Quelques rappels sur la bibliothèque `fftw3`

- transformation directe ou inverse en fonction du paramètre `sign` (`FFTW_FORWARD` ou `FFTW_BACKWARD`)
  ```
  fftw_plan fftw_plan_dft_2d(int rows, int cols, fftw_complex *in, fftw_complex *out,
                             int sign, FFTW_ESTIMATE)
  ```
  `in` : données en entrée, `out` : données transformées
- exécution du `plan` : `void fftw_execute(fftw_plan plan)` ;
- libération mémoire : `void fftw_destroy_plan(fftw_plan plan);`
- `fftw_complex` correspond au type `complex` du `c99`, exemple d'utilisation :
  `fftw_complex c = a+I*b; double re = creal(c); double im = cimag(c).`
  Attention : `creal` et `cimag` sont uniquement des *accesseurs* et non des *modificateurs*.

Notes importantes sur `fft.c` et `test-fft.c`

Note importante : dans ces deux programmes, vous pouvez implanter des fonctions annexes (non spécifiées) qui vous sembleront utiles pour leur développement. Par contre, vous ne pouvez pas modifier, la signature des fonctions déjà présentes dans ces deux fichiers, ni dans les fichiers d'entête.

Fonction `test_forward_backward` du programme `test-fft.c`

Écrire la fonction `void test_forward_backward(char *name)`
— elle permet de tester l'enchaînement des fonctions `forward` et `backward`
— elle produit une image reconstruite nommée `FB-<name>.ppm` où `name` est le nom de l'image de test.
Attention à la gestion du nom du chemin vers le fichier `FB-<name>.ppm` (`FB-<name>.ppm` ≠ `../data/FB-<name>.ppm`).

`../data/lena-gray.ppm`

`./test-fft ../data/lena-gray.ppm`
→ `FB-lena-gray.ppm`

Fonction `forward` du module `fft.c` (transformée directe)

Rappel : pour une fonction en 1D f de taille N, la transformée de Fourier directe FT de f en un point x du domaine spatial est

$$FT(f(x))(u) = \sum_{x=0}^{N-1} f(x) \exp(-i2\pi ux/N) \tag{1}$$

avec $u \in [0, \ldots, N[$ les points dans le domaine fréquentiel.

Écrire la fonction `fftw_complex *forward(int rows, int cols, unsigned short *gray_image)` du module `fft.c` qui met en œuvre une transformée de Fourier directe :

1. construction d'une image complexe à partir d'une image source en niveaux de gris (`gray_image`) de taille `rows`×`cols`. L'image complexe est un vecteur de taille : `rows*cols*sizeof(fftw_complex)`. La partie réelle de chaque élément de l'image est initialisée avec le niveau de gris de l'image source et la partie imaginaire avec 0
2. allocation d'une structure de données complexe (ayant la même taille que l'image source) afin de recevoir le résultat de la transformée directe
3. initialisation et calcul de la transformée directe en utilisant les fonctions `fftw_plan_dft_2d`, `fftw_execute` et `fftw_destroy_plan` de la bibliothèque `fftw3`
4. libération mémoire des données intermédiaires (image complexe, plan, etc)

Fonction `backward` du module `fft.c` (transformée inverse)

Rappel : pour une fonction en 1D F de taille N, la transformée de Fourier inverse FT^{-1} de F en un point u du domaine fréquentiel est

$$FT^{-1}(F(u))(x) = \frac{1}{N}\sum_{u=0}^{N-1} F(u)\exp(i2\pi ux/N) = f(x) \tag{2}$$

avec $x \in [0, \ldots, N[$ les points dans le domaine spatial.

Écrire la fonction `unsigned short *backward(int rows, int cols, fftw_complex *freq_repr)` du module `fft.c` qui met en œuvre une transformée de Fourier inverse.

1. allocation d'une structure de donnée afin de recevoir le résultat de la transformée inverse
2. initialisation et calcul de la transformée inverse en utilisant les fonctions `fftw_plan_dft_2d`, `fftw_execute` et `fftw_destroy_plan` de la bibliothèque `fftw3`
3. extraction de la partie réelle I^R du nombre complexe produit par la transformée pour obtenir la représentation spatiale
4. libération des données intermédiaires
5. **attention** : la bibliothèque `fftw3` n'applique pas le facteur de normalisation de l'équation (2), penser à le faire

Fonction `test_reconstruction` du programme `test-fft.c`

Écrire la fonction `void test_reconstruction(char *name)`

— elle permet de tester l'enchaînement des fonctions `freq2spectra` et `spectra2freq`

— elle produit une image reconstruite nommée `FB-ASPS-<name>.ppm` où `name` est le nom de l'image de test
Attention : même remarques sur les noms des images en sortie.

`../data/lena-gray.ppm`

`./test-fft ../data/lena-gray.ppm`
→ `FB-ASPS-lena-gray.ppm`

Spectres d'amplitude (`as`) et de phase (`ps`)

`void freq2spectra(int rows, int cols, fftw_complex *freq_repr, float *as, float *ps);`

Cette fonction permet à partir d'une représentation fréquentielle (`freq_repr`), de calculer le spectre d'amplitude (`as`) et de phase (`ps`).

`void spectra2freq(int rows, int cols, float *as, float *ps, fftw_complex *freq_repr);`

Cette fonction permet à partir des spectres d'amplitude (`as`) et de phase (`ps`), de calculer une représentation fréquentielle (`freq_repr`).

Fonction `test_display` du programme `test-fft.c`

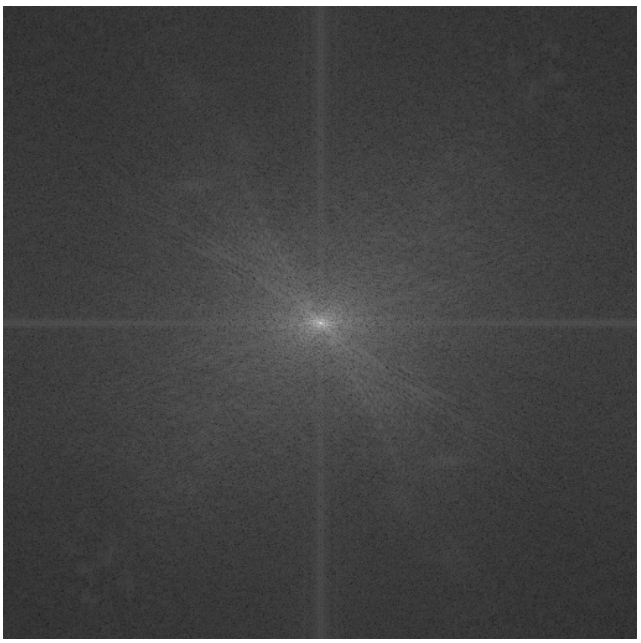
Écrire la fonction `void test_display(char *name)`

— elle permet de construire les images d'amplitude et de phase en utilisant `freq2spectra`

— elle produit deux images : une image d'amplitude nommée `AS-<name>.ppm`, une image de phase `PS-<name>.ppm` où `name` est le nom de l'image de test

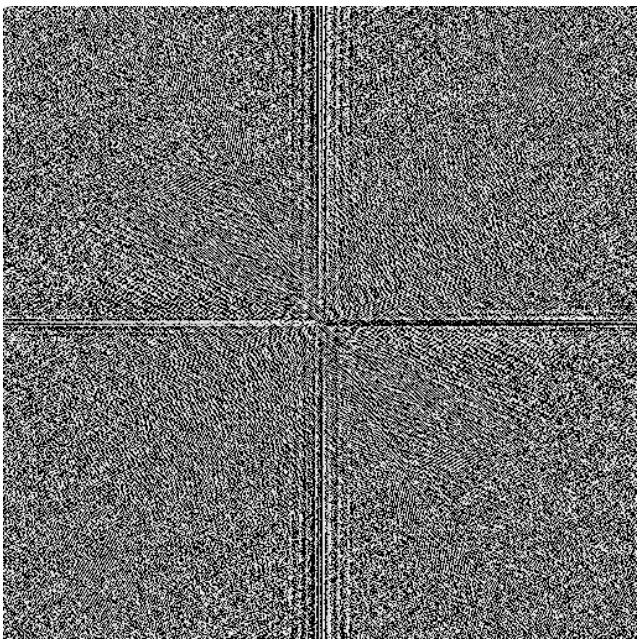
— les deux spectres sont centrés, c'est-à-dire que les basses fréquences se situent dans le centre de l'image (ajout d'une fonction de (dé)centrage ?)

— le spectre d'amplitude admet une dynamique des valeurs très élevées, par exemple, $F(0,0) = \sum_{i=0}^{M} \sum_{j=0}^{N} f(i,j)$ est la somme des toutes les intensités de l'image. Penser à appliquer une transformation non-linéaire pour ramener les valeurs visualisables entre $[0, 255]$



`./test-fft ../data/lena-gray.ppm`

→ `AS-lena-gray.ppm`



`./test-fft ../data/lena-gray.ppm`

→ `PS-lena-gray.ppm`

Fonction `test_add_frequencies` du programme `test-fft.c`

Écrire la fonction `void test_add_frequencies(char *name)`

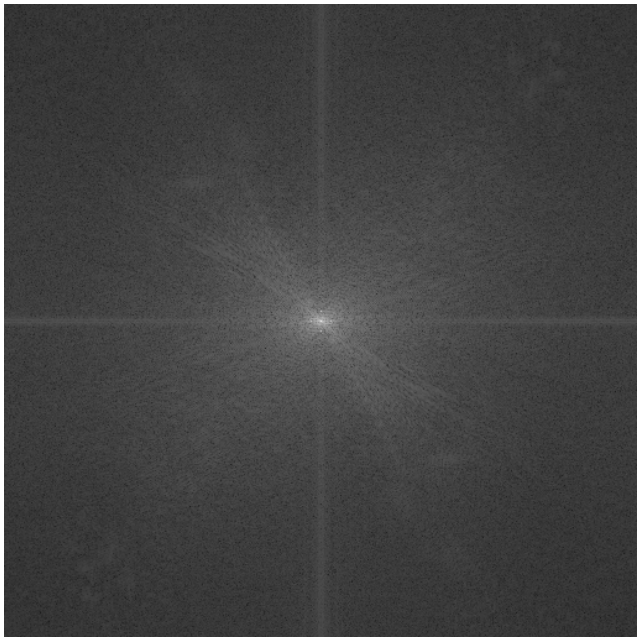
— elle permet de construire une image dont le spectre d'amplitude a été modifié

— elle modifie le spectre d'amplitude avec deux fonctions sinusoïdales de fréquence 8 de manière horizontale et verticale et dont les amplitudes ont la valeur de $0.25 \times \max$ où max est la valeur maximale de toutes les amplitudes du spectre de l'image dans le cas d'une image en niveau de gris (dans le cas d'une image couleur, le max est celui du canal rouge)

— l'image reconstruite est sauvée dans un fichier nommé `FREQ-<name>.ppm` où `name` est le nom de l'image de test

— la nouvelle image d'amplitude est sauvée dans un fichier nommé `FAS-<name>.ppm` où `name` est le nom de l'image de test

`./test-fft ../data/lena-gray.ppm`
→ `FREQ-lena-gray.ppm`



`./test-fft ../data/lena-gray.ppm`
→ `FAS-lena-gray.ppm`