Projet informatique libre : compression de données numériques sonores

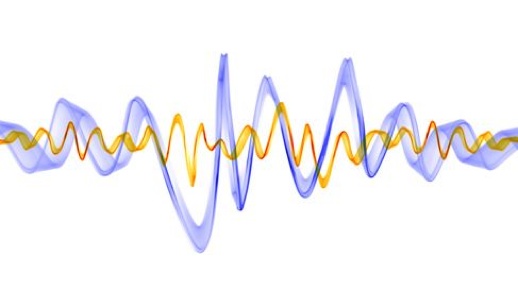


Table des matières

Introduction 4

I. La conversion analogique numérique 4

1. Décrire le son 5

2. Traiter le signal sonore numériquement 6

3. De la nécessité de passer du domaine temporel au domaine fréquentiel 7

II. La Transformée de Fourier 8

1. Passage du domaine temporel au domaine fréquentiel 8

2. L'algorithme de Transformée de Fourier Rapide (FFT), version récursive 9

3. L'algorithme de Transformée de Fourier Rapide (FFT), version itérative 10

III. Implémentation de l'algorithme de FFT 11

1. Manipuler de très grands entiers 11

2. Lecture et écriture d'un fichier WAVE 11

3. Quantification des échantillons 13

4. Transformée de Fourier 14

IV. Conclusion 15

V. Bibliographie 17

Introduction

L'intérêt de la compression de données

La quantité d'information numérique qui circule sur Internet, et d'autres réseaux, est de plus en plus importante. Cette expansion a été permise par l'avènement du haut débit d'une part, et de la fibre optique plus récemment, mais aussi par les progrès dans le domaine de la compression.

Dans ce projet, on va s'intéresser à la compression de données numériques représentant un signal sonore. On peut se demander l'intérêt, dans le sens où aujourd'hui, les connexions internet dans la plupart des villes de France sont suffisamment rapides pour transmettre en quelques secondes des fichiers musicaux non compressés. Il faut savoir qu'on peut étendre les résultats présentés à des données vidéo et cela justifie l'intérêt porté à l'amélioration des algorithmes utilisés puisque les quantités d'information mises en jeu sont alors importantes, même en regard des vitesses de connexion actuelles.

Objectif de notre projet

Nous souhaitons, à partir d'un enregistrement numérique non compressé (un .wav tout ce qu'il y a de plus banal), obtenir un autre fichier numérique, compressé, c’est-à-dire occupant moins de place en mémoire. Il faudra qu'on puisse ensuite décompresser ce fichier pour retrouver un .wav. On utilisera pour cela une méthode de compression psycho acoustique : on élimine de l'information, mais de telle façon que cela ne s'entende pas.

Plan de ce compte rendu

Dans un premier temps, on présentera les objets mathématiques et numériques sur lesquels on va travailler. On expliquera aussi l'utilité du passage du domaine temporel au domaine fréquentiel. Si les notions d'échantillonnage et de quantification vous sont familières, ainsi que la structure d'un fichier .wav, vous pouvez sauter cette partie !

Dans un second temps, on étudiera la transformée de Fourier, et on présentera notre cheminement pour obtenir un algorithme de calcul le plus efficace possible. C'est dans cette partie qu'on traitera de la transformée de Fourier rapide tout d'abord dans une version récursive, puis dans une version itérative. Si la seule chose qui vous intéresse est la partie informatique, passez directement à la troisième partie.

Ensuite, on passera à la phase d'implémentation, et d'optimisation de la vitesse de calcul par le code et au test de l'efficacité de nos algorithmes.

# La conversion analogique numérique

## Décrire le son

Un son n'est qu'une vibration mécanique de l'air ou de tout autre fluide compressible qui se propage sous forme d'ondes mécaniques. Un capteur (un microphone ou notre oreille) est capable de générer un signal électrique proportionnel à l'amplitude de ces ondes. Il existe deux manières de représenter ce signal (et donc le son) :

* La **représentation temporelle**, consiste à donner son amplitude en fonction du temps. On modélise ainsi le signal par une fonction mathématiquede (on prend comme départ de l'enregistrement) dans un intervalle d'amplitudes du signal que peut générer notre capteur avec . Cette représentation est la plus immédiate, c'est celle que l'on obtient directement en sortie d'un microphone.
* D'autre part, d'après la **théorie de Fourier** : on peut associer à tout signal T-périodique (dans ou ),sa série de Fourier

Et d'après le **théorème de Dirichlet**, si est suffisamment régulière (continue, dérivable et de dérivée continue par morceau), cette série converge vers :

où chaque terme de la deuxième somme est une harmonique de fréquence multiple de la fréquence fondamentale

Ainsi, on peut donner la **représentation fréquentielle** d'un signal, aussi appelé spectre, qui contient l'amplitude , la fréquence et la phase de chaque harmonique du signal. En effet, si est de durée finie, on le prolonge par périodicité sur, sinon, on peut le découper en intervalles (qu'on appellera frames) et prolonger ceux-ci par périodicité sur . On peut alors appliquer le théorème et décrire (sur chaque frame) par les coefficients. On modélise alors le signal (sur chaque frame) par une fonction mathématique.

On verra comment passer d'une représentation à l'autre dans la deuxième partie.

## Traiter le signal sonore numériquement

Pour pouvoir traiter le signal sonore numériquement, il faut d'abord le transformer en une suite de valeurs binaires (formées de 0 et de 1) qu'un ordinateur est capable de manipuler. C'est ce qu'on appelle la **conversion analogique numérique**. Elle répond à deux nécessités :

* cette **suite de valeurs doit être finie**, car chaque valeur occupe une place donnée en mémoire, et la capacité mémoire d'un ordinateur est finie.
* pour chaque valeur, on doit faire un **compromis entre précision et place mémoire occupée**. Si l'on veut stocker un entier par exemple, cela prend relativement peu de place : avec 16 bits on peut décrire n'importe quel entier naturel de à . Mais si l'on veut stocker un réel avec un très grand nombre de décimales, il convient de l'arrondir à un nombre de décimales raisonnable, qui dépend des capacités de notre machine.

Dans une première étape, on va donc discrétiser le signal , en prélevant sa valeur à intervalles réguliers généralement. Cette opération est appelée **l'échantillonnage.** On appelle **période d'échantillonnage** la durée qui sépare deux mesures, **fréquence d'échantillonnage** son inverse : et échantillon chaque valeur relevée.

Cela se traduit par l'obtention d'une représentation temporelle discrète, qu'on modélise mathématiquement par une suite où définie par : .

On peut se demander comment choisir la fréquence d'échantillonnage, en effet plus celle-ci est faible, moins notre suite possèdera de valeurs et moins elle occupera de place en mémoire. Mais si on relève trop peu d'échantillons, notre représentation discrète ne comportera pas toutes les subtilités du signal d'origine.

C'est ici qu'intervient le **théorème d'échantillonnage de Shannon** :

Si un signal a une fréquence maximale, alors sa représentation discrète, par des échantillons régulièrement espacés exige une fréquence d'échantillonnage telle que : .

Or, on considère que l'oreille humaine n'est capable de capter que des signaux sonores de fréquences comprises entre et , ou encore de pulsation comprises entre et

. On peut donc tronquer la représentation spectrale et considérer que :

et

En prenant . En pratique, on utilise souvent (pour des raisons historiques).

Dans une seconde étape, on va approximer chaque valeur de la suite par une valeur occupant peu de place en mémoire. Cette opération est appelée **quantification**. Dans une majorité des fichiers musicaux actuels, la valeur d'un échantillon est stockée sur 16 bits. Ces 16 bits permettent de décrire valeurs différentes. C'est pourquoi on divise l'intervalle décrit par nos échantillons en intervalles. On parle de quantification uniforme si ces intervalles sont de même longueur, c'est ainsi que l'on va précéder dans un premier temps. On appelle **pas de quantification** cette longueur, qui vaut .

On approxime alors chaque échantillon par une valeur de l'intervalle de quantification auquel il appartient,(on peut fixer l'ouverture ou la fermeture des bornes de l'intervalle selon différentes conventions). On appelle **erreur de quantification** la quantité . On veut bien évidemment minimiser cette erreur, pour représenter l'échantillon de façon la plus précise possible. Pour cela, il parait logique, dans le cadre d'une quantification uniforme, d'utiliser la valeur centrée pour ainsi majorer l'erreur par . Puis on code sur bits, par la valeur binaire de .

Résumons toutes les étapes, en prenant l'exemple du signal suivant, échantillonné à une fréquence (le signal est une superposition de 3 sinusoïdes, avec ) et quantifié sur 3 bits.

## De la nécessité de passer du domaine temporel au domaine fréquentiel

Sur ce graphique, on voit qu'à partir des échantillons, on serait bien en peine de reconstruire le signal originel, notamment sur la portion [5,9] où les oscillations ont été totalement gommées. Pourtant , donc le problème ne provient pas de l'échantillonnage : c'est le pas de quantification et donc l'erreur de quantification sur chaque échantillon qui sont trop importants. Pour réduire la place occupée par un fichier musical en mémoire on pourrait augmenter le pas de quantification, mais cet exemple montre que cette solution a ses limites.

C'est une solution utilisé pour certains signaux sonores dont la qualité n'a pas besoin d'être aussi grande que celle d'une musique : ceux des conversations téléphoniques par exemple, qui peuvent être quantifiés sur 8 bits seulement.

Si l'oreille humaine détecte facilement la pauvre qualité d'un signal sur 8 bits, elle est en revanche beaucoup moins sensible à des légères modifications de tonalité, c’est-à-dire des coefficients . Une solution employée pour réduire la taille d'un enregistrement peut donc être de stocker sa représentation fréquentielle au lieu de sa représentation temporelle, que l'on pourra elle quantifier sur un nombre plus restreint de bits sans que cela soit perceptible.

C'est ainsi que l'on va précéder dans la partie II. C'est une méthode de compression qualifiée de **psycho-acoustique**, car on exploite les caractéristiques de l'oreille humaine, pour supprimer des informations qui ne sont pas réellement traitées par celle-ci.

Dans la partie IV, on justifiera a posteriori ce choix, en comparant un signal obtenu par une augmentation directe du pas de quantification et un signal obtenu en passant au domaine fréquentiel et en quantifiant les coefficients .

# La Transformée de Fourier

## Passage du domaine temporel au domaine fréquentiel

La transformée de Fourier est une extension du développement en séries de Fourier pour les fonctions non périodiques. Pour obtenir la représentation fréquentielle d'un signal intégrable (pas forcément périodique), on calcul sa **transformée de Fourier** définie par :

Pour un signal suffisamment régulier (s'il est continu et sa transformée de Fourier est intégrable), on peut retrouver le signal d'origine par la **transformée de Fourier inverse** définie par :

Si l'on discrétise le signal en l'échantillonnant, il est représenté par une suite de échantillons telles que où est la période d'échantillonnage. On peut alors utiliser des **transformées discrètes**:

Si on a respecté le critère de Shannon lors de l'échantillonnage, **la transformée de Fourier discrète (notée TFD par la suite) permet d'évaluer une représentation spectrale discrète, du signal discret en entrée, sur la fenêtre de temps fini .** C'est exactement ce que l'on souhaite : pour chaque frame (fenêtre de temps fini), on veut obtenir un spectre échantillonné pour pouvoir le quantifier (cf I.). On retrouve le signal discret d'origine en appliquant la TFD inverse à sa TFD.

Effectuons plusieurs remarques :

* En appliquant ces deux transformations à une suite de échantillons, on obtient toujours échantillons, rien ne sert de faire le calcul de par exemple puisque la transformée de Fourier discrète et son inverse son périodique de période . **Pour toute la suite on suppose que est une puissance de 2**. Donc toutes les fréquences comprises entre et sont les mêmes que celles comprises entre et
* Dans le cas d'un signal sonore, est réel, et on a alors la relation qui se vérifie rapidement : .
* Finalement, toute l'information spectrale est contenue entre les fréquences N/2 et N par exemple.

## L'algorithme de Transformée de Fourier Rapide (FFT), version récursive

On peut voir directement par l'expression de la transformée de Fourier discrète que son calcul direct pour échantillons demande multiplications complexes. Cette transformée aurait des applications très limitées s'il n'existait pas d'algorithme plus rapide pour la calculer. C'est ici qu'intervient la Transformée de Fourier Rapide (notée par la suite FFT pour Fast Fourier Transform en anglais), algorithme fondamental de notre projet qui se base sur une stratégie "diviser pour régner" et que nous allons présenter maintenant :

**Première étape** : Découper le problème en problème de plus petite taille. Remarquons qu'avec :

(Rappelons que comme notre signal est réel, dans notre cas, on ne calculera que les premiers échantillons et on obtiendra les suivants par symétrie)

Avec cette expression, on voit que chaque calcul d'une transformée de Fourier fera appel à multiplications complexes, éventuellement à des opérations de copies des coefficients (de cout proportionnel à N) ainsi qu'à la transformée de Fourier de ses coefficients pairs et à la transformée de Fourier de ses coefficients impairs, concernant toutes les deux échantillons. Notons le nombre de multiplications complexes pour calculer la transformée de Fourier de échantillons. On a donc la relation de récurrence :

**Deuxième étape** : étudier le "cas trivial", problème de plus petite taille que l'on ne peut pas découper.

On a supposé que était une puissance de 2, soit tel que

Chaque appel récursif divise le nombre d'échantillon par 2, jusqu'à ce que l'on atteigne le calcul de la transformée de Fourier d'un échantillon. Avec les notations précédentes, on a alors : , il n'y a qu'à recopier le résultat ! Pour notre calcul on va donc prendre .

**Calcul de la complexité de l'algorithme** :

Or : et d'où :

Cette estimation de complexité ne prend pas en compte les additions complexes, qui ont un temps calcul négligeable devant celui des les multiplications complexes.

Finalement, l'algorithme de la transformée de Fourier rapide permet un calcul en à la place d'un calcul en ! Cela représente un gain de temps considérable !

Supposons qu'on choisisse de travailler avec des paquets de échantillons et qu'on utilise une machine capable d'effectuer 1 million de de multiplications complexes par secondes. Alors qu'un calcul naïf prendrait , avec la FFT, on atteindra une vitesse de .

C'est 100 fois plus rapide… et quand on sait que pour une musique échantillonnée à 44100 Hz, on a environ 43 paquet par secondes de musique, on comprend que les traitements que l'ont va effectuer grâce à la FFT serait impossible sans elle ! Il faudrait quasiment 3 heures pour traiter une musique de 4 minutes…

## L'algorithme de Transformée de Fourier Rapide (FFT), version itérative

On peut encore améliorer l'efficacité de notre algorithme. En effet, les appels récursifs ont un cout en temps de calcul, et surtout en occupation mémoire. On va donc déterminer comment effectuer notre FFT de façon itérative.

Si au fur et à mesure des appels récursifs dans le calcul précédent, on indice pour les coefficients impairs et pour les coefficients pairs, lorsque l'on arrive à un cas trivial, il est repéré par une suite de indices. Pour mieux visualiser ce qui va suivre, prenons un exemple avec , les appels récursifs sont résumés sur le schéma suivant :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 000 | 010 | 100 | 110 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 001 | 011 | 101 | 111 |

|  |  |
| --- | --- |
|  |  |
| 011 | 111 |

|  |  |
| --- | --- |
|  |  |
| 001 | 101 |

|  |  |
| --- | --- |
|  |  |
| 000 | 100 |

|  |  |
| --- | --- |
|  |  |
| 010 | 110 |

|  |
| --- |
|  |
| 000 |

|  |
| --- |
|  |
| 100 |

|  |
| --- |
|  |
| 010 |

|  |
| --- |
|  |
| 110 |

|  |
| --- |
|  |
| 001 |

|  |
| --- |
|  |
| 101 |

|  |
| --- |
|  |
| 011 |

|  |
| --- |
|  |
| 111 |

L'action de "recombinaison" qui clôt la version récursive de l'algorithme correspond en fait à un parcours, des feuilles vers la racine et niveau par niveau. Pour obtenir une version itérative, on peut donc tout d'abord réorganiser notre tableau contenant les coefficients dans l'ordre qui figure sur la dernière ligne du schéma précédent, puis recombiner les éléments 2 à 2 pour retrouver l'avant dernière ligne, puis de nouveau 2 à 2 pour retrouver la seconde ligne, et encore 2 à 2 pour calculer la transformée de Fourier souhaité, en première ligne.

On a noté sous chaque coefficient son indice codé en binaire, pour le tri des coefficients, on peut remarquer qu'il suffit d'inverser les chiffres de ce code. Exemple : le coefficient qui était à l'origine en position termine en position .

On voit qu'on utilisera les coefficients exponentiels pour qu'on a tout intérêt à enregistrer dans un tableau, initialisé lorsque l'on ouvre notre outil de compression !

Pour la transformée de Fourier discrète inverse, l'algorithme est le même, seuls les coefficients exponentiels sont à modifier en et il faut ajouter une division finale par .

# Implémentation de l'algorithme de FFT

## Manipuler de très grands entiers

Lors du passage des algorithmes papier au code, nous avons obtenu à plusieurs reprises la même erreur : un **overflow sur le type Integer**. Non habitués à travailler avec des entiers supérieurs à en valeur absolue, nous avions en effet utilisé le type usuel Integer.

Or celui-ci est codé sur 32 bits et ne peut donc contenir de tels entiers. Après quelques recherches dans le manuel de référence d'ADA95, on a résolu le problème avec l'utilisation du **type**  qui permet de stocker des entiers compris dans l'intervalle . On a aussi déclaré un sous type qui correspondant à la partie positive de cet intervalle. Et surtout, on a pratiqué des tests unitaires poussés sur toutes les fonctions, surtout sur le package qui contient plusieurs outils que l'on a codé pour manipuler les nombres, que l'on va évoquer par la suite.

## Lecture et écriture d'un fichier WAVE

La structure commune à tous les fichiers WAVE est la suivante. Le fichier démarre par un entête sur 44 octets ou plus contenant les informations suivantes :

|  |  |  |  |
| --- | --- | --- | --- |
| Octets n° | Item | Signification | Valeur attendue\* |
| 1 à 4 | mot "RIFF" | Indication sur le format de fichier | 52 49 46 46 |
| 5 à 8 | File\_Size | Taille du fichier moins ces 8 premiers octets |  |
| 9 à 12 | mot "WAVE" | Indication sur le format de fichier | 57 41 56 45 |
| 13 à 16 | mot "fmt " | Indication sur le format audio | 66 6D 74 20 |
| 17 à 20 | Bloc\_Size | Nombre d'octets restant dans l'en tête, jusqu'au mot "data" |  |
| 21 à 22 | Audio\_Format | Numéro de format du fichier | 01 00 |
| 23 à 24 | Nb\_Canaux | Nombre de canaux | 01 00 |
| 25 à 28 | Frequence | Fréquence d'échantillonnage en Hz |  |
| 29 à 32 | Octets\_par\_sec | Nombre d'octets lu par secondes, soit :  Frequence \* Octet\_par\_bloc |  |
| 33 à 34 | Octets\_par\_bloc | Nombre d'octets par bloc (= au précédent /8 en mono) |  |
| 35 à 36 | Bits\_par\_echantillon | Nombre de bits par échantillon |  |

\* pour un WAVE mono non compressé en hexadécimal

Après ces 34 octets viennent éventuellement des informations complémentaires, facultatives. Puis le bloc de données démarre par le mot "data" (64 61 74 61) et le nombre d'octets de données, sur 4 octets.

Pour la lecture et l'écriture dans les fichiers, on a choisi d'utiliser le package , qui permet comme son nom l'indique un **accès direct** et non obligatoirement séquentiel. On peut ainsi accéder à l'échantillon souhaité sans lire tous les autres auparavant, ou à une information de l'entête sans le lire depuis le début. On a fait le choix de **l'instancier avec le type**  et non le type comme on pourrait s'y attendre. Chaque character étant codé par son code ASCII sur un octet, l'idée était de pouvoir lire **le fichier octet par octet**.

On a ensuite développé les fonctions , , , et (pour Integer et Long\_Long\_Integer) au nom assez explicite, pour pouvoir passer des valeurs entières naturelles des octets lu dans le fichier, aux valeurs entières signées (en complément à 1) codées sur plusieurs octets sur lesquelles on peut travailler.

On a commencé par développer une fonction afin de vérifier que le fichier que l'on considère a bien le format attendu, en faisant les vérifications qui figurent dans la colonne de droite du tableau précédent. Puis on a développé les fonctions permettant d'accéder aux informations intéressantes de l'entête : fréquence d'échantillonnage, octets par échantillon, position du début du bloc de données, nombre de frames à récupérer dans ce dernier (on a choisi de travailler avec des frames de 512 échantillons). Ces fonctions ont également passée des tests unitaires, avec divers fichiers généré à partir d'un logiciel d'audio, et parfois modifié avec un éditeur hexadécimal.

L'idée était toujours de développer de façon la plus propre possible, car les algorithmes les plus importants du projet devaient s'appuyer sur des bases fiables.

Enfin, on a pu écrire la fonction de lecture, dont voici le code somme toute assez simple une fois les outils précédents développés. Elle prend en argument l'adresse du fichier wav, et le numéro de la frame qui nous intéresse et retourne uniquement cette frame sous la forme d'un tableau de , de type indicé de 0 à 511.

On utilise les fonctions développées précédemment pour déterminer la position du début du bloc données (), puis pour se placer au niveau de la N-ième frame. On lit ensuite chaque échantillon (boucle sur ), en sommant octet par octet (boucle sur )

FUNCTION Lire\_Frame(Adresse : IN String ; Numero : IN natural) RETURN FFT.Tab\_TQ IS

Fichier : File\_Type;

Depart : CONSTANT Natural := Amorce\_Data\_Bloc (Adresse);

Bits : CONSTANT Natural := Nb\_Bits\_Par\_Echantillon (Adresse);

Pas : CONSTANT Natural := Nb\_Bits\_Par\_Echantillon (Adresse)/8;

Res : FFT.Tab\_TQ := (others => 0);

Buffer : Character;

BEGIN

Open(Fichier,In\_File,Adresse);

FOR J IN Res'RANGE LOOP

FOR K IN 1..Pas LOOP

Read(Fichier,Buffer,Count(Depart+(Numero-1)\*Frame\_Size\*Pas+Pas\*J+(K-1)));

Res(J) := Res(J)+Long\_Long\_Integer(Character'Pos(Buffer))\*Long\_Long\_Integer(16\*\*(2\*(K-1)));

END LOOP;

END LOOP;

Close(Fichier);

RETURN Res;

END Lire\_Frame;

Pour l'écriture, nous avons développé deux fonctions. La première initialise le fichier en quelque sorte : elle le créé, et rempli l'entête, suite à son appel, la taille du bloc de données vaut 0. On utilise ensuite la procédure , qui prend en argument le nom du fichier, un et insère cette nouvelle frame à la fin du fichier. Elle fonctionne sur le même principe que , en utilisant les fonctions développées précédemment pour se placer au bon endroit dans le fichier.

Nous avons testées ces trois fonctions individuellement, puis toutes les 3 ensembles, en procédant à des copies de fichier wave.

## Quantification des échantillons

Pour chaque frame extrait d'un fichier wave, les 512 échantillons sont quantifiés sur un certain nombre de bits (souvent 16 ou 24 bits).En leur appliquant la transformée de Fourier, on va obtenir 256 échantillons complexes (512, divisé par 2 grâce aux propriétés de symétrie évoquées dans le II) qui eux ne sont pas échantillonnés (ils sont de type float). Pour pouvoir les enregistrer dans un nouveau fichier, on a développé une fonction (F comme fréquentiel) qui les quantifie sur un nombre de bits spécifié (inférieur à 16 ou 24 bits pour que cela soit effectivement une compression).

Pour cela, et afin d'avoir une quantification la plus précise possible, au cours de la transformée de Fourier, on tient à jour une variable , qui contient le maximum des parties réelles et imaginaires des 256 coefficients, que l'on transmet ensuite à la fonction . Ainsi, le coefficient est codé par le plus grand entier codable sur bits, et on quantifie en utilisant tous les valeurs disponibles sur bits.

On obtient ainsi 256 échantillons complexes quantifiés. Si on leur applique la transformée de Fourier inverse, on obtient à nouveau 512 échantillons non quantifiés. On a ainsi développé une autre fonction . Celle-ci est légèrement différente de la précédente.

En effet, cette fois-ci, on ne veut pas quantifier sur la pleine échelle, car si on le faisait, on gommerait les différences d'amplitude entre les différentes frames. Les passages calmes se retrouveraient avec une amplitude maximale au même niveau que les passages avec plus de dynamiques. C'est pourquoi, lors de la transformée de Fourier Discrète, on stocke un ratio, qui est égal à l'amplitude maximal de la frame, sur l'amplitude maximale du fichier, que l'on utilise ensuite dans .

## Transformée de Fourier

On a suivi l'algorithme itératif proposé précédemment. La fonction reçoit la frame à traiter (, le nombre de bits par échantillon dans le signal d'origine afin de calculer le ratio évoqué précédemment, et un tableau contenant tous les facteurs exponentiels, qui est généré par une autre fonction ().

Le fait de rendre l'algorithme récursif initial en algorithme itératif a aussi pour intérêt de réduire la complexité mémoire : pour cette fonction, on n'utilise que deux tableaux : un tableau de travail, dans lequel on fait les combinaisons linéaires successives pour calculer chaque coefficient du spectre échantillonné et un tableau dans lequel on les stocke, qui fait partie du résultat.

FUNCTION TFD (Coeffs : IN Tab\_TQ ; Nb\_bits\_origine : IN natural ; Expo : IN Tab\_Exp) RETURN Resultat\_TFD IS

Travail : Tab\_F;

Res : Resultat\_TFD;

Ampli\_Max : Long\_Long\_Integer := 0;

BEGIN

-- calcul du ratio d'occupation en amplitude

Res.Maximum :=0.0;

FOR I IN Coeffs'RANGE LOOP

IF ABS(Coeffs(I)) > Ampli\_Max THEN

Ampli\_Max := ABS(Coeffs(I));

END IF;

END LOOP;

Res.Occupation := (Ampli\_Max,Big\_Natural(2\*\*(Nb\_Bits\_Origine-1)-1))

FOR K IN Res.Tab'RANGE LOOP

-- première itération pour remplir le tableau de travail

FOR I IN Travail'RANGE LOOP

Travail(I) := Float(Coeffs(2\*I)) + Float(Coeffs(2\*I+1))\*Expo(1,K);

END LOOP;

-- et puis on itère jusqu'à n'avoir plus qu'une valeur

FOR J IN 1..Parametre\_Frame LOOP

FOR I IN 0..Frame\_Size/(2\*\*(J+1))-1 LOOP

Travail((2\*\*(J))\*I) := Travail((2\*\*J)\*I) + (Expo(J,K) \* Travail(((2\*\*J)\*I+2\*\*(J-1))));

END LOOP;

END LOOP;

Res.Tab(K) := Travail(0);

IF Abs(Travail(0).Re) > Res.Maximum THEN

Res.Maximum := Abs(Travail(0).Re);

END IF;

IF ABS(Travail(0).Im) > Res.Maximum THEN

Res.Maximum := Abs(Travail(0).Im);

END IF;

END LOOP;

RETURN Res;

END TFD;

L'algorithme de transformer de Fourier inverse est très similaire, Il suffit juste d’enlever le calcul du ratio d’occupation et d’ajouter une division par 512 sur chaque coefficient une fois l’algorithme fini.

# Conclusion

Après avoir effectué ce projet, il en ressort un bilan très positif. Nous avons fait beaucoup d’erreurs dans la réalisation des différentes taches, ce qui nous permettra d’être plus efficaces sur nos futurs projets. En effet, nous avons mis longtemps à définir ensemble la totalité du programme en entrant trop souvent dans des points de détails futiles, ce qui nous a empêché d’avancer séparément sur nos parties. D’autre part nous avons appris, après avoir effacé quelques centaines de lignes de code, à bien vérifier au papier nos solutions. En effet, pour la lecture du fichier nous avons, dans un premier temps, pensé à Ada.Sequential\_IO, qui s’est avéré par la suite être tellement peu approprié à notre projet que l’acteur audio\_io à été totalement réécrit avec Ada.Direct\_IO. Et ce n’est pas la seule erreur rencontrée dans notre projet. Cependant, au fur et à mesure des erreurs, nous avons appris à les éviter.

De plus, le fait d’avoir été conseillé sur des outils de travail comme github par notre encadrant à été un vrai plus à la réalisation de notre projet. Si nous n’en avions pas vu l’intérêt réel au début, il s’est avéré que la réalisation de ce devoir a été grandement facilitée grâce à cet outil qui ne manquera pas de nous servir lors de nos prochains travaux.

L’inertie inhérente au démarrage d’un projet nous a aussi beaucoup surpris. Nous avons mis un temps certain pour installer nos protocoles de travail, comprendre le fonctionnement de github et appréhender ce que nous allions devoir faire dans notre code. Puis, quand nous avons démarré plus sérieusement le projet, les avancées ont été beaucoup plus rapides que nous ne l’avions estimé. La mise en place finale du code n’est finalement que la partie immergée de l’iceberg que représente le travail effectué.

Nous avions aussi pensé à effectuer une compression entropique par la méthode d’Hoffmann, qui aurait eut de meilleurs résultats et des algorithmes très intéressants à présenter. Cependant si une ébauche de code a été effectuée, nous en étions déjà à une centaine d’heure et près de 2 000 lignes écrites sur le projet, et nous avons manqué de temps pour le réaliser.

Au final, ce projet nous aura apporté tant sur des compétences de travail en équipe, que de gestion de projet et d’algorithmique, ce qui nous servira très certainement lors de nos prochains projets, que ce soit en informatique ou dans d’autres secteurs.

En vous remerciant de nous avoir lus et encadrés,

Etienne THIERY et Bastien MICHEAU

# Bibliographie

En papier

Cours d'analyse Signal en 2ème année "Modélisation Informatique et Communications" à l'INSA de Toulouse (Polycopié par Danièle FOURNIER, Violaine ROUSSIER-MICHON, Martin AIME et Fabien MONFREDA)

Voici l'implémentation réalisée de la FFT en ADA, qui reçoit en argument le tableau des coefficients , ainsi que celui des coefficients exponentiels et retourne le tableau des premiers coefficients (les autres sont obtenus par symétrie)

Sur le Web

[Open Classroom - Ada - Les Fichiers](http://fr.openclassrooms.com/informatique/cours/apprendre-a-programmer-avec-ada/les-fichiers-6)

[Spécifications de ada.Direct\_IO](http://www.infeig.unige.ch/support/ada/gnatlb/a-direio.html)