Transformée de Fourier discrète

Transformée de Fourier discrète

La notion de *transformée de Fourier* joue un rôle primordial en théorie du signal : à la représentation temporelle d'une fonction intégrable s elle substitue une représentation « fréquentielle » f définie par la relation :

$$\mathrm{S}(u) = \int_{-\infty}^{+\infty} s(x)\,\mathrm{e}^{-2\pi i x u}\,\mathrm{d}x$$

D'une certaine façon, on peut voir dans $\mathrm{S}(u)$ la « quantité » du signal s représentée par la fréquence u.

Cette représentation est réversible, dans le sens où il est possible de calculer s à partir de S par le biais de la formule (que nous admettrons) :

$$s(x) = \int_{-\infty}^{+\infty} \mathrm{S}(u)\,\mathrm{e}^{2\pi i x u}\,\mathrm{d}u$$

Cette transformation biunivoque permet de représenter un même signal de deux manières différentes : dans l'espace temporel avec la fonction s fonction du temps ; dans l'espace des fréquences avec la fonction S fonction de la fréquence.

Les formules ci-dessus sont adaptées à un signal analogique (c'est-à-dire continu au sens physique du terme) mais pas lorsqu'on analyse un signal numérique s, connu uniquement par un échantillonnage discret $[s_0, s_1, \ldots, s_{n-1}]$. On remplace alors cette transformation par sa version discrète : la transformée de Fourier discrète (ou DFT, pour *Discrete Fourier Transform*), définie par les relations :

$$\forall k \in [\![0, n-1]\!], \quad \mathrm{S}_k = \sum_{\ell=0}^{n-1} s_\ell\,\mathrm{e}^{-2i\pi k\ell/n}$$

Cette formule possède elle-aussi un caractère réversible :

$$\forall \ell \in [\![0, n-1]\!], \quad s_\ell = \frac{1}{n}\sum_{k=0}^{n-1} \mathrm{S}_k\,\mathrm{e}^{2i\pi k\ell/n}$$

Désormais, nous dirons que le tableau $s = [s_0, \ldots, s_{n-1}]$ est la représentation temporelle du signal numérique s, et le tableau $\mathrm{S} = [\mathrm{S}_0, \mathrm{S}_1, \ldots, \mathrm{S}_{n-1}]$ sa représentation fréquentielle.

Question 1. Rédiger une fonction `dft(s)` qui prend pour argument le tableau s et renvoie le tableau S, puis une fonction `idft(S)` qui réalise la transformation réciproque. Quelle est la complexité (exprimée en fonctions de n) de vos fonctions ?

Rappel. Le nombre complexe $\mathrm{e}^{i\theta}$ est représenté en Python par `exp(1j * theta)`, où `exp` est une fonction du module `numpy`.

Échantillonnage et théorème de Shannon

Considérons tout d'abord une fonction continue très simple $\phi(x)$ représentant un signal analogique :

$$\phi(x) = 3\sin(2\pi x) + 2\cos(3\pi x).$$

Cette fonction est la superposition de deux signaux sinusoïdaux de fréquences respectives 1 et 3/2 (voir figure 1).

Numériser le signal revient à considérer un échantillonnage de la fonction ϕ aux dates $\mathtt{t} = [0, t_e, 2t_e, \ldots, (n-1)t_e]$, où t_e est la *période d'échantillonnage* du signal, et $f_e = \dfrac{1}{t_e}$ la *fréquence d'échantillonnage*.

Le *théorème de Shannon* affirme que pour qu'un signal puisse être entièrement reconstruit à partir des échantillons, il faut et il suffit que $f_e > 2f_{\max}$ où $f_{\max}$ est la plus grande des fréquences présente dans le spectre du signal continu.

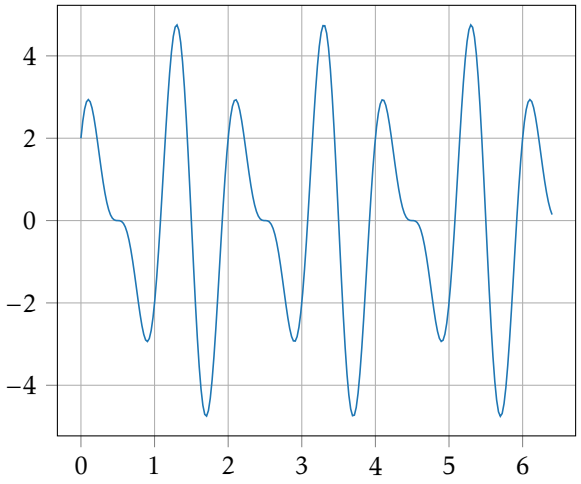


FIGURE 1 – Une représentation du signal analogique ϕ.

Question 2. Ici nous avons $f_{\max} = 3/2$ donc nous allons choisir $f_e = 5$ et $n = 256$.

a) Définir en Python les tableaux `t` $= [0, t_e, \dots, (n-1)t_e]$ et `s` $= [\phi(0), \phi(t_e), \dots, \phi((n-1)t_e)]$ correspondant à l'échantillonnage du signal temporel.

b) Calculer les tableaux `f` $= [0, f_e/n, \dots, (n-1)f_e/n]$ et `S = dft(s)` correspondant à la représentation fréquentielle du signal.

c) Visualiser la représentation spectrale du signal en plaçant en abscisse le tableau `f` et en ordonnée le *module* des valeurs du tableau `S` (n'oubliez pas que les valeurs de ce tableau sont des nombres complexes).

Vous allez observer quatre pics : les deux premiers aux fréquences 1 et 3/2, ce qui était attendu, et leurs images miroir aux fréquences $f_e - 1$ et $f_e - 3/2$. L'apparition de ces deux fréquences fantômes vient du fait que les signaux ϕ et $x \mapsto \phi(nt_e - x)$ possèdent le même échantillonnage. C'est là la raison du théorème de Shannon : lorsque $f_e > 2f_{\max}$ les fréquences fantômes ne se superposent pas avec les vraies fréquences, et on obtient ces dernières en observant le spectre sur la première moitié de l'intervalle fréquentiel, à savoir l'intervalle $[0, f_e/2]$.

d) Recommencer cette expérience en choisissant cette fois une fréquence d'échantillonnage qui ne respecte pas le théorème de Shannon : $f_e = 2{,}5$. Vous n'allez plus observer qu'une seule fréquence, car les fréquences et leurs fantômes se sont superposés ; c'est le phénomène de repliement de spectre (*aliasing* en anglais), qui perturbe l'analyse du spectre fréquentiel.

Vous pouvez recommencer l'expérience avec la valeur limite $f_e = 3 = 2f_{\max}$: vous n'observerez plus que trois pics, deux d'entre-eux se sont superposés.

Reconstitution d'un signal périodique

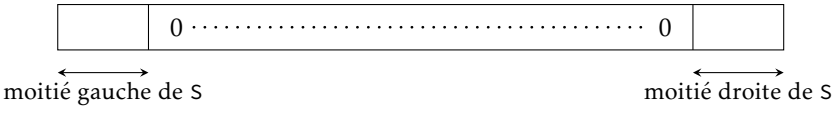
Nous allons maintenant montrer comment la transformée de Fourier discrète permet d'améliorer la qualité d'un signal à partir d'un échantillon de faible qualité. Pour cela nous allons partir d'un échantillonnage de la fonction test, mais cette fois avec une faible valeur de n (mais toujours avec une fréquence d'échantillonnage qui respecte les hypothèses du théorème de Shannon).

Question 3. Dans cette question nous allons prendre $f_e = 5$, mais cette fois seulement $n = 32$ (soit un faible nombre d'échantillons).

a) Calculer la représentation temporelle `s` du signal ϕ pour ces valeurs, puis visualiser cette représentation temporelle (en plaçant le tableau `t` en abscisse et le tableau `s` en ordonnée). On obtient bien entendu une représentation temporelle du signal assez médiocre.

b) Procéder à l'analyse spectrale de ce signal en calculant la représentation fréquentielle `S = dft(s)`, puis le visualiser dans l'espace des fréquences. On notera qu'il est encore possible de distinguer les deux pics de fréquence ainsi que leurs fantômes.

c) Nous allons augmenter artificiellement le nombre d'échantillons en rajoutant des valeurs nulles entre les pics de fréquence et leurs fantômes. Pour cela construire un tableau `S2` de taille $8n$ (soit 256 valeurs) défini ainsi :



Nous disposons maintenant d'un spectre fréquentiel huit fois plus fin. Il reste à calculer `s2 = idft(S2)` pour obtenir un échantillonnage (partiellement virtuel) à 256 points. Visualisez-le et comparer le résultat obtenu avec la figure 1. Vous aller constater que nous avons effectivement obtenu une meilleure représentation temporelle du signal ϕ (pour le tracé, n'oubliez pas que l'intervalle de temps $[0, nt_e]$ doit être lui aussi découpé avec un pas 8 fois plus fin).

Transformée de Fourier rapide

Les fonctions `dft` et `idft` sont de complexité quadratique (c'est-à-dire en $O(n^2)$), ce qui les rend inutilisables pour de grandes valeurs de n. L'algorithme de transformée de Fourier rapide (FFT, pour *Fast Fourier Transform*) que nous allons étudier maintenant, réalise le même calcul avec une complexité en $O(n\log n)$, ce qui le rend bien plus efficace.

Il s'agit d'un algorithme récursif, basé sur le principe « diviser pour régner ». Considérons un échantillonnage de taille $2n$ $s = [s_0, s_1, \dots, s_{n-1}, s_n, \dots, s_{2n-1}]$, et découpons-le en deux échantillons de taille n en séparant les termes d'indices pairs et impairs :

$$s^p = [s_0, s_2, \dots, s_{2n-2}] \quad \text{et} \quad s^i = [s_1, s_3, \dots, s_{2n-1}].$$

Deux appels récursifs permettent de calculer les tableaux $S^p = [S^p_0, \dots, S^p_{n-1}]$ et $S^i = [S^i_0, \dots, S^i_{n-1}]$ avec

$$S^p_k = \sum_{\ell=0}^{n-1} s_{2\ell}\, e^{-2i\pi k\ell/n} \quad \text{et} \quad S^i_k = \sum_{\ell=0}^{n-1} s_{2\ell+1}\, e^{-2i\pi k\ell/n}.$$

Question 4. Notre objectif est de calculer le tableau $S = [S_0, \dots, S_{2n-1}]$ avec $S_k = \sum_{\ell=0}^{2n-1} s_\ell\, e^{-i\pi k\ell/n}$.

a) Montrer que pour $k \in [\![0, n-1]\!]$ on a $S_k = S^p_k + e^{-i\pi k/n} S^i_k$.

b) Montrer que pour $k \in [\![0, n-1]\!]$ on a $S_{n+k} = S^p_k - e^{-i\pi k/n} S^i_k$.

c) En s'appuyant sur ces formules, rédiger en Python une fonction récursive `fft(s)` qui prend pour argument un échantillonnage discret *s dont la taille est une puissance de 2* et qui renvoie sa transformée de Fourier discrète S.

d) Exprimer la complexité $C(2n)$ de cet algorithme pour un échantillon de taille $2n$ en fonction de $C(n)$, et en déduire que $C(n) = O(n\log n)$ lorsque n est une puissance de 2.

Question 5. Justifier les formules $s_\ell = \frac{1}{2}\left(s^p_\ell + e^{i\pi\ell/n} s^i_\ell\right)$ et $s_{n+\ell} = \frac{1}{2}\left(s^p_\ell - e^{i\pi\ell/n} s^i_\ell\right)$ pour $\ell \in [\![0, n-1]\!]$, et en déduire une fonction `ifft(S)` qui réalise la transformée de Fourier inverse.

Question 6. Récupérer sur le site `http://pc-etoile.schola.fr` le fichier `flute.wav`. Il s'agit d'un fichier audio monophonique d'une durée de 5 secondes (vous pouvez l'écouter si votre ordinateur possède un logiciel adapté à sa lecture). Le script qui suit vous permet d'en importer les données essentielles :

```
import scipy.io.wavfile as wave
rate, data = wave.read('flute.wav')  # modifier le chemin d'accès si nécessaire
```

La variable `rate` contient la fréquence d'échantillonnage f_e ; `data` la numérisation du signal. Pour appliquer la fonction `fft` il est nécessaire que la taille de ce tableau soit une puissance de 2 aussi allons-nous ne prendre en compte que les 2^{16} premières données de ce fichier :

```
s = data[:2**16]
```

a) Définir les tableaux `f` et `S = fft(s)` puis visualiser la représentation fréquentielle du signal en se limitant au 1500 premières valeurs de ces deux tableaux (ce qui correspond grosso modo à se limiter aux fréquences inférieures à 1000 Hz). Vous devriez observer la fréquence fondamentale du son produit ainsi que ses deux premières harmoniques.

b) À l'aide du tableau ci-dessous, reconnaître la note jouée :

note	DO	RÉ	MI	FA	SOL	LA	SI	DO	RÉ
Fréquence (Hz)	261,6	293,7	329,6	349,2	392,0	440	493,9	523,2	587,3

Multiplication rapide de polynômes

La transformée de Fourier rapide permet aussi de réaliser un algorithme de multiplication rapide (en $O(n\log n)$) de deux polynômes de degré inférieurs ou égaux à $n-1$, basé sur le principe de l'interpolation polynomiale : un polynôme de degré inférieur ou égal à $n-1$ est défini de manière unique par un échantillon de n de ses valeurs (principe de l'interpolation de Lagrange).

Nous allons représenter en machine le polynôme $P = \sum_{k=0}^{n-1} a_k X^k$ par le tableau `P` $= [a_0, a_1, \ldots, a_{n-1}]$ et nous admettrons que l'algorithme qui suit réalise le produit de deux polynômes P et Q de degrés inférieurs ou égaux à $n-1$:

- représenter P et Q par deux tableaux `P` et `Q` de taille $2n$;
- calculer `p = fft(P)` et `q = fft(Q)` ;
- calculer le tableau `r` $= \left[p_i q_i \;\middle|\; i \in [\![0, 2n-1]\!]\right]$;
- calculer `R = ifft(r)`.

Le tableau obtenu `R` représente le polynôme PQ.

Question 7.

a) Rédiger une fonction `produit(P, Q)` qui prend pour arguments deux tableaux représentant deux polynômes P et Q et renvoie un tableau représentant le produit PQ. On prendra garde au fait que les fonctions `fft` et `ifft` exigent de leurs arguments d'avoir une taille égale à une puissance de 2. Il faudra donc commencer par déterminer un entier $n = 2^p$ vérifiant $\deg P < n$ et $\deg Q < n$.

b) Vérifier la validité de votre algorithme en effectuant le produit de deux polynômes choisis arbitrairement, d'une part avec la fonction que vous avez écrite, d'autre part en utilisant la classe `Polynomial` dont le fonctionnement est décrit dans le mémo de Centrale.

♦ ♦
♦