Introduction aux filtres numériques

1. Introduction

Le filtrage du signal numérique intervient dans de nombreux domaines : transmission du signal, traitement du son, des images, traitement des données scientifiques, etc. Ce document explique le principe de fonctionnement d'un filtre numérique et donne quelques exemples de filtres.

On se limite au cas des signaux unidimensionnels. On considère principalement des filtres à réponse impulsionnelle finie (filtres RIF). Une étude plus détaillée des filtres RIF est présentée dans Filtres à réponse impulsionnelle finie. Voir aussi Exemples de filtres RIF.

On verra aussi un exemple de filtre à réponse impulsionnelle infinie, le filtre intégrateur.

2. Signal échantillonné

Lorsqu'un signal analogique $x(t)$ est échantillonné avec une période d'échantillonnage T_e, on obtient la suite suivante :

$$x_n = x(nT_e) \tag{1}$$

où n est un entier. La suite x_n constitue un signal numérique.

Lorsque le spectre de $x(t)$ a une fréquence maximale f_{max}, la condition de Nyquist-Shannon permet de s'assurer que toute l'information du signal analogique est présente dans le signal échantillonné : la fréquence d'échantillonnage $f_e = 1/T_e$ doit être supérieure à $2f_{max}$.

3. Filtres à réponse impulsionnelle finie

3.a. Définition

Un filtre numérique à réponse impulsionnelle finie ([1]) réalise le calcul d'un signal de sortie y_n en fonction du signal d'entrée x_n de la manière suivante :

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k} \tag{2}$$

La valeur de la sortie à l'instant n est donc une combinaison linéaire des valeurs de l'entrée à l'instant n et aux $N-1$ instants antérieurs. Si les valeurs de l'entrée ne sont disponibles qu'à partir de l'indice 0, la première valeur de la sortie est y_{N-1}, puisqu'il faut au moins N éléments en entrée pour faire le calcul. Pour voir la signification des coefficients h_k, considérons en entrée une impulsion, définie par :

$$x_0 = 1 \tag{3}$$

$$x_n = 0 \; (n \neq 0) \tag{4}$$

La sortie est alors :

$$y_n = 0 \; (n < 0) \quad (5)$$
$$y_0 = b_0 x_0 = b_0 \quad (6)$$
$$y_1 = b_1 x_0 = b_1 \quad (7)$$
$$\dots \quad (8)$$
$$y_{N-1} = b_{N-1} x_0 = b_{N-1} \quad (9)$$
$$y_n = 0 \; (n \geq N) \quad (10)$$

Les coefficients $b_0, b_1 ... b_{N-1}$ constituent donc la réponse impulsionnelle du filtre.

L'opération définie par la relation (2) est un produit de convolution. La sortie est donc le produit de convolution de l'entrée par la réponse impulsionnelle. Ce type de filtrage est aussi appelé filtrage par convolution.

Le produit de convolution peut être calculé avec la fonction `scipy.signal.convolve`, en suivant la syntaxe suivante :

```
y = scipy.signal.convolve(x,b,mode='valid')
```

L'option `mode='valid'` permet de limiter le calcul aux points valides : le premier point calculé (y_{N-1} dans la relation (2)) est obtenu par combinaison des N premiers points du tableau `x`, le dernier point par combinaison des N derniers points. Le tableau `y` a ainsi $N-1$ points de moins que le tableau `x`.

On peut aussi utiliser l'option `mode='same'` qui conserve le nombre de points. Pour ce faire, les $N/2$ premiers points sont calculés en utilisant une partie de la réponse impulsionnelle, de même pour les $N/2$ derniers points.

3.b. Réponse fréquentielle

Considérons un signal d'entrée sinusoïdal, de fréquence f et d'amplitude unité :

$$x_n = \exp(i2\pi \, f \, nT_e) \quad (11)$$

Le signal en sortie s'écrit :

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k} \quad (12)$$
$$= \exp(i2\pi \, n \, f \, T_e) \sum_{k=0}^{N-1} b_k \exp(-i2\pi \, f \, k \, T_e) \quad (13)$$
$$= x_n H(Z) \quad (14)$$

avec :

$$H(Z) = \sum_{k=0}^{N-1} b_k Z^{-k} \quad (15)$$
$$Z = \exp(i2\pi \, f \, T_e) \quad (16)$$

On voit donc que la sortie est dans ce cas proportionnelle à l'entrée. $H(z)$ est la fonction de transfert en Z, analogue à la fonction de transfert des signaux continus. La réponse fréquentielle est finalement :

$$H_f(f) = \sum_{k=0}^{N-1} b_k \exp(-i2\pi\, k\, f\, T_e) \tag{17}$$

On remarque que cette réponse fréquentielle est en fait une fonction de fT_e, qui est le rapport de la fréquence du signal sinusoïdal sur la fréquence d'échantillonnage $f_e = 1/T_e$. D'après le théorème de Shannon, ce rapport doit être inférieur à $1/2$. La réponse fréquentielle est de période f_e mais en pratique on la trace sur l'intervalle $[0, f_e/2]$.

La fonction suivante renvoit le module et l'argument de la réponse fréquentielle. La réponse impulsionnelle et le nombre de points de la courbe sont fournis en argument.

```
import math
import cmath
import numpy

def reponseFreq(b,nf):
    f = numpy.arange(0.0,0.5,0.5/nf)
    g = numpy.zeros(f.size)
    phi = numpy.zeros(f.size)
    for m in range(f.size):
        H = 0.0
        for k in range(b.size):
            H += b[k]*cmath.exp(-1j*2*math.pi*k*f[m])
        g[m] = abs(H)
        phi[m] = cmath.phase(H)
    return (f,g,numpy.unwrap(phi))
```

On peut aussi utiliser la fonction `scipy.signal.freqz`.

4. Exemples

4.a. Synthèse d'un signal bruité

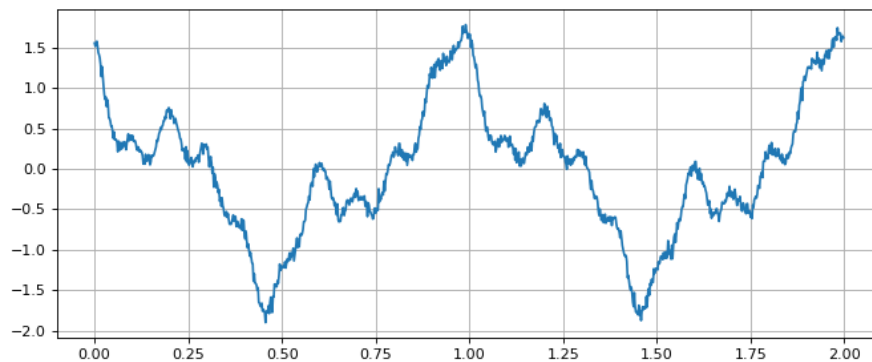
Pour tester les différents filtres, nous allons utiliser un signal périodique auquel on ajoute un bruit gaussien :

On commence par définir le signal par ses harmoniques :

```
def signal(t):
    return 1.0*math.cos(2*math.pi*t) \
            +0.5*math.cos(3*2*math.pi*t+math.pi/3) \
            +0.2*math.cos(5*2*math.pi*t+math.pi/5) \
            +0.2*math.cos(10*2*math.pi*t)
```

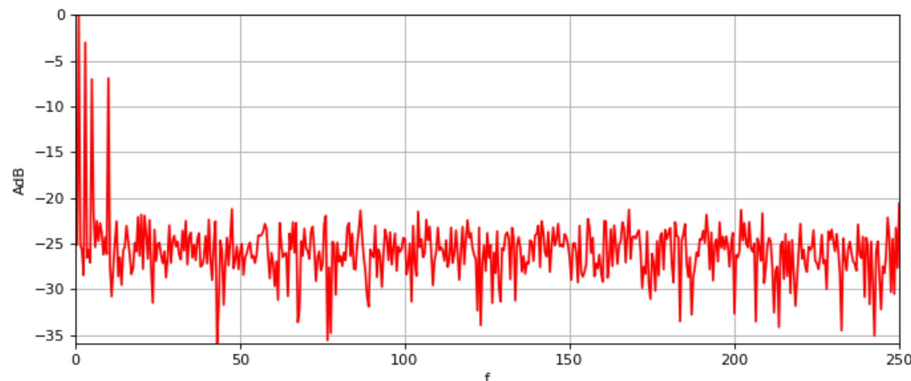
Au moment d'échantillonner ce signal, on ajoute un bruit gaussien :

```
import numpy
import random
from matplotlib.pyplot import *

fe=500
T=2.0
N=int(fe*T)
te = T/N
x = numpy.zeros(N)
t = numpy.zeros(N)
sigma = 0.05
for k in range(N):
    t[k] = k*te
    x[k] = signal(t[k])+random.gauss(0,sigma)
figure(figsize=(10,4))
plot(t,x)
grid()
```



On calcule aussi le spectre du signal discret au moyen de sa transformée de Fourier discrète :

```
from numpy.fft import fft
tfd = fft(x)
freq = numpy.arange(x.size)*1.0/T
spectre = 10*numpy.log10(numpy.absolute(tfd))
spectre = spectre-spectre.max()
figure(figsize=(10,4))
plot(freq,spectre,'r')
axis([0,fe/2,spectre.min(),spectre.max()])
xlabel("f")
ylabel("AdB")
grid()
```



On repère sur ce spectre les raies correspondant aux harmoniques définies dans le signal. Le bruit se manifeste à toute fréquence (c'est un bruit blanc). La fréquence d'échantillonnage a été choisie très largement supérieure aux fréquences utiles du signal. On parle dans ce cas de sur-échantillonnage. Cette condition est nécessaire pour effectuer un filtrage efficace du bruit.

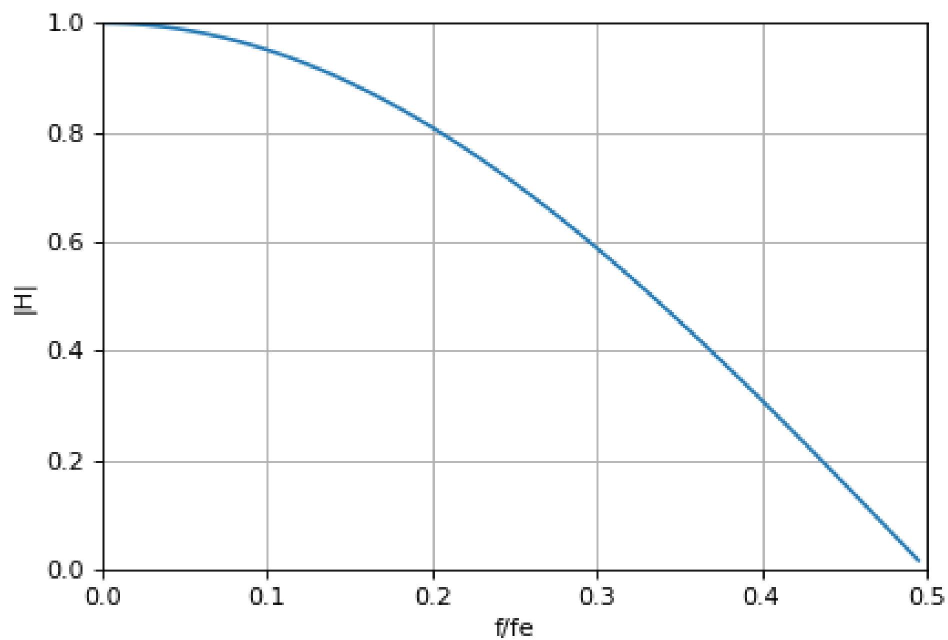
4.b. Filtre moyenneur

Ce filtre calcule la moyenne arithmétique de deux valeurs consécutives :

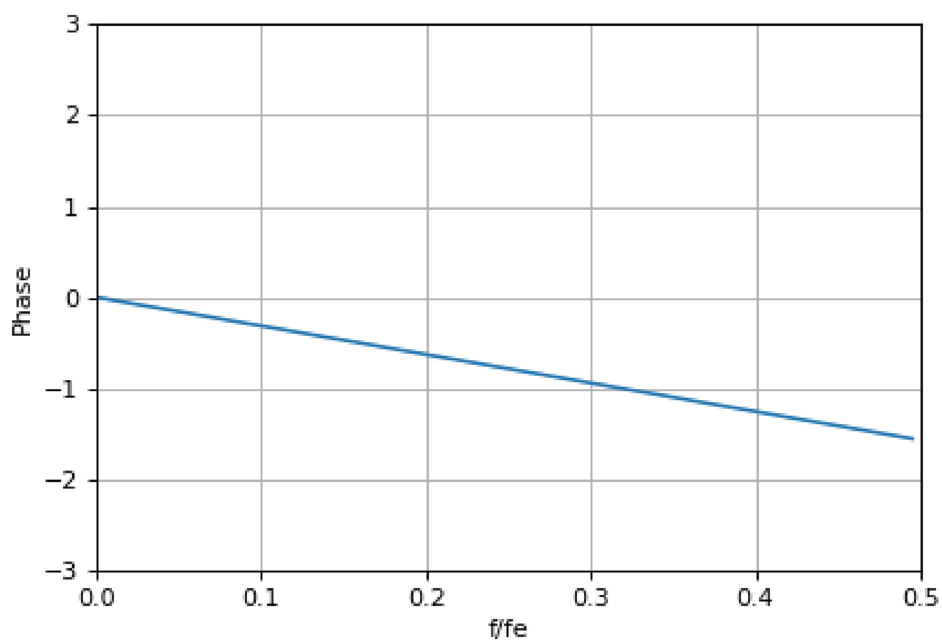
$$y_n = \frac{1}{2}(x_n + x_{n-1}) \tag{18}$$

On trace le module et l'argument de sa réponse fréquentielle :

```
b = numpy.array([0.5,0.5])
(f,g,phi)=reponseFreq(b,100)
figure(figsize=(6,4))
plot(f,g)
xlabel('f/fe')
ylabel('|H|')
axis([0,0.5,0,1])
grid()
```



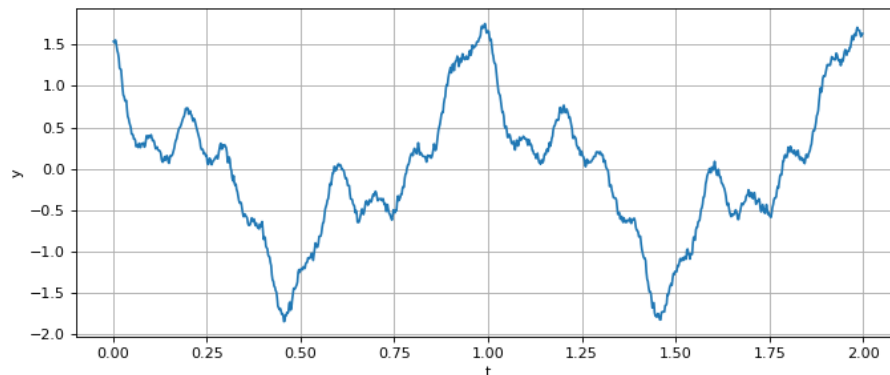

```
figure(figsize=(6,4))
plot(f,phi)
xlabel('f/fe')
ylabel('Phase')
axis([0,0.5,-3,3])
grid()
```



Le filtre moyenneur est un filtre passe-bas (peu sélectif). Le déphasage varie linéairement avec la fréquence.

Pour obtenir le signal discret filtré, il suffit d'effectuer la convolution avec la réponse impulsionnelle. Le premier point calculé correspond à l'instant T_e, c'est pourquoi l'échelle de temps doit être modifiée.

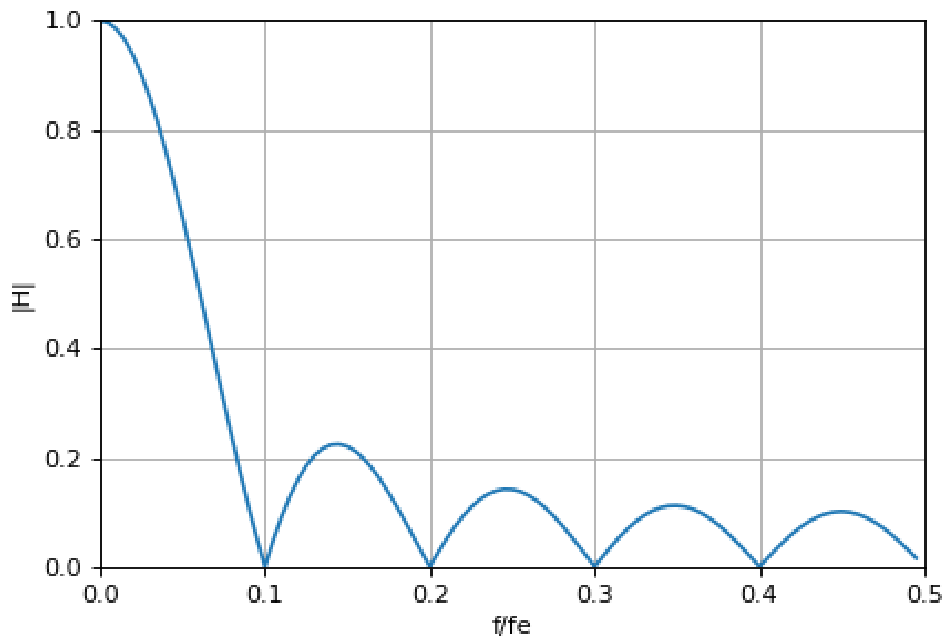
```
import scipy.signal
b = np.array([0.5,0.5])
y = scipy.signal.convolve(x,b,mode='valid')
ny = y.size
ty = te+np.arange(ny)*te

figure(figsize=(10,4))
plot(ty,y)
xlabel('t')
ylabel('y')
grid()
```



On voit que le bruit est légèrement réduit par ce filtre passe-bas. Pour augmenter l'efficacité du filtrage, il faut augmenter l'ordre N du filtre (la longueur de la réponse impulsionnelle).

Une première idée consiste à calculer une moyenne sur un plus grand nombre de points. Voici la réponse fréquentielle pour une moyenne sur 10 points :

```
b = numpy.ones(10)/10.0
(f,g,phi)=reponseFreq(b,100)
figure(figsize=(6,4))
plot(f,g)
xlabel('f/fe')
ylabel('|H|')
axis([0,0.5,0,1])
grid()
```



Il y a des rebonds dans la bande atténuée qui en font un très mauvais filtre. Nous allons voir comment modifier les coefficients de la réponse impulsionnelle pour obtenir une meilleure réponse.

4.c. Filtre passe-bas gaussien

Pour réduire le bruit d'un signal, on utilise le plus souvent un filtre gaussien, dont la réponse impulsionnelle est une gaussienne.

La gaussienne est la fonction définie par :

$$f(t) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{t^2}{2\sigma^2}\right) \tag{19}$$

On choisit une longueur de réponse impulsionnelle impaire $N = 2P + 1$, ce qui permet de centrer la gaussienne sur l'indice $k = P$. La réponse impulsionnelle est :

$$b_k = f(k - P) \tag{20}$$

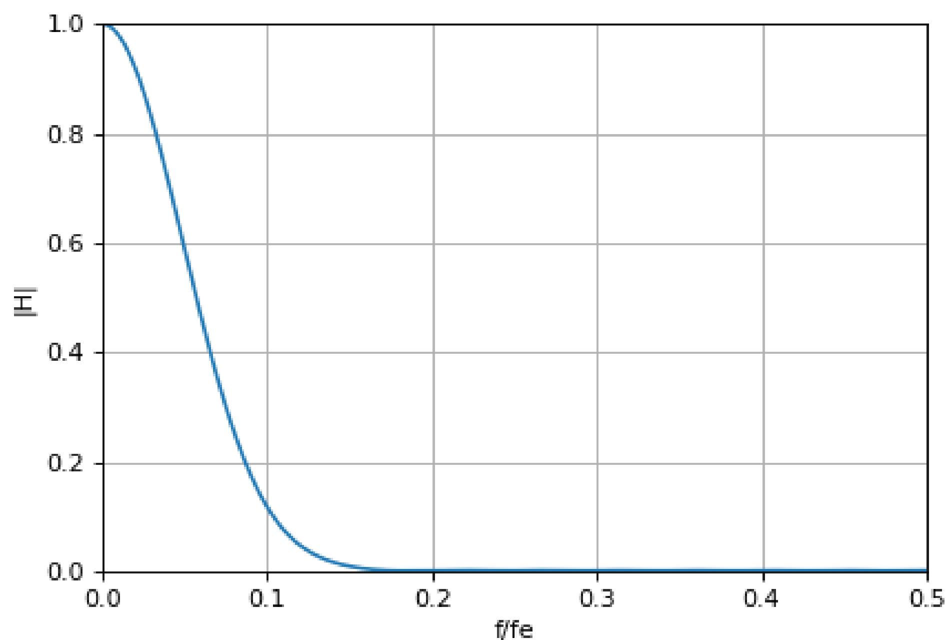
Si P est donné, on choisit σ pour avoir une valeur ϵ très faible sur les bords :

$$\sigma = \frac{P}{\sqrt{-2\ln\epsilon}} \tag{21}$$

La réponse impulsionnelle est normalisée pour que la somme de ses coefficients soit égale à 1. Voici un exemple :

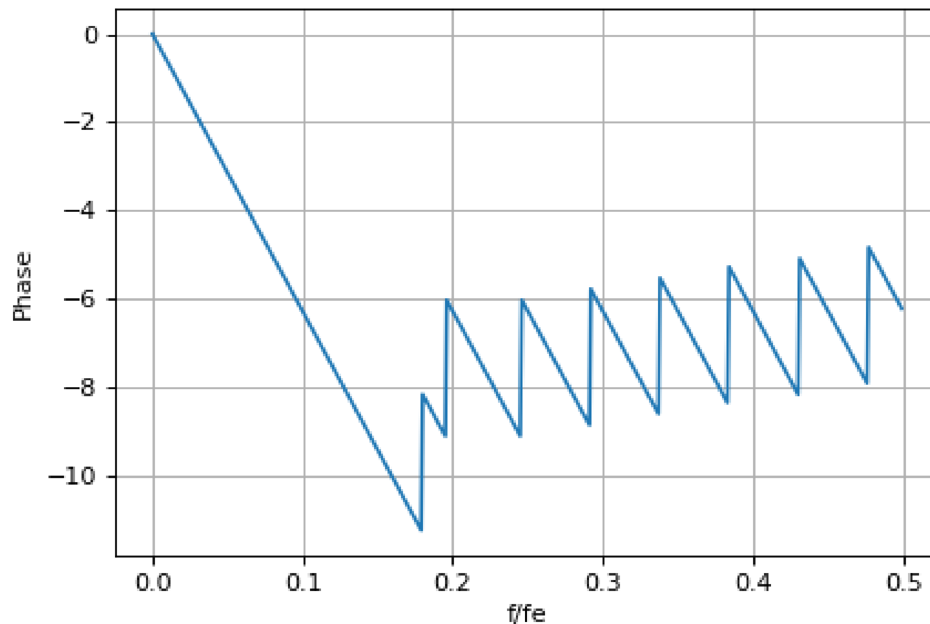
```
P_gauss=10
b_gauss = numpy.zeros(2*P_gauss+1)
epsilon=0.01
sigma=P_gauss/math.sqrt(-2.0*math.log(epsilon))
som = 0.0
for k in range(2*P_gauss+1):
```

```
    b_gauss[k] = math.exp(-(k-P_gauss)**2/(2*sigma**2))
    som += b_gauss[k]
b_gauss = b_gauss/som
(f,g,phi)=reponseFreq(b_gauss,500)
figure(figsize=(6,4))
plot(f,g)
xlabel('f/fe')
ylabel('|H|')
axis([0,0.5,0,1])
grid()
```



La réponse fréquentielle d'un filtre gaussien est aussi une gausienne. Voyons la phase :

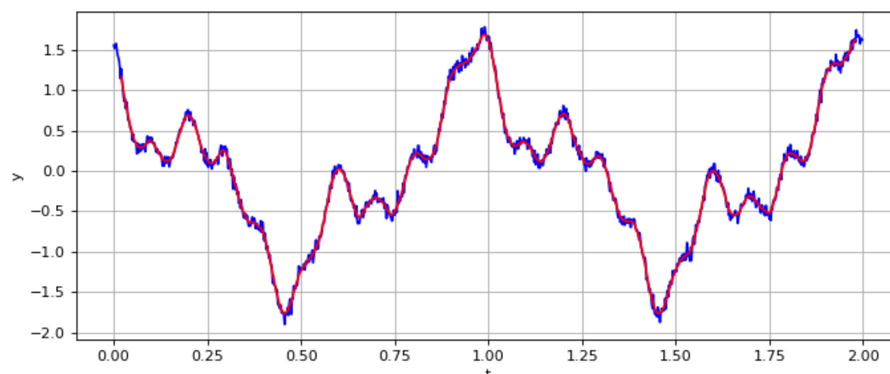
```
figure(figsize=(6,4))
plot(f,phi)
xlabel('f/fe')
ylabel('Phase')
grid()
```



Dans la bande passante (et même au delà), la phase varie linéairement avec la fréquence. Il y a donc un décalage temporel constant entre l'entrée et la sortie, égal à PT_e.

Voyons l'effet sur le signal bruité défini plus haut. L'échelle de temps est calculée de manière à annuler le décalage entre l'entrée et la sortie (cela n'est pas faisable dans un filtre fonctionnant en temps réel).

```
y = scipy.signal.convolve(x,b_gauss,mode='valid')
ny = y.size
ty = (np.arange(ny)+P_gauss)*te
figure(figsize=(10,4))
plot(t,x,'b')
plot(ty,y,'r')
xlabel('t')
ylabel('y')
grid()
```



L'usage de l'option `mode='valid'` dans l'appel de `scipy.signal.convolve` permet d'appliquer la convolution seulement sur les échantillons du signal qui ont assez de voisins à

droite et à gauche pour appliquer la convolution complète. En conséquence, le signal filtré est tronqué au début et à la fin.

```
print(len(x))
--> 1000

print(len(y))
--> 980
```

Le bruit est éliminé sans que la forme du signal ne soit modifiée. On remarque que P points au début et à la fin du signal ne sont pas traités. Dans un filtre temps-réel, cela ne pose pas de problème car les $2P$ points non filtrés sont au début du signal.

4.d. Filtre passe-bas en sinus cardinal

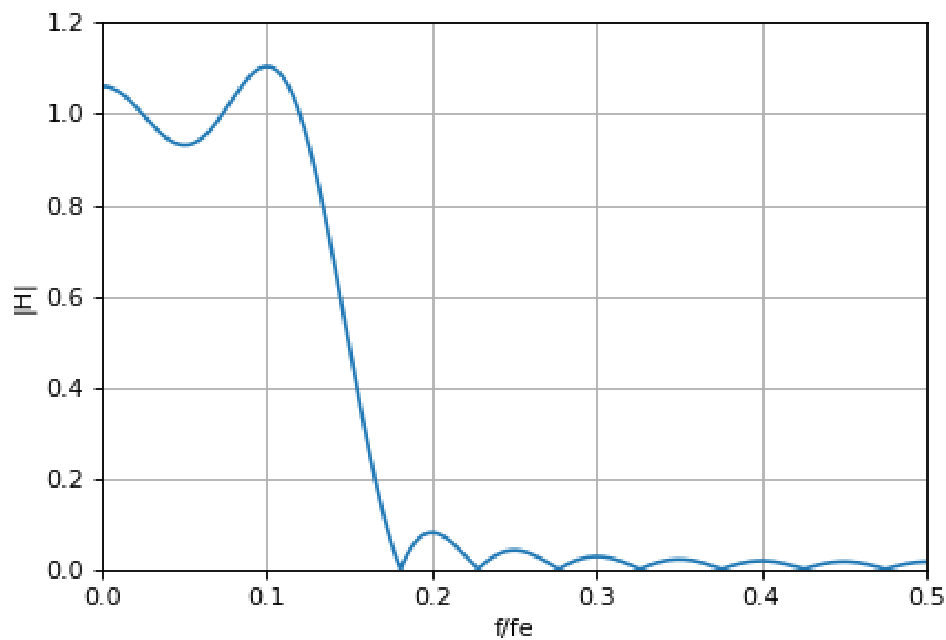
Le filtre passe-bas gaussien est efficace pour éliminer le bruit. Sur sa réponse fréquentielle, on voit en effet qu'il élimine totalement le bruit sur la bande de fréquence entre 0.15 et 0.5. En revanche, il n'est pas très sélectif. Une meilleure sélectivité signifie une décroissance plus rapide du gain après la coupure. Cela peut être obtenu par un filtre dont la réponse impulsionnelle est un sinus cardinal. On définit une fréquence de coupure f_c et on pose :

$$a = \frac{f_c}{f_e} \tag{22}$$

qui doit être inférieur à $1/2$. La réponse impulsionnelle est définie par :

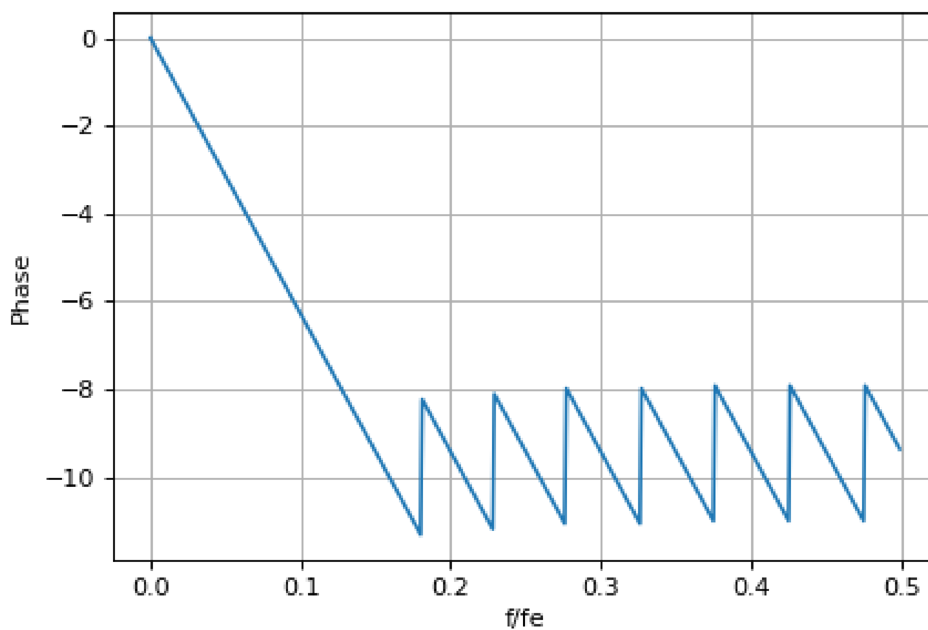
$$b_k = 2a\, sinc(2\pi(k-P)a) \tag{23}$$

```
P=10
b = numpy.zeros(2*P+1)
def sinc(u):
    if u==0:
        return 1.0
    else:
        return math.sin(u)/u
a=0.15
for k in range(2*P+1):
    b[k] = 2*a*sinc(2*math.pi*(k-P)*a)
(f,g,phi)=reponseFreq(b,500)
figure(figsize=(6,4))
plot(f,g)
xlabel('f/fe')
ylabel('|H|')
axis([0,0.5,0,1.2])
grid()
```



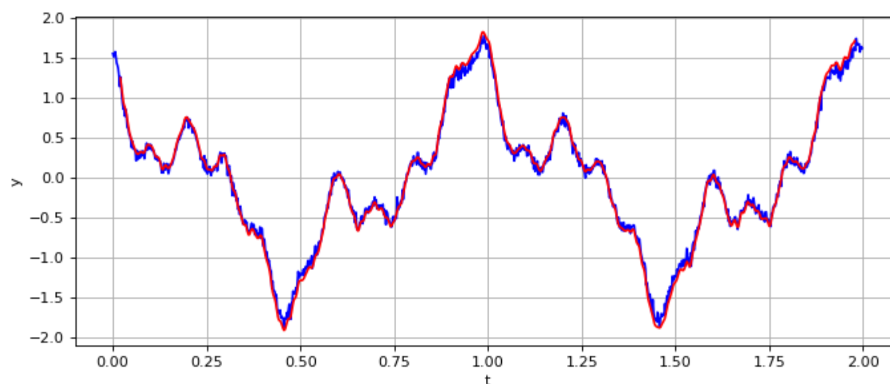
Ce filtre est beaucoup plus sélectif que le filtre gaussien, mais a l'inconvénient de comporter des ondulations dans la bande passante et dans la bande atténuée.

```
figure(figsize=(6,4))
plot(f,phi)
xlabel('f/fe')
ylabel('Phase')
grid()
```



La phase est bien linéaire dans la bande passante. Voyons l'effet sur le signal :

```
y = scipy.signal.convolve(x,b,mode='valid')
ny = y.size
ty = (np.arange(ny)+P)*te
figure(figsize=(10,4))
plot(t,x,'b')
plot(ty,y,'r')
xlabel('t')
ylabel('y')
grid()
```



Ce filtre est moins efficace que le filtre gaussien pour réduire le bruit, en raison de la présence de rebonds dans la bande atténuée.

4.e. Filtre dérivateur

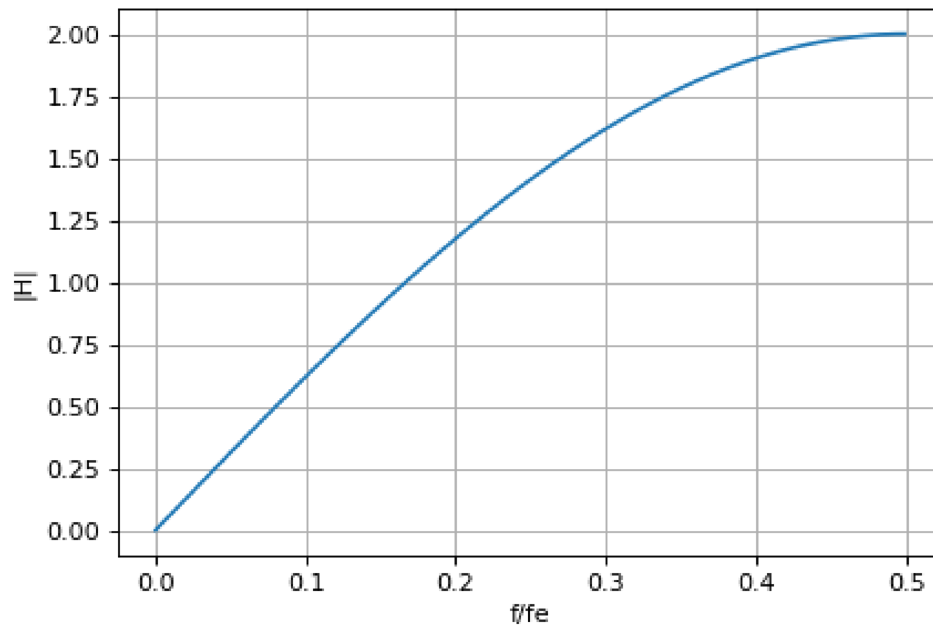
La dérivation est une opération courante en traitement du signal. Elle est utilisée pour calculer la vitesse d'un phénomène, ou détecter des variations rapides dans un signal ou dans une image.

La manière la plus simple de définir une dérivation est :

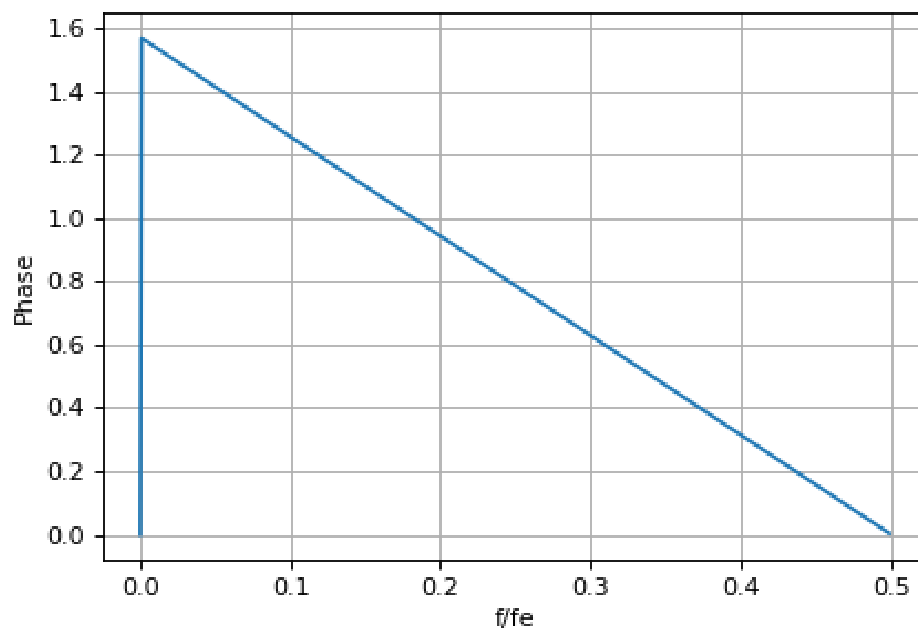
$$y_n = \frac{1}{T_e}(x_n - x_{n-1}) \tag{24}$$

Sa réponse impulsionnelle est $(1, -1)$. Voyons sa réponse fréquentielle :

```
b = numpy.array([1.0,-1.0])
(f,g,phi)=reponseFreq(b,500)
figure(figsize=(6,4))
plot(f,g)
xlabel('f/fe')
ylabel('|H|')
grid()
```

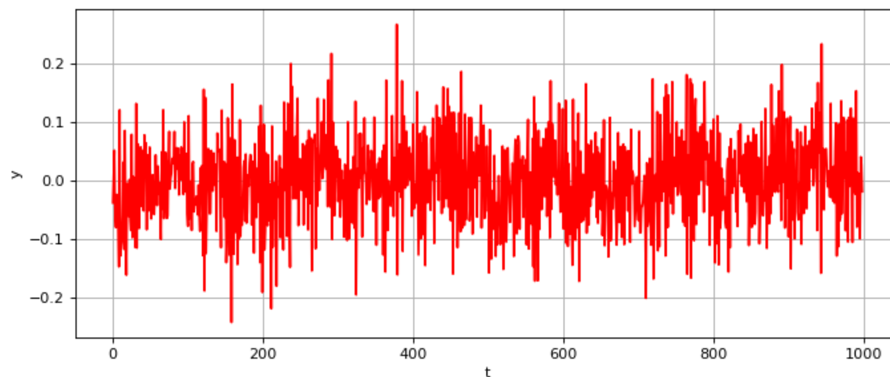



```
figure(figsize=(6,4))
plot(f,phi)
xlabel('f/fe')
ylabel('Phase')
grid()
```



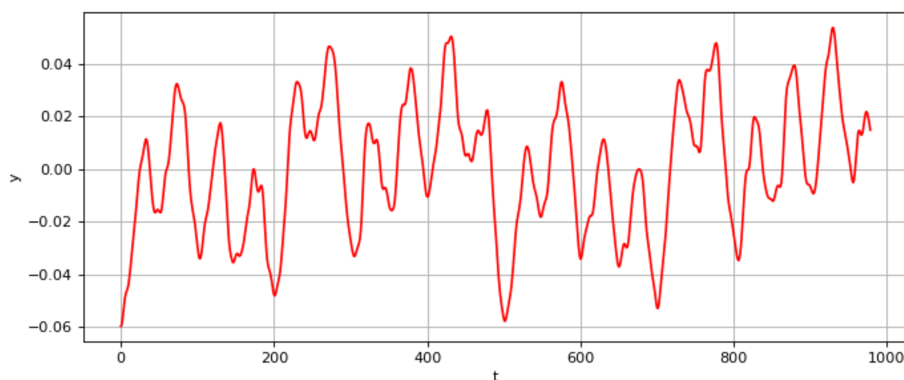
Le filtre dérivateur est passe-haut. Voyons son effet sur le signal bruité :

```
y_deriv = scipy.signal.convolve(x,b,mode='valid')
ny = y_deriv.size
ty = te+np.arange(ny)
figure(figsize=(10,4))
plot(ty,y_deriv,'r')
xlabel('t')
ylabel('y')
grid()
```



Le bruit est tellement amplifié par le filtre dérivateur qu'il couvre complètement le signal utile. Cela est dû au caractère passe-haut du dérivateur. Pour dériver un signal bruité, il faut impérativement effectuer un filtrage passe-bas au préalable, par exemple un filtrage gaussien :

```
y = scipy.signal.convolve(x,b_gauss,mode='valid')
y_deriv = scipy.signal.convolve(y,b,mode='valid')
ny = y_deriv.size
ty = np.arange(ny)+(P_gauss+1)*te
figure(figsize=(10,4))
plot(ty,y_deriv,'r')
xlabel('t')
ylabel('y')
grid()
```



5. Filtre intégrateur

Un filtre intégrateur peut être obtenu par la relation suivante :

$$y_n = y_{n-1} + T_e \frac{x_n + x_{n-1}}{2} \quad (25)$$

Voir Filtres intégrateur et dérivateur pour la justification de cette relation.

Cette relation est différente de la relation (2) d'un filtre à réponse impulsionnelle finie : la sortie à l'instant n dépend de l'état de la sortie à l'instant $n-1$. Ce type de filtre est appelé filtre récursif. Pour l'appliquer à un signal, il faut fixer une condition initiale, ici la valeur y_0.

Appliquons ce filtre à l'impulsion unité, avec la condition initiale $y_0 = 0$. On obtient :

$$y_1 = y_0 + \frac{x_0}{T_e} = \frac{1}{T_e} \quad (26)$$

$$y_2 = y_1 = \frac{1}{T_e} \quad (27)$$

$$y_3 = y_2 = \frac{1}{T_e} \quad (28)$$

$$\ldots \quad (29)$$

La réponse impulsionnelle est un échelon. C'est un exemple de réponse impulsionnelle infinie. Un filtre est stable si sa réponse impulsionnelle tend vers zéro. Le filtre intégrateur est donc instable, mais sa réponse impulsionnelle est tout de même bornée (stabilité marginale).

La réponse fréquentielle de ce filtre peut être obtenue en suivant la même méthode qu'en 3.b. On obtient ainsi la fonction de transfert en Z :

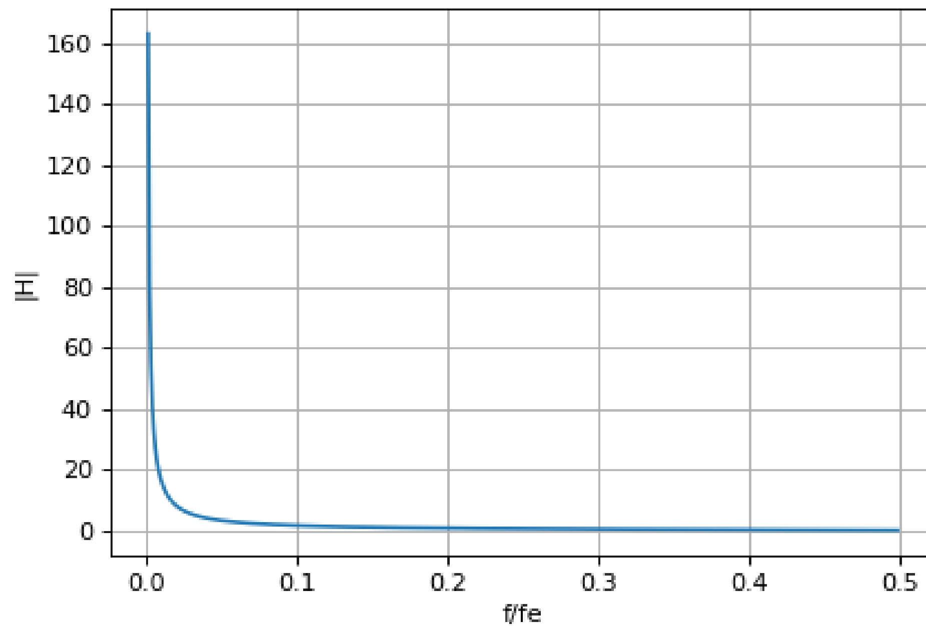
$$H(Z) = \frac{T_e}{2} \frac{1 + Z^{-1}}{1 - Z^{-1}} \quad (30)$$

et la réponse fréquentielle se déduit en posant :

$$Z = \exp(i2\pi f T_e) \quad (31)$$

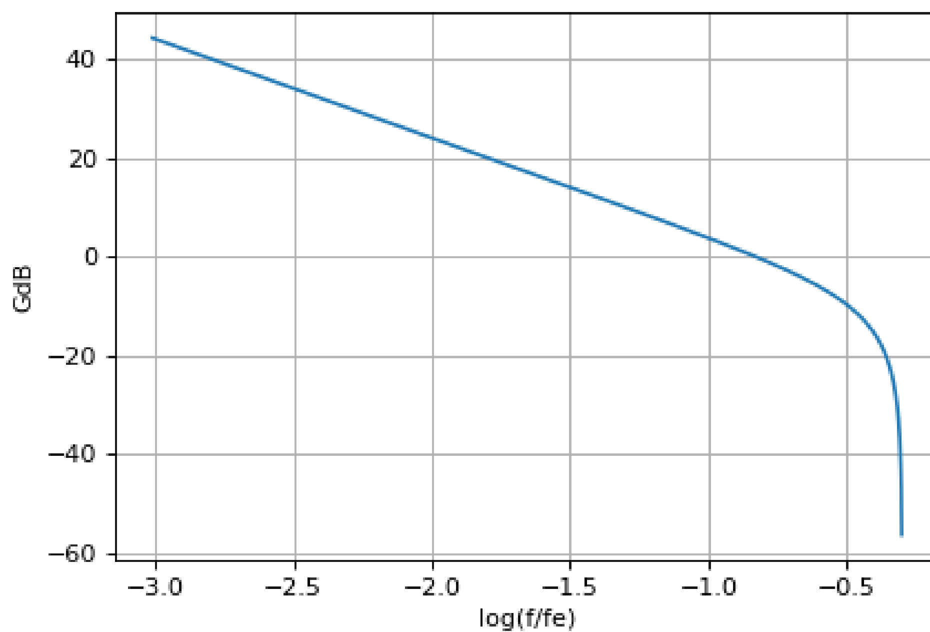
D'une manière générale, la fonction de transfert en Z d'un filtre linéaire est une fraction rationnelle en Z^{-1}. La fonction `scipy.signal.freqz` permet d'obtenir la réponse fréquentielle d'un filtre défini par sa fonction de transfert en Z :

```
b=[0.5,0.5]
a=[1,-1]
[w,h] = scipy.signal.freqz(b,a)
figure(figsize=(6,4))
plot(w/(2*math.pi),numpy.abs(h))
xlabel('f/fe')
ylabel('|H|')
grid()
```



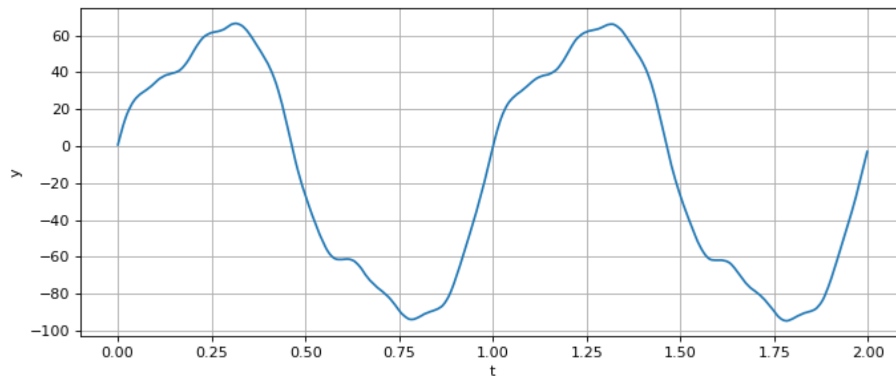
L'échelle logarithmique (diagramme de Bode) est préférable :

```
figure(figsize=(6,4))
plot(numpy.log10(w/(2*math.pi)),20*numpy.log10(numpy.abs(h)))
xlabel('log(f/fe)')
ylabel('GdB')
grid()
```



Voyons l'application de ce filtre au signal bruité. La fonction `scipy.signal.lfilter` permet de le faire. La condition initiale par défaut est nulle.

```
y = scipy.signal.lfilter(b,a,x)
figure(figsize=(10,4))
plot(t,y)
xlabel('t')
ylabel('y')
grid()
```



Au contraire du filtre dérivateur, le filtre intégrateur réduit le bruit, en raison de son caractère passe-bas.

Références

[1] M. Bellanger, *Traitement numérique du signal*, (Dunod, 1998)