TP Traitement du signal ; la DFT

Le code permettant le chargement d'un wav est sur http://www.lirmm.fr/~chaumont/download/cours/TS/tp
Vous ferez une implémentation en C++.

Question 1 : Création d'un son (wav) correspondant à la note la.

Vous aller générer un signal sinusoïdal de fréquence 440 Hz (ce qui correspond à la fréquence du la). Vous ferez durer ce signal 10 secondes. Pour sauvegarder le signal obtenu, vous devez utiliser le constructeur Wave :

```
Wave::Wave(unsigned char* data8, long int data_nb, short channels_nb, int
sampling_freq)
```

Puis appeler la méthode :

```
Wave::write(char* fileName);
```

Vous créerez donc un fichier « wav » contenant un signal sinusoïdal de fréquence 440 Hz, mono, dont les données sont sur 8 bits, et dont la fréquence d'échantillonnage est par exemple 44100 Hz. Remarquez que pour un « wav » 8 bits, l'amplitude nulle se trouve à la valeur 127. Vous trouverez en Annexe une explication plus détaillée du format « wav ».

Question 2 : DFT

Implémenter la Transformée de Fourier Discrète et la transformée inverse tels que :

$$X(k) = \frac{1}{T}\sum_{t=0}^{T-1} x(t)e^{-j\frac{2\pi}{T}k.t}$$

$$x(t) = \sum_{k=0}^{T-1} X(k)e^{2\pi j\frac{kt}{T}}$$

```
void DFT (int T, double *x1, double *y1) ;
void IDFT(int T, double *x1, double *y1) ;
```

Question 3 : DFT - IDFT

Faire des affichages intermédiaires et observer le comportement réversible sur un signal réel quelconque (par exemple 64 échantillons d'un des fichiers « wav » donné (« siffler.wav » par exemple est un « wav » mono, 8 bits).

Question 4 : module et phase

Implémenter une fonction qui calcule l'amplitude et la phase à partir d'une représentation réelle et imaginaire.
Implémenter une fonction de décalage telle que la fréquence nulle soit au milieu du tableau.
Afficher les valeurs d'amplitude pour quelques signaux.

Question 5 : Vérification de quelques propriétés de la transformée de Fourier

A partir d'un signal réel symétrique en 0 observer la transformée de Fourier. Quel est la propriété que l'on observe ?

A partir d'un signal réel constant (par exemple = 1) observer la transformée de Fourier. Quelle est la propriété que l'on observe ?

A partir d'un signal réel périodique pur pair (exemple x(t) = cos(wt)) observer la transformée de Fourier. On prendra par exemple w = 2*Pi/T et également w = 4*Pi/T avec T le nombre d'échantillons. Quelle est la propriété que l'on observe ?

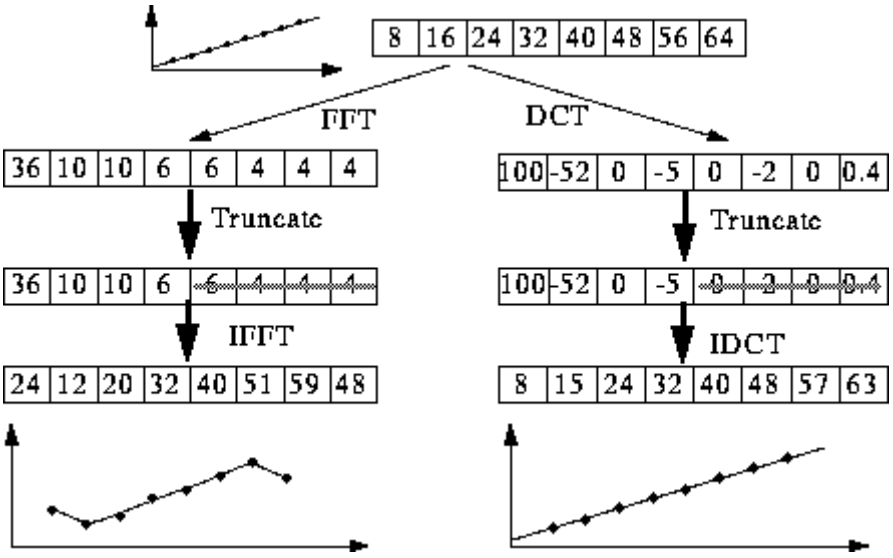
A partir d'un signal réel périodique pur impair (exemple x(t) = sin(wt)) observer la transformée de Fourier. On prendra par exemple w = 2*Pi/T et également w = 4*Pi/T avec T le nombre d'échantillons. Quelle est la propriété que l'on observe ?

Question 4 :

Faire la transformée de Fourier sur les 64 premier échantillons, puis les 64 suivant etc.. d'un signal réel (charger un « wav » mono, 8 bits ; « siffler.wav » par exemple). Ne garder que les 16 harmoniques (cad 32 fréquences) de plus basses fréquences (on met donc à zéro tout le reste), reconstruire le signal temporel et le sauvegarder dans un fichier « wav ». Observer l'effet.

Question 5 :

Implémenter la DCT et l'IDC et observer la différence de concentration d'énergie entre un signal transformé par DFT et un signal transformé par DCT.



Annexe : Le format wav

Source de la notice : http://sub0.developpez.com/index.htm

Header d'un fichier wav :

```
// CHUNK TYPE
char file_type[4];      // (4 octets) : Constante "RIFF" i.e id du format
int file_size;          // (4 octets) : file_size est nb d'octets restant à lire
char file_id[4];        // (4 octets) : Identifiant "WAVE"
// CHUNK FORMAT
char chunk_id[4];       // (4 octets) : Identifiant "fmt "
int chunk_size;         // (4 octets) : Nombre d'octets pour définir le chunk
short format;           // (2 octets) : Format de fichier (1: PCM,  ...)
short channels_nb;      // (2 octets) : Nombre de canaux (1 = mono, 2 = stéréo)
int sampling_freq;      // (4 octets) : Fréquence d'échantillonnage (en Hertz)
int bytes_per_second;   // (4 octets) : Nombre d'octets par seconde de musique
short bytes_per_sample; // (2 octets) : Nombre d'octets par échantillon
short depth;            // (2 octets) : Nombre de bits par donnée (8 ou 16)
// CHUNK DONNéES
char data_id[4];        // (4 octets) : Constante "data"
int data_size;          // (4 octets) : nombre d'octets restant
// DATA
unsigned char* data8;   // Tableau de données (cas données 8 bits)
short* data16;          // Tableau de données (cas 16 bits)
long int data_nb;       // Nombre de données
```

L'amplitude : En 8 bits, l'amplitude possède une résolution de 256 valeurs. En 16 bits, 65536 valeurs, etc... Il existe aussi le 24 et 32 bits.

La fréquence : La fréquence d'échantillonnage, correspond au nombre d'échantillons (sample) pour une seconde d'enregistrement. Ainsi 44100 Hz signifie 44100 échantillons pour une seconde de son mémorisé. Plus la fréquence d'échantillonnage est élevée, meilleure est la qualité du signal sonore, plus les données digitales sont proches de l'original.

Le débit : On peut calculer le "débit" (ko/s) d'un signal avec les paramètres: depth (8 ou 16 bits), channels_nb (mono ou stéréo) et la fréquence 'sampling_freq'. Par exemple, en 8 bits mono 44100Hz, cela nous donne pour une seconde d'enregistrement: 44100 octets (1 octet=8 bits). En stéréo, c'est le double. En 16 bits stéréo, c'est le quadruple. Ainsi, avec la qualité du CD audio (16 bits stéréo 44,1 kHz), on obtient 176400 octets par seconde.

Le format PCM : C'est le format de fichier "standard" pour les signaux sonores, car les données sont brutes, c'est-à-dire qu'elles ne sont ni modifiées, ni compressées. Le fichier possède une en-tête de 44 octets, permettant de connaître le type du signal: son format, sa fréquence, le nombre de voies, etc... En 8 bits, la valeur de l'amplitude est non signée, et en 16 bits, l'amplitude est signée.

L'ordre des données : Après les 44 octets de l'en-tête, viennent les données. Les données ont un ordre bien défini. Dans le cas d'un signal 8 bits mono, c'est 1 seul octet par sample, donc les données se suivent normalement. Pour un signal 8 bits stéréo, ce sont 2 octets par sample, l'octet de la voie de gauche, puis l'octet de la voie de droite: L,R, L,R, L,R, etc... Dans le cas d'un signal 16 bits mono, ce sont 2 octets par sample, l'octet de poids faible, puis l'octet de poids fort dans le cas d'une représentation en mémoire « little endian ». Dans le cas d'un signal 16 bits stéréo, ce sont 4 octets par sample; Deux octets pour la voie de gauche, et deux pour la voie de droite : L,L,R,R, L,L,R,R, L,L,R,R, etc...

Le format des données : Le signal 8 bits (mono ou stéréo) possède des données non-signées, c'est-à-dire que le point (l'amplitude) le plus bas vaut zéro, le point du milieu vaut 127, et le point le plus haut vaut 255. Avec un signal 16 bits, les choses sont différentes, les données sont signées: le point le plus bas vaut -32268, le point du milieu vaut zéro, et le point le plus au vaut 32767.

Attention! Toutes ces explications ne sont valables que pour le format PCM, car il existe d'autres types de compression pour les signaux WAV. Cependant, le format PCM est le plus utilisé pour le traitement des samples.

Fréquences des notes dans 3 systèmes, LA=440 Hz

Note	Juste intonation	Gamme de Pythagore	**Gamme tempérée**
DO	264,00	260,74	**261,63**
DO♯	275,00	278,44	**277,18**
RE	297,00	293,33	**293,66**
MI♭	316,80	309,03	**311,13**
MI	330,00	330,00	**329,63**
FA	352,00	347,65	**349,23**
FA♯	371,25	371,25	**369,99**
SOL	396,00	391,11	**392,00**
SOL♯	412,50	417,66	**415,30**
LA	**440,00**	**440,00**	**440,00**
SI♭	475,20	463,54	**466,16**
SI	495,00	495,00	**493,88**
DO	528,00	521,48	**523,25**

Sources : Wikipédia