Rod Quentin, Dagier Thomas, Pertuzati Gustavo 1ere ITI

# Rapport de Mathématiques

L'objectif de ce Travail Pratique est d'approfondir les notions de transformation de Fourrier vues en classe (Réels et Complexes). Pour cela, nous utilisons les fonctions de transformations de Fourrier rapides (FFT) disponibles sur Python dans la librairie Numpy.

Le premier objectif est de décoder un signal sonore qui a donc des valeurs réelles. Nous travaillons alors avec les fonctions IFFT et IRFFT. Avec cette transformation, la liste de Ck contient uniquement les k positifs. Python s'occupe seul des coefficients négatifs en calculant le conjugué complexe.

Python IDFT : $$S[n] = \frac{1}{M}\sum_{k=0}^{M-1} C[k] \times e^{i2\pi\frac{nk}{M}}$$

Python DFT: $$C[k] = \frac{1}{M}\sum_{n=0}^{M-1} S[n] \times e^{-i2\pi\frac{nk}{M}}$$

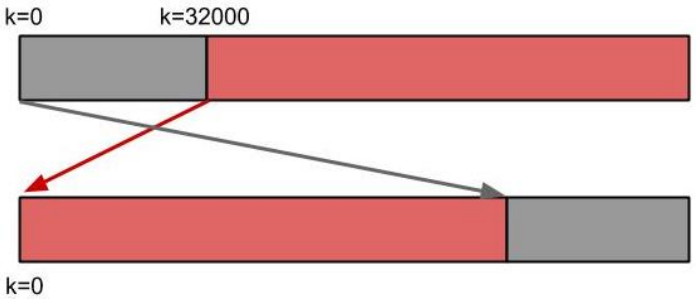
## Deuxième partie

Dans un premier temps, nous nous sommes répartis chacun la réalisation d'une fonction. Selon nous, les candidats les plus probables sont : Le miroir, la translation et le saute-mouton. Afin de vérifier le fonctionnement de chacune, nous avons effectué les transformations sur les fichiers de tests. Comme nous ne trouvions pas le message caché et que les fonctions de déphasage et de saute-mouton étaient validées avec les tests, nous avons concentré tous nos efforts sur les fonctions de miroir et de translation. Ce dernier, s'avère être le traitement utilisé.

La translation : Pour effectuer la translation inverse, il faut savoir à partir de quel k elle a été effectuée. On sait que chaque Ck correspond à k*FreqSignal. Également, la fréquence d'échantillonnage est de 44100Hz. Nous calculons la fréquence du signal :

$$FreqSignal = \frac{freqEchantillonage}{nbrEchantillons} = \frac{44100}{\text{len(list\_ck\_sample)}*2} = \frac{44100}{176257*2} = 0.12510Hz.$$

Nous multiplions la longueur par 2 car nous avons que la moitié des Ck avec l'IFFT. Il suffit de résoudre l'équation 4000 = k*FreqSignal. Soit $k = \frac{4000}{0.12510} = 31974$. Nous avons donc translaté tous les éléments sur la gauche. Les Ck du début du fichier ont été mis à la suite. Voici une illustration :




Le message caché est :

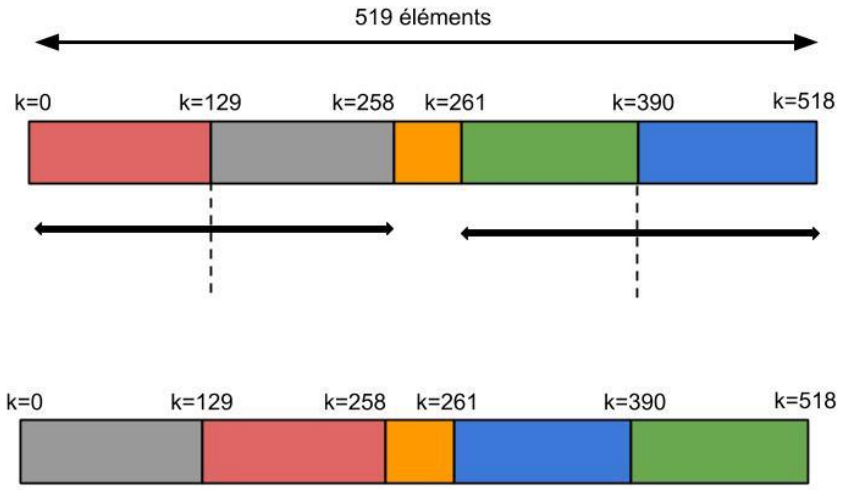
"<écho carrément flippant> De toute façon vous êtes complètement nuls vous ne trouverez jamais le message caché <rires carrément flippant>"

Troisième partie

Troisième partie

Nous avons commencé par regarder le message et observer sa forme. Durant le cours de maths nous avons remarqué que la forme d'un signal "miroiré" correspondait avec celle de notre signal à transformer (selon les exemples qui nous ont été montré). De ce fait nous avons, dans un premier temps, concentré nos efforts sur le miroir. Ici, la difficulté réside dans le fait qu'il faut miroirer k et –k car c'est un signal complexe.

En d'autres termes, comme la fréquence de miroir est à ¼ de la longueur du signal on peu miroirer aisément les valeurs comprises entre 0 et la moitié (en admettant que l'on ne bouge ni 0 ni la fréquence miroir ni les valeurs qui auraient remplacé celles que l'on ne veut pas bouger). Pour le reste du signal c'est la même chose mais au lieu de travailler avec des indices négatifs on reporte le signal pour que la fréquence miroir soit à ¾ de la longueur du signal (ou –¼ ).




En pensant que notre fonction marchait mais que nous ne trouvions pas le résultat, nous nous sommes tournés vers le "racinateur d'amplitude" et le "diviseur de phase" qui ont tout de suite marché. Petit à petit, nous avons travaillé sur toutes les autres fonctions en restant persuadés que notre filtre était un miroir. Le fichier de données contenant des erreurs avant la transformation, nous avons de nouveau testé le bon fichier avec le miroir en utilisant une "deep copy" sur le signal d'origine et finalement réussi à trouver notre "MESSAGE CAHCE AVEC ERERURS".