TP4A suite – Electronique numérique : TFD

Problématique :
Comment exploiter la Transformée de Fourier Discrète pour accéder au spectre d'un signal s(t) après l'avoir échantillonné ?

Compétences expérimentales au programme :

Électronique numérique.	
Échantillonnage, fréquence d'échantillonnage. Conséquences expérimentales du théorème de Nyquist- Shannon.	Capacité numérique : calculer, à l'aide d'un langage de programmation, la transformée de Fourier discrète d'un signal numérique.
Filtrage numérique.	Capacité numérique : à l'aide d'un langage de programmation, simuler un filtrage numérique et visualiser son action sur un signal périodique.

Objectifs :
- Calculer la TFD d'un signal à valeurs réelles (généré sous Python ou issu d'une acquisition) en utilisant la fonction rfft de la bibliothèque numpy.fft
- Filtrer numériquement un signal et vérifier l'effet du filtrage en comparant les spectres des signaux d'entrée et de sortie obtenus avec la fonction rfft.

A faire pour le jeudi 07/11 : Fin TP4B (résolution numérique équation diffusion thermique) + Fin TP4A (filtrage numérique) + Lire ce document et répondre aux questions ✎.

A) De la Transformée de Fourier à la Transformée de Fourier Discrète

♦ La **Transformée de Fourier** (TF) $S(f)$ de la fonction $s(t)$ s'obtient en calculant l'intégrale :

$$S(f) = \int_{-\infty}^{+\infty} s(t) \cdot \exp(-i2\pi f t) \cdot dt \qquad \text{avec } i^2 = -1$$

La composante $S(f)$ est un nombre complexe, on représente en général $|S(f)|$.
Les fréquences $f \in \mathbb{R}$ dans le cas général mais les valeurs $f \geq 0$ suffisent pour décrire un signal réel.

♦ Lorsqu'on étudie un signal $s(t)$ en TP, on réalise l'acquisition de ce signal sur une **durée T_{acq} finie**. On fait alors l'approximation suivante pour la TF :

$$S(f) \approx \int_{0}^{T_{acq}} s(t) \cdot \exp(-i2\pi f t) \cdot dt$$

Cette approximation a pour conséquence un **élargissement des raies** : dans le cas d'un signal purement sinusoïdal de fréquence f_1, le spectre comporte une raie centrée sur f_1 et de largeur $\Delta f \approx \frac{1}{T_{acq}}$.

Rq : on exploitera ce résultat lorsqu'on étudiera le paquet d'ondes en EM et les trains d'ondes en optique.

♦ Lorsqu'on fait l'acquisition du signal $s(t)$ en **l'échantillonnant avec une période d'échantillonnage T_e** alors on obtient un signal discret dont les termes sont :

$$s_n = s(t_n) \text{ avec } t_n = n \cdot T_e \text{ pour } n \in [\![0, N-1]\!] \text{ avec } N = \frac{T_{acq}}{T_e}$$

Dans ce cas, on approxime la TF par la **somme discrète** suivante (méthode des rectangles à gauche) :

$$S(f) \approx T_e \cdot \sum_{n=0}^{N-1} s_n \cdot \exp(-i2\pi f t_n) = T_e \cdot \sum_{n=0}^{N-1} s_n \cdot \exp(-i2\pi f n T_e)$$

En considérant la liste de fréquences suivante :

$$f_k = k \cdot \frac{1}{T_{acq}} = \frac{k}{NT_e} = \frac{k}{N} f_e \text{ pour } k \in [\![0, N-1]\!]$$

avec f_e la fréquence d'échantillonnage

on introduit la **Transformée de Fourier Discrète** (TFD) du signal échantillonné qui correspond à une liste dont les termes complexes sont définis par :

$$\underline{c_k} = \sum_{n=0}^{N-1} s_n \cdot \exp\left(-i2\pi \frac{n \cdot k}{N}\right) \text{ pour } k \in [\![0, N-1]\!]$$

Rq : $\underline{c_0} \in \mathbb{R}$ *et* $\underline{c_{N-k}} = \underline{c_k}^*$

Conformément au programme, on utilisera la fonction **rfft** de la bibliothèque **numpy.fft** sous python pour calculer la TFD d'un signal à valeurs réelles (r pour réel et fft pour **Fast Fourier Transform**[*]). On peut obtenir la liste des fréquences à l'aide de **rfft.rfftfreq** ou générer une liste des $f_k = \frac{k}{NT_e} = \frac{k}{N} f_e$ (cf §B).

[*] Le calcul de la TFD est fortement accéléré quand on utilise un algorithme appelé FFT pour Fast Fourier Transform qui a été mis au point dans les années 1960 par J.W. Cooley et John Tukey. L'algorithme FFT réduit le nombre d'opérations pour passer d'une complexité temporelle en $O(N^2)$ (calcul de N sommes de N termes) à une complexité en $O(Nln(N))$, cf annexe.
C'est aussi l'algorithme FFT qui est utilisé par les oscilloscopes numériques et les logiciels de traitements de données, tels que LatisPro, pour obtenir le spectre d'un signal.
NB : Pour optimiser un très grand nombre de traitements, on choisit un **nombre de points d'acquisition** $\boldsymbol{N = 2^p}$ (une puissance de 2).

Pour N pair, l'algorithme FFT de calcul de la TFD donne l'évaluation suivante du spectre de $s(t)$:

- la composante continue du signal $s(t)$ est : $\frac{1}{N}\underline{c_0}$
- pour $k \in \left[\!\left[0, \frac{N}{2}\right]\!\right]$, l'amplitude de la composante de fréquence $f_k = k \cdot \frac{1}{T_{acq}} = \frac{k}{N} f_e$ du signal $s(t)$ est : $\frac{2}{N}\left|\underline{c_k}\right|$

Ce spectre calculé se rapproche du spectre réel du signal $s(t)$ dans la mesure où les conditions suivantes sont respectées : **critère de Nyquist-Shannon, échantillonnage d'un signal périodique sur un nombre entier de périodes**.

B) Mise en œuvre : Calcul de la TFD d'un signal

Objectif : Compléter un programme Python pour obtenir le spectre d'un signal (fichier sur Cahier de Prépa).

➲ 1. Créer la liste des instants t ainsi que le signal s.
— échantillonné à $fe = 4\ kHz$;
— sur une durée $Tacq = 1s$;
— comportant une composante sinusoïdale de fréquence $f1 = 50\ Hz$ et d'amplitude $1\ V$ et d'une autre composante sinusoïdale de fréquence $f2 = 400\ Hz$ d'amplitude $0{,}5\ V$ correspondant à un bruit que l'on souhaite éliminer.
— on notera N le nombre de points et Te la période d'échantillonnage.
On pourra utiliser $np.arange(\text{start}, \text{stop}, \text{step})$ qui génère des valeurs dans l'intervalle $[start, stop)$ espacées de $step$.
➲ 2. Tracer ce signal sur une durée correspondant à 5 périodes $T1 = 1/f1$.
➲ 3. Générer $fourier$ la TFD de s à l'aide de la fonction $rfft$ de la bibliothèque $numpy.fft$:
➲ 4. Vérifier que le résultat $fourier$ est conforme aux informations présentes dans les notes ci-dessous extraites de la documentation de la fonction $numpy.fft.rfft$.

Notes

When the DFT is computed for purely real input, the output is Hermitian-symmetric, i.e. the negative frequency terms are just the complex conjugates of the corresponding positive-frequency terms, and the negative-frequency terms are therefore redundant. This function does not compute the negative frequency terms, and the length of the transformed axis of the output is therefore `n//2 + 1`.

When `A = rfft(a)` and fs is the sampling frequency, `A[0]` contains the zero-frequency term 0*fs, which is real due to Hermitian symmetry.

If *n* is even, `A[-1]` contains the term representing both positive and negative Nyquist frequency (+fs/2 and -fs/2), and must also be purely real. If *n* is odd, there is no term at fs/2; `A[-1]` contains the largest positive frequency (fs/2*(n-1)/n), and is complex in the general case.

➲ 5. Justifier que pour N pair, les valeurs de $fourier$ correspondent à la liste de fréquences :

$$f_k = k \cdot \frac{1}{T_{acq}} \text{ pour } k \in [\![0, \frac{N}{2}]\!]$$

➲ 6. Générer la liste des fréquences correspondant au spectre calculé à l'aide de $np.fft.rfftfreq(N, d = Te)$ avec N nombre de points et Te la période d'échantillonnage.
➲ 7. Tracer alors le spectre du signal : vérifier que le spectre obtenu est conforme à vos attentes et vérifier que la demi-largeur des raies est égale à $1/Tacq$.

➲ 8. Adapter le programme précédent pour obtenir le spectre d'un signal acquis avec LatisPro.
Réaliser l'acquisition avec des paramètres bien choisis puis exporter les courbes vers un fichier csv.
Ouvrir le fichier csv avec EXCEL, remplacer les « , » par des « . », si elle existe, supprimer la ligne qui donne le nom des colonnes. Enregistrer ce fichier sous le format txt avec séparateur tabulation.
Sous python :

- importer la bibliothèque os,
- utiliser la fonction $os.chdir(r"C: \dots")$ pour préciser le répertoire dans lequel se trouve le fichier de données,
- pour convertir le fichier texte de données en un tableau comportant 2 colonnes utiliser la fonction $np.loadtxt : t, U = np.loadtxt(" \dots .txt", unpack = True)$.

C) Retour sur le filtrage numérique – Analyse spectrale

La fonction de transfert d'un filtre passe-bas s'écrit sous forme canonique :

$$\underline{H} = \frac{H_0}{1 + j\omega\tau}$$

Dans la suite, on prendra $H_0 = 1$.

✎➲ 9. Donner l'équation différentielle entre les signaux de sortie $s(t)$ et d'entrée $e(t)$.

On considère un signal $e(t)$ échantillonné avec une période d'échantillonnage T_e.

La correspondance qui permet de passer d'un filtre analogique à un filtre numérique n'est pas unique. Le plus simple est d'utiliser le schéma d'Euler explicite, le pas de temps étant égal à la période d'échantillonnage.

✎➲ 10. Donner l'expression approchée de la dérivée $\frac{ds}{dt}$ en fonction de la période d'échantillonnage T_e et des échantillons s_n et s_{n+1}.

✎➲ 11. En déduire une relation de récurrence permettant de calculer s_{n+1} à partir de $e_n = e(t_n = nT_e)$ et $s_n = s(t_n = nT_e)$.

Ainsi, on peut construire point par point le signal numérique s à partir de e.

On considère une tension d'entrée somme de deux tensions sinusoïdales :

$$e(t) = e_1(t) + e_2(t)$$
$$e_1(t) = 2.\cos(2\pi f_1 t)$$
$$e_2(t) = 0{,}5.\cos(2\pi f_2 t)$$

Avec $f_1 = 10\ Hz$ et $f_2 = 500\ Hz$

On souhaite isoler le signal de basse fréquence. Pour cela, on filtre numériquement.

➲ 12. Proposer des valeurs, en les justifiant, pour la fréquence de coupure f_c du filtre permettant d'obtenir un filtrage adéquat.

➲ 13. Compléter le programme Python correspondant (fichier disponible sur Cahier de Prépa).

➲ 14. Comparer le spectre du signal d'entrée à celui du signal de sortie du filtre.

DOC 1 : Documentation fonction rfft de la bibliothèque numpy.fft

numpy.fft.rfft

fft.rfft(*a, n=None, axis=-1, norm=None*) [source]

Compute the one-dimensional discrete Fourier Transform for real input.

This function computes the one-dimensional *n*-point discrete Fourier Transform (DFT) of a real-valued array by means of an efficient algorithm called the Fast Fourier Transform (FFT).

Parameters: a : *array_like*

Input array

n : *int, optional*

Number of points along transformation axis in the input to use. If *n* is smaller than the length of the input, the input is cropped. If it is larger, the input is padded with zeros. If *n* is not given, the length of the input along the axis specified by *axis* is used.

axis : *int, optional*

Axis over which to compute the FFT. If not given, the last axis is used.

norm : *{"backward", "ortho", "forward"}, optional*

> *New in version 1.10.0.*

Normalization mode (see **numpy.fft**). Default is "backward". Indicates which direction of the forward/backward pair of transforms is scaled and with what normalization factor.

> *New in version 1.20.0:* The "backward", "forward" values were added.

Returns: out : *complex ndarray*

The truncated or zero-padded input, transformed along the axis indicated by *axis*, or the last one if *axis* is not specified. If *n* is even, the length of the transformed axis is `(n/2)+1`. If *n* is odd, the length is `(n+1)/2`.

Raises: IndexError

If *axis* is not a valid axis of *a*.

DOC 2 : Documentation fonction rfftfreq de la bibliothèque numpy.fft

numpy.fft.rfftfreq

`fft.rfftfreq(n, d=1.0)` [source]

Return the Discrete Fourier Transform sample frequencies (for usage with rfft, irfft).

The returned float array *f* contains the frequency bin centers in cycles per unit of the sample spacing (with zero at the start). For instance, if the sample spacing is in seconds, then the frequency unit is cycles/second.

Given a window length *n* and a sample spacing *d*:

```
f = [0, 1, ...,     n/2-1,     n/2] / (d*n)   if n is even
f = [0, 1, ..., (n-1)/2-1, (n-1)/2] / (d*n)   if n is odd
```

Unlike **fftfreq** (but like **scipy.fftpack.rfftfreq**) the Nyquist frequency component is considered to be positive.

Parameters: `n` : *int*
Window length.

`d` : *scalar, optional*
Sample spacing (inverse of the sampling rate). Defaults to 1.

Returns: `f` : *ndarray*
Array of length `n//2 + 1` containing the sample frequencies.

Examples

```
>>> signal = np.array([-2, 8, 6, 4, 1, 0, 3, 5, -3, 4],
dtype=float)
>>> fourier = np.fft.rfft(signal)
>>> n = signal.size
>>> sample_rate = 100
>>> freq = np.fft.fftfreq(n, d=1./sample_rate)
>>> freq
array([  0.,  10.,  20., ..., -30., -20., -10.])
>>> freq = np.fft.rfftfreq(n, d=1./sample_rate)
>>> freq
array([  0.,  10.,  20.,  30.,  40.,  50.])
```

Annexe : Principe de l'algorithme de J.W. Cooley et John Tukey

La TFD du signal numérique à N échantillons $s_E = [s_0, s_1, \ldots, s_{N-1}]$ est une liste de N nombres complexes $[\underline{c}_0, \underline{c}_1, \ldots, \underline{c}_{N-1}]$ définie par :

$$\forall k \in [\![0, N-1]\!], \quad \underline{c}_k = \sum_{j=0}^{N-1} s_j \exp\left(-i\frac{2\pi k j}{N}\right) \quad (\text{avec } i^2 = -1). \tag{5.1}$$

On vérifie facilement que $\underline{c}_0$ est réel et que :

$$\forall k \in [\![1, N-1]\!], \quad \underline{c}_{N-k} = \underline{c}_k^*. \tag{5.2}$$

Des relations de récurrence pour les TFD Dans la formule (5.1) on sépare les termes correspondant à j pair de ceux correspondant à j impair :

$$\begin{aligned}\underline{c}_k &= \sum_{j'=0}^{N/2-1} s_{2j'} \exp\left(-i\frac{2\pi k(2j')}{N}\right) + \sum_{j'=0}^{N/2-1} s_{2j'+1} \exp\left(-i\frac{2\pi k(2j'+1)}{N}\right) \\ &= \sum_{j'=0}^{N/2-1} s_{2j'} \exp\left(-i\frac{2\pi k j'}{N/2}\right) + \exp\left(-i\frac{2\pi k}{N}\right) \sum_{j'=0}^{N/2-1} s_{2j'+1} \exp\left(-i\frac{2\pi k j'}{N/2}\right)\end{aligned}$$

Dans cette expression on reconnaît deux autres TFD :

- si on pose $\underline{P}_k = \sum_{j'=0}^{N/2-1} s_{2j'} \exp\left(-i\frac{2\pi k j'}{N/2}\right)$, alors $[\underline{P}_0, \underline{P}_1, \ldots, \underline{P}_{N/2-1}]$ est la TFD du signal à $N/2$ échantillons $[s_0, s_2, \ldots, s_{N-4}, s_{N-2}]$;
- si on pose $\underline{I}_k = \sum_{j'=0}^{N/2-1} s_{2j'+1} \exp\left(-i\frac{2\pi k j'}{N/2}\right)$, alors $[\underline{I}_0, \underline{I}_1, \ldots, \underline{I}_{N/2-1}]$ est la TFD signal à $N/2$ échantillons $[s_1, s_3, \ldots, s_{N-3}, s_{N-1}]$.

Ainsi, en notant :

$$w = \exp\left(-i\frac{2\pi}{N}\right).$$

on peut dire que :

$$\text{pour } 0 \leq k < \frac{N}{2}, \quad \underline{c}_k = \underline{P}_k + w^k \underline{I}_k, \tag{5.3}$$

D'autre part, pour $0 \leq k < N/2$:

- d'après la propriété (5.2) de la TFD : $\underline{c}_{N/2+k} = \underline{c}^*_{N/2-k}$;
- d'après (5.3) : $\underline{c}^*_{N/2-k} = \underline{P}^*_{N/2-k} + \exp\left(i\frac{2\pi(N/2-k)}{N}\right)\underline{I}^*_{N/2-k}$;
- d'après la propriété (5.2) dans le cas de la TFD d'un signal à $N/2$ échantillons : $\underline{P}^*_{N/2-k} = \underline{P}_k$ et $\underline{I}^*_{N/2-k} = \underline{I}_k$.

Finalement :

$$\text{pour } 0 \leq k < \frac{N}{2}, \quad \underline{c}_{N/2+k} = \underline{P}_k - w^k \underline{I}_k \tag{5.4}$$

Les relations (5.3) et (5.4) ramènent le calcul d'une TFD d'un signal à N échantillons à deux calculs de TFD de signaux à $N/2$ échantillons

Algorithme pour le calcul de la TFD d'un signal à $N = 2^p$ échantillons :

1. si $N = 1$ (*i.e.* $p = 0$), le programme renvoie la liste $[s_0]$;
2. si $N > 1$ (*i.e.* $p > 0$), on fait deux appels récursifs pour calculer les TFD des signaux à $N/2 = 2^{p-1}$ échantillons constitués par les échantillons de rang pair et les échantillons de rang impair du signal original, puis on calcule la TFD du signal original par les formules (5.3) et (5.4).