Lecture d'un fichier audio

-

Création d'un spectrogramme.

Sommaire

1 : Introduction

Pour pouvoir traiter ce sujet, il nous faut d'abord rappeler ce qu'est un son d'un point de vue physique. Un son est une onde produite par la vibration mécanique d'un support fluide ou solide et propagé grâce à l'élasticité du milieu environnant sous forme d'ondes longitudinales. Dans l'air, le son se propage sous forme d'une variation de pression créée par la source sonore.

Cette source sonore même complexe peut être étudiée simplement par une analyse spectrale, c'est à dire la décomposition de cette onde en une somme d'ondes simples (sinusoïdales) à l'aide d'une Transformée de Fourier (TF).

Le spectrogramme est un diagramme associant à chaque instant T d'un signal, son spectre de fréquences. Les premiers spectrogrammes ou « sonagrammes » étaient construits de manière analogique et mécanique.

De nos jours l'outil informatique ayant bien évolué, un ordinateur domestique possède une puissance de calcul suffisante pour réaliser un spectrogramme rapidement. Dans ce projet, nous réaliserons un programme capable d'extraire les données utiles d'un son (contenu dans un fichier audio), puis d'effectuer les calculs nécessaires pour la construction d'un spectrogramme. Enfin ce même programme synthétisera l'image correspondant au spectrogramme calculé.

2 : Bien organiser son travail

Pour traiter ce sujet, nous avons procédé ainsi :

- Recherches internet sur les fichiers audio et les spectrogrammes
- Lecture d'un fichier audio dans un programme [C]
- Construire une TF dans un programme [C]
- Analyse de la TF et vérification des résultats attendus
- Amélioration du programme : effectuer un grand nombre de TF successives
- Création du spectrogramme dans une fenêtre graphique
- Amélioration de l'ergonomie du programme
- Ajout de fonctionnalités supplémentaires permettant le traitement des données sonores représentées

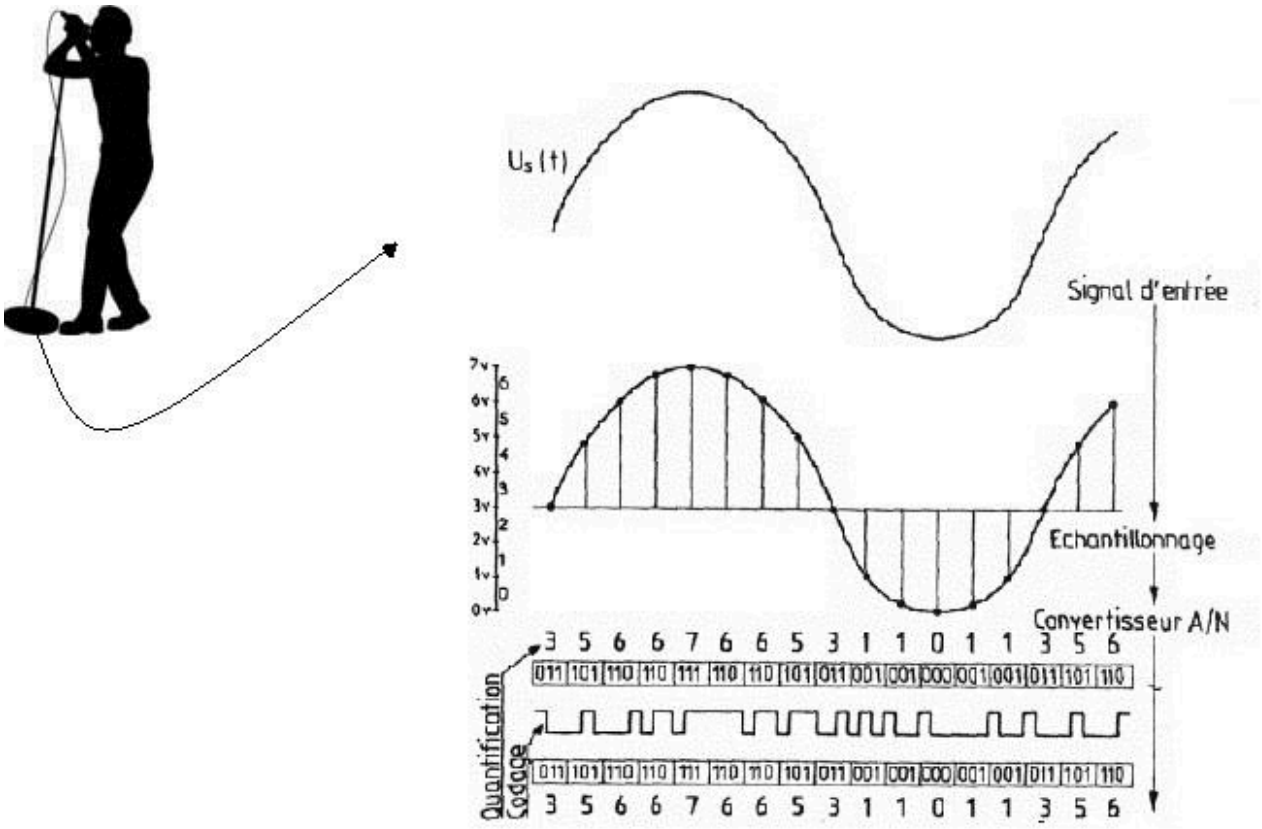
3 : Lecture d'un fichier audio dans un programme [C]

Informatisation d'un fichier audio :
Pour enregistrer une onde sonore sur un support numérique, il existe de nombreuses étapes qui peuvent se résumer à :

- Acquisition analogique par un micro qui transforme l'onde sonore en un signal électrique proportionnel
- Échantillonnage de ce signal électrique
- Conversion analogique/numérique des points échantillonnés
- Enregistrement des données numériques dans un fichier audio

Qu'est-ce que l'échantillonnage ?

Il consiste en la capture, à intervalles réguliers, de valeurs discrètes du signal que l'on souhaite numériser.

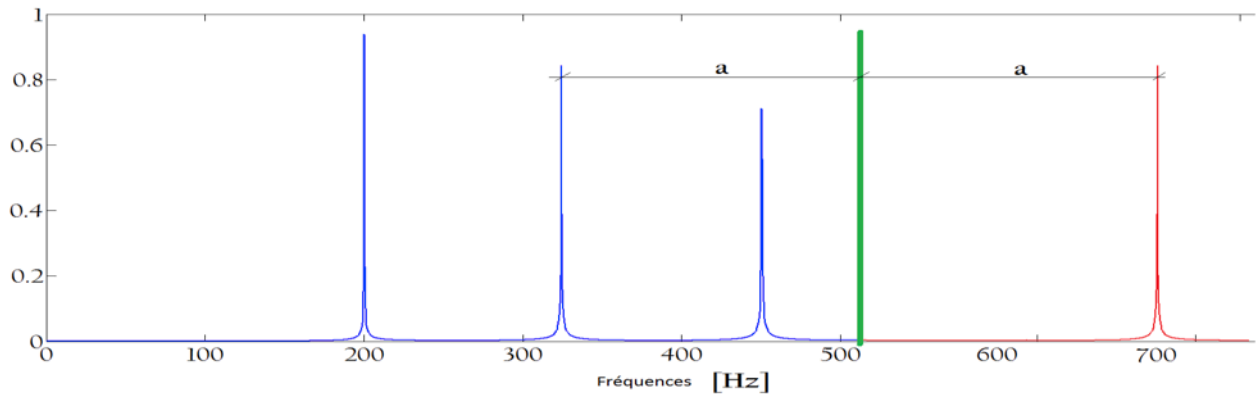


Dans quelles conditions réaliser un échantillonnage ?

Le critère de Nquist-Shannon implique que la fréquence d'échantillonnage soit au minimum supérieure au double de la fréquence maximale contenue dans le signal à échantillonner. La fréquence d'échantillonnage correspond au nombre de valeurs capturées en une seconde.

Si le critère de Nquist-Shannon n'est pas respecté, on observe alors un recouvrement (ou « aliasing ») du spectre du signal étudié. Les hautes fréquences supérieures à la moitié de la fréquence d'échantillonnage vont alors être déplacées dans les basses fréquences.

Sur le schéma suivant, nous avons en vert la fréquence maximale pouvant être échantillonnée. On remarque alors qu'il y a un déplacement du pic rouge vers les basses fréquences de manière symétrique par rapport à la fréquence maximale pouvant être correctement échantillonnée.



Pour éviter ce phénomène, il existe deux solutions :

- Utiliser une fréquence d'échantillonnage supérieure.
- Utiliser un filtre passe-bas supprimant les fréquences non gérées par l'échantillonnage.

Quel format de fichier audio utiliser ?

Il existe un très grand nombre de formats audio utilisés dont certains sont des formats propriétaires et d'autres libres de droits. Parmi ces formats, on distingue 2 grandes familles :

- Les formats sans perte de données (WAV, AIFF, FLAC ...).
- Les formats avec compression de données. Pour un espace de stockage réduit, le signal audio est simplifié selon des algorithmes de compression avec pour coût une faible perte de qualité par rapport au signal initial (MP3, AAC...).

Pour une intégration simplifiée dans notre programme, nous avons choisi le format « WAVE » pour son entête basique et son absence de compression.

Structure d'un fichier « WAVE » :

On distingue 2 parties dans un fichier WAVE : l'entête et la partie DATA.

L'entête mesure 56 octets et contient les informations sur le format utilisé pour encoder le son (nombre d'octets par point échantillonné, nombre de canaux, fréquence d'échantillonnage, longueur du fichier...).

La partie DATA commence à l'octet 57 et contient toutes les valeurs des points échantillonnés. (cf. annexes)

Comment ouvrir un fichier WAVE dans un programme [C] ?

- Il nous faut créer une structure (WAVFILE dans notre programme) rigoureusement identique à celle de l'entête d'un fichier WAVE.

```
struct wavfile //définit la structure de l entete d un wave
{
char id[4]; // doit contenir "RIFF"
int totallength; // taille totale du fichier moins 8 octets
char wavefmt[8]; // doit etre "WAVEfmt "
int format; // 16 pour le format PCM
short pcm; // 1 for PCM format
short channels; // nombre de channels
int frequency; // frequence d echantillonage
int bytes_per_second; // nombre de octets par secondes
short bytes_by_capture; // nombre de bytes par echantillon
short bits_per_sample; // nombre de bit par echantillon.
char data[4]; // doit contenir "data"
int bytes_in_data; // nombre de bytes de la partie data
};
```

- Créer une variable HEADER qui prend la forme de cette structure.

```
struct wavfile header;
```

- Charger le fichier dans le programme avec la commande « fopen ».

```
FILE *wav = fopen(fichieraudio,"rb");
```

Dans le programme notre fichier audio sera appelé WAV. On remarque aussi l'argument « rb » (read binary) dans la commande « fopen ». Le « b » est ajouté pour permettre au compilateur [C] de Windows de faire la distinction entre un fichier « TXT » et un fichier de données brutes.

- Lecture de l'entête et initialisation du Header

```
fread(&header,sizeof(header),1,wav)
```

« fread » est une fonction permettant la lecture de données brutes (binaires) d'un fichier dans un programme [C]. Il y a 4 paramètres à indiquer :

- La variable où les données seront enregistrées (tableau, structure ou variable simple)
- Taille des données à lire (en nombre d'octets)
- Nombre de fois que doit être effectuée la lecture
- Fichier à lire

Dans notre cas le programme va lire une fois le fichier WAV. Sa lecture se fera sur la taille du HEADER (correspondant à celle de l'entête) et les données lues seront enregistrées dans le HEADER.

Le HEADER ainsi rempli contient toutes les caractéristiques de notre fichier WAVE, notamment la fréquence d'échantillonnage (header.frequency), le nombre de canaux (header.channels), le nombre d'octets par échantillon (header.bytes_by_capture), le nombre d'octets dans la partie DATA (header.bytes_in_data). Les fichiers WAVE utilisés seront des mono (header.channels = 1). On peut alors déterminer le nombre d'échantillons contenus dans la partie DATA (= header.bytes_in_data / header.bytes_by_capture).

Nous verrons plus tard que le nombre de points à analyser doit être égal à une puissance de 2. Nous allons alors déterminer la puissance de 2 supérieure au nombre d'échantillons.

```
nbech=(header.bytes_in_data/header.bytes_by_capture);
printf ("le fichier audio contient %d echantillons\n",nbech);
while (nbech>taille)
{
taille=taille*2;
puissance=puissance+1;
}
printf ("nombre de points traites pour le spectrogramme : 2^%d=%d\n",puissance,taille);
```

Pour la suite de notre programme, « taille » correspondra à la puissance de 2 supérieure au nombre total d'échantillons du fichier WAVE lu.

Création d'un tableau dynamique qui contiendra les échantillons.

On utilisera un tableau de type alloc, car la taille du tableau ne pourra être déterminée qu'après exécution du programme.

```
double **tab=NULL;
tab=malloc( (taille) * sizeof(double));
if (tab == NULL)
{
exit(0);
}
for(i=0;i<(taille);i++)
{
tab[i]=malloc( 2 * sizeof(double));
if (tab[i] == NULL)
{
exit(0);
}
```

Ce tableau est une matrice à 2 dimensions de 2 colonnes et « taille » lignes.

Il s'agit d'un tableau de nombres complexes, car nous verrons par la suite que l'algorithme de TF utilisé, nécessite un tableau de cette forme.

- Enfin nous allons remplir ce tableau de nouveau avec la fonction « fread ».

```
for(i=0;i<taille;i++)
{
tab[i][0]=0;
tab[i][1]=0;
}
// [...]
while( fread(&value,(header.bits_per_sample)/8,1,wav) )
{
tab[i][0]=value;
}
```

Tant que la lecture du fichier est possible, la boucle « while » s'exécute.

4 : Construire une Transformée de Fourier dans un programme [C]

Rappel physique d'une TF à temps continu :
La TF à temps continu d'un signal x(t) est défini par :

$$\hat{x}(\mu) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi\mu t}\, dt$$

Lorsque le signal x(t) est discrétisé, c'est-à-dire que l'on dispose d'un échantillon discret et fini de ce signal, l'intégrale qui définit la TF ci-dessus peut être approchée par une somme discrète et finie (Discret Fourier Transform) :

$$\hat{x}_m = \sum_{m=1}^{N} x_k\, e^{-i2\pi \frac{(k-1)(m-1)}{N}}$$

Lorsque N est une puissance de 2 il existe un algorithme de calcul rapide pour effectuer cette somme, appelée « Fast Fourier Transform » ou « FFT ».

Pour notre programme nous avons utilisé un algorithme de FFT appelé Cooley-Tuckey.

Le principe de cet algorithme est de décomposer le calcul d'une DFT de taille N en une somme de N1 DFTs de taille N2. Chacune des N1 DFTs peut à son tour être décomposée selon le même principe - éventuellement avec des N1 et N2 différents – ce qui donne une expression récursive. De plus, la somme des N1 DFTs constitue elle aussi une DFT de taille N1, ce qui revient à dire que l'on transforme une DFT unidimensionnelle de taille N en une DFT bidimensionnelle de taille N1*N2.

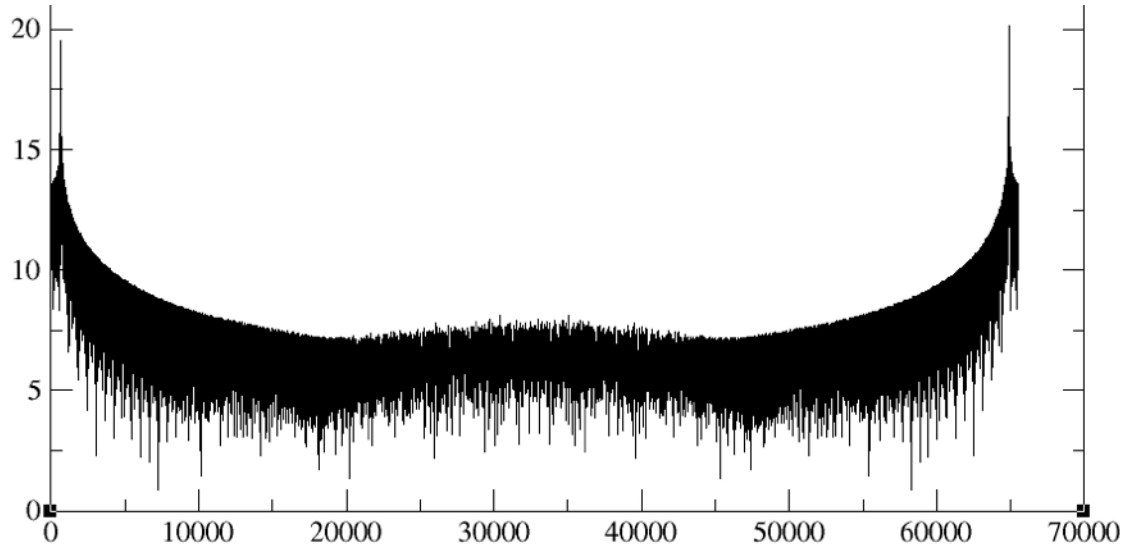
C'est la technique de « diviser pour mieux régner ».

De plus l'algorithme que l'on utilisera dans notre programme fait appel à l'opérateur « butterfly » ainsi qu'à la technique du « bit réversal » pour simplifier les calculs informatiques en se basant sur la récursivité de ces calculs.

5 : Analyse des résultats après FFT

Pour tester notre programme, nous avons généré une onde sinusoïdale de fréquence 440Hz à l'aide du logiciel Audacity, que l'on exporte ensuite en fichier WAVE. On exécute notre programme avec ce fichier en récupérant le résultat de la FFT par la copie dans le tableau malloc « tabfft » de même dimension que le tableau « tab » dans un fichier DATA à l'aide de la fonction « fprintf ».

Dans ce fichier est inscrit le logarithme du module des points (complexes) générés par l'algorithme de la FFT. On effectue une représentation dans xmgrace :



L'axe des abscisses n'est pas à l'échelle (les points sont simplement numérotés). On observe bien un pic dans les basses fréquences correspondant au « La » (440Hz). Il nous reste alors à définir le vecteur des fréquences pour bien renseigner la légende la TF, ainsi qu'à éliminer la seconde moitié des points (ici de 32500 à 65000), car la TF est symétrique et seule la moitié nous intéresse.

On considère alors les « taille/2 » premiers points. Le point numéro « n » aura une fréquence de :

$$\nu = n \times \frac{frequence\ max}{nombre\ de\ points} = n \times \frac{\frac{frequence\ d\ échantillonage}{2}}{\frac{taille}{2}} = n \times \frac{fréquence\ d'echantillonnage}{taille}$$

6 : Amélioration du programme :

La FFT fonctionnant, nous devons alors modifier notre programme pour qu'il effectue une succession de FFT afin de construire notre spectrogramme. Il y a deux façons de procéder :

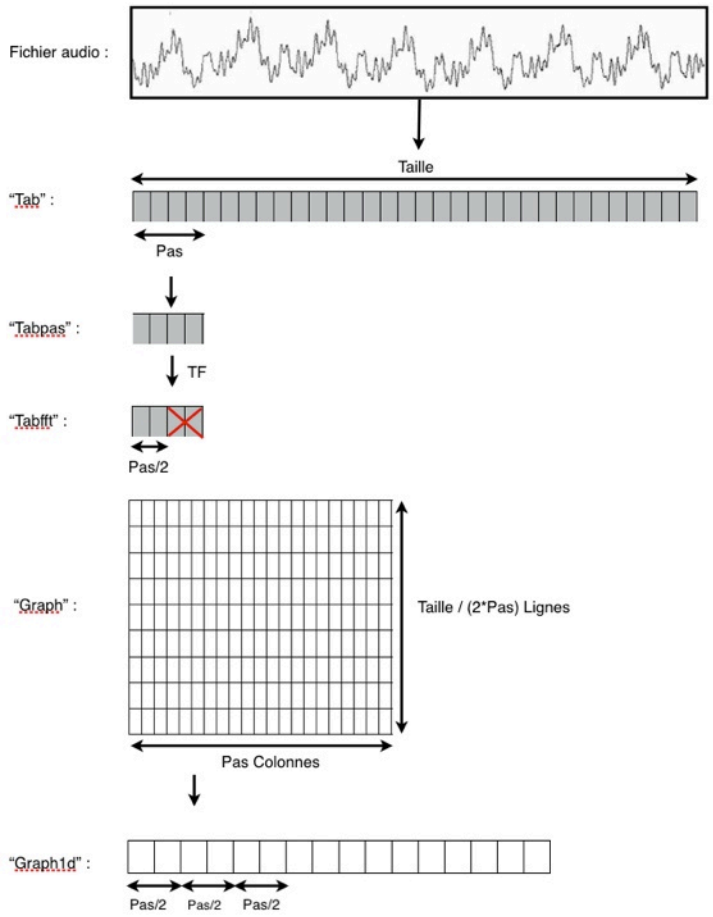
- Par chevauchement : On considère une fenêtre temporelle dt que l'on déplace progressivement le long de l'onde temporelle où on effectue une TF à chaque pas.
- Par segmentation : on découpe l'onde temporelle en N segments de même dimension où N est une puissance de 2.

Ces deux méthodes sont à peu près équivalentes lorsque l'on travaille avec un grand nombre d'échantillons et une petite fenêtre (ou segment). Pour notre programme, nous avons choisi la méthode de segmentation.

On définit la variable « pas » qui est égale au nombre de segments du fichier audio. On crée un nouveau tableau malloc appelé « tabpas » qui contiendra chaque segment du tableau « tab ». On réalise ensuite une FFT du tableau « tabpas » qui remplit en résultat le tableau « tabfft ». On crée alors un nouveau tableau malloc, appelé « graph », de « pas » colonnes et « Taille / 2*pas » lignes. À l'aide d'une double « boucle for », on enregistre dans chaque colonne de « graphe », la moitié de la FFT de chaque segment.

CF : shéma tab

```
for (j=0;j<pas;j++)
{
for(i=0;i<(taille/pas);i++)
{
tabpas[i][0]=tab[i+j*(taille/pas)][0];
}
tf_cooley_tukey(r,n,tabpas,tabfft);
for (i=0;i<((taille/pas)/2);i++)
{
graph[i][j]=log(sqrt(tabfft[i][0]*tabfft[i]
[0]+tabfft[i][1]*tabfft[i][1]));
}
}
```



7 : Création du spectrogramme dans une fenêtre graphique

Choix de Pgplot :
Pgplot est une librairie en Fortran, adapté au langage [C], permettant d'effectuer des représentations graphiques de données traitées par un programme.

Dans notre cas nous souhaitons représenter un spectrogramme, c'est-à-dire un graphique à trois dimensions qui sont : le temps, les fréquences et une amplitude. Il convient alors de construire une image (deux dimensions) comportant des points ou régions de cet espace, colorés sur une échelle de gris ou de couleurs (troisième dimension). Ainsi l'axe horizontal représentera une donnée temporelle, celui vertical, une donnée fréquentielle, et enfin l'amplitude sera codée par un dégradé de gris ou de couleurs.

Quelles sont les commandes adaptées à notre cas ?
Plusieurs commandes nous seront nécessaires pour utiliser Pgplot au sein du programme afin d'afficher correctement notre spectrogramme.

La liste détaillée de ces commandes est disponible en annexe.

Conversion de notre matrice en un vecteur 1D
Pgplot ne gère pas les matrices ou tableaux à deux dimensions, mais seulement les tableaux à une dimension ou vecteur. Pour utiliser la commande cpgimag il nous faut alors convertir notre tableau « graph » en « graphıd ».

```
for (i=0;i<((taille/pas)/2);i++)
{
for (j=0;j<pas;j++)
{
graph1d[i*pas+j]=(float)graph[i][j];
}
```

Affichage du spectrogramme
Apres avoir converti notre matrice, il nous reste à définir les autres paramètres de la fonction « cpgimag ».

```
jdim=1.*taille/(2*pas);
idim=pas;
imin=1;
imax=idim;
jmin=1;
jmax=jdim;
xmin=0;
ymin=0;
ymax=taille*(1./header.frequency);
xmax=(1.*header.frequency/2);
tr[0]=ymin;
tr[1]=(ymax-ymin)/idim;
tr[2]=0;
tr[3]=xmin;
tr[4]=0;
tr[5]=(xmax-xmin)/jdim;
printf("min : %f max: %f\n",a1,a2);
cpgopen("/xw");
cpgenv(ymin,ymax,xmin,xmax,0,0);
cpgimag(graph1d, idim, jdim, 1, idim, 1, jdim, a1, a2, tr);
cpglab("temps","frequences","spectrogramme");
```

Amélioration de l'ergonomie du programme :

Le programme fonctionne, mais il reste brut et difficile d'utilisation. Nous avons donc choisi de rajouter et d'améliorer son interface afin de pouvoir analyser des spectrogrammes générés.

Dans la console étaient affichées des informations utiles au développement du programme (valeurs intermédiaires, numéros de la boucle...) qui ne sont plus nécessaires au programme fini. Nous avons donc retiré ces lignes et affiné la présentation des informations importantes pour l'utilisation du programme.

Ensuite pour un gain de temps considérable, nous avons inclus dans la console des demandes de saisies pour configurer le programme afin d'afficher un spectrogramme « sur mesure ». Cette étape est importante, car auparavant il était nécessaire de recompiler le programme pour changer ces paramètres. Nous préférons cette méthode à celle de l'utilisation d'un fichier « config », car à chaque paramètre entré, le programme pourra vérifier en temps réel la conformité de ce dernier.

Nous avons inséré un menu dans la console après l'affichage du spectrogramme permettant à l'utilisateur de choisir une des fonctionnalités proposées pour analyser ce dernier. Nous ajoutons alors, de nouvelles fonctionnalités au programme pour une meilleure interprétation du résultat, tel que :

- La possibilité d'effectuer un zoom sur une région du spectrogramme

L'utilisateur sélectionne deux points pour effectuer le zoom sur le spectrogramme. Le tableau graph1d est alors recalculé pour créer un nouveau spectrogramme correspondant au zoom désiré.

- L'analyse d'un point particulier du spectrogramme

Sur ce « zoom » pourra alors être sélectionné un point dont le programme donnera ses coordonnées temporelle et fréquentielle.

- Un utilitaire permettant de retrouver une note de musique à partir de sa fréquence

L'utilisateur entre une fréquence et le programme retourne le nom de la note la plus proche ainsi que sa fréquence correspondante

- Le calcul de la vitesse d'un objet bruyant par effet doppler

L'utilisateur doit saisir la fréquence de l'objet durant son approche et son éloignement pour que le programme en détermine sa vitesse.

8 : Conclusion

Au cours de ce projet, nous avons appris à manipuler un fichier binaire (audio) dans un programme [C]. Nous avons également compris comment réaliser une Transformée de Fourier en [C]. Nous nous sommes familiarisés à l'utilisation de Pgplot permettant des représentations graphiques commandées par un programme. Ainsi nous avons pu réaliser un programme capable de créer le spectrogramme d'un fichier audio proposant aussi quelques outils pour son analyse.

Ce programme présente un fort intérêt pour la communauté scientifique. En effet, de part le partage de son code source, notre programme pourra servir de base à l'élaboration de logiciels plus complexes qui font appel au traitement numérique du son :

- La création d'un filtre, pour atténuer une gamme de fréquences du fichier audio, dans le but de générer un nouveau fichier audio filtré
- Le support d'autres formats audio non compressés (AIFF, FLAC...)
- Le support de fichier audio multicanaux (stéréo, 5.1...)
- Une représentation en 3D du spectrogramme
- La création d'une partition de musique par analyse « note par note » d'un fichier audio
- Le calcul de la vitesse de rotation d'un moteur thermique ou électrique
- La reconnaissance vocale
- La création d'ambiance sonore (reproduire le son d'une cathédrale)

9 : Remerciements

Nous remercions M. Franck CELESTINI de nous avoir encadré, épaulé et guidé.
Nous remercions M. Eric ARESTIDI pour son aide, sa disponibilité et pour nous avoir éclairé dans l'utilisation de PGPLOT.

10 : Bibliographie

Acquisition d'un fichier WAVE dans un programme [C] :

http://yannesposito.com/Scratch/fr/blog/2010-10-14-Fun-with-wav/

Structure de l'entête d'un fichier WAVE :

http://col2000.free.fr/vocal/formawav.htm

http://fr.wikipedia.org/wiki/WAVEform_audio_format

Définitions de spectrogramme et exemples :

http://fr.wikipedia.org/wiki/Spectrogramme

http://fr.wikipedia.org/wiki/Sonogramme

http://audioblog.sonatura.com/?p=255

II : ANNEXES

Structure d'un fichier WAVE :

adr:	HEXA												ASCII
0000	52 49 46 46	30 00 01 00	57 41 56 45	66 6D 74 20	RIFF0...WAVEfmt
0010	10 00 00 00	01 00 01 00	11 2B 00 00	11 2B 00 00	.........+...+..
0020	01 00 08 00	66 61 63 74	04 00 00 00	00 00 01 00	....fact........
0030	64 61 74 61	00 00 01 00	80 80 80 80	80 80 82 84	data............
0040	88 8E A6 88	5E 50 5A 4A	64 76 84 90	A2 9C 9A 9E	....^PZJdv......
0050	8C 7C 78 72	5E 5A 62 60	60 72 7E 82	92 9E 9A 96	.|xr^Zb``r~.....
0060	9A 90 7C 76	70 62 60 66	68 6C 7E 86	8A 92 9C 9A	..|vpb`fhl~.....

etc.

L'entête mesure 56 octets, et s'étend donc de l'adresse 00h à l'adresse 38h.

Les fichiers WAV utilisent le format RIFF ; définition du format RIFF :

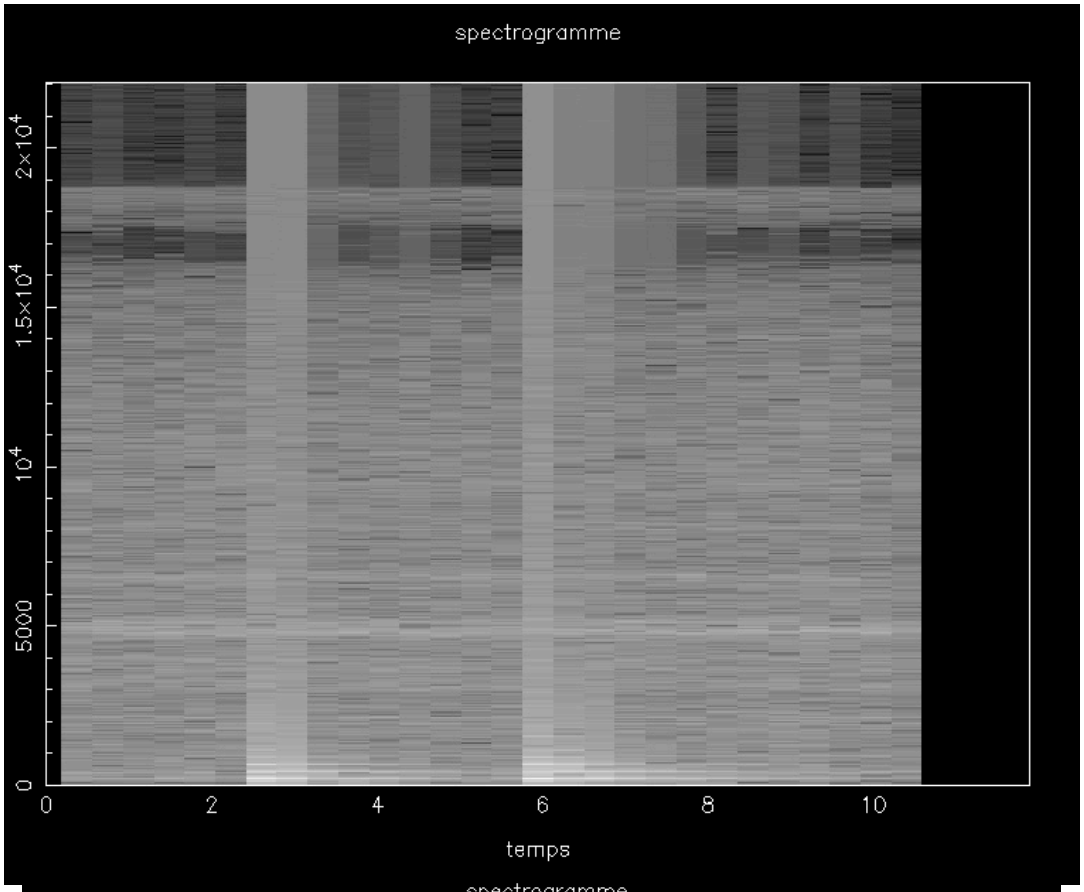
Octets 1 à 4 : Caractères 'RIFF' [52h 49h 46h 46h] identifiant le format.
Octets 5 à 8 : Longueur du groupe de données au format WAV =
[30h.00h.01h.00h] = 00010030h = 65584 octets,
(le reste du fichier, car 65584+8 = 65592 octets).
Octets 9 à 16 : Caractères 'WAVEfmt ' identifiant le format WAV.

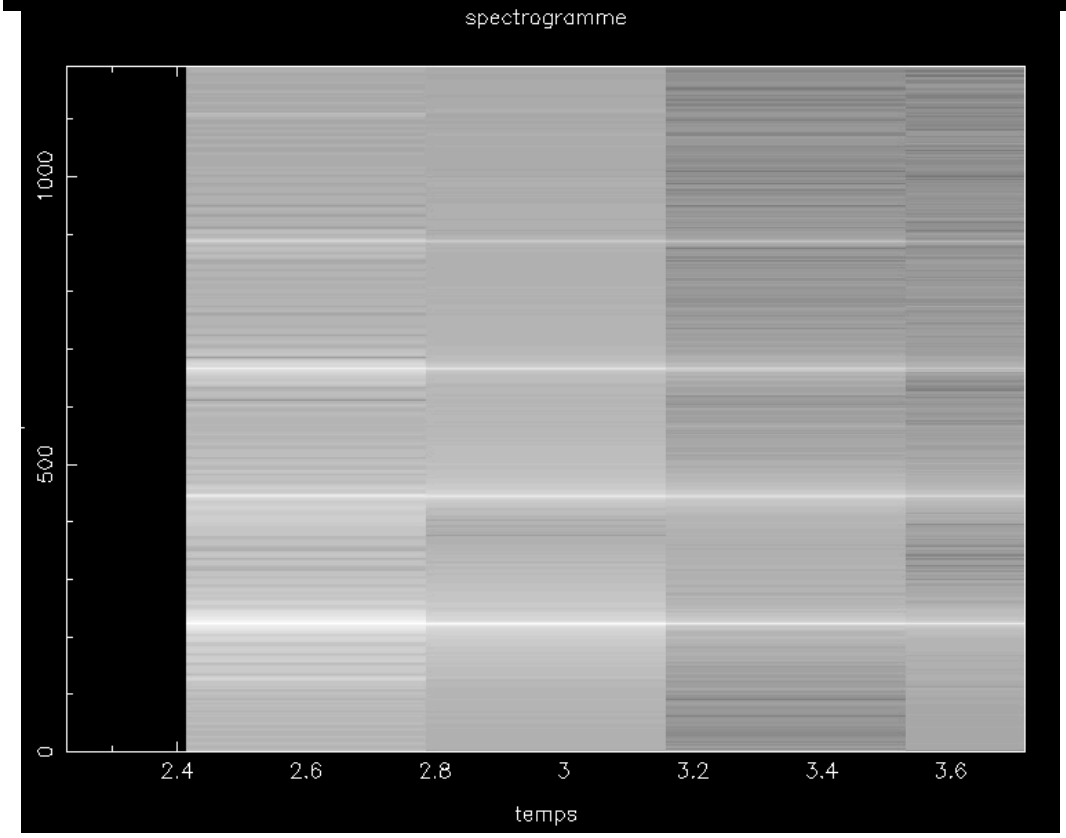
Les octets 17 à 56 définissent les paramètres du format 'WAV' :

Octets 17 à 20 : [10.00.00.00] = 00000010h = 16 = nombre d'octets
utilisés après pour définir le format.
Octets 21 à 22 : [01.00] = 0001h = 1 = numéro de format du fichier
(pas de compression, format PCM classique).
Octets 23 à 24 : [01.00] = 0001h = 1 : nombre de canaux : ici, mono.
Octets 25 à 28 : [11.2B.00.00] = 00002B11h = 11025, fréquence
d'échantillonnage (en Hz), c'est à dire le nombre
d'échantillons pas seconde.
Octets 29 à 32 : [11.2B.00.00] = 00002B11h = 11025, nombres d'octets par
seconde, ce qui revient au même car un échantillon mesure
un octet et l'on est en mono.
Octets 33 à 34 : [01.00] = 0001h = 1 = Produit du nombre de canaux par le
nombre d'octets par échantillon (ici 1x1=1).
Octets 35 à 36 : [08.00] = 0008h = 8 bits par échantillon (valeurs
possibles : 8, 12 ou 16).
Octets 37 à 48 : [66.61.63.74][04.00.00.00][00.00.01.00] = 'fact', 4, 65536.
Champ sur lequel je n'ai pas de précision ; je pense
que 'fact' annonce des informations, que les 4 octets
suivants indiquent la taille de ces infos (4 octets),
et que dans ce cas, l'info est le nombre d'échantillons,
lequel est repris après par 'data', soit 65536.
Octets 49 à 52 : 'data' : annonce l'arrivée des données.
Octets 53 à 56 : [00.00.01.00] = 00010000h = 65536 : taille des données.

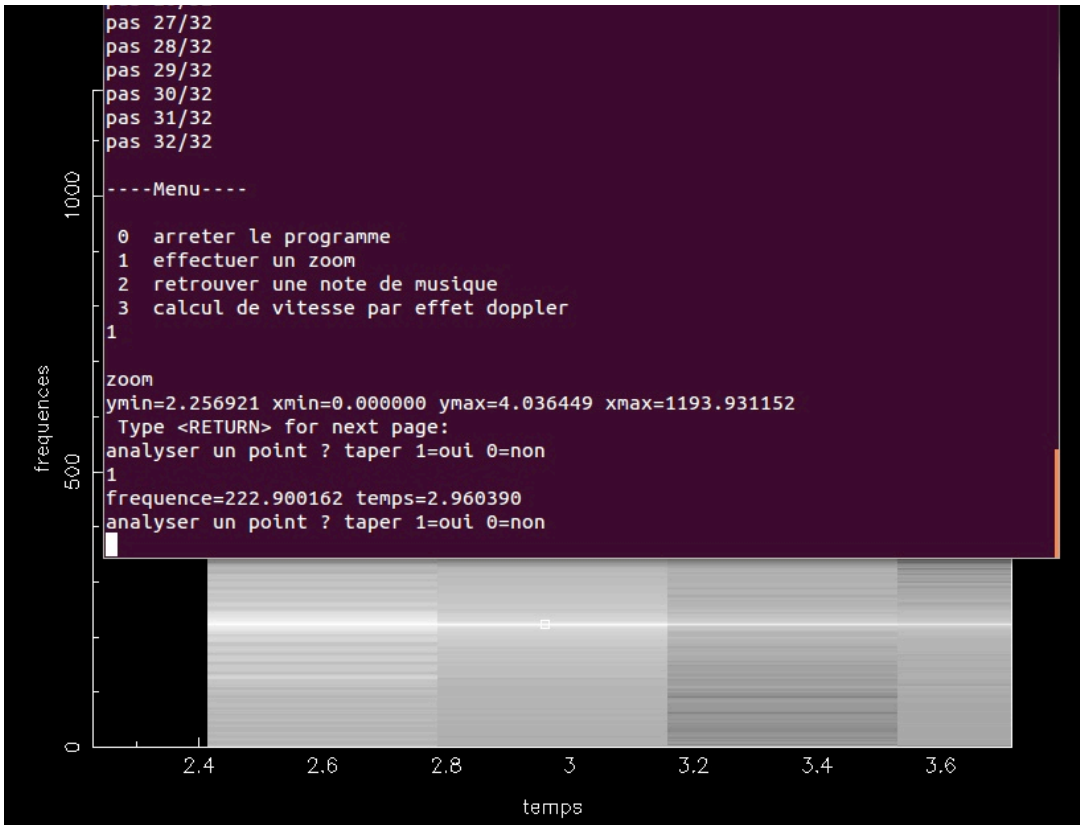
Spectrogramme d'une note «la» 220 Hz jouée à la guitare et le zoom correspondant

pas = 32

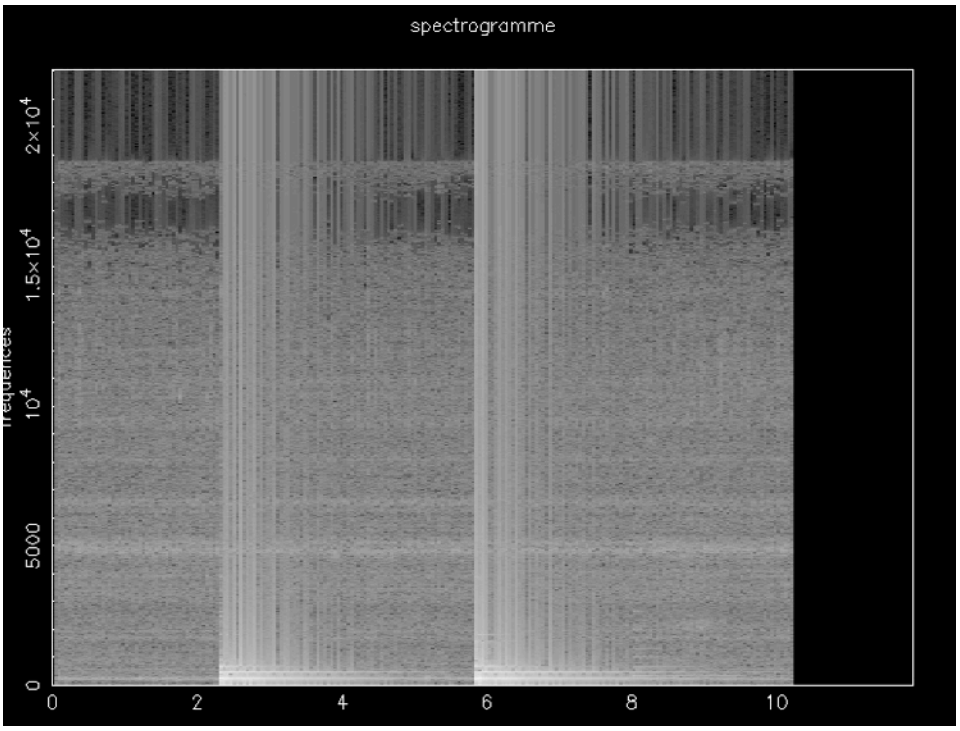




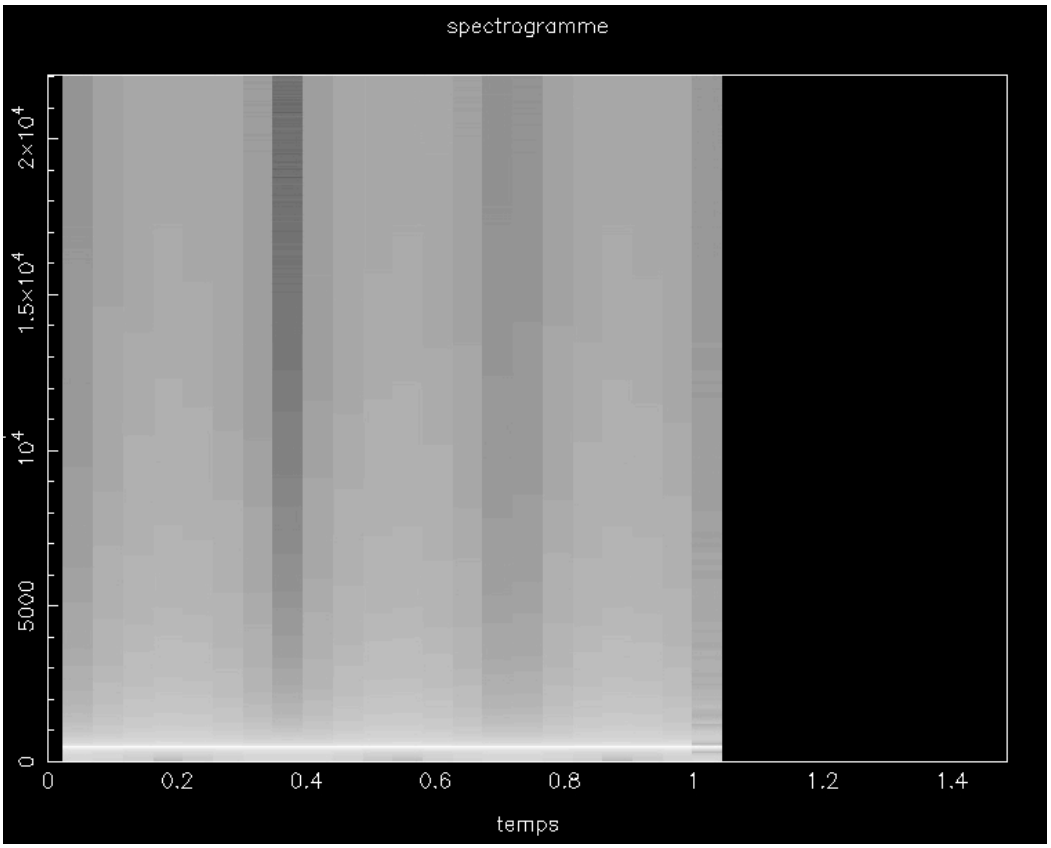
Analyse d'un point vers les 220 Hz



Spectrogramme d'une note «la» 220 Hz jouée à la guitare ; pas=256



«La» pur 440 Hz ; pas=16



Commandes Pgplot :

- cpgopen(char* dev) : démarre PgPlot, l'argument « dev » peut valoir :
  "/xw" : le graphe est tracé sur l'écran (X Windows)
  "/ps" : le graphe est envoyé dans le fichier PostScript pgplot.ps

- cpgimag(z, idim, jdim, i1, i2, j1, j2, a1, a2, tr) : Affiche une image (ou fonction de deux variables) ou une partie d'une image en niveaux de gris ou en fausses couleurs.
  z : Tableau contenant l'image, de dimension idim*jdim. La structure du tableau z
  idim : Nombre de lignes de l'image
  jdim : Nombre de colonnes de l'image
  i1, i2, j1, j2 : Limites de la sous image à représenter
  a1, a2 : Seuils bas et haut de la représentation. Toute valeur de la fonction inférieure à a1 sera représentée comme si elle était égale à a1, et toute valeur de la fonction supérieure à a2 sera représentée comme si elle était égale à a2.
  tr : Tableau de passage permettant de calculer les valeurs (x,y) en coordonnées utilisateur d'un point de l'image se trouvant à la ligne i et la colonne j.

- cpgenv(xmin, xmax, ymin, ymax, just, axis) : Établit le maximum et le minimum du graphe en abscisse et en ordonnée, calcule la correspondance entre les pixels sur l'écran/imprimante et les coordonnées utilisateur, trace un cadre autour du graphe et les axes.
  xmin, xmax : Abcisses maximum et minimum du graphe (en coordonnées utilisateur).
  ymin, ymax : Ordonnées maximum et minimum du graphe.
  just : Si just=1, le repère du graphe est orthonormé. Pour toute autre valeur de just, il ne l'est pas.
  axis : Agit sur le tracé du cadre et des axes.

- cpglab(char* xlabel, char* ylabel, char* titre) : Met un label sur les deux axes et un titre sur le graphe.
  xlabel : Label de l'axe des abscisses
  ylabel : Label de l'axe des ordonnées
  titre : Titre du graphe

- cpgncur

- cpgask(int n) : Par défaut, les fonctions cpgenv et cpgend demandent à l'utilisateur de taper <RETURN> avant de passer à la fenêtre suivante ou de fermer PgPlot. L'appel de cpgask avant ces fonctions permet de modifier leur comportement :
  Si n=1 : on demande à l'utilisateur de taper <RETURN>
  Si n=0 : on ne demande pas à l'utilisateur de taper <RETURN>

- cpgend() : Ferme PgPlot.

- cpgrect(float x1, float x2, float y1, float y2) : Dessine un rectangle. Le coin en bas à gauche est aux coordonnées (x1,y1), le coin en haut à droite aux coordonnées (x2,y2).
  On peut changer la couleur du rectangle par un appel préalable à la fonction cpgsci. De même un appel préalable cpgsfs permettra de dessiner un rectangle plein, vide ou hachuré.

- cpgsci(int color) : Change la couleur du pinceau. Le nombre color indique la couleur, de 0 à 15.

- cpgsch(float taille) : Change la taille des caractères. taille=1 est la valeur par défaut, taille=2 fait des caractères 2 fois plus gros.

- cpgsfs (int style) : Change le style de tracé de l'intérieur des polygones. L'argument style peut valoir :
  1 : les polygones sont pleins (valeur par défaut)
  2 : les polygones sont creux
  3 : les polygones sont hachurés
  4 : les polygones sont hachurés dans les 2 sens

Code source du programme :

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>
#define MAX(a,b) ( (a)>(b) ? (a) : (b))
#define MIN(a,b) ( (a)<(b) ? (a) : (b))
#define PI 3.14159265358979323385
#include <cpgplot.h>



void notemusique (float note)
{
    char nomnote[10],nomnoteprec[10];
    float frequencenote=0, frequencenoteprec=0;
    int i;
    FILE *diconote=fopen("diconote.dat","r");
    while (note>frequencenote)
    {
        frequencenoteprec=frequencenote;
        fscanf (diconote,"%s %f",nomnote, &frequencenote);

    }
    if ((frequencenote-note)>(note-frequencenoteprec))
    {
        void rewind(FILE* diconote);
        note=frequencenoteprec;
        frequencenote=0;
        frequencenoteprec=0;
        fclose(diconote);
        FILE *diconote2=fopen("diconote.dat","r");
        while (note>frequencenote)
        {
            fscanf (diconote2,"%s %f",nomnote, &frequencenote);
        }
        printf ("la note est : %s de frequence=%f\n",nomnote,frequencenote);
    }
    else
    {
        printf ("la note est : %s de frequence=%f\n",nomnote,frequencenote);
    }
}



/* Cette fonction effectue le "bit reversal" sur un indice
donnÈ
@param: l'indice sur lequel s'effectue le "bit reversal"
@param: le nombre de bit dont on a besoin pour Ècrire chaque
indice
@return: l'indice obtenu aprËs le "bit reversal"
*/
unsigned int tf_reversion(unsigned int indice,int nb)
{
    unsigned int p, k,i,size;
    size=32;
    p=indice >> nb-1;
    for(i=1;i<nb;i++)
        {
        k=indice << i+size-nb;
        k=k >> size-1;
        k=k << i;
        p+=k;
        }
    return(p);
}
```

```
/* Cette fonction calcule la fft
@param: logarithme en base 2 de N(nombres de points)
@param: valeur utilisÈ dans l'algorithme
@param: tableau des Èchantillons (partie rÈelle + partie
imaginaire)
@param: tableau des calculs (partie rÈelle + partie
imaginaire)
*/
void tf_cooley_tukey(int r,unsigned int n,double **f,double **t)
{
    int k;
    unsigned int ne,ns,n1,n2,i,j,tj;
    double arg,tr1,ti1,tr2,ti2,cr,ci;
    double *tcos=NULL;
    tcos=malloc( n * sizeof(double));
    if (tcos == NULL)
        {
            exit(0);
        }
    double *tsin=NULL; //tableau de l'onde calculeÈe par le transformÈe de fourrier
    tsin=malloc(n * sizeof(double));
    if (tsin == NULL)
        {
            exit(0);
        }
    // prÈparation des calculs (il n'existe pas de complexe en c,on doit traiter les
imaginaires et les rÈels sÈparÈment.
    n2=n/2;
    arg=PI/n2;
    for(i=0;i<n2;i++)
        {
        tcos[i]=cos(-arg*i);
        tcos[i+n2]=-tcos[i];
        tsin[i]=sin(-arg*i);
        tsin[i+n2]=-tsin[i];
        }
    for (i=0;i<n;i++)
        {// "bit reversal"
        j=tf_reversion(i,r);
        t[j][0]=f[i][0];
        t[j][1]=f[i][1];
        }
    ne=1;n1=2;ns=n;
    for (k=0;k<r;k++)
        {
//        printf("Transformation T%d :\n",k);
        for (i=0;i<n;i++)
        for (i=0;i<=n-n1;i+=n1)
        for(j=i;j<=i+ne-1;j++)
            {
            tj=j*n2;
            while (tj>=n) tj-=n;
            cr=tcos[tj];
            ci=tsin[tj];
            // opÈrateur "butterfly" 1ere partie
            tr1=t[j][0]+cr*t[j+ne][0]-ci*t[j+ne][1];
            ti1=t[j][1]+cr*t[j+ne][1]+ci*t[j+ne][0];
            tj=(j+ne)*n2;
            while (tj>=n) tj-=n;
            cr=tcos[tj];
            ci=tsin[tj];
            // opÈrateur "butterfly" 2eme partie
            tr2=t[j][0]+cr*t[j+ne][0]-ci*t[j+ne][1];
            ti2=t[j][1]+cr*t[j+ne][1]+ci*t[j+ne][0];
            t[j][0]=tr1;t[j][1]=ti1;
            t[j+ne][0]=tr2;t[j+ne][1]=ti2;
            }
        ne=n1;n1=2*ne;ns=n2;n2=ns/2;
        }
    free(tcos);
    tcos = NULL;
    free(tsin);
```

```
    tsin = NULL;
}



struct wavfile //dÈfinit la structure de l entete d un wave
{
    char        id[4];          // doit contenir "RIFF"
    int         totallength;        // taille totale du fichier moins 8 bytes
    char        wavefmt[8];     // doit etre "WAVEfmt "
    int         format;             // 16 pour le format PCM
    short       pcm;              // 1 for PCM format
    short       channels;         // nombre de channels
    int         frequency;          // frequence d echantillonage
    int         bytes_per_second;   // nombre de bytes par secondes
    short       bytes_by_capture;
    short       bits_per_sample;  // nombre de bytes par echantillon
    char        data[4];        // doit contenir "data"
    int         bytes_in_data;      // nombre de bytes de la partie data
};



int main(int argc, char *argv[])
{


    int i=0,j=0; // correspondent aux indice des tableaux de complexes (partie rÈelle et
imaginaire)
    int taille=1; //variable qui correspondra par la suite a la longueur du tableau
(puissance de 2)
    int nbech=0; //nombre d echantillons extraits du fichier audio
    int puissance=0; //variable qui serta incrÈmentÈe qui correspond a l indice de la
puissance de 2 pour la taille du tableau
    int pas=1;
    float a1,a2; //min et max de l'intensitÈ (log) de la tf
    char fichieraudio[100];
    int pow=1;
    int powpas=0;
    double sommepas=0;
    int exitchoix=0;
    int choix=0;
    float tempstotal=0;
    double pasfrequence;
    double pastemps;
    int idim, jdim, imin, imax, jmin, jmax;
    float tr[6],xmin,xmax,ymin,ymax;
    float xcur[2],ycur[2];
    int npt=0;
    int idim2, jdim2;
    float xanalyse[2],yanalyse[2];
    int analyse,nptanalyse=0;
    float note;
    float freqa, freqe, doppler;

/*----------------------selection du fichier audio-------------------------------*/
    printf ("entrer le nom du fichier audio a analyser :\n");
    scanf("%s", fichieraudio);
    printf ("nom du fichier : %s\n", fichieraudio);
/*--------------fin de selection du fichier audio-------------------------------*/



/*---------------------ouverture du wave-----------------------------------------*/
    FILE *wav = fopen(fichieraudio,"rb"); //ouverture du fichier wave
    struct wavfile header; //creation du header

    if ( wav == NULL )
    {
        printf("\nne peut pas ouvrir le fichier demande, verifier le nom\n");
        printf("ne pas oublier l'extention .wav\n");
        exit(1);
```

```
    }
/*---------------------fin de ouverture du wave---------------------------------*/


/*---------------------lecture de l entete et enregistrement dans header--------*/
    //initialise le header par l'entete du fichier wave
    //verifie que le fichier possÈde un entete compatible
    if ( fread(&header,sizeof(header),1,wav) < 1 )
    {
        printf("\nne peut pas lire le header\n");
        exit(1);
    }
    if (    header.id[0] != 'R'
         || header.id[1] != 'I'
         || header.id[2] != 'F'
         || header.id[3] != 'F' )
    {
        printf("\nerreur le fichier n'est pas un format wave valide\n");
        exit(1);
    }
    if (header.channels!=1)
    {
        printf("\nerreur : le fichier n'est pas mono\n");
        exit(1);
    }
/*----------------fin de lecture de l entete et enregistrement dans header-------*/


/*-------------------------determiner la taille des tableaux--------------------*/
    nbech=(header.bytes_in_data/header.bytes_by_capture);
    printf ("\nle fichier audio contient %d echantillons\n",nbech);
    while (nbech>taille)
    {
        taille=taille*2;
        puissance=puissance+1;
    }
    printf ("nombre de points traites pour le spectrogramme : 2^%d=%d
\n",puissance,taille);
    tempstotal=(1./header.frequency)*taille;
    printf ("temps total du spectrogramme : %f s\n",tempstotal);
/*------------------fin de determiner la taille des tableaux--------------------*/


/*---------------------selection du pas-----------------------------------------*/
    while (choix==0)
    {
        printf ("\nselection du PAS :\n");
        printf ("entrer 1 pour une simple transformee de Fourier\n");
        printf ("ou une puissance de 2 pour un spectrogramme d'un nombre egal de PAS
\n");
        scanf  ("%d",&pas);
        while (pow<pas)
        {
            pow=pow*2;
            powpas++;
        }
        if (pow!=pas || pas<1)
        {
            printf("le PAS est invalide\n");
        }
        else
        {
            printf ("pas=%d\n", pas);
         pasfrequence=1.*(header.frequency/2)/(taille/(2*pas));
            printf ("precision frequence : %.16lf hz\n",pasfrequence);
            pastemps=tempstotal/pas;
            printf ("precision temps : %.16lf s\n",pastemps);
            printf ("\ntapez 1 pour continuer ou 0 pour choisir de nouveau le pas\n");
            scanf ("%d",&choix);
```

```
        }
    }
/*--------------fin de selection du pas-----------------------------------------*/


/*---------------------creation des tableaux dynamiques-------------------------*/
    double **tab=NULL; //tableau de l'onde temporelle
    tab=malloc( (taille) * sizeof(double));
    if (tab == NULL)
    {
        exit(0);
    }
    for(i=0;i<(taille);i++)
    {
        tab[i]=malloc( 2 * sizeof(double));
        if (tab[i] == NULL)
        {
            exit(0);
        }
    }
    double **tabfft=NULL; //tableau calculÈ par la transformÈe de fourrier
    tabfft=malloc((taille/pas) * sizeof(double));
    if (tabfft == NULL)
    {
        exit(0);
    }
    for(i=0;i<(taille/pas);i++)
    {
        tabfft[i]=malloc( 2 * sizeof(double));
        if (tabfft[i] == NULL)
        {
            exit(0);
        }
    }
    double **tabpas=NULL;
    tabpas=malloc((taille/pas) * sizeof(double));
    if (tabpas == NULL)
    {
        exit(0);
    }
    for(i=0;i<(taille/pas);i++)
    {
        tabpas[i]=malloc( 2 * sizeof(double));
        if (tabpas[i] == NULL)
        {
            exit(0);
        }
    }
    float **graph=NULL; //tableau du graph
    graph=malloc(((taille/pas)/2) * sizeof(float));
    if (graph == NULL)
    {
        exit(0);
    }
    for(i=0;i<((taille/pas)/2);i++)
    {
        graph[i]=malloc( (pas) * sizeof(float));
        if (graph[i] == NULL)
        {
            exit(0);
        }
    }
    float *graph1d=NULL; //tableau du graph
    graph1d=malloc( (taille/2) * sizeof(float));
    if (graph1d == NULL)
    {
        exit(0);
    }
/*--------------------fin de creation des tableaux dynamiques-----------------*/
```

```
/*----------------------initialisation des tableaux dynamiques--------------------*/
    for(i=0;i<taille;i++)
    {
        tab[i][0]=0;
        tab[i][1]=0;
    }
    for(i=0;i<(taille/pas);i++)
    {
        tabpas[i][0]=0;
        tabpas[i][1]=0;
        tabfft[i][0]=0;
        tabfft[i][1]=0;
    }
    for (j=0;j<pas;j++)
    {
        for(i=0;i<((taille/pas)/2);i++)
        {
            graph[i][j]=0;
        }
    }
    for (i=0;i<(taille/2);i++)
    {
        graph1d[i]=0;
    }
/*----------------------fin de initialisation des tableaux dynamiques-------------*/


/*----------------------remplissage du tableau tab avec les echantillons----------*/
    i=0;
    short value=0;
    FILE *dat=fopen("data.dat","w"); //fichier data des echantillons
    FILE *dat2=fopen("dataout.dat","w");//fichier.dat des fft
    FILE *dat3=fopen("graphout.dat","w");//fichier.dat du graph1d de pgplot
    while( fread(&value,(header.bits_per_sample)/8,1,wav) )
    { //lecture des echantillons et enregistrement dans le tableau
        tab[i][0]=value;
        i++;
    }
    printf("\nnombre d'echantillons lus : %d\n",i);
    printf("nombre de valeurs sauvegardees %d\n",i);
    for (i=0;i<taille;i++)
    {
        fprintf(dat,"%lf %lf\n", tab[i][0], tab[i][1]);
//permet de sauvegarder le tableau dans le fichier data.dat pour vÈrification manuelle
des donnÈes
    }
/*-----------------fin de remplissage du tableau avec les echantillons-----------*/


/*----------------------fft-------------------------------------------------------*/
    int r;
    unsigned int n;
    printf("\ndebut de la creation du spectrogramme\n");
    printf("taper 1 pour continuer\n");
    int pause=0;
    scanf("%d", &pause);
    if (pause!=1)
    {
        exit(0);
    }
    printf("veuillez patienter...\n");
    r=puissance-powpas;
    n=taille/pas;
    for (j=0;j<pas;j++)
    {
        printf("pas %d/%d\n",j+1,pas);
        sommepas=0;
        for(i=0;i<(taille/pas);i++)
        {
```

```
            tabpas[i][0]=tab[i+j*(taille/pas)][0];
            sommepas=sommepas+tabpas[i][0];
        }
        if (sommepas==0)
        {
            for (i=0;i<(taille/pas);i++)
            {
                tabfft[i][0]=0;
                tabfft[i][1]=0;
            }
        }
        else
        {
            tf_cooley_tukey(r,n,tabpas,tabfft);
        }
        for (i=0;i<((taille/pas)/2);i++)
        {
            graph[i][j]=log(sqrt(tabfft[i][0]*tabfft[i][0]+tabfft[i][1]*tabfft[i][1]));
            if (graph[i][j]<0)
            {
                graph[i][j]=0;
            }
            if (i==0&&j==0)
            {
                a1=graph[i][j];
                a2=graph[i][j];
            }
            if(a1>graph[i][j])
            {
                a1=graph[i][j];
            }
            if(a2<graph[i][j])
            {
                a2=graph[i][j];
            }
            fprintf(dat2,"%f pas=%d spectr:%lf\n",graph[i][j],j,tabpas[i][0]);
//permet de sauvegarder le tableau dans le fichier dataout.dat pour vÈrification
manuelle des donnÉes
        }
    }
/*--------------------fin de fft-------------------------------------------------*/


/*---------------------representation graphique-----------------------------------*/
    jdim=1.*taille/(2*pas);
    idim=pas;
    imin=1;
    imax=idim;
    jmin=1;
    jmax=jdim;
    while (exitchoix==0)
    {
        for (i=0;i<((taille/pas)/2);i++)
        {
            for (j=0;j<pas;j++)
            {
                graph1d[i*pas+j]=(float)graph[i][j];
            }
        }
//        for (i=0;i<(taille/2);i++)
//        {
//            fprintf(dat3,"%f\n",graph1d[i]);
//        }

        xmin=0;
        ymin=0;
        ymax=taille*(1./header.frequency);
        xmax=(1.*header.frequency/2);

        tr[0]=ymin;
        tr[1]=(ymax-ymin)/idim;
```

```
        tr[2]=0;
        tr[3]=xmin;
        tr[4]=0;
        tr[5]=(xmax-xmin)/jdim;
        //printf("min : %f  max: %f\n",a1,a2);
        cpgopen("/xw");
        cpgenv(ymin,ymax,xmin,xmax,0,0);
        cpgimag(graph1d, idim, jdim, 1, idim, 1, jdim, a1, a2, tr);
        cpglab("temps","frequences","spectrogramme");

        printf ("\n----Menu----\n");
        printf ("\n 0  arreter le programme\n");
        printf (" 1  effectuer un zoom\n");
        printf (" 2  retrouver une note de musique\n");
        printf (" 3  calcul de vitesse par effet doppler\n");
        scanf ("%d",&choix);
        switch (choix)
        {
            case 0:
                cpgend();
                exit(0);
            break;

            case 1:
                printf ("\nzoom\n");
                cpgsci(7);
                cpgsch(1.5);
                cpgncur(2,&npt,ycur,xcur,6);

                xmax=MAX(xcur[0],xcur[1]);
                xmin=MIN(xcur[0],xcur[1]);
                ymin=MIN(ycur[0],ycur[1]);
                ymax=MAX(ycur[0],ycur[1]);
                if (xmin<0)
                {
                    xmin=0;
                }
                if (xmax>(header.frequency/2))
                {
                    xmax=(header.frequency/2);
                }
                if (ymin<0)
                {
                    ymin=0;
                }
                if (ymax>pas)
                {
                    ymax=pas;
                }

                printf ("ymin=%f xmin=%f ymax=%f xmax=%f\n",ymin,xmin,ymax,xmax);

                jmin=(int)(xmin*(taille/(2*pas))/(header.frequency/2));
                jmax=(int)(xmax*(taille/(2*pas))/(header.frequency/2));
                imin=(int)(ymin*pas/tempstotal);
                imax=(int)(ymax*pas/tempstotal);
                idim2=imax-imin;
                jdim2=jmax-jmin;
                ymin=imin*tempstotal/pas;
                ymax=imax*tempstotal/pas;
                xmin=jmin*(header.frequency/2)/(taille/(2*pas));
                xmax=jmax*(header.frequency/2)/(taille/(2*pas));
                //printf ("jmin=%d jmax=%d imin=%d imax=%d idim2=%d jdim2=%d
\n",jmin,jmax,imin,imax,idim2,jdim2);
                cpgsfs(2);
                cpgrect(ymin,ymax,xmin,xmax);

                float *graph1dbis=NULL;
                graph1dbis=malloc( (idim2*jdim2) * sizeof(float));
                if (graph1dbis == NULL)
                {
                    exit(0);
```

```
                }

                for (i=jmin;i<jmax;i++)
                {
                    for (j=imin;j<imax;j++)
                    {
                        graph1dbis[(i-jmin)*idim2+(j-imin)]=(float)graph[i][j];
                    }
                }

                tr[0]=ymin;
                tr[1]=(ymax-ymin)/idim2;
                tr[2]=0;
                tr[3]=xmin;
                tr[4]=0;
                tr[5]=(xmax-xmin)/jdim2;

                cpgend();
                cpgopen("/xw");
                cpgenv(ymin,ymax,xmin,xmax,0,0);
                cpgimag(graph1dbis, idim2, jdim2, 1, idim2, 1, jdim2, a1, a2, tr);
                cpglab("temps","frequences","spectrogramme");

                analyse=1;
                while (analyse==1)
                {
                    printf ("analyser un point ? taper 1=oui 0=non\n");
                    scanf ("%d",&analyse);
                    if (analyse==1)
                    {
                        cpgncur(1,&nptanalyse,yanalyse,xanalyse,6);
                        printf ("frequence=%f temps=%f\n",xanalyse[0],yanalyse[0]);
                    }
                }

                cpgask(1);
                cpgend();
                free(graph1dbis);
                graph1dbis = NULL;
            break;

            case 2:
                cpgend();
                printf ("\nentrer la frequence pour retrouver la note de musique
(30-7902Hz) :\n");
                scanf ("%f",&note);
                if (note>30 && note<7902)
                {
                    notemusique(note);
                }
            break;

            case 3:
                cpgend();
                printf ("\neffet doppler :\n");
                printf("\nentrer la frequence a l approche\n");
                scanf ("%f",&freqa);
                printf("entrer la frequence a l eloignement\n");
                scanf ("%f",&freqe);
                doppler=(340-((340*freqe)/freqa))/(1+(freqe/freqa));
                printf("vitesse=%f m*s-1 soit %f Km/h\n",doppler,(doppler*3.6));
            break;

            default:
              cpgask(1);
              cpgend();
                printf ("\nchoix invalide\n");
            break;
        }
    }
/*---------------fin de representation graphique---------------------------------*/
```

```
/*----------------------liberation de la memoire----------------------------------*/
    //liberation de la ram des malloc
    for(i=0;i<taille;i++)
    {
        free(tab[i]);
        tab[i] = NULL ;
    }
    for (i=0;i<(taille/pas);i++)
    {
        free(tabpas[i]);
        tabpas[i] = NULL ;
        free(tabfft[i]);
        tabfft[i] = NULL ;
    }
    for (i=0;i<((taille/pas)/2);i++)
    {
        free(graph[i]);
        graph[i] = NULL ;
    }
    free(tab);
    tab = NULL;
    free(tabfft);
    tabfft = NULL;
    free(tabpas);
    tabpas = NULL;
    free(graph);
    graph = NULL;
    free(graph1d);
    graph1d = NULL;
    fclose(wav);
    fclose(dat);
    fclose(dat2);
    fclose(dat3);
/*----------------------fin de liberation de la memoire---------------------------*/


    exit(0);
}//fin du main
```