# IT_124 - TP II : les séries de Fourier complexes avec la FFT.

## Rappels théoriques

Rappel général de la DFT

On rappelle les formules de la DFT (Transformée de Fourier discrète) et de la IDFT (inverse de la DFT) pour un signal donné par une liste $S$ de longueur $M$. Les coefficients de Fourier $c_k$ sont calculés pour des $k$ positifs et négatifs, a priori, mais on a vu qu'on peut se ramener au cas d'indices positifs car $c_{-k} = c_{M-k}$.

$$c_k = \frac{1}{M} \sum_{n=0}^{M-1} S[n] \cdot e^{-i2\pi \frac{nk}{M}}. \tag{DFT}$$

$$S[n] = \sum_{k=0}^{M-1} c_k \cdot e^{i2\pi \frac{nk}{M}} \tag{IDFT}$$

On rappelle également que si le signal (la liste) $S$ ne contient que des valeurs rélles, alors $c_{-k} = \overline{c_k}$ (le conjugué complexe). Dans ce cas, l'amplitude $\chi_k$ de la $k$-ème harmonique est donnée par $2|c_k| = 2|c_{-k}|$, et son déphasage est donné par l'angle de $c_k$. Si $S$ contient des valeurs complexes, alors les modules et angles de $c_k$ et $c_{-k}$ peuvent être différents.

La DFT en Python 3

En Python 3, la librairie `numpy` permet de calculer la DFT et la IDFT de manière rapide (la célèbre FFT - Fast Fourier Transform), mais avec de petites différences dans les formules (sinon ça serait trop facile, n'est-ce pas). Soit donc $S$ une liste de nombre complexes de longueur $M$. Alors la fonction `numpy.fft.fft(S)` retourne une liste $C$ de même taille que $S$ dont les entrées sont :

$$C[k] = \sum_{n=0}^{M-1} S[n] \cdot e^{-i2\pi \frac{nk}{M}}. \tag{PYTHON-DFT}$$

Comme $C[-k] = C[M-k]$ en Python, on a directement accès aux indices négatifs. Notez la différence avec (DFT) : il n'y a *PAS* le facteur $1/M$, chaque $C[k]$ calculé par la FFT de Python est égal au 'vrai' coefficient $c_k$ multiplié par $M$. En effet, le facteur se trouve dans la transformée inverse, que l'on appelle par `numpy.fft.ifft(C)`, qui calcule ceci :

$$S[n] = \frac{1}{M} \sum_{k=0}^{M-1} C[k] \cdot e^{i2\pi \frac{nk}{M}} \tag{PYTHON-IDFT}$$

Ces deux fonctions sont inverses l'une de l'autre, à savoir `numpy.fft.ifft(numpy.fft.fft(S))` retourne $S$ (à des approximations de calcul près). Python est capable d'utiliser des nombres complexes, *mais utilise le symbole j plutôt que i* : pour entrer $2 - 3i$, on écrit `2 -3j`, et python le traitera en nombre complexe automatiquement. Afin d'éviter les confusions avec une éventuelle variable appelée `j` (mauvaise idée, ceci dit), il vaut mieux noter $1 + i$ par `1+1j` que par `1+j`. La fonction `numpy.abs(c)` renvoie le module du nombre complexe `c` et `numpy.angle(c)` son angle.

Si la liste $S$ ne contient que des nombres réels, on peut utiliser les fonctions `numpy.fft.rfft(S)` et `numpy.fft.irfft(C)` qui utilisent le fait que $C[-k] = \overline{C[k]}$ et ne calculent donc que la moitié des coefficients.

**Attention :** `numpy.fft.fft`, `numpy.fft.ifft`, etc, retournent des 'numpy array', qui sont des listes d'un type particulier qui se manipulent un peu différemment. Par exemple, si `A, B` sont deux 'numpy array', alors `A + B` représente la somme terme à terme et non la concaténation des listes. De même, `A[10:20] = 0` met les entrées d'indices de 10 à 19 à 0 (ce qu'on ne peut pas écrire aussi simplement pour les listes standard). Ceux qui craignent la confusion peuvent transformer toutes les 'numpy array' en liste en écrivant `list(A)`.

Pour plus de détails, entre autres les différences lorsque $M$ est pair ou impair et de ce qui se passe 'au milieu' de $C$ avec `fft` et `rfft`, se référer à :

`https://docs.scipy.org/doc/numpy/reference/routines.fft.html#module-numpy.fft`

Les fréquences

Si $S$ est un signal temporel samplé à fréquence d'échantillonage $f_s$ et de longueur $M$, sa longueur temporelle (en secondes) est donc de $M/f_s$. On considère que c'est la période du signal, et donc sa fréquence est de $f_s/M$. Donc, $c_k$ et $c_{-k}$ sont les coefficients correspondant à la fréquence $\frac{k \cdot f_s}{M}$.

---

**Travail demandé**

**Première partie – entrainement**

Refaire le TP1 en utilisant les routines `numpy.fft.rfft(S)` et `numpy.fft.irfft(C)` (ou, si vous préférez, `numpy.fft.fft(S)` et `numpy.fft.ifft(C)`). Les fichiers audio sont
`message_cache_noised0.wav`
`message_cache_noised1.wav`
`message_cache_noised2.wav`
`message_cache_noised3.wav.`
Ils sont samplés à $8192Hz$, durent $1s$, contiennent du bruit en dessous de $90Hz$ et en dessus de $2000Hz$, ainsi que 6 sinus de fréquences entières aléatoires entre $100Hz$ et $2000Hz$ et de grande amplitude. La FFT de Python vous retournera direcement une liste de longueur 8192 (ou la moitié si vous utilisez `rfft`) des $c_k$, il faudra donc en mettre certains à 0 avant d'appliquer la IFFT.

**Deuxième partie – des traitements sur fichiers audio (à valeurs rélles)**

Chaque groupe recevra deux fichiers audio de même longueur (contenant bien évidemment un message caché) qui ont subi un des traîtements suivants :

- Un 'miroir de fréquences' : on choisit la fréquence $f_{\text{mirror}} = 6000Hz$, et on fait un miroir autour d'elle : le coefficient de la fréquence $f_{\text{mirror}} - a$ est échangé avec celui de la fréquence $f_{\text{mirror}} + a$. Donc, le coefficient correspondant à la fréquence de $3000Hz$ est échangé avec celui correspondant à la fréquence de $9000Hz$, celui de $5500Hz$ avec celui de $6500Hz$, etc. (Formellement, on a 2 coefficients correspondant à $3000Hz$, celui avec un indice positif et celui avec un indice négatif, mais si le signal est réel, comme dans le cas d'un signal audio, on peut utiliser `numpy.fft.rfft` et faire comme s'il n'y en avait qu'un seul.)
- Un 'changement de note linéaire' (linear pitch shift) : on choisit une fréquence $f_{\text{translation}} = 4000Hz$, et le coefficient de la fréquence $f$ est envoyé sur celui de la fréquence $f + f_{\text{translation}}$. Donc le coefficient correspondant à la fréquence de $1000Hz$ devient celui de la fréquence de $5000Hz$, celui de $100Hz$ devient celui de $4100Hz$, etc.
- Un 'échange amplitude/déphasage' entre deux signaux : on prend deux signaux (de même longueur), et sur chaque coefficient de Fourier (complexe) on prend l'amplitude du premier et le déphasage du second pour recréer un signal, et réciproquement pour recréer un deuxième signal.
- Un 'saute-mouton mélangeur de fréquences' : on prend deux signaux (de même longueur), on prend successivement 10 coefficients de Fourier du premier, puis 10 du second, puis 10 du premier, etc, pour recréer un signal. On prend ensuite 10 du second, puis 10 du premier, puis 10 du second, etc, pour recréer un deuxième signal.

Il faudra trouver quel est le traîtement infligé aux deux signaux et leur faire subir le traîtement inverse afin de retrouver le message caché (une fois de plus). Les fichiers audio sont en mono, samplés à $44100Hz$ et de format `wav`. Il faudra donc utiliser la librairie wave et les fonctions données dans le premier TP pour les transformer en liste (et réciproquement). Par contre, leur durée n'est *pas* d'une seconde.

**Troisième partie – des traitements sur des signaux à valeurs complexes**

On prend des signaux complexes, qui se trouvent être des courbes écrivant un certain message (caché, comme de bien entendu). Ici, la fréquence réelle du signal importe peu, car il ne s'agit pas vraiment de signaux temporels, la vitesse à laquelle la courbe est parcourue n'a pas d'importance, ce qui l'est, c'est la phrase qui finit par être écrite. On peut tout de même faire des DFT (et IDFT) sur ces signaux, et les traîtements qu'on leur fait subir sont plutôt décrits comme étant directement appliqués sur les entrées et indices de la liste `C[k]`. Chaque groupe recevra un fichier `.data` contenant les coordonnées des points d'une courbe qui a subi un des traîtements suivants :

- Un 'miroir de fréquences', comme dans le cas audio. Cette fois-ci, le miroir se fait autour de l'indice correspondant à 1/4 de la longueur de la liste `C`. Notons `k = int(M/4)`, où `M` est la longueur de la liste,

alors `C[k]` reste sur `C[k]`, `C[k-1]` est échangé avec `C[k+1]` lorsque `k>0`, etc. Exception : `C[0]` reste sur `C[0]`. Attention : comme le signal a des valeurs complexes, il faut prendre en compte les indices négatifs, qui sont donc mis en miroir autour de `-k`.

- Un 'changement de note linéaire' (linear pitch shift), comme dans le cas audio : on pose `L = int(0.1*M)`, où `M` est la longueur de la liste, et ensuite (pour les indices positifs) `C[k]` est envoyé sur `C[k+L]`. Attention : comme le signal a des valeurs complexes, il faut comprendre ce qui se passe avec les indices négatifs.
- Un 'filtre passe-haut linéaire' : le coefficient `C[k]` est multiplié par `0.1 + 0.9*numpy.abs(k)/int(M/2)`, où `M` est la longueur de la liste. (En gros, on multiplie terme à terme avec la liste `numpy.linspace( 0.1 , 1 , int(M/2) )` lorsque `k` est positif.)
- Un 'racinateur d'amplitude' : le module du coefficient `C[k]` est mis à la racine carrée.
- Un 'diviseur de phase' : l'angle du coefficient `C[k]` est divisé par 2.

Chaque traîtement rend la phrase illisible. Il faudra donc trouver quel est ce traîtement et faire subir l'inverse au signal pour trouver (encore une fois) le message caché. Pour celà, il faudra bien évidemment se rappeler comment on affiche des choses avec `matplotlib`. Les deux fonctions suivantes peuvent servir (pour les utiliser on a bien sûr besoin d'avoir écrit `import numpy` quelque part avant) :

`numpy.real(c)` retourne la partie réelle d'un nombre complexe `c`,
`numpy.imag(c)` retourne la partie imaginaire d'un nombre complexe `c`.

Donc, par exemple, si $S$ est une liste de nombre complexes,

```
x , y = [ numpy.real(c) for c in S ] , [ numpy.imag(c) for c in S ]
```

retourne les listes des coordonnées horizontales et verticales.

La librairie `pickle` permet de lire et écrire des fichiers de données contenant des valeurs de variables. On l'importe avec `import pickle`. Pour lire un fichier nommé `donnees.data` et mettre les valeurs dans la variable `s` :

```
with open("donnees.data", "rb") as filehandle:
    s = pickle.load(filehandle)
```

Pour sauver les valeurs de la variable `s` dans un fichier nommé `donnees.data` :

```
with open("donnees.data", "wb") as filehandle:
    pickle.dump( s , filehandle)
```