l'effet de retard

Question 7.4.3: Pour le filtre décrit par l'équation (7.2), nous avons :

$$y(k) = x(k) - g \cdot y(k - \tau)$$

Supposons que nous ayons une impulsion unité comme entrée, c'est-à-dire $x(k) = \delta(k)$, où $\delta(k)$ est la fonction impulsion unité. Dans ce cas, lorsque $k = 0$, $x(k) = 1$, et lorsque $k \neq 0$, $x(k) = 0$. Appliquons cette impulsion unité à l'entrée du filtre :

$$y(k) = \delta(k) - g \cdot y(k - \tau)$$

Pour $k = n\tau$, nous avons :

Si $n = 0$:

$$\begin{aligned} y(n\tau) &= \delta(0) - g \cdot y(0 - \tau) \\ &= 1 - g \cdot y(-\tau) \\ &= 1 \end{aligned}$$

il n'y a pas d'entrée négative donc $y(-\tau) = 0$.

Si $n \geqslant 1$:

$$\begin{aligned} y(n\tau) &= \delta(n\tau) - g \cdot y((n-1)\tau) \\ &= 0 - g \cdot y((n-1)\tau) \\ &= -g \cdot y((n-1)\tau) \end{aligned}$$

par téléscopage, on obtient $y(n\tau) = (-g)^n$

On conclut alors que :

$$h(k) = \begin{cases} (-g)^n \text{ si } k = n\tau \\ 0 \text{ sinon} \end{cases}$$

Question 7.4.4: Pour que le filtre soit stable, il faut que sa réponse impulsionnelle soit absolument sommable. En d'autres termes, si $h(k)$ est la réponse impulsionnelle du filtre, alors la condition de stabilité est la suivante :

$$\sum_{n=-\infty}^{+\infty} |h(n)| < \infty$$

Pour déterminer la condition de stabilité, nous devons calculer la somme des valeurs absolues de $h(k)$ sur toutes les valeurs possibles de k. Dans ce cas, k peut prendre toutes les valeurs entières.

$$\sum_{n=-\infty}^{\infty} |h(n)| = \sum_{k=-\infty}^{\infty} |(-g)^{k\tau}|$$

La série ci-dessus converge si $|g| < 1$ d'après le critère spécial des séries altérnés. Si $|g| \geqslant 1$, le filtre sera instable car sa réponse impulsionnelle ne sera pas absolument sommable. Donc le filtre de la figure 7.1 est stable si et seulement si $|g| < 1$

Question 7.4.5: Les vecteurs a et b pour le filtre h sont définis comme suit :

$$a = [1, -g, 0, 0, \ldots, 0]$$

$$b = [1, 0, 0, 0, \ldots, 0]$$

où g est la valeur du coefficient de rétroaction du filtre.
Le vecteur a représente les coefficients de la partie récursive du filtre, tandis que le vecteur b représente les coefficients de la partie non-récursive. Ici, la partie récursive de a commence avec $a_1 = -g$ pour refléter la rétroaction négative du filtre, et les autres coefficients sont mis à zéro car il n'y a pas d'autres termes récursifs dans l'équation.

Manipulation 7.4.1: Le graphique obtenu représente la réponse impulsionnelle du filtre hh en fonction du temps discret. Il montre comment le filtre réagit à une impulsion unité d'entrée et comment la sortie évolue au fil du temps.

Programme 7.4.1: Ce programme génère la réponse impulsionnelle du filtre :

```
g = 0.9; Fe = 44100;
tau = 0.25*Fe; N = 10*Fe;
X = zeros(1,N); X(1) = 1;
a = zeros(1, tau+1); a(1) = 1;
a(tau+1) = g; b = 1;
Y = filter(b, a, X);
stem(Y)
```

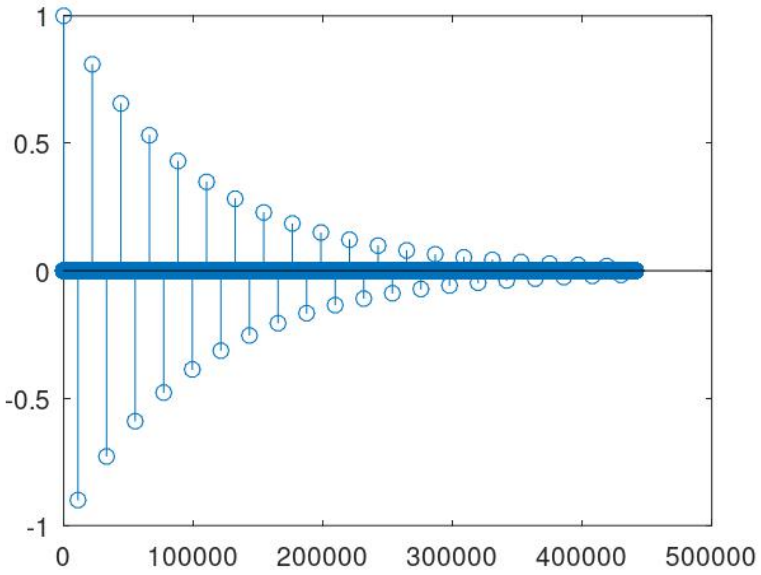


FIGURE 2 – Réponse impulsionelle du filtre en fonction du temps

Question 7.4.6: La réponse impulsionelle en fréquence du filtre est par définition :

$$\frac{\hat{Y}}{\hat{X}} = \frac{\hat{X} - g \cdot TF[Y(k-\tau)]}{\hat{X}} = \frac{\hat{X} - g\hat{Y}e^{-2j\pi\nu\tau}}{\hat{X}} = 1 - ge^{-2j\pi\nu\tau}\frac{\hat{Y}}{\hat{X}}$$

on obtient alors :

$$\hat{h}(\nu) = \frac{1}{1 + ge^{-2j\pi\nu\tau}}$$

Le module et la phase sont alors respectivement :

$$|\hat{h}(\nu)| = \frac{1}{\sqrt{1 + 2g\cos(2\pi\nu\tau) + g^2}} \qquad \text{et} \qquad \angle\hat{h}(\nu) = -\operatorname{Arctan}\left(\frac{g\sin(2\pi\nu\tau)}{1 + g\cos(2\pi\nu\tau)}\right)$$

Programme 7.4.2: Ce programme affiche les mesures théoriques et experimentales.

```
% Parametres
g = 0.9;
Fe = 44100;
tau = 0.25 * Fe;
N = 10 * Fe;
nu = linspace(-0.5, 0.5, N); % Frequences reduites entre -0.5 et 0.5

% Calcul de la reponse en frequence theorique
h_theorique = 1 ./ (1 + g * exp(-2j * pi * nu * tau));

% Calcul de la reponse impulsionnelle numerique
X = zeros(1, N);
X(1) = 1;
a = zeros(1, tau + 1);
a(1) = 1;
a(tau + 1) = -g;
b = 1;
Y = filter(b, a, X);

% Calcul de la DFT de la reponse impulsionnelle numerique
h_numerique = fft(Y);

% Tracage des modules
figure;
plot(nu, abs(h_theorique), 'b', 'LineWidth', 2, 'DisplayName', 'Theorique');
hold on;
plot(nu, abs(h_numerique), 'r', 'LineWidth', 2, 'DisplayName', 'Numerique');
xlabel('Frequence reduite (\nu)');
ylabel('Module de la reponse en frequence');
title('Comparaison de la reponse en frequence theorique et numerique');
legend('Location', 'Best');
grid on;
```

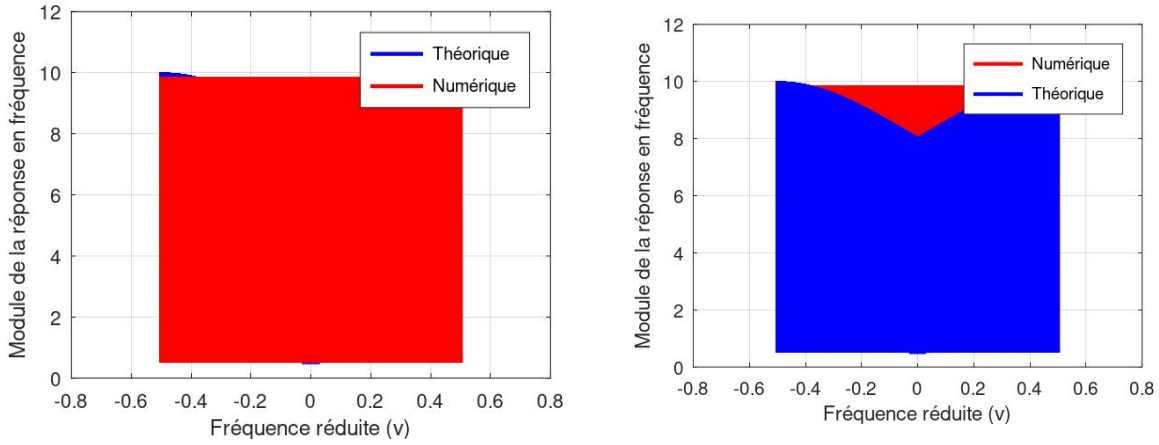


FIGURE 3 – Comparaison de la réponse en fréquence théorique et numérique

4.3 Application de l'effet de réverbération

Manipulation 7.4.2: Le signal le plus adapté pour approximer la réponse impulsionnelle de la pièce est le lignal xe1 car il se rapproche le plus d'une impulsion de Dirac.

Programme 7.4.3: Ce programme calcule la corrélation croisée entre les deux signaux audio, xe1 et xe2, et affiche les résultats sous forme de graphiques :

```
% Load the 'signal' package
pkg load signal

% Charger les donnees du fichier signal_excita
load('input/signal_excitation.mat');

% Calculer la correlation croisee entre les si
correlation_xe1 = xcorr(xe1, xe1);
correlation_xe2 = xcorr(xe2, xe2);

% Afficher les resultats
figure;
subplot(2, 1, 1);
plot(correlation_xe1);
title('Correlation croisee entre xe1 et xe1');

subplot(2, 1, 2);
plot(correlation_xe2);
title('Correlation croisee entre xe2 et xe2');
```

Programme 7.4.4: Ce programme calcule la corrélation croisée entre les deux signaux audio, xe1 et xe2, et affiche les résultats sous forme de graphiques :

```
% Charger les donnees du fichier signal_excitation.mat
load('input/signal_excitation.mat');
```

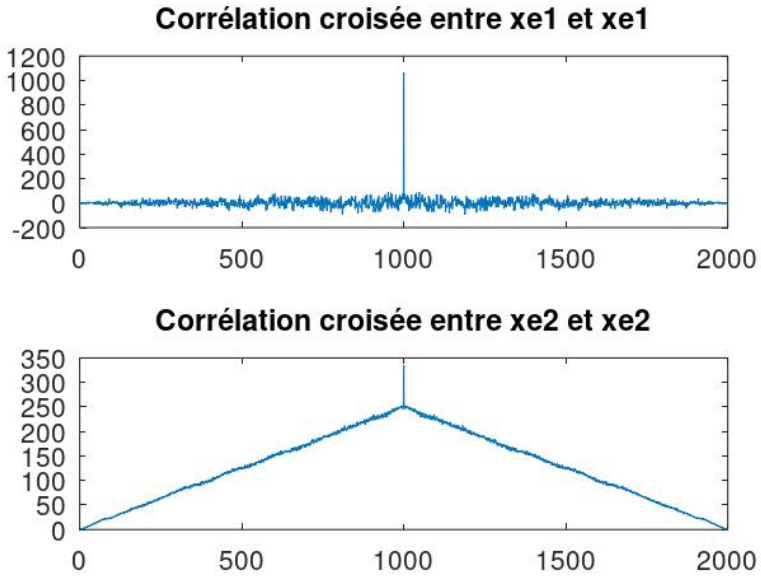


FIGURE 4 – Autocorrélation des deux signaux d'excitation

```
% Appeler la fonction simule_piece pour simuler l'effet de la piece
xe_simule = simule_piece(xe1, 44100);

% Calculer la correlation croisee entre le signal capte (xe_simule) et le signal d
correlation_yx = xcorr(xe_simule, xe1);

% Effectuer la transformation de Fourier inverse pour obtenir la reponse impulsion
h_estimee = ifft(correlation_yx);

% Trace de la reponse impulsionnelle estimee en fonction du temps
figure;
plot(1:length(h_estimee), h_estimee);
title('Reponse impulsionnelle estimee de la piece');
xlabel('Temps (echantillons)');
ylabel('Amplitude');
```

Programme 7.4.5: Ce programme calcule le produit de convolution du signal d'entré avec la réponse impulsionelle éstimée du filtre.

```
function y = effet_reverb(x, h_estimee)
    % Convolution du signal x avec la reponse impulsionnelle estimee
    y = filter(h_estimee, 1, x);
end
```

Manipulation 7.4.3: Temps d'exécution de la fonction effet reverb : 35.9500 secondes

```
% Charger les donnees du fichier signal_excitation.mat
load('input/signal_excitation.mat');
```

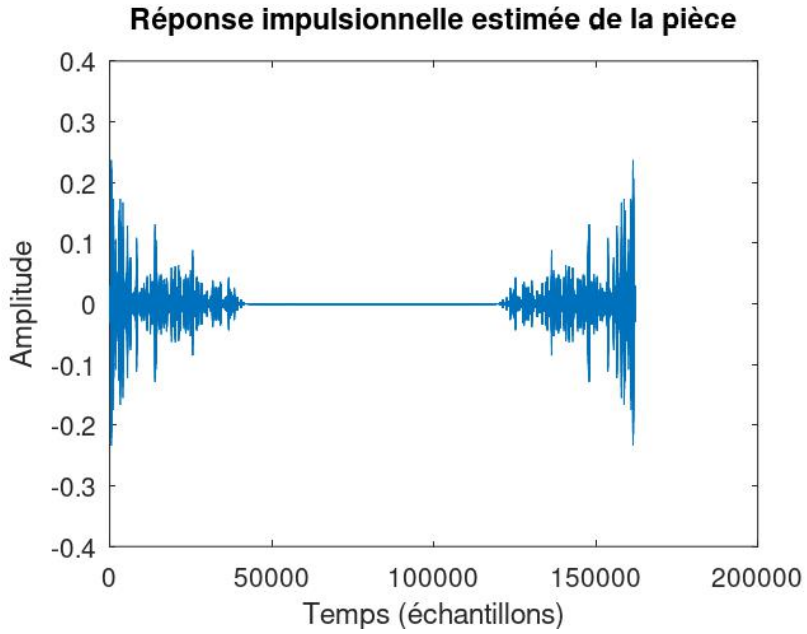


FIGURE 5 – Autocorrélation des deux signaux d'excitation

```
% Appeler la fonction simule_piece pour simuler l'effet de la piece
xe_simule = simule_piece(xe1, 44100);

% Calculer la correlation croisee entre le signal capte (xe_simule) et le signal d
correlation_yx = xcorr(xe_simule, xe1);

% Effectuer la transformation de Fourier inverse pour obtenir la reponse impulsion
h_estimee = ifft(correlation_yx);

% Charger les donnees audio (fichier "nylon_guitar.wav")
[x, Fe] = audioread('input/nylon-guitar.wav');

% Mesurer le temps de calcul de la fonction effet_reverb
tic; % Debut du chronometre
y_reverb = effet_reverb(xe_simule, h_estimee); % Appel de la fonction
temps_execution = toc; % Fin du chronometre

% Afficher le temps de calcul
fprintf('Temps d''execution de la fonction effet_reverb : %.4f secondes\n', temps_
```

Programme 7.4.6: Ce programme réalise un effet de réverbération sur un signal audio en utilisant le filtrage dans le domaine fréquentiel. Temps de calcul : 0.036413 secondes.

```
function y_reverb = effet_reverb_FFT(x, h_estimee)
    N = max([length(x) length(h_estimee)]);
    % Calcule la transformee de Fourier discrete de la reponse impulsionnelle
    H = fft(h_estimee, N);
```

```
    % Calcule la transformee de Fourier discrete du signal d'entree
    X = fft(x, N);

    % Effectue la multiplication dans le domaine frequentiel
    Y = H .* X;

    % Calcule l'inverse de la transformee de Fourier discrete pour obtenir le sign
    y_reverb = ifft(Y, N);
end
```

Question 7.4.7: La méthode utilisant la FFT pour effectuer la convolution est équivalente à la convolution classique de la section 7.4.3 en termes de résultat final. Les deux méthodes calculent la convolution entre un signal d'entrée et une réponse impulsionnelle pour obtenir un signal de sortie filtré. Cependant, la méthode basée sur la FFT peut être plus efficace en termes de temps de calcul, surtout lorsque les signaux d'entrée et de réponse impulsionnelle sont longs. Elle exploite la propriété de la transformée de Fourier qui permet de réaliser une convolution dans le domaine fréquentiel en effectuant une simple multiplication. Cela peut conduire à des gains significatifs de vitesse de calcul par rapport à la convolution classique, qui nécessite de calculer explicitement la somme des produits.

4.4 Application de l'effet de retard

Programme 7.4.7: Ce programme implémente un effet de retard audio en appliquant un filtre de retard à un signal d'entrée. Temps de calcul : 0.621052 secondes.

```
function y_delayed = effet_delay(x, delay_time, g, Fe)
    tau_samples = round(delay_time * Fe);

    y_delayed = zeros(size(x));

    % Application du filtre
    for n = 1:length(x)
        if n - tau_samples > 0
            y_delayed(n) = x(n) - g * x(n - tau_samples);
        else
            y_delayed(n) = x(n);
        end
    end
end
```

Programme 7.4.8: Ce programme implémente un effet de retard avec un filtre de moyenne glissante créant ainsi un effet de délai avec réverbération. Temps de calcul avec effet delay filtre : 2.262355 secondes.

```
function y_delayed = effet_delay_filtre(x, delay_time, g, K, Fe)
    % Calcul du nombre d'echantillons de retard
    tau_samples = round(delay_time * Fe);

    % Creation du vecteur de sortie initialise a zero
    y_delayed = zeros(size(x));
```

```
    % Appliquer le filtre de retard
    for n = 1:length(x)
        if n - tau_samples > 0
            y_delayed(n) = x(n) - g/K * sum(y_delayed(n - tau_samples:min(n - 1, n
        else
            y_delayed(n) = x(n);
        end
    end
end
```

Programme 7.4.9: Ce programme calcule et trace numériquement la réponse en fréquence d'un filtre $h_r(k)$.

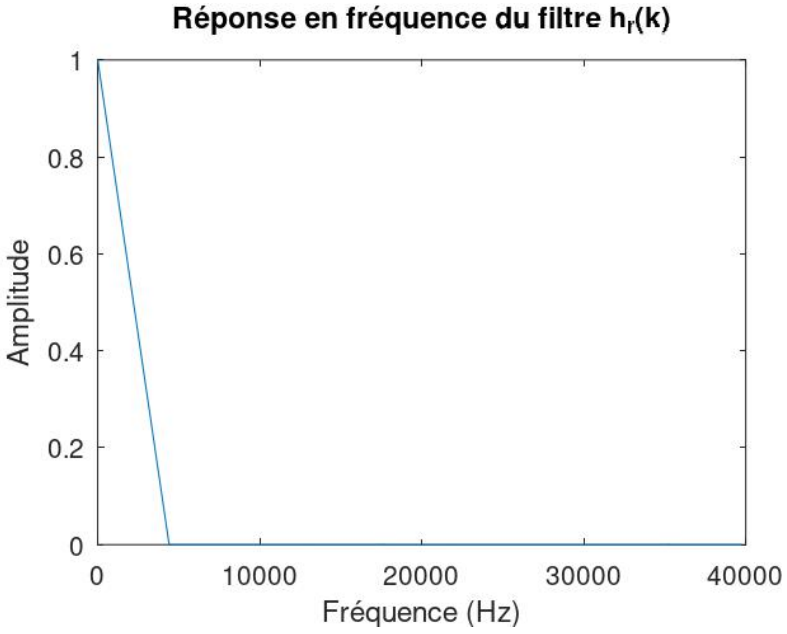
```
% Parametres du filtre
K = 10; % Longueur du filtre
Fs = 44100; % Frequence d'echantillonnage (Hz)

% Creation du filtre h_r(k)
h_r = ones(1, K) / K;

% Calcul de la reponse en frequence
N = length(h_r); % Longueur de la reponse impulsionnelle
f = (0:N-1) * Fs / N; % Frequences correspondantes

H_r = fft(h_r, N); % Transformee de Fourier discrete de h_r(k)

% Trace de la reponse en frequence
figure;
plot(f, abs(H_r));
title('Reponse en frequence du filtre h_r(k)');
xlabel('Frequence (Hz)');
ylabel('Amplitude');
```


Réponse en fréquence du filtre $h_r(k)$
Amplitude
Fréquence (Hz)
0
0.2
0.4
0.6
0.8
1
0
10000
20000
30000
40000