# Travail pratique - Les transformées de Fourier

Mathématiques en technologie de l'information

## 1 Objectif

Implémenter de façon naïve les transformées de Fourier discrètes en deux dimensions avec pour but:

1. Tenter de comprendre leur utilité en traitement d'image.
2. Tenter de comprendre leur utilité en compression d'image.

## 2 Les transformées de Fourier en plusieurs dimensions

Nous avons vu pendant le cours la définition de la transformée de Fourier discrète en une dimension donc nous n'allons pas trop insister sur le sujet ici mais reproduire le résultat principal.

Soit un signal $f[n]$ discret et de longueur $N$ ($n$ va de 0 à $N-1$), nous pouvons écrire sa transformée, $\hat{f}[k]$ ($k$ allant de $0,..,N-1$) de Fourier comme [1]

$$\hat{f}[k] = \sum_{n=0}^{N-1} f[n] e^{-\frac{2\pi i n k}{N}}. \tag{1}$$

A l'inverse, on peut calculer la transformées de Fourier discrète inverse de $\hat{f}[k]$ comme

$$f[n] = \frac{1}{N} \sum_{k=0}^{N-1} \hat{f}[k] e^{\frac{2\pi i k n}{N}}, \tag{2}$$

ce qui permet de déterminer le signal d'origine à partir de sa transformée de Fourier.

On peut généraliser la transformée de Fourier pour des signaux en plus d'une dimension. Ici, on va s'intéresser aux signaux bidimensionnels. Un tel signal peut représenter par exemple les niveaux de gris d'une image. On a que la transformée de Fourier pour un signal qui dépend de deux coordonnées, $f[n_1, n_2]$, $n_1 = 0,..,N_1-1$, $n_2 = 0,..,N_2-1$, a une transformée de Fourier discrète, $\hat{f}[k_1, k_2]$, $k_1 = 0,..,N_1-1$ et $k_2 = 0,..,N_2-1$, donnée par

$$\hat{f}[k_1, k_2] = \sum_{n_1=0}^{N_1-1} e^{-\frac{2\pi i n_1 k_1}{N_1}} \left( \sum_{n_2=0}^{N_2-1} f[n_1, n_2] e^{-\frac{2\pi i n_2 k_2}{N_2}} \right). \tag{3}$$

1. On représente $f[n]$ et $\hat{f}[k]$ comme des listes (ou vecteurs) de longueur $N$.

De cette formule, on peut déduire que faire la transformée de Fourier en deux dimensions n'est rien d'autre que faire la transformée de Fourier uni-dimensionnelle dans chacune des dimensions séparément.

De même la transformée de Fourier inverse à deux dimensions s'écrit

$$f[n_1, n_2] = \frac{1}{N_1 \cdot N_2} \sum_{k_1=0}^{N_1-1} e^{\frac{2\pi i n_1 k_1}{N_1}} \left( \sum_{k_2=0}^{N_2-1} \hat{f}[k_1, k_2] e^{\frac{2\pi i n_2 k_2}{N_2}} \right). \tag{4}$$

Souvenez-vous qu'on a vu que la transformée de Fourier discrète et son inverse peuvent s'écrire sous la forme d'un produit matrice vecteur. Cela peut vous aider à rendre votre code plus performant en utilisant les fonctionnalités de la librairie `numpy`.

On peut donc faire une analyse de Fourier d'une image presque aussi facilement qu'on le ferait pour un signal unidimensionnel.

# 3 Travail à réaliser

Dans ce travail, vous avez un cahier des charges relativement peu précis. Vous devez vous débrouiller pour atteindre un but fixé, mais le chemin est assez peu balisé. C'est à dessein. On peut discuter oralement de chaque partie si vous voulez vous assurer de ce que vous devez faire, mais je ne veux pas trop vous cadrer et vous laisser vous débrouiller.

En résumé, il faudra à l'aide des transformées de Fourier discrètes filtrer une image. Puis écrire un algorithme permettant de la compresser/décompresser (avec pertes).

## 3.1 Implémenter les fonctions `tfd()` et `tfdi()`

Il s'agit d'écrire une fonction vous même pour calculer la transformée de Fourier discrète à une, puis à deux dimensions (`tfd()` et `tfd2()` respectivement). Pensez à écrire un programme pour **valider** vos transformées de Fourier.

Pour ce faire utilisez des fonctions dont vous pouvez facilement calculer analytiquement les transformées de Fourier (les sinus/cosinus s'y prêtent particulièrement).

Vous pouvez également comparer vos résultats avec les fonction `fft()` et `fft2()` de python.

Dans un deuxième temps implémentez les transformées de Fourier inverses en une et deux dimensions (`itfd()` et `itfd2()` respectivement). Assurez-vous que ces fonctions marchent bien. Un bon test est que `tfdi(tfd(signal)) == signal`. En d'autres termes, la transformée de Fourier inverse de la transformée de Fourier d'un signal, doit donner le signal lui-même.

Vous pouvez également comparer vos résultats avec les fonctions `ifft()` et `ifft2()` de `numpy`.

## 3.2 Filtrage de bruit

Sur `cyberlearn`, vous trouverez dans un premier temps **une seule image** très bruitée nommée `cache.png`. Tapie dans le bruit, une personnalité est cachée prête à bondir. Il s'agit ici de trouver laquelle.

Vous devrez donc utiliser vos fonctions `tfd2()` et `tfdi2()` pour filtrer les images.

Histoire de vous familiariser avec le filtrage, créez un signal simple contenant deux fréquences. Un exemple pourrait être d'échantillonner

$$f(t) = 2.3 \cdot \sin(2\pi t) + 0.1 \cdot \sin(10\pi t), \tag{5}$$

pour $t = [0, 1]$. Quelle est la période de ce signal? Calculez la transformée de Fourier de ce signal, puis mettez à zéro le coefficient correspondant à la plus haute fréquence dans le résultat obtenu. Faites ensuite la transformée de Fourier inverse, du signal avec un seul des pics. Voilà, si tout s'est bien passé vous venez de filtrer la haute fréquence de votre signal.

Maintenant que vous êtes une professionnelle du filtrage, passons la vitesse supérieure. Chargez l'image que vous trouvez sur `cyberlearn`. Normalement, il s'agit d'une image en niveaux de gris encodées sur 16 bits avec que du bruit dessus (on voit pas grand chose...).

Calculez la transformée de Fourier discrète à deux dimensions de ce signal. Inspectez le signal (à l'aide de la fonction `plot_surface()` qui se trouve dans `from mpl_toolkits.mplot3d` par exemple). Comme pour le signal uni-dimensionnel, filtrez les hautes fréquences (mettez les amplitudes à zéro) au delà d'une certaine fréquence [2]. En deux dimensions, cela revient à mettre à zéro les $\hat{f}[k_1, k_2]$ $k_1 > k_{1,\min}$ ou $k_2 > k_{2,\min}$, où $k_{1,\min}$ et $k_{2,\min}$ sont les fréquences à partir desquelles vous voulez filtrer.

Puis, calculez la transformée de Fourier inverse. Si vous avez filtré correctement votre image, un visage devrait apparaître. Savez-vous de qui il s'agit?

Lorsque vous aurez terminé cette première étape, plus d'images seront déposées sur `cyberlearn` et il faudra également les filtrer et déterminer les noms des gens s'y trouvant.

## 3.3 Compression d'image

Sur `cyberlearn` vous trouverez une image nomée `transmetropolitan.pgm` [3]. Il faudra la lire, et à l'aide de vos transformées de Fourier la compresser et l'écrire sur le disque. A vous de définir un format de compression avec pertes. Vous devrez également fournir l'utilitaire pour la décompresser et l'afficher. Écrivez un code assez générique pour que l'utilisatrice puisse choisir un degré de compression allant par exemple de 0 à 9: 0 étant pas compressé et 9 compressé au maximum (quel que soit ce maximum).

Votre compression étant écrite à l'aide de nombres à virgule flottante double précision, ils prennent plus de place que des entiers 8 bits (comme c'est le cas pour

2. Attention le spectre est "à double" donc n'oubliez pas de garder les parties symétriques également.
3. Pour les fans de bande dessinée je vous recommande la lecture de ces comics.

les fichiers PGM). Afin d'obtenir une conversion, il faut également transformer les amplitudes en entiers 8 bits avant de les écrire. A l'inverse lorsque vous lirez votre image compressée, il faut transformer vos entiers 8 bits en nombres à virgule flottante.

# 4 Rendu

La note est une combinaison de la note du code et du rapport.

Il faut rendre un rapport de quelques pages (quelques: **plus petit** que 5). Ce rapport doit être bref et expliquer votre travail. Je veux un minimum de mathématiques et un maximum d'explications de haut niveau sur comment fonctionne votre filtrage et votre compression/décompression. Vous **devez** faire ce travail par groupe de 2 et aucune exception ne sera faite. Vous devez rendre le rapport et le code sur `cyberlearn` le tout dans une archive aux noms des deux personnes du groupe. Je devrais pouvoir exécuter vos codes pour voir quels sont les visages que vous découvrez et comment vous comprimez vos images.

# 5 Conseils et remarques

Les librairies `python` que j'ai essayées pour faire moi-même ce TP sont:

1. `numpy` pour les manipulations de matrices et le calcul de TFR (les fonctions `fft.ifft2()` et `fft.fft2()`).
2. `imageio` pour la manipulation d'images (en particulier `imread()` et `imwrite()`).
3. `matplotlib` pour la visualisation des images et ds spectres (en particulier `pyplot.imshow()` et `plot_surface()` respectivement).

Vous n'êtes pas obligés de les utiliser. Vous pouvez également utiliser n'importe quel autre language pour effectuer ce TP.