TD2 : Représentations temporelles et spectrales

Les fichiers de ce TD se trouvent à l'adresse suivante :

`http://dept-info.labri.fr/~hanna/ImageSon/TD2`

1 Représentation temporelle

Le fichier `son.c` permet d'ouvrir un fichier son au format `wav`, et de remplir un tableau de `N` réels avec les valeurs des échantillons représentant le signal sonore.

Nous souhaintons visualiser ces échantillons. Pour cela, il existe un module d'interface `C` très pratique, appelé `gnuplot_i.h`.

Dans un premier temps, il faut créer une fonction `plot_init` qui va initialiser l'affichage. Cette fonction définit un contrôleur de type `gnuplot_ctrl *` (dans l'exemple ci-dessous la variable `h`). Ensuite, le style de la courbe peut être défini à l'aide de la fonction `gnuplot_setstyle`.

```
#include "gnuplot_i.h"

static gnuplot_ctrl *h = NULL;

...
h = gnuplot_init ();
gnuplot_setstyle (h, "lines");
```

Exercice 1 : Écrire la fonction `plot_init` d'initialisation de l'affichage.

Ensuite, il faut préciser l'axe des abscisses en définissant un tableau de réels (par exemple `x_axis`). La fonction `gnuplot_plot_xy` prend en paramètres le contrôleur, l'axe des abscisses, l'axe des ordonnées (valeurs à afficher, ici les échantillons), le nombre de valeurs, et enfin le nom du graphe (chaîne de caractères).

```
gnuplot_plot_xy (h, x_axis, data, N, "son");
```

Exercice 2 : Écrire la fonction `plot_temporal` d'affichage des échantillons. Attention : il est nécessaire une fois la courbe affichée à l'écran de suspendre l'exécution du programme (pour avoir le temps de voir le résultat). Vous pourrez faire appel à la fonction `sleep` qui prend en paramètre un entier indiquant un nombre de secondes.

1.1 Test d'affichage des échantillons

Exercice 3 : En utilisant les fonctions précédentes, modifier le programme principal (fonction `main`) pour afficher les N échantillons d'un son, par segments de N échantillons. Ainsi, l'affichage d'un son composé de $10 \times N$ échantillons doit donner lieu à l'affichage successif de 10 graphes. Chaque graphe est alors affiché durant un temps limité (2 secondes par exemple). Ensuite, le graphe suivant est affiché durant le même temps, et ainsi de suite.

Exercice 4 : Modifier l'affichage pour avoir une échelle de temps en secondes.

2 Représentation spectrale

2.1 Transformée de Fourier discrète

La bibliothèque FFTW fournit les types réel (`fftw_real`, équivalent au type standard `double`) et surtout complexe `fftw_complex`, en représentation cartésienne (parties réelle et imaginaire).

Utilisation de la bibliothèque FFTW :

— en-tête :

```
#include <complex.h>
#include <fftw3.h>
```

— compilation :

```
gcc -I/usr/include -c exemple.c -o exemple.o
gcc exemple.o -o exemple -L/usr/lib -lfftw3 -lm
```

La transformée de Fourier discrète (DFT, *Discrete Fourier Transform*) est une opération mathématique définie par :

$$S[m] = \sum_{n=0}^{N-1} s[n]\, e^{-i\frac{2\pi}{N}nm} \quad (0 \leq m < N)$$

Exercice 5 : Écrire la fonction

```
void dft (double s[N], complex S[N])
```

qui calcule la transformée de Fourier discrète d'une trame `s` de N échantillons et range le résultat dans le spectre `S` composé de N nombres complexes.

Exercice 6 : On souhaite afficher le module du spectre. Quelle formule mathématique permet de calculer le module d'un nombre complexe ?

Exercice 7 : La bibliothèque complexe fournit une fonction pour cette opération. Quelle est-elle ?

Pour pouvoir retrouver un nombre complexe à partir de son module (son amplitude), il faut connaître aussi son argument (sa phase). C'est la fonction `carg` de la bibliothèque complexe qui sera alors utilisée.

Exercice 8 : Écrire une fonction

```
void cartesian_to_polar (complex S[N], double amp[N], double phs[N])
```

qui transforme les complexes du spectre `S` de la représentation cartésienne (parties réelles et imaginaires) par défaut en représentation polaire (amplitudes et phases).

Exercice 9 : Écrire un programme qui permet la visualisation du spectre de Fourier à court terme d'un signal. Ce programme doit afficher l'amplitude de spectres de trames successives.

Exercice 10 : L'affichage du module du spectre complet (indices de 0 à $N-1$) est-il pertinent ? Pourquoi ?

3 Transformée de Fourier rapide

La transformée de Fourier rapide (FFT, *Fast Fourier Transform*) réduit le nombre d'opérations nécessaires grâce à une structure algorithmique particulière mais aussi parfois en pré-calculant notamment les valeurs des exponentielles complexes. C'est le rôle du plan qui doit donc être mis en place une seule fois :

```
static fftw_plan plan;
fftw_complex data_in[N];
fftw_complex data_out[N];
 ...
plan = fftw_plan_dft_1d(N, data_in, data_out, FFTW_FORWARD, FFTW_ESTIMATE);
```

Ce plan est ainsi disponible pour un nombre quelconque de transformées (autant de fois que nécessaire) :

```
#define N 1024
 ...
fftw_real s[N];  /* domaine temporel */
 ...
fftw_complex data_in[N];
fftw_complex data_out[N];
 ...
for (i=0; i<N; i++)
  {
    data_in[i] = s[i];
  }
 ...
fftw_execute (plan);
```

Le plan doit ensuite être détruit :

```
fftw_destroy_plan (plan);
```

Exercice 11 : Utiliser la bibliothèque FFTW (`www.fftw.org`) pour remplacer la transformée de Fourier discrète – et lente – précédente par la transformée de Fourier rapide (FFT) : écrire la fonction `fft` correspondante.

Exercice 12 : Du fait du fonctionnement par l'intermédiaire du plan, vous aurez besoin d'une fonction d'initialisation `fft_init` (à appeler avant tout appel à la fonction `fft`) et d'une fonction `fft_exit` (à appeler avant de quitter votre programme). Écrire les fonctions `fft_init` et `fft_exit`.

Conseil : En cas de problème avec la transformée de Fourier, il peut être utile de la tester sur un signal simple, le plus simple étant une sinusoïde d'amplitude a et de fréquence f, du type :

$$s[n] = a\cos(2\pi fn/F_e)$$

Remarque : la constante π est disponible dans la bibliothèque mathématique...

4 Performances

On souhaite estimer le gain de performance apporté par l'utilisation de la FFT par rapport à l'utilisation de la DFT. Pour cela, on doit disposer d'une estimation du temps de calcul nécessaire à chacune des deux solutions grâce à la fonction standard `clock` :

```
#include <stdio.h>

#include <time.h>

...

clock_t t1, t2;
double delta_t;

t1 = clock ();

/* le code à chronométrer */
...

t2 = clock ();
```

```
delta_t = t2 - t1;

printf ("le temps écoulé est de %f secondes\n", delta_t / CLOCKS_PER_SEC);
```

Exercice 13 : Calculer les temps de calcul nécessaires à la DFT et la FFT lors de l'exécution du programme pour une trame du fichier `toms.aiff`. Les résultats sont-ils exploitables ? Pourquoi ?

Exercice 14 : Quelle est la complexité théorique des deux algorithmes testés ?

Exercice 15 : Mesurer le temps moyen nécessaire à la DFT lors de l'analyse du fichier `toms.aiff` complet.

Exercice 16 : Quel est alors le temps théorique nécessaire à la FFT pour ce même fichier ?

Exercice 17 : Mesurer le temps moyen nécessaire à la FFT pour analyser ce fichier.
Cette mesure correspond-elle au temps théorique attendu ? Pourquoi ?