Etude de signaux sonores stationnaires en vue de leur reproduction

Jean-François Argentino

14 septembre 2002

Table des matières

Table des figures

Introduction et problématique

Pour clôturer mes études d'ingénieur, j'ai effectué un stage d'une durée de 5 mois au sein de la société GENESIS. Pendant cette période, un projet m'a été confié. Ce rapport présente donc le travail que j'ai fait, ainsi que les fondements théoriques sur lesquels il se base.

L'une des spécialités cette société est la simulation d'environnements sonores. Pour répondre à ce genre de problème, deux approches sont envisageables :

- L'enregistrement des signaux sonores générés par le système pour l'ensemble des états que l'on souhaite simuler. Puis la relecture des échantillons au moment de la simulation.
  Cette technique ne nécessite pas ou peu d'analyse, mais elle possède deux inconvénients majeurs. Le premier est que la quantité de mémoire nécessaire pour stocker les enregistrements est trop importante au vue de la qualité des sons restitués. De plus, un simulateur d'environnement sonore basé sur ce principe ne pourra évoluer qu'au prix du ré-enregistrement des signaux à lire.

- La synthèse en temps réel des signaux sonores par algorithme pour laquelle une analyse du système à simuler est nécessaire. Dans ce cas, ce n'est plus la quantité de mémoire qui est critique mais la puissance de calcul du générateur. En effet, l'architecture de ce dernier est conditionnée par la modélisation choisie.

Pour cette dernière méthode, il existe deux types d'algorithmes :

- Les algorithmes de modélisation physique où on tachera de décrire le système à simuler par des équations mécaniques rendant compte du son qu'il produit. Il faudra alors que le simulateur résolve en temps réel ces équations pour pouvoir synthétiser le son demandé. Ce genre de système d'équations devient très vite trop complexe pour des

systèmes à simuler un peu évolués, et la qualité du rendu sonore n'est pas toujours à la hauteur.

– A l'inverse, les algorithmes de modélisation du signal sonore ne décrivent pas la source du son, mais bien le son lui-même. Il faut donc trouver un modèle mathématique décrivant plus ou moins fidèlement le signal sonore, selon la qualité exigée.

Il est possible d'adopter en plus une approche dite perceptive. En effet, les sons à simuler sont de natures très diverses, il est donc impossible d'en sortir un modèle mathématique générique. En revanche, le récepteur est toujours identique puisqu'il s'agit de l'oreille humaine. La psychoacoustique (voir chapitre 2 page 9) est la science qui analyse la perception des sons par l'Homme, elle nous permet alors de sélectionner dans un signal ce qui sera vraiment perçu par l'auditeur. Des méthodes de compression audio avec perte (la norme MP3 par exemple) sont basées sur de tels principes.

Le modèle choisi par les ingénieurs de la société GENESIS est une décomposition du signal sonore en un nombre fini de sinusoïdes et de bandes de bruit à spectre contiguë. C'est un modèle perceptif, c'est à dire que ces paramètres seront déterminés selon des principes psychoacoustiques. Ainsi les sons d'origine, à spectres en général très complexes, seront simplifiés pour ne garder que les composantes pertinentes à l'oreille. Le nombre de composantes à garder dépendant essentiellement de la qualité exigée, de la puissance de calcul du générateur de son mais également de l'efficacité de la méthode de sélection.

Jusqu'à présent, les ingénieurs de GENESIS sélectionnaient "à l'oreille" les paramètres à utiliser lors de la synthèse, et cela au prix d'un temps précieux. Cependant, un traitement automatique des composantes pertinentes d'un son est envisageable en programmant certains des concepts de la psychoacoustique. J'ai donc développé au cours de mon stage un ensemble de fonctions pour le logiciel *MATLAB*©, permettant d'aider à l'analyse de sons stationnaires, ainsi qu'à la sélection de leurs paramètres pertinents selon le modèle retenu (sinus + bruit). L'utilisation de cette boîte à outils se veut la plus souple possible, en effet l'utilisateur a accès à l'ensemble des paramètres internes qu'il pourra alors adapter à ses besoins.

Après une présentation de la société GENESIS (page 6), je développerai succinctement les concepts de psychoacoustique sur lesquels s'appuient les fonctions écrites (page 9). Enfin, le dernier chapitre expliquera en détails le fonctionnement de chacune d'elles (page 18). De plus, un CD-ROM est fourni. Il contient la boîte à outils, un certain nombre de sons pour pouvoir la tester, le présent rapport ainsi que ses sources et le support de la présentation orale.

Chapitre 1

Présentation de l'entreprise

GENESIS est une société de haute technologie en acoustique basée dans l'Europôle de l'Arbois à Aix en Provence. Elle est créée en août 1999 par Patrick Boussard accompagné de 6 ingénieurs. Son activité se décompose en deux parties :

- la réalisation d'études et d'expertises liées à la simulation ou à la perception des environnements sonores,

- la conception, la production et la commercialisation de systèmes audionumériques génériques ou spécifiques aux besoins des clients.

FIG. 1.1 – L'equipe de GENESIS (avec deux stagiaires).

GENESIS repose sur une équipe de 10 salariés. Sa partie recherche, développement et production s'appuie sur des spécialistes dans 4 domaines

de compétence :

- Analyse et synthèse de son, transformation et traitement du signal audionumérique.

- Simulation d'environnement sonore en trois dimensions, spatialisation des sources virtuelles.

- Psychoacoustique - étude quantitative et qualitative de la perception et de la sensation auditive.

- Conception et réalisation de systèmes audionumériques, Informatique, électronique analogique et numérique, industrialisation, production.

Son savoir-faire est cultivé et approfondi par une collaboration permanente avec des centres de recherches, en particulier avec :

- L'IRCAM (Institut de Recherche et Coordination Acoustique/Musique) à Paris.

- Le Laboratoire de Mécanique et d'Acoustique (LMA) du CNRS à Marseille (Equipes Psychoacoustique et Informatique Musicale).

De plus, elle accueille dans son équipe trois étudiants effectuant leur thèse, ce qui renforce son potentiel en recherche et développement.

FIG. 1.2 – L'Harmo.

Les principales réalisations de la société GENESIS sont :

- l'environnement sonore du simulateur de vol pour les pilotes du

SuperPuma (client Eurocopter),

– une étude sur les bruits de circulation routière et ferroviaire (ministère des transports et de l'équipement, ministère de l'environnement),

– une machine dédiée , l'Harmo (fig. 1.2), permettant l'harmonisation [1] en temps réel de bandes sons. GENESIS a reçu le prix de l'innovation technologique Satisfecit pour ce produit lors de sa participation au SATIS (octobre 2001).

Enfin, GENESIS a su gagner la confiance d'importants groupes industriels pour des études ou des simulations d'environnements sonores ; on peut citer notamment P.S.A., Michelin, la S.N.C.F., Renault...

[1] changement de la durée d'un son en modifiant le moins possible son timbre, sa hauteur...

Chapitre 2

Notions de psychoacoustique

La psychoacoustique est la branche de l'acoustique qui étudie la perception des sons par l'homme. Il s'agit donc d'étudier les relations entre les stimuli sonores et les sensations, voir l'absence de sensations, qu'ils provoquent chez l'auditeur. Ainsi il sera possible de simplifier un son de manière à ne garder que ses composantes pertinentes, c'est à dire vraiment entendues par l'oreille. Une telle simplification peut servir pour de la compression audio (comme dans le standard MP3 par exemple), ou bien comme cela nous intéresse pour la synthèse de son lorsque le synthétiseur a une puissance de calcul limitée.

Il faut tout de même garder à l'esprit que la plupart des résultats de la psychoacoustique ont été déterminés expérimentalement. De plus, ces résultats sont valables pour un auditeur moyen, c'est à dire un sujet n'ayant pas de problème d'ouïe, mais qui n'a pas non plus une acuité auditive entraînée. En effet, les phénomènes psychoacoustiques sont en grande partie dus à la physiologie de l'oreille qui elle-même varie d'un sujet à l'autre. Ainsi, une adaptation des résultats en fonction du sujet cible peut être envisageable, par exemple l'utilisation des concepts psychoacoustiques ne sera pas la même suivant que la cible est constituée de personnes âgées ou de musiciens confirmés.

Nous allons donc voir au cours de ce chapitre les quelques principes théoriques sur lesquels s'appuie la boîte à outils développée.

2.1 Courbes d'isosonie

Courbes reliant la sensation de force du signal sonore sur l'auditeur, exprimée en sones, et l'intensité acoustique réelle, mesurée en décibels SPL [1].

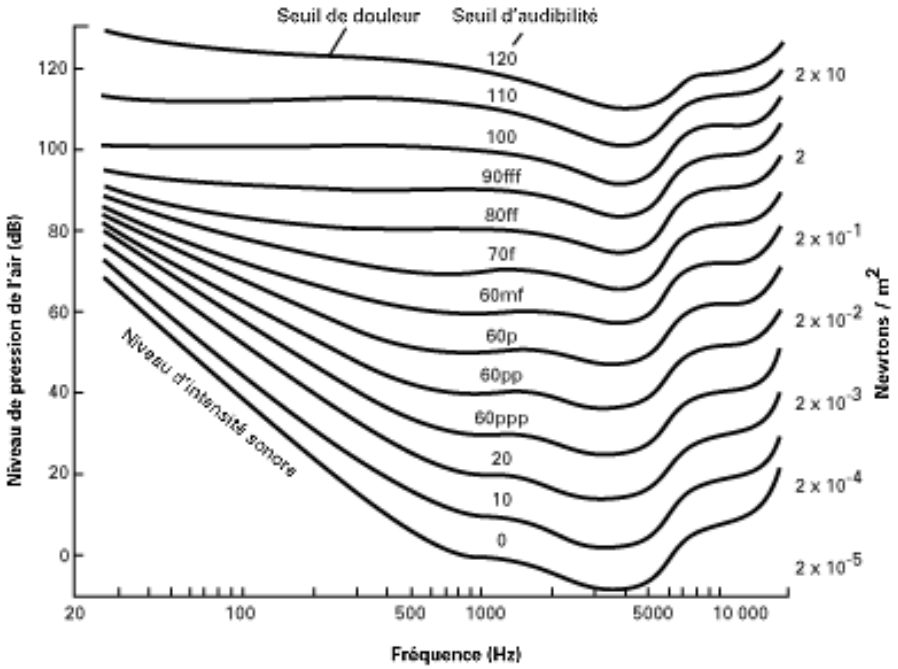

FIG. 2.1 – Courbes d'isosonie.

Ces courbes, déterminées expérimentalement au cours des années trente [2], permettent d'établir la sensation de perception égale d'une intensité acoustique donnée, et cela pour l'ensemble du spectre fréquentiel audible. Elles sont extrêmement utiles lorsqu'il s'agit de déterminer les régions spectrales significatives sur le plan perceptif lors d'opérations d'encodage et de compression de signaux audio-numériques.

[1] Le dB SPL (pour Sound Pressure Level) quantifie l'intensité acoustique d'un stimulus sonore. Il est définie par $I_{SPL} = 20 * \log_{10}\left(\frac{P}{P_0}\right)$ où P_0 représente est la référence de 20 μ Pa. Ainsi un niveau de 0 dB SPL correspond à un signal à la limite de l'audition à 1 kHz.

[2] H. Fletcher et W. Munson, "Relation between loudness and masking", *J. Acoust. Soc. Amer.*, vol. 9, pp. 1-10, 1937.

En particulier, la courbe du seuil absolue d'audition se place pour des signaux sonores à la limite de la sensation de perception. Une approximation possible [3] de ce seuil en fonction de la fréquence est :

$$S(f) = 3.64 * f^{-0.8} - 6.5 * e^{-0.6*(f-3.3)^2} + 10^{-3} * f^4$$

où f est en kHz et S en dB SPL.

[3] E. Terhardt, "Calculating virtual pitch", *Hearing res.*, vol. 1, pp. 155-182, 1979.

2.2 Bandes critiques

Pour une fréquence centrale donnée, la bande critique est la plus petite largeur spectrale pour laquelle une même zone de la membrane basilaire est excitée. En effet, lorsque une onde sonore arrive au pavillon de l'oreille, elle est transmise par la chaîne des osselets (non représentée figure 2.2) à l'oreille interne. Cette dernière se compose essentiellement de la cochlée (ou limaçon en raison de sa forme). Celle-ci est constituée de deux canaux, les rampes vestibulaire et tympanique, séparés par la la cloison cochléaire. Cette cloison comprend la membrane de Reissner, la membrane tectoriale, la membrane basilaire ainsi que l'organe de Cortie pris entre ces deux dernières. La vibration sonore engendre des ventres d'onde le long de la membrane basilaire à des positions spécifiques de sa fréquence. Or l'organe de Cortie qui repose sur la membrane basilaire, est constituée de cellules ciliées au contact desquelles prennent naissance les fibres du nerf auditif.

C'est l'espacement de ces cellules ciliées le long de la membrane basilaire qui engendre le phénomène des bandes critiques. En effet chaque récepteur nerveux se trouve accordé à une certaine fréquence due à sa position sur la membrane basilaire. Ainsi, l'oreille effectue une transformation fréquences/positions, et c'est bien une information de position qui est transmise au cerveau par le nerf auditif.

En conséquence d'une telle transformation, le fonctionnement de l'oreille interne s'apparente à celui d'un banc de filtres passe-bandes à fort taux de recouvrement [4], de plus la réponse en fréquence de chacun de ces filtres est asymétrique et leur bande-passante dépend de la fréquence centrale [5], la bande-critique correspondrait à la largeur de la bande-passante. En conséquence, si deux fréquences pures appartenant à la même bande critique se présentent au même instant à l'oreille, il est possible que l'auditeur n'en perçoive qu'une. Ainsi, une échelle des fréquences rendant compte de ces phénomènes a été inventée, le Bark, et elle sera utilisée avantageusement à la place de l'échelle de Hertz lorsqu'il est question de perception humaine de signaux sonores. Un bark est défini comme la largeur d'une bande critique, donc dans l'échelle de Bark les filtres sont de bande-passante constante.

Pour un auditeur moyen, la largeur (en Hz) de bande critique en fonction

[4] i.e. les bandes passantes des filtres se chevauchent.

[5] D. D. Greenwood, "Critical bandwith and the frequency coordinates of the basilar membrane", *J. Acoust. Soc. Amer.*, pp. 1344-1356, Oct. 1961.

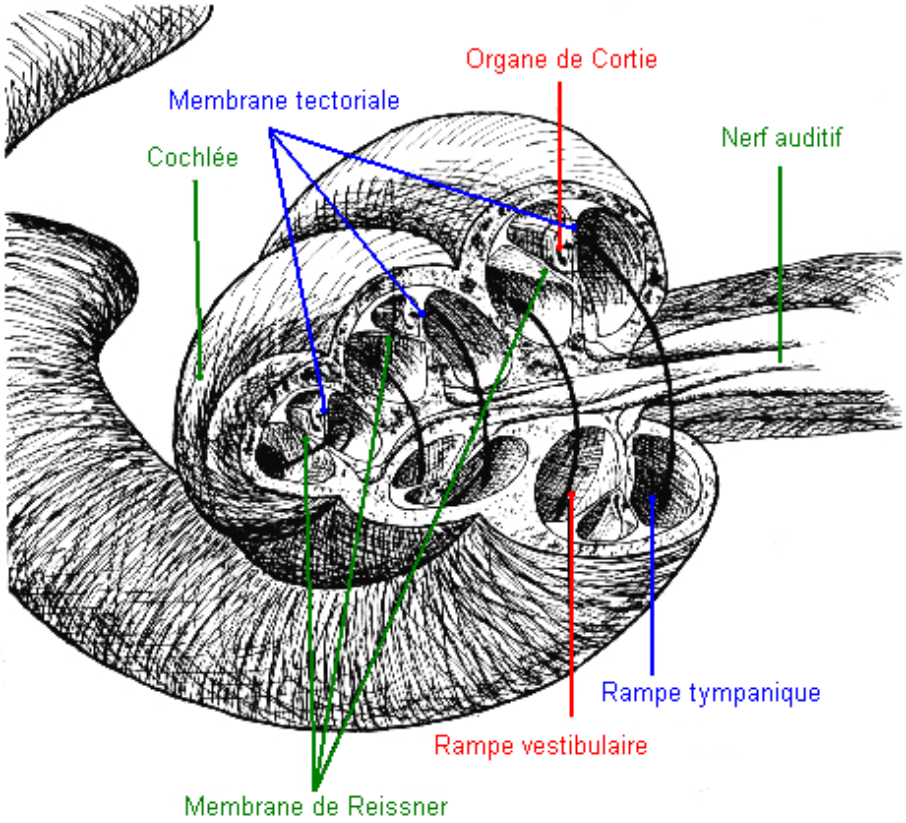

FIG. 2.2 – Coupe de la cochlée.

de la fréquence centrale peut être estimée par :

$$BP_{crit}(f) = 25 + 75 * \left[1 + 1.4 * \left(\frac{f}{1000}\right)^2\right]^{0.69}$$

de plus, la formule :

$$B(f) = 13 * \arctan\left(7.6 * 10^{-4} * f\right) + 3.5 * \arctan\left[\left(\frac{f}{7500}\right)^2\right]$$

est souvent utilisée pour convertir les Hertz en Bark [6].

[6]Ces deux formules sont tirées de : E. Zwicker et H. Fastl, "Psychoacoustics facts en models", 1990.

Bark	Hertz	Bark	Hertz	Bark	Hertz
1	101	9	1079	17	3822
2	204	10	1255	18	4554
3	308	11	1456	19	5411
4	417	12	1690	20	6414
5	530	13	1968	21	7617
6	651	14	2302	22	9166
7	781	15	2711	23	11414
8	922	16	3211	24	15405

FIG. 2.3 – Equivalence Bark/Hertz.

2.3 Masquage fréquentiel

On parle de masquage lorsque une composante sonore est rendue inaudible à cause de la présence d'une autre composante. Le masquage fréquentiel, ou masquage simultané, peut arriver lorsque deux stimuli sonores ou plus sont présentés au système auditif. Plus précisément, un signal sonore puissant peut engendrer une excitation de la membrane basilaire suffisamment forte pour empêcher l'audition d'un signal plus faible situé dans la même bande critique. Malgré le fait que les spectres de signaux audios réels peuvent générer une infinité de scenarii de masquage différents, il est commun de séparer ce phénomène en trois problèmes distincts et génériques, facilitant ainsi son exploitation dans des algorithmes audio-numériques.

2.3.1 Masquage d'un ton par un autre ton

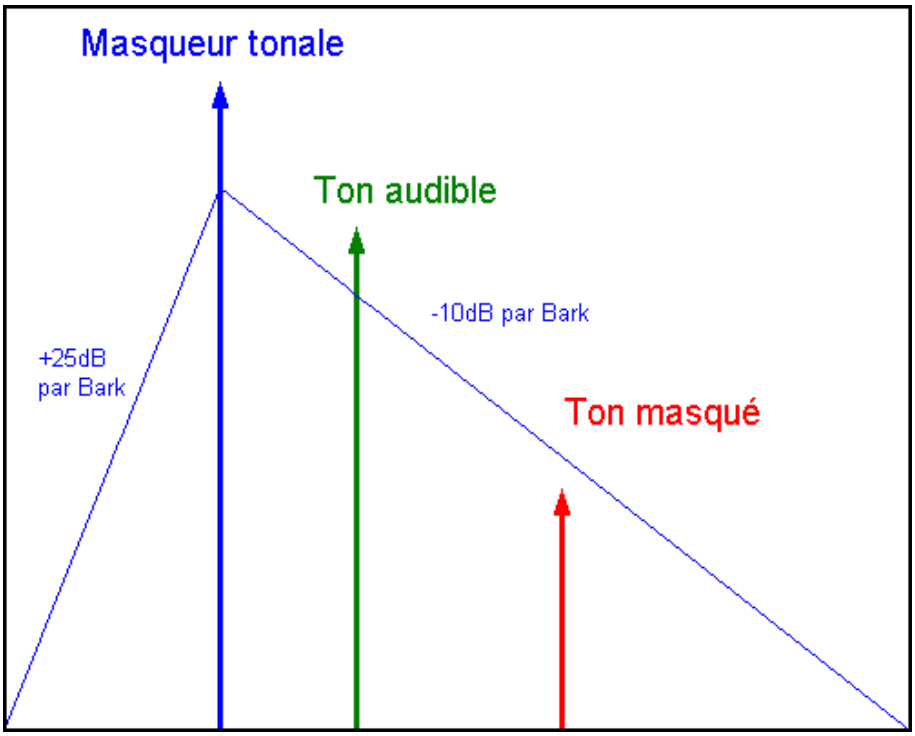

FIG. 2.4 – Ton masqué par un ton.

Dans le cas d'interaction de deux signaux purement sinusoïdaux, on considérera qu'un ton est inaudible si il est situé en dessous de la courbe de masquage (cf figure 2.4) induite par la fréquence ayant localement le plus d'énergie. Il est à noter que cette courbe ne s'étale pas uniquement sur la bande critique de la fréquence considérée.

2.3.2 Masquage d'un ton par un bruit

Il peut arriver qu'un ton soit masqué par le bruit de fond situé dans la même bande critique. Cela arrive lorsque l'énergie moyenne de la bande critique considérée est supérieure d'au moins 4 dB à l'énergie du signal sinusoïdal. Une étude de ce phénomène pourra être trouvée dans [R. Hellman, "Assymetry of maskink between noise and tone", Percept. Psychphys, vol. 11, pp. 241-246, 1972].

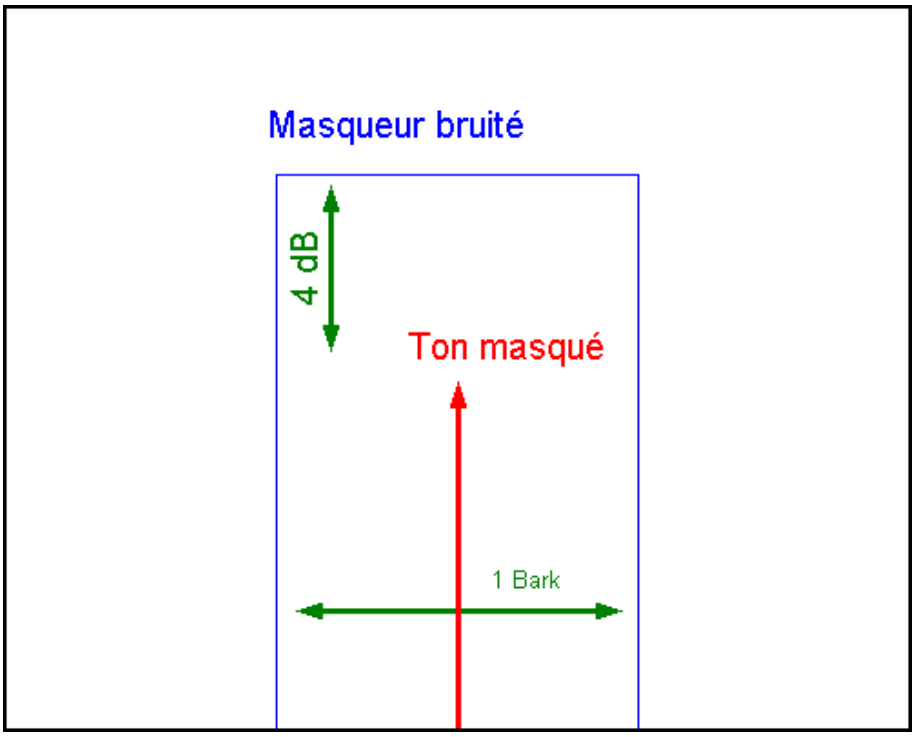

FIG. 2.5 – Ton masqué par une bande de bruit.

2.3.3 Masquage d'un bruit par un ton

La situation inverse est également possible. C'est à dire qu'un ton masquera le bruit de fond de sa bande critique si l'énergie du signal pure est supérieure d'au moins 24 dB à l'énergie moyenne de la bande critique considérée. Une étude complète de ce cas ainsi que du cas où un ton masque un autre ton pourra être trouvée dans [B. C. J. Moore, J. I. Alcántara et T. Dau, "Masking patterns for sinusoidal and narrow-band noise maskers", *J. Acoust. Soc. Amer.*, vol. 104, no. 2.1, pp 1023-1038, jan. 1998].

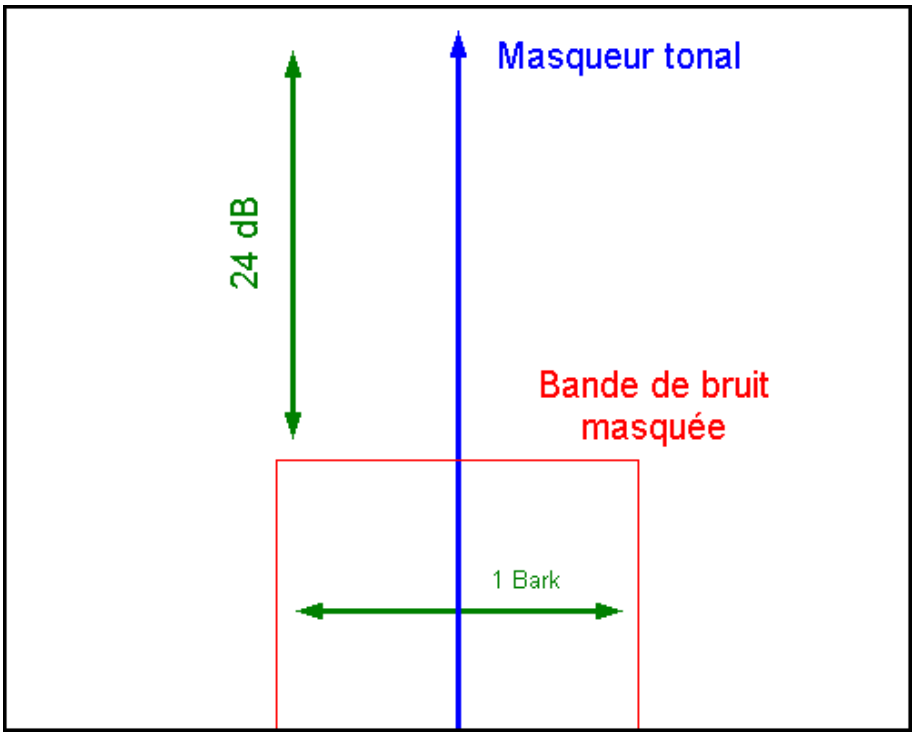

FIG. 2.6 – Bande de bruit masquée par un ton.

On remarque une dissymétrie entre ces deux derniers phénomènes. Elle rend compte du fait que l'oreille humaine distingue plus facilement un signal sinusoïdal noyé dans du bruit que l'inverse.

Chapitre 3

Manuel d'utilisation de la boîte à outils

Présentation

Ce manuel a pour but d'expliquer le fonctionnement et l'utilisation des fonctions MATLAB© contenues dans le répertoire "*SoundToolBox*".

A noter tout de même qu'une aide en ligne est accessible sous MATLAB©, il suffit de taper "`help` **nom_de_commande**" à la ligne de commande pour voir l'aide sur **nom_de_commande**, ou "`help soundtoolbox`" pour voir l'ensemble des fonctions de la boite à outils accompagné d'un descriptif rapide de chacune d'elles. De plus les codes sources ont été largement commentés.

Le but de cette boite à outils est d'aider à l'analyse de sons **STATIONNAIRES**, d'automatiser un certains nombres de procédures, et enfin d'extraire des composantes pertinentes au sens psychoacoustique en vu de leur synthèse en temps réel.

Les fonctions écrites ont été classées par catégories. Chaque chapitre correspondant à une catégorie particulière :

1. fonctions ayant uniquement rapport à des notions psychoacoustiques ; (page 22)
2. fonctions effectuant des traitements "classiques" en théorie du signal ; (page 26)
3. fonctions d'automatisation pour l'analyse de caractéristiques spectrales ; (page 30)

4. fonctions pour extraire les composantes pertinentes d'un son complexe en vu de sa simplification ; (page 37)
5. fonctions annexes de sauvegarde, de préparation de sons... (page 46)

Enfin, le dernier chapitre explique comment installer et utiliser cette boite à outils avec *MATLAB*© (page 55).

Conventions typographiques :

```
[ out1, out2 ] = cmd( in1 [, in2, in3 ] )
```

est la commande *MATLAB*© `cmd` d'arguments d'entrée `in1, in2` et `in3` dont les deux derniers sont optionnels ; d'arguments de sortie `out1` et `out2`. Si le comportement de la fonction est changé lorsqu'il n'y a pas d'argument de sortie, cela est précisé.

Descriptif rapide des fonctions

– Fonctions psychoacoustiques :

`bark2hz`..... conversion Bark → Hertz.

`critical_band`... Calcul des fréquences limites d'une fenêtre fréquentielle.

`hz2bark`..... Conversion Hertz → Bark.

`threshold`..... Seuil d'audition en fonction de la fréquence.

– Fonctions classiques de traitement de signaux :

`dirac_sequence`... Generation d'un peigne de dirac de largeurs fixes.

`get_lpsd`..... Calcul et affichage de la densité spectrale de puissance d'un son en dB.

`get_psd`..... Calcul et affichage de la densité spectrale de puissance d'un son.

`normalize`..... Normalisation d'un signal temporel en énergie.

– Fonctions d'analyse de signaux numériques :

`analyse_harm`..... Automatisation de `mean_harmonics`.

`get_all_tones`.... Détection des partiels d'un son.

`get_funds`..... Recherche des fréquences fondamentales parmi un tableau de fréquences.

`get_harmonics`.... Détection de suites harmoniques d'un son.

`get_harm_dir`..... Détection et sauvegarde de suites harmoniques des sons d'un repertoire.

`mean_harmonics`... Détection de suites harmoniques extrait de plusieurs fichiers wave.

– Fonctions permettant la simplification de sons complexes :

`analyse_pert`..... Automatisation de `mean_pertinents`.

`analyse_pert_harm`... Automatisation de `mean_pert_harm`.

`clone_wave`..... Copie d'un fichier wave grâce à `mean_pertinents`.

`decimation`..... Procédure de decimation des composantes tonales d'un son.

`get_noise_band`..... Evaluation des niveaux de bandes de bruit equivalent a une densité spectrale de puissance .

`get_pertinents`..... Extraction des composantes pertinentes d'un son.

`get_pert_dir`..... Processus d'extraction appliqué à tous les sons d'un répertoire.

`get_pert_harm`..... Extraction des séquences harmoniques pertinentes.

`mean_pertinents`..... Extraction et moyenne des composantes pertinentes de sons.

`mean_pert_harm`..... Extraction et moyenne des séquences harmoniques pertinentes.

`synthetize`..... Synthèse d'un son à partir de ses composantes pertinentes.

– Fonctions diverses :

`cell2file`..... Sauvegarde d'un tableau de cellules dans un fichier texte.

`cell_fusion`..... Fusion de deux tableaux de cellules représentant des séquences harmoniques.

`cell_sort`..... Tri par ordre croissant des fréquences fondamentales d'un tableau de cellules.

`cut_wavfile.....` Découpage d'un fichier wave en plusieurs fichiers.

`fusion.....` Fusion de deux tableaux d'une même séquence harmonique en un seul.

`get_all_dir.....` Récupération des noms des sous répertoires.

`get_all_files...` Récupération de tous les noms de fichier contenant une chaîne de caractères.

`insertion.....` Insertion d'un tableau de 25 niveaux de bruit dans un fichier.

`noises2file.....` Sauvegarde d'un tableau dans un fichier.

`now2str.....` Affichage de la date et de l'heure.

`prepare_dir.....` Preparation d'un répertoire de fichiers wave en vu de leur traitement.

`sec2hour.....` Conversion d'un nombre de secondes en heures, minutes, secondes, millisecondes.

`tones2file.....` Sauvegarde d'un tableau de fréquences et d'amplitudes dans un fichier.

3.1 Fonctions psycho-acoustiques

3.1.1 Bark2hz

La fonction $\texttt{F}_{\texttt{Hz}}$ `= bark2hz(` $\texttt{F}_{\texttt{bark}}$ `)` convertie $\texttt{F}_{\texttt{bark}}$ représentant une valeur ou un tableau de valeurs en Bark, en son (ses) correspondant(s) en Hertz. Pour ce faire, la valeur en Hertz est approchée grâce à la formule :

$$\texttt{F}_{\texttt{app}} = 1960 * \frac{0.53 + \texttt{F}_{\texttt{bark}}}{26.28 - \texttt{F}_{\texttt{bark}}}$$

puis la fonction *MATLAB*© `fsolve` (toolbox optim) est utilisée pour résoudre l'équation :

$$\texttt{hz2bark}(\texttt{F}_{\texttt{Hz}}) - \texttt{F}_{\texttt{bark}} = 0$$

la valeur $\texttt{F}_{\texttt{app}}$ servant de base de recherche de la solution pour `fsolve`. La figure 2.3 montre l'équivalent en Hertz des fréquences 1 à 24 Bark.

3.1.2 Critical_band

La fonction `[ F_down, F_up ] = critical_band( F, bw )` calcule les fréquences limites basse (F_{down}) et haute (F_{up}) en hertz d'une fenêtre de largeur `bw` bark centrée géométriquement sur la fréquence `F` en Hertz. Nous avons donc les relations suivantes :

$$F = \sqrt{F_{down} * F_{up}}$$

et

$$F_{up} - F_{down} = bw \text{ (en Hz)}$$

.

Pour accélérer le traitement, une simplification a été effectuée, si bien qu'au delà de 12 kHz, la fenêtre ne fait plus la largeur spécifiée (à 15 kHz 0.75 bark de large alors pour 1 bark demandé).

```
>> [F_down,F_up]=critical_band(440,1);...
        hz2bark(F_up)-hz2bark(F_down)

ans =

   1.0128

>> [F_down,F_up]=critical_band(12e3,1);...
        hz2bark(F_up)-hz2bark(F_down)

ans =

  0.9294
```

3.1.3 Hz2bark

La fonction `F_bark = hz2bark( F_Hz )` convertie la (les) fréquence(s) $\mathtt{F_{Hz}}$ en bark selon la formule :

$$\mathtt{F_{bark}} = 13 * \arctan\left(7.6 * 10^{-4}\right) * \mathtt{F_{Hz}} + 3.5 * \arctan\left(\frac{\mathtt{F_{Hz}}}{7500}\right)^2$$

La figure 3.1 montre la correspondance entre l'échelle des hertz et celle des bark.

```
>> plot( 20 :20000, hz2bark(20 :20000) ) ;
>> grid on ; xlabel('Hertz') ; ylabel('Bark') ;
```

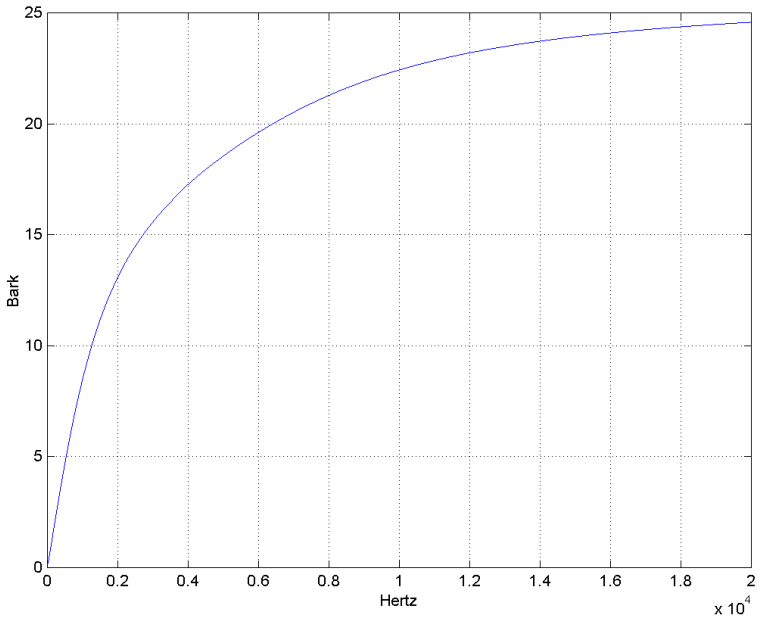

FIG. 3.1 – Equivalence Hertz/Bark.

3.1.4 Threshold

La fonction `pondération = threshold( F_Hz )` permet d'estimer la pondération due au seuil d'audition absolu grâce à la formule :

$$\texttt{seuil}_{\texttt{dB}} = 3.64 * \left(\frac{\texttt{F}_{\texttt{Hz}}}{1000}\right)^{-0.8} - 6.5 * e^{-0.6*\left(\frac{\texttt{F}_{\texttt{Hz}}}{1000}-3.3\right)^2} + 10^{-3} * \left(\frac{\texttt{F}_{\texttt{Hz}}}{1000}\right)^{4}$$

Nous avons donc :

$$\texttt{ponderation} = 10^{\frac{\texttt{seuil}_{\texttt{dB}}}{10}}$$

où on divise par 10 puisque cette pondération s'appliquera à des niveaux énergétiques.

A noter que si l'argument de sortie n'est pas précisé, la fonction affiche le seuil en dB SPL comme le montre la figure 3.2.

```
>> threshold(20 :20000)
```

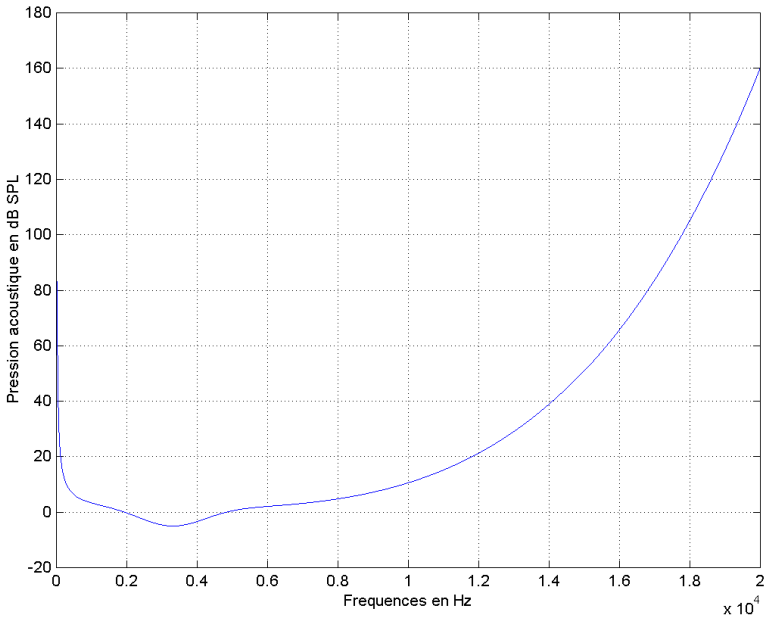

FIG. 3.2 – Pression acoustique équivalente au seuil d'audition.

3.2 Fonctions classiques

3.2.1 Dirac_sequence

`sequ = dirac_sequence( f0 [, nb, fs, len, wid ] )` génére une séquence de `nb` portes de largeur `wid` Hz, la séquence fait `len` échantillons fréquentiels, la fréquence d'échantillonnage étant de `fs`, la distance entre le centre de chacune des portes est de `f0` Hz. Les valeurs par défaut sont `nb = 0`, `fs = 44100`, `len = 32768` et `wid = 4`.

Si `nb` est égal à 0, le nombres de portes est maximal, et c'est le cas par défaut. Enfin si l'argument de sortie n'est pas précisé, la séquence est affichée (figure 3.3), cela peut éventuellement servir pour les phases de test.

```
>> dirac_sequence(2500)
```

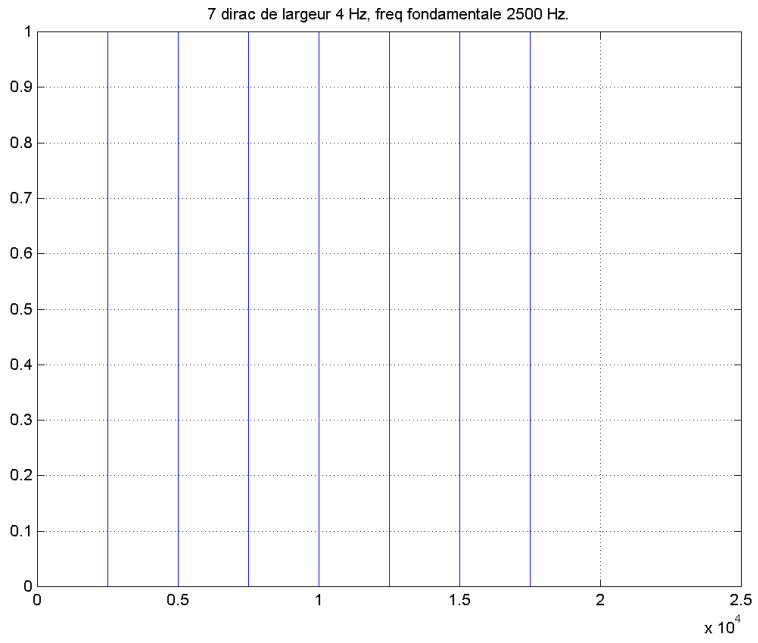

FIG. 3.3 – Séquence de dirac.

3.2.2 Get_psd et get_lpsd

`dsp = get_psd( sig )` et `dspl = get_lpsd( sig )` calculent respectivement la densité spectrale de puissance en linéaire et en dB à partir du tableau de valeurs `sig`. Ces fonctions peuvent également être utilisées sous la forme :

`dsp = get_psd( wavfile )` et `dspl = get_lpsd( wavfile )`,

où l'argument `wavfile` est une chaîne de caractères contenant le nom d'un fichier au format wave. L'argument de sortie `dsp` sera la densité spectrale de puissance regroupant les fréquences négatives et positives, alors que `dspl` ne contient que les fréquences positives.

Si l'argument de sortie n'est pas précisé, le résultat du calcul est affiché (figure 3.4), il est alors possible d'ajouter la fréquence d'échantillonnage `fs` pour avoir l'axe des abscisses en hertz et un dernier argument précisant le numéro de figure, ce qui peut être pratique lorsque l'on veut afficher plusieurs densités spectrales de puissance à la suite.

Enfin, précisons que le zero-padding est effectué à l'intérieur de la fonction mais pas le fenêtrage. Le nombre d'échantillons du signal sur lequel est appliqué la transformée de Fourier rapide est porté à $2^{17} = 131072$ échantillons, ou si le nombre d'origine est supérieur, à la puissance de 2 strictement supérieure. Ceci permet à la fois de gagner en précision fréquentielle (nombre d'échantillons) et en précision sur les niveaux (nombre de zéro à la suite du signal utile)

```
>> get_lpsd('9_vh_100_2200ft.wav')
```

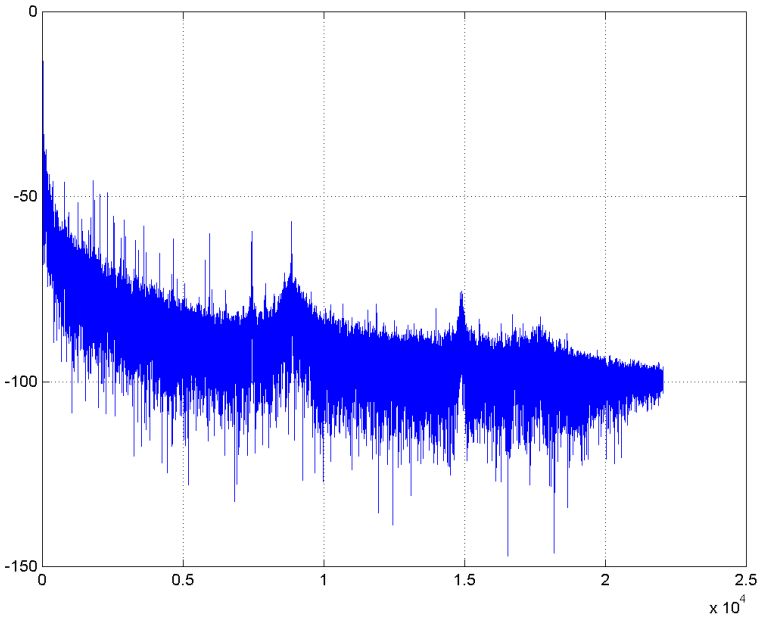

FIG. 3.4 – Affichage typique de la fonction `get_lpsd`.

3.2.3 Normalize

La fonction `Y = normalize( X )` normalise le signal `X` en puissance, nous avons donc :

$$\mathtt{Y} = \sqrt{\frac{\mathtt{N}}{\sum_{n=1}^{N} x_n^2}} * \mathtt{X}$$

et par construction :

$$\frac{1}{\mathtt{N}} \sum_{n=1}^{\mathtt{N}} y_n^2 = 1$$

où `N` est le nombre d'échantillons x_n du signal `X` et les y_n sont les échantillons du signal `Y`.

Un exemple d'utilisation sous *MATLAB*© : normalisation en puissance d'un signal sinusoïdal de fréquence 440 Hz, de 2048 échantillons :

```
>> X=sin(2*pi*440*(0:2047)/44100); sum( X.^2 )/2048
```

```
ans =

  0.5014

>> Y=normalize(X); sum( Y.^2 )/2048

ans =

  1.0000
```

3.3 Fonctions d'analyses

3.3.1 Get_all_tones

La fonction

```
[freq, peak, dsp] = get_all_tones( signal, fs [, thr, win ])
```

permet d'extraire les fréquences pures d'un son `signal` échantillonné à `fs` Hz. Pour ce faire, la densité spectrale de puissance du signal fenêtré par `win` (par défaut `win` est une fenêtre de Hanning) est calculée, puis le résultat est convolué avec l'image spectrale de la fenêtre, enfin une détection de pic est effectuée sur la sortie de la convolution. Est considéré comme un pic tout point dont les voisins immédiats ont une valeur strictement inférieure, et sa propre valeur est supérieure ou égale au produit de `thr` avec la valeur maximale de la densité spectrale de puissance entr 20 Hz et la demi fréquence d'échantillonnage (`thr` = 5e-4 par défaut). L'argument d'entré `signal` peut être remplacé par le nom d'un fichier au format wave. Dans ce cas, il est inutile de préciser la fréquence d'échantillonnage.

L'argument `freq` est un tableau des fréquences détectées en Hertz, `peak` est une densité spectrale de puissance ne contenant que des zéros et des pics là où il y a des fréquences, enfin `dsp` est la densité spectrale de puissance originale du signal. Si il n'y a pas d'argument de sortie, la densité spectrale de puissance originale est affichée en pointillés ainsi que les pics détectés. La figure 3.5 montre un zoom d'un tel graphique.

```
>> get_all_tones('9_vh_100_2200ft', 1e-3)
```

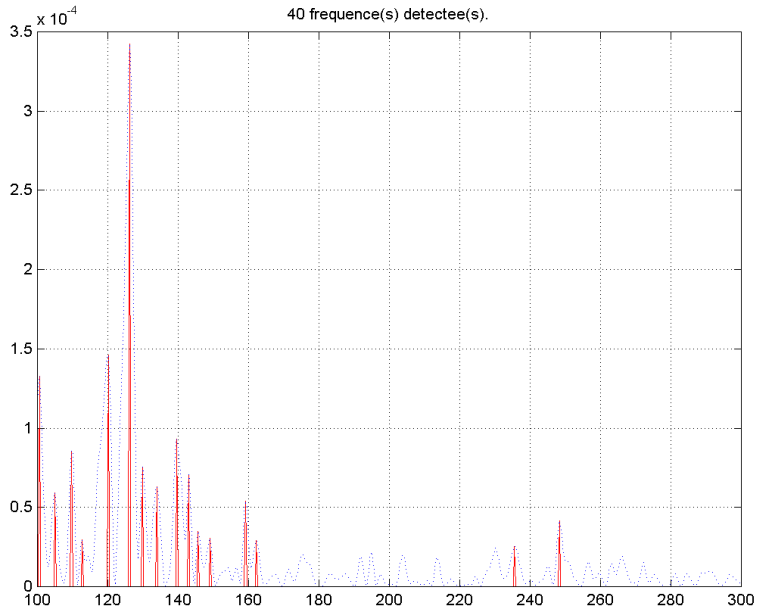

FIG. 3.5 – Affichage typique de la fonction `get_all_tones` après un zoom.

3.3.2 Get_funds

La fonction `array_out = get_funds( array_in [, wid ] )` extrait les fréquences fondamentales du tableau `array_in`, c'est à dire les fréquences qui ne sont pas un multiple d'une autre. Attention car `array_in` est bien un tableau de fréquences seulement, ce n'est pas un tableau de couples fréquences/amplitudes. Le tableau `array_out` contient donc toutes les fréquences de `array_in` moins les fréquences `F(i)` qui vérifient :

$$\exists \texttt{F0} \in \texttt{array\_in}, \exists \texttt{n} \in \mathbf{N}, \left| \texttt{F(i)} - \frac{\texttt{F0}}{\texttt{n}} \right| \leq \frac{\texttt{wid}}{2}$$

La valeur par défaut de `wid` est de 4 Hz, l'exemple suivant montre bien l'influence de ce paramètre.

```
>> Frequ = [ 440, 885, 550, 1100, 1652 ];
>> get_funds( Frequ )

ans =

    440   550   885

>> get_funds( Frequ, 10 )

ans =

    440   550
```

3.3.3 Get_harmonics et Get_harm_dir

La fonction `Harm = get_harmonics( sig, fs, F0 [, wid, thr ])` cherche la suite harmonique dans le son `sig` dont la fréquence fondamentale est `F0` ou `F0`/2. Pour cela les fréquences pures du son sont détectées grâce à `get_all_tones` avec le paramètre `thr` (voir page 30). Alors, parmi les fréquences détectées, celles qui vérifient :

$$\mathtt{F} \in \left[\mathtt{F0} - \frac{\mathtt{wid}}{2}, \mathtt{F0} + \frac{\mathtt{wid}}{2}\right]$$

sont sélectionnées, où `wid` vaut 4 Hz par défaut. Puis pour chacune de ces fréquences `F`, une séquence de 5 portes de largeur `wid` est générée grâce à `dirac_sequence`, cette séquence est alors multipliée point à point par la densité spectrale de puissance des fréquences pures du son (sortie `peak` de la fonction `get_all_tones` page 30). On regarde quel est le dernier élément non nul, sa position (Hertz) est divisée par son ordre, c'est alors une meilleure estimation de la fréquence fondamentale. Ce cheminement est réitéré avec la nouvelle valeur de fondamentale, en générant une séquence d'un nombre de portes égal à 5 plus l'ordre trouvé. La boucle s'arrête lorsque l'ordre trouvé à ce tour est le même que celui du tour précédent, ou bien lorsque la dernière harmonique trouvée est la plus grande possible.

`F0` peut être un tableau de plusieurs fréquences, auquel cas le traitement décrit précédemment est appliqué pour chacune d'elles. Mais attention au cas particulier où `F0` ne contient que 2 fréquences, on recherche alors toutes les séquences harmoniques dont la fondamentale est comprise entre `F0(1)` et `F0(2)`.Enfin, `signal` peut être remplacer par le nom d'un fichier au format wave, alors l'argument `fs` ne doit plus être précisé.

La sortie `Harm` est un tableau de cellules, chacune d'elles contenant une séquence harmonique, c'est à dire un tableau de couples fréquences/amplitudes, les fréquences étant toutes multiples de la première. Le tableau de cellules est classé par ordre croissant de fondamentale. Si `Harm` n'est pas précisé, la fonction sauvegarde les résultats dans un fichier texte portant le même nom que `signal` si ce dernier est un nom de fichier wave, ou bien une chaîne de caractère `'date,heure'` (sortie de now2str page 52), le tout précédé de `'harmonics_'`.

La fonction `get_harm_dir( dir [, F0, wid, thr ])` applique la fonction `get_harmonics` décrite ci-dessus à tous les fichiers wave contenus dans le répertoire `dir`.

L'exemple suivant montre l'influence du paramètre `wid` dans la longueur des séquences détectées, de plus on peut remarquer que dans ce cas la fréquence de 130 Hz n'est pas la fondamentale.

```
>> h = get_harmonics( '9_vh_100_2200ft', 130 );
>> h{1}

ans =

    65.2725  129.8721  194.8082
     0.0057    0.0174    0.0094

>> h = get_harmonics( '9_vh_100_2200ft', 130, 10 ) ;
>> h{1}

ans =

   62.5809  128.0216  193.2941  256.3797  379.8592
    0.0144    0.0272    0.0091    0.0079    0.0092
```

3.3.4 Mean_harmonics et analyse_harm

Voici certainement la fonction de cette famille qui doit le plus servir.

```
Harm = mean_harmonics( fichiers [, F0, wid, thr ])
```

recherche dans tous les fichiers wave contenus dans la cellule `fichiers` les suites harmoniques de fondamentales `F0` et/ou `F0/2`. `get_harmonics` s'applique sur chacun des fichiers avec les paramètres passés à `mean_harmonics`, puis les résultats sont rassemblés dans un seul tableau de cellules en fusionnant les cellules représentant les séquences harmoniques de même fondamentale (voir `fusion` page 50), ou en insérant à la bonne place les nouvelles séquences (voir `insertion` page 51).

Il est possible de spécifier une chaîne de caractères à la place de la cellule `fichiers`, alors le traitement se fera sur tous les fichiers wave du répertoire courant dont le nom contient la chaîne passée en paramètre.

L'utilisation de `mean_harmonics` est identique à celle de `get_harmonics` (hormis pour le premier paramètre bien-sûr). C'est à dire que l'on peut spécifier une fréquence minimale et maximale pour les fondamentales des sequences à rechercher, ou encore une liste de fréquences candidates. Le format de la sortie est identique, et si aucun argument de sortie n'est précisé, les résultats sont enregistrés dans un fichier texte dont le nom dépend du premier argument d'entrée. Si ce dernier est un tableau de noms de fichiers, alors le nom du fichier texte sera `'mean_harm_date,heure.txt'`, si à l'inverse le premier argument est la chaîne de caractère `'str'` le nom sera `'mean_harm_str.txt'`.

La fonction `analyse_harm( dir [, F0, wid, thr, len, nf ])` prépare l'ensemble des sons au format wave du répertoire `dir` grâce à la routine `prepare_dir` à laquelle on passe les paramètres `len` et `nf` (voir page 53). Puis `mean_harmonics` est appliquée sur chacun des sous-répertoires ainsi créés. A noter que si les sons avaient deja été préparés (lors d'un précédent appel, direct ou indirect, de `prepare_dir`), `analyse_harm` ne le refera pas.

Le résultat de l'exemple suivant est à comparer avec celui de la fonction `get_harmonics` page 34.

```
>> h = mean_harmonics( 'vh_100_2200ft', 130, 10 );
>> h{1}

ans =

  Columns 1 through 6

   66.6358  133.6402  199.2855  267.0829  331.4095  380.8685
    0.0151    0.0193    0.0092    0.0084    0.0083    0.0100

  Columns 7 through 8

  464.6461  776.2047
    0.0085    0.0107
```

3.4 Fonctions pour la simplification des sons

3.4.1 Clone_wave

La fonction `clone_wave( wavfile [, adjust, nb, wid, bw, tnmr ] )` permet de reproduire le fichier au format wave `wavfile` en un fichier wave portant le même nom avec le préfixe `SYNTH_`. Le fichier `wavfile` est découpé en plusieurs petits fichiers wave grâce à `cut_wavfile` (page 49), la fonction `mean_pertinents` est appelée en lui passant les fichiers créés, ainsi que les arguments `nb`, `wid`, `bw` et `tnmr` (voir page 43 pour plus de détails). Alors avec les caractéristiques ainsi extraites, un son de même longueur et de même niveau que l'original peut être synthétisé grâce à la fonction `synthetize` (page 45) à qui on passe `adjust`. Cela permet de vérifier que la modélisation est bien adaptée aux types de sons traités, et éventuellement d'affiner les paramètres du model.

Il est également possible de se servir de cette fonction sous la forme :

`clone_wave( wavcell [, nb, wid, bw, tnmr ] )`,

où `wavcell` est le tableau des noms de fichier wave à traiter avec `clone_wave`.

3.4.2 Decimation

La fonction

```
[ppsd, rpsd] = decimation( odsp, tpsd, FS [, nb, bw, TNMR ] )
```

permet de simplifier un son complexe stationnaire en ne gardant que `nb` composantes tonales (20 par défaut) de la densité spectrale de puissance des fréquences pures `tpsd` produite par `get_all_tones` (page 30). La densité spectrale de puissance d'origine `opsd` est également nécessaire pour deux raisons. Premièrement pour vérifier que chacune des raies spectrales n'est pas masquée par le bruit qui l'entoure. En second lieu, le modèle de synthèse choisi remplace tout ce qui n'est pas retenu de la densité spectrale de puissance originale en bande de bruit d'énergie moyenne équivalente.

Voici comment est effectuée la sélection. Pour chacune des raies spectrales, une fenêtre de largeur `bw` bark (1 par défaut) centrée géometriquement sur la raie est définie grâce à la fonction `critical_band` (page 23), on vérifie alors que la raie concernée dépasse de plus de `TNMR` dB (5 dB par défaut) le niveau d'énergie moyen contenu dans cette bande. Si ce n'est pas le cas, le ton est masqué par le bruit environnant, sinon il faut vérifier que la raie considérée est la plus grande dans cette même bande, alors seulement elle est considérée comme étant une composante tonale pertinente. Cette méthode est une approximation du masquage tonale, en effet le véritable phénomène de masquage fait intervenir des pentes autour de la fréquence considérée, mais cela suffit pour notre besoin. Il suffit ensuite de garder les `nb` fréquences dont les amplitudes (pondérées par le seuil d'audition absolue) sont les plus importantes.

Sont récupérés `ppsd` une densité spectrale de puissance constituée uniquement des `nb` tons jugés les plus pertinents, et `rpsd` qui contient la densité spectrale de puissance originale à laquelle sont retirées les susdites fréquences pertinentes. Si `nb` est nul, on garde toutes les fréquences pures qui ne sont pas masquées par le bruit et qui sont les plus importantes de leur bande critique.

```
>> [f, tdsp, dsp] = get_all_tones('9_vh_100_2200ft');
>> fs=44100;
>> [ddsp, rdsp] = decimation(dsp,tdsp,44100);
>> incr = fs/(2*length(dsp)); F=0:incr:fs/2-incr;
>> plot(F,tdsp,F,ddsp)
```

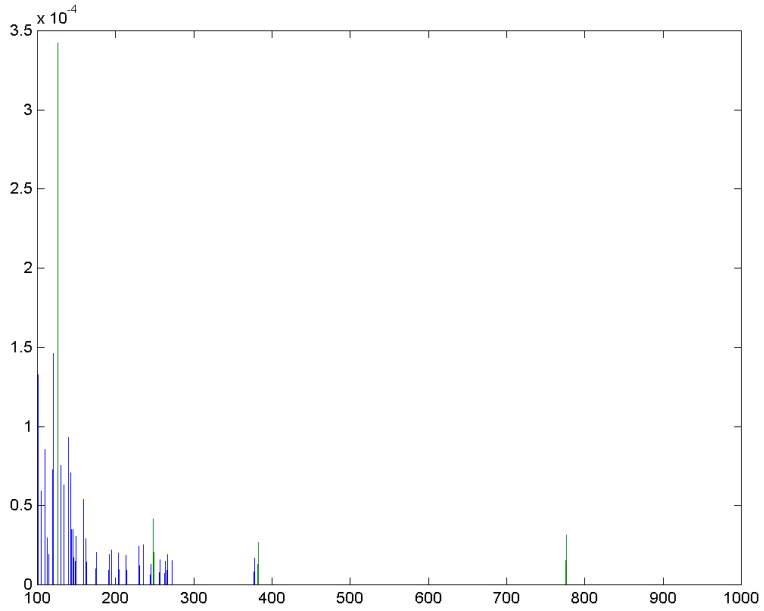

FIG. 3.6 – Comparaison entre les tons jugés pertinents (vert) et l'ensemble des tons présents (bleu).

3.4.3 Get_noise_band

La fonction `noises = get_noise_band( PSD, fs , len [, nbw ])` sépare la densité spectrale de puissance `PSD` en bandes contigües de `nbw` bark de large (1 bark par défaut), et sur chacune d'elles moyenne l'énergie. Ainsi `noises` est un tableau de $\frac{25}{\texttt{nbw}}$ niveaux de bruit. `len` est le nombre d'échantillons du signal original, avant zero-padding, cette longueur est nécessaire pour calculer le niveau d'énergie. Typiquement `PSD` est le deuxième argument de sortie de `decimation` (page 38). Le tableau ne regroupe que les niveaux de bandes de bruit, soit la racine carrée de la variance de la bande considérée, il suffira ainsi de multiplier simplement cette valeur par la densité spectrale de puissance dans la bande considérée. A noter que si `nbw` est égal à 0, la fonction est court-circuitée et `noises` a pour valeur `PSD`. Mais attention, ceci ne peut être utilisé directement avec `synthetize`.

3.4.4 Get_pertinents et get_pert_dir

La fonction

```
[tone, noise, cbt] = get_pertinents( s, fs [, n, bw, r, win ])
```

retourne les paramètres jugés pertinents à l'oreille du signal `s` échantillonné à la fréquence `fs`. Il est également possible de préciser le nombre `n` de fréquences à garder (par défaut 20), si on veut toutes les garder il suffit de mettre `n` à 0. Les largeurs d'analyse en bark sont passées par le paramètre `bw`, ainsi `bw(1)` est la largeur de la fenêtre de masquage fréquentiel, et `bw(2)` celle des bandes de bruit (par défaut `bw = [ 0.5, 1 ]`), à noter que si `bw` est un scalaire, ce sera `bw(1)` et `bw(2)` sera mis à sa valeur par défaut de 1 bark. `r` est le rapport en dB en dessous duquel une fréquence pure est masquée par le bruit avoisinant (c'est à dire l'énergie contenue dans la bande de `bw(1)` bark de large centrée sur la fréquence en question), par défaut `r` est de 5 dB. Enfin `win` permet de spécifier le type de fenêtre temporelle à utiliser avant de calculer la densité spectrale de puissance (par défaut une fenêtre de Hanning est utilisée).

La démarche est la suivante, on recherche toutes les fréquences pures présentes dans le signal avec la fonction `get_all_tones` (page 30) à laquelle on passe le paramètre `win`, les `nb` fréquences jugées les plus importantes sont sélectionnées grâce à `decimation` (page 38) qui prend en paramètres `nb`, `bw(1)` et `r`. Enfin, la densité spectrale de puissance diminuée des fréquences sélectionnées est partagée en bande de largeur `bw(2)` bark, et sur chacune d'elles le niveau d'énergie est moyenné.

On récupère donc un tableau `tone` de couple fréquences/amplitudes, un tableau de niveaux de bandes de bruit `noise`. Le dernier argument de sortie est également un tableau de couples fréquences/amplitudes, mais les fréquences gardées sont les plus pertinentes pour chacune des bandes de bruit. En d'autres termes, il y aura autant de fréquences pures sélectionnées que de bandes de bruit.

La fonction `get_pert_dir( dir [, n, bw, r, win ] )` applique `get_pertinents` à tous les fichiers wave contenus dans le répertoire `dir`.

3.4.5 Get_pert_harm

La fonction

```
[harm, tone, noise] = get_pert_harm( file [, n, w, bw ])
```

recherche les sequences harmoniques dans le fichier wave `file` dont le fondamental (et/ou le premier harmonique) est une fréquence jugée pertinente au sens psychoacoustique. Pour cela, la fonction `get_pertinents` est appelée avec les arguments `n` et `bw` (voir page 41), les fréquences récupérées sont transmises à `get_funds` avec le paramètre `w` (voir page 32), éliminant ainsi toutes fréquences multiples, celles qui restent sont alors envoyées à la fonction `get_harmonics` avec le paramètre `w` (voir page 33).

La fonction retourne un tableau de cellules contenant les séquences harmoniques trouvées (`harm`), le tableau des couples fréquences/amplitudes jugés pertinents `tone` et le tableau des niveaux de bruits (`noise`). Si aucun argument de sortie n'est précisé, les résultats sont enregistrés dans un fichier texte portant le même nom que le fichier wave `file` avec le préfixe `'pert_harm_'`.

3.4.6 Mean_pertinents et analyse_pert

La fonction

```
[tones, noises, cbt] = mean_pertinents( files [, n, w, bw, r ])
```

permet d'effectuer la recherche des composantes pertinentes dans un ensemble de fichiers wave représentant le même son, l'argument `file` est alors une cellule contenant les noms des fichiers à traiter.

On procède de la manière suivante : un tableau `hits` de triplet (fréquences, niveaux cumulés, hit) est créé à partir des sorties de la fonction `get_pertinents` appliquée à chacun des fichiers de `files`, ainsi chaque fois qu'une fréquence deja présente dans `hits` est à nouveau jugée comme pertinente, son amplitude est ajoutée au niveau cumulé et le compteur hit est incrémenté. Puis, lorsque l'ensemble des fichiers a été traité, les `n` fréquences dont les niveaux cumulés (pondérés par le seuil d'audition) sont les plus importants, sont sélectionnées, l'intérêt de ce type de selection est qu'elle tient compte à la fois du niveau et du nombre d'apparitions des fréquences. Ainsi une fréquence qui apparaît très souvent, même avec un niveau moyen, sera privilégiée face à une fréquence qui ne serait présente que dans un seul fichier, même avec un niveau important. On s'affranchit donc des accidents de prise de son.

Sont récupérés un tableau `tones` de couple fréquences/amplitudes (les amplitudes étant simplement récupérées en divisant le niveau cumulé par le nombre de hit), un tableau de niveaux moyens de bandes de bruit `noises`. Le dernier argument de sortie est également un tableau de couples fréquences/amplitudes ne contenant que les fréquences les plus pertinentes pour chacune des bandes de bruit. Si aucun argument de sortie n'est précisé, les résultats sont enregistrés dans un fichier texte.

La fonction `analyse_pert( dir [, n, w, bw, r, len, nf ])` prépare l'ensemble des sons au format wave du répertoire `dir` grâce à la routine `prepare_dir` à laquelle on passe les paramètres `len` et `nf` (voir page 53). Puis `mean_pertinents` est appliquée sur chacun des sous-répertoires ainsi créé. A noter que si les sons avaient déjà été préparés (lors d'un précédent appel, direct ou indirect, de `prepare_dir`), `analyse_pert` ne le refera pas.

3.4.7 Mean_pert_harm et analyse_pert_harm

La fonction

```
[Harm, tone, noise] = mean_pert_harm( files [, n, w, bw, r])
```

effectue le même type de traitement que `get_pertinent_harms` (page 42) mais sur un ensemble de fichiers wave représentant le même son. Pour ce faire, on commence par appliquer `mean_pertinents` (voir page 43), puis sont sélectionnées parmi les fréquences jugées pertinentes les candidates à être fondamentales grâce à `get_funds` (voir page 32), enfin les séquences harmoniques sont recherchées avec `mean_harmonics` (voir page 35).

A l'instar de toutes les fonctions préfixées par `mean_`, l'argument `files` peut être un tableau des noms des fichiers à analyser, ou une chaîne de caractères, dans ce cas la fonction appliquera son traitement sur tous les fichiers wave du répertoire courant dont le nom contient la chaîne.

La sortie `Harm` est un tableau de séquences harmoniques, `tones` est le tableau de couples fréquences/amplitudes jugés pertinents, et enfin `noises` et le niveau de chaque bande de bruits. Si il n'y a aucun argument de sortie, les résultats sont sauvegardés dans un fichier texte dont le nom est `'mean_pert_harm_str.txt'` si le premier argument est une chaîne de caractères, `'mean_pert_harm_date,heure.txt'` si le premier argument est un tableau de noms de fichier.

La fonction `analyse_pert_harm( dir [, n, w, bw, r, len, nf ])` prépare l'ensemble des sons au format wave du répertoire `dir` grâce à la routine `prepare_dir` à laquelle on passe les paramètres `len` et `nf` (voir page 53). Puis `mean_pertinent_harms` est appliquée sur chacun des sous-répertoires ainsi créés. A noter que si les sons avaient déjà été préparés (lors d'un précédent appel, direct ou indirect, de `prepare_dir`), `analyse_pert_harm` ne le refera pas.

3.4.8 Synthetize

La fonction `signal = synthetize( tones [, noises, adjust, fs, s, nf])` est le générateur de signal de cette boite à outils. L'argument d'entrée `tones` est un tableau 2 lignes, n colonnes représentant les couples fréquences/amplitudes des sinus que l'on veut additionner. `noises` est un vecteur de niveau par bandes de bruit, ainsi le spectre audible est séparé en autant de bandes de largeur fixe (dans l'échelle des barks) qu'il a de case à `noises`, la densité spectrale de puissance du signal de sortie aura alors pour chacune de ces bandes la valeur correspondante du tableau `noises` multipliée par `adjust` (1 par défaut), en effet il est apparu que pour des sons très riches en fréquences pures, le son synthétisé directement avec les niveaux de bruit calculés semble trop bruité, dans ce cas un `adjust` inférieur à 1 permet de corriger cet effet. Si on veut générer un signal uniquement constitué de bruit, il suffit de mettre [] à la place de `tones`.Par défaut, `noises` est nul, ce qui équivaut à un signal sans bruit. Il est également possible de préciser la fréquence d'échantillonnage du signal grâce à `fs` (44,1kHz par défaut), la durée en seconde du signal avec `s` ($\frac{32768}{\texttt{fs}}$ par défaut, soit un signal de 32768 échantillons). Enfin `nf` permet la normalisation en amplitude du signal (`nf = 1`), ce qui peut être nécessaire dans le cas d'un signal généré trop faible (inaudible), ou à l'inverse trop fort (saturation). Par défaut la normalisation n'est pas active (`nf = 0`).

Une version avec deux arguments de sortie `tonal` et `noisy` est disponible, dans ce cas les partie tonale (uniquement constituée de sinusoïdes) et la bruitée (uniquement constituée de bandes de bruit de largeur constante dans l'échelle des barks) sont séparées. Il est alors possible de reconstituer le signal final en additionnant ces deux vecteurs. Si trois arguments de sortie sont cités, ce seront respectivement le signal total, la fréquence d'échantillonnage et 16 représentant le nombre de bits de quantification. Ceci permet d'avoir, si besoin est, une sortie homogène à celle de la fonction `wavread`. Enfin, l'absence de paramètre de sortie implique que le son synthétisé sera enregistré dans le répertoire courant au format wave, son nom sera `'SYNTH_date,heure.wave'`.

3.5 Fonctions annexes

3.5.1 Cell2file

La fonction `cell2file( carray, file [, nb_min])` enregistre de la manière la plus lisible le tableau de cellules `carray` dans le fichier `file`. `carray` est typiquement la sortie de `get_harmonics` ou `mean_harmonics`... Si `file` existe déjà, la sauvegarde se fait à la suite du fichier.

Le paramètre `nb_min` permet de préciser la longueur minimale des cellules à sauvegarder, par défaut à 1.

Il existe aussi la forme

```
cell2file( carray, file , header [, nb_min])
```

il est possible de spécifier une entête `header` qui sera écrite juste avant les donnés proprement dites.

Enfin, le paramètre `file` peut être remplacé par un identifiant de fichier, c'est à dire la valeur de retour de la fonction *MATLAB*© `fopen`. Dans ce cas, le fichier ne sera pas fermé à l'intérieur de `cell2file`. C'est à l'utilisateur de faire appel à la fonction *MATLAB*© `fclose`.

3.5.2 Cell_fusion

La fonction `cout = cell_fusion( cin1, cin2 [, wid ] )` permet de fusionner deux tableaux de cellules `cin1` et `cin2` pour n'en faire qu'un seul `cout`. Ainsi les cellules représentant les mêmes séquences harmoniques seront fusionnées grâce à `fusion` (page 50. Sont considérée comme représentant la même séquence harmonique deux cellules dont les fréquences fondamentales respectives `F1` et `F2` vérifient l'une des relations suivantes :

$$|\texttt{F0} - \texttt{F1}| \leq \frac{\texttt{wid}}{2}$$

ou

$$\left|\frac{\texttt{F0}}{2} - \texttt{F1}\right| \leq \frac{\texttt{wid}}{2}$$

ou encore

$$\left|\texttt{F0} - \frac{\texttt{F1}}{2}\right| \leq \frac{\texttt{wid}}{2}$$

La valeur de `wid` est de 4 Hz par défaut. Attention tout de même que les tableaux de cellules à fusionner soient classés par ordre croissant de fondamentaux pour que l'algorithme marche, on pourra donc utiliser `cell_sort` (page 49).

```
>> c1{1} = [ 440*(1:1:4)', ones(4,1) ]';
>> c1{2} = [ 550*(1:1:3)', ones(3,1) ]';
>> c2{1} = [ 880*(1:1:4)', 2*ones(5,1) ]';
>> c2{2} = [ 555*(1:2:4)' 2*ones(4,1) ]';
>> c3 = cell_fusion( c1, c2 )

c3 =

    [2x6 double]    [2x3 double]    [2x4 double]

>> c3{1}, c3{2}, c3{3}

ans =

  1.0e+003 *

    0.4400    0.8800    1.3200    1.7600    2.6400    3.5200
    0.0010    0.0015    0.0010    0.0015    0.0020    0.0020
```

```
ans =

     550        1100        1650
       1           1           1


ans =

     555        1665        2775        3885
       2           2           2           2


>> c3 = cell_fusion( c1, c2, 10 )

c3 =

    [2x7 double]    [2x5 double]

>> c3{1}, c3{2}

ans =

  1.0e+003 *

    0.4400    0.8800    1.3200    1.7600    2.6400    3.5200
    0.0010    0.0015    0.0010    0.0015    0.0020    0.0020


ans =

  1.0e+003 *

    0.5525    1.1000    1.6575    2.7625    3.8850
    0.0015    0.0010    0.0015    0.0015    0.0020
```

3.5.3 Cell_sort

La fonction `cout = cell_sort( cin )` permet de classer le tableau de cellules `cin` dans l'ordre croissant de ses fondamentaux.

```
>> c = cell(1,3);
>> c{1} = [ 680*(1:5)', ones(5,1)]';
>> c{2} = [ 550*(1:3)', ones(3,1)]';
>> c{3} = [ 1000*(1:6)', ones(6,1)]';
>> c = cell_sort( c );
>> c{1}(1,1), c{2}(1,1), c{3}(1,1)

ans =

   550


ans =

   680


ans =

  1000
```

3.5.4 Cut_wavfile

La function `cut_wavfile( wavfile [, dirout, len, nf ] )` découpe le fichier wave `wavfile` en plusieurs fichiers au format wave de longueur `len` (32768 échantillons par défaut), les enregistre dans le répertoire `dirout` (répertoire courant par défaut). Le format des noms des fichiers ainsi créés est le suivant : 'X_nomdorigine.wav' si le fichier d'origine est mono, 'voieY_X_nomdorigine.wav' si le fichier d'origine est stéréo. Le dernier paramètre `nf` permet de spécifier si les fichiers ainsi créés seront normalisés en amplitude (`nf = 1`) ou pas (`nf = 0` valeur par défaut).

3.5.5 Fusion

La fonction `out = fusion( in1, in2 )` permet, à partir des deux tableaux représentant la même séquence harmonique, de n'en faire qu'un. Deux fréquences `F1` et `F2` seront alors considérées comme égales si elles correspondent au même ordre d'harmonique, c'est à dire si elles vérifient :

$$round\left(\frac{\texttt{F1} - \texttt{F2}}{\texttt{F0}}\right) = 0$$

où `F0` est la fréquence fondamentale des deux séquences. Le tableau `out` contient donc toutes les fréquences de `in1` et `in2`, avec l'amplitude et la fréquence moyenne là où ces dernières coïncident. Voici un exemple de fusion de deux séquences harmoniques :

```
>> a = [ 440*(1:5)', ones(5,1) ]';
>> b = [ 480*(1:6)', 2*ones(6,1) ]';
>> fusion( a, b )

ans =

  1.0e+003 *

    0.4600    0.9200    1.3800    1.8400    2.3000    2.8800
    0.0015    0.0015    0.0015    0.0015    0.0015    0.0020
```

3.5.6 Get_all_dir

La fonction `subdir = get_all_dir( [ dir ] )` recherche tous les sous-répertoires du répertoire `dir` (répertoire courant par défaut), puis crée un tableau `subdir` contenant les noms des sous-répertoires trouvés. On peut voir un exemple d'utilisation de cette fonction :

```
>> cd PREPARED
>> subdir = get_all_dir;
>> subdir( 2 )

ans =

'vh_100_2200ft'
```

3.5.7 Get_all_files

La fonction `noms = get_all_files( str )` retourne un tableau contenant les noms des fichiers wave dont le nom contient la chaîne de caractères `str`. Voici un exemple d'utilisation de cette fonction :

```
>> files = get_all_files( 'vh_100' );
>> files(3)

ans =

    '12_vh_100_2200ft.wav'
```

3.5.8 Insertion

`Cout = insertion( array, Cin, index )` permet d'insérer le tableau de couples fréquences/amplitudes `array` dans le tableau de cellules `Cin`, à la cellule `index`. L'exemple suivant illustre les possibilités de cette fonction :

```
>> c = cell(1,2);
>> c{1} = 'debut'; c{2} = 'fin';
>> c

c =

'debut'    'fin'

>> c = insertion('au milieu',c,2)

c =

'debut'    'au milieu'    'fin'

>> c = insertion('apres',c,4)

c =

'debut'    'au milieu'    'fin'    'apres'
```

3.5.9 Noises2file

La fonction `noises2file( noises, file [, header ] )` permet d'enregistrer le tableau de niveaux de bruit `noises` dans le fichier nommé `file`. On peut ajouter un texte à écrire avant les données grâce à `header`. L'argument `file` peut être remplacé par un identifiant de fichier, c'est à dire la valeur de retour de la fonction *MATLAB*© `fopen`. Dans ce cas, le fichier ne sera pas fermé à l'intérieur de `cell2file`. C'est à l'utilisateur de faire appel à la fonction *MATLAB*© `fclose`.

3.5.10 Now2str

La fonction `str = now2str([ f ])` convertit le flottant `f`, sortie de la fonction *MATLAB*© `now`, en une chaîne de caractères `str` representant la date et l'heure. Si `f` n'est pas précisé, l'appel de `now` se fait à l'intérieur de la fonction. L'exemple suivant l'illustre.

```
>> n = now;
>> now2str( n )

ans =

18-Jul-2002,17h59mn44s

>> now2str

ans =

18-Jul-2002,17h59mn58s
```

3.5.11 Prepare_dir

La fonction `prepare_dir([ dir, len, nf ])` prépare de manière automatique les fichiers wave contenus dans le répertoire `dir` (le répertoire courant si non précisé) en vue de leur traitement ultérieur. Un sous-répertoire nommé `'PREPARED'` est créé, puis à l'intérieur de ce dernier, autant de sous-répertoires qu'il n'y a de fichiers wave. Chacun de ces sous-répertoires portant le nom d'un fichier wave (à l'extension près bien sûr). Puis la fonction `cut_wavfile` entre en scène, avec les paramètres `dir`, `len` et `nf`, les résultats étant rangés dans le sous-répertoire approprié. Les valeurs par défaut sont : `len = 32768` et `nf = 0` c'est à dire que la normalisation est désactivée. Voici un exemple de l'utilisation de cette fonction :

```
>> dir

.       ..      vh_80_2200ft.wav    vh_100_2200ft.wav

>> prepare_dir
>> dir


.
..
PREPARED
vh_80_2200ft.wav
vh_100_2200ft.wav

>> cd PREPARED; dir

.       ..      vh_80_2200ft    vh_100_2200ft

>> cd vh_100_2200ft
>> dir


.
..
0_vh_100_2200ft.wav
1_vh_100_2200ft.wav
2_vh_100_2200ft.wav
3_vh_100_2200ft.wav

    etc...
```

3.5.12 Sec2hour

La fonction `str = sec2hour( f )` convertie un flottant `f` représentant un nombre de secondes, en une chaîne de caractères formatée comme suit :`'h :mn :s.ms'`. Si `f` est inférieur à une seconde, seule les millisecondes seront affichées. L'exemple suivant montre l'utilisation de cette fonction :

```
>> sec2hour( 6856.25 )

ans =

1:54:16.250

>> sec2hour( .2565 )

ans =

256.5 ms
```

3.5.13 Tones2file

La fonction `tones2file( tones, file )` permet d'enregistrer le tableau de couples fréquences/amplitudes dans le fichier texte nommé `file`. Ce dernier paramètre pouvant être remplacé par un identifiant de fichier, c'est à dire la valeur de retour de la fonction *MATLAB*© `fopen`. Dans ce cas, le fichier ne sera pas fermé à l'intérieur de `cell2file`. C'est à l'utilisateur de faire appel à la fonction *MATLAB*© `fclose`.

3.6 Installation de la boite à outils, utilisation et conclusion.

Pour installer la boite à outils, c'est très simple, copier le répertoire "SoundToolBox" sur l'ordinateur, ouvrez *MATLAB*©, sélectionner le menu "File", sélectionner le sous-menu "Set Path...", la boite de dialogue "Path Browser" apparaît, sélectionner le menu "Path", l'entré "Add to Path..." ouvre une boite de dialogue avec une zone d'édition, il suffit de remplir cette dernière avec le chemin du répertoire "SoundToolBox" précédemment copié, cliquer sur "OK", fermer la boite de dialogue "Path Browser". Une nouvelle boite apparaît alors, si vous voulez utiliser la boite à outils uniquement pour cette session de *MATLAB*©, cliquer sur "Non", sinon sur "Oui". Vous pouvez maintenant vous servir de la "SoundToolBox".

Pour bien utiliser la boite à outils, on copie les sons à traiter dans un répertoire, puis on appelle la fonction `prepare_dir` (section 3.5.11 page 53) en lui passant en paramètres le chemin du répertoire où se trouvent les sons et éventuellement le nombre d'échantillons que doivent contenir les fichiers une fois préparés. Dans un premier temps, on utilisera la fonction `clone_wave` (page 37) pour adapter les paramètres à notre besoin et aux sons à synthétiser. Une fois qu'un compromis entre la qualité auditive et le nombre de fréquences pures et de bandes de bruit a été trouvé, on pourra lancer `analyse_pert` avec les paramètres satisfaisant. Si les sons à analyser ont la même origine physique, mais avec des paramètres différents (bruit de machine tournante à des régimes moteur différents par exemple), il peut être avantageux d'utiliser `analyse_pert_harm` pour identifier les suites harmoniques communes et ainsi pouvoir analyse manuellement leurs évolutions. En effet, si la finalité est la reproduction d'un environnement sonore variant lentement dans le temps, les outils développés ne peuvent se substituer à une analyse réelle de l'évolution des paramètres du son en fonction de l'état à simuler. Ils permettront tout de même un gain de temps en analysant chacun des régimes différents de manière indépendante.

Il peut arriver que les approximations dans le modèle psychoacoustique (notamment dans les fonctions `critical_band` et `decimation` (pages 23 et 38)) entraînent une différence audible entre le signal original et le signal de synthèse. C'est alors à l'utilisateur d'agir sur les paramètres passés aux fonctions, et éventuellement de n'utiliser ces outils que dans un but de présélection des tons à garder pour la synthèse.

Dernier point, il faut garder à l'esprit que les phénomènes non stationnaires (les bruits de pales d'hélicoptère ou de culbuteurs de moteur diesel par exemple) ne sont absolument pas reproductibles par ce type de modèle. Dans pareils cas, il faudra prévoir la synthèse de tels signaux par un sous-système spécifique (un bruit blanc modulé en amplitude par un signal carré suffit pour simuler des pales d'hélicoptères).

En revanche, la modélisation du signal sonore en un nombre fini de sinus et de bandes de bruit a été validée par des tests d'écoute. Les résultats sont le plus souvent très satisfaisant en gardant un nombre très faible de données. Ainsi, des souffles aérodynamiques ou des bruits de roulements sont fidèlement reproduits avec seulement 20 sinus et 25 bandes de bruit.

Packages *MATLAB*© nécessaires :

1. `optimize` pour la fonction `fsolve`

Conclusion sur le stage

Sur un plan technique, ce stage a confirmé mon savoir faire dans certains domaines comme le traitement du signal ou encore la programmation. Mais plus important, il a démontré que je pouvais être autonome et m'adapter. En effet, Le sujet initial du stage, "l'étude de la distorsion photo-sonique", n'a pu être mené à son terme à cause de l'absence de signaux de tests et d'une faisabilité incertaine. En fin de stage, j'ai également abordé un troisième sujet, l'encapsulation sous forme d'une bibliothèque de lien dynamique (DLL Windows) de l'algorithme de transposition temporelle se trouvant au coeur de l'Harmo (voir présentation de l'entreprise page 6).

De plus, le sujet de l'étude des sons stationnaires s'étant concrétisé par le développement d'une boîte à outils entièrement configurable par l'utilisateur, une partie importante de ce projet a servi à la rédaction du manuel d'utilisation (page 18). Il est certain que l'absence d'un tel manuel aurait grandement diminué l'intérêt d'une telle boîte d'outils, puisqu'alors elle aurait été difficilement utilisable par quelqu'un d'autre que moi.

Enfin, ce stage dans une petite entreprise m'a fait prendre conscience des difficultés qu'elles peuvent rencontrer. Car même si les compétences de la société GENESIS dans l'étude et le traitement des signaux sonores ne sont plus à démontrer, cette dernière a déposé le bilan. J'ai donc assisté aux processus administratifs qui accompagnent cette situation, me permettant d'apprendre quelques petites choses du droit du travail et de la vie juridique des entreprises.

J'ai donc vécu ce stage comme une véritable transition de l'école vers le monde professionnel, et je pense être enfin prêt à travailler en entreprise.