Filtres à réponse impulsionnelle infinie

1. Définition

Soit $x(t)$ un signal continu échantillonné avec une période T_e. Le signal discret correspondant est la suite

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

La transformée en Z ([1],[2]) du signal discret est définie par :

$$X(z) = \sum_{n=-\infty}^{n=+\infty} x_n z^{-n} \tag{2}$$

Un filtre linéaire causal (invariant dans le temps) est défini par une relation de récurrence de la forme :

$$a_0 y_n = \sum_{k=0}^{N-1} b_k x_{n-k} - \sum_{k=1}^{M-1} a_k y_{n-k} \tag{3}$$

Lorsque les coefficients a_k sont tous nuls, il s'agit d'un filtre à réponse impulsionnelle finie. Dans le cas contraire, le calcul de la sortie y_n à l'instant n dépend de l'état de la sortie aux instants antérieurs.

La transformée en Z d'un signal décalé dans le temps s'écrit :

$$TZ(x_{n-k}) = z^{-k} X(z) \tag{4}$$

En appliquant la transformée en Z à la relation de récurrence du filtre, on obtient la fonction de transfert en Z :

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots} \tag{5}$$

2. Synthèse par transformation bilinéaire

2.a. Principe

Le point de départ est la fonction de transfert en Z d'un filtre intégrateur :

$$H(z) = \frac{T_e}{2} \frac{1 + z^{-1}}{1 - z^{-1}} \tag{6}$$

La fonction de transfert d'un intégrateur analogique est :

$$H(s) = \frac{1}{s} \tag{7}$$

L'identification entre ces deux fonctions de transfert permet de définir une transformation de z vers s (transformation bilinéaire du plan complexe) :

$$s = \frac{2}{T_e} \frac{z - 1}{z + 1} \tag{8}$$

Le demi-plan $(re(s) < 0)$ du plan s se transforme en le disque unité du plan z. En introduisant la pulsation analogique ω_a et la pulsation numérique ω_n, on obtient :

$$j\omega_a = \frac{2}{T_e}\frac{\exp(j\omega_n T_e) - 1}{\exp(j\omega_n T_e) + 1} \tag{9}$$

Après simplification on obtient :

$$\omega_a = \frac{2}{T_e}\tan\left(\frac{\omega_n T_e}{2}\right) \tag{10}$$

Sachant que la pulsation numérique doit vérifier :

$$\omega_n < \frac{2\pi}{2T_e} \tag{11}$$

on a

$$\frac{\omega_n T_e}{2} < \frac{\pi}{2} \tag{12}$$

La méthode de synthèse de filtre par transformation bilinéaire consiste à définir un filtre analogique $H(s)$ avec des fréquences de coupure analogiques obtenues par la relation (10), en fonction des fréquences de coupure numériques souhaitées. Le changement de variable défini par la transformation bilinéaire (8) permet d'obtenir la fonction de transfert en Z du filtre numérique. On doit ensuite étudier la stabilité du filtre et tracer sa réponse fréquentielle.

2.b. Exemple : filtre du premier ordre

On considère un filtre passe-bas analogique du premier ordre :

$$H(s) = \frac{1}{1 + \frac{s}{\omega_a}} \tag{13}$$

On souhaite une fréquence de coupure numérique égale au quart de la fréquence de Nyquist (huitième de la fréquence d'échantillonnage) :

$$\frac{\omega_n T_e}{2} = \frac{\pi}{8} \tag{14}$$

La pulsation de coupure analogique correspondante est :

$$\omega_a = \frac{2}{T_e}\tan\left(\frac{\pi}{8}\right) = \frac{2}{T_e}\alpha \tag{15}$$

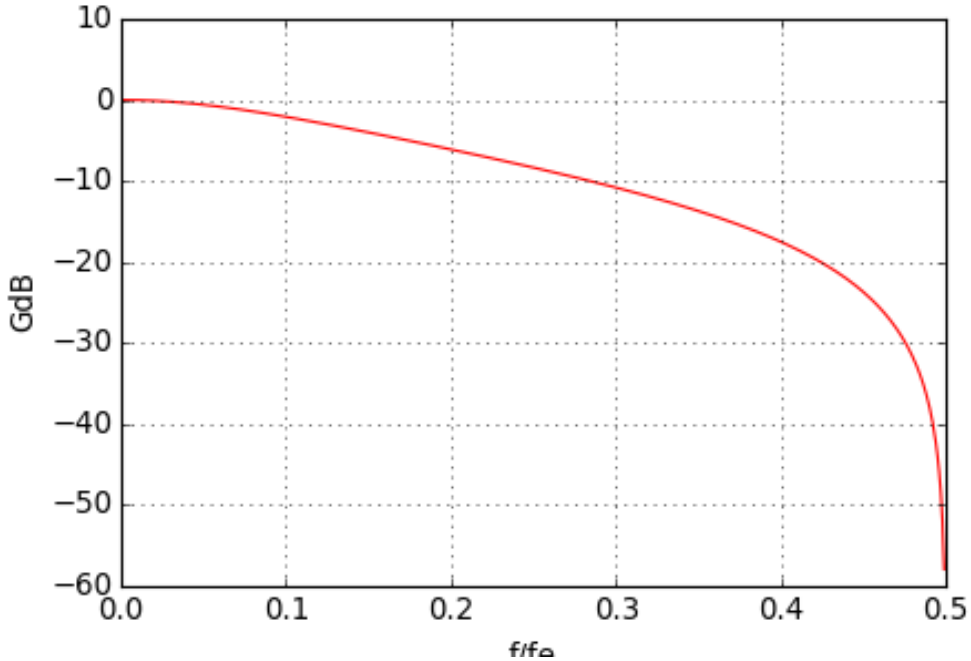
L'application de la transformation bilinéaire conduit à :

$$H(z) = \frac{\alpha(z+1)}{\alpha(z+1) + z - 1} = \frac{\alpha + \alpha z^{-1}}{(\alpha+1) + (\alpha-1)z^{-1}} \tag{16}$$

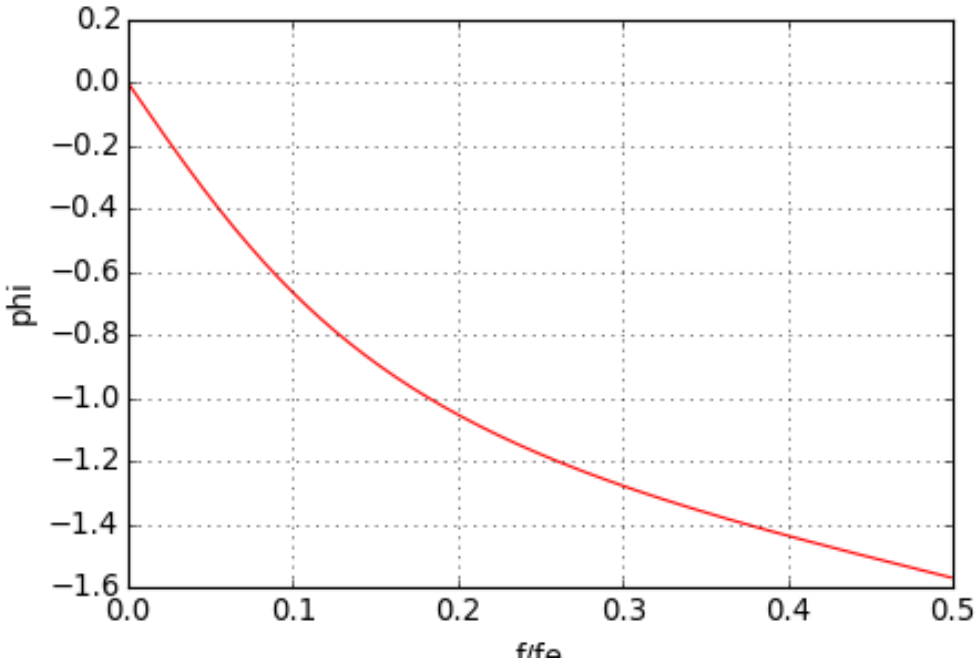
La réponse fréquentielle est obtenue en posant $z = \exp(j2\pi\, fT_e)$. Elle peut être obtenue avec la fonction `scipy.signal.freqz`. Pour cela, il faut fournir la fonction de transfert en Z sous la forme :

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots} \tag{17}$$

```
import numpy
import math
from matplotlib.pyplot import *
import scipy.signal
alpha = math.tan(math.pi/8)
b=[alpha,alpha]
a=[alpha+1,alpha-1]
[w,h] = scipy.signal.freqz(b,a)
figure(figsize=(6,4))
plot(w/(2*math.pi),20*numpy.log10(numpy.abs(h)),'r')
xlabel('f/fe')
ylabel('GdB')
grid()
```




```
figure(figsize=(6,4))
plot(w/(2*math.pi),numpy.angle(h),'r')
xlabel('f/fe')
ylabel('phi')
grid()
```



La fréquence de coupure est $1/8 = 0.125$. On remarque que la phase n'est pas tout à fait linéaire dans la bande passante, contrairement aux filtres RIF à phase linéaire.

Pour obtenir la réponse impulsionnelle du filtre, il faut tout d'abord décomposer la fonction de transfert en fractions rationnelles simples. La fonction `scipy.signal.residuez` fournit cette décomposition sous la forme suivante :

$$\frac{B(z)}{A(z)} = \frac{r_0}{1 - p_0 z^{-1}} + \frac{r_1}{1 - p_1 z^{-2}} + \cdots + k_0 + k_1 z^{-1} + \cdots$$

```
[r,p,k] = scipy.signal.residuez(b,a)
```

```
print(r)
--> array([ 1.])

print(p)
--> array([ 0.41421356])

print(k)
--> array([-0.70710678])
```

On a donc :

$$H(z) = \frac{z}{z - p_0} + k_0 \tag{18}$$

La réponse impulsionnelle est la transformée en Z inverse de la fonction de transfert. En consultant un tableau des transformées en Z usuelles ([3]), on obtient la réponse impulsionnelle :

$$h_n = p_0^n u_n + k_0 \delta_n \tag{19}$$

où u_n est l'échelon unité et δ_n l'impulsion unité.

La réponse impulsionnelle est infinie (pour $n > 0$ on a $h_n = p_0^n$). Comme $p_0 < 1$, la réponse impusionnelle tend vers zéro, ce qui démontre la stabilité du filtre.

Si l'on intéresse seulement à la stabilité du filtre, il suffit de calculer ses pôles :

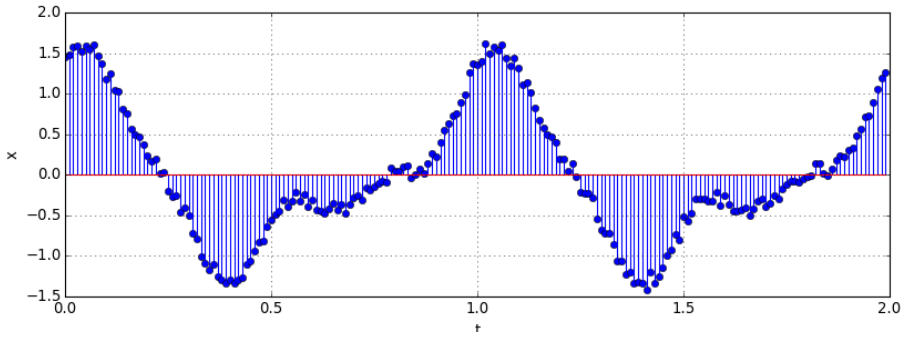
```
(zeros,poles,gain) = scipy.signal.tf2zpk(b,a)


print(poles)
--> array([ 0.41421356])
```

Le filtre est stable si ses pôles ont un module strictement inférieur à 1.

Comme exemple d'application, nous allons filtrer un signal de période 1 comportant du bruit haute. La fréquence d'échantillonnage est choisie supérieure à 2 fois la plus grande fréquence du spectre.

```
import random
def signal(t):
    return numpy.cos(2*math.pi*t)+0.5*numpy.cos(2*2*math.pi*t-math.pi/3)+0.2*numpy.co
fe = 100
te=1.0/fe
t = numpy.arange(start=0.0,stop=2.0,step=te)
n = t.size
x = numpy.zeros(n)
for k in range(n):
    x[k] = signal(te*k)
figure(figsize=(12,4))
stem(t,x)
xlabel('t')
ylabel('x')
grid()
```



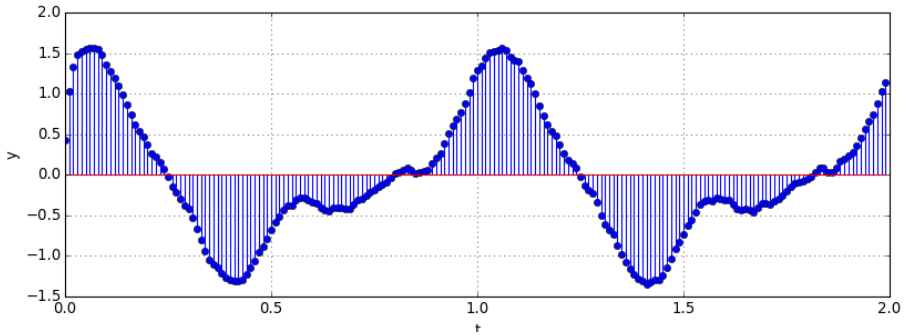
Le filtre passe-bas précédent a une fréquence de coupure égale au huitième de la fréquence d'échantillonnage, soit 12.5. Toutes les harmoniques du signal sont donc dans la bande passante.

Pour réaliser le filtrage, il faut appliquer la relation de récurrence suivante :

$$y_n = \frac{1}{\alpha+1}\left((\alpha-1)y_{n-1} + \alpha x_n + \alpha x_{n-1}\right) \tag{20}$$

La fonction `scipy.signal.lfilter` effectue ce calcul. Pour calculer le premier élément y_0, il faut disposer des valeurs de y_{-1} et de x_{-1}, qui constituent des conditions initiales.

```
zi = scipy.signal.lfiltic(b,a,y=[0],x=[0])
[y,zf] = scipy.signal.lfilter(b,a,x,axis=-1,zi=zi)
figure(figsize=(12,4))
stem(t,y)
xlabel('t')
ylabel('y')
grid()
```



2.c. Exemple : filtre de Butterworth

Le filtre de Butterworth passe-bas d'ordre 2 est défini par :

$$H(s) = \frac{1}{s^2 + \sqrt{2}s + 1} \tag{21}$$

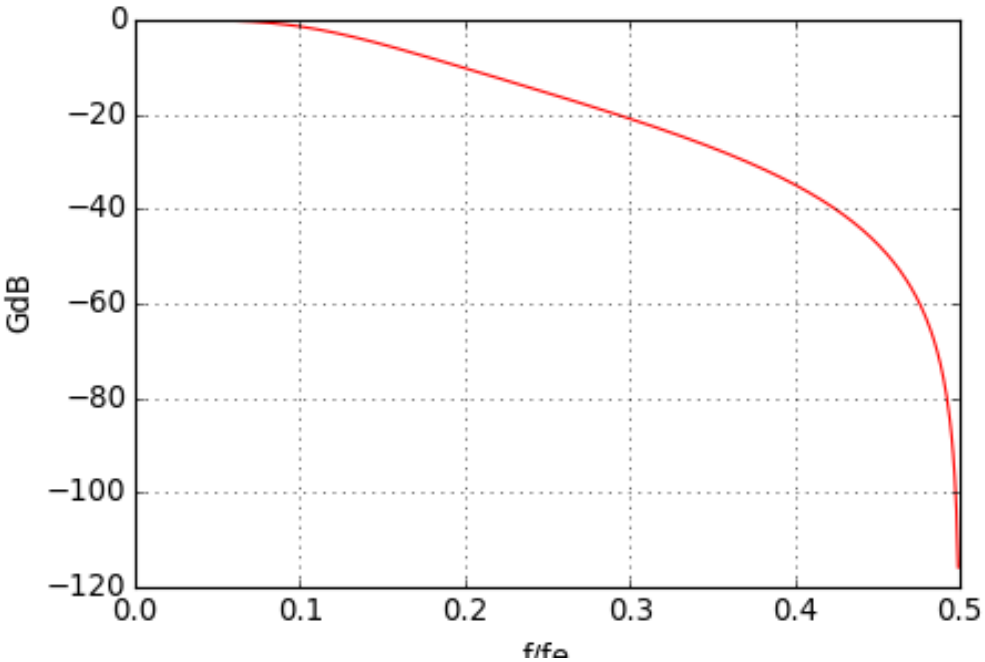
La fonction `scipy.signal.bilinear` effectue la transformation bilinéaire suivante :

$$s = 2\frac{z-1}{z+1} \tag{22}$$

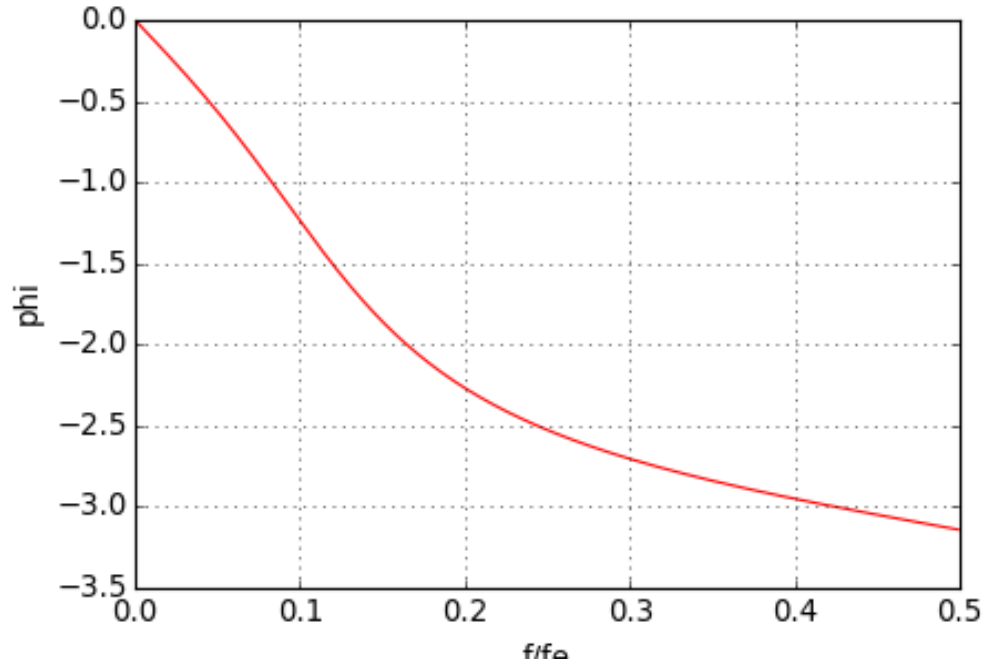
Attention : le facteur 2 n'est pas mentionné dans la documentation officielle de scipy.

Il faut tout d'abord introduire la fréquence de coupure analogique ω_a dans la fonction de transfert $H(s)$, ce qui consiste à remplacer s par $\frac{sT_e}{2\alpha}$. Comme la transformation bilinéaire (22) s'applique à sT_e, il suffit de remplacer s par $s/(2\alpha)$:

```
alpha = math.tan(math.pi/8)
b=[1]
a=[1.0/(2*alpha)**2,math.sqrt(2)/(2*alpha),1]
[b,a]=scipy.signal.bilinear(b,a)
[w,hf] = scipy.signal.freqz(b,a)
figure(figsize=(6,4))
plot(w/(2*math.pi),20*numpy.log10(numpy.abs(hf)),'r')
xlabel('f/fe')
ylabel('GdB')
grid()
```




```
figure(figsize=(6,4))
plot(w/(2*math.pi),numpy.angle(hf),'r')
xlabel('f/fe')
ylabel('phi')
grid()
```



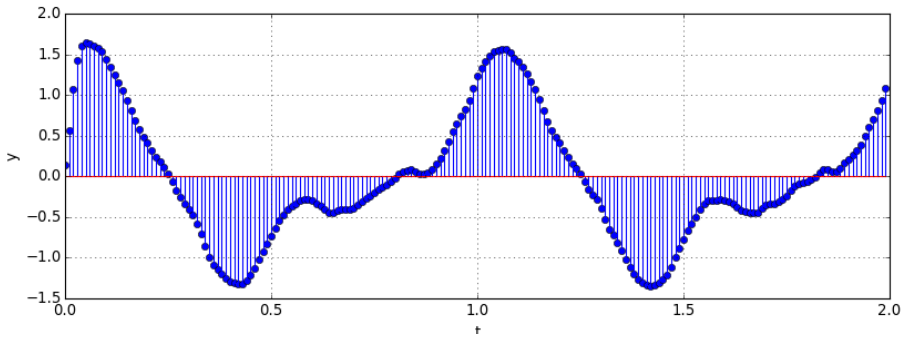
Voici les coefficients du filtre :

```
print(b)
--> array([ 0.09763107,  0.19526215,  0.09763107])
```

```
print(a)
--> array([ 1.         , -0.94280904,  0.33333333])
```

Application du filtre au signal bruité précédent :

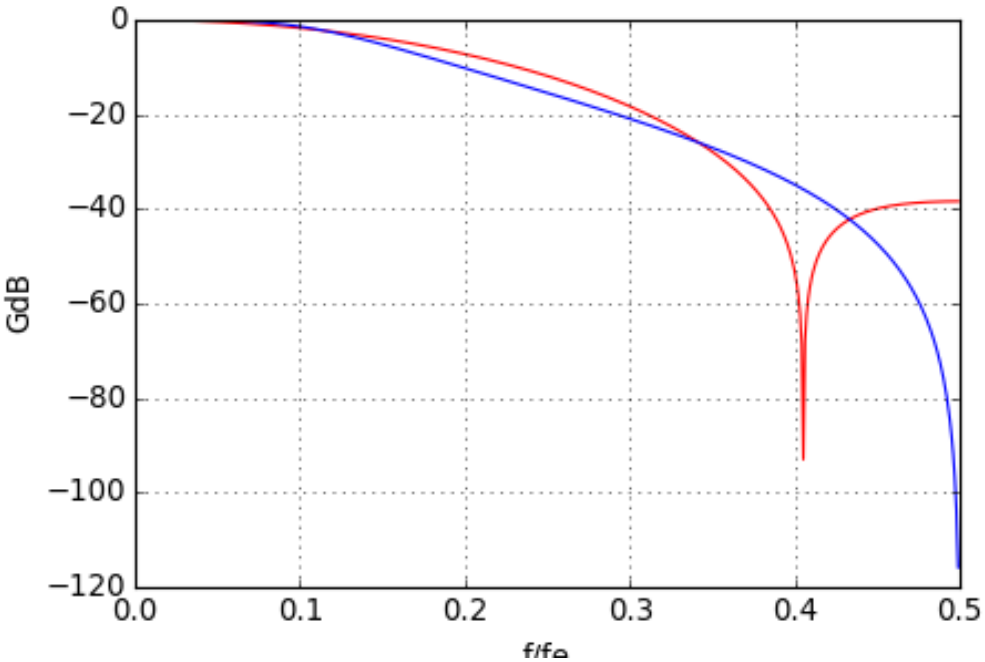
```
zi = scipy.signal.lfiltic(b,a,y=[0,0],x=[0,0])
[y,zf] = scipy.signal.lfilter(b,a,x,axis=-1,zi=zi)
figure(figsize=(12,4))
stem(t,y)
xlabel('t')
ylabel('y')
grid()
```



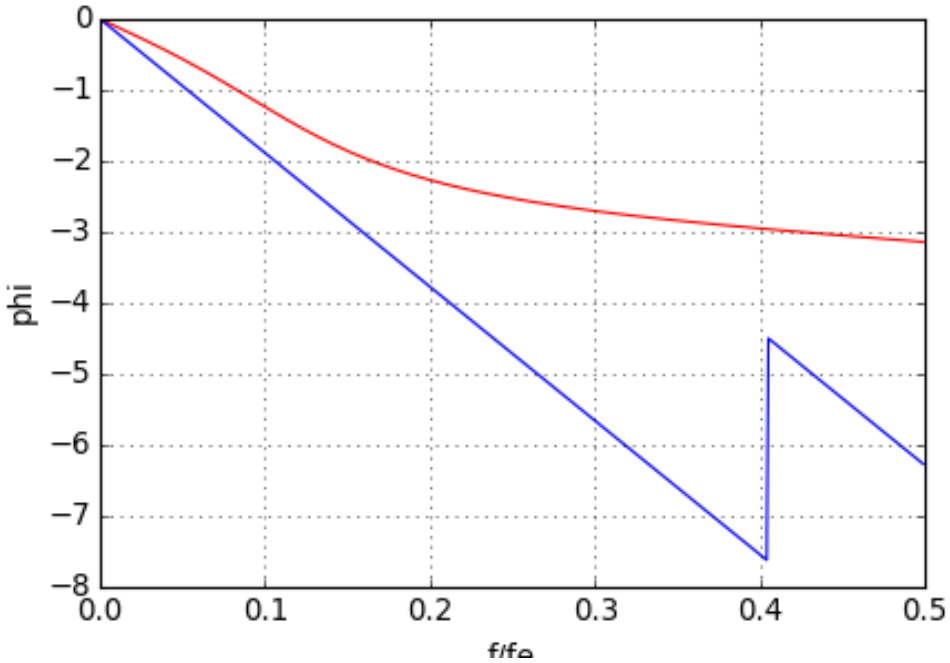
Comme prévu, la qualité du filtrage est bien meilleure qu'avec le filtre d'ordre 1.

Il est intéressant de comparer ce filtre avec un filtre à réponse impulsionnelle finie :

```
P=3
h = scipy.signal.firwin(numtaps=2*P+1,cutoff=[1.0/8],nyq=0.5,window='hann')
b=h
a=[1]
[w1,hf1] = scipy.signal.freqz(b,a)
figure(figsize=(6,4))
plot(w1/(2*math.pi),20*numpy.log10(numpy.abs(hf1)),'r')
plot(w/(2*math.pi),20*numpy.log10(numpy.abs(hf)),'b')
xlabel('f/fe')
ylabel('GdB')
grid()
```




```
figure(figsize=(6,4))
plot(w/(2*math.pi),numpy.unwrap(numpy.angle(hf)),'r')
plot(w1/(2*math.pi),numpy.unwrap(numpy.angle(hf1)),'b')
xlabel('f/fe')
ylabel('phi')
grid()
```



Pour une atténuation similaire il faut un filtre RIF à 7 coefficients au lieu de 3 pour le filtre RII. Le filtre RII ne comporte pas d'ondulations dans la bande atténuée. En

revanche, la phase n'est pas linéaire dans la bande passante pour le filtre RII, ce qui peut causer une distorsion des signaux.

2.d. Utilisation de scipy.signal.iirfilter

La fonction `scipy.signal.iirfilter` permet de calculer un filtre IIR à partir de filtres de Butterworth, Chebyshev, etc. Cette même fonction permet d'ailleurs de calculer la fonction de transfert d'un filtre analogique.

Pour un filtre de Butterworth, il suffit de préciser l'ordre du filtre et les fréquences de coupures (divisées par la fréquence de Nyquist). Voici par exemple le calcul d'un filtre passe-bas de Butterworth numérique d'ordre 4 :

```
fc = 0.2 #rapport fc sur frequence d'echantillonnage
(b,a)=scipy.signal.iirfilter(4,[fc*2],btype="lowpass",ftype="butter")
```

Voici les coefficients du numérateur de la fonction de transfert en Z :

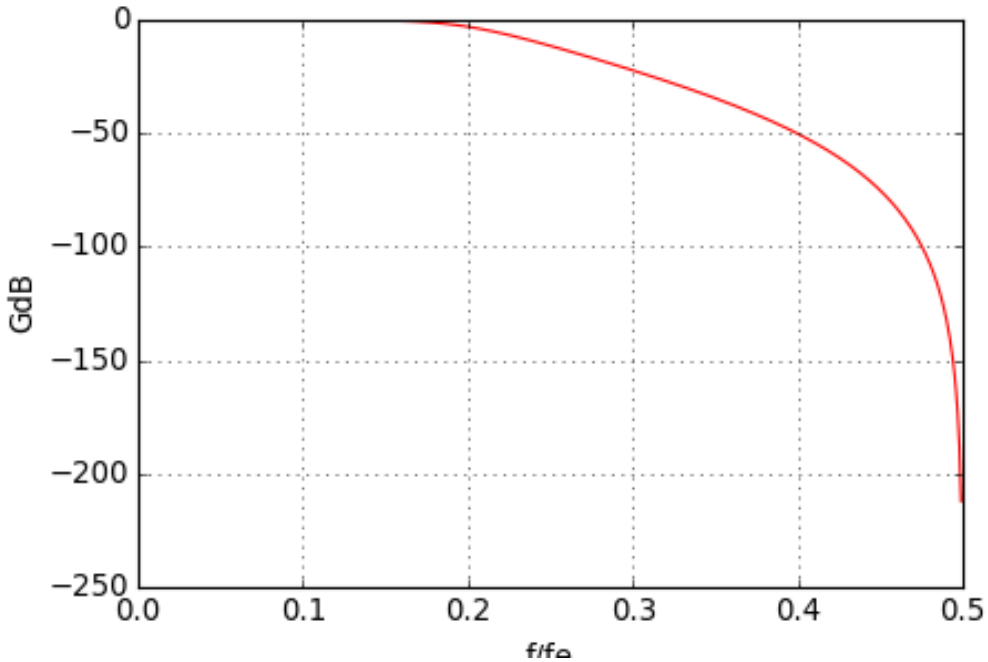
```
print(b)
--> array([ 0.04658291,  0.18633163,  0.27949744,  0.18633163,  0.04658291])
```

et ceux du dénominateur :

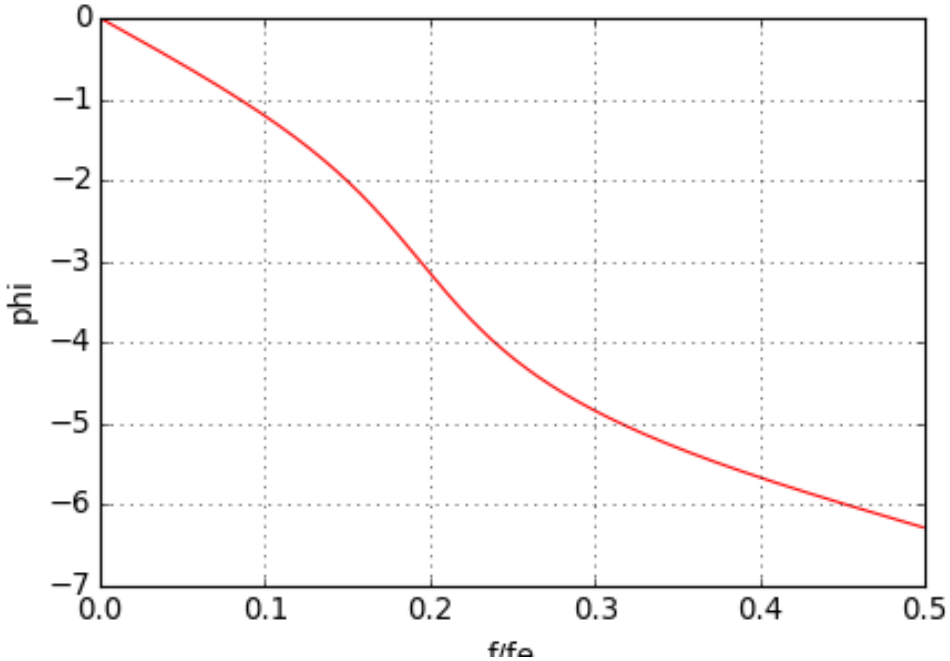
```
print(a)
--> array([ 1.        , -0.7820952 ,  0.67997853, -0.1826757 ,  0.03011888])
```

On trace la réponse fréquentielle :

```
[w,hf] = scipy.signal.freqz(b,a)
figure(figsize=(6,4))
plot(w/(2*math.pi),20*numpy.log10(numpy.abs(hf)),'r')
xlabel('f/fe')
ylabel('GdB')
grid()
```




```
figure(figsize=(6,4))
plot(w/(2*math.pi),numpy.unwrap(numpy.angle(hf)),'r')
xlabel('f/fe')
ylabel('phi')
grid()
```



Références

[1] M. Bellanger, *Traitement numérique du signal*, (Dunod, 1998)
[2] E. Tisserand, J.F. Pautex, P. Schweitzer, *Analyse et traitement des signaux*, (Dunod, 2008)
[3] Tan Li, Jiang Jean, *Digital signal processing : fundamentals and applications*, (Elsevier, 2013)