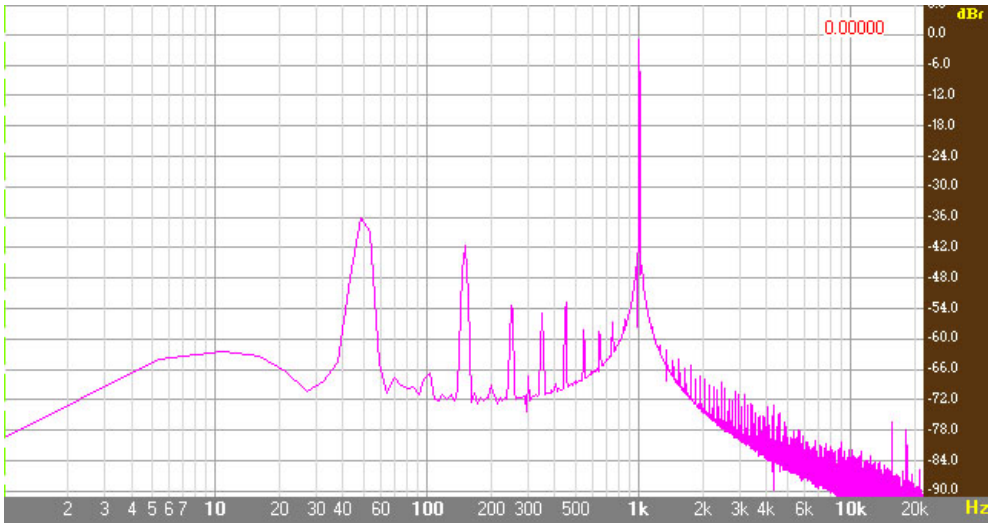
Université de Nantes
IUT de Saint Nazaire
Département Mesures Physiques

Année 2013-2014

Travaux Pratiques d'initiation au traitement des signaux de mesure

F. Auger

7 février 2014



Module	"Techniques de Traitement du Signal"
Pré-requis	Modules "Mathématiques I" et "Mathématiques II"
Objectifs	Comprendre les bases mathématiques et les techniques de traitement du signal
Programme	**Bases mathématiques du traitement du signal** Les différents types de signaux (déterministes, continus, discrets). Représentation mathématique des signaux. Analyse spectrale. Convolution. Corrélation. Échantillonnage. Filtrage. **Mise en œuvre des techniques de traitement du signal** Traitement des signaux à temps continu : convolution et corrélation, analyse spectrale et interprétation énergétique, modulation (d'amplitude, de phase, de fréquence), échantillonnage, antirepliement. Traitement des signaux à temps discret : filtrage, convolution, corrélation, FFT, reconstitution de signal à temps continu. **Outils matériels et logiciels pour le traitement du signal** logiciels industriels, analyseur spectral.
Compétences	à l'issue de ce module, l'étudiant doit connaître les opérations mathématiques mises en œuvre dans le traitement du signal et savoir exploiter leurs potentialités.

Table des Matières

Déroulement souhaité des séances de TP

séance	exercices
1	1, 2, 3,
2	7, 9, 11 12,
3	13 15, 16
4	17, 19, 20
5	21, 22, 23
6	examen de TP

Dans vos comptes-rendus de TP, vous aurez souvent l'occasion d'écrire des équations mathématiques. Il est donc très vivement conseillé de faire un compte-rendu **manuscrit** plutôt que dactylographié.

Chapitre 1

Travaux pratiques de traitement du signal

1.1 Introduction

L'objectif de ces exercices de travaux pratiques est de montrer quelques possibilités de mise en œuvre pratique des techniques de traitement du signal. Quelques logiciels libres vont être utilisés pour cela, en particulier les logiciels SoX, *Audacity* et *Visual Analyser*. Ce n'est évidemment pas la maîtrise de ces logiciels qui est importante, mais les techniques de traitement du signal qu'ils permettent de pratiquer et de mettre en œuvre sur des situations réelles.

Le Logiciel SoX[1] peut être qualifié de *couteau suisse du traitement du signal*. Il est principalement destiné à la manipulation de signaux acoustiques, mais il peut parfaitement être utilisé pour des signaux d'autres origines : absolument rien n'empêche de s'en servir pour des signaux provenant d'un capteur quelconque (autre qu'un microphone). C'est un logiciel *libre* (que vous avez donc la liberté d'utiliser, d'étudier et de modifier) et *multiplateforme* (il peut donc être exécuté sur des ordinateurs fonctionnant sous windows, sous linux ou sous Mac Os X). C'est un logiciel en ligne de commande (voir figure 1.1), ce qui veut dire que son utilisation ne se fait pas au moyen d'une interface homme-machine évoluée utilisant des menus, des boutons et des champs de saisie, mais en appelant le programme suivi de paramètres indiquant les signaux sur lesquels on souhaite travailler et les opérations que l'on souhaite leur appliquer. Ce mode d'utilisation permet d'utiliser SoX dans des programmes d'application écrits dans n'importe quel langage. La forme générale de ces lignes de commande est systématiquement de la forme

```
sox InputFile OutputFile Effect EffectParameters
```

Durant ces exercices pratiques, on pourra utiliser SoX sous windows soit à partir d'une fenêtre de commande `DOS`[2], soit en écrivant des fichiers de commandes[3], d'extension `.bat`, que l'on lancera ensuite. Ces fichiers de commandes correspondent à des programmes contenant des instructions exécutées séquentiellement les unes après les autres.

La documentation de ce logiciel décrit succintement l'ensemble des possibilités de ce logiciel. L'objectif de ces exercices pratiques est de présenter quelques-unes des fonctions de génération,

[1] Ce logiciel est disponible pour les systèmes d'exploitation Windows, Linux et Mac OsX à l'adresse http://sox.sourceforge.net . Il a été crée en 1991 par Lance Norskog. La toute première version s'appelait AUX, diminutif de *Aural Exchange*. Depuis 1996, il est maintenu par Chris Bagwel. Ce logiciel en était à sa version 14.3.0 en janvier 2010, 14.3.1 en janvier 2011 et 14.4.0 en mars 2012, ce qui montre la vitalité et la maturité de ce développement.

[2] Sous windows, on pourra utiliser avec profit le programme `Console` de Marko Bozicovic, disponible à l'adresse http://sourceforge.net/projects/console/

[3] Voir http://en.wikipedia.org/wiki/Batch_file. Pour plus de renseignements sur les commandes disponibles sous `DOS`/windows, voir http://en.wikipedia.org/wiki/List_of_MS-DOS_commands.

Figure 1.1: Exemple d'utilisation de SoX dans une fenêtre de commande windows.

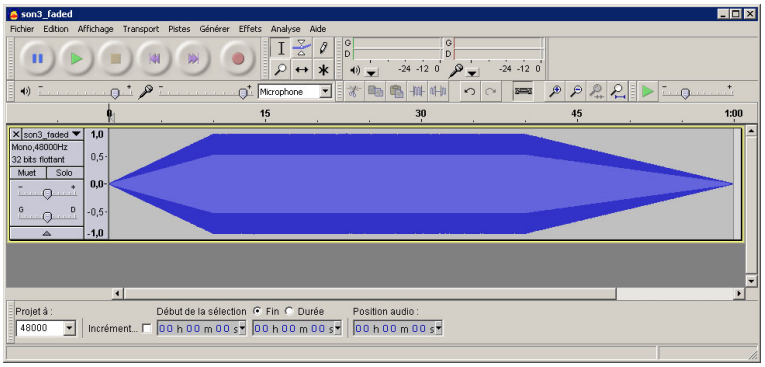


Figure 1.2: Fenêtre principale du logiciel Audacity.

d'analyse et de traitement de signaux offertes par ce logiciel, pour bien comprendre et utiliser correctement ses possibilités. On s'intéressera uniquement aux traitements qui ne sont pas spécifiques de l'acoustique, incitant vivement ceux qui sont intéressés par les effets acoustiques et musicaux à les étudier ultérieurement.

`Ch` est un interpréteur `C/C++` multiplateforme conçu pour le calcul scientifique ainsi que pour l'enseignement de l'algorithmique et de la programmation informatique. Il est totalement conforme à l'ISO C 1990 et intègre une grande partie de l'ISO C 1999. Il n'accepte cependant qu'une partie du C++. Il a d'abord été développé par Harry H. Cheng (University of California, Davis), qui a ensuite confié son évolution et sa commercialisation à la société *SoftIntegration Inc.* Une version étudiante est disponible gratuitement pour les utilisateurs qui s'enregistrent sur le site `http://www.softintegration.com`.

`Audacity` est également un logiciel[4] libre et multiplateforme. Ses fonctionnalités sont parfois comparables, mais bien souvent complémentaires de celles de SoX : il possède lui aussi des fonctions de génération, de traitement et d'analyse de signaux acoustiques. Il fonctionne dans un environnement graphique (voir figure 1.2), ce qui permettra de visualiser facilement des signaux.

Enfin, `Visual Analyser` est un logiciel gratuit[5] sous windows, qui permet de disposer d'un oscilloscope, d'un analyseur de spectre, d'un générateur BF et d'un fréquence-mètre (voir figure 1.3). Par défaut, il est relié à la carte son du PC, mais il est aussi possible de l'utiliser avec d'autres cartes d'acquisition.

[4]Ce logiciel est disponible à l'adresse `http://audacity.sourceforge.net` .
[5]Ce logiciel est disponible à l'adresse `http://www.sillanumsoft.com/` .

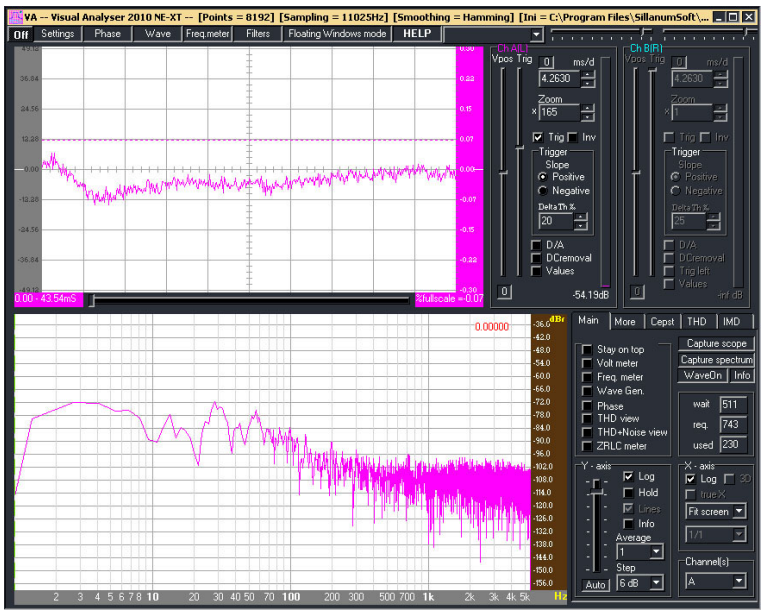


Figure 1.3: Fenêtre principale du logiciel `Visual Analyser`.

1.2 La génération des signaux élémentaires

1.2.1 Rappels préliminaires

Un signal est une *fonction* d'une ou plusieurs variables qui a la particularité d'être engendré par un phénomène physique et d'être porteur d'information et non d'énergie. Les signaux de parole, les signaux acoustiques et les tensions électriques fournies par des capteurs sont des exemples simples de signaux. On distingue différents types de signaux :

- les signaux à temps continu, qui ont une valeur (mesurable) à tout instant.
- les signaux à temps discret, qui n'ont de valeurs (mesurables) qu'à certains instants.
- les signaux échantillonnés, qui sont des signaux à temps discret pour lesquels l'intervalle de temps entre deux mesures consécutives est constant.
- les signaux déterministes, dont la valeur peut être parfaitement prédite grâce à un modèle mathématique qui permet de calculer leur valeur exacte à tout instant.
- les signaux aléatoires, qui ne peuvent être parfaitement prédits, parce que ce n'est pas possible ou pas nécessaire.
- les signaux causaux (ou causals), qui sont nuls avant l'instant pris comme origine des temps. Ils traduisent les répercussions d'un évènement particulier qui s'est produit à l'instant pris comme origine des temps.
- les signaux stationnaires, qui véhiculent la même information pendant toute leur durée d'observation. Tous les signaux périodiques sont stationnaires, mais les signaux stationnaires ne sont pas tous périodiques : par exemple, le signal déterministe défini par l'expression $x(t) = 10\ \sin(2\pi 50t) + 5\ \sin(2\pi 50\sqrt{2}t)$ n'est pas periodique, car c'est la somme de deux sinusoïdes dont les fréquences sont dans un rapport irrationnel.

1.2.2 Génération de formes d'ondes élémentaires

Pour tester les traitements et les analyses qui seront étudiées par la suite, il peut être utile de générer des signaux tests synthétiques (qui ne proviennent pas de l'observation d'un phénomène physique réel). La commande à utiliser pour cela est `synth`. Le fichier de commandes ci-dessous (`Generation1.bat`) comprend quelques lignes de commentaires (commençant par l'instruction `rem`) et 6 appels au programme SoX :

```
rem generation de signaux elementaires avec SoX
rem F. Auger, IUT Saint-Nazaire, dep. MP, dec. 2009

sox -n s1.mp3 synth 3.5 sine 440
sox -n s2.wav synth 90000s sine 660:1000
sox -n s3.mp3 synth 1:20 triangle 440
sox -n s4.mp3 synth 1:20 trapezium 440
sox -V4 -n s5.mp3 synth 6 square 440 0 0 40
sox -n s6.mp3 synth 5 noise

for %%i in (200,300,400) do ^
sox -n s7_%%i.mp3 synth 15 sine %%i

rem ecoute du resultat : le fichier de sortie est le canal
rem de sortie par defaut (-d), c'est a dire la carte son

sox s1.mp3 -d

pause
```

- La première commande génère un fichier `s1.mp3` contenant un enregistrement de 3.5 s d'une sinusoïde de fréquence 440 Hz. La commande `-n` (null) est utilisée pour indiquer l'absence de signal d'entrée.
- La deuxième commande génère un fichier `s2.wav` contenant un enregistrement de 90 000 échantillons (*samples* en anglais) d'un signal sinusoïdal dont la fréquence varie linéairement de 660 à 1000 Hz.
- La troisième commande génère un fichier `s3.mp3` contenant un enregistrement de une minute et 20 secondes d'un signal triangulaire.
- La quatrième commande génère un fichier `s4.mp3` contenant un enregistrement de une minute et 20 secondes d'un signal trapézoïdal.
- La cinquième commande génère un fichier `s5.mp3` contenant un enregistrement de 6 secondes d'un signal carré ayant un offset nul (pas de composante continue) une phase nulle et un rapport cyclique de 50 La commande `-V4` permet de choisir la quantité d'information fournie par SoX à l'utilisateur : son bavardage (*verbosity*) va de 0 (aucune information renvoyée à l'utilisateur) à 4 (affichage du plus grand nombre d'informations).
- La sixième commande génère un fichier `s6.mp3` contenant un enregistrement de 5 secondes d'un bruit blanc de valeur moyenne nulle et compris entre -1 et 1.
- La septième commande utilise une structure de répétition pour générer 3 fichiers contenant un enregistrement de 15 secondes d'une sinusoïde . La variable `%%i` est utilisée à la fois pour le nom du fichier et pour la fréquence de la sinusoïde . Le symbole `^` indique que l'instruction continue à la ligne suivante.
- La dernière commande envoie le contenu du fichier `s1.mp3` vers le périphérique par défaut (`-d`), c'est à dire vers la carte son. Cela permet d'écouter ce signal.

Exercice I.1: *Écrire un fichier de commandes qui génère* 10 *fichiers contenant des enregistrements de* 2.5 *s de sinusoïdes dont les fréquences sont linéairement espacées entre* 100 *et* 1000 *Hz. Quelle est l'expression analytique de ces signaux ?*

1.2.3 Opérations élémentaires sur les signaux

Quelques opérations élémentaires peuvent être appliquées sur des signaux existants. Le fichier de commandes ci-dessous (`Generation2.bat`) en effectue quelques-unes :

```
rem transformations elementaires de signaux avec SoX
rem F. Auger, IUT Saint-Nazaire, dep. MP, jan. 2010

sox s1.mp3 s1_faible1.mp3 vol -6 dB
sox s1.mp3 s1_faible2.mp3 vol -0.4 amplitude

sox s1.mp3 s2.mp3 DeuxSons1.mp3
sox s2.mp3 -v 0.6 s1_faible1.mp3 DeuxSons2.mp3

sox s1.mp3 s1_avec_silence1.mp3 pad 1
sox s1.mp3 s1_avec_silence2.mp3 pad 1 0.5
sox s1.mp3 s1_avec_silence3.mp3 pad 1 5000s@3 0.5

sox -m s3.mp3 s4.mp3 SommeSons.mp3

sox s1.mp3 s1_avec_offset.mp3 dcshift 0.05

sox s1.mp3 s1_a_l_envers.mp3 reverse

sox s3.mp3 s3_morceau.mp3 trim 1.5 2

sox s3.mp3 s3_faded.mp3 fade t 10 1:00 20

rem pause
```

- Les deux premières commandes appliquent des gains sur le signal contenu dans le fichier `s1.mp3`. Il est de -6 dB dans le premier cas (ce qui correspond à un facteur 0.5) et de -0.4 dans le second (ce qui correspond à une réduction de l'amplitude de 60 % et à un déphasage de -180 degrés).
- Les deux commandes suivantes mettent deux signaux à la suite l'un de l'autre pour générer une *séquence* de signaux. Dans le deuxième cas, une réduction de l'amplitude du premier signal par un facteur $\times 0.6$ est appliquée sur le premier signal.
- Les trois commandes suivantes ajoutent des valeurs nulles (donc du silence) au signal contenu dans le fichier `s1.mp3`. La première rajoute une seconde de silence au début, la seconde une seconde au début et 0.5 s à la fin, et la troisième insère également 5000 échantillons (*samples*) à partir de la troisième seconde du signal.
- La commande suivante ajoute (*mix*) les deux signaux `s3.mp3` et `s4.mp3`.
- La commande suivante ajoute une composante continue égale à 0.05.
- La commande suivante retourne un signal (la fin devient le début, le début devient la fin).
- La commande suivante extrait un morceau du signal contenu dans le fichier `s3.mp3`, en commençant à 1.5 s et avec une durée de 2 s. Il permet d'extraire facilement un morceau de signal pour l'analyser ou le modifier.
- La dernière commande permet d'appliquer une modulation d'amplitude sur un signal. Cela permet de faire varier progressivement l'amplitude d'un signal, au début et à la fin. Le paramètre `-t` correspond à une modulation linéaire. Le deuxième paramètre indique que l'amplitude du signal passe de 0 à sa valeur maximale pendant les 10 premières secondes, le suivant indique que la décroissance commence au bout d'une minute et le dernier indique que la décroissance linéaire s'achève au bout de 20 s. Les deux derniers paramètres sont facultatifs. D'autres types de modulation sont disponibles, comme l'indique la documentation du logiciel.

Exercice I.2: *Écrire un fichier de commandes qui génère un fichier que l'on appellera* `succession.mp3` *contenant les* 10 *signaux générés lors de la question 1, placés*

successivement les uns après les autres. Écrire ensuite un autre fichier de commandes qui génère un second fichier appelé `Sommesons.mp3`*, qui contient la somme des* 10 *fichiers générés lors de la question 1. Quelle est l'expression analytique de ce deuxième signal ? Quelle propriété mathématique vérifie t-il ?*

Exercice I.3: *Écrire un fichier de commandes qui génère un enregistrement d'une minute d'un signal égal à la somme de deux sinusoïdes de fréquences* 440 *et* 441 *Hz et de même amplitude. Quelle est l'allure de ce signal ? Justifiez cette allure en utilisant l'une ou l'autre de ces deux relations*

$$\begin{aligned}\cos(\theta_1)+\cos(\theta_2) &= 2\cos\left(\frac{\theta_1+\theta_2}{2}\right)\cos\left(\frac{\theta_1-\theta_2}{2}\right)\\ \sin(\theta_1)+\sin(\theta_2) &= 2\sin\left(\frac{\theta_1+\theta_2}{2}\right)\cos\left(\frac{\theta_1-\theta_2}{2}\right)\end{aligned}$$

Écrire enfin un fichier de commandes qui extrait 3 *secondes du fichier*

`AriaCantilenaVillaLobosPetibon.mp3`

à partir de 5 *min* 24 *s, puis écouter et visualiser ce morceau avec* `Audacity`*. Faire le lien avec la question précédente.*

Exercice I.4: *Générer d'abord un fichier* `Sinus.mp3` *contenant une sinusoïde de fréquence* 440 *Hz, puis un fichier* `Bruit.mp3` *contenant un bruit aléatoire, tous les deux d'une durée de* 7 *secondes. Utiliser ensuite une commande du type*

```
sox -m Sinus.mp3 -v 0.01 Bruit.mp3 SinusBruite.mp3
```

pour additionner la sinusoïde au bruit, ce dernier étant atténué par un facteur 0.01*. En dessous de quelle valeur du facteur d'atténuation le bruit n'est-il plus perceptible à l'oreille ?*

Utiliser enfin une commande du type

```
sox -m Bruit.mp3 -v 0.01 Sinus.mp3 SinusBruite.mp3
```

pour additionner le bruit à la sinusoïde, cette dernière étant atténuée par un facteur 0.01*. En dessous de quelle valeur du facteur d'atténuation la sinusoïde n'est-elle plus perceptible à l'oreille ?*

Exercice I.5: *L'objectif de cet exercice est de montrer qu'il est possible de générer des fichiers assez complexes qui pourront ensuite être utilisés pour des essais ou des contrôles.*

Écrire un fichier de commandes qui génère un fichier `mp3` *contenant un enregistrement de* 2 *secondes de la somme de* 6 *sinusoïdes de fréquence* f_0*,* $2\,f_0$*,* $3\,f_0$*,* $4\,f_0$*,* $5\,f_0$ *et* $6\,f_0$*, pour* $f_0 = 523\,Hz$*. Générer ensuite* 2 *autres fichiers obtenus pour des valeurs respectives de* f_0 *de* 659 *Hz, et* 830 *Hz. Générer enfin un fichier contenant la succession de ces* 3 *sons, puis un dernier qui répète* 3 *fois le fichier précédent*[6]*.*

Pour faciliter la réalisation de cet exercice, on pourra utiliser avec un tableur le fichier appelé `GenerationCode.xls`*, qui génère automatiquement un ensemble d'instructions* SoX*, qu'il suffit ensuite de copier dans un fichier de commande.*

[6] En 1968, Jean-Claude Risset a construit sur ce principe un son ascendant perpétuel, qui donne l'impression de devenir toujours de plus en plus aigu.

Exercice I.6: *Écrire un fichier de commande qui*

- *génère deux signaux sinusoïdaux ayant une durée de* 10 *secondes et de fréquences* 12000 *et* 12440 *Hz. Ces deux signaux seront stockés dans des fichiers avec une extension* `.wav`.
- *calcule la somme de ces deux signaux et la stocke dans un troisième fichier avec une extension* `.wav`.

Écoutez ensuite le signal obtenu (de préférence avec un casque). Qu'entendez vous ? Justifiez ce résultat[7].

1.2.4 Utilisation d'autres formats de fichier

Dans les paragraphes précédents, les signaux ont été stockés dans des fichiers au format `mp3`. Mais SoX peut aussi utiliser un très grand nombre de formats de fichiers, que ce soit en lecture ou en écriture. Parmi les très nombreux formats utilisables, décrits dans la documentation du logiciel, on peut citer le format `.wav` utilisé par le système d'exploitation *windows*, et le format `.dat`, qui correspond à des fichiers de texte (en code ASCII), dans lesquels la première ligne correspond à la fréquence d'échantillonnage, et les lignes suivantes contiennent deux nombres : le temps et la valeur du signal. Par exemple le fichier de commandes ci-dessous (`Generation3.bat`)

```
rem generation d'un fichier texte au format .dat
rem F. Auger, IUT Saint-Nazaire, dep. MP, jan. 2010

sox s1.mp3 s1.dat silence 0 trim 0.1 10s
```

génère un fichier `s1.dat` contenant 10 valeurs successives (`10s`) du fichier `s1.mp3` prises 0.1 s après le début du signal (en ayant retiré les valeurs nulles qui sont au début du fichier). Le contenu du fichier `s1.dat` est le suivant :

```
; Sample Rate 48000
; Channels 1
               0      -0.68915808
  2.0833333e-005      -0.65037082
  4.1666667e-005      -0.60942747
       6.25e-005      -0.56646369
  8.3333333e-005      -0.52161656
   0.00010416667      -0.47505816
        0.000125      -0.42689275
   0.00014583333      -0.37733861
   0.00016666667      -0.32652545
       0.0001875      -0.27462651
```

Fabriquer des fichiers de ce type va permettre de manipuler des signaux de mesure qui ne sont pas d'origine acoustique avec SoX . Il suffit de normaliser ces signaux pour qu'ils soient toujours compris entre -1 et 1. En choisissant une fréquence d'échantillonnage adaptée à l'oreille humaine, cela permet d'écouter des phénomènes non acoustiques. Le programme ci-dessous (`GenerationFichierDat2010.ch`), écrit en langage C pour l'environnement Ch, lit des nombres dans un fichier texte, en extrait le maximum et le minimum et génère ensuite un fichier `.dat` en appliquant une règle de trois sur les valeurs du fichier texte, pour que les valeurs entre $x_{\min}$ et $x_{\max}$ deviennent des valeurs entre -1 et 1. Le résultat peut ensuite être transformé en fichier `mp3` et être écouté, à l'aide de la commande

```
sox -V4 Mesures.dat Mesures.mp3
```

[7] Cette technique est utilisée par les haut-parleurs directionnels pour générer des ondes acoustiques à l'aide de transducteurs ultrasonores. Voir K. Muira, "Haut-parleur directionnel à ultrasons", Elektor, mars 2011, pp 64–67.

```
//  Programme permettant de transformer un fichier de mesures au format texte
//  en fichier de mesures au format .dat
//  F. Auger, janvier 2010

#include <stdio.h>

int main()
{
 char   NomFichierTexte[] = "Mesures.txt" ;
 char   NomFichierDat[] = "Mesures.dat" ;
 FILE   *PointeurFichierTexte, *PointeurFichierDat ;
 double x, xmax, xmin, xnorm, Somme, Difference, Fe=16000.0, Te=1.0/Fe, t;

 // on ouvre une premiere fois le fichier pour trouver le min et le max
 PointeurFichierTexte = fopen(NomFichierTexte, "r") ;
 if ( PointeurFichierTexte == NULL )
  {printf("Impossible d'ouvrir le fichier %s en lecture \n",
          NomFichierTexte);}
 else
  {
   // premiere lecture dans le fichier pour initialiser xmin et xmax
   if (fscanf(PointeurFichierTexte,"%lf", &x) != EOF)
    {xmin=x ; xmax=x;}

   while (fscanf(PointeurFichierTexte,"%lf", &x) != EOF)
    {
     if (x>xmax) {xmax=x;}
     if (x<xmin) {xmin=x;}
    }
   fclose(PointeurFichierTexte);

   Somme=xmax+xmin; Difference = xmax-xmin; t=0;

   // generation du fichier .dat par une relecture du fichier texte
   PointeurFichierTexte=fopen(NomFichierTexte,"r");// ouverture en lecture
   PointeurFichierDat  =fopen(NomFichierDat,  "w");// ouverture en ecriture

   if ( PointeurFichierDat == NULL )
    {printf("Impossible d'ouvrir le fichier %s en ecriture\n",
            NomFichierDat);}
   else
    {
     // on precise la frequence d'echantillonnage du signal de sortie, qui
     // n'est pas forcement la meme que celle du signal d'entree
     fprintf(PointeurFichierDat,"; Sample Rate %8.2f\n", Fe);

     // un seul signal = une seule voie = un seul canal
     fprintf(PointeurFichierDat,"; Channels 1\n");

     while (fscanf(PointeurFichierTexte,"%lf", &x) != EOF)
      {
       xnorm=(2*x-Somme)/Difference;
       fprintf(PointeurFichierDat, "%f %f \n", t, xnorm);
       t=t+Te;
      }

     fclose(PointeurFichierTexte);// c'est fini, on ferme les fichiers
     fclose(PointeurFichierDat);
    }
  }
 return 0;
}
```

Exercice I.7: *Si x est compris entre $x_{\min}$ et $x_{\max}$, dans quelles intervalles sont compris*

$$\frac{x - x_{\min}}{x_{\max} - x_{\min}} \quad et \quad 2\frac{x - x_{\min}}{x_{\max} - x_{\min}} - 1 \quad ?$$

En déduire une justification de l'expression de **`xnorm`** *utilisée dans le programme. Modifier ensuite le programme précédent pour convertir au format* **`.dat`** *le fichier* **`ecg.txt`**. *Ce fichier contient un électrocardiogramme échantillonné à 720 Hz et acquis*

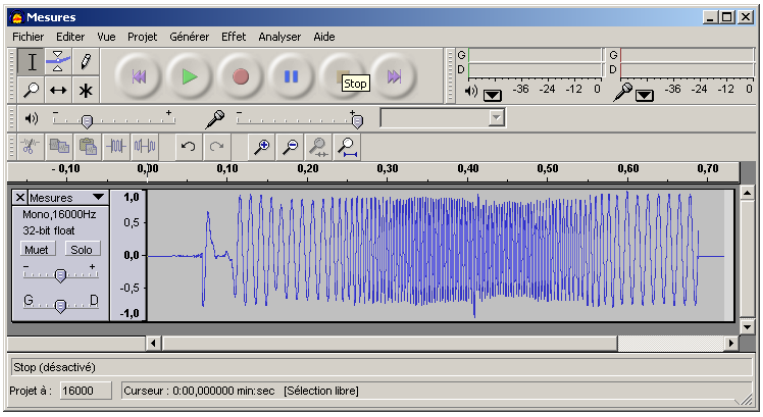


Figure 1.4: Résultat de la fabrication d'un fichier `mp3` à partir des courants électriques mesurés aux bornes d'une machine asynchrone.

avec un convertisseur analogique-numérique 12 bits[8]*. Ce fichier pourra ensuite être converti au format* `.wav` *avec* SoX*, puis visualisé avec* `Audacity`*.*

Ce format peut aussi être utilisé pour utiliser des signaux générés par SoX dans d'autres logiciels, par exemple dans un tableur du type `LibreOffice Calc`[9].

1.3 L'analyse des signaux

1.3.1 Mesure de caractéristiques temporelles et fréquentielles d'un signal

Trois commandes peuvent être utilisées pour analyser un signal à l'aide de certaines de ses caractéristiques :

– Le programme `soxi`, fourni avec SoX, délivre des informations sur la structure d'un fichier. Ainsi le fichier de commandes (`Analyse1.bat`)

```
rem analyse d'un fichier avec soxi
rem F. Auger, IUT Saint-Nazaire, dep. MP, jan. 2010

soxi son1.mp3
soxi son1.mp3 > son1_info_soxi.txt
pause
```

renverra les informations indiquées ci-dessous. Avec les première commande, ces informations sont affichées à l'écran. Avec la seconde, ces informations sont stockées dans un fichier `son1_info_soxi.txt`.

```
Input File     : 'son1.mp3'
Channels       : 1
Sample Rate    : 48000
Precision      : 16-bit
Duration       : 00:00:03.55 = 170496 samples ~ 266.4 CDDA sectors
File Size      : 28.4k
Bit Rate       : 64.0k
Sample Encoding: MPEG audio (layer I, II or III)
```

[8]Source : http://www.physionet.org/physiobank/database/aami-ec13/
[9]Voir http://fr.libreoffice.org/

– La commande `stat` de SoX détermine certains paramètres caractéristiques d'un signal. Ainsi le fichier de commandes[10] (`Analyse2.bat`)

```
rem analyse d'un fichier avec sox stat
rem F. Auger, IUT Saint-Nazaire, dep. MP, jan. 2010

sox son1.mp3 -n stat
sox son1.mp3 -n stat 2> son1_info_stat.txt
pause
```

renverra les informations ci-dessous. Avec la première commande, ces informations sont affichées à l'écran. Avec la seconde, ces informations sont stockées dans un fichier appelé `son1_info_stat.txt`.

```
Samples read:            169344
Length (seconds):      3.528000
Scaled by:         2147483647.0
Maximum amplitude:     0.983058
Minimum amplitude:    -0.977098
Midline amplitude:     0.002980
Mean    norm:          0.599965
Mean    amplitude:     0.000013
RMS     amplitude:     0.669044
Maximum delta:         0.059710
Minimum delta:         0.000000
Mean    delta:         0.034553
RMS     delta:         0.038529
Rough   frequency:          439
Volume adjustment:       1.017

Try: -t raw -U -1
```

– Enfin, la commande `stats` de SoX fournit d'autres paramètres caractéristiques d'un signal. Ainsi le fichier de commandes (`Analyse3.bat`)

```
rem analyse d'un fichier avec sox stats
rem F. Auger, IUT Saint-Nazaire, dep. MP, jan. 2010

sox son1.mp3 -n stats
sox son1.mp3 -n stats 2> son1_info_stats.txt
pause
```

renverra les informations ci-dessous. Avec la première commande, ces informations sont affichées à l'écran. Avec la seconde, ces informations sont stockées dans un fichier appelé `son1_info_stats.txt`.

```
DC offset   0.000013
Min level  -0.977098
Max level   0.983058
Pk lev dB      -0.15
RMS lev dB     -3.49
RMS Pk dB      -3.44
RMS Tr dB      -3.89
Crest factor    1.47
Flat factor     0.00
Pk count           2
Bit-depth      29/29
```

[10] Attention ! il ne faut pas mettre d'espace entre "2" et ">". Précisons à l'intention des spécialistes que le flux de sortie normal de SoX est `stderr` et non pas `stdout`. La syntaxe `2>` permet de faire une redirection du flux `stderr` vers un fichier, tandis que la commande `>` ou `1>` effectue une redirection du flux `stdout`. Si on n'utilise aucune redirection, les résultats apparaissent bien évidemment à l'écran.

```
Num samples     169k
Length s       3.528
Scale max   1.000000
Window s       0.050
```

La documentation de SoX fournit des informations sur la nature de ces paramètres caractéristiques, qui sont trop nombreux pour détailler chacun d'entre eux. Certains d'entre eux sont redondants.

Exercice I.8: *Analysez et vérifiez les résultats obtenus par le fichier de commandes*[11] *ci-dessous (`Analyse20130624.bat`). On s'intéressera en particulier à la valeur efficace du signal.*

```
rem analyse de signaux elementaires avec SoX
rem F. Auger, IUT Saint-Nazaire, dep. MP, june 2013

sox -n ana_s1.wav synth 5 sine 440
sox -n ana_s2.wav synth 5 triangle 440
sox -n ana_s3.wav synth 4 square 440 0 0 40

sox ana_s1.wav -n stat 2>  AnaStat.txt
sox ana_s2.wav -n stat 2>> AnaStat.txt
sox ana_s3.wav -n stat 2>> AnaStat.txt

rem pause
```

Exercice I.9: *Quel est la fréquence d'échantillonnage des signaux générés dans la question 1 ? Écrire un fichier de commandes qui analyse de façon très détaillée le fichier `SonRigolo3.mp3`. Commentez les résultats obtenus.*

1.3.2 Analyse temps-fréquence d'un signal

La structure de certains signaux peut être clairement mise en évidence en utilisant une représentation temps-fréquence. Cette représentation correspond à une distribution de l'énergie du signal dans un plan constitué des deux dimensions temporelles et fréquentielles. Cela permet de voir comment la représentation fréquentielle du signal évolue au cours du temps. L'image obtenue s'apparente à une partition musicale : le temps est en abscisse et la fréquence en ordonnée. Par exemple, le fichier de commandes (`Analyse4.bat`)

```
rem analyse d'un fichier avec sox spectrogram
rem F. Auger, IUT Saint-Nazaire, dep. MP, jan. 2010

sox -n s6a.wav synth 3 sine  660-2640
sox -n s6b.wav synth 3 sine 1320-5280
sox -n s6c.wav synth 3 sine 1980-7920
sox -m s6a.wav s6b.wav s6c.wav s6.wav

sox s6.wav -n spectrogram       -o s6_sp.png
sox s6.wav -n spectrogram -m    -o s6_sp2.png
sox s6.wav -n spectrogram -l    -o s6_sp3.png
sox s6.wav -n spectrogram -l -m -o s6_sp4.png
sox s6.wav -n spectrogram -l -m -S 0.5 -d 1.3 -o s6_sp5.png
```

[11] L'utilisation de la commande ≫ permet de mettre tous les résultats dans le même fichier : le flux de sortie est redirigé vers la fin du fichier `AnaStat.txt`.

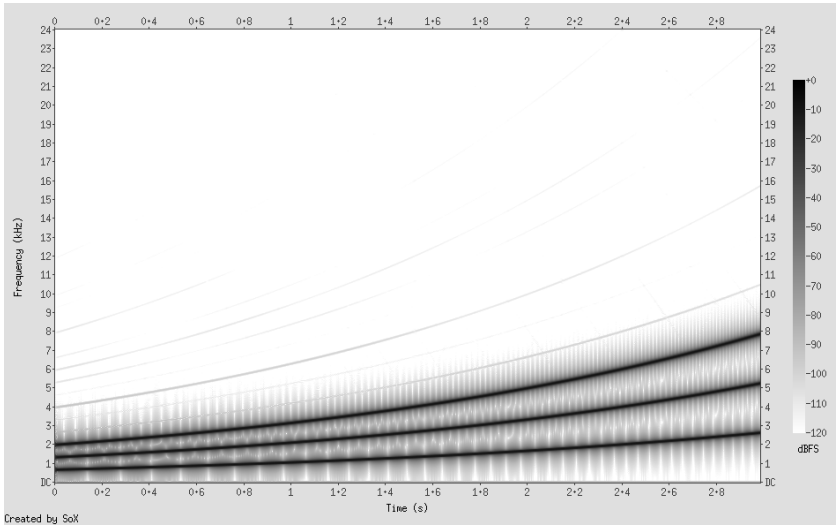


Figure 1.5: Représentation temps-fréquence du signal `s6.wav`, obtenue à l'aide d'un spectrogramme. Le sigle `dBFS` signifie *dB full scale* et indique que les niveaux de gris correspondent à la valeur en dB du spectrogramme normalisé par sa valeur maximale.

```
rem pause
```

génère un fichier `s6.mp3` contenant un enregistrement de 3 secondes de la somme de trois sinusoïdes dont les fréquences augmentent progressivement dans le temps. Ce signal est ensuite analysé à l'aide d'un spectrogramme, qui est une méthode particulière d'analyse temps-fréquence. Le résultat de cette analyse est stocké dans le fichier de sortie (`-o`) `s6_sp.png`. La commande suivante génère le même spectrogramme en utilisant des niveaux de gris plutôt que de la couleur, ce qui se prête mieux à une impression de cette représentation en utilisant une imprimante monochrome. Les deux commandes suivantes utilisent l'option[12] `-l`, qui permet d'avoir un fond blanc. La dernière commande effectue une analyse du signal en commençant (`-S`) à l'instant 0.5 s et sur une durée (`-d`) de 1.3 s. Le résultat de l'avant dernière commande est présenté figure 1.5. Entre autres choses, cette représentation montre clairement la composante fondamentale et ses harmoniques, dont les fréquences augmentent progressivement dans le temps. La documentation de SoX détaille les nombreux paramètres de la commande `spectrogram`.

Exercice I.10: *Utilisez des spectrogrammes pour faire la différence entre les trois signaux générés par les commandes suivantes :*

```
sox -n chirp1.wav synth 3 sine  1000:20000
sox -n chirp2.wav synth 3 sine  1000+20000
sox -n chirp3.wav synth 3 sine  1000/20000
```

Exercice I.11: *Compléter le fichier de commandes de l'exercice 9 pour construire un spectrogramme du signal `SonRigolo3.mp3` complet et de certaines de ses parties. Commentez les résultats obtenus.*

[12] Attention ! C'est la lettre L minuscule et non pas le chiffre 1 (un).

1.4 Le traitement des signaux

1.4.1 Filtrage “analogique”

Il est courant, parce qu'avantageux, d'utiliser des systèmes linéaires stationnaires pour conditionner des signaux de mesure, c'est à dire pour mieux faire ressortir l'information qu'ils véhiculent. En général, ces systèmes sont

- *dynamiques* : leur évolution est régie par une équation différentielle (pour des systèmes à temps continu) ou par une équation de récurrence (pour des systèmes à temps discret) ;
- *monovariables* :ils ont une seule grandeur d'entrée (appelée *excitation*) et une seule grandeur de sortie (appelée *réponse*) ;
- *causaux* : la valeur à l'instant t de leur sortie ne dépend que des valeurs de leur entrée aux instants $t' \leq t$. Cette propriété traduit le fait que les effets ne précèdent pas les causes.
- *linéaires* : ils vérifient le principe de superposition, qui déclare que la réponse d'un système à une somme d'excitations élémentaires est la somme des réponses du système à chacune des excitations prises séparément.
- *stationnaires* : si la même excitation est envoyée deux fois de suite à l'entrée d'un système, dans des conditions initiales strictement identiques, les deux réponses obtenues seront elles aussi strictement identiques. Cette propriété traduit le fait que les mêmes causes produisent les mêmes effets, ce qui revient à négliger l'évolution, par usure ou par vieillissement, du système.

Lorsqu'un système à temps continu vérifie toutes ces propriétés, on peut démontrer que sa sortie $y(t)$ se déduit de son entrée $x(t)$ par une opération appelée un *produit de convolution* :

$$y(t) = \int_0^{+\infty} h(\tau)\, x(t-\tau)\, d\tau$$

où $h(t)$ est une fonction caractéristique du système appelée sa réponse impulsionnelle. Une conséquence importante de ce résultat est que la réponse d'un système linéaire stationnaire à une sinusoïde est une sinusoïde de même fréquence, amplifiée et déphasée par des termes qui ne dépendent que du système utilisé et de la fréquence de la sinusoïde :

$$\begin{array}{lrcl} \text{si} & x(t) & = & X\, e^{\jmath 2\pi f t}, \\ \text{alors} & y(t) & = & H_{\mathcal{F}}(f)\, X\, e^{\jmath 2\pi f t}, \\ \text{avec} & H_{\mathcal{F}}(f) & = & \int_{-\infty}^{+\infty} h(\tau)\, e^{-\jmath 2\pi f \tau}\, d\tau \end{array}$$

Le coefficient de proportionnalité entre $y(t)$ et $x(t)$, $H_{\mathcal{F}}(f)$, est la transformée de Fourier de $h(t)$.

Plus généralement, si $y(t)$ est la réponse à une excitation $x(t)$ d'un système linéaire stationnaire, alors $Y_{\mathcal{F}}(f) = H_{\mathcal{F}}(f)\, X_{\mathcal{F}}(f)$, où $X_{\mathcal{F}}(f)$ et $Y_{\mathcal{F}}(f)$ sont les transformées de Fourier des signaux $x(t)$ et $y(t)$. Cette relation montre donc la possibilité de conditionner des signaux de mesure en utilisant des systèmes linéaires, qui vont se comporter dans le domaine fréquentiel comme un masque appliqué sur le signal d'éntrée, permettant de conserver des informations (en choisissant $H_{\mathcal{F}}(f) \approx 1$) et de faire disparaître des perturbations (en choisissant $H_{\mathcal{F}}(f) \approx 0$). Ils permettent donc d'extraire une information des perturbations dans lesquelles elle est noyée, si leurs localisations fréquentielles sont disjointes. Le logiciel SoX permet de filtrer facilement des signaux à l'aide d'équivalents à temps discret de filtres à temps continus du premier ou du second ordre. Le fichier de commandes ci-dessous (`Traitement1.bat`) illustre ces possibilités :

```
rem filtrage d'un signal avec les commandes sox
rem lowpass, highpass, bandpass et bandreject
rem F. Auger, IUT Saint-Nazaire, dep. MP, jan. 2010
```

```
sox SonRigolo3.mp3 SonRigolo_lp1.mp3 lowpass  -1  500
sox SonRigolo3.mp3 SonRigolo_hp2.mp3 highpass -2 2000 0.8q
sox SonRigolo3.mp3 SonRigolo_bp2.mp3 bandpass     200 5q
sox SonRigolo3.mp3 SonRigolo_br2.mp3 bandreject   200 0.6q
sox SonRigolo_bp2.mp3 SonRigolo_bp2b.mp3 gain -20

rem pause
```

- La première commande applique un filtre passe-bas (`lowpass`) du premier ordre (`-1`) sur le signal `SonRigolo3.mp3` avec une fréquence de coupure de 500 Hz. Le signal de sortie du filtre est stocké dans `SonRigolo_lp1.mp3`.
- La deuxième commande applique un filtre passe-haut (`highpass`) du second ordre (`-2`) sur le signal `SonRigolo.mp3` avec une fréquence de coupure de 2000 Hz et un facteur de qualité $Q = 0.8$. Le facteur de qualité Q est égale à $1/(2\,m)$, m étant le coefficient d'amortissement. Le signal de sortie du filtre est stocké dans `SonRigolo_hp2.mp3`.
- La troisième commande applique un filtre passe-bande (`bandpass`) du second ordre sur le signal `SonRigolo.mp3` avec une fréquence centrale de 200 Hz et un facteur $q = 5$. Le signal de sortie du filtre est stocké dans `SonRigolo_bp2.mp3`.
- La quatrième commande applique un filtre réjecteur ou coupe-bande (`bandreject`) du second ordre sur le signal `SonRigolo.mp3` avec une fréquence centrale de 200 Hz et un facteur $q = 0.6$. Le signal de sortie du filtre est stocké dans `SonRigolo_bp2.mp3`.
- Tous les filtres présentés ci-dessus ont un gain maximual unitaire. Pour amplifier ou atténuer le signal, on peut utiliser l'action `gain` de SoX. La cinquième commande applique un gain de -20 dB (donc une atténuation d'un facteur 10) du signal obtenu à la sortie du filtre passe-bande et stocke le résultat dans `SonRigolo_bp2b.mp3`.

Le logiciel SoX permet également de tracer le module de la réponse fréquentielle de ces filtres en utilisant la commande `--plot gnuplot`. Il génère alors un fichier de commandes pour le programme de tracé de courbes `gnuplot`[13]. SoX n'applique alors aucun traitement sur le signal d'entrée, mais il est quand même nécessaire de le désigner pour que le programme connaisse sa fréquence d'échantillonnage. Par exemple, le fichier de commandes ci-dessous (`Traitement2.bat`) permet d'obtenir les quatre réponses fréquentielles de la figure 1.6 :

```
rem trace de la reponse frequentielle des filtres obtenus avec
rem les commandes sox lowpass, highpass, bandpass et bandreject
rem F. Auger, IUT Saint-Nazaire, dep. MP, jan. 2010

sox --plot gnuplot SonRigolo3.mp3 -n lowpass  -1  500      > Courbe1.plt
sox --plot gnuplot SonRigolo3.mp3 -n highpass -2 2000 0.8q > Courbe2.plt
sox --plot gnuplot SonRigolo3.mp3 -n bandpass     200 5q   > Courbe3.plt
sox --plot gnuplot SonRigolo3.mp3 -n bandreject   200 0.6q > Courbe4.plt

pause
```

Les fichiers générés par SoX, ayant une extension `.plt`, sont des fichiers de commandes qui vont permettre à `gnuplot` de produire un graphique. Par défaut, `gnuplot` calcule 100 valeurs de la fonction représentée. Si cela ne suffit pas pour obtenir une représentation graphique satisfaisante de cette fonction, on peut utiliser une instruction du type `set samples 300` après la commande

```
set ylabel 'Amplitude Response (dB)'
```

Le fichier de commandes `gnuplot` généré par SoX ne permet malheureusement pas de zoomer sur certaines zones du graphique. Pour pouvoir le faire, il suffit d'ouvrir le fichier de commandes `gnuplot` (par exemple le fichier `Courbe1.plt`) avec un éditeur de textes et de retirer le `[-35:25]` dans la commande

```
plot [f=10:Fs/2] [-35:25] 20*log10(H(f))
```

[13] Le programme `gnuplot` est disponible à l'adresse http://www.gnuplot.info .

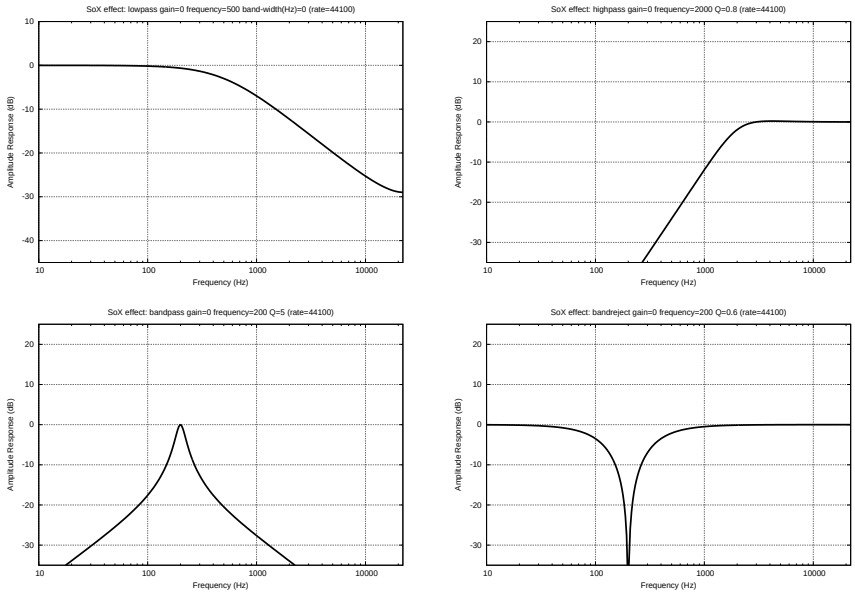


Figure 1.6: Réponses fréquentielles obtenues avec les commandes `lowpass`, `highpass`, `bandpass` et `bandreject` complétées par la commande `--plot gnuplot`.

qui impose les bornes inférieures et supérieures de l'axe des ordonnées.

Pour obtenir des réponses temporelles caractéristiques d'un filtre, on peut utiliser (en le personnalisant au cas par cas) le programme ci-dessous (`TestFiltre.ch`), qui génère un signal comprenant une impulsion, un échelon et une rampe, puis en filtrant ce signal en utilisant la commande SoX appropriée. La figure 1.7 montre le signal produit par ce programme ainsi que le résultat obtenu avec la commande

```
sox TestFiltre.wav TestFiltre2.wav lowpass -1 10
```

```
//  Programme permettant de generer un fichier pour tester des filtres
//  F. Auger, juin 2013

#include <stdio.h>

int main()
{
 char   NomFichierDat[] = "TestFiltre.dat" ;
 FILE   *PointeurFichierDat ;
 int i, NbPoints=2000;
 double Fe=16000.0, Te=1.0/Fe, t=0.0, Amp=0.9;

 // generation du fichier .dat par une relecture du fichier texte
 PointeurFichierDat  =fopen(NomFichierDat,  "w");// ouverture en ecriture

 if ( PointeurFichierDat == NULL )
  {printf("Impossible d'ouvrir le fichier %s en ecriture\n",
          NomFichierDat);}
 else
  {
   // on precise la frequence d'echantillonnage du signal
   fprintf(PointeurFichierDat,"; Sample Rate %8.2f\n", Fe);

   // un seul signal = une seule voie = un seul canal
   fprintf(PointeurFichierDat,"; Channels 1\n");

   for(i=0;i<NbPoints;i=i+1)
    {fprintf(PointeurFichierDat, "%f %f \n", t, 0.0); t=t+Te;}

   // l'impulsion
   fprintf(PointeurFichierDat, "%f %f \n", t, Amp); t=t+Te;

   for(i=0;i<NbPoints;i=i+1)
```

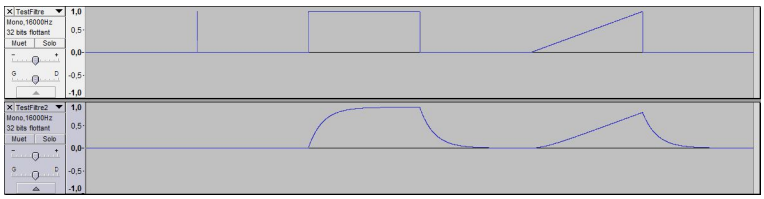


Figure 1.7: Signal obtenu avec le programme `TestFiltre.ch` et réponse d'un filtre passe-bas "analogique" du premier ordre de fréquence de coupure égale à 10 Hz à ce signal.

```
    {fprintf(PointeurFichierDat, "%f %f \n", t, 0.0); t=t+Te;}

   // l'echelon
   for(i=0;i<NbPoints;i=i+1)
    {fprintf(PointeurFichierDat, "%f %f \n", t, Amp); t=t+Te;}

   for(i=0;i<NbPoints;i=i+1)
    {fprintf(PointeurFichierDat, "%f %f \n", t, 0.0); t=t+Te;}

   // la rampe
   for(i=0;i<NbPoints;i=i+1)
    {fprintf(PointeurFichierDat, "%f %f \n",
             t, Amp*(i+1)/NbPoints); t=t+Te;}

   for(i=0;i<NbPoints;i=i+1)
    {fprintf(PointeurFichierDat, "%f %f \n", t, 0.0); t=t+Te;}

   fclose(PointeurFichierDat);
  }
 return 0;
}
```

Exercice I.12: *Rappelons tout d'abord que la fonction de transfert d'un filtre passe-bande du second ordre de pulsation centrale ω_0 et d'amortissement z s'écrit*

$$\begin{aligned} H(p) &= \frac{2\,z\,\omega_0\,p}{p^2+2\,z\,\omega_0\,p+\omega_0^2} \\ &= \frac{\frac{\omega_0}{Q}\,p}{p^2+\frac{\omega_0}{Q}\,p+\omega_0^2} = \frac{1}{1+Q\left(\frac{p}{\omega_0}+\frac{\omega_0}{p}\right)} \\ \text{avec}\quad \omega_0 &= 2\pi\,f_0 \quad \text{et} \quad Q=\frac{1}{2\,z}=\frac{\omega_0}{\Delta\omega}=\frac{f_0}{\Delta f} \end{aligned}$$

L'objectif de cet exercice est d'étudier la variation de la fréquence centrale réelle et la bande passante Δf d'un filtre passe-bande du second ordre en fonction de son facteur de qualité Q et de sa fréquence centrale souhaitée f_0. On tracera pour cela la réponse fréquentielle des filtres passe-bande obtenus en prenant différentes valeurs du facteur qualité Q (par exemple $Q = 2^{n/2}$, pour n allant de -2 à 3) et de f_0 (par exemple $f_0 = 300\,2^k$ pour k allant de 0 à 5), et on mesurera la fréquence centrale réelle et la bande passante Δf sur chaque réponse, pour les comparer aux valeurs théoriques. Pour cela, on pourra utiliser deux structures de répétitions imbriquées comme dans le fichier de commandes ci-dessous (`passe_bande.bat`) :

```
rem F. Auger, IUT Saint-Nazaire, dep. MP, mars 2012
rem %%i et %%j sont deux variables
rem le symbole ^ en fin de ligne indique
rem que la commande continue sur la ligne suivante

for %%f in (300,600,1200,2400,4800,9600) do ^
for %%Q in (0.5, 0.707, 1, 1.414, 2, 2.8) do ^
sox --plot gnuplot SonRigolo3.mp3 -n bandpass %%f %%Qq > bode_%%f_%%Q.plt

rem pause
```

Exercice I.13: *Filtrer le premier signal obtenu dans l'exercice 2 avec un filtre réjecteur de fréquence centrale égale à 500 Hz et de bande de réjection égale à 100 Hz. Mesurer l'amplitude des sinusoïdes successives pour en déduire des valeurs expérimentales de la largeur de bande de réjection et du module de la réponse fréquentielle de ce filtre, que l'on portera sur la courbe théorique.*

Exercice I.14: *La commande*

```
sox SonRigolo.mp3 SonRigolo_lp2.mp3 lowpass -2 500 3q
```

permet d'appliquer l'équivalent discret d'un filtre passe-bas analogique du second ordre, de gain statique égal à 1, de fréquence propre $f_0 = 500$ Hz et de facteur qualité $Q = 3$ sur le signal `SonRigolo.mp3`*. Le signal de sortie de ce filtre est stocké dans le fichier* `SonRigolo_lp2.mp3`*. On rappelle que la fonction de transfert d'un filtre passe-bas du second ordre de fréquence propre f_0 et de facteur qualité Q s'écrit*

$$H(p) = \frac{\omega_0^2}{p^2 + \frac{\omega_0}{Q}\, p + \omega_0^2} = \frac{\omega_0^2}{p^2 + 2\, z\, \omega_0\, p + \omega_0^2}$$

avec $\omega_0 = 2\pi\, f_0$ et $Q = \frac{1}{2\,z}$, soit $z = \frac{1}{2\,Q}$. La courbe du module de la réponse fréquentielle de ce filtre (exprimée en dB) peut être obtenue par la commande[14]

```
sox --plot gnuplot SonRigolo.mp3 -n lowpass -2 500 3q > lowpass2.plt
```

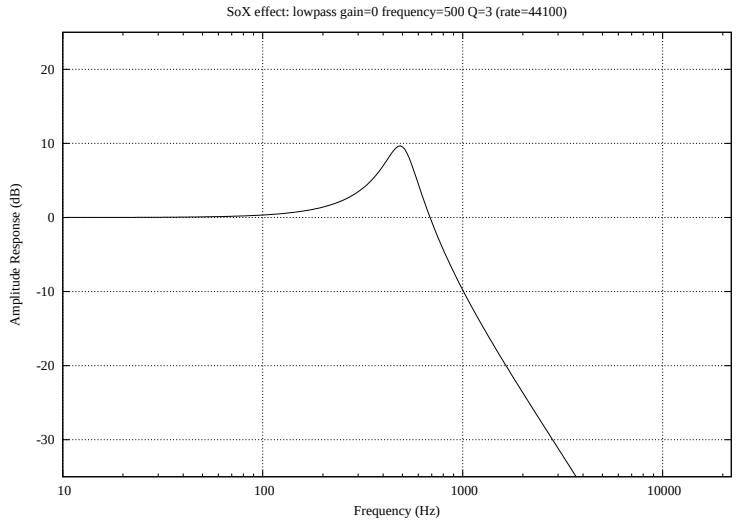


Figure 1.8: Réponse fréquentielle d'un filtre passe-bas du second ordre obtenue avec la commande `lowpass` complétée par la commande `--plot gnuplot`.

qui génère un fichier qui permettra avec `gnuplot` *d'obtenir un graphique du type de celui de la figure 1.8.*

Lorsque Q est supérieur à $\sqrt{2}/2$, ce filtre présente une résonance (c'est à dire un maximum supérieur à 1 à une fréquence non nulle) à la fréquence

$$f_r = f_0 \sqrt{1 - \frac{1}{2Q^2}} = f_0 \sqrt{\frac{2Q^2 - 1}{2Q^2}}$$

[14] La courbe obtenue avec cette commande est construite à partir de 100 valeurs de la réponse fréquentielle. Si cela ne suffit pas pour obtenir une représentation graphique satisfaisante, on peut modifier le fichier `.plt` en rajoutant après la commande `set ylabel 'Amplitude Response (dB)'` une instruction du type `set samples 300`. Cette instruction permettra de produire une courbe à partir de 300 valeurs de la réponse fréquentielle. À noter que pour pouvoir zoomer dans ce graphique, il suffit de retirer le "`[-35:25]`" dans l'instruction `plot [f=10:Fs/2] [-35:25] 20*log10(H(f))`.

Le module de la réponse fréquentielle à cette fréquence est égal à

$$G_r = |H(\jmath 2\pi f_r)| = \frac{2Q^2}{\sqrt{4Q^2-1}} = Q\sqrt{1+\frac{1}{4Q^2-1}}$$

On pourra remarquer que G_r est indépendant de f_0. La fréquence de coupure à -3 dB est égale à

$$f_c = f_0\sqrt{1-\frac{1}{2Q^2}+\sqrt{2-\frac{1}{Q^2}+\frac{1}{4Q^4}}}$$

Calculer $H(\jmath\omega_0)$. Quelles sont les limites de f_r, G_r et f_c quand Q tend vers l'infini ? Quelles sont les limites de f_r, G_r et f_c quand Q tend vers $\sqrt{2}/2$? Écrire un fichier de commande qui permet de tracer avec SoX *la réponse fréquentielle des filtres passe-bas du second ordre obtenus avec deux valeurs de Q (2 et 20) et deux valeurs de f_0 (1000 et 10000 Hz). Mesurer sur ces quatre courbes la fréquence de résonance f_r, le gain à la résonance G_r, le gain à la fréquence f_0 et la fréquence de coupure f_c. Comparer ces résultats aux valeurs théoriques et conclure.*

1.5 Filtrage numérique

Le logiciel SoX permet également de réaliser des opérations de filtrage à temps discret à l'aide des commandes `fir` et `biquad`. Le fichier de commandes ci-dessous (`Traitement3.bat`) illustre les possibilités d'utilisation de ces deux commandes :

```
rem filtrage a temps discret a l'aide des commandes sox fir et biquad
rem F. Auger, IUT Saint-Nazaire, dep. MP, jan. 2010

sox SonRigolo.mp3 SonRigolo_fir1.mp3 fir 0.1 0.2 0.4 0.3

sox SonRigolo.mp3 SonRigolo_fir2.mp3 fir CoeffsFIR.txt

sox SonRigolo.mp3 SonRigolo_biq.mp3 biquad 0.6 0.2 0.4 1 -1.5 0.6

sox --plot gnuplot SonRigolo.mp3 -n fir 0.2 0.2 0.2 0.2 0.2 > CourbeFir.plt

rem pause
```

- La première commande fabrique à partir du signal $x[n]$ contenu dans le fichier `SonRigolo.mp3` un nouveau signal $y[n]$ obtenu à l'aide d'un filtre à réponse impulsionnelle finie (*FIR*, *finite impulse response*) du troisième ordre, d'équation

$$y[n] = 0.1\,x[n] + 0.2\,x[n-1] + 0.4\,x[n-2] + 0.3\,x[n-3]$$

- La deuxième commande utilise un autre filtre à réponse impulsionnelle finie dont les coefficients sont contenus dans le fichier `CoeffsFIR.txt`. Dans ce fichier, les coefficients sont séparés par des espaces ou des passages à la ligne (retour chariot ou touche entrée du clavier). Les lignes commençant par le caractère `#` sont des lignes de commentaires. Voici un exemple de contenu du fichier `CoeffsFIR.txt` :

```
# filtre RIF du 4eme ordre
# F. Auger, jan 2010
0.35
0.62
0.51
0.62
0.35
```

- La troisième commande fabrique à partir du signal $x[n]$ contenu dans le fichier `SonRigolo.mp3` un nouveau signal $y[n]$ obtenu à l'aide d'un filtre à réponse impulsionnelle infinie (*IIR*, *infinite impulse response*) du second ordre, d'équation

$$y[n] = 0.6\,x[n] + 0.2\,x[n-1] + 0.4\,x[n-2] + 1.5\,y[n-1] - 0.6\,y[n-2]$$

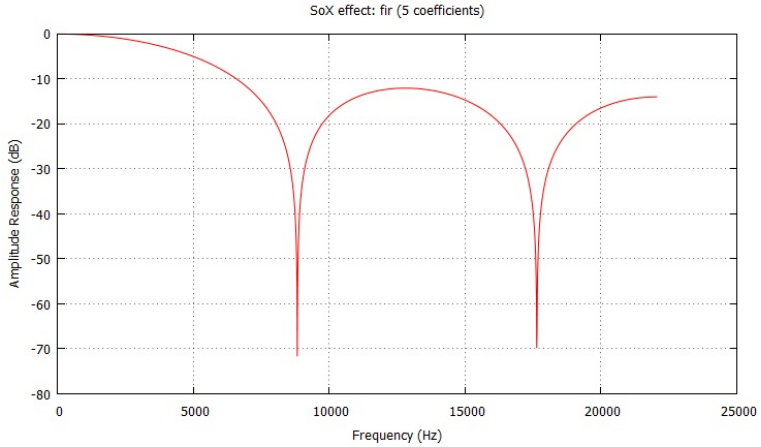


Figure 1.9: Réponse fréquentielle du filtre à réponse impulsionnelle finie obtenu avec la quatrième commande du fichier `Traitement3.bat`.

– La quatrième commande trace le module de la réponse fréquentielle d'un filtre à réponse impulsionnelle finie du quatrième ordre, d'équation

$$y[n] = \frac{1}{5}\left(x[n] + x[n-1] + x[n-2] + x[n-3] + x[n-4]\right)$$

Le résultat est présenté à la figure 1.9.

Exercice I.15: *Filtrer un signal constitué d'une succession de sinusoïdes de fréquences convenablement choisies avec un filtre moyenneur de longueur* 10*, d'équation*

$$y[n] = \frac{1}{10}\left(\sum_{k=0}^{9} x[n-k]\right)$$

En déduire des mesures expérimentales du module de la réponse fréquentielle de ce filtre aux fréquences successives de ce signal, que l'on comparera à sa valeur théorique. Vérifier expérimentalement que la réponse fréquentielle de ce filtre est nulle à la fréquence $F_e/10$ et déterminer expérimentalement sa fréquence de coupure.

Exercice I.16: *La commande* `sinc` *permet de filtrer un signal avec un type particulier de filtres numériques à réponse impulsionnelle finie. Le fichier de commandes ci-dessous (*`Partiel2010c.bat`*) illustre ses possibilités d'utilisation :*

```
rem Filtrage avec la commande sinc de SoX
rem F. Auger, IUT Saint-Nazaire, dep. MP, mars 2010

sox InputFile.mp3 OutputFile1.mp3 sinc -3500
sox InputFile.mp3 OutputFile2.mp3 sinc -3500 -n 43
sox InputFile.mp3 OutputFile3.mp3 sinc 4000

sox --plot gnuplot InputFile.mp3 -n sinc -12000 -n 151 > FreqResp.plt

rem pause
```

– *La première commande applique sur le signal contenu dans le fichier* `InputFile.mp3` *un filtre passe-bas dont la fréquence de coupure (à* -6 *dB) est de* 3500 *Hz. Le signal obtenu à la sortie du filtre est stocké dans le fichier* `OutputFile1.mp3`*.*
– *La seconde commande fait la même chose, en précisant que* `InputFile.mp3` *doit être filtré avec un filtre d'ordre* 42 *(la sortie du filtre est alors calculée à partir de* 43 *valeurs du signal d'entrée). Si l'utilisateur ne l'impose pas, comme dans le premier cas,* SoX *choisit lui-même l'ordre du filtre.*

- *La troisième commande applique un filtre passe-haut dont la fréquence de coupure (à -6 dB) est de 4000 Hz.*
- *La dernière commande trace le module de la réponse fréquentielle d'un filtre passe-bas d'ordre 150 et dont la fréquence de coupure (à -6 dB) est de 12 kHz.*

Concevoir un signal comprenant une succession de sinusoïdes dont les fréquences vont de 200 à 1400 Hz, par pas de 50 Hz. Chaque sinusoïde aura une durée de 1.5 s. Filtrer ensuite ce signal à l'aide d'un filtre passe-bas d'ordre 215 et de fréquence de coupure (à -6 dB) égale à 800 Hz. Sous quelle forme s'écrit la relation entre le signal $y[n]$ obtenu à la sortie de ce filtre et le signal $x[n]$ appliqué à l'entrée ? Mesurer les amplitudes des sinusoïdes obtenues à l'entrée et à la sortie de ce filtre et en déduire des valeurs expérimentales du module de sa réponse fréquentielle, que l'on comparera à la courbe théorique que l'on tracera à l'aide du logiciel.

On souhaite filtrer un signal pour ne conserver que ses composantes comprises entre 310 et 1240 Hz, à l'exception de celles comprises entre 490 et 510 Hz. Proposer une solution permettant de faire cela avec des filtres de ce type et l'appliquer sur le signal généré au début de cet exercice, pour vérifier que l'on obtient bien la réponse fréquentielle désirée.

1.5.1 Échantillonnage

Bien comprendre les conditions de bon échantillonnage des signaux est indispensable pour manipuler et traiter correctement des signaux à temps discret. Rappelons tout d'abord que l'échantillonnage est une opération qui fait passer d'un signal à temps continu $x(t)$ à un signal à temps discret $x[n]$, obtenu en prenant la valeur de $x(t)$ aux instants multiples de la période d'échantillonnage : $x[n] = x(n\,T_e)$. Il est donc facile de passer de $x(t)$ à $x[n]$. Mais à l'inverse, dans quelles conditions peut-on reconstruire $x(t)$ à partir de $x[n]$? En utilisant des signaux sinusoïdaux simples, on peut facilement mettre en évidence que l'échantillonnage fait perdre de l'information :

- Le signal $x(t) = A\,\cos(2\pi(f+F_e)t+\varphi)$, où $F_e = 1/T_e$ est appelée la fréquence d'échantillonnage, conduit après échantillonnage au signal

$$\begin{aligned} x[n] &= A\,\cos(2\pi f T_e n + 2\pi F_e T_e n + \varphi) \\ &= A\,\cos(2\pi f T_e n + \varphi) \end{aligned}$$

  Après échantillonnage, cette sinusoïde de fréquence $f + F_e$ sera donc vue comme une autre sinusoïde de fréquence f. Il en est de même pour toute sinusoïde de fréquence $f + k\,F_e$, où k est un nombre entier positif. Ce phénomène, qui provient simplement de la périodicité des fonctions trigonométriques, est appelé *altération fréquentielle par translation*. Pour éviter que l'échantillonnage ne modifie la fréquence d'une sinusoïde à cause de ce phénomène, il est nécessaire que cette fréquence soit inférieure à F_e.
- Les deux signaux sinusoïdaux

$$z_1(t) = A\,\cos\left(2\pi\left(\frac{F_e}{2} + \Delta f\right)t\right) \quad \text{et} \quad z_2(t) = A\,\cos\left(2\pi\left(\frac{F_e}{2} - \Delta f\right)t\right),$$

  qui correspondent à deux sinusoïdes de même amplitude et de fréquences symétriques par rapport à $F_e/2$, conduisent après échantillonnage au même signal à temps discret $z_1[n] = z_2[n] = (-1)^n\,A\,\cos(2\pi\Delta f T_e n)$. Ce phénomène, qui découle des propriétés élémentaires des fonctions trigonométriques (en particulier des expressions de $\cos(\theta_1 + \theta_2)$ et $\cos(\theta_1 - \theta_2)$), est appelé *altération fréquentielle par symétrie* ou repliement spectral. Pour éviter que l'échantillonnage ne modifie la fréquence d'une sinusoïde à cause de ce second phénomène, il est nécessaire que cette fréquence soit inférieure à $F_e/2$.

En s'appuyant sur ces deux résultats simples, C.E. Shannon a démontré qu'un signal quelconque $x(t)$ peut être échantillonné correctement à la fréquence d'échantillonnage F_e si deux conditions sont vérifées :

- Il existe une fréquence $f_{\max}$ telle que $X(f)$, la transformée de Fourier de $x(t)$, est nulle pour toute fréquence f supérieure à $f_{\max}$. Cela signifie que le signal ne doit pas possèder d'énergie au delà de $f_{\max}$, qui est la plus grande fréquence contenue dans le signal.
- la fréquence d'échantillonnage F_e doit être choisie au moins égale au double de $f_{\max}$: $F_e \geq 2f_{\max}$.

Si ces deux conditions sont vérifiées, il est alors possible de calculer la valeur du signal à temps continu $x(t)$ en tout instant t à partir des valeurs de $x[n] = x(nT_e)$, par une expression appelée *formule d'interpolation de Shannon*.

En conclusion, pour qu'un signal puisse être correctement échantillonné à la fréquence F_e, il donc faut que les deux conditions suivantes soient vérifiées :

- toute *l'information* contenue dans le signal doit être en deçà de $F_e/2$. Si cette condition n'est pas vérifiée, il faut soit changer de fréquence d'échantillonnage soit accepter de perdre de l'information ;
- aucune *énergie* non-négligeable ne doit se trouver au delà de $F_e/2$. Si cette condition n'est pas vérifée, il faut utiliser un filtre passe-bas afin de retirer les composantes du signal situées au delà de $F_e/2$, afin de vérifier le théorème de Shannon. Un tel filtre est appelé un filtre antirepliement.

Exercice I.17: *Le programme ci-dessous (qui correspond au fichier* `Echantillonnage.ch`*), écrit en langage C pour l'environnement de calcul scientifique* `Ch`*, génère un fichier au format* `.dat` *contenant un enregistrement de 20 secondes d'une sinusoïde dont la fréquence varie de $f_i = 400$ Hz à $f_f = 6000$ Hz.*

```
//  Programme permettant de generer la version echantillonnee
//  d'une sinusoide de frequence croissante
//  F. Auger, janvier 2010

#include <stdio.h>
#include <math.h>

int main()
{
 char   NomFichierDat[] = "Echantillonnage.dat" ;
 FILE   *PointeurFichierDat ;
 double MySignal, Fe, Te, t=0.0, Duree, fi, ff, Omegai, alpha ;
 double Amp=0.9, Phi, MyPi ;

 Duree=20.0;           // duree du signal en secondes
 Te=40e-6 ; Fe=1.0/Te; // periode et frequence d'echantillonnage
 fi=400.0; ff=6000.0;  // frequences initiales et finales

 MyPi=M_PI; Omegai = 2*MyPi*fi; alpha=2*MyPi*(ff-fi)/Duree;

 // ouverture du fichier en ecriture
 PointeurFichierDat = fopen(NomFichierDat, "w") ;

 if ( PointeurFichierDat == NULL )
  {printf("Impossible d'ouvrir le fichier %s en ecriture\n",
          NomFichierDat);
  }
 else
  {
   // on precise la frequence d'echantillonnage du signal de sortie
   fprintf(PointeurFichierDat,"; Sample Rate %8.2f\n", Fe);

   // un seul signal = une seule voie = un seul canal
   fprintf(PointeurFichierDat,"; Channels 1\n");

   while (t<Duree)
    {
     Phi=Omegai*t+0.5*alpha*t*t; // phase instantanee
```

```
      MySignal=Amp*cos(Phi);      // calcul du signal
      fprintf(PointeurFichierDat, "%f %f \n", t, MySignal);
      t=t+Te;
     }

   fclose(PointeurFichierDat);   // c'est fini, on ferme
  }

 return 0;
}
```

Le signal sinusoïdal est de la forme $x(t) = 0.9\,\cos(\varphi(t))$, *avec* $\varphi(t) = \omega_i\, t + \alpha\, t^2/2$. *Calculer* $\Omega(t) = \frac{d\varphi}{dt}(t)$ *et* $\frac{\Omega(t)}{2\pi}$. *Comment varie la fréquence en fonction du temps ? Lancer le programme avec une période d'échantillonnage* $T_e = 40$ μs *et écouter le résultat. On pourra transformer le fichier* `.dat` *en fichier* `.wav` *en utilisant la commande*

```
rem conversion d'un fichier .dat en .mp3
rem F. Auger, IUT Saint-Nazaire , dep. MP, jan. 2010

sox Echantillonnage.dat Echantillonnage.wav
```

Relancer ensuite ce programme après avoir fait passer la période d'échantillonnage à 125 μs *et écouter le résultat de nouveau. Faire un spectrogramme du signal obtenu. Que s'est-il passé ? Faire le lien avec le théorème de Shannon.*

Exercice I.18: *Utiliser le programme de l'exercice 17 pour générer un signal sinusoïdal d'une durée de* 20 *secondes, dont la fréquence augmente linéairement de* 400 *à* 8000 *Hz et échantillonné avec une période d'échantillonnage de* 170 μs. *Convertir ensuite le fichier obtenu au format* `.wav`. *Écouter (au casque) et faire enfin un spectrogramme du résultat. Analyser et expliquer le résultat obtenu.*

1.5.2 Ré-échantillonnage

Bien évidemment, SoX ne manipule que des signaux échantillonnés. Tous les signaux enregistrés sur des supports numériques sont à temps discret. L'une des possibilités intéressantes de ce logiciel est le ré-échantillonnage des signaux, c'est à dire le changement de leur fréquence d'échantillonnage. C'est même un des points forts de ce logiciel. Cette opération s'effectue avec la commande `rate`, comme l'illustre la commande ci-dessous,

```
sox SonRigolo.mp3 SonRigolo2.mp3 rate 32k
soxi SonRigolo.mp3  >  SonRigolo_info.txt
soxi SonRigolo2.mp3 >> SonRigolo_info.txt
```

qui fabrique un nouveau signal `SonRigolo2.mp3` en ré-échantillonnant[15] à 32 kHz le signal contenu dans le fichier `SonRigolo.mp3`, dont la fréquence d'échantillonnage d'origine est de 44.1 kHz. Les deux commandes `soxi` qui suivent produisent le fichier ci-dessous, qui indique clairement la fréquence d'échantillonnage (*sample rate*) des deux fichiers :

```
Input File     : 'SonRigolo.mp3'
Channels       : 1
Sample Rate    : 44100
Precision      : 16-bit
Duration       : 00:00:03.58 = 157834 samples = 268.425 CDDA sectors
File Size      : 28.6k
Bit Rate       : 64.0k
Sample Encoding: MPEG audio (layer I, II or III)
```

[15] Dans le standard MP3, les fréquences d'échantillonnage autorisées sont 48 kHz, 44.1 kHz, 32 kHz, 24 kHz, 22050 Hz, 16 kHz, 12 kHz, 11025 Hz et 8 kHz.

```
Input File     : 'SonRigolo2.mp3'
Channels       : 1
Sample Rate    : 32000
Precision      : 16-bit
Duration       : 00:00:03.60 = 115200 samples ~ 270 CDDA sectors
File Size      : 21.6k
Bit Rate       : 48.0k
Sample Encoding: MPEG audio (layer I, II or III)
```

Exercice I.19: *Quelles opérations* SoX *doit-il effectuer pour ré-échantillonner correctement un signal dont la fréquence d'échantillonnage d'origine est égale à 40 kHz à une nouvelle fréquence $F_{e2} = 10$ kHz ? Même question pour une fréquence F_{e2} égale à 16 kHz.*

Exercice I.20:

- *Analyser le signal* `EmissionFranceInter.mp3` *pour déterminer le nombre d'échantillons qu'il contient et sa fréquence d'échantillonnage d'origine.*
- *Extraire de ce signal les 2 premières minutes à l'aide de la commande* `trim` *vue au paragraphe 1.2.3, et enregistrer cet extrait dans un fichier* `.wav`*.*
- *Ré-échantillonner ce nouveau signal en prenant des fréquences d'échantillonnage de plus en plus petites, et faire un tableau indiquant le nombre d'échantillons et la taille du fichier obtenu (en octets) en fonction de la nouvelle fréquence d'échantillonnage. Analyser les résultats obtenus. À partir de quelle fréquence d'échantillonnage le signal devient-il inintelligible ?*

1.6 L'analyse spectrale

L'analyse spectrale, c'est à dire la représentation de la distribution de l'énergie d'un signal dans le domaine fréquentiel, est un outil utilisé dans de très nombreux domaines (acoustique, mécanique vibratoire, génie bio-médical, télécommunications, astronomie, sciences de la terre ...). Basée sur la transformée de Fourier, elle constitue un outil efficace d'analyse des signaux stationnaires. Son emploi sur quelques signaux va être proposé ici pour montrer ses possibilités d'utilisation, en insistant bien sur le fait que le champ d'application de l'analyse spectrale est très loin de se réduire aux seuls exemples présentés.

Un point de départ possible pour présenter cette technique est la transformée de Fourier d'un signal $x(t)$, qui est la fonction de la variable réelle f (appelée fréquence) et à valeurs complexes définie par :

$$X(f) = \int_{-\infty}^{+\infty} x(t)\, e^{-\jmath 2\pi f t}\, dt$$

Comme le signal n'est observé et enregistré que pendant une durée finie L à partir d'un instant initial t_0, on souhaite pouvoir approcher $X(f)$ à l'aide de la transformée de Fourier de la seule partie observée de $x(t)$,

$$\hat{X}(f) = \int_{t_0}^{t_0+L} h(t-t_0)\, x(t)\, e^{-\jmath 2\pi f t}\, dt = \int_0^L h(u)\, x(t_0+u)\, e^{-\jmath 2\pi f u}\, du$$

où $h(t)$ est une fonction de pondération choisie par l'utilisateur, qui permet de préciser l'importance de $x(t)$ dans le calcul de $\hat{X}(f)$. Cette intégrale est ensuite calculée de manière approchée en subdivisant l'intervalle $[0; L]$ en N sous-intervalles de même largeur $T_e = L/N$ et en utilisant une

approximation de chaque intégrale par la méthode des rectangles :

$$\begin{aligned}\hat{X}(f) &= \sum_{n=0}^{N-1} \int_{n\,T_e}^{(n+1)\,T_e} h(u)\,x(t_0+u)\,e^{-\jmath 2\pi f u}\,du \\ &\approx Te \sum_{n=0}^{N-1} h(n\,T_e)\,x(t_0+n\,T_e)\,e^{-\jmath 2\pi f n\,T_e} \\ &= \frac{L}{N} \sum_{n=0}^{N-1} h(n\,T_e)\,x(t_0+n\,T_e)\,e^{-\jmath 2\pi f n\,T_e}\end{aligned}$$

Le choix le plus courant consiste à calculer cette représentation fréquentielle aux fréquences $f = m\,\frac{F_e}{N}$. Ceci conduit à une expression

$$\tilde{X}\left(m\,\frac{F_e}{N}\right) = \frac{L}{N} \sum_{n=0}^{N-1} h(n\,T_e)\,x(t_0+n\,T_e)\,e^{-\jmath 2\pi \frac{mn}{N}}$$

qui est très souvent calculée à l'aide d'un algorithme appelé transformée de Fourier rapide (FFT), et dont on représente graphiquement le module carré $|\tilde{X}(f)|^2$ (ou sa valeur en dB) en fonction de la fréquence. Le nombre d'échantillons N détermine donc la durée de l'enregistrement, $L = N\,T_e$ et le pas de discrétisation de l'axe fréquentiel, $\Delta f = F_e/N$.

Exercice I.21: *On souhaite calculer la transformée de Fourier d'un enregistrement de longueur finie d'un signal sinusoïdal de fréquence f_0, défini par $x(t) = A\,\cos(2\pi f_0\,t + \varphi)$. On supposera que l'enregistrement de ce signal a pour durée L, qu'il débute en $t_0 = -L/2$ et qu'il se termine en $+L/2$. On choisira comme "fenêtre de lissage" du signal une fenêtre rectangulaire,*

$$h(t) = \begin{cases} 1 & \text{si} \quad -L/2 \le t \le +L/2 \\ 0 & \text{sinon} \end{cases},$$

Montrer que la transformée de Fourier de ce segment de sinusoïde s'écrit sous la forme :

$$\hat{X}(f) = \tfrac{1}{2}\,A\,e^{+\jmath\varphi}\,H(f-f_0) + \tfrac{1}{2}\,A\,e^{-\jmath\varphi}\,H(f+f_0).$$

On précisera l'expression de $H(f)$ et on calculera $H(0)$. Si l'amplitude A de la sinusoïde est estimée en mesurant $\frac{2|\hat{X}(f_0)|}{H(0)}$, donner un majorant de la valeur absolue de l'erreur relative commise. On pourra pour cela utiliser l'inégalité triangulaire :

$$|x| - |y| \le |x+y| \le |x| + |y|$$

Conclure sur les applications pratiques du résultat obtenu.

1.6.1 Mise en pratique

Exercice I.22: *À l'aide du fichier de commandes ci-dessous (qui correspond au fichier* `AnalyseSpectrale1.bat`*), on peut générer un signal égal à la somme de deux sinusoïdes de fréquences 400 et 500 Hz. Sur combien d'échantillons l'analyse spectrale de ce signal doit-elle être calculée pour arriver à distinguer les raies des deux fréquences sur le spectre fourni par* `Visual Analyser` *? Examiner l'effet du choix de la fenêtre de pondération et du nombre de moyennes sur la représentation obtenue.*

```
rem generation de signaux elementaires avec SoX
rem F. Auger, IUT Saint-Nazaire, dep. MP, fev. 2010

sox -n AnaSpec1.mp3 synth 3:00 sine 400
sox -n AnaSpec2.mp3 synth 3:00 sine 500

sox -m AnaSpec1.mp3 AnaSpec2.mp3 AnaSpec4.mp3

rem pause
```

Exercice I.23: *En utilisant* SoX *et* `Visual Analyser`*, effectuer une analyse du signal* `SignalStationnaire1.mp3` *ainsi que d'un autre signal appelé* `SignalStationnaire2.mp3`*, afin d'en déterminer le plus précisément possible le contenu. Après avoir réglé les paramètres de votre analyse spectrale, ne pas hésiter à utiliser les fonctions d'enregistrement (*`settings/capture/capture spectrum`*), et en particulier l'enregistrement d'un spectre dans un fichier texte (*`File/Save Spectrum as text`*).*

Fiche d'appréciation

N'hésitez pas à donner votre avis sur ce document, et suggérer des modifications pour améliorer la qualité de cet outil de formation. Vous êtes invité pour cela à utiliser cette page, en y écrivant votre nom et votre commentaire, puis à la remettre dans mon casier dans la salle des enseignants du DUT Mesures Physiques.

Votre nom :

Votre commentaire :