TP - Filtrage numérique

Dans les séances précédentes, nous avons revu l'effet d'un filtrage analogique sur le spectre d'un signal : un filtre passe-bas permet par exemple d'extraire la valeur moyenne du signal d'entrée en isolant la composante à basse fréquence. La conversion analogique-numérique d'un signal passe par un échantillonnage à une fréquence f_e respectant le critère de Shannon-Nyquist :

$$f_e > 2.f_{max}$$

f_{max} désignant la fréquence la plus grande dans le spectre du signal à numériser. L'échantillonnage du signal fait apparaître des composantes supplémentaires dans le spectre d'un signal de fréquence f_o : pics à $f_e \pm f_o$, $2f_e \pm f_o$ ...

Il peut être intéressant de remplacer une étape de filtrage analogique par une étape de filtrage numérique réalisée une fois le signal analogique converti en signal numérique.

- Le filtrage numérique offre davantage de souplesse : on peut modifier les paramètres du filtres sans changer de composant dans le circuit, il n'y a plus de problème de mise en cascade des filtres (voir TP filtres en cascade).
- Le filtrage numérique ne convient par contre pas pour des signaux de fortes puissances. Il est plus lourd à mettre en œuvre qu'un filtre analogique (CAN / filtrage / CNA).

L'outil nous permettant d'analyser l'effet d'un filtrage est la transformée de Fourier. Nous allons dans un premier temps, nous intéresser à la TFD transformée de Fourier discrète qui permet d'obtenir le spectre d'un signal constitué d'une série de valeurs $s[i]$ prises à des instants $t[i]$ ce qui correspond bien à un signal échantillonné. Nous verrons ensuite des exemples de filtres numériques.

De la TF à la TFD

On considère un signal $s(t)$ a priori non périodique, dont on a réalisé l'acquisition pendant une durée T_{acq} : N points tous les $T_e = 1/f_e$ (f_e fréquence d'échantillonnage supposée respecter le critère de Shannon Nyquist).

1- Comment s'assurer expérimentalement que le critère de Shannon -Nyquist est respecté ?

2- En vous basant sur le TP échantillonneur-bloqueur, justifier par une construction graphique que :

$$T_{acq} = NT_e$$

La transformée de Fourier d'une fonction $f(t)$ s'obtient en calculant l'intégrale :

$$S(f) = \int_{-\infty}^{+\infty} s(t) \exp(-i2\pi ft)\ dt$$

La transformée de Fourier inverse permettant de retrouver le signal $s(t)$:

$$s(t) = \int_{-\infty}^{+\infty} S(f) \exp(i2\pi ft)\ df$$

Dans le cas où l'acquisition a lieu sur une durée T_{acq}, on approxime le calcul de la TF :

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

Dans le cas d'un signal réel, les fréquences positives $f > 0$ suffisent à décrire le spectre du signal.
On considère cette fois que le signal $s(t)$ est un signal discret avec :

$$s_k = s(t_k)\ ;\ t_k = kT_e$$

On peut approximer la TF par la somme discrète suivante :

$$S(f) = T_e \sum_{k=0}^{N-1} s_k \exp(-i2\pi f k T_e)$$

En choisissant les fréquences dans la liste :

$$f_n = n\frac{1}{T_{acq}} = \frac{n}{NT_e}$$

On obtient la transformée de Fourier discrète (TFD) :

$$S_n = S_(f_n) = \sum_{k=0}^{N-1} s_k \exp(-i2\pi nk/N)$$

Le programme recommande d'utiliser la fonction **rfft** de la bibliothèque **numpy.fft** pour calculer la TFD d'un signal (voir détails à la fin de l'énoncé).

Signal à analyser

On souhaite créer un signal :
— échantillonné à $f_e = 4\text{kHz}$;
— sur une durée T_{acq} de 1s ;
— comportant une composante de 50Hz et d'amplitude 1 et d'une autre composant de fréquence 400Hz d'amplitude 0.5 correspondant à un bruit que l'on souhaite éliminer.

1- Créer le signal à étudier : une liste signal et une liste t de mêmes longueur N.
2- Obtenir son spectre à l'aide de la fonction **rfft**. Commenter.

Filtre à moyenne glissante

Les données de départ sont dans la liste signal, les données filtrées dans la liste filtre1 comprenant elle aussi N termes. La ième composante filtre1[i] sera la moyenne entre signal[i], signal[i-1]... jusqu'à signal[i-nfiltre] soit sur nfiltre valeurs, on parle de moyenne mobile ou glissante.

1- Écrire le programme associé en prenant nfiltre=10. Tracer filtre1 en fonction de t.
2- Tracer le spectre de filtre1. Observer l'influence de nfiltre.

Justification : comme on fait une moyenne sur 10 valeurs de M, toute variation du signal de fréquence supérieure à $f_e/10$ est gommée par le filtre. Il s'agit bien d'un filtre passe-bas supprimant les fréquences supérieures à $f_e/10$. Ici 10 mesures avec f_e=4kHz, cela fait une moyenne sur 10*0,25 ms soit une période du signal à 400Hz correspondant au bruit, d'où l'explication de la disparition totale du pic.

Filtre passe-bas d'ordre 1

La forme canonique de la fonction de transfert d'un filtre passe-bas d'ordre 1 est :

$$\underline{H}(j\omega) = \frac{H_o}{1 + j\frac{\omega}{\omega_c}}$$

ω_c désignant la pulsation de coupure à -3dB du filtre.

1 Montrer, à partir de l'expression de la fonction de transfert que le signal d'entrée et le signal de sortie du filtre sont liés par la relation :

$$\frac{dv_s}{dt} + \omega_c v_s = H_o \omega_c v_e$$

2- En écrivant :

$$\frac{dv_s}{dt} = \frac{v_s[t] - v_s[t - T_e]}{T_e}$$

exprimer $v_s(t)$ en fonction de $v_s(t - T_e)$ et $v_e(t)$.

Application :

1. Écrire le programme associé en prenant une fréquence de coupure de l'ordre de 60Hz, on appellera filtre2 la fonction correspondante. Tracer filtre2 en fonction de t.
2. Définir la transformée de Fourier de filtre2 et tracer son spectre. Commenter.
3. Adapter la méthode à un filtrage passe bas du second ordre.
4. Comment adapter la méthode pour effectuer un filtre anti-aliasing ?

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*.

See also

`numpy.fft`
For definition of the DFT and conventions used.

`irfft`
The inverse of **`rfft`**.

`fft`
The one-dimensional FFT of general (complex) input.

`fftn`
The *n*-dimensional FFT.

`rfftn`
The *n*-dimensional FFT of real input.

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.

If the input *a* contains an imaginary part, it is silently discarded.

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.])
```