UV 4.8 Parole, Musique et Sons
-2-

Effets Audionumériques

Vincent Choqueuse

contact : vincent.choqueuse@univ-brest.fr

Table des matières

Travaux Pratiques 1

Effets Linéaires invariants dans le temps

Objectifs: Un grand nombre d'effets sonores peuvent être modélisés par des systèmes linéaires. Ces systèmes sont couramment désignés sous le terme générique "filtre".
Dans ce TP, nous allons nous intéresser à l'implémentation des effets linéaires invariants en temps. Cette classe comporte par exemple
- l'effet de delay,
- l'effet de réverberation,
- les effets de spatialisation.

1.1 Avant propos

Considérons un signal sonore $x[n]$ ($n = 0, \cdots, N-1$) échantillonné à une fréquence d'échantillonnage de F_e Hz. Les échantillons du signal sonore sont ensuite modifiés par un système (voir figure 1.1). Les échantillonsl obtenus en sortie du système sont notés $y[n]$.

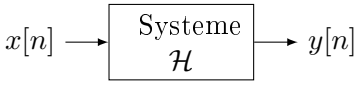


FIGURE 1.1 – Modification du signal

Dans ce chapitre, nous allons nous focaliser sur les systèmes linéaires invariants en temps (désignés par l'acronyme anglais LTI) . Ces systèmes possèdent les propriétés suivantes :

- Linéarité. Un système linéaire respecte le principe de superposition. Soit deux entrées $x_1[n]$ et $x_2[n]$ dont les sorties sont respectivement notées $y_1[n]$ et $y_2[n]$. Un système est linéaire si et seulement si il produit pour l'entrée $z[n] = \alpha x_1[n] + \beta x_2[n]$ le signal de sortie $\alpha y_1[n] + \beta y_2[n]$.
- Invariance en temps. Un système invariant en temps est un système qui répercute tout retard en entrée sur la sortie du système. Mathématiquement, l'invariance en temps est respectée si et seulement le système produit pour l'entrée $x[n - \tau]$ ($\tau \in \mathbb{N}$) le signal de sortie $y[n - \tau]$.

Les systèmes invariants en temps sont entièrement caractérisés par leur réponse impulsionnelle

c-a-d leur réponse à l'impulsion $\delta[n]$, où $\delta[n]$ est définie par

$$\delta[n] = \begin{cases} 1 & \text{si } n = 0 \\ 0 & \text{ailleurs} \end{cases} \tag{1.1}$$

Notons $h[n]$ la réponse impulsionnelle du système. La réponse d'un système linéaire invariant en temps à une entrée quelconque $x[n]$ est donnée par le produit de convolution (discret) entre l'entrée et la réponse impulsionnelle c-a-d [OS09]

$$y[n] = h[n] * x[n] \tag{1.2}$$

$$= \sum_{k=-\infty}^{\infty} h[k]x[n-k] \ (n \in \mathbb{Z}) \tag{1.3}$$

Dans la classe des systèmes linéaires invariants en temps, une sous-classe importante correspond aux filtres décrit par la relation de récurrence suivante :

$$a_0 y[n] = \sum_{m=0}^{M} b_m x[n-m] - \sum_{k=1}^{N} a_k y[n-k] \tag{1.4}$$

où a_k $(k > 0)$ et b_m correspondent respectivement aux coefficients de la partie récursive et non-récursive du filtre. Lorsque $a_k = 0$ pour tout $k > 0$, la filtre ne possède pas de partie récursive. Nous parlons alors de filtre à réponse impulsionnelle finie (FIR) [1]. A l'opposé, lorsque le filtre possède une partie récursive nous parlons de filtre à réponse impulsionnelle infinie (IIR).

1.2 Effet de delay

Les effets de retard (delay) sont très utilisés en musique. Ces effets permettent d'ajouter de l'espace en simulant les réflexions des ondes acoustiques dans un espace clos.

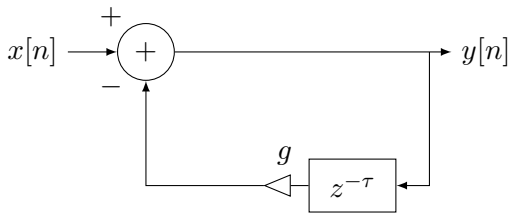


FIGURE 1.2 – Effet de delay (le symbole z^{τ} désigne un retard de τ échantillons).

Pour simuler un effet de retard, nous allons considérer le système décrit par le schéma de la figure 1.2. Mathématiquement, cet effet peut être modélisé par la relation de récurrence suivante ($\tau \in \mathbb{N}$) :

$$y[n] = x[n] - gy[n-\tau] \tag{1.5}$$

Remarquons que cette équation s'obtient en posant $a_0 = b_0 = 1$, $a_\tau = g$ et les autres coefficients à 0 dans (1.4). Le système décrit par (1.5) désigne un filtre IIR puisque le système possède une partie récursive non nulle. La partie récursive permet de simuler les réflexions

1. Pour montrer que l'absence de partie récursive implique une réponse impulsionnelle finie, il suffit d'identifier l'équation (1.4) avec (1.3) lorsque $N = 0$. On constate alors que $h[k] = 0$ lorsque $k > M$

du son. Ainsi le signal de sortie est renvoyé avec τ échantillons de retard et un coefficient d'atténuation g. Naturellement, le coefficient g doit être strictement inférieur à 1 pour éviter des problèmes de stabilité. Nous allons dans les manipulations suivantes implémenter cet effet sous Matlab. Dans toutes les questions qui suivent, le signal $x[n]$ est supposé échantillonné à $F_e = 44100$Hz.

1.2.1 Analyse théorique

Dans ce paragraphe, nous allons analyser la réponse impulsionnelle et la réponse fréquentielle de l'effet de delay.

Question 1.1. En utilisant l'équation (1.5), montrer que la réponse impulsionnelle théorique est égale à :

$$h[n] = \begin{cases} (-g)^k & \text{si } n = k\tau \\ 0 & \text{sinon} \end{cases} \tag{1.6}$$

Nous allons déterminer sous Matlab la réponse impulsionnelle du filtre en programmant (bêtement) l'équation (1.5). La réponse impulsionnelle s'obtient en posant $x[n] = \delta[n]$ dans (1.5). Les échantillons $y[n]$ seront ensuite calculés les uns après les autres via une boucle `for`.

Manipulation 1.1. Créer un nouveau script nommé `RI_delay` permettant de calculer la sortie du filtre $y[n]$ pour $n = 0, 1, \cdots, 10F_e$. Le facteur d'atténuation et le retard seront respectivement fixés à 0.9 et 0.25s. Afficher puis lire le résultat $y[n]$. Joindre la représentation temporelle de $y[n]$ à votre compte rendu. Comparer la réponse impulsionnelle obtenue avec la réponse impulsionnelle théorique (1.6).

Pour calculer la réponse du filtre, nous avons utilisé une boucle (et également une condition). Matlab possède une commande permettant de calculer directement la réponse d'un filtre défini par l'équation générale (1.4) :

```
>> y=filter(b,a,x);
```

où :
- `x` est un vecteur contenant le signal d'entrée,
- `a`$= [a_0, \cdots, a_N]$ est un vecteur comportant les coefficients de la partie récursive,
- `b`$= [b_0, \cdots, b_M]$ est un vecteur comportant les coefficients de la partie non-récursive.

Question 1.2. Pour le filtre de l'équation (1.5), déterminer le vecteur `a` et le vecteur `b`.

Manipulation 1.2. Créer un nouveau script nommé `analyse_delay` permettant de reproduire la manipulation 1.1 via la fonction `filter`. Joindre la représentation temporelle de $y[n]$ à votre compte rendu.

La réponse impulsionnelle décrit le comportement filtre dans le domaine temporel. Comme mentionné auparavant, la sortie du filtre s'obtient en convoluant la réponse impulsionnelle par l'entrée. Le produit de convolution est un opérateur compliqué. Pour mieux comprendre le comportement du filtre, il est plus simple de raisonner à partir de sa réponse fréquentielle. Cette réponse décrit la modification (module, phase) de chaque composante fréquentielle entre l'entrée et la sortie du filtre.

Théoriquement, la réponse fréquentielle s'obtient en deux temps : 1) calcul de la transformée en $\mathcal{Z}$ de l'équation (1.4) puis 2) passage au domaine fréquentiel en posant $z = e^{2j\pi f/F_e}$. En appliquant la procédure précédente à l'équation (1.5), nous obtenons :

$$H(f) = \frac{1}{1 + ge^{-2j\pi f\tau/F_e}} \tag{1.7}$$

Le module et la phase sont respectivement donnés par

$$|H(f)| = \frac{1}{\sqrt{1 + 2g\cos(2\pi f\tau/F_e) + g^2}} \tag{1.8}$$

$$\text{Arg}[H(f)] = -\text{atan}\left(\frac{-g\sin(2\pi f\tau/F_e)}{1 + g\cos(2\pi f\tau/F_e)}\right) \tag{1.9}$$

Nous pouvons alors constater que la module du filtre est maximal pour $f_k = \frac{(2k+1)F_e}{2\tau}$ $(k \in \mathbb{Z})$ et que $|H(f_k)| = \frac{1}{1-g}$. Notre effet est donc un filtre en peigne.

Matlab possède une fonction permettant d'accéder rapidement à la représentation fréquentielle d'un filtre décrit par l'équation générale (1.4).

```
>>fvtool(b,a,'Fs',Fe);
```

où :
- `a`= $[a_0, \cdots, a_N]$ est un vecteur comportant les coefficients de la partie récursive,
- `b`= $[b_0, \cdots, b_M]$ est un vecteur comportant les coefficients de la partie non-récursive,
- `Fe` correspond à la fréquence d'échantillonnage (en Hz).

Cette fonction `fvtool` permet d'éviter le calcul analytique de la réponse fréquentielle.

Manipulation 1.3. Ajouter à votre script `analyse_delay`, la fonction `fvtool` pour permettre la visualisation de la réponse fréquentielle. Une fois le script lancé, aller dans le menu `Analysis > Analysis Parameters` puis régler le paramètre `Number of Points` à $F_e = 44100$ Hz. Joindre le module de la représentation fréquentielle à votre compte rendu. A quelles fréquences apparaissent les maxima de la réponse fréquentielle ? Déterminer l'amplitude des maxima ? Justifier ces valeurs à partir de l'analyse théorique précédente.

1.2.2 Application

Nous allons à présent programmer l'effet de delay sous forme d'une fonction Matlab. Nous testerons ensuite cette fonction sur un son de piano.

Manipulation 1.4. Programmer 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 F_e (en Hz).

La fonction renverra en sortie le signal `x` modifié par le filtre de l'équation (1.5).

Nous allons tester cette fonction sur un accord de piano.

Manipulation 1.5. Lancer le script test_effet_delay avec comme paramètres : $\tau = 0.25$s et $g = 0.9$.

A l'écoute, nous observons que cet effet produit des sonorités "synthétiques". Pour rendre l'effet plus naturel, nous allons modifier le son après réflexion en filtrant le signal dans la boucle de retour. Cette modification est illustrée par la figure 1.3.

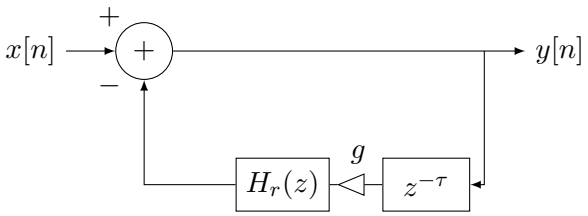


FIGURE 1.3 – Effet de delay ($H_r(z)$ désigne un filtre de réponse impulsionnelle $h_r[n]$).

Dans la boucle de retour, nous choisissons un filtre dont la réponse impulsionnelle est donnée par :

$$h_r[n] = \begin{cases} \frac{1}{K} & \text{si } n = 0, 1, ..., K-1 \\ 0 & \text{sinon} \end{cases} \tag{1.10}$$

Ce filtre réalise une moyenne glissante sur K échantillons. L'effet de delay s'exprime alors par la relation de récurrence suivante :

$$y[n] = x[n] - \frac{g}{K} \sum_{k=0}^{K-1} y[n - \tau - k] \tag{1.11}$$

Manipulation 1.6. Programmer 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 K,
- la fréquence d'échantillonnage F_e (en Hz).

La fonction renverra en sortie le signal `x` modifié par le filtre de l'équation (1.11).

Manipulation 1.7. Modifier le script `test_effet_delay` pour tester votre fonction avec $\tau = 0.25$, $g = 0.9$ et $K = 10$. Sauvegarder le résultat sonore dans le fichier piano_delay_filtre.wav.

L'écoute du résultat montre que le filtre $h_r[n]$ semble "couper" progressivement les hautes-fréquences du signal. Ce comportement peut se vérifier en utilisant la fonction `fvtool`.

1.3 Effet de réverbération

La réverbération permet d'ajouter de l'espace au son en simulant sa propagation dans un environnement le plus souvent fermé (salle, amphithéâtre, arène, église, etc). Physiquement, la réverbération est un mécanisme complexe faisant intervenir des phénomènes de réflexion, réfractions, etc. Ces phénomènes sont traités par une discipline nommée l'"acoustique des salles". Lorsque l'environnement n'évolue pas en fonction du temps, la réverbération peut être modélisée par un système linéaire invariant en temps. La réverbération est alors complétement décrite par sa réponse impulsionnelle.

Différentes approches peuvent être utilisées pour simuler les effets de réverbération.

- L'approche algorithmique (algorithme de Moorer, Feedback Delay Network,...),
- L'approche physique (lancer de rayon, technique image-source, méthode des éléments-finis),
- L'approche par convolution.

Dans cette sous-section, nous allons implémenter la troisième approche. L'approche par convolution s'implémente facilement et produit des résultats très convaincants. Elle est composée des étapes suivantes :

1. Acquisition de la réponse impulsionnelle $h[n]$ (l'empreinte acoustique) d'un lieu en générant une sonorité proche d'une impulsion (coup de feu, éclatement d'un ballon)
2. Convolution d'un son $x[n]$ par la réponse impulsionnelle.

L'un des intérêts majeurs de cette technique réside dans le fait que seule la première étape nécessite de se déplacer dans un lieu spécifique avec un dispositif d'enregistrement. Une fois l'acquisition de la réponse impulsionnelle réalisée, la deuxième étape peut être réalisée tranquillement chez soit.

Plusieurs réponses impulsionnelles sont disponibles dans le répertoire `effet_reverb`. Ces réponses sont causales et finie c-a-d que $h[n] = 0$ pour $n < 0$ et pour $n \geq L$.

1.3.1 Convolution classique

Manipulation 1.8. Créez un script nommé `analyse_reverb.m` réalisant les opérations suivantes :

- Chargement de la réponse impulsionnelle `RI_epic.wav` dans le vecteur `h`,
- Affichage de la réponse impulsionnelle en fonction du temps (en sec).
- Lecture de la réponse impulsionnelle `h`.

Joindre la réponse impulsionnelle à votre compte rendu.

L'application de l'effet de réverbération s'obtient en convoluant la réponse impulsionnelle avec un signal source c-a-d $y[n] = h[n] * x[n]$. Comme la réponse impulsionnelle est causale, la convolution s'exprime sous la forme :

$$y[n] = \sum_{k=0}^{L-1} h[k]x[n-k] \ (n \in \mathcal{N}) \tag{1.12}$$

En identifiant cette équation avec l'équation (1.4), nous remarquons que cette convolution peut être implémentée par un filtre FIR dont les coefficients sont donnés par :

$$a_k = \delta[k] \tag{1.13}$$
$$b_k = h[k] \tag{1.14}$$

L'effet de réverbération pourra donc être implémentée sous Matlab via la fonction `filter`.

Manipulation 1.9. Programmer la fonction `effet_reverb`. Cette fonction prendra en paramètre d'entrée :

- un signal `x`.

La fonction renverra en sortie le signal `x` convolué par la réponse impulsionnelle contenue dans le fichier `RI_epic.wav`.

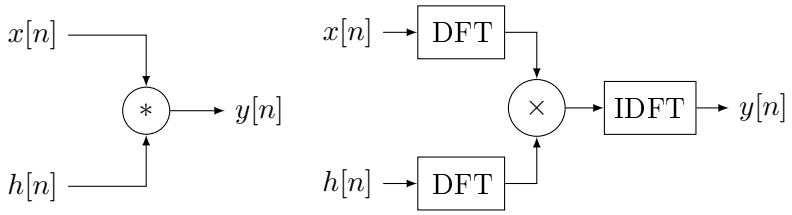
Manipulation 1.10. Lancer le script `test_effet_reverb` pour tester votre fonction. Notez le temps de calcul nécessaire à l'exécution de la fonction `effet_reverb`.

1.3.2 Convolution rapide

Pour diminuer le temps de calcul, nous allons effectuer l'effet de réverbération dans le domaine fréquentiel via une technique nommée "convolution rapide". La convolution équivaut dans le domaine fréquentielle à une simple multiplication c-a-d :

$$DFT\left[h[n] * x[n]\right] = DFT\left[h[n]\right].DFT\left[x[n]\right] \tag{1.15}$$

où $DFT\left[h[n]\right]$ et $DFT\left[x[n]\right]$ représentent respectivement la Transformée de Fourier Discrète de $h[n]$ et de $x[n]$. Pour pouvoir réaliser la multiplication, ces deux transformées de Fourier doivent être calculées sur un même nombre de points, noté `NFFT`. Le filtrage est alors obtenu par la méthodologie décrite par la figure 1.4.



(a) Technique classique (b) Convolution rapide

FIGURE 1.4 – Technique d'implémentation de la convolution.

Question 1.3. Quel(s) inconvénient(s) présente cette approche ? (un peu d'imagination).

Manipulation 1.11. Creer 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 $h[n]$ et $x[n]$. 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`.

Le calcul de la DFT peut être réalisé plus rapidement lorsque la longueur du signal est une puissance de 2. Pour satisfaire cette condition, nous allons ajouter plusieurs zéros à la suite des signaux $x[n]$ et $h[n]$ (zero-padding).

Manipulation 1.12. Dans la fonction `effet_reverb_FFT`, remplacez le calcul de `NFFT` par :

```
>>NFFT=2^nextpow2(max(length(h),length(x)));
```

Notez alors le temps de calcul nécessaire à l'exécution de la fonction `effet_reverb_FFT`.

Nous allons ajouter à votre fonction de nouvelles fonctionnalités.

Manipulation 1.13. Le répertoire `effet_reverb` contient 3 réponses impulsionnelles. Ajouter à votre fonction `effet_reverb_FFT` une deuxième entrée de type entier, nommée `RI`, permettant de choisir la réponse impulsionnelle. Modifier votre fonction pour que la réponse impulsionnelle corresponde aux fichiers :

- `RI_bunker.wav` lorsque `RI`= 1,
- `RI_epic.wav` lorsque `RI`= 2,

– `RI_stairwell.wav` lorsque `RI`= 3.

Testez le bon fonctionnement de votre programme.

Notre effet applique une réverbération importante au signal. La plupart des effets audios possèdent un paramètre Dry-Wet permettant de calibrer l'importance de l'effet. Lorsque l'effet est réglé sur Dry, le signal de sortie correspond au signal d'entrée sans effet. A l'inverse, lorsque l'effet est réglé sur Wet, le signal de sortie correspond au signal d'entrée après réverbération. En réglant, le paramètre sur des positions intermédiaires, il est possible de contrôler l'importance de l'effet. La figure 1.5 montre une implémentation possible du paramètre Dry-Wet.

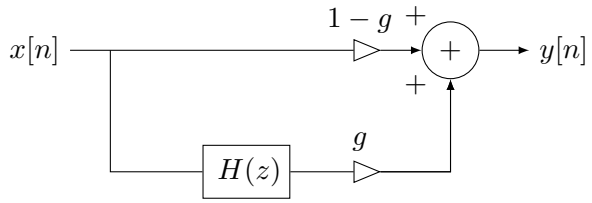


FIGURE 1.5 – Effet audionumérique avec contrôle Dry-Wet.

La valeur de g contrôle l'importance de l'effet :

$$g = \begin{cases} 0 & \text{signal Dry} \\ 0 < g < 1 & \text{signaux Dry-Wet mélangés} \\ 1 & \text{signal Wet} \end{cases} \tag{1.16}$$

Manipulation 1.14. Ajouter à votre fonction `effet_reverb` une troisième entrée, nommée `g`, permettant de contrôler l'importance de l'effet. Testez le bon fonctionnement de votre programme.

1.4 Effet de spatialisation au casque

Pour connaitre la position spatiale d'une source sonore dans l'espace, l'homme utilise ses deux oreilles ainsi que son cerveau. La différence entre les signaux arrivant à l'entrée du canaux auditifs droit et gauche permettent aux cerveaux d'estimer la position spatiale. Pour simuler un effet de spatialisation, nous allons donc artificiellement synthétiser ces différences.

La figure 1.6 illustre la propagation du signal entre son émission et son entrée dans les canaux auditifs. Le système liant la source sonore $x[n]$ et les signaux reçus $y_g[n]$ et $y_d[n]$ est un système comportant une entrée et deux sorties.

1.4.1 Spatialisation par ITD

La figure 1.6 montre que le trajet du signal entre son émission et l'oreille gauche n'est pas le même qu'entre son émission et l'oreille droite. Par exemple lorsque la source sonore est plus proche de l'oreille gauche, le trajet entre la source et l'oreille gauche sera plus court que le trajet entre la source et notre deuxième oreille. Cette différence de distance se traduit par une différence de temps à l'arrivée (ITD : Interaural Time Difference). En champ lointain, la valeur de l'ITD (en seconde) peut s'exprimer en fonction de la position azimutale θ de la source sous la forme [Zol11] :

$$\forall\theta = [-\pi\ \pi],\ \mathrm{ITD}(\theta) = \frac{r}{v}\left(\sin(\theta) + \theta\right) \tag{1.17}$$

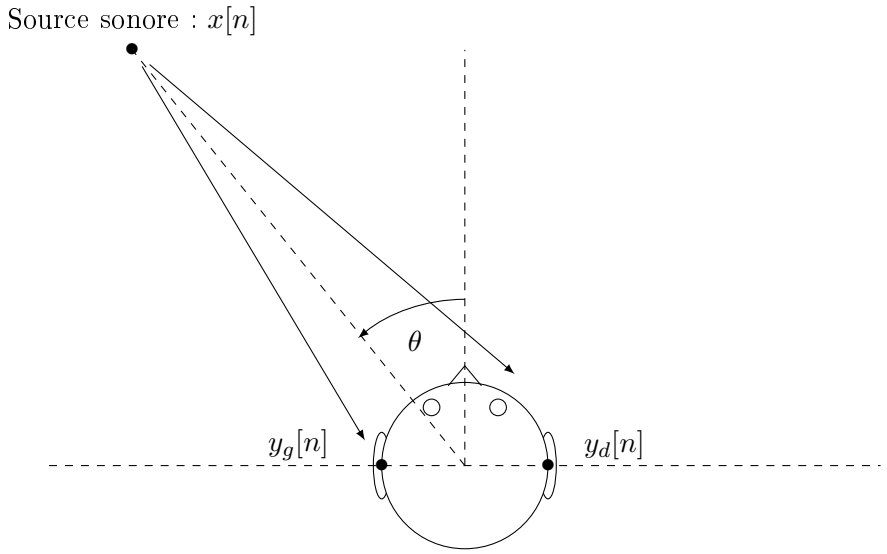


FIGURE 1.6 – Propagation du son entre son émission et son entrée dans les canaux auditifs

où r correspond au rayon de la tête (en m) et v correspond à la vitesse du son dans l'air (en m/s).

Manipulation 1.15. Créer un script nommé `show_ITD` permettant d'afficher l'ITD pour θ variant de -90^o à 90^o par pas de 1^o lorsque $v = 330$m/s et $r = 0.08$m. Joindre votre courbe à votre compte rendu. Est ce que l'ITD permet, sans ambiguïté, de discriminer les sources sonores situées devant et derrière nous ?

Le placement azimutal d'une source peut être simulé en introduisant un décalage temporel, égal à l'ITD, entre le signal parvenant à l'oreille gauche et celui parvenant à l'oreille droite. L'effet de spatialisation peut alors être implémenté par le schéma de la figure 1.7, où τ correspond à la valeur de l'ITD (en échantillon) et c correspond à un retard constant. L'utilisation d'un retard constant suffisamment élevé permet d'obtenir un filtre causal ($h[n] = 0$ si $n < 0$) lorsque τ est négatif.

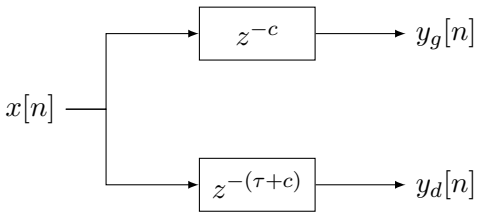


FIGURE 1.7 – Spatialisation sonore par synthèse de l'ITD. La valeur $\tau = F_e.\mathrm{ITD}(\theta)$ correspond à l'ITD (en échantillon) et c correspond à un retard constant.

Mathématiquement, les deux canaux s'expriment alors sous la forme :

$$y_g[n] = x[n - c] \tag{1.18}$$
$$y_d[n] = x[n - (F_e.\mathrm{ITD}(\theta) + c)] \tag{1.19}$$

Manipulation 1.16. Programmer la fonction `effet_spatialisation`. Cette fonction prendra comme paramètres d'entrée :

- un signal `x`,
- l'azimut θ (en deg),
- la fréquence d'échantillonnage F_e (en Hz).

La fonction devra renvoyer en sortie une matrice `y`, de taille $\mathbb{R}^{2\times N}$ contenant sur la première ligne le canal de gauche $y_g[n]$ et sur la seconde ligne le canal de droite $y_d[n]$. Les valeurs de $y_g[n]$ et $y_d[n]$ seront déterminées respectivement à partir des équations (1.18) et (1.19) pour $c = 10^{-3}.Fe$, $v = 330\text{m/s}$ et $r = 0.08\text{m}$.

Manipulation 1.17. Tester votre fonction en lançant le script `test_effet_spatialisation.m`.

L'ITD n'est pas le seul indice exploité par notre cerveau pour déterminer l'origine spatiale d'une source sonore. En particulier, cet indice ne permet pas de discriminer certaines positions spatiales notamment dans le plan vertical. Dans le paragraphe suivant, nous allons utiliser une technique permettant de simuler la provenance spatiale d'une source dans un espace tridimensionnel : la synthèse binaurale.

1.4.2 Spatialisation par synthèse binaurale (Facultatif)

Pour déterminer l'origine spatiale d'une source dans un espace tridimensionnel, notre cerveau exploite le filtrage introduit par notre morphologie. En effet entre leur émission et leur arrivée dans les canaux auditifs, les ondes acoustiques subissent des modifications causées par les réflexions du signal avec notre corps : épaule, tête, structure externe de l'oreille, etc. Ces modifications dépendent de la position spatiale de la source sonore. Mathématiquement, la modification du signal causée par notre morphologie peut être modélisée par un filtre linéaire invariant en temps. Dans la littérature, la réponse impulsionnelle de ce filtre est nommée HRIR (Head Related Impulse Response).

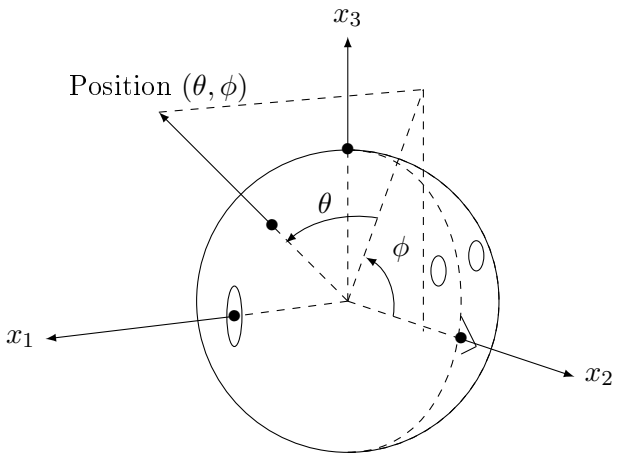


FIGURE 1.8 – Positionnement tridimensionnel des sources

Notons $h_g^{(\theta,\phi)}[n]$ et $h_d^{(\theta,\phi)}[n]$ les HRIR de l'oreille gauche et droite relatives à la position (θ, ϕ) (voir la figure 1.8). En pratique, l'acquisition des réponses impulsionnelles peut être très long car ces réponses doivent être mesurées pour un grand nombre de positions spatiales (θ, ϕ). Plusieurs campagnes de mesures ont été réalisées par des laboratoires de recherche. Dans cette

section, nous utiliserons les HRIR mesurées par le CIPIC [ADTA01]. La campagne du CIPIC comporte les HRIRs de 45 individus. Pour chaque individu, les HRIR des oreilles gauche et droite ont été mesurées à une fréquence d'échantillonnage $F_e = 44100$Hz pour 1250 position spatiales. Le répertoire `standard_hrir_database` contient les réponses impulsionnelles du CIPIC de 9 individus. Pour l'individu `x`, les réponses impulsionnelles sont stockées dans un fichier Matlab nommé `hrir_final` situé dans le sous-répertoire `Subject_00x`.

Lorsque le répertoire courant est `effet_spatialisation`, le chargement du fichier `hrir_final` relatif au premier individu est réalisé en utilisant l'instruction suivante :

```
>> load('standard_hrir_database\Subject_001\hrir_final.mat');
```

Après chargement, les réponses impulsionnelles sont disponibles dans le workspace de Matlab sous le nom `hrir_l` (gauche) et `hrir_r` (droite). Ces deux variables sont des matrices tridimensionnelles de taille $25 \times 50 \times 200$. La première dimension correspond à l'azimut θ, la seconde à élévation ϕ et la troisième au numéro échantillon k ($k = 1, ..., 200$). Pour les HRIR du CIPIC, les 25 azimuts et les 50 élévations sont spécifiées par les vecteurs suivants (en deg) :

```
>> liste_azimut=[-80 -65 -55 -45:5:45 55 65 80];
>> liste_elevation=-45+5.625*[0:49];
```

Par exemple, la syntaxe `hrir_temp(1,:)=hrir_l(2,9,:)` permet d'extraire la réponse impulsionnelle $h_g^{(\theta,\phi)}[n]$ pour $(\theta, \phi) = (-65^o, 0^o)$.

Manipulation 1.18. Créer un script nommée `show_IR` permettant de visualiser la réponse impulsionnelle gauche $h_g^{(\theta,\phi)}[k]$ du premier individu pour :
- une source située devant lui c-a-d $(\theta, \phi) = (0^o, -45^o)$,
- une source située derrière lui c-a-d $(\theta, \phi) = (0^o, 225^o)$.

Joindre les 2 représentations temporelles à votre compte-rendu. Est ce que ces deux HRIRs sont identiques ?

La représentation des réponses impulsionnelles ne permet pas de bien mettre en évidence les différences entre les deux filtres. Pour mieux mettre en évidence ces différences, nous allons représenter le module de leur réponse fréquentielle.

Manipulation 1.19. Modifier le script `show_IR` pour permettre la visualisation de la réponses fréquentielle du filtre $h_g^{(\theta,\phi)}[k]$ pour les mêmes positions que précédemment c-a-d $(\theta, \phi) = (0^o, -45^o)$ et $(\theta, \phi) = (0^o, 225^o)$. Pour calculer et représenter les réponses fréquentielles, il est conseillé d'utiliser la fonction `fvtool` (voir la sous-section 1.2.1). Joindre le module des 2 représentations fréquentielles à votre compte-rendu. Comparer le module de ces deux représentations fréquentielles.

La placement tridimensionnel (ϕ, θ) d'une source sonore peut être simulé en filtrant la source avec deux filtres de réponses impulsionnelles $h_g^{(\theta,\phi)}[k]$ et $h_d^{(\theta,\phi)}[k]$ (voir figure 1.9). Mathématiquement, le filtrage s'obtient en convoluant la source sonore $x[n]$ avec les réponses impulsionnelles c-a-d

$$y_g[n] = \sum_{k=0}^{L-1} h_g^{(\theta,\phi)}[k] x[n-k] \quad (1.20)$$

$$y_d[n] = \sum_{k=0}^{L-1} h_d^{(\theta,\phi)}[k] x[n-k] \quad (1.21)$$

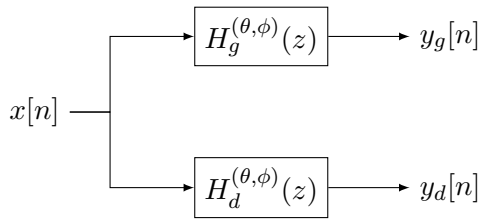


FIGURE 1.9 – Spatialisation sonore par synthèse binaurale.

Manipulation 1.20. Programmer la fonction `effet_binaural`. Cette fonction prendra comme paramètres d'entrée :

- un signal `x`,
- un numéro d'individu `k` (`k`$= 1, ..., 9$)
- une azimut θ appartenant à l'ensemble `liste_azimut` ($\theta = -80^o$ ... 80^o),
- une élévation ϕ appartenant à l'ensemble `liste_elevation` ($\phi = -45^o$ ... 230.625^o).

La fonction renverra en sortie une matrice `y` de taille $\mathbb{R}^{2\times M}$ contenant sur la première ligne le canal de gauche $y_g[n]$ et sur la seconde ligne le canal de droite $y_d[n]$. Les valeurs de $y_g[n]$ et $y_d[n]$ seront déterminées en implémentant les convolutions (1.20) et (1.21) via la fonction `filter`.

Manipulation 1.21. Tester votre fonction en lançant le script `test_effet_binaural.m`. Ce code simule un déplacement de 360^o sur le plan azimutal. Sauvegarder le résultat sonore. Modifier le numéro d'individu `k` afin d'obtenir la meilleure sensation de rotation. A votre avis, quelle est la limitation majeure de cette technique de spatialisation ?

1.5 Liste des programmes

```
1:  %% Programme de test de l'effet de delay
2:
3:  %parametres de l'effet
4:  tau=    ???;
5:  g=      ???;
6:
7:  %Chargement du fichier son
8:  [x,Fe]=wavread('piano_chord.wav');
9:
10: %application de l'effet
11: tic;
12: y=effet_delay(x,tau,g,Fe);
13: temps=toc;
14: fprintf('Temps de calcul: %f s\n',temps);
15:
16: %lecture du son original et avec effet
17: wavplay(x,Fe);
18: wavplay(y,Fe);
```

Programme 1.1 – Test effet de delay

```
1:  %% Programme de test de l'effet de reverberation
2:
3:  %parametres de l'effet
4:
5:  %Chargement du fichier son
6:  [x,Fe]=wavread('piano_chord.wav');
7:
8:  %application de l'effet
9:  tic;
10: y=effet_reverb(x);
11: temps=toc;
12: fprintf('Temps de calcul: %f s\n',temps);
13:
14: %lecture du son original et avec effet
15: wavplay(x,Fe);
16: wavplay(y,Fe);
```

Programme 1.2 – Test effet de reverberation

```
%% Test effet de spatialisation

Fe=44100;

%application de l'effet
x=randn(1,0.25*Fe);
for azimut=[-180:10:180]
    y=effet_spatialisation(x,azimut,Fe);
    if size(y,1)==2
        wavplay(y',Fe);
     else
         wavplay(y,Fe);
    end
end
```

Programme 1.3 – Test effet de spatialisation

```
%% Test effet binaural

Fe=44100;
k=4;
liste_azimut=[-80 -65 -55 -45:5:45 55 65 80];
liste_elevation=-45+5.625*[0:49];
trajectoire=1;

%choix de la trajectoire
if trajectoire==1
   t_azimut=[liste_azimut fliplr(liste_azimut)];
   t_elevation=[zeros(1,25) 180*ones(1,25)];
else
   t_azimut=zeros(1,50);
   t_elevation=liste_elevation;
end

%application de l'effet
x=randn(1,0.25*Fe);
for indice=1:50
    y=effet_binaural(x,k,t_azimut(indice),t_elevation(indice));
    if size(y,1)==2
        wavplay(y',Fe);
     else
         wavplay(y,Fe);
    end
end
```

Programme 1.4 – Test effet binaural

Travaux Pratiques 2

Effets Linéaires variants dans le temps

Objectifs: Pour donner du mouvement au son, il est possible d'utiliser des effets dont les propriétés évoluent en fonction du temps. Par exemple, les musiciens électroniques ont souvent recours à des filtres passe-bas de fréquence de coupure variable. Dans ce cas de figure, les effets ne peuvent plus être modélisés par des filtres invariants en temps.
Dans ce TP, nous allons nous intéresser à l'implémentation des effets suivants :
– l'effet de vibrato,
– l'effet de Flanger,
– l'effet de chorus,
– les filtres paramétriques.

2.1 Avant propos

Dans ce chapitre, nous allons nous intéresser aux filtres linéaires variants dans le temps obtenus à partir d'une relation de récurrence similaire à celle du chapitre précédent (voir équation (1.4)). La variation des propriétés du filtre sera simplement obtenue en modifiant les coefficients de la relation de récurrence en fonction du temps.

En modifiant les coefficients en fonction du temps, le filtre est décrit par la relation suivante :

$$\sum_{k=0}^{N} a_k[n]y[n-k] = \sum_{m=0}^{M} b_m[n]x[n-m] \tag{2.1}$$

Sous Matlab, cette relation de récurrence ne peut pas être implémentée via la fonction `filter` car les coefficients $a_k[n]$ et $b_m[n]$ ne sont pas constants. L'implémentation d'un filtre variant dans le temps sera alors obtenue manuellement en calculant la relation de récurrence via une boucle définie `for`.

2.2 Effet de vibrato

Le vibrato est un effet sonore obtenu en modulant de manière périodique la hauteur d'un son. En chant, le vibrato est obtenu en modulant les propriétés des cordes vocales. L'effet de vibrato est également bien connu des guitaristes. En effet, la plupart des guitares possèdent une tige métallique appelée vibrato qui est rattachée au chevalet. Cette tige permet de moduler la tension des cordes et donc la hauteur du son.

L'effet de vibrato est l'un des premiers effets à avoir été reproduit artificiellement. Pour reproduire cet effet, la technique la plus classique consiste à moduler de manière périodique un retard τ. La vitesse de lecture devient alors variable ce qui introduit une modulation de hauteur.

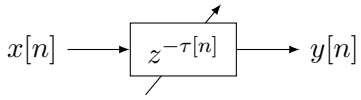


FIGURE 2.1 – Effet de vibrato. Le symbole $z^{-\tau[n]}$ désigne un retard de $\tau[n]$ échantillons. Ce retard varie de manière périodique en fonction du temps.

Mathématiquement, le signal de sortie s'exprime en fonction de l'entrée sous la forme

$$y[n] = x\left[n - \tau[n]\right] \tag{2.2}$$

où $\tau[n]$ désigne le retard en échantillon ($\tau[n] \in \mathbb{Z}$). La plupart du temps, le retard est généré par un oscillateur basse fréquence (LFO). Nous utiliserons ici un LFO sinusoïdal :

$$\tau[n] = \frac{\Delta.F_e}{2}\left(m[n] + 1\right) \tag{2.3}$$

où :

- $m[n]$ est une sinusoïde de fréquence f_0 Hz possédant une amplitude crête unitaire c-a-d $-1 \leq m[n] \leq 1$.
- Δ est un paramètre permettant de contrôler l'importance du vibrato. En particulier, il est possible de démontrer facilement que $0 \leq \tau[n] \leq \Delta.Fe$ ce qui correspond à un retard compris entre 0 et Δ secondes.

En pratique, l'implémentation de l'effet de vibrato n'est pas triviale. En effet, comme le retard est généré par un LFO, il n'y a aucune garantie que $\tau[n]$ soit un entier. Dans ce cas, la valeur de $x\left[n - \tau[n]\right]$ est inconnue et $y[n]$ ne peut pas être calculée. Dans les sous-sections suivantes, nous présentons deux techniques permettant de résoudre ce problème.

2.2.1 Implémentation par arrondi

Pour déterminer $x\left[n - \tau[n]\right]$, la technique la plus simple consiste à approcher $\tau[n]$ par un entier. L'effet de vibrato peut alors être implémenté en calculant les échantillons suivants

$$y[n] = x\left[n - \tau_2[n]\right] \tag{2.4}$$

où $\tau_2[n] = \lfloor \tau[n] \rfloor \in \mathbb{Z}$ correspond au plus grand entier inférieur ou égal à $\tau[n]$. Sous Matlab, le calcul de $\lfloor \tau[n] \rfloor$ s'obtient en utilisant la fonction `floor(.)`.

Manipulation 2.1. Programmer la fonction `effet_vibrato.m`. Cette fonction prendra comme paramètres d'entrée :

- un signal `x`,
- la fréquence d'échantillonnage F_e (en Hz),
- la fréquence du LFO f_0 (en Hz),
- La profondeur du vibrato Δ (en s).

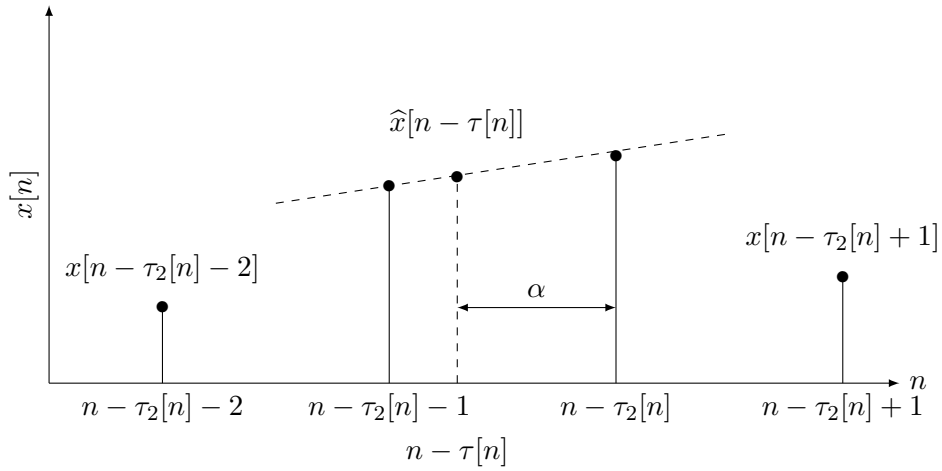


FIGURE 2.2 – Estimation de $x[n-\tau[n]]$ par interpolation linéaire.

La fonction renverra le signal `x` après application de l'effet de vibrato.

Manipulation 2.2. Modifier le script test_effet_vibrato.m pour obtenir un vibrato avec pour paramètres : $f_0 = 4$Hz et $\Delta = 0.005$s. Est ce que le signal après effet vous semble de bonne qualité ? Quelle est l'origine des parasites sonores ?

2.2.2 Implémentation par interpolation linéaire

Pour améliorer le rendu sonore, nous allons interpoler la valeur de $y[n] = x[n-\tau]$ à partir de ses deux échantillons voisins $x[n-\tau_2]$ et $x[n-(\tau_2+1)]$ (voir figure 2.2). En effectuant une interpolation linéaire, il est possible de démontrer via la théorème de Thales que la valeur interpolée est égale à :

$$y[n] = (1-\alpha).x\left[n-\tau_2[n]\right] + \alpha.x\left[n-(\tau_2[n]+1)\right] \tag{2.5}$$

où $\alpha = \tau[n] - \tau_2[n]$.

Manipulation 2.3. Modifier votre fonction `effet_vibrato.m` pour intégrer l'interpolation de l'équation (2.5). Tester votre fonction, lancer le script test_effet_vibrato. Est ce que le résultat sonore vous semble de meilleure qualité ?

Á noter qu'il existe d'autres techniques d'interpolation plus efficaces mais plus complexes. Le lecteur intéressé par cette problématique pourra se référer à l'article [LVKL96].

2.3 Effet de Flanger

Sous sa forme classique, l'effet de Flanger est décrit par la figure 2.3. Cet effet est produit en mixant un signal original avec une version retardée de celui-ci. Tout comme pour l'effet de vibrato, la valeur du retard est modulée en fonction du temps par une sinusoïde de fréquence f_0 Hz. Toutefois, l'effet de Flanger s'obtient en utilisant une fréquence f_0 plus faible, de l'ordre de quelques dixièmes d'Hertz. Musicalement, l'effet de Flanger est comparable au son produit par un avion passant au dessus de notre tête : le son parvenant à nos oreilles est la somme du son

direct et de ses réflexions sur le sol. Cette sommation provoque des interférences constructives ou destructives qui accentuent ou diminuent certaines fréquences [RR07, Zol11].

Dans les parties suivantes, nous allons nous intéresser à l'implémentation de l'effet de Flanger sous sa forme classique et avec feedback.

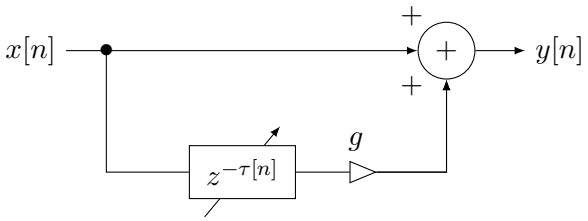


FIGURE 2.3 – Effet de Flanger (sans feedback)

2.3.1 Flanger sans feedback

L'effet de Flanger classique est décrit par la figure 2.3. Mathématiquement, la sortie s'exprime sous la forme :

$$y[n] = x[n] + g.x\left[n - \tau[n]\right] \tag{2.6}$$

L'implémentation de cet effet s'obtient facilement à partir du code de l'effet de vibrato. Pour obtenir un rendu sonore de meilleure qualité, il est conseillé d'implémenter le retard variable en utilisant une interpolation linéaire.

Manipulation 2.4. Programmer la fonction `effet_Flanger.m`. Cette fonction prendra comme paramètres d'entrée :

- un signal `x`,
- la fréquence d ?échantillonnage F_e (en Hz),
- la fréquence du LFO f_0 (en Hz),
- la profondeur du vibrato Δ (en s).
- la valeur du gain g.

La fonction devra renvoyer le signal `x` après application de l'effet de Flanger.

Manipulation 2.5. Modifier puis lancer le script test_effet_Flanger.m pour obtenir un effet de Flanger avec pour paramètres : $f_0 = 0.25$Hz, $\Delta = 0.003$ et $g = 0.8$.

2.3.2 Flanger avec feedback

Pour obtenir un effet de Flanger plus riche, il est possible de réinjecter la sortie retardée vers l'entrée. Dans cette configuration, l'effet de Flanger est décrit par la figure 2.4.

Mathématiquement, les différents signaux s'expriment alors sous la forme :

$$d[n] = u\left[n - \tau[n]\right] \tag{2.7}$$

$$u[n] = x[n] + g_2.d[n] \tag{2.8}$$

$$y[n] = x[n] + g.d[n] \tag{2.9}$$

Manipulation 2.6. Programmer la fonction `effet_Flanger_feedback.m`. Par rapport au programme précédent, cette nouvelle fonction prendra comme paramètre supplémentaire le gain g_2. La fonction devra renvoyer le signal `x` après application de l'effet de Flanger avec feedback.

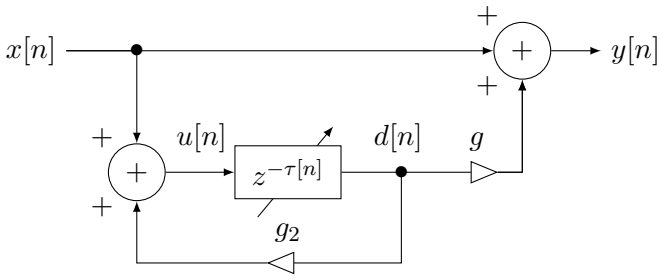


FIGURE 2.4 – Effet de Flanger avec feedback

Manipulation 2.7. Modifier le script test_effet_Flanger.m pour obtenir un effet de Flanger avec feedback ayant pour paramètres : $f_0 = 0.25$Hz, $\Delta = 0.003$, $g = 0.8$ et $g_2 = 0.7$. Que se passe t-il lorsque la valeur de g_2 approche l'unité ?

2.4 Effet de Chorus

Le Chorus est un effet obtenu en superposant un signal original avec plusieurs recopies de lui-même. Lorsque les recopies sont similaires, le signal produit est une simple amplification du signal original. Pour obtenir un résultat plus intéressant, il est possible de modifier légèrement les différentes recopies en ajoutant des retards, des altérations de la fréquence fondamentale etc [RR07]. Le signal obtenu est alors similaire à celui produit par plusieurs instruments de même type jouant la même note.

Pour obtenir un effet de Chorus, une technique possible consiste à moduler chaque recopie par un retard variant dans le temps. La figure 2.5 présente l'architecture d'un effet de Chorus composé de trois voies.

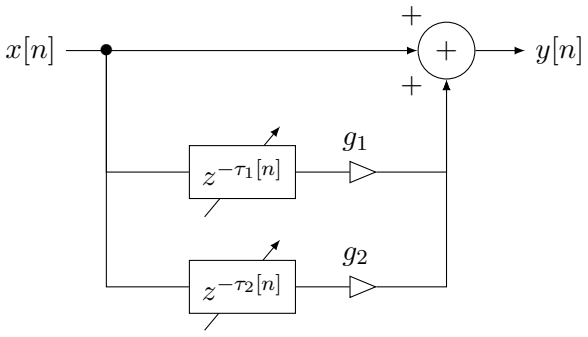


FIGURE 2.5 – Effet de Chorus

Mathématiquement, le signal après effet s'exprime sous la forme :

$$y[n] = x[n] + g_1 x\left[n - \tau_1[n]\right] + g_2 . x\left[n - \tau_2[n]\right] \tag{2.10}$$

Il est possible d'utiliser différents types de LFO pour la modulation des retards $\tau_1[n]$ et $\tau_2[n]$. Dans notre implémentation, nous utiliserons un LFO sinusoïdal avec une composante continue de μ secondes.

$$\tau_1[n] = F_e.\left(\mu m[n] + \Delta_1\right) \tag{2.11}$$

$$\tau_2[n] = F_e.\left(\mu m[n] + \Delta_2\right) \tag{2.12}$$

où $m[n]$ est une sinusoïde d'amplitude crête unitaire et de fréquence f_0 Hz. Les paramètres Δ_1 et Δ_2 permettent de contrôler la profondeur de modulation. Plus précisément, nous pouvons démontrer que les deux LFOs génèrent des retards compris entre $\mu-\Delta_k$ et $\mu+\Delta_k$ secondes ($k = 1, 2$). Concernant l'implémentation du retard, nous utiliserons la technique par interpolation linéaire.

Manipulation 2.8. Programmer la fonction `effet_chorus.m`. Cette fonction prendra comme paramètres d'entrée :

- un signal `x`,
- la fréquence échantillonnage F_e (en Hz),
- la fréquence du LFO f_0 (en Hz),
- le retard moyen μ (en sec),
- la profondeur du premier vibrato Δ_1 (en s),
- la profondeur du second vibrato Δ_2 (en s),
- la valeur du gain g_1,
- la valeur du gain g_2.

La fonction renverra le signal `x` après application de l'effet de Chorus.

Manipulation 2.9. Modifier puis lancer le script test_effet_chorus.m pour obtenir un effet de chorus ayant pour paramètres : $f_0 = 0.25$Hz, $\mu = 0.012$s, $\Delta_1 = 0.007$s, $\Delta_2 = 0.005$s et $g1 = g2 = 0.9$.

A noter, qu'il est possible d'améliorer la qualité de l'effet en utilisant des sinusoïdes déphasées pour les deux LFO. Une autre amélioration consiste à utiliser un bruit blanc filtré pour moduler les deux retards. Enfin en pratique, l'effet de Chorus prend tout son intérêt lorsqu'il est utilisé dans une configuration stéréo. En spatialisant les différentes recopies du signal, l'effet de Chorus produit alors une sensation d'immersion beaucoup plus importante.

2.5 Filtre paramétrique

Pour ajouter du mouvement au son, beaucoup de musiciens ont recours à des filtres variants dans le temps. Dans ce type d'application, le musicien n'agit pas directement sur les coefficients du filtre mais plutôt sur des paramètres de haut-niveau comme la fréquence de coupure, le facteur de qualité, etc. Par exemple en musique électronique, "l'ouverture" progressive d'un filtre passe-bas, c-a-d l'augmentation de sa fréquence de coupure, permet d'introduire en douceur les composantes hautes-fréquences. A inverse, l'ouverture d'une filtre passe-haut produit un effet plus agressif et inquiétant. Pour permettre aux musiciens d'agir sur ces paramètres haut niveaux, les ingénieurs ont proposé des architectures particulières reliant simplement et directement les paramètres modifiables par le musicien et les coefficients a_n et b_m du filtre. Ces filtres sont appelée filtres paramétriques. Nous utiliserons ici une architecture particulièrement populaire dans le domaine de l'audionumérique : le filtre à état variable.

L'architecture du filtre à état variable est décrit dans la figure 2.6. Ce filtre possède la particularité de renvoyer 3 sorties. Les sorties nommées $y_h[n]$, $y_b[n]$ et $y_l[n]$, correspondent respectivement à des comportements passe-haut, passe-bande et passe-bas. Ces sorties s'expriment mathématiquement sous la forme :

$$y_h[n] = x[n] - y_l[n-1] - f_2 y_b[n-1] \quad (2.13a)$$

$$y_b[n] = f_1 y_h[n] + y_b[n-1] \quad (2.13b)$$

$$y_l[n] = f_1 y_b[n] + y_l[n-1] \quad (2.13c)$$

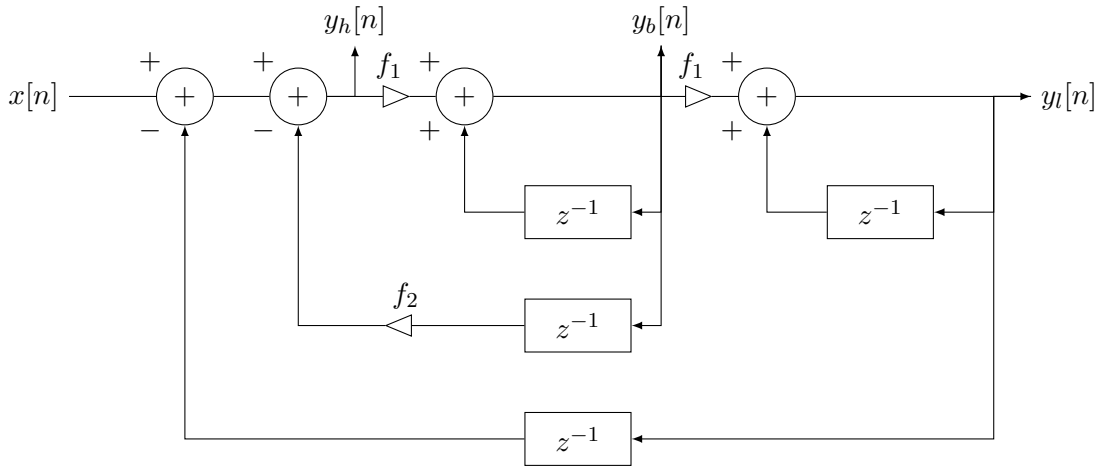


FIGURE 2.6 – Filtre à état variable. Le filtre possède trois sorties permettant d'obtenir un passe-haut, un passe-bande et un passe-bas.

Notons f_c la fréquence du coupure (ou centrale) des différents filtres et Q le facteur de qualité du filtre. Ces deux paramètres sont reliées aux paramètres f_1 et f_2 via les relations [Zol11] :

$$f_1 = 2\sin(\pi f_c/F_e) \quad (2.14)$$
$$f_2 = 1/Q \quad (2.15)$$

2.5.1 Analyse Théorique

Nous allons dans un premier temps analyser le comportement théorique du filtre. En utilisant l' équation (2.13), il est possible d'obtenir les équations de récurrence suivantes :

$$y_h[n] + a_1 y_h[n-1] + a_2 y_h[n-2] = x[n] - 2x[n-1] + x[n-2] \quad (2.16)$$
$$y_b[n] + a_1 y_b[n-1] + a_2 y_b[n-2] = f_1 x[n] - f_1 x[n-1] \quad (2.17)$$
$$y_l[n] + a_1 y_l[n-1] + a_2 y_l[n-2] = f_1^2 x[n] \quad (2.18)$$

où les paramètres a_1 et a_2 sont respectivement donnés par :

$$a_1 = f_1^2 + f_1 f_2 - 2 \quad (2.19)$$
$$a_2 = 1 - f_1 f_2 \quad (2.20)$$

Pour analyser ces 3 filtres, il est possible d'utiliser la fonction Matlab `fvtool` (voir la sous-section 1.2.1)

Manipulation 2.10. Créer un script `analyse_filtre.m` permettant d'afficher la réponse fréquentielle du filtre pour chacune des 3 sorties. La fréquence d'échantillonnage sera fixée à $F_e = 44100$Hz et les paramètres du filtres seront fixées à $f_c = 1$kHz et $Q = 2$. Joindre les 3 représentations fréquentielles à votre compte rendu.

Question 2.1. Déterminer graphiquement la pente, en dB par octave, du filtre passe-haut, passe-bande et passe-bas.

2.5.2 Application

Manipulation 2.11. Programmer la fonction `effet_filtre_variable.m`. Cette fonction prendra comme paramètres d'entrée :

- un signal `x`,
- la fréquence échantillonnage F_e (en Hz),
- un vecteur f_c contenant les valeurs de la fréquence de coupure (en Hz). Ce vecteur sera de même taille que le signal `x`.
- un vecteur Q contenant les valeurs du facteur de qualité. Ce vecteur sera de même taille que le signal `x`.

La fonction devra renvoyer les trois sorties `yl`,`yb` et `yh`.

Manipulation 2.12. Lancer le script test_effet_filtre_variable.m pour tester votre fonction. Par défaut, le script simule l'ouverture du filtre passe-bas : la fréquence de coupure évolue linéairement de 100Hz à $F_e/6$Hz. Il est possible de tester le fonctionnement de votre filtre passe-bande et passe-haut en remplaçant simplement `yl` par `yb` ou `yh` dans la dernière ligne du script.

Nous allons tester notre filtre lorsque la fréquence de coupure f_c est générée par un LFO sinusoïdal.

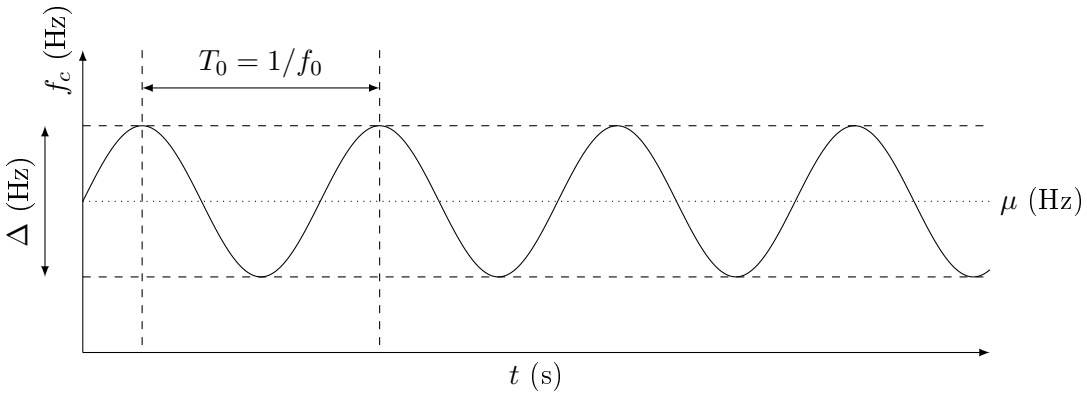


FIGURE 2.7 – LFO sinusoïdal utilisé pour contrôler la fréquence de coupure du filtre.

Manipulation 2.13. Modifier le script test_effet_filtre_variable.m pour que la fréquence de coupure f_c soit générée par le LFO de la figure 2.7. Les paramètres du LFO seront fixés à : $f_0 = 3$Hz, $\Delta = 1900$ Hz, $\mu = 2000$Hz.

Dans la plupart des productions musicales actuelles, la fréquence du LFO f_0 est synchronisée sur le tempo du morceau. Comme le fichier `daftpunk.wav` comporte un nombre entier de mesures, la synchronisation du LFO peut être implémentée facilement.

Manipulation 2.14. Modifier le script test_effet_filtre_variable.m pour que la fréquence du LFO soit égale à $f_0 = \frac{l.F_e}{N}$, où N correspond au nombre d'échantillons compris dans le fichier `daftpunk.wav` et où l est une puissance de 2. Lancer le script pour $l = 8, 16, 32, 64$.

2.6 Liste des programmes

```
1:  %% Programme de test de l'effet de vibrato
2:
3:  %parametres de l'effet
4:  f0=     ???;
5:  delta=  ???;
6:
7:  %Chargement du fichier son
8:  [x,Fe]=wavread('choir.wav');
9:
10: %application de l'effet
11: tic;
12: y=effet_vibrato(x,Fe,f0,delta);
13: temps=toc;
14: fprintf('Temps de calcul: %f s\n',temps);
15:
16: %lecture du son original et avec effet
17: wavplay(x,Fe);
18: wavplay(y,Fe);
```

Programme 2.1 – Test sonore de l'effet de vibrato

```
1:  %% Test de l'effet de flanger
2:
3:  %parametres de l'effet
4:  f0=     ???;
5:  delta=  ???;
6:  g=      ???;
7:
8:  %Chargement du fichier son
9:  [x,Fe]=wavread('daftpunk.wav');
10:
11: %application de l'effet
12: tic;
13: y=effet_flanger(x,Fe,f0,delta,g);
14: temps=toc;
15: fprintf('Temps de calcul: %f s\n',temps);
16:
17: %lecture du son original et avec effet
18: wavplay(x,Fe);
19: wavplay(y,Fe);
```

Programme 2.2 – Test sonore de l'effet de Flanger

```
%% Programme de test de l'effet de chorus

%parametres de l'effet
f0=     ??? ;
mu=     ??? ;
delta1= ??? ;
delta2= ??? ;
g1=     ??? ;
g2=     ??? ;

%Chargement du fichier son
[x,Fe]=wavread('guitare.wav');
x=x/3;
%application de l'effet
tic;
y=effet_chorus(x,Fe,f0,mu,delta1,delta2,g1,g2);
temps=toc;
fprintf('Temps de calcul: %f s\n',temps);

%lecture du son original et avec effet
wavplay(x,Fe);
wavplay(y,Fe);
```

Programme 2.3 – Test sonore de l'effet de chorus

```
%% Programme de test du filtre à état variable.

%Chargement du fichier son
[x,Fe]=wavread('daftpunk.wav');

%variation de la fréquence de coupure de 100hz à Fe/2;
fc=linspace(100,Fe/6,length(x));
Q=2*ones(1,length(x));

%application de l'effet
tic;
[yh,yb,yl]=effet_filtre_variable(x,Fe,fc,Q);
temps=toc;
fprintf('Temps de calcul: %f s\n',temps);

%lecture du son original et avec effet
wavplay(x,Fe);
wavplay(yl,Fe);
```

Programme 2.4 – Test sonore du filtre à état variable

Travaux Pratiques 3

Synthèse sonore musicale

Objectifs: La synthèse sonore est un ensemble de techniques permettant de générer des signaux audio. Ces signaux sont générés à partir de différents procédés analogiques ou numériques.
Dans ce TP, nous allons nous intéresser à l'implémentation des synthèses sonores numériques suivantes :
– la synthèse additive : simulation d'un orgue hammond,
– la synthèse soustractive : simulation de la TB303,
– la synthèse par modèles physiques.
Pour l'ensemble de ce TP, la fréquence d'échantillonnage sera fixée à $F_e = 44100$Hz.

3.1 Avant Propos

Nous allons tout d'abord créer plusieurs fonctions permettant de générer des enveloppes.

Manipulation 3.1. Programmez la fonction `enveloppe_exp` permettant de générer une enveloppe exponentielle (voir figure 3.1). Cette fonction prendra comme paramètres d'entrée :
- la durée totale (en s),
- la constante de temps τ.

Manipulation 3.2. Programmez la fonction `enveloppe_ADSR` permettant de générer une enveloppe ADSR (voir figure 3.2). Cette fonction prendra comme paramètres d'entrée :
- la durée de la note (en s),
- l'attaque (en s),
- le déclin (en s),
- le sustain,
- le relâchement (en s).

Indication : Il est recommandé d'utiliser la fonction `linspace` pour générer cette enveloppe.

3.2 Synthèse additive : Orgue Hammond (simplifié !)

3.2.1 Synthèse de l'instrument

L'architecture d'un orgue Hammond composé de 9 oscillateurs sinusoïdaux est donnée par la figure 3.3. Chaque oscillateur génère une sinusoïde dont la fréquence dépend de la fréquence

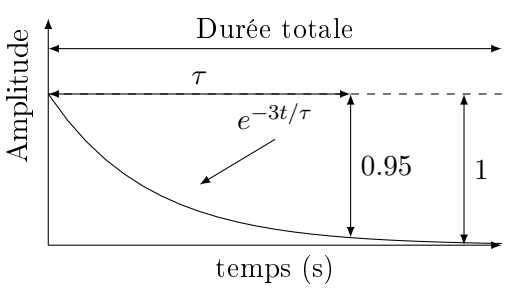


FIGURE 3.1 – Enveloppe exponentielle

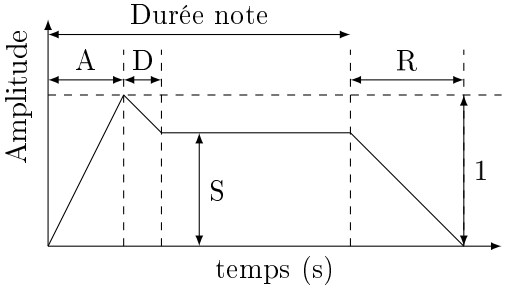


FIGURE 3.2 – Enveloppe ADSR

fondamentale f_1 (voir tableau 3.1). L'amplitude du signal est multiplié par une enveloppe de type ADSR. Dans l'ensemble de cette section, nous fixerons une attaque de 0.02s, un decay de 0.02s, un sustain égal à 0.75 et un relâchement de 0.03s.

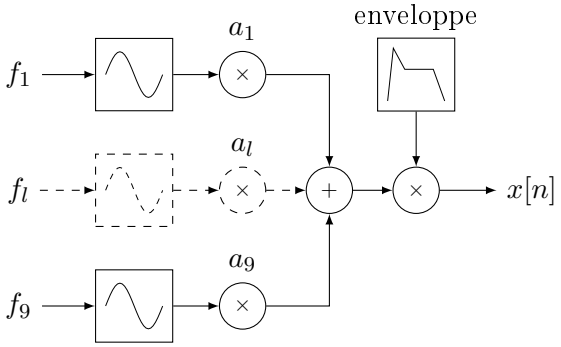


FIGURE 3.3 – Orgue Hammond

Oscillateur	1	2	3	4	5	6	7	8	9
Fréquence	$\frac{1}{2}f_1$	f_1	$\frac{3}{2}f_1$	$2f_1$	$3f_1$	$4f_1$	$5f_1$	$6f_1$	$8f_1$

TABLE 3.1 – Correspondance entre la fréquence fondamentale et la fréquence générée.

Pour notre implémentation, nous allons admettre que le musicien calibre la fréquence fondamentale, la durée d'une note ainsi que l'amplitude des harmoniques a_l ($l = 1, 2, \cdots, 9$).

Manipulation 3.3. Programmez la fonction `orgue_Hammond` permettant de générer une note. Cette fonction prendra comme paramètres d'entrée :

- la fréquence fondamentale f_1 d'une note (en Hz),
- la durée d'une note (en s),
- les amplitudes des harmoniques a_l. Ces amplitudes seront spécifiées en entrée sous la forme d'un vecteur ligne composé de 9 éléments.

Manipulation 3.4. Testez votre orgue en lançant la commande :

```
>> son=orgue_Hammond(220,2,[0 1 0.2 0.2 0.1 0.1 0.2 0 0]);
```

puis en écoutant le résultat.

3.2.2 Implémentation de la cabine Leslie

L'utilisation d'une enceinte rotative permet d'enrichir le son produit en introduisant de l'effet Doppler. Cet effet peut être approché en ajoutant conjointement un effet de vibrato et de tremolo (voir figure 3.4). Dans notre implémentation, ces effets seront obtenus en modélisant le retard $\tau[n]$ et l'amplitude $\alpha[n]$ par les fonctions suivantes :

$$\tau[n] = \tau_0 + \sigma_\tau \sin(2\pi f_m n/F_e) \tag{3.1}$$
$$\alpha[n] = \alpha_0 + \sigma_\alpha \sin(2\pi f_m n/F_e) \tag{3.2}$$

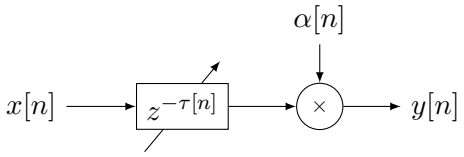


FIGURE 3.4 – Simulation de la cabine Leslie.

Manipulation 3.5. Programmez la fonction `cabine_leslie` permettant de simuler l'effet introduit par une enceinte rotative. Cette fonction prendra comme paramètres d'entrée :
- un signal `x`,
- la fréquence de rotation f_m (en Hz),
- un vecteur `[`τ_0 σ_τ`]` contenant les paramètres du vibrato,
- un vecteur `[`α_0 σ_α`]` contenant les paramètres du tremolo.

Indication : Pour l'implémentation de l'effet de vibrato, il est fortement conseillé de s'inspirer du travail réalisé dans le TP sur les effets linéaires variant dans le temps.

Manipulation 3.6. Créez un script nommé `mon_orgue` permettant de tester votre orgue et sa cabine leslie. Plus précisément, le script devra
- generer une note de fréquence fondamentale 110Hz pendant 5s
- simuler l'effet de la cabine Leslie avec les paramètres suivants : $f_m = 5$Hz, $\tau_0 = 10$, $\sigma_\tau = 10$, $\tau_1 = 0.9$ et $\sigma_\alpha = 0.1$.

3.2.3 Le riff secret

Nous allons implémenter un riff d'orgue Hammond composé de 8 notes. La partition du riff est donnée dans le tableau 3.2 et pourra être jouée en boucle.

Temps (s)	0	0.28	0.56	0.84	1.12	1.4	1.68	1.96
Note	Do$^\#$	Sol$^\#$	Fa$^\#$	Sol$^\#$	Mi	Fa$^\#$	Re$^\#$	Mi
f_1 (Hz)	277.18	415.30	369.99	415.30	329.63	369.99	311.13	329.63
durée (s)	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28

TABLE 3.2 – Le riff secret.

Manipulation 3.7. Créez un script nommé `riff_secret_1` permettant de jouer la partition précédente avec un orgue Hammond et sa cabine Leslie.
- Les notes seront générées en utilisant la fonction

```
>> son=orgue_Hammond(f1,d,[0 1 0 0.55 0.30 0.2 0 0.25 0]);
```

– L'effet introduit par la cabine Leslie sera ajouté une fois le riff généré en utilisant la fonction

```
>> riff_fx=cabine_leslie(riff,0.25,[10 10],[0.9 0.2]);
```

3.3 Synthèse soustractive : TB303

Dans cette section, nous allons tenter d'approcher (modestement) le son de la TB303.

3.3.1 Synthèse de l'instrument : Partie 1

Dans cette sous section, nous allons uniquement nous intéresser à la modélisation de l'instrument en désactivant l'enveloppe contrôlant la fréquence de coupure du filtre. L'instrument est modélisé par le diagramme de la figure 3.5.

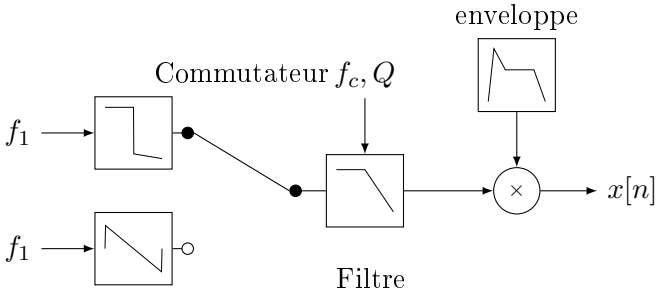


FIGURE 3.5 – TB303

Les signaux sources seront générés via les fonctions `square(.)` et `sawtooth()`. Le filtre passe-bas sera modélisé par un filtre de second ordre de pente -12dB/octave dont l'équation de récurrence est donnée par :

$$s[n] = b_0 e[n] + b_1 e[n-1] + b_2 e[n-2] - a_1 s[n-1] - a_2 s[n-2] \tag{3.3}$$

où les coefficients sont spécifiés dans le tableau 3.3.

Coef	b_0	b_1	b_2	a_1	a_2
Valeur	$\frac{K^2Q}{K^2Q+K+Q}$	$\frac{2K^2Q}{K^2Q+K+Q}$	$\frac{K^2}{K^2Q+K+Q}$	$\frac{2Q(K^2-1)}{K^2Q+K+Q}$	$\frac{K^2Q-K+Q}{K^2Q+K+Q}$

TABLE 3.3 – Expression des coefficients de l'équation de récurrence pour un filtre passe-bas de second ordre [Zol11]. La valeur de K est donné par $K = \tan(\pi f_c/F_e)$.

Nous modéliserons l'enveloppe du synthétiseur par une enveloppe exponentielle avec une constante de temps $\tau = 5$s.

Manipulation 3.8. Programmez la fonction `TB303` permettant de générer une note de TB303. Cette fonction prendra comme paramètres d'entrée :
- la fréquence fondamentale f_1 d'une note (en Hz),
- la durée d'une note (en s),
- la forme d'onde : `'sqr'` pour le carré et `'saw'` pour la dent de scie,

– la fréquence de coupure du filtre f_c (en Hz),
– la résonance Q.

Manipulation 3.9. Testez votre TB303 en lançant la commande :

```
>> son=TB303(110,2,'saw',1000,2);;
```

et en écoutant le résultat.

3.3.2 Synthèse de l'instrument : Partie 2

Dans cette sous-section, nous allons améliorer notre modèle de TB303 en ajoutant une enveloppe sur la fréquence de coupure du filtre (VCF). L'enveloppe contrôlant la fréquence de coupure du filtre est décrite par la figure 3.6 et sera paramétrée par la constante de temps τ. Cette enveloppe sera générée facilement en utilisant la fonction `enveloppe_exp`. L'opération de filtrage sera réalisée en implémentant manuellement l'équation de récurrence (3.3).

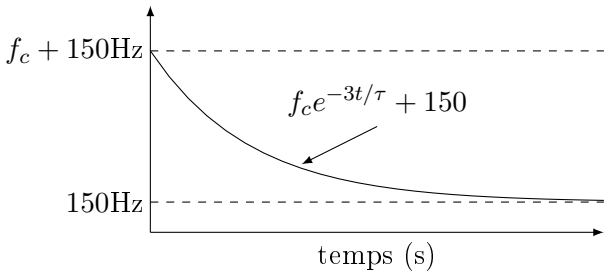


FIGURE 3.6 – Enveloppe exponentielle controlant la fréquence de coupure du filtre.

Manipulation 3.10. Programmez la fonction `TB303_v2` permettant de générer une note de TB303. Cette fonction prendra comme paramètres d'entrée :
– la fréquence fondamentale f_1 d'une note (en Hz),
– la durée d'une note (en s),
– la forme d'onde : `'sqr'` pour le carré et `'saw'` pour la dent de scie,
– la fréquence de coupure du filtre f_c (en Hz),
– la constante de temps τ de l'enveloppe du filtre (paramètre nommé decay sur la machine d'origine),
– la résonance Q.

Manipulation 3.11. Testez votre TB303 en lançant la commande :

```
>> son=TB303_v2(110,2,'saw',1000,0.2,2);
```

puis en écoutant le résultat. N'hésitez pas à inclure une enveloppe ADSR avec une attaque et un relâchement de quelques dizaine de ms et un sustain égal à 1 pour éviter des clics numériques en début et fin de sonorité.

3.3.3 Le riff secret

Nous allons implémenter un pattern de TB303 composé de 16 notes. La partition du riff est donnée par le tableau 3.4. Le riff pourra être recopié plusieurs fois afin de boucler le pattern.

Temps (s)	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2	2.2	2.4	2.6	2.8	3
Note	La	La	Ré	La	La	Do	La	La	Ré	La	La	Do	La	La	Ré	Do
f_1 (Hz)	55	55	73.42	55	55	65.41	55	55	73.42	55	55	65.41	55	55	65.41	73.42
durée (s)	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2

TABLE 3.4 – Pattern de TB303.

Manipulation 3.12. Créez un script nommé `riff_secret_2` permettant de jouer la partition précédente avec une TB303. Les notes seront générées en utilisant la fonction

```
>> son=TB303_v2(f0,d,'saw',10000,1.3,2);
```

Modifiez votre script pour simuler une ouverture de filtre allant de 1000 à 15000Hz.

3.4 Synthèse sonore par modèle physique : Algorithme KS

Dans cette section, nous allons implémenter l'algorithme de Karplus-Strong. Plus précisément nous allons nous intéresser à la simulation des sonorités produites par des cordes pincées. L'algorithme KS est décrit dans la figure 3.7. La valeur du retard N est directement liée à la fréquence fondamentale de la note par la relation $f_1 = F_e/N$. Remarquons que l'algorithme KS peut être décrit par la relation de récurrence suivante

$$s[n] = e[n-N] + g\alpha s[n-N] + g(1-\alpha)s[n-N-1], \tag{3.4}$$

son implémentation pourra donc être réalisée rapidement en utilisant la fonction `filter` de Matlab.

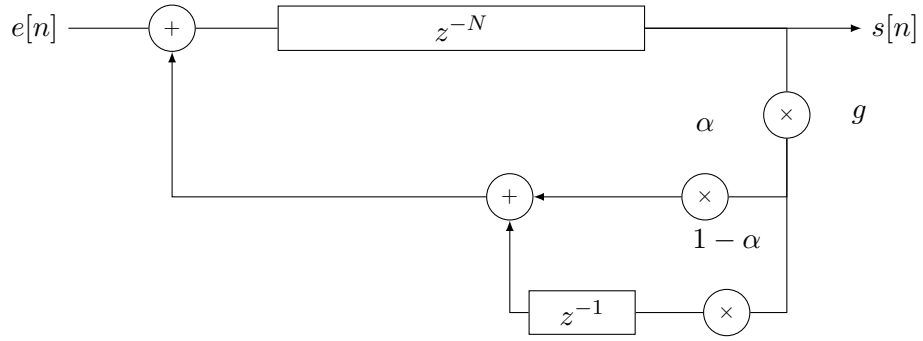


FIGURE 3.7 – Modèle avec pertes d'énergie [KS83]

3.4.1 Synthèse avec une entrée aléatoire

Dans cette sous section, nous allons exciter le guide d'onde par un signal d'entrée $e[n]$ composé de N échantillons aléatoire. Le signal $e[n]$ sera généré à partir d'un bruit blanc gaussien de moyenne nulle et de variance unitaire. Ce bruit sera implémenté en utilisant la fonction `randn` de Matlab.

Manipulation 3.13. Programmer la fonction `algorithme_KS` permettant de générer une note de fréquence fondamentale f_1. Cette fonction prendra comme paramètres d'entrée :

– la fréquence fondamentale f_1 d'une note (en Hz),

- la durée totale d'une note (en s),
- le coefficient d'amortissement g,
- le coefficient α.

Manipulation 3.14. Testez votre algorithme en lançant la commande :

```
>> son=algorithme_KS(55,2,0.99,0.5);
```

puis en écoutant le résultat.

3.4.2 Synthèse commutée

Dans cette sous section, nous allons exciter le guide d'onde par un signal échantillonné $e[n]$. Ce signal contient à la fois l'influence de l'excitation et du résonateur. Nous parlons alors de synthèse commutée puisque les éléments corde-résonateur sont commutés [SVD95].

Manipulation 3.15. Programmez la fonction `algorithme_KS_v3` permettant de générer une note de fréquence fondamentale f_1. Cette fonction prendra comme paramètres d'entrée :

- la fréquence fondamentale f_1 d'une note (en Hz),
- la durée totale d'une note (en s),
- le coefficient d'amortissement g,
- le coefficient α.
- le nom d'un fichier sonore au format wav comportant les échantillons $e[n]$.

Manipulation 3.16. Testez votre algorithme en lançant la commande :

```
>> son=algorithme_KS(55,2,0.99,0.5,'excitation_guitare.wav');
```

et en écoutant le résultat. Relancez l'expérience avec le fichier excitation_piano.wav.

Bibliographie

[ADTA01] V. R. Algazi, R.O. Duda, D.M. Thompson, and C Avendano. The CIPIC HRTF Database. Proceeding IEEE Workshop on Applications of Signal Processing to Audio and Electroacoustics, 2001.

[FM33] H Fletcher and W.A Munson. Loudness, its definition, measurement and calculation. Journal of the Acoustical Society of America, pages 82–108, 1933.

[ISO03] Equal-Loudness Level Contours, ISO 226, 2003.

[KM81] Steven M Kay and Stanley Lawrence Marple. Spectrum Analysis-A Modern Perspective. Proceeding of the IEEE, 69(11), 1981.

[KS83] Kevin Karplus and Alex Strong. Digital synthesis of plucked-string and drum timbres. Computer Music Journal, 7(2) :43–55, 1983.

[LVKL96] T.I. Laakso, V. Valimaki, M Karjalainen, and U.K. Laine. Splitting the Unit Delay - Tools for Fractional Delay Filter Design. IEEE Signal Processing Magazine, 13 :30–60, 1996.

[OS09] A.V. Oppenheim and R.W. Schafer. Discrete-time signal processing. 3rd edition, 2009.

[RR07] C Roads and J-d Reydellet. L'audionumérique : Musique et informatique. Dunod, Cambridge, 2007.

[SM97] Petre Stoica and Randolph L. Moses. Introduction to Spectral Analysis. Prentice-Hall, New Jersey, 1997.

[SVD95] Julius O Smith and Scott A Van Duyne. Commuted piano synthesis. In Proceedings of the 1995 International Computer Music Conference, Banff, pages 319–326, 1995.

[Zol11] U Zolzer. DAFX : Digital Audio Effects DAFX : Digital Audio Effects. Wiley, 2nd edition, 2011.