Filtres passe-bande analogique et numérique

1. Introduction

Les filtres passe-bande permettent de sélectionner une bande de fréquence dans un signal. Ils sont particulièrement utilisés en radio-communication (télévision, téléphonie, etc), pour sélectionner une bande de fréquence contenant l'information que l'on souhaite décoder. En traitement du signal audio, ils sont utilisés dans les égaliseurs, qui permettent par exemple d'équilibrer les signaux issus des différents microphones lors d'une prise de son.

Les filtres passe-bandes ont pendant longtemps été réalisés sous forme analogique. Aujourd'hui, ces filtres sont supplantés par leur équivalent numérique, bien plus facile à configurer et offrant une sélectivité supérieure.

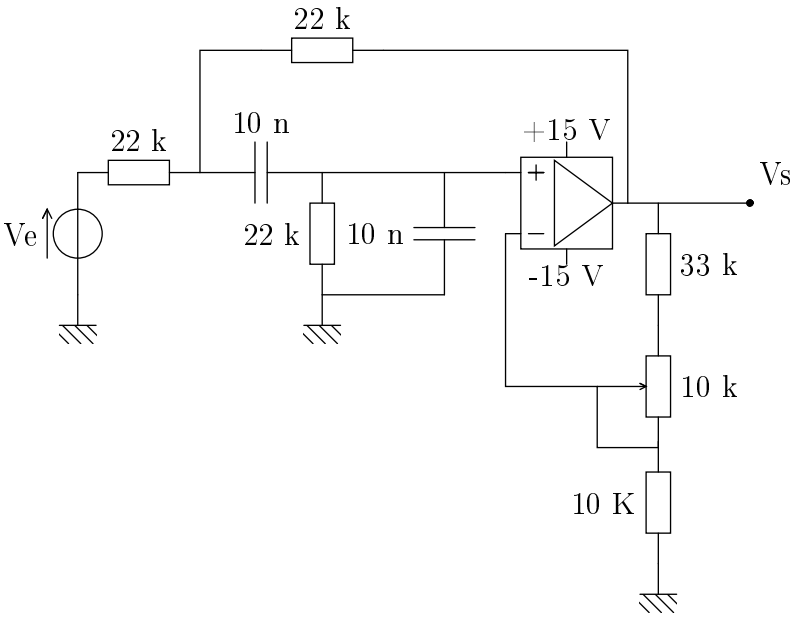
Ce document présente un exemple de filtre passe-bande analogique, qui permet de sélectionner une bande de fréquence très étroite. On verra l'effet de ce filtre sur un signal périodique.

Dans un deuxième temps, nous verrons comment réaliser un filtre numérique passe-bande, sous forme de filtre à réponse impulsionnelle finie.

2. Filtre analogique

2.a. Définition

Le filtre utilisé est de type Sallen et Key. Voici son schéma :



Pour une étude complète de ce filtre, voir le document Filtres actifs de Sallen et Key. Le bloc constitué de l'amplificateur, des deux résistances et du potentiomètre à sa droite est un amplificateur de gain K, ajustable entre 4.3 et 5.3 avec le potentiomètre.

La fréquence pour laquelle le gain est maximal est :

$$f_0 = \frac{\sqrt{2}}{2\pi RC} = 1023\ Hz \tag{1}$$

La gain maximal est :

$$A = \frac{K}{5-K} \tag{2}$$

Le rapport de la largeur de bande passante (fréquences de coupures définies à -3 decibels) sur la fréquence du maximum est :

$$m = \frac{\Delta f}{f_0} = \frac{5-K}{\sqrt{2}} \tag{3}$$

On en déduit la relation entre le gain maximal et m :

$$A = \frac{5-\sqrt{2}m}{\sqrt{2}m} \tag{4}$$

La valeur maximale de K est 5 ; au delà, le circuit est instable. Pour cette valeur, le gain est en principe infini.

2.b. Étude de la bande passante

L'étude expérimentale se fait en régime sinusoïdal. Pour différentes positions du potentiomètre, on relève la fréquence du maximum du gain avec son incertitude, le gain maximal et les deux fréquences de coupure. Les données sont entrées dans un tableau avec Libre Office. La largeur de bande passante et le coefficient m sont calculés, afin de tracer le gain maximal A en fonction de m. Lorsque la saisie est terminée, le fichier est enregistré sous forme CSV (séparateur de champ : tabulation).

Le tableau est récupéré dans python. Pour cela, il faut convertir les virgules des séparateurs décimaux en points :

```
import StringIO
import numpy

s = open('gainBandePassante.csv').read().replace(',','.')
data = numpy.loadtxt(StringIO.StringIO(s),skiprows=1,unpack=True)
fmax = data[0]
G = data[2]
fc1=data[3]
fc2=data[4]
```

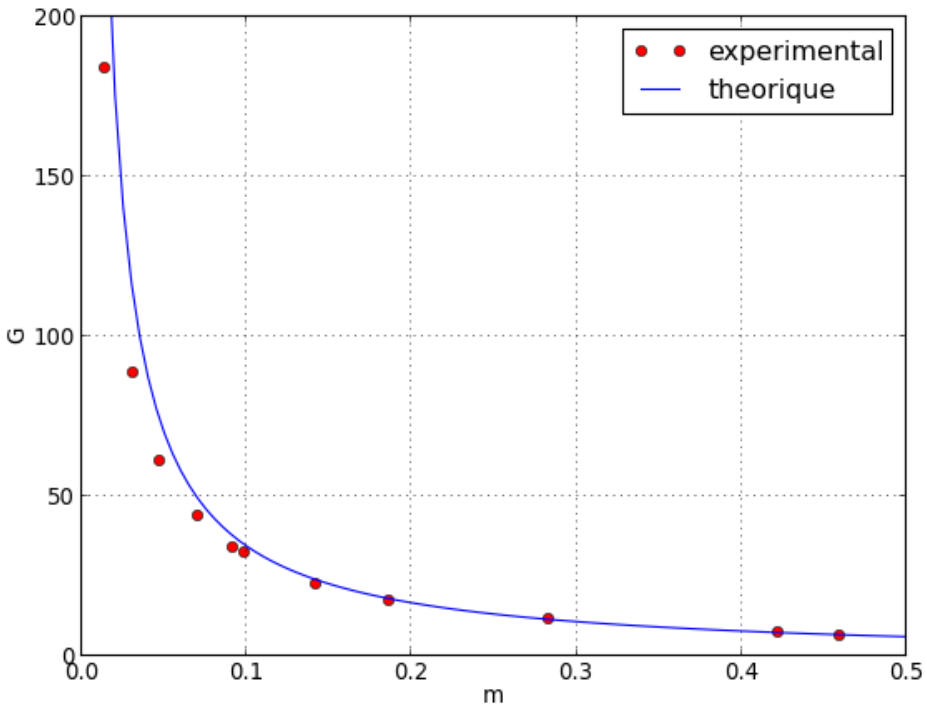
Remarquer l'utilisation de StringIO, qui permet de traiter une chaine de caractères comme un fichier. La première ligne du fichier est sautée car elle contient les légendes de colonnes. L'option unpack permet de récupérer le tableau sous forme transposée, avec les colonnes disposées en ligne.

On calcule la largeur de bande passante et le coefficient m :

```
Df = fc2-fc1
m=Df/fmax
```

On trace le gain maximal en fonction de m, points expérimentaux et courbe théorique :

```
from matplotlib.pyplot import *
import math

figure(figsize=(8,6))
plot(m,G,'ro',label="experimental")
xlabel("m")
ylabel("G")
grid()

def gain(m):
    return (5.0-math.sqrt(2)*m)/(math.sqrt(2)*m)
n = 100
m1 = numpy.arange(1,n+1)*0.5/n
G1 = numpy.zeros(n)
for k in range(n):
    G1[k] = gain(m1[k])
plot(m1,G1,'b',label="theorique")
legend(loc='upper right')
axis([0,0.5,0,200])
```



Le filtre peut être très sélectif, avec une bande passante très étroite, lorsque K est proche de 5 (attention toutefois à ne pas passer dans la zone instable). En contrepartie, le gain est alors très élevé, ce qui oblige à utiliser des signaux de très faible amplitude en entrée. Les générateurs de signaux possèdent une fonction d'atténuation -20 décibel pour cela.

La fréquence centrale (maximum du gain) est relevée à 1014 Hz à plus ou moins 2 Hz. Cette valeur peut varier d'un exemplaire à l'autre du filtre, en raison de la dispersion des valeurs des résitances et des capacités.

2.c. Filtrage d'un signal périodique

Le signal est généré sur la sortie audio de l'ordinateur, avec le programme Pure Data syntheseHarmonique.pdf, qui permet de créer un signal avec un fondamental et 3 harmoniques (on peut facilement ajouter des blocs si l'on veut plus d'harmoniques).

Le gain K est réglé avec le potentiomètre pour obtenir un gain maximal d'environ $A = 100$.

La fréquence fondamentale du signal est choisie à la moitié de la fréquence f_0 du filtre, soit 507 Hz, ce qui permet de placer l'harmonique de rang 2 au centre de la bande passante, là où le gain est maximal. On relève au préalable de gain en décibel pour les fréquences de 507 Hz et 2028 , relativement au gain maximal : on obtient -33 décibels. Cela signifie que les harmoniques de rang 1 et 3 sont atténuées de -33 décibels par rapport à celle de rang 2.

Pour effectuer l'analyse spectrale des signaux, on fait leur acquisition avec la centrale Sysam SP5, en utilisant le module python d'interface présenté dans CAN Eurosmart : interface pour Python. Le programme suivant effectue l'acquision, avec une fréquence d'échantillonnage de 20 kHz, très largement suffisante, et une durée totale de $T = 5$ s. Il trace le signal en entrée et en sortie du filtre et les spectres correspondants. Les données sont enregistrées dans un fichier.

echantillonnage.py

```
import pycan.main as pycan
from matplotlib.pyplot import *
import numpy
import math
import numpy.fft
import os
import scipy.signal

os.chdir("C:/Users/fred/Documents/electro/TP/passeBande")
nom = "signal-6"

can = pycan.Sysam("SP5")
can.config_entrees([0,1],[1.0,10.0]) # calibre de 1 V pour l'entree, 10 V pour la sor

fe=20000.0
T=5.0
te=1.0/fe
N = int(T/te)
print(N)
can.config_echantillon(te*10**6,N) # periode d'echantillonnage en microsecondes et no
```

```
can.acquerir()
temps = can.temps()
entrees = can.entrees()
t0=temps[0]
u0=entrees[0]
t1=temps[1]
u1=entrees[1]
numpy.savetxt('%s.txt'%nom,[t0,u0,t1,u1])
te = t0[1]-t0[0]
fe = 1.0/te
N = t0.size
T = t0[N-1]-t0[0]
can.fermer()

figure()
plot(t0,u0,'b')
plot(t1,u1,'r')
xlabel("t (s)")
ylabel("u (V)")
axis([0.1,0.12,-10,10])
grid()
savefig("%s-signal.pdf"%nom)

tfd0=numpy.fft.fft(u0*scipy.signal.get_window("hann",N))
a0 = numpy.absolute(tfd0)/N
tfd1 = numpy.fft.fft(u1*scipy.signal.get_window("hann",N))
a1 = numpy.absolute(tfd1)/N
f=numpy.arange(N)*1.0/T
figure()
plot(f,a0,'b')
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,a0.max()])
grid()
title("avant filtrage")
savefig("%s-spectre-A.pdf"%nom)
figure()
plot(f,a1,'r')
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,a1.max()])
grid()
title("apres filtrage")
savefig("%s-spectre-B.pdf"%nom)

show()
```

La fonction suivante lit un fichier de données et trace le signal en entrée et en sortie.

Le signal d'entrée est multipliée par 10 pour le tracé.

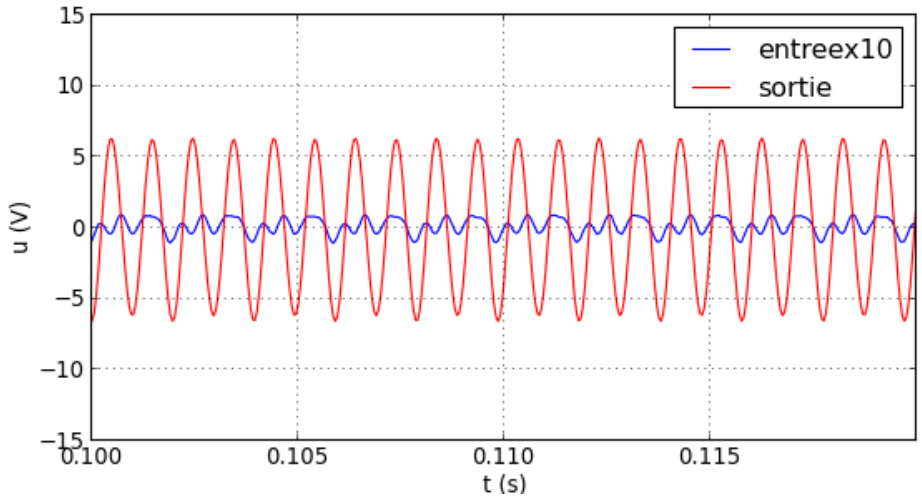
```
import numpy
import math
from matplotlib.pyplot import *
import numpy.fft
import scipy.signal

def traceSignaux(nom):
    [t0,u0,t1,u1] = numpy.loadtxt(nom)
    figure(figsize=(8,4))
    plot(t0,u0*10,'b',label='entreex10')
    plot(t1,u1,'r',label='sortie')
    xlabel('t (s)')
    ylabel('u (V)')
    axis([0.1,0.12,-15,15])
    legend(loc='upper right')
    grid()
```

La fonction suivante trace les spectres entre 0 et 2 kHz :

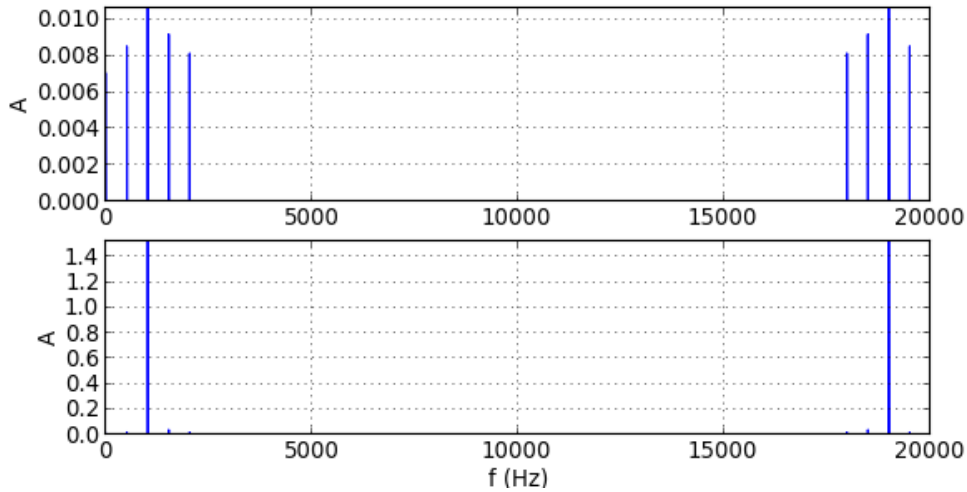
```
def traceSpectres(nom):
    [t0,u0,t1,u1] = numpy.loadtxt(nom)
    N = t0.size
    T = t0[N-1]-t0[0]
    te = t0[1]-t0[0]
    fe = 1.0/te
    a0 =numpy.absolute(numpy.fft.fft(u0*scipy.signal.get_window("hann",N)))/N
    a1 =numpy.absolute(numpy.fft.fft(u1*scipy.signal.get_window("hann",N)))/N
    f=numpy.arange(N)*1.0/T
    subplot(211)
    plot(f,a0,'b')
    ylabel('A')
    axis([0,fe,0,a0.max()])
    grid()
    subplot(212)
    plot(f,a1,'b')
    xlabel('f (Hz)')
    ylabel('A')
    axis([0,fe,0,a1.max()])
    grid()
```

Voici le résultat pour un signal comportant 4 harmoniques de même amplitudes ($0.039\ V$) :

```
traceSignaux("signal-2.txt")
```



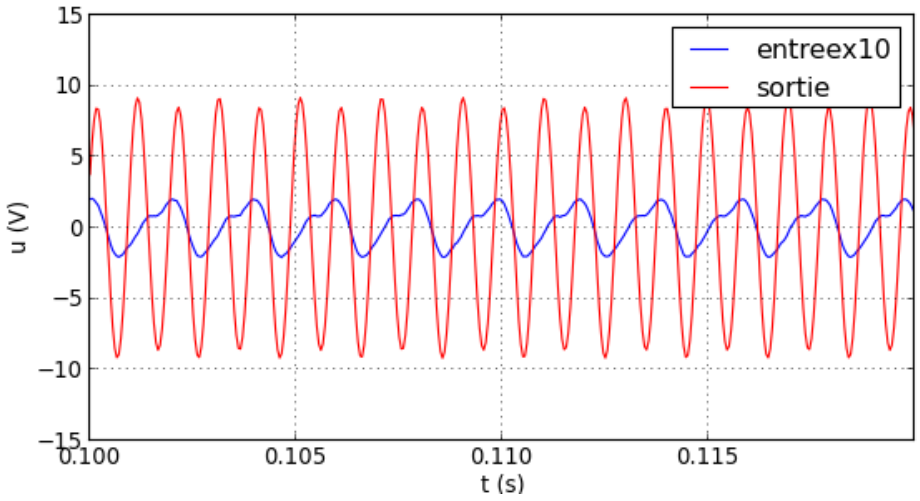

```
traceSpectres("signal-2.txt")
```



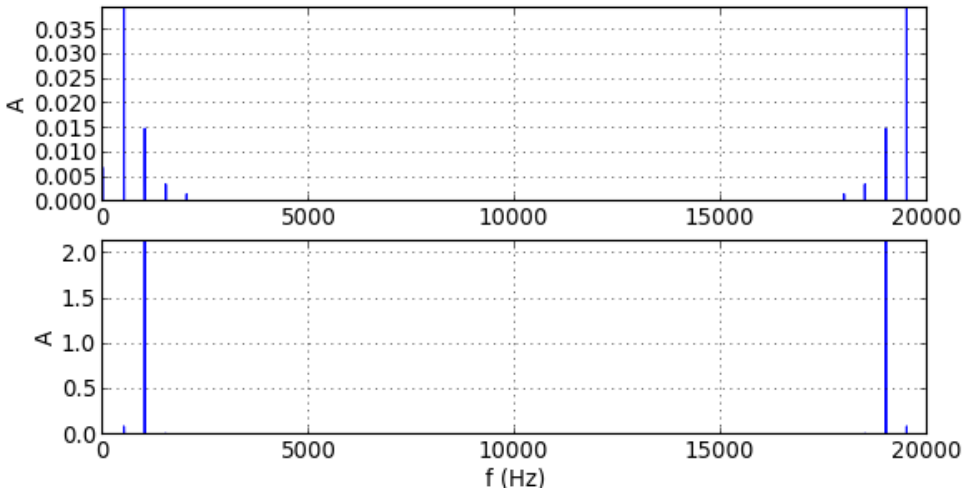
On remarque que les harmoniques de rang 1,3 et 4 ont pratiquement disparu en sortie, ce qui explique la forme sinusoïdale du signal de sortie, avec une fréquence de 1014 Hz. Ce filtre effectue donc la sélection de l'harmonique de rang 2.

Voyons les résultats avec des harmoniques d'amplitude décroissante :

```
traceSignaux("signal-4.txt")
```

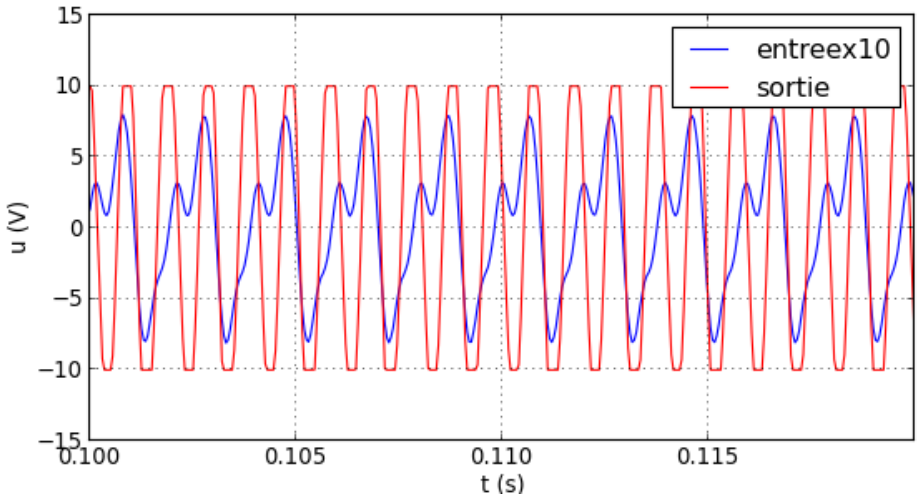



```
traceSpectres("signal-4.txt")
```

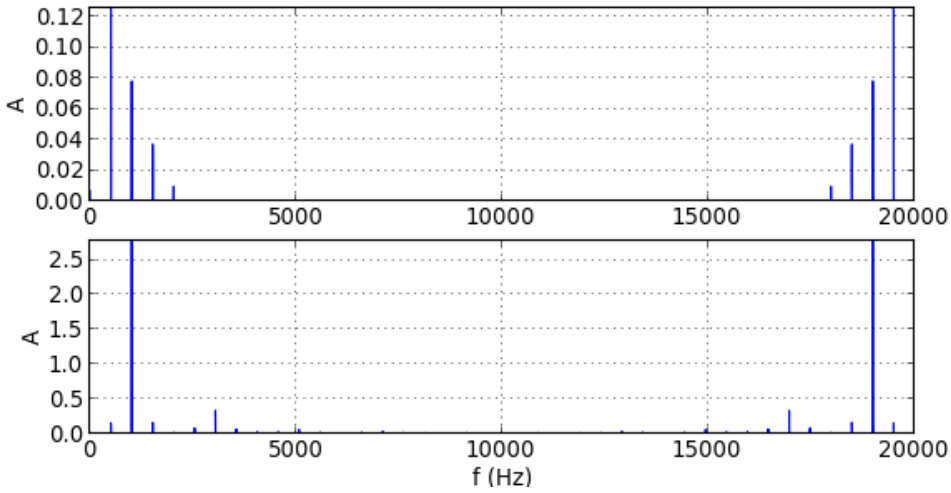


Le résultat suivant est obtenu avec un signal dont les harmoniques sont décroissantes, avec une saturation en sortie (la saturation est due au convertisseur analogique-numérique et non pas au filtre) :

```
traceSignaux("signal-5.txt")
```




```
traceSpectres("signal-5.txt")
```



La saturation est un effet non linaire, qui se traduit par l'apparition de nouvelles harmoniques dans le spectre. L'analyse spectrale est d'ailleurs un moyen très sensible de détection des effets non linéaires.

3. Filtre numérique

3.a. Filtre passe-bande RIF

Réaliser un filtre analogique très sélectif comme le précédent est relativement aisé, mais sa fréquence centrale est fixe. Par ailleurs, il est très difficile de réaliser un filtre à large bande passante dont la décroissance du gain dans la bande atténuée soit très rapide. Voir le document Filtres passe-bas analogiques et numérique pour cette notion de gain dans la bande atténuée.

Les filtres numériques présentent une grande souplesse de réglage et permettent de réaliser des décroissances très fortes dans la bande atténuée, associées à des réponses très plates dans la bande passante.

Nous allons voir comment calculer un filtre numérique RIF (réponse impulsionnelle finie). Voir le document Introduction aux filtres numériques pour une présentation générale du filtrage numérique, et Exemples de filtres RIF pour le calcul des filtres RIF.

Le filtre réalisé doit sélectionner une bande autour de la fréquence 1000 Hz, avec des fréquences de coupure de 900 Hz et 1100 Hz. Pour définir un filtre numérique, il faut considérer le rapport des fréquences de coupure sur la fréquence d'échantillonnage :

$$a = \frac{f_{c1}}{f_e} = \frac{900}{2000} \tag{5}$$

$$b = \frac{f_{c2}}{f_e} = \frac{1100}{2000} \tag{6}$$

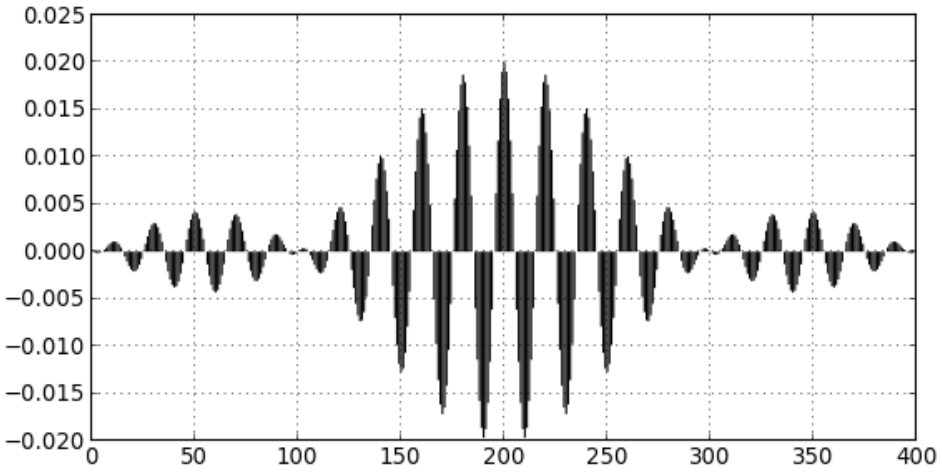
Ces rapports ne doivent pas être trop petits. Ici, la valeur d'environ 1/20 est raisonnable. Pour une valeur de l'ordre de 1/100, il serait préférable d'abaisser la fréquence d'échantillonnage.

Le filtre passe-bande idéal doit avoir un gain $G(x)$ de 1 dans la bande passante, nul ailleurs, où x désigne le rapport de la fréquence sur la fréquence d'échantillonnage. Pour calculer la réponse impulsionnelle d'un tel filtre, on prolonge la fonction $G(x)$ par périodicité (de période 1) et on calcule ses coefficients de Fourier. La réponse impulsionnelle ainsi obtenue est ([1]) :

$$\begin{aligned} g_0 &= 2(b-a) \\ g_k &= \frac{\sin(k2b)-\sin(2ka)}{k\pi} \end{aligned} \tag{7}$$

Pour obtenir une réponse impulsionnelle finie, il faut la tronquer à un rang P, c'est-à-dire garder seulement les indices $-P \leq k \leq P$. L'indice de troncature doit être d'autant plus élevé que l'on souhaite une forte sélectivité et que les coefficients a et b sont faibles. Voici le calcul de la réponse impulsionnelle pour $P = 200$.

```
import math
P=200
h = numpy.zeros(2*P+1)
a=900.0/20000.0
b=1100.0/20000.0
def g(k):
    if k==0:
        return 2*(b-a)
    else:
        return (numpy.sin(2*math.pi*k*b)-numpy.sin(2*math.pi*k*a))/(k*math.pi)
for i in range(2*P+1):
    h[i] = g(i-P)
indices = numpy.arange(2*P+1)
figure(figsize=(8,4))
vlines(indices,[0],h)
grid()
```



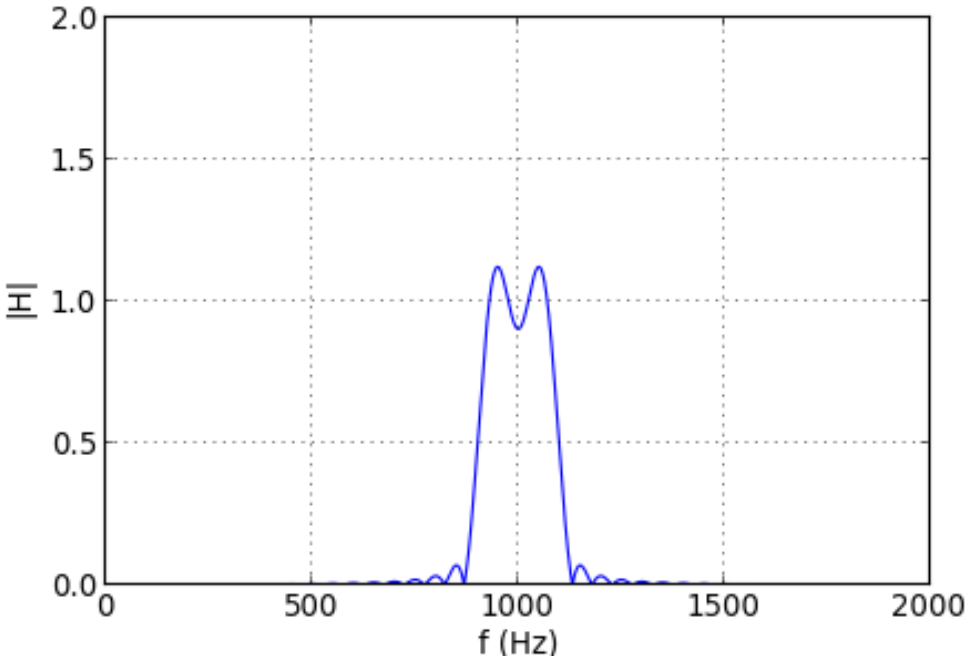
Avant d'appliquer ce filtre, il faut étudier sa réponse fréquentielle, c'est-à-dire son gain et son déphasage en fonction de la fréquence. La méthode pour le faire est expliquée dans Introduction aux filtres numériques. Voici la fonction qui permet de le faire :

```
def reponseFreq(h):
    N = h.size
    def Hf(f):
        s = 0.0
        for k in range(N):
            s += h[k]*numpy.exp(-1j*2*math.pi*k*f)
        return s
    f = numpy.arange(start=0.0,stop=0.5,step=0.0001)
    hf = Hf(f)
    g = numpy.absolute(hf)
    phi = numpy.unwrap(numpy.angle(hf))
    return [f,g,phi]
```

Remarque : la fonction `scipy.signal.freqz` fournit la réponse fréquentielle d'un filtre donné sous forme de fonction de transfert en Z.

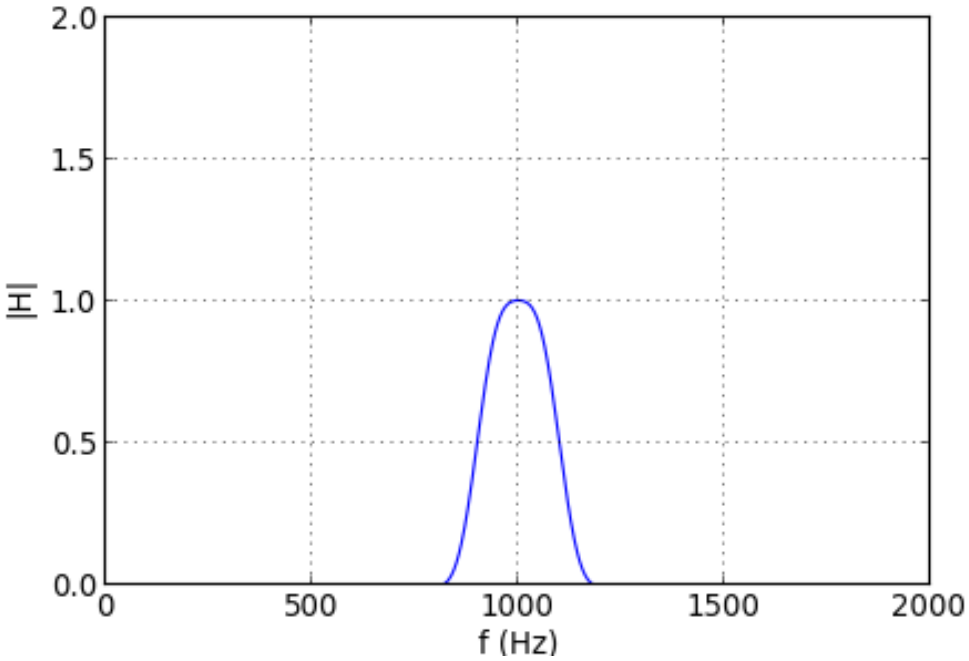
Voici donc la réponse fréquentielle du filtre :

```
(f,g,phi)=reponseFreq(h)
fe = 20000
figure(figsize=(6,4))
plot(f*fe,g)
xlabel('f (Hz)')
ylabel('|H|')
axis([0,2000,0,2])
grid()
```



On est bien sûr assez loin du filtre idéal, parce-que la réponse impulsionnelle a été tronquée au rang P. Le plus gênant est la présence d'ondulations dans la bande passante. Pour les réduire, on utilise un fenêtrage progressif de la réponse impulsionnelle, par exemple un fenêtrage de Hamming :

```
import scipy.signal
h = h*scipy.signal.get_window("hamming",2*P+1)
(f,g,phi)=reponseFreq(h)
figure(figsize=(6,4))
plot(f*fe,g)
xlabel('f (Hz)')
ylabel('|H|')
axis([0,2000,0,2])
grid()
```



Il n'y a plus d'ondulations. On remarque que le gain à la coupure est 1/2. Si l'on veut retrouver la définition de la coupure utilisée en filtrage analogique, il faut modifier les paramètres a et b.

3.b. Application

Nous allons appliquer le filtre précédent au signal périodique de l'expérience précédente, obtenu par acquisition avec la centrale Sysam SP5. On commence par récupérer les échantillons :

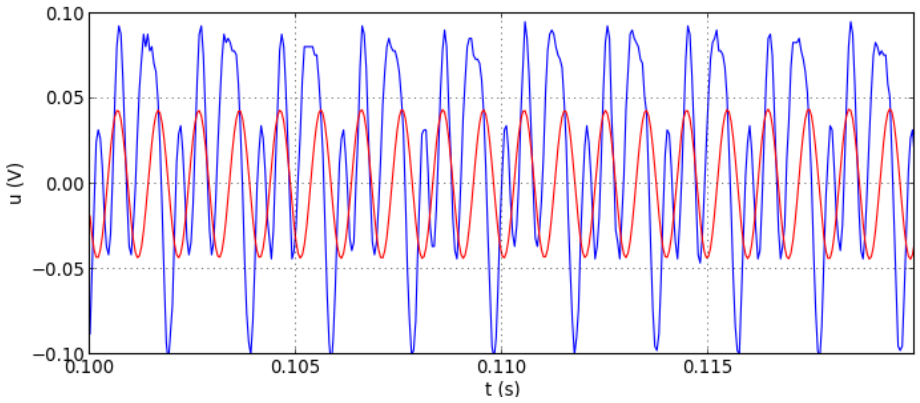
```
[t0,u0,t1,u1] = numpy.loadtxt("signal-2.txt")
N = t0.size
T = t0[N-1]-t0[0]
te = t0[1]-t0[0]
fe = 1.0/te


print(fe)
--> 20000.0
```

Pour filtrer les échantillons (ici u_0), il faut effectuer le produit de convolution avec la réponse impulsionnelle finie. En pratique, les filtres numériques effectuent ce calcul en temps réel, au fur et à mesure que les données arrivent en provenance du convertisseur analogique-numérique. Avec python, on peut programmer explicitement le produit de convolution, ou plus simplement utiliser la fonction `scipy.signal.convolve` :

```
y = scipy.signal.convolve(u0,h,mode='valid')
```

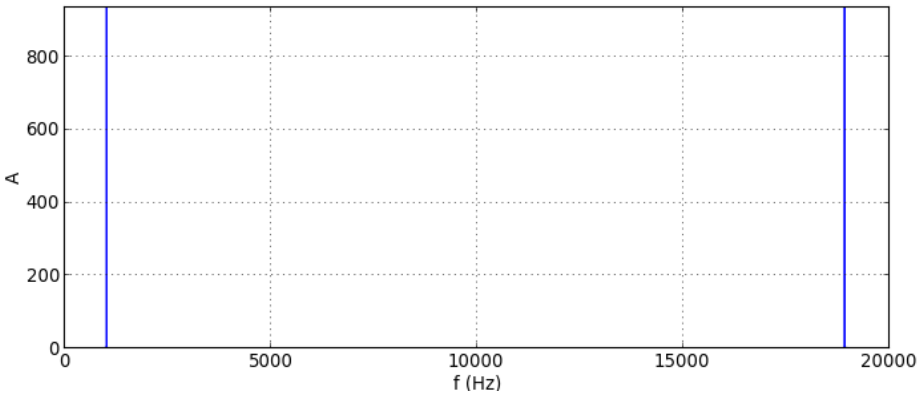
L'option `mode='valid'` permet de calculer la convolution seulement pour les points valides, c'est-à-dire à partir du point d'indice $2P$. Le résultat de la convolution comporte donc $2P+1$ points de moins que le tableau initial. En effet, il faut au moins $2P+1$ points disponibles en entrée pour calculer la sortie. Il faut donc redéfinir l'échellle de temps pour tracer le signal. Il y a plusieurs manières de le faire. Nous allons simuler l'effet d'un filtre numérique temps réel, pour lequel le premier point calculé correspond à l'instant $t = (2P+1)T_e$:

```
ny = y.size
t = numpy.zeros(ny)
for k in range(ny):
    t[k] = (2*P+1)*te+te*k
figure(figsize=(10,4))
plot(t0,u0,'b')
plot(t,y,'r')
xlabel("t (s)")
ylabel("u (V)")
axis([0.1,0.12,-0.1,0.1])
grid()
```



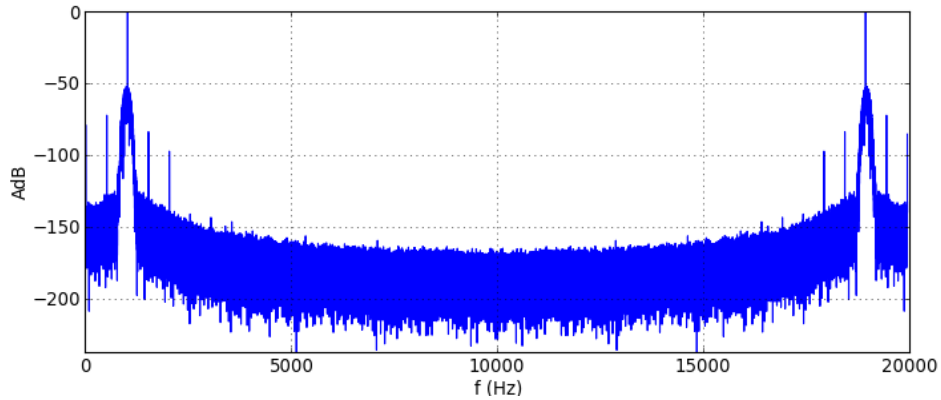
L'effet du filtrage est une sélection de l'harmonique de rang 2, comme pour le filtre analogique étudié plus haut. Voyons le spectre de la sortie :

```
a = numpy.absolute(numpy.fft.fft(y*scipy.signal.get_window("hann",ny)))
f = numpy.arange(ny)*1.0/T
figure(figsize=(10,4))
plot(f,a)
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,a.max()])
grid()
```



On voit que les harmoniques de rang 1, 3 et 4 ont complètement disparu. Avec le filtre analogique précédent, elles étaient encore visibles sur ce spectre. Pour voir cela plus en détail, on trace l'amplitude en échelle logarithmique (décibels) :

```
adb = 20*numpy.log10(a)
adb = adb-adb.max()
figure(figsize=(10,4))
plot(f,adb)
xlabel("f (Hz)")
ylabel("AdB")
axis([0,fe,adb.min(),0])
grid()
```



Les harmoniques de rang 1, 3 et 5 sont bien présentes, mais à un niveau ne dépassant pas −70 décibels par rapport à l'harmonique de rang 2. Elles sont donc extrêmement faibles. Un tel résultat est très difficile à obtenir avec un filtre analogique, dont la décroissance du gain dans la bande atténuée n'est pas assez rapide.

Référence

[1] E. Tisserand, J.F. Pautex, P. Schweitzer, *Analyse et traitement des signaux*, (Dunod, 2008)