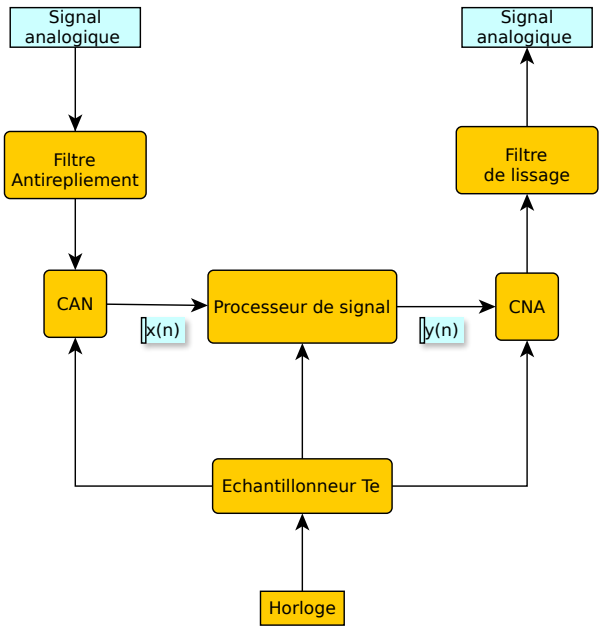
Filtrage numérique

1. Introduction

Nous avons vu dans les TP précédents comment un convertisseur analogique/numérique permet d'obtenir un signal numérique, c'est-à-dire une suite de nombres correspondant aux échantillons du signal prélevés à la fréquence d'échantillonnage.

Le filtrage numérique consiste à modifier le signal numérique de manière à obtenir différents effets, comme le filtrage passe-bas, passe-bande, la dérivation, etc. Il existe des microprocesseurs spécialisés dans le filtrage numérique (DSP : Digital Signal Processor), capables de traiter des signaux numériques en *temps réel* avec une cadence très élevée (plusieurs millions d'échantillons par seconde).

Le schéma suivant montre le principe d'une chaîne complète de traitement numérique, comportant un convertisseur analogique/numérique (CAN), un processeur de signal numérique, et un convertisseur numérique/analogique (CNA).



Les filtres numériques sont utilisés dans les appareils électroniques (téléphones, téléviseurs, audio, etc.). Il permettent d'obtenir des filtrages très efficaces beaucoup plus facilement que les filtres analogiques, à un coût bien plus faible. Par ailleurs, un filtre numérique est défini par une suite de coefficients, qu'il est très facile de modifier pour changer le filtrage (par exemple la fréquence de coupure). Une telle souplesse est impossible à obtenir avec un filtre analogique.

Dans de nombreux cas, il n'y a pas de conversion numérique/analogique à l'issu du traitement, mais stockage ou transmission numérique (par exemple en prise de son ou d'image numérique). Le filtrage numérique peut alors intervenir de manière différée, au moment de la lecture ou de la réception du signal.

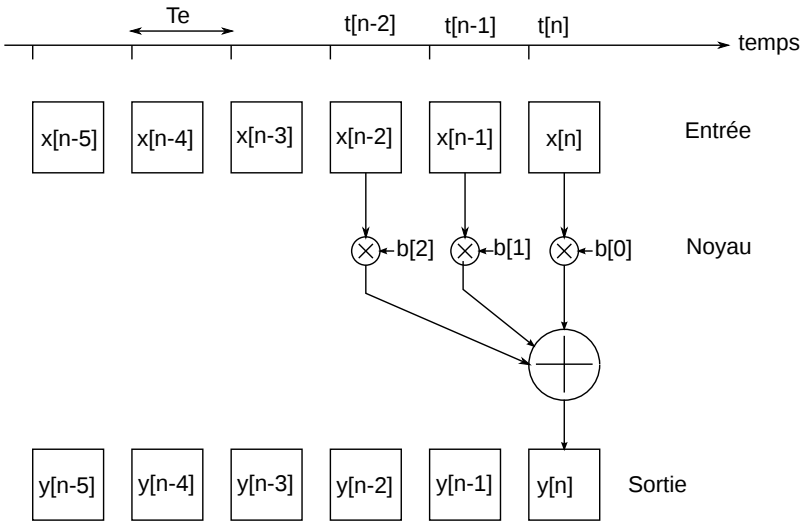
L'objectif de ce TP est de découvrir le principe d'un filtrage numérique par convolution. Nous allons faire la conversion analogique/numérique d'un signal sinusoïdal pour lui appliquer un filtre numérique passe-bas. On pourra ainsi étudier expérimentalement la réponse fréquentielle de ce type de filtre, avant de l'appliquer à des signaux réels. Le filtrage sera fait sur

l'ordinateur après l'acquisition et non pas en temps réel comme dans un processeur de signal. Néanmoins, nous simulerons le fonctionnement d'un filtre temps réel.

2. Filtrage par convolution

2.a. Principe

La figure suivante montre le fonctionnement d'un filtrage numérique par convolution.



Seuls les 6 derniers échantillons de l'entrée x_n sont représentés. La convolution se fait avec un noyau, qui dans cet exemple comporte trois coefficients $b[0], b[1], b[2]$. Le signal filtré est noté y_n. Pour obtenir le dernier échantillon du signal filtré, on effectue une combinaison linéaire des trois derniers échantillons du signal d'entrée :

$$y_n = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2}$$

Plus généralement, si N est le nombre d'élements du noyau, l'échantillon n de la sortie se calcule en faisant une combinaison linéaire des N derniers échantillons de l'entrée :

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

Cette relation définit la *convolution discrète*, et permet donc de réaliser un filtrage par convolution. L'effet du filtre dépend évidemment des coefficients choisis pour le noyau.

Dans le schéma ci-dessus, y_n et x_n sont considérés comme simultanés. Dans un filtre fonctionnant en temps réel, y_n est calculé dès que x_n est disponible et le temps de calcul doit être inférieur à la période d'échantillonnage. Si le temps de calcul est faible par rapport à T_e, y_n et x_n sont pratiquement simultanés.

Le filtre de convolution le plus simple est le filtre moyenneur, qui réalise la moyenne arithmétique des deux derniers échantillons :

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

Le filtre suivant définit y_n comme la moyenne arithmétique des N derniers échantillons :

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} x_{n-k}$$

2.b. Réponse impulsionnelle

Une impulsion est un signal numérique défini par :

$$\begin{aligned} x_0 &= 1 \\ x_n &= 0 \ (n \neq 0) \end{aligned}$$

La sortie est alors :

$$\begin{aligned} y_n &= 0 \ (n < 0) \\ y_0 &= b_0 x_0 = b_0 \\ y_1 &= b_1 x_0 = b_1 \\ &\dots \\ y_{N-1} &= b_{N-1} x_0 = b_{N-1} \\ y_n &= 0 \ (n \geq M) \end{aligned}$$

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

Un filtre de convolution est dit à *réponse impulsionnelle finie* (RIF) car le nombre de termes de la réponse impulsionnelle est fini.

Les filtres numériques RIF sont toujours stables : si l'entrée reste bornée, la sortie l'est aussi.

2.c. Réponse fréquentielle

Pour obtenir la réponse fréquentielle théorique du filtre, on considère un signal d'entrée sinusoïdal, de fréquence f et d'amplitude unité. L'échantillonnage à la période T_e donne le signal numérique suivant :

$$x_n = \exp(i 2\pi \, f \, n T_e)$$

Le signal en sortie s'écrit :

$$\begin{aligned} y_n &= \sum_{k=0}^{N-1} b_k x_{n-k} \\ &= \exp(i 2\pi \, n \, f \, T_e) \sum_{k=0}^{N-1} b_k \exp(-i 2\pi \, f \, k \, T_e) \\ &= x_n H(Z) \end{aligned}$$

avec :

$$H(Z) = \sum_{k=0}^{N-1} b_k Z^{-k} \tag{2}$$

$$Z = \exp(i2\pi\, f\, T_e) \tag{3}$$

On voit donc que la sortie (écrite sous forme complexe) est 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 :

$$H_f(f) = \sum_{k=0}^{N-1} b_k \exp(-i2\pi\, k\, f\, T_e) \tag{4}$$

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 représentation graphique de la réponse fréquentielle consiste à tracer le module de H_f et son argument en fonction de la fréquence sur l'intervalle $[0, f_e/2]$.

2.d. Filtrage d'un signal en mémoire

Nous n'allons pas effectuer un filtrage en temps réel, mais le filtrage d'un signal numérique stocké en mémoire après la conversion analogique/numérique. On remarque que le filtrage par convolution nécessite au moins N termes en entrée pour s'appliquer.

La fonction suivante effectue le fitrage par convolution :

```
def filtrage_convolution(x,b):
    N = len(b)
    ne = len(x)
    y = numpy.zeros(ne)
    for n in range(N-1,ne):
        accum = 0.0
        for k in range(N):
            accum += b[k]*x[n-k]
        y[n] = accum
    return y
```

Les $N-1$ premiers échantillons de la sortie sont nuls car le filtrage ne commence qu'à partir de l'échantillon d'indice $N-1$. Afin de simuler le fonctionnement d'un filtre temps réel, l'échantillon y_n de la sortie est considéré comme simultané avec l'échantillon x_n de l'entrée.

2.e. Filtre moyenneur

Considérons le filtre suivant, qui définit y_n comme la moyenne des 10 derniers échantillons de l'entrée :

$$y_n = \frac{1}{10} \sum_{k=0}^{9} x_{n-k}$$

La suite de ces coefficients (qui est aussi sa réponse impulsionnelle) est placée dans un tableau :

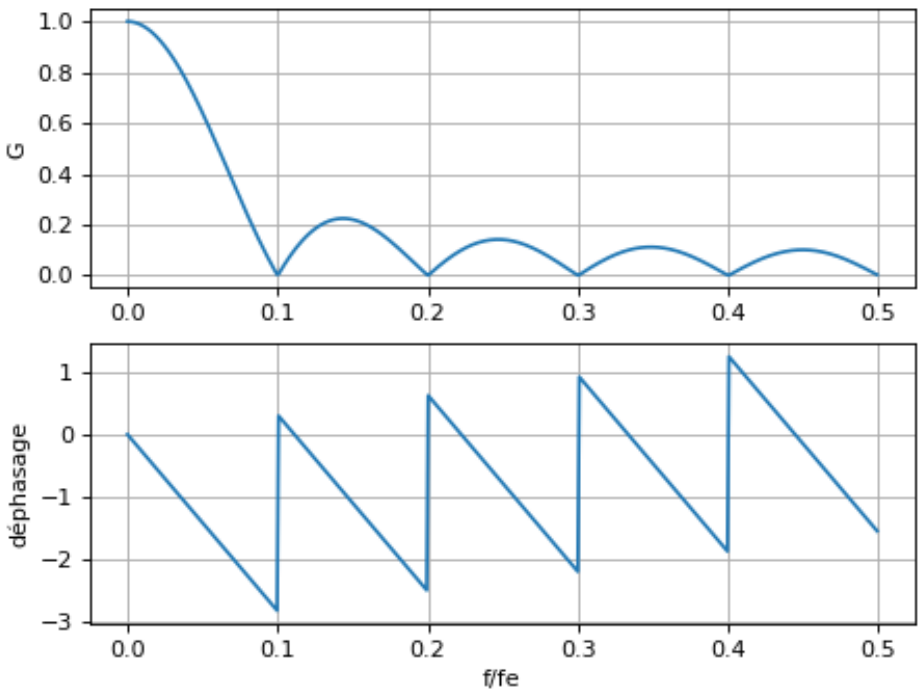
```
import numpy
b = numpy.ones(10)*1/10
```

La réponse fréquentielle peut être obtenue avec la fonction `scipy.signal.freqz` :

```
import scipy.signal
w,H=scipy.signal.freqz(b)
```

Le tableau `w` contient les pulsations (500 points par défaut) et le tableau `H` contient les valeurs complexes de la réponse fréquentielle. Voici comment tracer le gain et le déphasage en fonction de la fréquence :

```
from matplotlib.pyplot import *
figure()
subplot(211)
plot(w/(2*numpy.pi),numpy.absolute(H))
ylabel("G")
grid()
subplot(212)
plot(w/(2*numpy.pi),numpy.unwrap(numpy.angle(H)))
xlabel("f/fe")
ylabel("déphasage")
grid()
```



Le filtre moyenneur est donc un filtre passe-bas. Pour les filtres numériques, il est d'usage de définir la fréquence de coupure comme la fréquence pour laquelle le gain vaut 1/2. La fréquence de coupure de ce filtre est donc $0,06 f_e$. Le fait d'avoir multiplié la moyenne par 1/10 permet d'obtenir un gain de 1 à très basse fréquence. Le déphasage dans la bande passante varie parfaitement linéairement avec la fréquence, ce qui est la propriété recherchée pour le filtrage des signaux périodiques, car elle implique que le retard est constant.

Le filtre moyenneur est rarement utilisé car les caractéristiques de sa bande atténuante sont mauvaises : le gain comporte des rebonds dans la bande atténuante, ce qui fait que la décroissance du gain est très lente. Par exemple, à une fréquence $0{,}25 f_e$, soit 4 fois la fréquence de coupure, le gain est de 0,14.

Nous allons voir qu'il est possible de définir un filtre passe-bas avec une décroissance du gain dans la bande atténuante beaucoup plus rapide.

2.f. Filtre passe-bas

Pour définir un filtre passe-bas, il faut tout d'abord calculer le rapport de la fréquence de coupure sur la fréquence d'échantillonnage (qui doit être inférieur à $1/2$) :

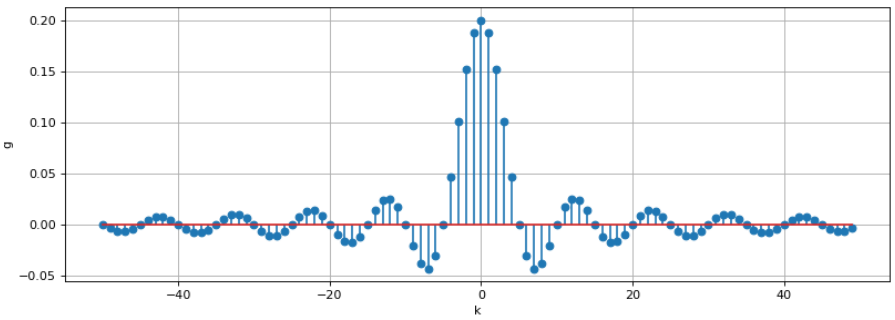
$$a = \frac{f_c}{f_e} \tag{5}$$

On appelle filtre idéal un filtre qui laisse passer sans atténuation toutes les sinusoïdes de fréquences inférieures à la fréquence de coupure, et qui annule toutes les sinusoïdes de fréquences supérieures à la coupure. Son déphasage doit varier linéairement avec la fréquence dans la bande passante. Un filtre passe-bas idéal est irréalisable car sa réponse impulsionnelle est infinie. Elle est donnée par :

$$g(k) = 2a \frac{\sin(2\pi\, ka)}{2\pi\, ka} = 2a \,\mathrm{sinc}(2\, ka) \tag{6}$$

La fonction sinc est la fonction *sinus cardinal* définie par $\mathrm{sinc}(u) = \sin(\pi u)/(\pi u)$. Voici par exemple la réponse impulsionnelle pour $a = 0{,}1$ (en fait une partie de cette réponse) :

```
a=0.1
k=numpy.arange(-50,50)
g=2*a*numpy.sinc(2*k*a)
figure(figsize=(12,4))
stem(k,g)
xlabel("k")
ylabel("g")
grid()
```



Cette réponse impulsionnelle est infinie et on ne peut donc pas construire un filtre de convolution idéal. La réponse impulsionnelle est rendue finie par troncature. Soit P l'indice de troncature. La réponse impulsionnelle finie comporte alors $N = 2P + 1$ termes, correspondant aux indices k allant de $-P$ à P. Par ailleurs, la réponse impulsionnelle permettant d'obtenir le noyau du filtre doit être définie pour des indices positifs ou nuls, ce qui s'obtient en décalant les indices de $+P$. Finalement, la réponse impulsionnelle est définie par :

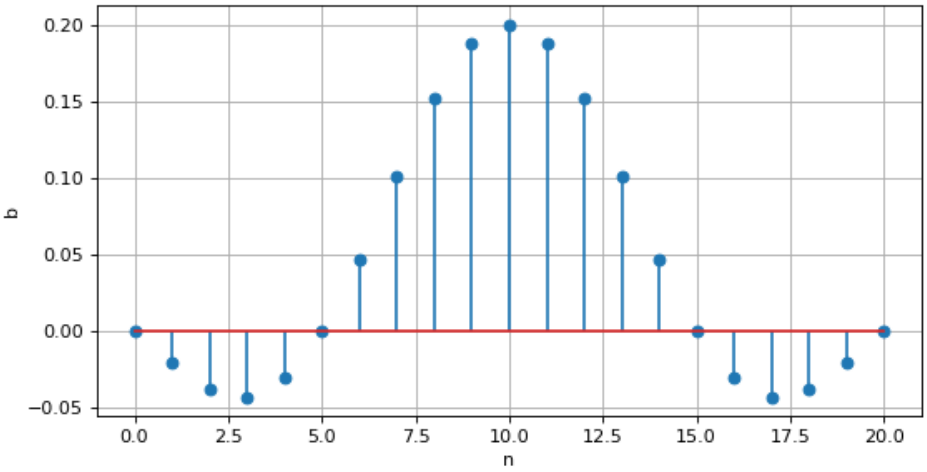
$$b_k = g(k - P) \tag{7}$$

l'indice k variant de 0 à $2P$.

L'indice de troncature devra être d'autant plus grand que a est faible. Il faut donc éviter une fréquence d'échantillonnage trop grande par rapport à la fréquence de coupure (le sur-échantillonnage n'est pas toujours souhaitable en filtrage numérique). Un critère simple est $Pa > 1$. Pour un coefficient a donné, la sélectivité du filtre augmente avec P. À la limite $P \to \infty$, on obtient un filtre idéal.

Voici un exemple de filtre passe-bas et sa réponse impulsionnelle :

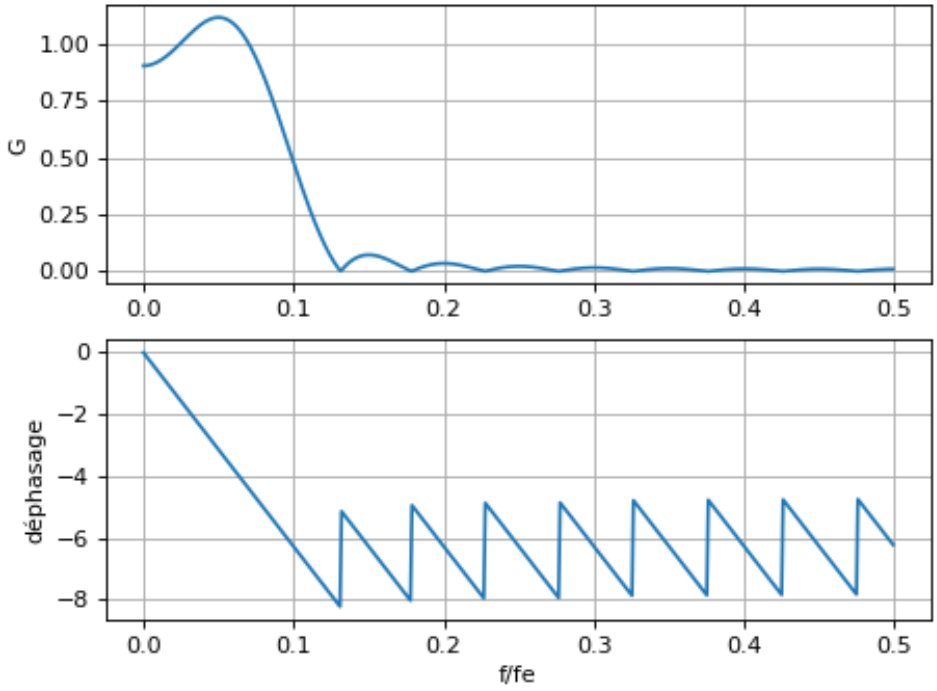
```
a=0.1
P = 10
b = numpy.zeros(2*P+1)
n = range(2*P+1)


for k in range(2*P+1):
    b[k] = 2*a*numpy.sinc(2*(k-P)*a)

figure(figsize=(8,4))
stem(n,b)
xlabel("n")
ylabel("b")
grid()
```



On voit que le maximum de la réponse impulsionnelle (le maximum du sinus cardinal) est décalé de P échantillons par rapport à l'échantillon de sortie calculé. Cela signifie que le signal de sortie sera retardé de P échantillons.

Voici la réponse fréquentielle de ce filtre :

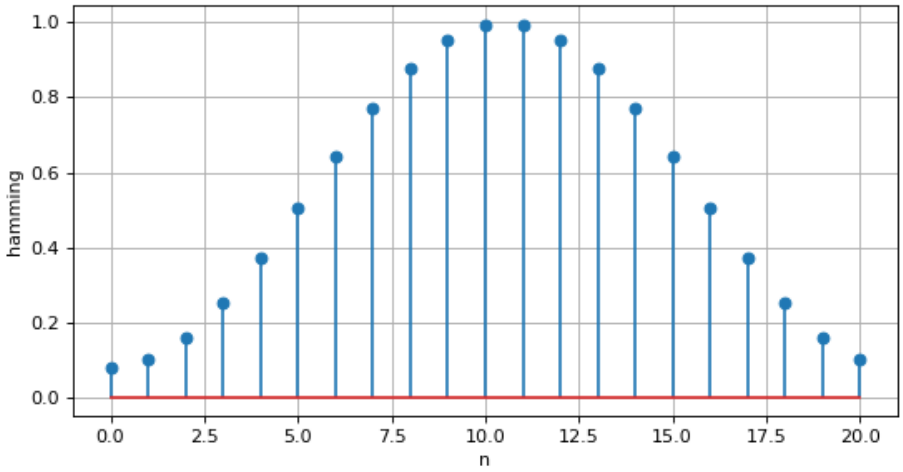
```
w,H=scipy.signal.freqz(b)

figure()
subplot(211)
plot(w/(2*numpy.pi),numpy.absolute(H))
ylabel("G")
grid()
subplot(212)
plot(w/(2*numpy.pi),numpy.unwrap(numpy.angle(H)))
xlabel("f/fe")
ylabel("déphasage")
grid()
```



Le gain à la fréquence de coupure est égal à $1/2$, et non pas à $1/\sqrt{2}$ comme il est d'usage pour un filtre analogique.

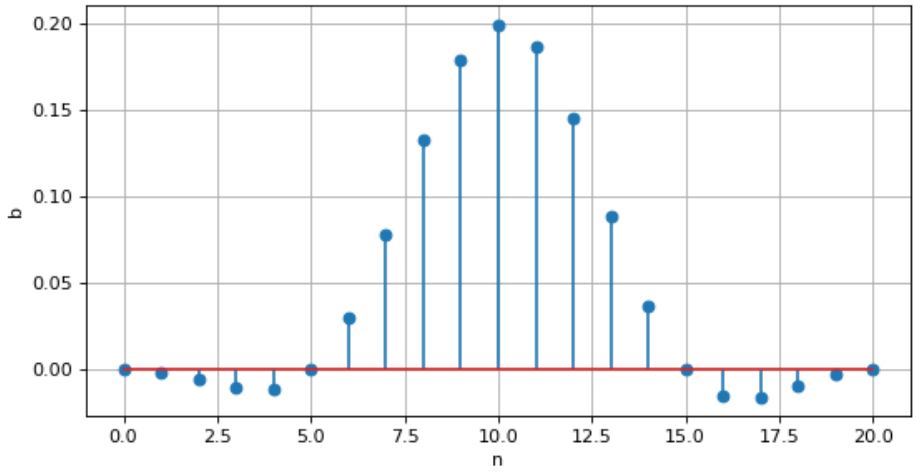
Pour tracer le déphasage, nous avons utilisé la fonction `numpy.unwrap`, qui permet d'éviter que l'angle soit ramené dans l'intervalle $[-\pi, \pi]$. En effet, le retard de la sortie par rapport à l'entrée peut dépasser la moitié de la période. On remarque que le déphasage dans la bande passante est parfaitement linéaire par rapport à la fréquence. En revanche, le gain dans la bande passante présente un maximum alors qu'il est préférable d'avoir un gain décroissant. Pour éliminer cet effet, on doit multiplier la réponse impulsionnelle par une fenêtre, c'est-à-dire une fonction qui réduit progressivement les valeurs de la réponse lorsqu'on s'approche du bord de l'intervalle. Voici par exemple la fenêtre de Hamming :

```
import scipy.signal
fen = scipy.signal.get_window("hamming",b.size)
figure(figsize=(8,4))
stem(n,fen)
xlabel("n")
ylabel("hamming")
grid()
```



Voici la nouvelle réponse impulsionnelle :

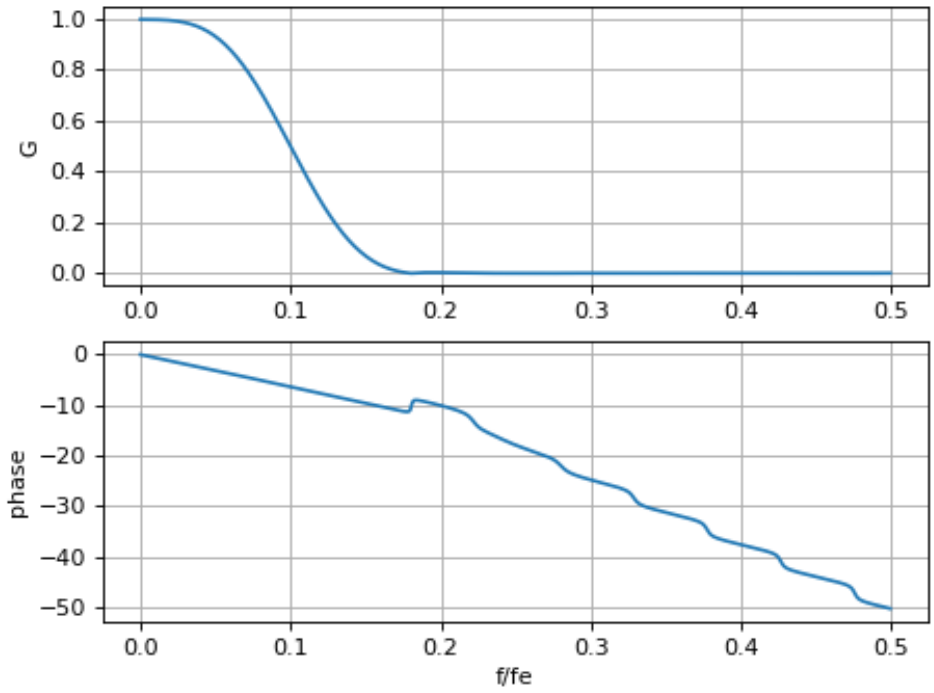
```
b = b*fen
figure(figsize=(8,4))
stem(n,b)
xlabel("n")
ylabel("b")
grid()
```



et la réponse fréquentielle :

```
w,H=scipy.signal.freqz(b)

figure()
subplot(211)
plot(w/(2*numpy.pi),numpy.absolute(H))
ylabel("G")
grid()
subplot(212)
```

```
plot(w/(2*numpy.pi),numpy.unwrap(numpy.angle(H)))
xlabel("f/fe")
ylabel("phase")
grid()
```



Avec la fenêtre de Hamming, le gain est bien décroissant.

3. Réalisation du filtrage

Le générateur de signaux est utilisé pour générer une tension sinusoïdale, dont on fait l'acquisition avec la carte SysamSP5. La fréquence d'échantillonnage est fixée à $f_e = 10\,\text{kHz}$.

Le script python filtreNumerique.py effectue l'acquisition sur la voie EA0, le calcul de la réponse impulsionnelle, le filtrage par convolution, et trace le signal d'entrée et de sortie sur la même figure. Il calcule aussi le gain du filtre en faisant le rapport des valeurs efficaces.

La fréquence de coupure du filtre passe-bas est $f_c = 1000\,\text{Hz}$. On commence par $P = 10$.

[1] Quelle est la fréquence maximale de la sinusoïde qui permet de respecter la condition de Nyquist-Shannon ?

[2] Ajuster l'amplitude à $1{,}0\,\text{V}$ et la fréquence à $100\,\text{Hz}$. Le gain est calculé numériquement par le script (rapport des valeurs efficaces). Pour obtenir le retard, on fera un relevé direct sur les courbes. Lorsque la fréquence augmente, il sera nécessaire de faire plusieurs relevés du retard et de calculer la moyenne.

[3] Relever le gain et le retard pour les fréquences $100\,\text{Hz}$, $200\,\text{Hz}$, $300\,\text{Hz}$, etc., jusqu'à la fréquence maximale où la mesure du retard est encore possible. Reporter ces valeurs dans un tableau.

[4] Que peut-on dire du retard de la sortie par rapport à l'entrée ? Exprimer ce retard en fonction de P et T_e.

[5] Tracer le gain en fonction de la fréquence et le déphasage en fonction de la fréquence.

Pour augmenter la sélectivité du filtre, il faut augmenter la taille de la réponse impulsionnelle, c'est-à-dire augmenter P.

[6] Faire des mesures de gain et de déphasage pour $P = 20$ et tracer les courbes.

[7] Comparer la sélectivité à celle du filtre pour $P = 10$.

[8] Tester le filtre sur un signal triangulaire. Quel problème ce signal pose-t-il pour le filtrage numérique ?

[9] Réaliser un filtre numérique permettant de transformer un signal triangulaire de fréquence $100\,\mathrm{Hz}$ en signal quasi sinusoïdal.

[10] Au moyen du fichier PureData syntheseHarmonique.pd, générer un signal périodique comportant 3 harmoniques, tous dans la bande passante du filtre. Ajouter du bruit. Vérifier que le filtre permet de réduire le bruit en altérant très peu la forme du signal.