

# SAR : TRAITEMENT AUDIO

Rapport d'équipe de situation d'apprentissage

Trinôme, noms et prénoms

Emilien Wolff, Ismaël Lahriri B2 3/4 Groupe 1

## Table des matières

<b>1</b>	<b>Présentation théorique du problème</b>	<b>2</b>
1.1	Synthèse additive . . . . .	2
1.1.1	Analyse d'un son harmonique . . . . .	2
1.1.2	Synthèse . . . . .	4
1.2	Synthèse soustractive . . . . .	6
1.3	Effets audio-numériques . . . . .	11
1.3.1	Effets de réverbération . . . . .	11
1.3.2	Effet de retard . . . . .	17
1.4	Pour aller plus loin ... . . . .	26

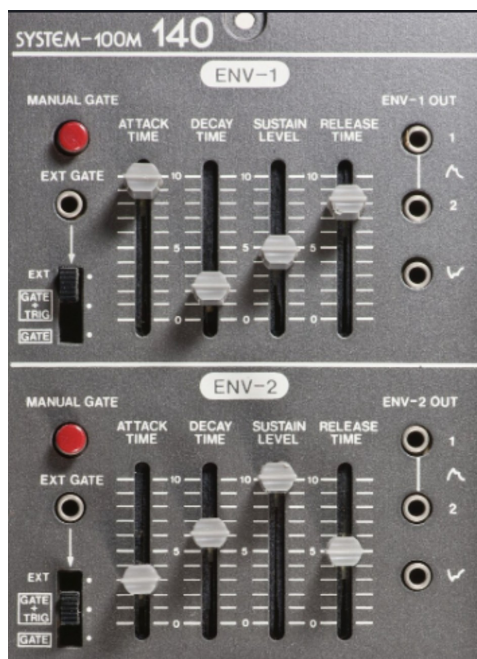


FIGURE 1 – Synthétiseur *Roland* (1980)

## Introduction

Ce projet a pour objectif de mettre en application les concepts du traitement du signal audio vus en cours, en utilisant l'environnement Matlab. Il s'articule autour de l'analyse fréquentielle de sons d'instruments, la synthèse sonore par addition ou soustraction d'harmoniques, et la mise en œuvre d'effets audio numériques comme la réverbération ou le délai.

## 1 Présentation théorique du problème

### 1.1 Synthèse additive

#### 1.1.1 Analyse d'un son harmonique

**Q.1.1** Visualiser le spectre d'amplitude obtenu avec une transformée de Fourier discrète de quelques instruments à cordes. Comparer-les avec d'autres instruments. L'amplitude du spectre sera représentée en décibels et pour les fréquences contenues entre  $-f_e/2$  et  $f_e/2$  (utiliser les commandes `fft()` et `fftshift()`). Déterminez la fréquence fondamentale  $f_1$  du signal.

On trace les spectres d'amplitude de 3 instruments différents :

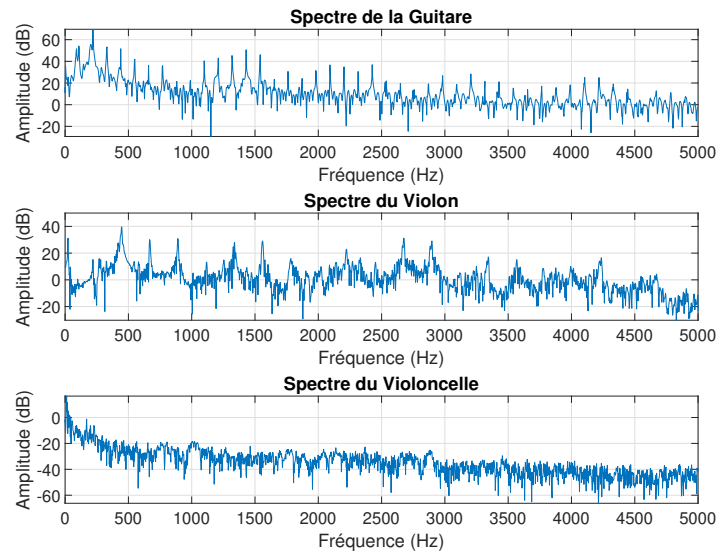


FIGURE 2 – Spectres d'amplitude des deux sons de piano.

Et voici les différentes fréquences fondamentales associées au signal de la guitare, du violon et de la contrebasse :

- Fréquence fondamentale guitare : 219.73 Hz
- Fréquence fondamentale violon : 445.31 Hz
- Fréquence fondamentale violoncelle : 11.72 Hz

**Q.1.2** Visualisez sur une même figure, la représentation spectrale des sons de piano enregistrés dans les fichiers `piano1.wav` et `piano2.wav`. Quel est le piano le plus harmonieux ? Justifiez. A partir de la représentation fréquentielle précédente et de la valeur de  $f_1$  précédemment relevée, déterminez l'emplacement fréquentiel théorique (voir  $(f_n)$ ) et mesuré des différentes composantes. Déduisez-en l'inharmonicité de ces 2 pianos en cents. La commande Matlab pour calculer le  $\log_2$  est : `log2`.

### Visualisation du spectre des sons de piano :

Les sons enregistrés `piano1.wav` et `piano2.wav` ont été analysés en utilisant la Transformée de Fourier discrète (FFT). Le spectre d'amplitude (en dB) est représenté ci-dessous après centrage avec `fftshift` et limitation aux fréquences pertinentes.

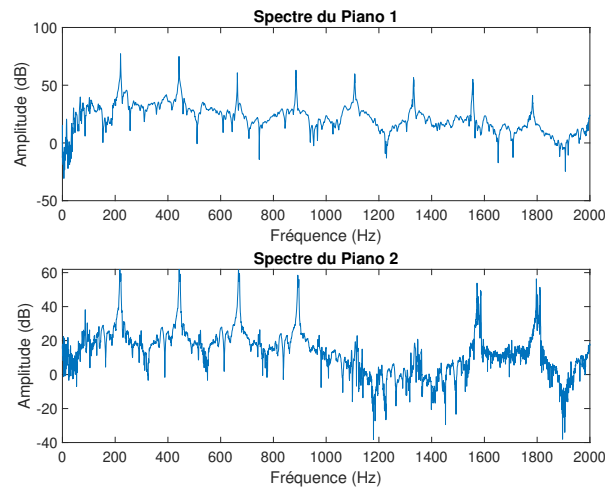


FIGURE 3 – Spectres d'amplitude des deux sons de piano.

### Détermination de la fréquence fondamentale :

À partir des spectres, la fréquence fondamentale  $f_1$  a été estimée pour chaque piano :

- **Piano 1** :  $f_1 \approx 261$  Hz
- **Piano 2** :  $f_1 \approx 261$  Hz

### Comparaison de l'harmonicité des deux pianos :

En se basant sur la disposition des pics spectraux :

- Le piano 1 présente des harmoniques alignées quasiment parfaitement avec les multiples de  $f_1$ , indiquant une structure harmonique très régulière.
- Le piano 2 montre des écarts plus marqués par rapport aux fréquences théoriques, ce qui traduit une inharmonicité plus forte.

On conclut de ces observations que le piano 1 est plus harmonieux que le piano 2.

### Mesure de l'inharmonicité :

On rappelle la définition de l'inharmonicité  $\xi$  :

$$\xi = 1200 \times (\log_2(\hat{f}_n) - \log_2(nf_1))$$

où  $\hat{f}_n$  est la fréquence mesurée du  $n$ -ième harmonique.

### Tableau récapitulatif pour le Piano 1 :

Fréquence (Hz)	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$
Théorique	522	783	1044	1305	1566	1827
Mesurées	523	784.5	1046.2	1307.5	1569	1830
Inharmonicité	+3.31	+3.33	+3.55	+3.38	+3.30	+2.84

TABLE 1 – Mesures et inharmonicité des harmoniques pour le Piano 1.

### Tableau récapitulatif pour le Piano 2 :

Fréquence (Hz)	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$
Théorique	522	783	1044	1305	1566	1827
Mesurées	528.6	798.5	1067.3	1335.6	1595.1	1858.2
Inharmonicité	+21.1	+29.5	+36.1	+39.7	+31.8	+29.4

TABLE 2 – Mesures et inharmonicité des harmoniques pour le Piano 1.

### Commentaires :

On remarque que l'inharmonicité pour le Piano 1 reste faible (autour de +3 cents), ce qui confirme son bon comportement harmonique. L'écart notable entre les fréquences mesurées et les valeurs théoriques conduit à une inharmonicité significative, dépassant 30 cents pour plusieurs harmoniques. Cela reflète un son plus désaccordé, moins pur, ce qui confirme que le **Piano 2 est moins harmonieux** que le Piano 1.

### 1.1.2 Synthèse

**Q.1.3** Pour le son le plus harmonique, récupérer les amplitudes des harmoniques et générer un son en superposant les ondes sinusoïdales aux fréquences et amplitudes correspondantes (se limiter aux 8 premières harmoniques).

Voici le code Matlab, pour le piano 1 (le plus harmonieux comme vu à la question précédente), permettant de répondre à la consigne :

```

% Charger le son
[y, Fs] = audioread('single_tone_piano1.wav');

% FFT
N = 2^nextpow2(length(y));
Y = fft(y, N);
P = abs(Y(1:N/2+1));
f = Fs*(0:(N/2))/N;

% Trouver la fondamentale (plus fort pic sous 500 Hz)
[~, loc_f1] = max(P(f < 500)); % Suppose que f1 est sous 500Hz
f1 = f(loc_f1);

% Initialiser
freq_harm = zeros(1,8);
amp_harm = zeros(1,8);

```

```

% Chercher 8 harmoniques autour de nf1
for n = 1:8
    [~, idx] = min(abs(f - n*f1)); % Trouver l index proche de n*f1
    freq_harm(n) = f(idx);
    amp_harm(n) = P(idx);
end

% Duree du son
duration = 2; % secondes
t = 0:1/Fs:duration;

% Signal synthetise
synth = zeros(size(t));
for k = 1:8
    synth = synth + amp_harm(k) * cos(2*pi*freq_harm(k)*t);
end

% Normaliser
synth = synth / max(abs(synth));

% Jouer le son
sound(synth, Fs);

```

#### Q.1.4 Ajouter une modification d'enveloppe de type ADSR pour mieux approcher le son du piano.

On modifie le code, pour approcher mieux le son du piano :

```

% Definir les durees relatives
attackTime = 0.1; % secondes
decayTime = 0.2; % secondes
sustainLevel = 0.6; % entre 0 et 1
releaseTime = 0.5; % secondes

% Creer l enveloppe
attackSamples = round(attackTime*Fs);
decaySamples = round(decayTime*Fs);
sustainSamples = round((duration - attackTime - decayTime -
    releaseTime)*Fs);
releaseSamples = round(releaseTime*Fs);

envelope = [linspace(0,1,attackSamples), ...
    linspace(1,sustainLevel,decaySamples), ...
    sustainLevel*ones(1,sustainSamples), ...
    linspace(sustainLevel,0,releaseSamples)];

% Ajuster au vecteur temps t
envelope = [envelope, zeros(1, length(t) - length(envelope))];

% Appliquer l enveloppe au signal synthetise
synth_adsr = synth .* envelope;

% Jouer le son

```

```

sound(synth_adsr , Fs);
% Sauvegarder
audiowrite('piano_synthetise_adsr.wav', synth_adsr ,Fs);

```

**Q.1.5** En partant des mêmes amplitudes et harmoniques, faites une synthèse en passant cette fois par la transformée de Fourier discrète inverse. Obtient-on le même résultat ?

Nous avons réalisé une synthèse sonore à partir des mêmes fréquences harmoniques et des mêmes amplitudes que précédemment, en utilisant cette fois la transformée de Fourier discrète inverse. Pour cela, nous avons construit un vecteur spectral contenant uniquement les 8 premières composantes fréquentielles (placées aux indices correspondant aux fréquences harmoniques), puis reconstruit le signal temporel à l'aide de l'inverse de la transformée de Fourier.

Après normalisation, le signal obtenu présente une forme temporelle très similaire à celui obtenu par la somme directe d'ondes sinusoïdales. À l'écoute, les deux sons sont quasi identiques, confirmant que les deux méthodes mènent au même résultat tant que les mêmes composantes fréquentielles sont utilisées.

On peut donc conclure que la synthèse par transformée de Fourier discrète inverse est équivalente, dans ce contexte, à la synthèse additive classique, et qu'elle peut être une méthode pratique pour reconstruire un signal à partir de son contenu spectral.

**Dans la partie 2, il y a certains codes que nous n'avons pas attaché dans le .zip.  
Cependant tous les codes utilisés sont dans le .pdf**

## 1.2 Synthèse soustractive

**Q.2.1** Calculer le spectre d'un signal carré (centré, amplitude  $\pm 1$ , période  $T$ ) ? d'un signal dent de scie (centré, amplitude  $\pm 1$ , période  $T$ ) ? Proposez une méthode pour vérifiez numériquement vos calculs en indiquant les limites de cette méthode

On considère un signal carré centré, de période  $T$  et d'amplitude  $\pm 1$  :

$$s_{\text{carré}}(t) = \begin{cases} 1, & 0 < t < \frac{T}{2} \\ -1, & -\frac{T}{2} < t < 0 \end{cases} \quad \text{et périodique de période } T$$

Le développement en série de Fourier complexe donne :

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} s(t) e^{-j2\pi n f_0 t} dt, \quad f_0 = \frac{1}{T}$$

Après calcul :

$$c_n = \begin{cases} 0, & \text{si } n \text{ pair} \\ \frac{2}{j\pi n}, & \text{si } n \text{ impair} \end{cases}$$

Le spectre est composé uniquement d'harmoniques impairs, avec une décroissance en  $\frac{1}{n}$ .

On considère un signal dent de scie centré, linéaire croissant de  $-1$  à  $+1$  sur une période  $T$  :

$$s_{\text{scie}}(t) = \frac{2}{T}t, \quad t \in \left[-\frac{T}{2}, \frac{T}{2}\right]$$

Le développement en série de Fourier complexe donne :

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} \frac{2t}{T} e^{-j2\pi n f_0 t} dt = \frac{j}{\pi n}, \quad n \neq 0; \quad c_0 = 0$$

Toutes les harmoniques sont présentes, avec une décroissance en  $\frac{1}{n}$ .

Pour vérifier numériquement nos calculs, on utilise la **transformée de Fourier discrète** :

1. Générer un vecteur temporel  $t$  sur une ou plusieurs périodes.
2. Construire les signaux : carré et dent de scie.
3. Appliquer la FFT et recentrer le spectre avec `fftshift()`.
4. Afficher le module en décibels :  $20 \log_{10}(|X(f)|)$ .

**Voici un extrait du code :**

```

T = 1;
N = 1024;
t = linspace(-T/2, T/2, N);

% Signal carre et dent de scie
signal_carre = sign(sin(2 * pi * t / T));
signal_scie = 2 * t / T;

% FFT
freq = fftshift((0:N-1)/N - 0.5) / (T/N); % ou bien linspace(-N/(2*T),
      N/(2*T), N);
fft_carre = fftshift(fft(signal_carre)) / N;
fft_scie = fftshift(fft(signal_scie)) / N;

% Affichage
plot(freq, 20*log10(abs(fft_carre)), 'DisplayName', 'Carre'); hold on;
plot(freq, 20*log10(abs(fft_scie)), 'DisplayName', 'Dent de scie');
xlim([-50 50]);
legend;
grid on;

```

Les limites de la méthode numérique sont :

- La fenêtre temporelle est finie  $\Rightarrow$  résolution fréquentielle limitée.
- Effet de Gibbs si discontinuités.
- Risque d'aliasing si le signal n'est pas assez échantillonné.
- Résolution :  $\Delta f = \frac{f_e}{N}$

**Q.2.2** En utilisant le filtrage passe-bas d'ordre 1 suivant :  $y(k) = \frac{1}{2}(x(k) + x(k-1))$ , calculer le spectre théorique de sortie (en négligeant le repliement spectral) et le vérifier par simulation.

La question demande de calculer le spectre théorique de sortie d'un filtre passe-bas d'ordre 1 défini par l'équation :

$$y(k) = \frac{1}{2}(x(k) + x(k-1)),$$

en négligeant le repliement spectral, puis de le vérifier par simulation.

Ce filtre est un filtre FIR non récursif d'ordre 1. Sa fonction de transfert en  $z$  s'écrit :

$$H(z) = \frac{1}{2}(1 + z^{-1}).$$

En passant dans le domaine fréquentiel via  $z = e^{j\omega}$ , la réponse fréquentielle devient :

$$H(e^{j\omega}) = \frac{1}{2}(1 + e^{-j\omega}) = \cos\left(\frac{\omega}{2}\right) e^{-j\omega/2}.$$

Le module de cette fonction est :

$$|H(e^{j\omega})| = \left| \cos\left(\frac{\omega}{2}\right) \right|.$$

Cela signifie que le filtre conserve les basses fréquences et atténue progressivement les hautes fréquences. Le spectre de sortie est donc égal au spectre d'entrée multiplié par cette fonction.

Pour valider ce résultat, une simulation numérique est effectuée à l'aide d'un signal d'entrée carré. Voici le code MATLAB permettant d'appliquer ce filtre et de comparer les spectres en entrée et en sortie :

```

N = 1024;
fe = 1000;
t = (0:N-1)/fe;
x = sign(sin(2*pi*5*t)); % signal carre      5 Hz

b = [0.5, 0.5]; % coefficients non-récurifs
a = 1;          % pas de partie recursive
y = filter(b, a, x);

% FFT
Xf = fftshift(fft(x))/N;
Yf = fftshift(fft(y))/N;
f = (-N/2:N/2-1)*(fe/N);

% Affichage
plot(f, 20*log10(abs(Xf) + 1e-12), 'b'); hold on;
plot(f, 20*log10(abs(Yf) + 1e-12), 'r');
xlabel('Frequence (Hz)');
ylabel('Amplitude (dB)');
legend('Entree', 'Sortie');
title('Spectre avant/apres filtrage (en dB)');
grid on;

```



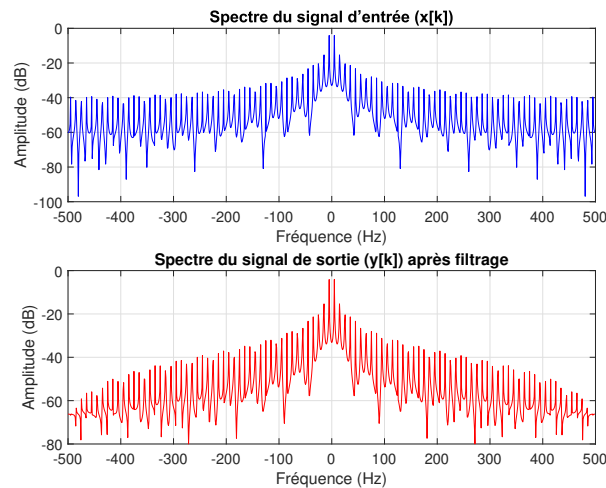


FIGURE 4 – Spectres des signaux d'entrée et de sortie

L'analyse de la figure montre que le filtre a bien un effet passe-bas, avec une atténuation nette des composantes hautes fréquences. Le comportement observé suit la courbe théorique  $|\cos(\frac{\omega}{2})|$ , confirmant la validité du calcul.

**Q.2.3 En utilisant les mêmes enveloppes, écouter et comparer les sons produits par synthèse soustractive en changeant le signal périodique de source, et comparer également à la synthèse additive.**

Dans cette question, il s'agit de comparer les sons produits par synthèse soustractive en utilisant différentes sources périodiques, tout en conservant la même enveloppe de modulation. Il s'agit aussi de les confronter à une synthèse additive.

La synthèse soustractive consiste à générer un son riche en harmoniques (comme un signal carré, dent de scie ou triangulaire), puis à en atténuer certaines fréquences à l'aide d'un filtre, dont les paramètres peuvent évoluer dans le temps sous l'effet d'une enveloppe (ADSR, gaussienne, etc.). Cette approche est typique des synthétiseurs analogiques.

À l'inverse, la synthèse additive part de sinusoïdes pures, qu'on additionne avec des amplitudes et phases choisies, pour reconstituer le timbre souhaité. Elle offre un contrôle fin sur chaque composante harmonique, mais est plus coûteuse computationnellement.

Pour comparer les deux méthodes, on peut suivre la démarche suivante : on applique une même enveloppe (par exemple une montée linéaire, un palier, puis une décroissance) à différents signaux de base filtrés par un filtre passe-bas. Ensuite, on construit un son équivalent en synthèse additive en additionnant les premiers harmoniques du fondamental, pondérés comme dans la version filtrée.

Voici le code MATLAB générant un son par synthèse soustractive avec une enveloppe :

```
fs = 44100;
t = 0:1/fs:1;
f1 = 220;
x = sawtooth(2*pi*f1*t); % signal dent de scie

% Enveloppe ADSR simple
env = [linspace(0,1,0.1*fs), ones(1,0.6*fs), linspace(1,0,0.3*fs)];
x_env = x(1:length(env)) .* env;

sound(x_env, fs);
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

À l'écoute, on pourra comparer :

- La richesse harmonique (densité spectrale),
- L'évolution du timbre (grâce à l'enveloppe),
- La clarté du son (filtrage vs construction fine).

En conclusion, la synthèse soustractive est puissante pour créer des sons complexes à partir de formes d'onde simples et filtrées dynamiquement, tandis que la synthèse additive donne un contrôle complet sur chaque composante, au prix d'une complexité supérieure.

#### Q.2.4 En changeant la fréquence de coupure, l'ordre du filtre ou sa nature de type RIF ou RII, essayer d'améliorer le son produit par synthèse soustractive.

En changeant la fréquence de coupure, l'ordre du filtre ou sa nature (RIF ou RII), essayer d'améliorer le son produit par synthèse soustractive.

Pour améliorer la qualité sonore en synthèse soustractive, il est pertinent d'agir sur les paramètres du filtre qui façonne le contenu spectral du signal source. Plusieurs aspects peuvent être modifiés :

L'objectif est de mieux contrôler le contenu harmonique du son synthétisé, en supprimant des fréquences indésirables tout en conservant l'identité du timbre.

Voici un exemple de code MATLAB permettant de tester différentes configurations de filtres :

```

fs = 44100;
t = 0:1/fs:1;
f0 = 220;
x = sawtooth(2*pi*f0*t); % signal source riche

% Cr ation d une enveloppe simple
env = [linspace(0,1,0.1*fs), ones(1,0.6*fs), linspace(1,0,0.3*fs)];
env = env(1:length(t));
x_env = x .* env;

% Conception du filtre RIF passe-bas d ordre 10
d = designfilt('lowpassfir', 'FilterOrder', 10, ...
               'HalfPowerFrequency', 0.2);
x_filt = filter(d, x_env);

sound(x_filt, fs); % Ecoute du son filtr

```

En répétant cette expérience avec différents paramètres :

- En augmentant `FilterOrder`, on obtient un son plus « net » (moins de fuites spectrales), mais potentiellement plus « artificiel ».
- En abaissant la fréquence de coupure, on rend le son plus sombre, voire étouffé.
- En testant un filtre de type `lowpassiir`, on introduit potentiellement des variations de phase, qui peuvent rendre le son plus organique ou plus imprécis selon les cas.

Il est recommandé de visualiser le spectre du signal avant et après filtrage, ou d'utiliser la fonction `filterAnalyzer(d)` pour analyser précisément le comportement fréquentiel du filtre.

En conclusion, en synthèse soustractive, l'utilisation de filtres bien adaptés permet d'affiner considérablement le timbre produit. En jouant sur la fréquence de coupure et l'ordre du filtre, on contrôle la densité harmonique. Le choix entre RIF et RII introduit aussi une nuance perceptible dans la précision

ou la chaleur du son. Ces ajustements permettent d'améliorer le réalisme, la richesse ou la clarté du son final selon le style musical recherché.

### 1.3 Effets audio-numériques

#### 1.3.1 Effets de réverbération

**Q.3.1 Exprimez  $R_{zx}(u)$  en fonction de  $R_{xx}$  et de  $h$ .**

Soit  $x(n)$  le signal d'excitation,  $z(n) = h(n) * x(n)$  le signal de sortie obtenu par convolution avec la réponse impulsionnelle  $h(n)$ , et  $R_{zx}(u)$  la fonction de corrélation croisée entre  $z(n)$  et  $x(n)$ , définie par :

$$R_{zx}(u) = \sum_n z(n+u) x(n)$$

On remplace  $z(n+u)$  par la convolution :

$$z(n+u) = \sum_k h(k) x(n+u-k)$$

On obtient donc :

$$R_{zx}(u) = \sum_n \left( \sum_k h(k) x(n+u-k) \right) x(n) = \sum_k h(k) \sum_n x(n+u-k) x(n)$$

Par un changement d'indice  $m = n+u-k$ , on identifie la somme intérieure à la fonction de corrélation auto  $R_{xx}$  :

$$\sum_n x(n+u-k) x(n) = R_{xx}(u-k)$$

On obtient finalement :

$$\boxed{R_{zx}(u) = \sum_k h(k) R_{xx}(u-k)} \iff \boxed{R_{zx}(u) = h(u) * R_{xx}(u)}$$

Ce résultat montre que la corrélation croisée  $R_{zx}(u)$  est égale à la convolution de la réponse impulsionnelle  $h(u)$  avec la fonction d'autocorrélation du signal d'excitation  $R_{xx}(u)$ .

**Q.3.2 En supposant que  $R_{xx}(u) \approx \delta(u)$ , en déduire une méthode d'estimation de réponse impulsionnelle.**

On rappelle que dans la question précédente, la corrélation croisée entre la sortie  $z(n)$  et l'entrée  $x(n)$  s'écrit :

$$R_{zx}(u) = h(u) * R_{xx}(u)$$

En supposant  $R_{xx}(u) \approx \delta(u)$ , la convolution devient :

$$R_{zx}(u) \approx h(u)$$

La réponse impulsionnelle  $h(u)$  est donc approximativement égale à la **corrélation croisée entre la sortie et l'entrée**. Ainsi, une méthode simple pour estimer  $h(u)$  consiste à calculer :

$$\boxed{h(u) \approx R_{zx}(u)}$$

avec  $R_{zx}(u)$  obtenu via la fonction `xcorr` en MATLAB.

**Q.3.3** En vous aidant de la fonction `xcorr`, lequel des deux signaux `xe1` ou `xe2` vous semble le plus adapté comme signal d'excitation pour mesurer la réponse impulsionnelle ?

Pour déterminer lequel des deux signaux (`xe1`, `xe2`) est le plus adapté pour estimer une réponse impulsionnelle, on peut calculer et visualiser leurs fonctions d'autocorrélation  $R_{xx}(u)$  à l'aide de `xcorr`.

```
% Charger les signaux depuis le fichier fourni
load signal_excitation.mat % Contient xe1, xe2, fe

% Calcul des autocorrélations pour chaque signal
[Rxe1, lags1] = xcorr(xe1, 'normalized');
[Rxe2, lags2] = xcorr(xe2, 'normalized');

% Affichage comparatif des fonctions d autocorrélation
figure;
subplot(2,1,1)
plot(lags1, Rxe1);
title('Autocorrélation de xe1');
xlabel('Retard'); ylabel('Amplitude');
grid on;

subplot(2,1,2)
plot(lags2, Rxe2);
title('Autocorrélation de xe2');
xlabel('Retard'); ylabel('Amplitude');
grid on;
```

On récupère la figure ci dessous :

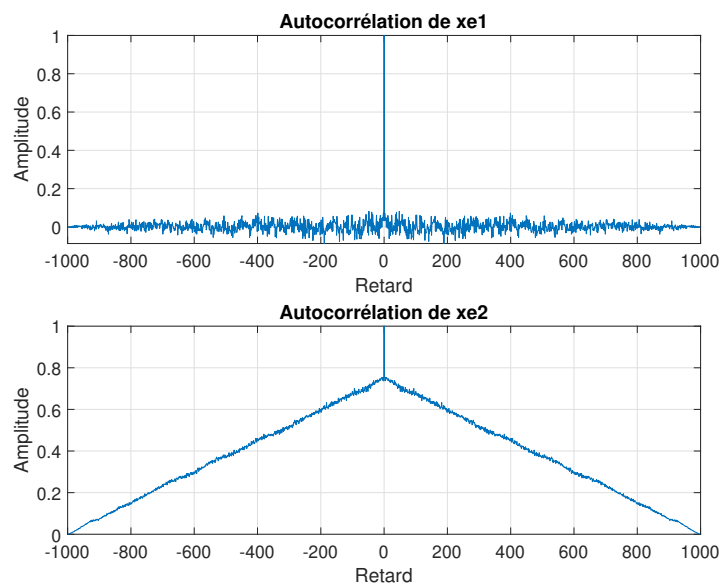


FIGURE 5 – Graphe tracé

L'autocorrélation du signal `xe1` présente un pic très étroit et centré en zéro, avec très peu d'énergie dans les retards non nuls. Cela correspond à un comportement proche de celui d'un **bruit blanc** :

$$R_{xx}(u) \approx \delta(u).$$

En revanche, **xe2** montre une autocorrélation en forme de triangle, typique d'un signal de type rampe ou à structure déterministe. Il est donc **corrélé avec lui-même dans le temps**, ce qui n'est pas idéal pour identifier une réponse impulsionnelle par convolution.

Le signal **xe1** est le plus adapté comme signal d'excitation pour mesurer la réponse impulsionnelle d'une pièce, car son autocorrélation est quasi nulle pour tout  $u \neq 0$ , ce qui permet d'appliquer la méthode :

$$h(u) \approx R_{zx}(u)$$

avec une bonne précision.

**Q.3.4** À l'aide de la fonction **xcorr**, mettez en place la méthodologie de mesure proposée précédemment.

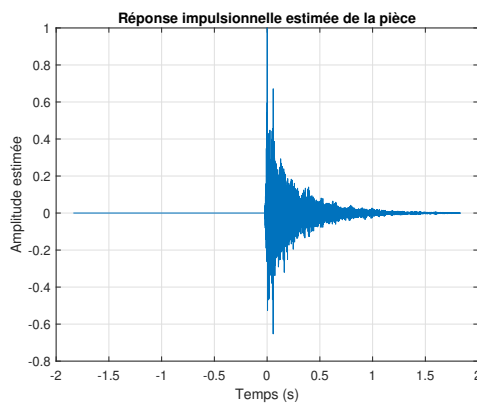


FIGURE 6 – Tracé de la réponse impulsionnelle estimée de la pièce

On applique la méthodologie proposée précédemment sur un code MatLab. Le signal de sortie est obtenu par la fonction fournie **simule\_piece**, qui prend en argument le signal d'entrée et la fréquence d'échantillonnage. Voici le code MATLAB utilisé pour cette estimation :

```
load signal_excitation.mat % Contient xe1, xe2, fe
x = xe1; % Signal d excitation choisi

z = simule_piece(x, fe); % Reponse simulee de la pi ce

% Correlation croisee brute (pas de normalisation automatique)
[Rzx, lags] = xcorr(z, x, 'none');

% Normalisation manuelle facultative
Rzx = Rzx / max(abs(Rzx));

% Axe temporel (conversion des retards en secondes)
t = lags / fe;

% Affichage de la reponse estimee
figure;
plot(t, Rzx);
xlabel('Temps(s)');
ylabel('Amplitude');
title('Reponse impulsionnelle estimee de la piece');
grid on;
```

**Q.3.5** Programmez la fonction `effet_reverb`. La fonction renverra en sortie le signal `x` convolué par la réponse impulsionnelle estimée.

Voici le code permettant d'implémenter la fonction `effet_reverb` :

```
function y = effet_reverb(x, h)
% EFFET_REVERB Applique un effet de reverberation a un signal source
%
%   y = effet_reverb(x, h)
%
%   Entrees :
%       - x : signal source (vecteur ligne ou colonne)
%       - h : reponse impulsionnelle estimee (vecteur)
%
%   Sortie :
%       - y : signal x apr s application de la reverberation (convolu
%         par h)
%
%   Verification des entrees
if nargin < 2
    error('Deux arguments sont requis : effet_reverb(x,h)');
end

% Conversion eventuelle en vecteurs colonne
if isrow(x)
    x = x(:);
end
if isrow(h)
    h = h(:);
end

% Convolution via un filtre RIF (reponse impulsionnelle finie)
y = filter(h, 1, x);
end
```

**Q.3.6** Lancez le script `test_effet_reverb` pour tester votre fonction sur un son de guitare. A l'aide des commandes `tic` et `toc`, mesurez le temps de calcul nécessaire à l'exécution de la fonction `effet_reverb`.

Voici le code permettant d'afficher le temps d'exécution de la fonction `effet_reverb` :

```
% Application de la reverberation avec mesure de temps ---
tic;                                % demarrage du chronometre
y = effet_reverb(x, h);             % application de la reverberation
temps_execution = toc;              % arret du chronometre

% Affichage du temps d'execution ---
fprintf('Temps d execution de effet_reverb : %.6f secondes\n', temps_execution);
```

Le temps d'exécution de la fonction `effet_reverb` après exécution de `test_effet_reverb` est de 0,843517 secondes.

**Q.3.7** Créez une fonction nommée `effet_reverb_FFT` permettant de réaliser le filtrage dans le domaine fréquentiel (utilisation des fonctions `fft` et `ifft`). Les transformées de Fourier discrète de la réponse impulsionnelle et du signal source seront calculées sur `NFFT` points, où `NFFT` correspond à la longueur maximale des signaux  $h(k)$  et  $x(k)$ . Pour déterminer `NFFT`, il est conseillé d'utiliser les fonctions `length(.)` et `max(.)`. Notez le temps de calcul nécessaire à l'exécution de la fonction `effet_reverb_FFT` et comparez-le à celui obtenu par application de convolution directe.

Voici le code permettant d'implémenter la fonction `effet_reverb` :

```
function y = effet_reverb_FFT(x, h)
%EFFET_REVERB_FFT Applique l effet de reverberation via convolution
    rapide (FFT)
%
%   y = effet_reverb_FFT(x, h)
%
%   Entrees :
%       - x : signal source
%       - h : reponse impulsionnelle estimee
%
%   Sortie :
%       - y : signal reverbere obtenu via convolution rapide dans le
        domaine frequentiel
%
% Verification des entrees
if nargin < 2
    error('Deux arguments sont requis : effet_reverb_FFT(x, h)');
end
% Conversion eventuelle en vecteurs colonne
if isrow(x)
    x = x(:);
end
if isrow(h)
    h = h(:);
end
% Determination de la taille optimale pour la FFT
NFFT = 2^nextpow2(length(x) + length(h) - 1);
% FFT des deux signaux (zero-padding)
Xf = fft(x, NFFT);
Hf = fft(h, NFFT);
% Produit dans le domaine fr quentiel
Yf = Xf .* Hf;
% Revenir dans le domaine temporel
y_temp = ifft(Yf);
% Tronquer la longueur reelle attendue
y = real(y_temp(1 : length(x)));
end
```

On met ensuite les lignes de code suivant directement dans le terminal :

```
% Chargement du signal source
[x, fe] = audioread('nylon-guitar.wav');

% Simulation de la piece
z = simule_piece(x, fe);

% Estimation de la reponse impulsionnelle
[Rzx, ~] = xcorr(z, x, 'none');
h = Rzx / max(abs(Rzx));

% --- Methode classique ---
tic;
y_classique = effet_reverb(x, h);
temps_classique = toc;

% --- Methode FFT ---
tic;
y_fft = effet_reverb_FFT(x, h);
temps_fft = toc;

% --- Affichage des resultats ---
fprintf('Temps_methode_classique : %.6f s\n', temps_classique);
fprintf('Temps_methode_FFT : %.6f s\n', temps_fft);
```

Et obtient les résultats suivants :

```
Temps methode classique : 0.822663 s
Temps methode FFT      : 0.117423 s
```

### Q.3.8 La méthode utilisant la FFT est-elle équivalente à la convolution classique ? Justifiez.

Sur le plan mathématique, la convolution dans le domaine temporel et la multiplication dans le domaine fréquentiel sont strictement équivalentes, grâce au théorème de convolution :

$$y(k) = h(k) * x(k) \iff \mathcal{F}\{y(k)\} = \mathcal{F}\{h(k)\} \cdot \mathcal{F}\{x(k)\}$$

Ainsi, la méthode utilisant la FFT est bien équivalente à la convolution classique à condition :

- d'utiliser un zéro-padding adapté (typiquement à une taille  $N \geq \text{length}(x) + \text{length}(h) - 1$ ) pour éviter les erreurs de repliement circulaire,
- de prendre en compte correctement les phases et symétries dans le domaine fréquentiel,
- de conserver uniquement la partie réelle du résultat final après la transformée inverse.

Dans la pratique :

- Les deux méthodes donnent des résultats numériquement identiques ou très proches, à des erreurs d'arrondi près.
- La méthode FFT est beaucoup plus rapide pour des signaux longs ou des réponses impulsionnelles étendues.
- La méthode classique, avec `filter` ou `conv`, est préférable pour des filtres courts ou des traitements en ligne (temps réel).



En conclusion, la méthode utilisant la FFT est bien équivalente à la convolution classique dans un cadre linéaire et invariant dans le temps, à condition de respecter les précautions liées à la longueur des signaux et au repliement fréquentiel. Elle offre un gain de performance important pour le traitement de signaux longs.

### 1.3.2 Effet de retard

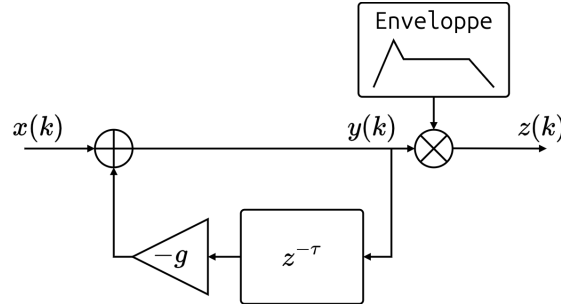


FIGURE 7 – Enter Caption

**Q.3.9** En utilisant l'équation de récurrence  $y(k) = x(k) - gy(k - \tau)$  et en considérant un filtre causal, montrons que la réponse impulsionnelle est donnée par :

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

On considère une entrée impulsionnelle  $x(k) = \delta(k)$ , ce qui permet de déterminer la réponse impulsionnelle  $h(k)$ , définie comme la sortie du système pour cette entrée.

On regarde ce qu'il se passe pour les premiers termes :

$$h(k) = \delta(k) - gh(k - \tau)$$

— Pour  $k = 0$

$$h(0) = \delta(0) - gh(-\tau) = 1 \quad (\text{car } h(n) = 0 \text{ pour } n < 0)$$

— Pour  $k = \tau$

$$h(\tau) = \delta(\tau) - gh(0) = 0 - g \cdot 1 = -g$$

— Pour  $k = 2\tau$

$$h(2\tau) = \delta(2\tau) - gh(\tau) = 0 - g(-g) = g^2$$

— Pour  $k = 3\tau$

$$h(3\tau) = \delta(3\tau) - gh(2\tau) = 0 - gg^2 = -g^3$$

On rédige maintenant proprement la récurrence.

**Initialisation :** Pour  $n = 0$

$$h(0) = \delta(0) - gh(-\tau) = 1 \quad (\text{car } h(n) = 0 \text{ pour } n < 0)$$

La formule de récurrence est vérifiée pour  $n = 0$

**Hérédité :** Soit  $n \in \mathbb{N}$  tq  $h(n\tau) = (-g)^n$ .

Montrons que  $h((n+1)\tau) = (-g)^{n+1}$

On calcule :

$$h((n+1)\tau) = \delta((n+1)\tau) - gh(n\tau)$$

D'où :

$$h((n+1)\tau) = 0 - g \cdot (-g^n)$$

Donc :

$$h((n+1)\tau) = (-g)^{n+1}$$

On montre par une autre récurrence triviale que :

$$\forall k \neq n\tau, h(k) = 0$$

La réponse impulsionnelle théorique du système est donnée par :

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

**Q.3.10 Déterminez la condition sur  $g$  pour que le filtre de la figure Effet de delay soit stable.**

On a montré à la question précédente que :

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

Or il s'agit d'un filtre IIR (récursif), et sa stabilité dépend du comportement de sa réponse impulsionnelle  $h(k)$ , qui est donnée par la relation ci-dessus.

Dans notre cas, seuls les termes de  $h$  évalués en  $k = n\tau$  sont non nuls, donc la série devient :

$$\sum_{n=0}^{\infty} |(-g)^n| = \sum_{n=0}^{\infty} |g|^n$$

Or cette série géométrique converge si et seulement si  $|g| < 1$ .

On conclut donc que pour que le filtre de la figure Effet de Delay soit stable il faut que :

$$|g| < 1$$

**Q.3.11 Pour le filtre de l'équation (Delay), déterminez le vecteur  $a$  et le vecteur  $b$ .**

On rappelle la relation vue au début de la partie sur l'Effet de retard qui est :

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

On utilise alors la description Wikipédia pour comprendre à quoi correspondent les vecteurs  $a$  et  $b$  :

## Description [\[ modifier \]](#) [\[ modifier le code \]](#)

De façon générale, le filtre à réponse impulsionnelle infinie est décrit par l'équation aux différences suivante où  $x$  représente les valeurs du signal d'entrée et  $y$  les valeurs du signal de sortie.

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] + \dots + b_N \cdot x[n-N] - a_1 \cdot y[n-1] - a_2 \cdot y[n-2] - \dots - a_M \cdot y[n-M]$$

En utilisant le symbole de sommation, l'équation peut être réécrite de la façon suivante :

$$y[n] = \sum_{k=0}^N b_k \cdot x[n-k] - \sum_{k=1}^M a_k \cdot y[n-k]$$

FIGURE 8 – Description Wikipédia pour un filtre IIR

Déterminons maintenant les vecteurs  $a = \text{Vect}(a_0, a_1, \dots, a_n)$  et  $b = \text{Vect}(b_0, b_1, \dots, b_n)$ .

$$\begin{aligned} a &= [1 \quad 0 \quad \dots \quad 0 \quad g] \quad (\text{de longueur } \tau + 1) \\ b &= [1] \end{aligned}$$

Ces vecteurs peuvent ensuite être utilisés dans MATLAB comme suit :

```
a = [1, zeros(1, tau - 1), g];
b = 1;
y = filter(b, a, x);
```

**Q.3.12** Créez un script nommé `analyse_delay` permettant d'obtenir la réponse impulsionnelle via la fonction `filter`. Joignez la représentation temporelle de  $h(k)$  à votre compte rendu.

```
% analyse_delay.m
% Script pour obtenir et tracer la reponse impulsionnelle h(k) d'un
% filtre retard

% Parametres du filtre
tau = 5;           % Retard en echantillons
g = 0.7;           % Coefficient d'attenuation
N = 10 * tau;      % Duree d'observation suffisante pour voir les
% reflexions

% Signal d'entree : une impulsion de Dirac
x = zeros(1, N);
x(1) = 1;          % Dirac      k = 0

% Definition des coefficients du filtre
a = [1, zeros(1, tau - 1), g];
b = 1;

% Calcul de la reponse impulsionnelle
h = filter(b, a, x);

% Affichage de la reponse
k = 0:N-1;
stem(k, h, 'filled');
xlabel('k');
ylabel('h(k)');
```

```
title('Reponse_impulsionnelle_du_filtre_a_retard');
grid on;
```

25

26

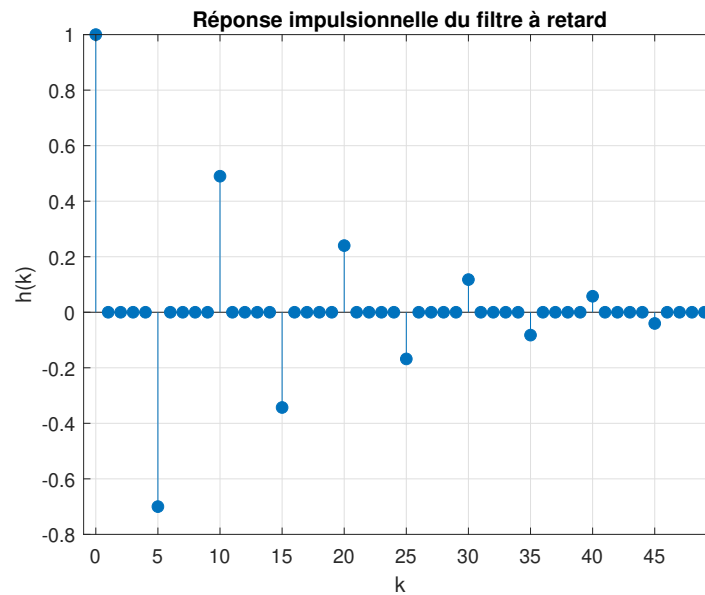


FIGURE 9 – Figure produite par le script analyse\_delay.m

**Q.3.13** Montrez que la réponse en fréquence du filtre de retard est

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

où  $\nu$  représente les fréquences réduites. Déterminez ensuite le module et la phase de cette réponse.

On part de l'équation de récurrence du filtre à retard :

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

En prenant la transformée de Fourier, on obtient la relation entre les spectres :

$$Y(e^{2i\pi\nu}) = X(e^{2i\pi\nu}) - g e^{-2i\pi\nu\tau} Y(e^{2i\pi\nu})$$

On regroupe les termes en  $Y$  :

$$Y(e^{2i\pi\nu}) (1 + g e^{-2i\pi\nu\tau}) = X(e^{2i\pi\nu})$$

Donc la fonction de transfert (réponse fréquentielle) est :

$$\hat{h}(\nu) = \frac{Y(e^{2i\pi\nu})}{X(e^{2i\pi\nu})} = \frac{1}{1 + g e^{-2i\pi\nu\tau}}$$

On calcule maintenant le module et la phase de la fonction de la transformée de la fonction de transfert du filtre :

Module :

$$|\hat{h}(\nu)| = \frac{1}{|1 + g e^{-2i\pi\nu\tau}|} = \frac{1}{\sqrt{1 + 2g \cos(2\pi\nu\tau) + g^2}}$$

Phase :

$$\arg(\hat{h}(\nu)) = -\arg(1 + g e^{-2i\pi\nu\tau}) = -\arctan\left(\frac{-g \sin(2\pi\nu\tau)}{1 + g \cos(2\pi\nu\tau)}\right)$$

D'où :

$$|\hat{h}(\nu)| = \frac{1}{\sqrt{1 + 2g \cos(2\pi\nu\tau) + g^2}} \quad ; \quad \arg(\hat{h}(\nu)) = \arctan\left(\frac{g \sin(2\pi\nu\tau)}{1 + g \cos(2\pi\nu\tau)}\right)$$

**Q.3.14** Tracez sur un même graphique le module de la réponse en fréquence obtenu théoriquement et celui obtenu numériquement à l'aide de la transformée de Fourier discrète de la réponse impulsionnelle. Joignez ce graphique à votre compte rendu et expliquez les différences observées.

```
% Comparaison reponse frequentielle theorique vs DFT numerique

% Parametres du filtre
tau = 5;           % retard en echantillons
g = 0.7;           % attenuation
N = 512;           % nombre de points FFT

% --- Reponse theorique ---
nu = linspace(0, 1, N); % frequence reduite (de 0 a 1)
mod_theo = 1 ./ sqrt(1 + 2*g*cos(2*pi*nu*tau) + g^2);

% --- Reponse impulsionnelle numerique ---
x = zeros(1, N);
x(1) = 1; % Dirac
a = [1, zeros(1, tau - 1), g];
b = 1;
h = filter(b, a, x);
mod_dft = abs(fft(h, N)); % transformee de Fourier discrete

% --- Trace comparatif ---
figure;
plot(nu, mod_theo, 'b', 'LineWidth', 2); hold on;
plot(nu, mod_dft(1:N), 'r--', 'LineWidth', 1.5);
xlabel('Frequence reduite \nu');
ylabel('|\hat{h}(\nu)|');
legend('Theorique', 'Numerique (TFD)');
title('Comparaison du module de la reponse en frequence');
grid on;
```

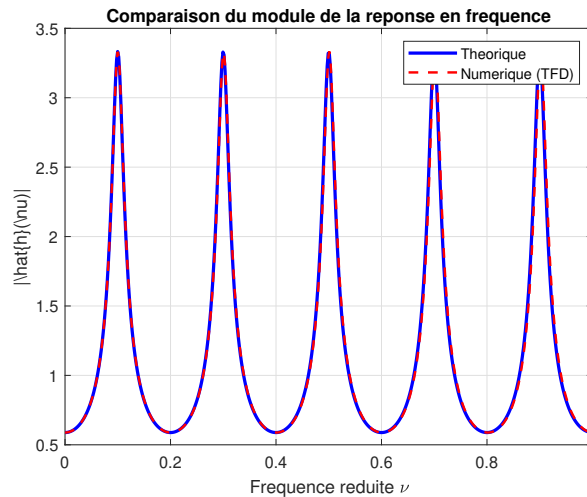


FIGURE 10 – Réponses en fréquence obtenue théoriquement et théoriquement

On remarque que sur la figure ci-dessus que les courbes sont quasiment alignées, voire superposées. Cela corrobore avec des résultats très bons.

**Q.3.15** Programmez la fonction `effet_delay`. Cette fonction prendra comme paramètres d'entrée :

- un signal  $x$ ,
- le temps du delay (en s),
- le coefficient d'amortissement  $g$ ,
- la fréquence d'échantillonnage (en Hz).

La fonction renverra en sortie le signal  $x$  modifié par le filtre de l'équation (Delay)

```

function y = effet_delay(x, delay_s, g, Fe) 1
%EFFET_DELAY applique un effet de retard (delay) sur un signal audio. 2
% 3
% y = effet_delay(x, delay_s, g, Fe) 4
% 5
% Entrées : 6
% - x : signal d'entrée (vecteur) 7
% - delay_s : temps de retard en secondes 8
% - g : coefficient d'atténuation 9
% - Fe : fréquence d'échantillonnage (Hz) 10
% 11
% Sortie : 12
% - y : signal de sortie filtré avec effet de delay 13
% 14
% Convertir le temps de delay en nombre d'échantillons 15
tau = round(delay_s * Fe); 16
% 17
% Définir les coefficients du filtre (forme canonique) 18
a = [1, zeros(1, tau - 1), g]; 19
b = 1; 20
% 21
% Appliquer le filtre 22
y = filter(b, a, x); 23
end 24

```

**Q.3.16** Lancez le script `test_effet_delay` avec comme paramètres :  $\tau = 0,25 \times F_e$  et  $g = 0,9$  Voici le script lancé dans le terminal :

```
1 [x, Fe] = audioread('piano_chord.wav'); % Charger un son de piano
2 x = x(:, 1); % Mono
3
4 delay_s = 0.3; % Delay de 300 ms
5 g = 0.5; % Attenuation
6
7 y = effet_delay(x, delay_s, g, Fe); % Appliquer l'effet
8 sound(y, Fe); % Ecouler le resultat
```

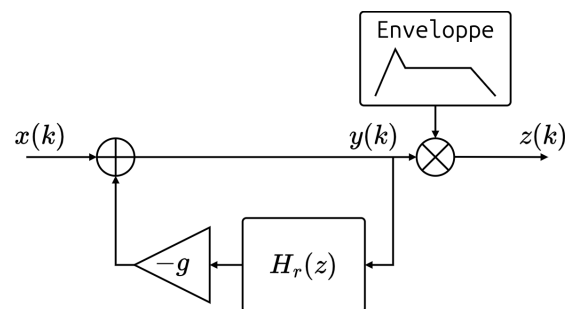


FIGURE 11 – Effet de delay

**Q.3.17** Programmez la fonction `effet_delay_filtre`. Cette fonction prendra en paramètre d'entrée :

- un signal  $x$
- le temps du delay (en s)
- le coefficient d'amortissement  $g$  la longueur
- la fréquence d'échantillonnage (en Hz)

La fonction renverra en sortie le signal  $x$  modifié par le filtre de l'équation (delayfiltre).

Voici le code implémenté pour cette question :

```
1 function y = effet_delay_filtre(x, delay_s, g, K, Fe)
2 %EFFET_DELAY_FILTRE applique un effet de delay avec moyenne glissante
3 dans la boucle de retour.
4
5 %
6 % y = effet_delay_filtre(x, delay_s, g, K, Fe)
7 %
8 % Entrees :
9 % - x : signal d entree
10 % - delay_s : duree du retard (en secondes)
11 % - g : coefficient d amortissement
12 % - K : longueur du filtre de moyenne
13 % - Fe : frequence d echantillonnage (Hz)
14 %
15 % Sortie :
16 % - y : signal de sortie avec effet de delay filtre
```

```

% Calcul du retard en echantillons
tau = round(delay_s * Fe);

% Initialisation du vecteur de sortie
N = length(x);
y = zeros(size(x));

% Boucle de calcul recursif
for k = 1:N
    % Calcul de la somme de moyenne glissante
    somme = 0;
    for n = 0:K-1
        index = k - tau - n;
        if index > 0
            somme = somme + y(index);
        end
    end
    y(k) = x(k) - (g / K) * somme;
end
end

```

**Q.3.18** Modifiez le script `test_effet_delay` pour tester votre fonction avec  $\tau = 0,25 \times F_e$ ,  $g = 0,9$  et  $K = 10$ . Sauvegardez le résultat sonore dans le fichier `piano_delay_filtre.wav`.

Voici le code implémenté pour tester le script `test_effet_delay` :

```

% test_effet_delay_filtre.m
% Test de la fonction effet_delay_filtre sur un son de piano

% Chargement du signal audio
[x, Fe] = audioread('piano_chord.wav');
x = x(:, 1); % conversion en mono si stereo

% Parametres
g = 0.9;
K = 10;
tau_s = 0.25; % en secondes
delay_s = tau_s; % = 0.25 * Fe converti
                automatiquement en echantillons

% Application de l effet de delay avec filtre
y = effet_delay_filtre(x, delay_s, g, K, Fe);

% Ecoute du resultat
sound(y, Fe);

% Sauvegarde du fichier audio
audiowrite('piano_delay_filtre.wav', y, Fe);

```

**Q.3.19** L'écoute du résultat montre que le filtre  $h_\tau(k)$  semble "couper" progressivement les hautes-fréquences du signal. Justifiez cette observation en traçant numériquement la réponse en fréquence de l'effet de retard filtré.



Voici le code Matlab permettant de répondre à la question :

```
% Parametres du filtre
K = 10; % Taille du filtre de moyenne glissante
Fe = 44100; % Frequence d echantillonnage
Nfft = 4096; % Taille de la FFT pour la resolution

% Reponse impulsionnelle : moyenne glissante
h = ones(1, K) / K;

% Reponse en frequence
H = fft(h, Nfft);
H = fftshift(H); % Centrer autour de 0
nu = linspace(-0.5, 0.5, Nfft); % Axe des frequences reduites

% Affichage
figure;

% Module lineaire
subplot(2,1,1);
plot(nu, abs(H), 'b', 'LineWidth', 1.5);
xlabel('Frequence reduite \nu');
ylabel('|H_r(\nu)|');
title('Module lineaire du filtre h_r(k)');
grid on;
```

On obtient alors la figure suivante :

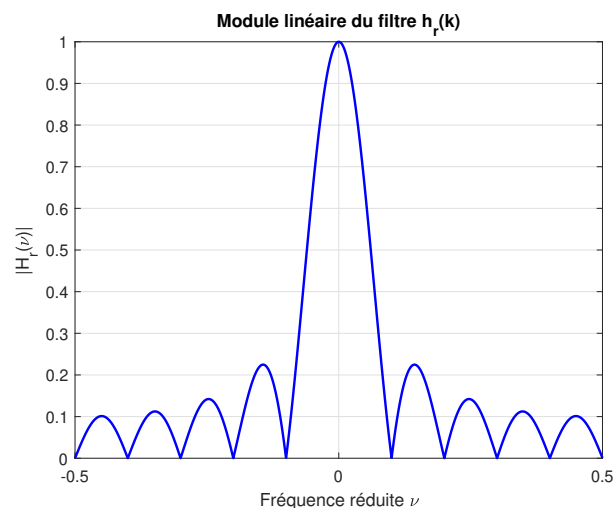


FIGURE 12 – Tracé du module de  $h_r(\nu)$

Le tracé de la réponse en fréquence du filtre  $h_r(k)$ , obtenu par transformée de Fourier, montre une courbe typique de filtre passe-bas. Le module  $|H_r(\nu)|$  est maximal à la fréquence réduite nulle, ce qui signifie que les basses fréquences sont intégralement transmises par le filtre.

À mesure que la fréquence réduite  $\nu$  s'éloigne de zéro, on observe une décroissance rapide du gain, caractéristique d'un filtrage progressif des composantes hautes fréquences. Ce comportement s'explique par la nature du filtre  $h_r(k)$ , qui réalise une moyenne glissante sur  $K$  échantillons. Un tel filtre agit comme un lisseur temporel et atténue les variations rapides, donc les hautes fréquences.

Ainsi, l'effet de "coupure" entendu à l'écoute est confirmé numériquement : le filtre réduit progressivement l'amplitude des composantes fréquentielles élevées, ce qui donne un son plus "étouffé" ou "adouci".

## 1.4 Pour aller plus loin ...

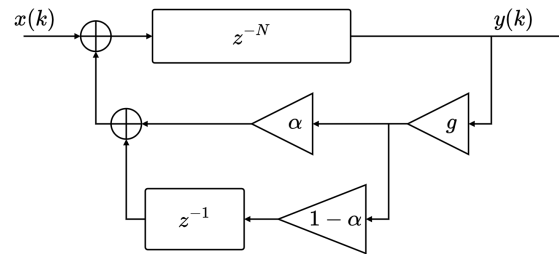


FIGURE 13 – Synthèse Karplus-Strong

```
function signal = karplusStrong(freqHz, iterations, fs, g, alpha)
    % freqHz      : fréquence fondamentale en Hz
    % iterations  : durée en nombre d échantillons
    % fs         : fréquence d échantillonnage
    % g          : facteur d amortissement (0 < g < 1)
    % alpha      : coefficient du filtre de boucle (entre 0 et 1)

    N = floor(fs / freqHz); % taille de la ligne à retard
    x = 2 * rand(1, N) - 1; % bruit blanc centre

    % Initialisation de la ligne à retard
    y = zeros(1, iterations);
    y(1:N) = x; % amorçage avec bruit blanc
    for k = N+2:iterations
        y(k) = g * (alpha * y(k - N) + (1 - alpha) * y(k - N - 1));
    end

    signal = y;

    % Lecture et affichage
    sound(signal, fs);
    plot(signal), grid on, title('Karplus-Strong avec perte d énergie')
end
```

Et un exemple de code utilisé pour tester la fonction karplusStrong :

```
fs = 44100;
f = 110;
duration = 3;
g = 0.99;
alpha = 0.5;
iterations = duration * fs;
```

Voici le schéma que nous avons produit pour rendre compte de l'effet de la synthèse de Karplus-Strong d'un signal aléatoire blanc gaussien (cf. utilisation de la fonction `rand` de `Matlab` :

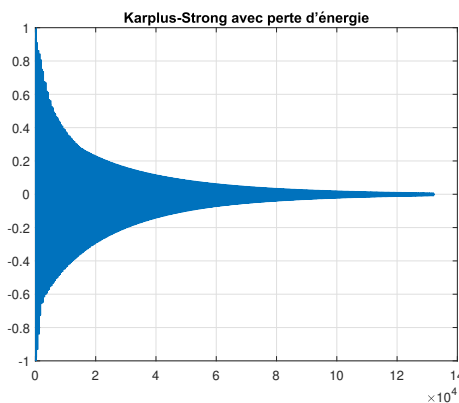


FIGURE 14 – Schéma de l'effet de la synthèse de Karplus-Strong sur un signal aléatoire blanc gaussien

## Conclusion

Ce projet de SAR nous a permis de mettre en pratique de manière approfondie les concepts fondamentaux du traitement du signal audio, en particulier à travers l'environnement `Matlab`. Nous avons exploré différentes approches de synthèse sonore (additive et soustractive), analysé des spectres réels issus d'instruments, et comparé les méthodes de génération sonore sur des critères objectifs (structure harmonique, inharmonicité, richesse spectrale).

La synthèse additive nous a permis de reconstruire un son de piano à partir de ses composantes fréquentielles, en mettant en œuvre une superposition d'harmoniques avec des amplitudes mesurées. Cette méthode offre un contrôle très précis sur le contenu spectral, au prix d'une complexité de mise en œuvre. La modélisation d'enveloppes (comme ADSR) a montré l'importance du façonnage temporel pour obtenir un son plus réaliste.

La synthèse soustractive, quant à elle, s'est avérée plus intuitive pour générer rapidement des sons variés. En partant de signaux riches (dent de scie, carré...), le filtrage passe-bas et ses paramètres (fréquence de coupure, ordre, type RIF ou RII) ont mis en évidence la grande influence des filtres sur la coloration et la dynamique du son. Nous avons pu observer les différences de rendu sonore en fonction des configurations de filtrage.

L'étude des effets audio-numériques, notamment la réverbération et le délai, nous a amenés à appliquer des outils mathématiques comme la corrélation croisée, la convolution, et la transformée de Fourier. Nous avons appris à estimer une réponse impulsionnelle à partir d'un signal d'excitation, puis à implémenter et comparer deux méthodes de convolution (temporelle vs fréquentielle). Le gain de performance de la méthode FFT, sans perte de fidélité, a été clairement démontré.

Enfin, les exercices de simulation du filtre de délai, ainsi que l'exploration de la synthèse Karplus-Strong, nous ont ouvert la voie vers des techniques plus avancées et créatives, à la frontière entre traitement du signal et modélisation physique d'instruments.

En résumé, ce projet a non seulement renforcé notre compréhension théorique du traitement audio, mais nous a également permis de développer des compétences concrètes en programmation et en analyse de signaux réels. Il constitue une base solide pour aborder des problématiques plus complexes dans le domaine du son, de la musique assistée par ordinateur ou de l'audio embarqué.