Capacité numérique : calculer, à l'aide d'un langage de programmation, la transformée de Fourier discrète d'un signal numérique

Série de Fourier d'une fonction à valeurs complexes

On cherche à obtenir le spectre d'un signal $t \mapsto s(t)$.

Comme l'acquisition de ce signal se fait pendant une durée T_a nécessairement finie, les calculs ne se font pas sur la fonction $t \mapsto s(t)$, mais sur la fonction s_{T_a}, T_a-périodique, qui ne s'identifie à la fonction s que sur l'intervalle $[0, T_a[$. La fonction s_{T_a} s'appelle « périodisée de s ». On calcule donc les coefficients d'une *série* de Fourier pour les fréquences $\frac{n}{T_a}$, avec $n \in \mathbb{Z}$.

Rappelons que toute fonction T_a-périodique du temps s, à valeurs complexes (ici : réelles), peut être décomposée en une somme de composantes sinusoïdales discrètes, dont les fréquences sont multiples de la fréquence fondamentale $f_a = \frac{1}{T_a}$. Le développement en série de Fourier s'écrit $\boldsymbol{s(t) = \sum_{n=-\infty}^{+\infty} C_n e^{\frac{2i\pi nt}{T_a}}}$, et les coefficients $C_n \in \mathbb{C}$ de la série de Fourier se calculent à l'aide de la formule suivante : $\boldsymbol{C_n = \frac{1}{T_a} \int_0^{T_a} s(t) e^{-\frac{2i\pi nt}{T_a}} \mathrm{d}t}$.

Le spectre de s fait donc intervenir des fréquences négatives, mais si le signal est réel, on a $C_{-n} = C^*_n$ (conjugué de C_n), d'où $s(t) = C_0 + \sum_{n=1}^{+\infty} C_{-n} e^{-\frac{2i\pi nt}{T_a}} + \sum_{n=1}^{+\infty} C_n e^{\frac{2i\pi nt}{T_a}}$, soit :

$$s(t) = C_0 + \sum_{n=1}^{+\infty} \left[C_n e^{\frac{2i\pi nt}{T_a}} + \left(C_n e^{\frac{2i\pi nt}{T_a}} \right)^* \right] = C_0 + 2 \cdot \mathrm{Re} \left[\sum_{n=1}^{+\infty} C_n e^{\frac{2i\pi nt}{T_a}} \right].$$

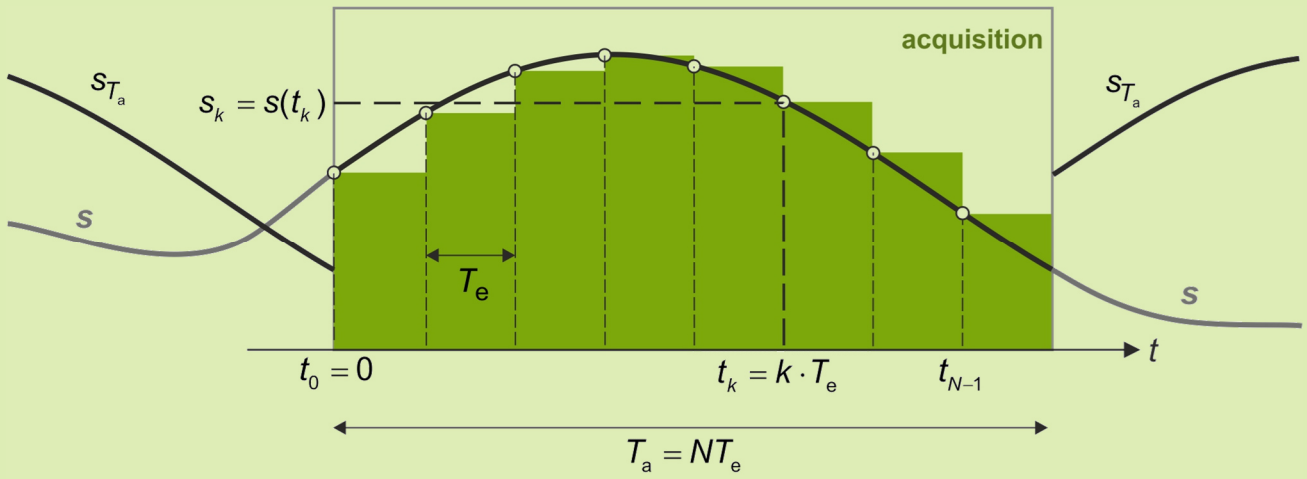
On peut donc ne considérer que les fréquences positives. Le spectre est alors par convention la représentation des modules $2|C_n|$ pour $f = \frac{1}{T_a}, \frac{2}{T_a}, \frac{3}{T_a}, \ldots, \frac{n}{T_a}$, et de celui, $|C_0|$, de la valeur moyenne $C_0 = c_0 = \frac{1}{T_0} \int_0^{T_0} f(t) \mathrm{d}t = \langle f \rangle$ correspondant à la fréquence $f = 0$.

Méthode numérique

Les calculs nécessitent que le signal soit numérisé, c'est-à-dire échantillonné et quantifié.

Pendant T_a on prélève N échantillons aux instants $t_k = k \cdot T_e$, avec $k \in [\![0, N-1]\!]$.

On a $\boldsymbol{t_N = T_a = N \cdot T_e}$.



L'algorithme calcule l'intégrale $C_n = \frac{1}{T_a}\int_0^{T_a} s(t)e^{-\frac{2i\pi nt}{T_a}}\,dt$ en l'approchant par la méthode des rectangles : $C_n \simeq \frac{1}{T_a}\sum_{k=0}^{N-1} s_k e^{-\frac{2i\pi nkT_e}{T_a}} \cdot T_e \Rightarrow C_n \simeq \frac{1}{N}\sum_{k=0}^{N-1} s_k e^{-\frac{2i\pi nk}{N}} = \frac{1}{N}S_n$ puisque $T_a = N \cdot T_e$.

La somme $S_n = \sum_{k=0}^{N-1} s_k e^{-\frac{2i\pi nk}{N}}$ est appelée transformée de Fourier discrète (TFD) du signal numérique $[s_0, s_1, ..., s_k, ...s_{N-1}]$.

On remarque que $S_{n+N} = \sum_{k=0}^{N-1} s_k e^{-2i\pi k} e^{-\frac{2i\pi nk}{N}} = S_n$ car $e^{-2i\pi k} = \left(e^{-2i\pi}\right)^k = 1$.

On retrouve que le spectre obtenu est périodique, de période $\frac{N}{T_a} = \frac{1}{T_e} = f_e$, fréquence d'échantillonnage. On ne calcule donc les C_n que pour $n \in [\![0, N-1]\!]$, donc pour autant de fréquences que d'échantillons du signal temporel.

Un algorithme de calcul direct de S_n, somme de N termes, pour $n \in [\![0, N-1]\!]$, a une complexité $O(N^2)$, donc il faut de longs temps de calcul quand N est grand.

Nous allons exposer l'algorithme de Cooley-Tukey dont la complexité est $O(N\log_2(N))$.

Supposons N pair : $N = 2N'$, avec $N' \in \mathbb{N}$. On peut séparer S_n en deux, **en regroupant les indices pairs et impairs** :

$$S_n = \sum_{k=0}^{N-1} s_k e^{-\frac{2i\pi nk}{N}} = \sum_{k=0}^{N/2-1} s_{2k} e^{-\frac{2i\pi n(2k)}{N}} + \sum_{k=0}^{N/2-1} s_{2k+1} e^{-\frac{2i\pi n(2k+1)}{N}}, \text{ soit :}$$

$$S_n = \sum_{k=0}^{N/2-1} s_{2k} e^{-\frac{2i\pi n(2k)}{N}} + e^{-\frac{2i\pi n}{N}} \cdot \sum_{k=0}^{N/2-1} s_{2k+1} e^{-\frac{2i\pi n(2k)}{N}} = P_n + w^n \cdot I_n, \text{ avec :}$$

— $w = e^{-\frac{2i\pi}{N}}$ **(racine *N ième* de l'unité)** ;

— $P_n = \sum_{k=0}^{N/2-1} s_{2k} e^{-\frac{2i\pi n(2k)}{N}} = \sum_{k=0}^{N'-1} s_{2k} e^{-\frac{2i\pi nk}{N'}}$;

— $I_n = \sum_{k=0}^{N/2-1} s_{2k+1} e^{-\frac{2i\pi n(2k)}{N}} = \sum_{k=0}^{N'-1} s_{2k+1} e^{-\frac{2i\pi nk}{N'}}$.

P_n et I_n sont donc les TFD des échantillons respectivement pairs et impairs, chacun de taille $N/2$. On ne calcule donc ainsi que $N/2$ coefficients P_n et I_n, pour $n \in [\![0, N/2-1]\!]$.

Pour calculer tous les S_n, avec $n \in [\![0, N-1]\!]$, on établit une autre relation :

$$S_{n+N/2} = \sum_{k=0}^{N/2-1} s_{2k} e^{-\frac{2i\pi n(2k)}{N}} \cdot \underbrace{e^{-2i\pi k}}_{1} + e^{-\frac{2i\pi n}{N}} \cdot \underbrace{e^{-i\pi}}_{-1} \sum_{k=0}^{N/2-1} s_{2k+1} e^{-\frac{2i\pi n(2k)}{N}} \cdot \underbrace{e^{-2i\pi k}}_{1} .$$

On peut donc calculer, pour $n \in [\![0, N/2-1]\!]$: $\begin{cases} \boldsymbol{S_n = P_n + w^n \cdot I_n} \\ \boldsymbol{S_{n+N/2} = P_n - w^n \cdot I_n} \end{cases}$ **(*)**.

En prenant N puissance de 2 : $N = 2^p$, avec $p \in \mathbb{N}$, on peut continuer le calcul, *récursivement*, jusqu'à tomber sur un échantillon de taille 1.

La complexité vérifie $C_N = 2C_{N/2} + K \cdot N$ puisqu'on remplace le calcul d'une TFD de N échantillons par le calcul de 2 TFD de $N/2$ échantillons, et qu'il faut $N/2$ boucles pour calculer les S_n avec les relations $(*)$. On a donc :

$C_N = 2[2C_{N/4} + KN/2] + K \cdot N = 4C_{N/4} + 2KN = 8C_{N/8} + 3KN = ... = NC_1 + pKN$, car il y a $p = \log_2(N)$ boucles à effectuer. La complexité est $\boldsymbol{C_N = O[N \log_2(N)]}$.

Q.1) Écrire sous Python une fonction récursive `TFD(s)` dont l'argument est un tableau numpy 1D contenant les contenant les $N = 2^p$ valeurs du signal échantillonné : $[s_0, s_1, ..., s_k, ...s_{N-1}]$, et qui renvoie le tableau numpy $[S_0, S_1, ..., S_k, ...S_{N-1}]$.

Pour pouvoir travailler sur des signaux complexes, les tableaux numpy doivent être de type complexe (data type) : `dtype = 'complex'`.

Q.2) Pour afficher le spectre correct, il faut encore placer les fréquences en abscisse, et pas les indices des coefficients S_n.

D'autre part, le signal étudié étant réel, son spectre est constitué des coefficients $|C_0| = \frac{|S_0|}{N}$ pour la fréquence nulle, et des coefficients $2|C_n| = 2\frac{|S_n|}{N}$ pour la fréquence $\frac{n}{T_a}$.

Expliquer pourquoi on ne trace en pratique le spectre que pour $n \in \left[\!\left[0, \frac{N}{2}\right]\!\right]$.

Écrire une fonction `FFT(signal,Ta)` qui a pour argument $[s_0, s_1, ..., s_k, ...s_{N-1}]$ et la durée d'acquisition T_a. La fonction doit tracer le signal temporel et son spectre, et renvoyer le tableau numpy $[S_0, S_1, ..., S_k, ...S_{N-1}]$.

Q.3) On définit la fonction temporelle : $t \mapsto f(t) = 1 + 3\cos(2\pi f_1 t) + 2\cos(2\pi f_2 t + \frac{\pi}{4}) + \sin(2\pi f_3 t)$ avec $f_1 = 3$ Hz, $f_2 = 4$ Hz et $f_3 = 50$ Hz. On échantillonne sur $[0, T_a[$, avec $T_a = 2$ s et $p = 10$.
Tracer le signal et son spectre ; afficher la fréquence d'échantillonnage.
Effectuer son analyse de Fourier à l'aide de `FDT`.
Tracer le spectre du signal et commenter.
Prendre successivement $T_a = 0{,}2$ s, $T_a = 10$ s et $T_a = 20$ s. Commenter à chaque fois l'allure du spectre.

Comme $S_n = \sum_{k=0}^{N-1} s_k w^{nk}$, la matrice de l'application linéaire qui transforme $[s_0, s_1, ..., s_k, ...s_{N-1}]$ en $[S_0, S_1, ..., S_k, ...S_{N-1}]$ est une matrice de Vandermonde $W = \begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & w & w^2 & ... & w^{N-1} \\ 1 & w^2 & w^4 & ... & w^{2(N-1)} \\ ... & ... & ... & ... & ... \\ 1 & w^{N-1} & w^{2(N-1)} & ... & w^{(N-1)^2} \end{bmatrix}$.

Q.4) Montrer que W est inversible, et que $W^{-1} = \frac{1}{N}\begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & w^{-1} & w^{-2} & ... & w^{-(N-1)} \\ 1 & w^{-2} & w^{-4} & ... & w^{-2(N-1)} \\ ... & ... & ... & ... & ... \\ 1 & w^{-(N-1)} & w^{-2(N-1)} & ... & w^{-(N-1)^2} \end{bmatrix}$.

On a donc $\boldsymbol{s_k = \frac{1}{N}\sum_{n=0}^{N-1} S_n e^{+\frac{2i\pi nk}{N}}}$.

Q.5) Écrire sous Python une fonction récursive `TFDI(S)` dont l'argument est un tableau numpy 1D $[S_0, S_1, ..., S_k, ...S_{N-1}]$ et qui renvoie le tableau numpy. $N[s_0, s_1, ..., s_k, ...s_{N-1}]$. On remarquera qu'il y a très peu de changements à faire par rapport à `FDT`.
Écrire une fonction `FFTI(spectre,Ta)` qui a pour argument $[S_0, S_1, ..., S_k, ...S_{N-1}]$ et qui trace le signal temporel. `Np.real(x)` permet d'extraire la partie réelle d'un nombre complexe `x`.
Vérifier sur le signal étudié au Q.3) que l'on obtient bien le signal échantillonné de départ.