Université Aix-Marseille 2017-2018
Master Mathématiques et applications 1ère année

Traitement du signal

TP1 : Prise en main de la transformée de Fourier discrète. Filtrage des hautes fréquences.

La notation $\mathbb{1}_A$ *indique que* $\mathbb{1}_A(x) = 1$ *si* $x \in A$ *et* 0 *sinon.*

AVERTISSEMENT : On vous donne dans ce TP des exemples pour vous aider à comprendre les commandes. Le but n'est pas de les recopier telles quelles mais de chercher à les comprendre. Vous pouvez les modifier à votre manière pour vous les approprier.

Tous les fichiers à télécharger dont vous avez besoin sont dans la rubrique TP1 du site Ametice.

REMARQUE IMPORTANTE : au cas où vous êtes perdu-e le dossier « AU SECOURS » est là pour vous fournir des programmes ou remarques qui peuvent vous aider. Mais le mieux est de chercher le plus possible à comprendre ce qui arrive.

Et bien sûr me plus important est de vous poser des questions, et de chercher à analyser ce que vous voyez ! C'est l'objectif ! !

1 Objectifs

1. Apprendre à utiliser l'outil numérique qui permet de calculer des transformées de Fourier. *Cet outil s'appelle la transformée de Fourier discrète.*
2. Pratiquer une opération de traitement du signal qui consiste à mettre à 0 les hautes fréquences d'un signal donné. On comprendra ce qu'on obtient en pratique à l'aide de ce qui a été vu en cours.

Il est demandé de rendre pour le 20 février un compte-rendu complet rédigé avec codes déposé sur AMETICE.

2 Recommandations diverses

— Avant de commencer à travailler on vous recommande d'ouvrir dans le menu principal en haut à gauche les préférences du navigateur Web que vous utilisez. Dans la rubrique Fichiers et applications/Téléchargements choisissez « Toujours demander où enregistrer les fichiers ».
— Dans tout ce qui suit vous êtes libre de travailler avec l'interface que vous préférez. On suggère pour tous ceux qui n'ont pas de préférence particulière de travailler avec Spyder. La section 5 pourra être utile à ceux qui veulent travailler avec Spyder mais n'en ont pas l'habitude.
— Dans tous les cas on vous conseille de travailler avec l'éditeur de texte que vous souhaitez sur un fichier procédure `tp1.py` que vous enregistrez dans le dossier dans lequel vous allez travailler. Vous pouvez créer bien sûr autant de fichiers procédures que vous le souhaitez. Cela vous permet de garder une trace de ce que vous aurez fait.

  Dans le cas où cela vous paraît adapté vous pouvez aussi créer les fonctions qui vous semblent adéquates.

  N'hésitez pas à mettre des commentaires ou explications dans vos codes à l'aide du signe #.

3 Numérisation d'un signal

3.1 Introduction

Un signal de la variable continue, c'est à dire une fonction f de la variable continue, doit être numérisé pour pouvoir être traité par un ordinateur.

On choisit l'intervalle de temps que l'on notera ici $[0, T]$ (sans perte de généralité), sur lequel on veut travailler. Puis on choisit un intervalle de temps $t_s = T/N$ (N entier), qui indique à quelle fréquence on va prendre les échantillons. On note $f_s = \frac{1}{t_s}$ qui est appelée « **fréquence d'échantillonnage** ».

Le signal numérisé est alors obtenu en calculant pour $0 \leq n \leq N-1$ $u[n] = f(nt_s) = f\left(\frac{n}{f_s}\right) = f\left(\frac{nT}{N}\right)$. Le vecteur $u = (u[0], \dots u[N-1]) = \left(f\left(\frac{0}{N}\right), \dots, f\left(\frac{(N-1)T}{N}\right)\right)$ correspond donc à la numérisation du signal f (voir figure 1).

On se rend compte sur cette figure 1 que si la fréquence d'échantillonnage f_s est trop petite, le signal pourrait être dégradé au moment de la numérisation. Ce point sera étudié plus tard dans le cours.

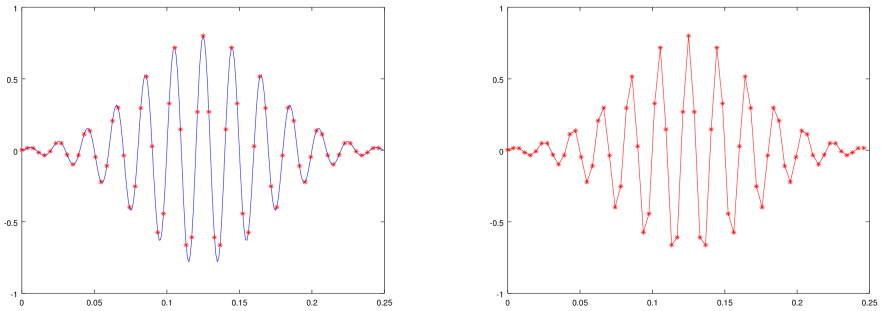

FIGURE 1 – A gauche : le signal original f représenté sur $[0, T]$ en bleu a été échantillonné et on a donc construit le vecteur $u = (u[0], \dots u[N-1])$ dont les valeurs correspondent aux ordonnées des points rouges. A droite : on trace en les reliant par des segments de droites les points $(t_n, u[n])$ avec $t_n = \frac{nT}{N}$.

3.2 Avec Python

Les fichiers de données que nous allons considérer sont des fichiers au format **.wav** comme par exemple le fichier **trompbattEssen.wav** et **trompEssen.wav** qui sont des extraits du morceau « Essentielles » d'Ibrahim Maalouf (pour l'écouter en entier le chercher sur Youtube).

Il faut pouvoir transformer ces fichiers en un vecteur dont les composantes sont les valeurs du signal aux instants nT/N, $n = 0, \dots, N-1$. Pour cela, on utilise le module `scipy.io.wavfile` et sa commande `read`.

Commandes Python

— *Commencez par écouter les morceaux de ces fichiers avec le lecteur de votre choix (par exemple VLC fonctionne très bien).*
— *Exportez les informations du fichier* `.wav` *à l'aide des commandes suivantes.*

```
import scipy.io.wavfile as wavfile

fs, vecson=wavfile.read('trompEssen.wav')
```

Ces commandes nous permettent d'obtenir le vecteur `vecson` *recherché et la fréquence d'échantillonnage* `fs` *avec laquelle le son a été numérisé, c'est-à-dire combien de valeurs par seconde ont été recueillies.*

— *Supposons à l'inverse que nous ayons synthétisé un signal* `vecson` *et que nous cherchions à le transformer en fichier* `.wav` *pour l'écouter. La commande est alors* `write` *du module* `scipy.io.wavfile`.

Cependant nous avons intérêt à ce que les valeurs codées dans le fichier `.wav` *ne soient pas trop grandes pour que les lecteurs standards puissent lire facilement le son. Nous ferons donc systématiquement la normalisation suivante*

```
import numpy as np

m=np.max(np.abs(vecson))
vecson=vecson/(1.1*m)
wavfile.write(vecson,fs,'newson.wav')
```

Ces commandes permettent de transformer le vecteur `vecson` *en un fichier* **newson.wav** *avec la fréquence* `fs` *désirée.*

4 Transformée de Fourier discrète

4.1 Introduction

L'outil qui va remplacer la transformée de Fourier (que nous avons étudié en cours et qui s'applique aux signaux de la variable continue dans $L^2(\mathbb{R})$) pour analyser les fréquences présentes dans un vecteur $u = (u[0], \dots u[N-1])$ est la **transformée de Fourier discrète.** Sa définition est la suivante.

Définition 1

On appelle transformée de Fourier discrète d'un vecteur $u = (u[0], u[1], .., u[N-1])$ *de* $\mathbb{C}^N$ *le vecteur* $\hat{u} \in \mathbb{C}^N$ *telle que*

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

En utilisant ce qu'on a fait dans la partie échantillonnage on voit qu'on a en fait avec (1)

$$\hat{u}[k] = \sum_{n=0}^{N-1} f\left(\frac{nT}{N}\right) e^{-2i\pi \frac{kn}{N}}, k = 0, ..., N-1$$

En considérant que la somme est une discrétisation de l'intégrale on pourrait écrire de manière formelle, en faisant le changement de variables $n/N = x$, puis $xT = t$

$$\hat{u}[k] \sim \int_0^1 f(xT) e^{-2i\pi kx} dx = \frac{1}{T} \int_0^T f(t) e^{-2i\pi \frac{kt}{T}} dt$$

On voit donc qu'on aurait ainsi avec $\hat{u}[k]$ une approximation de la transformée de Fourier de $g(t) = \frac{1}{T}\ \mathbb{1}_{[0,T]} f(t)$. Nous ne nous occuperons pas ici de formaliser mathématiquement et correctement ce raisonnement. Mais nous prendrons en compte que cet outil semble bien adapté pour calculer des transformées de Fourier dans un cas numérique.

Dans la partie 7 on vous demande de démontrer un certain nombre de propriétés. Vous pouvez faire ce travail chez vous ou admettre ces résultats pour le moment. Ils ressemblent à ce qu'on a pour la transformée de Fourier standard.

Ceux qui vont nous servir dans ce TP sont les suivants.

1. $\hat{u}$ peut être prolongée en une suite périodique de période N, c'est à dire telle que $\hat{u}[k+N] = \hat{u}[k]$ pour tout $k \in \mathbb{Z}$.
2. si $u = (u[0], u[1], .., u[N-1])$ est une suite finie à valeurs réelles, alors $\hat{u}$ vérifie la propriété dite de "symétrie hermitienne", c'est à dire :

$$\overline{\hat{u}[k]} = \hat{u}[N-k]$$

et que son prolongement périodique vérifie donc $\overline{\hat{u}[k]} = \hat{u}[-k]$.

3. On a la formule de reconstruction suivante pour tout $n = 0, .., N-1$.

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

4.2 Avec Python

La meilleure façon de calculer la transformée de Fourier discrète d'un vecteur `y` est l'algorithme de transformée de Fourier rapide (en anglais FFT pour Fast Fourier Transform) qui fonctionne en $\mathcal{O}(N\log(N))$ opérations. C'est un des algorithmes les plus rapides actuellement existants. Plusieurs librairies de Python ont implémenté cet algorithme. Nous nous proposons d'utiliser ici le module `numpy.fft` de la librairie `numpy`.

La fonction de cette librairie que nous pouvons utiliser pour calculer la transformée de Fourier discrète d'un vecteur u est ainsi `numpy.fft.fft`.

Commandes Python

— *on veut calculer la transformée de Fourier discrète d'un vecteur* `y`

```
import numpy as np
ychap=np.fft.fft(y)
```

— *pour reconstruire y à partir du vecteur $\hat{y}$ on va utiliser la commande* `numpy.fft.ifft` *de la librairie (*`ifft` *étant donné pour Inverse Fast Fourier Transform). Il peut y avoir des valeurs complexes dans y dues à des approximations numériques, donc on prendra toujours la partie réelle du résultat.*

```
y=np.fft.ifft(ychap)
y=y.real
```

— *L'intérêt de la transformée de Fourier est de pouvoir analyser les fréquences présentes dans le signal.*
En particulier la propriété de symétrie hermitienne nous invite à regarder la transformée de Fourier discrète d'un signal en utilisant $(\hat{u}[-N/2+1], \hat{u}[-N/2+2], \ldots, \hat{u}[-1], \hat{u}[0], \hat{u}[1] \ldots, \hat{u}[N/2])$ *afin d'avoir les fréquences bien centrées en 0 et pour faciliter l'interprétation.*

On s'intéresse ici à la transformée de Fourier discrète du signal $y = (y[0], .., y[N-1])$ avec $N = 1024$, et $y[n] = \cos(2\pi * \omega_0 \frac{n}{N})$ pour $n = 0, \ldots 1023$ et $\omega_0 = 400$. Nous nous proposons de comparer les deux codes suivants qui permettent de visualiser la transformée de Fourier discrète de y. Nous allons pour cela utiliser la librairie `matplotlib.pyplot`.

Remarque : *Dans notre exemple $f_s = N$ mais nous indiquons comment normaliser l'axe des fréquences dans le cas général.*

— *Fréquences non centrées*

```
import matplotlib.pyplot as mpt
ychap=np.fft.fft(y)
N=ychap.size
```

```
freq=np.arange(N,step=1)*fs/N
mpt.figure(1)
mpt.plot(freq,np.abs(ychap))
```

Voir la figure de gauche dans (2).

— *Fréquences centrées en* 0 *: on utilise pour cela la fonction* `fftshift` *de* `numpy.fft` *qui calcule* $(\hat{u}[-N/2], \hat{u}[-N/2+1], \ldots, \hat{u}[-1], \hat{u}[0], \hat{u}[1] \ldots, \hat{u}[N/2-1])$ *à partir de* $\hat{u}$*.*

```
ychap=np.fft.fft(y)
N=ychap.size
freq=np.arange(-np.floor(N/2),N-np.floor(N/2),step=1)*fs/N
mpt.figure(2)
mpt.plot(freq,np.abs(np.fft.fftshift(ychap)))
```

Voir la figure de droite dans (2).

Quelle figure vous semble la plus adéquate pour analyser les fréquences présentes dans le signal y ?

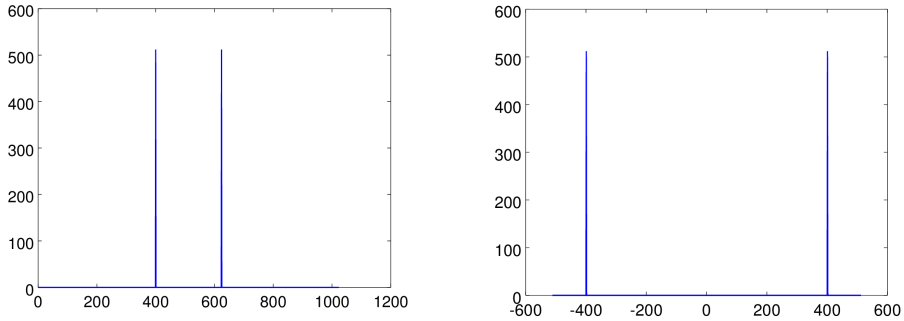

FIGURE 2 – Transformée de Fourier discrète du signal $u = (u[0], .., u[N-1])$ avec $N = 1024$, et $u[n] = \cos(2\pi * 400\frac{n}{N})$. A gauche $(|\hat{u}[0]|, \ldots |\hat{u}[N-1]|)$ en fonction de $0, \ldots N-1$. A droite $|\hat{u}[k]|$ en fonction de $k = -\frac{N}{2}, \ldots \frac{N}{2} - 1$. Ici $f_s = N$ et $T = 1$.

5 Mise en route avec Spyder

— Connectez-vous sur votre compte
— Créez un/des répertoires dans lesquels vous rangez votre travail.

— Lancez Spyder qui est accessible à partir de l'application Anaconda.

— Une fois Spyder lancé, créez à l'aide de l'éditeur de texte de Spyder (à gauche dans l'interface), ou d'un autre éditeur adapté, un fichier procédure `tp1.py` dans lequel vous travaillerez, puis que vous executerez dans la fenêtre de commande Spyder.

Vous pouvez créer bien sûr autant de fichiers procédures que vous le souhaitez. Cela vous permet de garder une trace de ce que vous aurez fait.

Dans le cas où cela vous parait adapté vous pouvez aussi créer les fonctions qui vous semblent adéquates.

Mettez des commentaires ou explications dans vos codes à l'aide du signe #.

— Placez vous dans ce bon répertoire si nécessaire à l'aide de la souris, ou sinon de la commande `cd` dans la fenêtre de commande Spyder.

— On vous recommande de configurer aussi les préférences de Spyder en particulier pour les figures. Ainsi ouvrez les préférences dans le menu principal **Python** en haut à gauche, et dans la rubrique **Console Ipython**, **graphics** cochez l'option **Backend : Automatic** afin que vous puissiez manipuler plus facilement les figures.

6 Travaux pratiques

Exercice 1 (*Sinusoïdes pures*)

1. On considère un vecteur discret $(y[0], y[1], .., y[N-1])$ avec $N = 4096$ et $\omega_0 = 100$ et $\omega_1 = 102$ tel que pour $n = 0, .., N-1$

$$y[n] = \sin(2\pi\omega_0 * n/N) + \sin(2\pi\omega_1 * n/N)$$

   (a) Créer sans boucle le vecteur y.

   (b) Le visualiser à l'aide des fonctions du module `matplotlib.pyplot`.

   (c) Calculer $\hat{y}$ à l'aide de la fonction `fft` du module `numpy.fft`.

   (d) Visualiser le vecteur dont les coordonnées sont

      i. les parties réelles des $\hat{y}[k]$ pour $k = 0, .., N-1$

      ii. les parties réelles des $\hat{y}[k]$ pour $k = -N/2, .., N/2-1$

      iii. les parties imaginaires des $\hat{y}[k]$ pour $k = -N/2, .., N/2-1$

      iv. les modules des $\hat{y}[k]$ pour $k = -N/2, .., N/2-1$

   (e) Peut-on vérifier sur les figures obtenues qu'il y a deux fréquences principales d'oscillation dans le signal ? Comment les adapter pour récupérer cette information ?

2. A l'aide de la fonction `write` du module `scipy.io.wavfile` transformez le vecteur y en un son en posant $f_s = 4096$.

   On a ainsi obtenu un fichier **.wav** *qui donne une seconde de son échantillonné à* $f_s = 4096$ *points par seconde*

3. Créer aussi sans boucle le vecteur x de taille $N = 4096$, tel que pour $n = 0, .., N-1$ et $\omega_2 = 400$

$$x[n] = \sin(2\pi\omega_2 * n/N)$$

   .

4. Le transformer en un son toujours avec $f_s = 4096$. Que se passe-t-il si on augmente ou diminue la valeur de ω_2 ?
   Dans la suite les sinusoïdes de ce type seront appelées sons purs.

Exercice 2 (*Identification de notes*)

Un son harmonique s est composé d'un ensemble de N sinusoïdes de fréquences $\omega_n = n\omega_0$, pour $n = 1, \ldots, N$. Ces sons purs sont appelés les harmoniques du son s, n est appeleé le rang de l'harmonique, et ω_0 est la fréquence fondamentale (harmonique de rang 1). On a donc pour un son harmonique une formule du type, (avec A_n l'amplitude de chaque harmonique),

$$s(t) = \sum_{n=1}^{K} A_n cos(n\omega_0 t)$$

Les instruments de musique produisent des notes qui sont en fait des sons harmoniques. Ainsi en première approximation un instrument de musique produit un son du type

$$f(t) = B(t) \left(\sum_{n=1}^{K} A_n cos(n\omega_0 t) \right)$$

où B est une fonction très régulière qui varie lentement. Si on suppose qu'elle est intégrable on peut alors calculer la transformée de Fourier de f au sens classique. On s'attend donc à ce que $\hat{B}$ décroisse rapidement vers 0. On obtient en tous cas

$$\hat{f}(\omega) = \sum_{n=1}^{K} \frac{A_n}{2}(\hat{B}(\omega - n\omega_0) + \hat{B}(\omega + n\omega_0))$$

1. Télécharger et transformer le son **pianoE.wav** à l'aide de la fonction `read` de `scipy.io.wavfile` (voir section 3.2).
2. Identifier la fréquence fondamentale ω_0 en calculant la transformée de Fourier discrète (voir section 4.2) et à l'aide de la page Wikipedia sur les fréquences des touches du piano.
3. Identifier à quelles fréquences sont les deux notes jouées par le trompettiste dans le morceau **2notesEssen.wav**.

Exercice 3 (*Filtrage passe-bas*)

1. Expliquer ce que renvoie la fonction suivante

```
def filtrage(omegac,y,fs):
      ychap=np.fft.fft(y)
      ychap=np.fft.fft(y)
      N=ychap.size
      M=int(np.floor(N*omegac/fs))
      mask=np.ones(N)
      coord=np.arange(M+1,N-M,1)
      mask[coord]=0
      yfiltchap=mask*ychap
      yfilt=np.fft.ifft(yfiltchap)
      yfilt=yfilt.real
      m=np.max(np.abs(yfilt))
      yfilt=yfilt/(1.1*m)
      return yfilt
```

2. Appliquer cette fonction au signal y de taille $N = 256$ tel que $y[n] = 1$ pour $N/4+1 \leq n \leq 3N/4$ et 0 sinon avec par exemple $\omega_c = 64$ et toujours $fs = N$. Que constatez-vous ?

   On imagine que le signal y est en fait la discrétisation de la fonction $f(t) = \mathbb{1}_{[a_0,b_0]}$ et qu'il est transformé en signal g par l'opération précédente. Pouvez-vous expliquer ce qu'on fait sur f pour obtenir g ? Quelles sont les propriétés de régularité de g ?
3. Télécharger les sons **trompEssen.wav** et **trompbattEssen.wav** et les transformer en vecteurs `numpy`. Leur appliquer la fonction `filtrage` avec par exemple $\omega_c = 4000$. Transformez de nouveau les vecteurs `yfilt` obtenus en son. Pouvez vous expliquer ce que vous obtenez ?

7 Partie mathématique à rendre avec le compte-rendu.

Exercice 4 (*Exercice préliminaire*)

1. Soit $k_0 \in \mathbb{Z}$ et $N \in \mathbb{N}^\star$. Donner la valeur de $\sum_{n=0}^{N-1} e^{2i\pi \frac{nk_0}{N}}$ selon les valeurs de k_0.
2. Soient α et β deux nombres réels. Mettre $e^{i\alpha} - e^{i\beta}$ sous la forme ue^{iv} où u et v sont des nombres réels.

Exercice 5

On note e^k le vecteur de $\mathbb{C}^N$ de coordonnées $(1, e^{2i\pi \frac{k}{N}}, e^{2i\pi \frac{2k}{N}}, ..e^{2i\pi \frac{(N-1)k}{N}})$

1. Montrer que $\hat{u}$ peut être prolongée en une suite périodique de période N, c'est à dire telle que $\hat{u}[k+N] = \hat{u}[k]$ pour tout $k \in \mathbb{Z}$.
2. Montrer que si $u = (u[0], u[1], .., u[N-1])$ est une suite finie à valeurs réelles, alors $\hat{u}$ vérifie la propriété dite de "symétrie hermitienne", c'est à dire :

   $$\overline{\hat{u}[k]} = \hat{u}[N-k]$$

   et que son prolongement périodique vérifie donc $\overline{\hat{u}[k]} = \hat{u}[-k]$.

3. Calculer les transformées de Fourier discrètes des signaux suivants
   (a) $\delta^{k_0} = (\delta^{k_0}[n])_{n=0,..,N-1}$, pour k_0 dans $\{0, .., N-1\}$ fixé, et $\delta^{k_0}[n] = 0$ si $n \neq k_0$ et $\delta^{k_0}[n] = 1$ si $n = k_0$
   (b) $y = \left(\cos(2\pi\omega_0 \frac{n}{N})\right)_{n=0,..,N-1}$ pour $\omega_0 = k_0$ fixé dans $\{0, .., N-1\}$ et pour $\omega_0 \in \mathbb{R}/\{0, .., N-1 \text{ modulo N}\}$.

   La question 2 nous invite à regarder la transformée de Fourier discrète d'un signal à l'aide de la symétrie hermitienne en représentant $(\hat{u}[-N/2+1], \hat{u}[-N/2+2], \ldots, \hat{u}[-1], \hat{u}[0], \hat{u}[1] \ldots, \hat{u}[N/2])$ *afin d'avoir les fréquences bien centrées en* 0 *et pour faciliter l'interprétation. Etudiez en particulier l'exemple 2 pour vous en convaincre.*

4. Soit $u \in \mathbb{C}^N$ donner $\hat{u}$ en fonction des e^k pour $k = 0, .., N-1$.
5. Montrer que pour le produit scalaire usuel sur $\mathbb{C}^N$ le système de vecteurs $\{\frac{1}{\sqrt{N}} e^k, k = 0, \ldots N-1\}$ est une base orthonormale de $\mathbb{C}^N$.
6. En déduire que l'on a $u = \frac{1}{N} \sum_{k=0}^{N-1} \langle u, e^k \rangle e^k$ et donc que pour tout $n = 0, \ldots N-1$ on a la formule de reconstruction (2)
7. Toujours en utilisant le résultat de la question (5) montrer que $\|u\|_2^2 = \sum_{n=0}^{N-1} |u[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |\langle u, e^k \rangle|^2$.
8. Soient deux suites finies $f = \{f[n], n = 0..N-1\}$ et $g = \{g[n], n = 0..N-1\}$ de même longueur N. On appelle produit de convolution circulaire de ces deux suites la suite $h = (h[n])$ définie par

   $$h[n] = \sum_{m=0}^{N-1} f[m].g[n-m] = (f \star g)[n] \text{ où les indices } n-m \text{ sont définis modulo } N \quad (3)$$

   On note $(\hat{f}[k])_{k=0,..,N-1}$ et $(\hat{g}[k])_{k=0,..,N-1}$ les transformées de Fourier discrètes de $(f[n])_{n=0,..,N-1}$ et $(g[n])_{n=0,..,N-1}$.
   Montrer que $h = (h[n])$ est une suite finie et que sa transformée de Fourier discrète notée $(\hat{h}[k])$ vérifie $\hat{h}[k] = \hat{f}[k].\hat{g}[k], \forall k = 0..N-1$.