Échantillonnage et reconstruction d'un signal périodique

1. Introduction

L'échantillonnage d'un signal continu est l'opération qui consiste à prélever des échantillons du signal pour obtenir un signal discret, c'est-à-dire une suite de nombres représentant le signal, dans le but de mémoriser, transmettre, ou traiter le signal.

L'échantillonnage intervient dans l'opération de conversion analogique-numérique, par exemple dans un dispositif de numérisation du son ou de l'image. Un autre exemple d'échantillonnage est celui que l'on fait pour obtenir la représentation graphique d'une fonction à une ou deux variables. D'une manière générale, l'échantillonnage intervient dans toute opération de conversion continu/discret.

Ce document présente le théorème de l'échantillonnage de Shannon, qui permet de savoir à quelle fréquence minimale il faut échantillonner un signal pour ne pas perdre l'information qu'il contient. On verra aussi comment se fait la reconstruction d'un signal continu à partir des échantillons, opération qui intervient dans la conversion numérique-analogique.

Pour expliquer l'échantillonnage et la reconstruction, il faut utiliser l'analyse spectrale et la transformée de Fourier discrète, abordées dans le document Introduction à l'analyse spectrale.

On s'intéressera à un signal temporel représenté par une fonction $u(t)$, où t est le temps, mais les résultats se transposent sans difficulté aux cas de fonctions d'autres variables, par exemple de variables d'espace. En particulier, les résultats seront utilisables pour l'échantillonnage d'une image, c'est-à-dire une fonction $I(x, y)$ de deux variables d'espace.

Pour simplifier, on se limitera au cas des signaux périodiques.

2. Échantillonnage

2.a. Théorème de Shannon

Soit $u(t)$ une fonction représentant un signal continu. On considère un échantillonnage périodique défini par :

$$t_k = kT_e \tag{1}$$
$$u_k = u(t_k) \tag{2}$$

où k est un entier. T_e est la période d'échantillonnage. $f_e = 1/T_e$ est la fréquence d'échantillonnage.

Le théorème de Shannon ([1]) concerne les signaux dont le spectre possède une fréquence maximale f_{max}, que l'on appelle des signaux à bande limitée. Par exemple, si $u(t)$ est un polynôme trigonométrique, la fréquence maximale est celle de la plus grande harmonique.

Théorème de Shannon : pour que le signal puisse être entièrement reconstruit à partir des échantillons, il faut et il suffit que :

$$f_e > 2f_{max} \tag{3}$$

La fréquence d'échantillonnage doit être strictement supérieure à deux fois la plus grande fréquence présente dans le spectre du signal continu (condition de Nyquist-Shannon). Si cette condition est vérifiée alors :

$$u(t) = \sum_{k=-\infty}^{+\infty} u_k \text{sinc}\left(\frac{t - kT_e}{T_e}\right) \tag{4}$$

où la fonction sinus cardinale est définie par :

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x} \tag{5}$$

Cette relation montre que le signal peut être reconstruit à partir des échantillons, ce qui signifie que toute l'information présente dans le signal original est conservée dans les échantillons. Nous verrons plus loin comment l'opération de reconstruction est effectuée en pratique.

La moitié de la fréquence d'échantillonnage est appelée la fréquence de Nyquist f_n et la condition de Nyquist-Shannon s'écrit donc $f_{max} < f_n$.

Lorsque la condition n'est pas vérifiée, on dit qu'il y a sous-échantillonnage. On parle de sur-échantillonnage lorsque la fréquence de Nyquist est beaucoup plus grande que f_{max}.

2.b. Fonction sinusoïdale

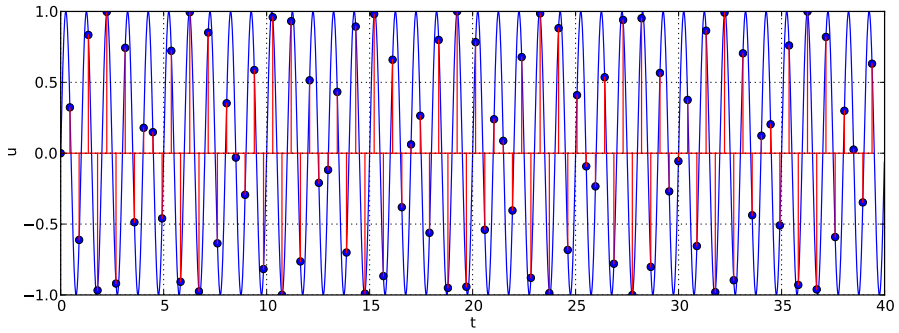
Pour illuster le théorème de Shannon, considérons tout d'abord le cas d'une fonction sinusoïdale. On définit une fonction de période 1 :

```
import math
def u(t):
    return math.sin(2*math.pi*t)
```

La fréquence maximale est évidemment $f_{max} = 1$. Nous allons effectuer deux échantillonnages de cette fonction. Le premier avec une fréquence grande devant 1, pour tracer la sinusoïde, le second avec une fréquence plus faible mais respectant la condition de Nyquist-Shannon (supérieure à 2)

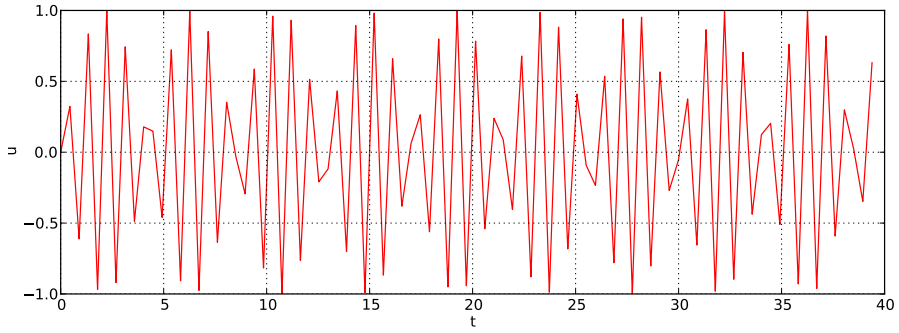
```
import numpy
from matplotlib.pyplot import *
T=40.0
fe1 = 100.0
te1 = 1.0/fe1
N1 = int(T*fe1)
t1 = numpy.arange(0,N1)*te1
x1 = numpy.zeros(N1)
for k in range(N1):
    x1[k] = u(t1[k])
fe2 = 2.234
te2 = 1.0/fe2
N2 = int(T*fe2)
t2 = numpy.arange(0,N2)*te2
```

```
x2 = numpy.zeros(N2)
for k in range(N2):
    x2[k] = u(t2[k])
figure(figsize=(12,4))
plot(t1,x1,"b")
stem(t2,x2,"r")
xlabel("t")
ylabel("u")
grid()
```



Si l'on relie les échantillons par des segments, on obtient bien sûr une très mauvaise représentation de la sinusoïde :

```
figure(figsize=(12,4))
plot(t2,x2,"r")
xlabel("t")
ylabel("u")
grid()
```

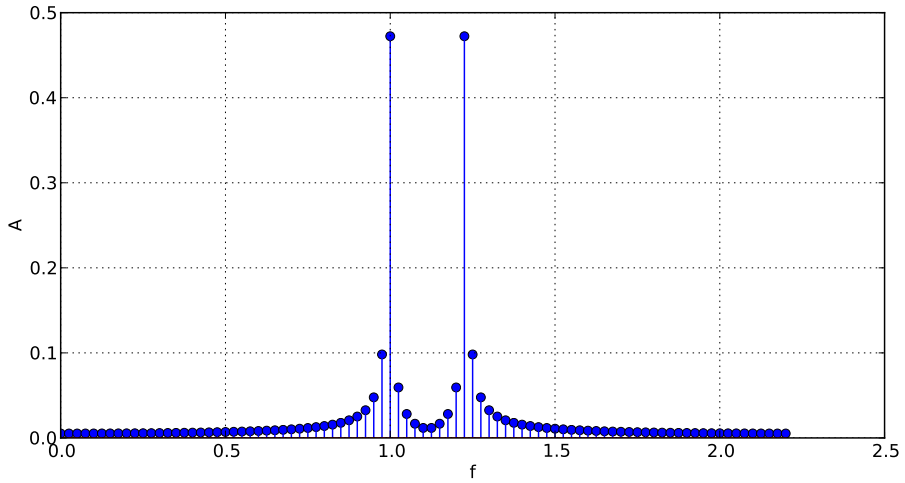


D'après le théorème de Shannon, il est pourtant possible de reconstruire complètement le signal. Pour le faire, nous allons plutôt nous placer dans l'espace des fréquences, en calculant la transformée de Fourier discrète des échantillons.

```
import numpy.fft
tfd = numpy.fft.fft(x2)
f = numpy.arange(0,N2)*1.0/T
```

```
figure(figsize=(10,5))
stem(f,numpy.absolute(tfd)/N2)
xlabel('f')
ylabel('A')
grid()


print(N2)
--> 89
```

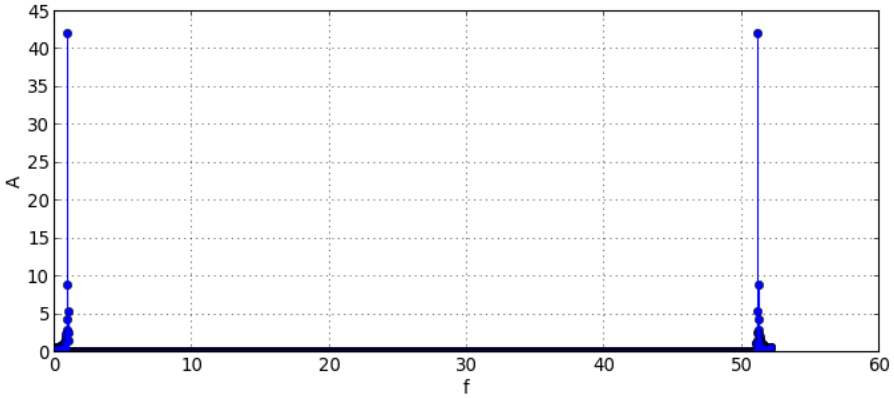


Le spectre du signal discret comporte deux maxima, le premier à la fréquence 1, et son image à la fréquence $2.234 - 1$. Nous allons couper la transformée de Fourier discrète en deux parties :

```
tfd_A = tfd[0:N2/2]
tfd_B = tfd[N2/2:N2]
```

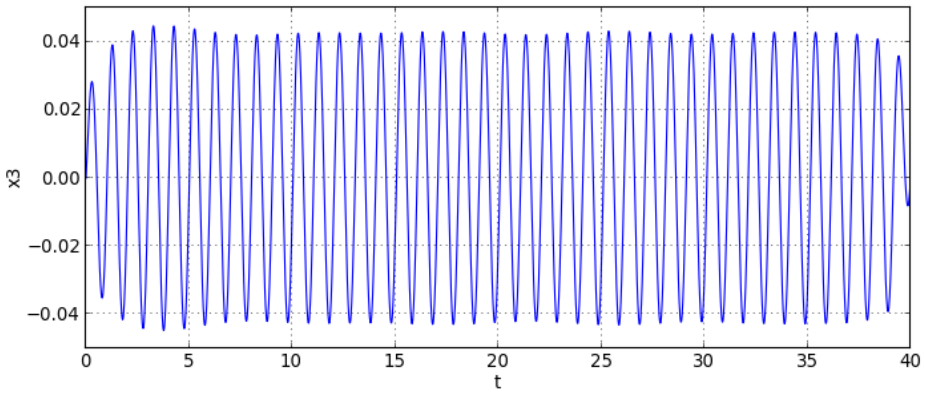
On peut à présent augmenter la fréquence d'échantillonnage en ajoutant des zéros entre ces deux parties. L'intervalle de fréquence entre deux points voisins reste $1/T$. La nouvelle fréquence d'échantillonnage se calcule à partir du nombre de points total.

```
nz = 2000
N3 = nz+N2
zeros = numpy.zeros(nz)
tfd3 = numpy.concatenate((tfd_A,zeros,tfd_B))
fe3 = N3/T
f = numpy.arange(0,N3)*1.0/T
figure(figsize=(10,4))
stem(f,numpy.absolute(tfd3))
xlabel('f')
ylabel('A')
grid()
```



Cette opération effectuée dans le domaine fréquentiel revient à augmenter la fréquence d'échantillonnage sans perdre d'information. Il reste à effectuer la transformée de Fourier discrète inverse :

```
x3 = numpy.fft.ifft(tfd3)
t3 = numpy.arange(0,N3)*1.0/fe3
figure(figsize=(10,4))
plot(t3,x3)
xlabel('t')
ylabel('x3')
grid()
```



Bien que le résultat ne soit pas parfait, nous obtenons une reconstruction de la sinusoïde initiale. La formule de Shannon (4) s'applique à un signal non limité dans le temps. En pratique, il a une durée finie T, c'est pourquoi la reconstruction est imparfaite. Nous verrons plus loin que la reconstruction se fait en pratique dans le domaine temporel et non pas de cette manière.

2.c. Signal périodique

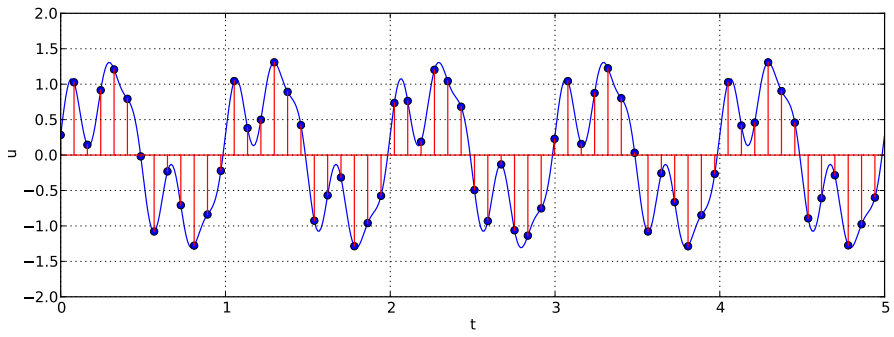
Une fonction périodique est décomposée en somme de fonctions sinusoïdales (série de Fourier). Voici un exemple avec 3 harmoniques, de rang 1,3 et 5 :

```
def u(t):
```

```
    return 1.0*math.sin(2*math.pi*t)\
          +0.5*math.sin(3*2*math.pi*t+math.pi/3)\
          +0.3*math.sin(5*2*math.pi*t-math.pi/6)
```

La plus grande fréquence du spectre du signal est celle de l'harmonique de rang 5. Pour respecter la condition de Nyquist-Shannon, il faut donc une fréquence d'échantillonnage supérieure à 10. On échantillonne 40 périodes avec une fréquence 12.345. La fréquence d'échantillonnage est choisie non multiple de celle du signal, comme c'est le plus souvent en réalité.

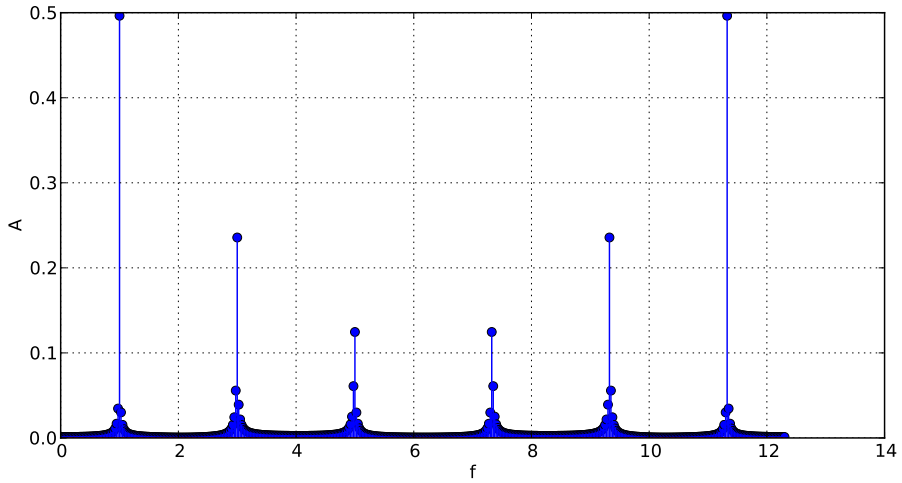
```
T=40.0
fe1 = 100
te1 = 1.0/fe1
N1 = int(T*fe1)
t1 = numpy.arange(0,N1)*te1
x1 = numpy.zeros(N1)
for k in range(N1):
    x1[k] = u(t1[k])
fe2 = 12.345
te2 = 1.0/fe2
N2 = int(T*fe2)
t2 = numpy.arange(0,N2)*te2
x2 = numpy.zeros(N2)
for k in range(N2):
    x2[k] = u(t2[k])
figure(figsize=(12,4))
plot(t1,x1,"b")
stem(t2,x2,"r")
xlabel("t")
ylabel("u")
axis([0,5,-2,2])
grid()
```



Voyons le spectre du signal discret :

```
tfd = numpy.fft.fft(x2)
f = numpy.arange(0,N2)*1.0/T
```

```
figure(figsize=(10,5))
stem(f,numpy.absolute(tfd)/N2)
xlabel('f')
ylabel('A')
grid()
```



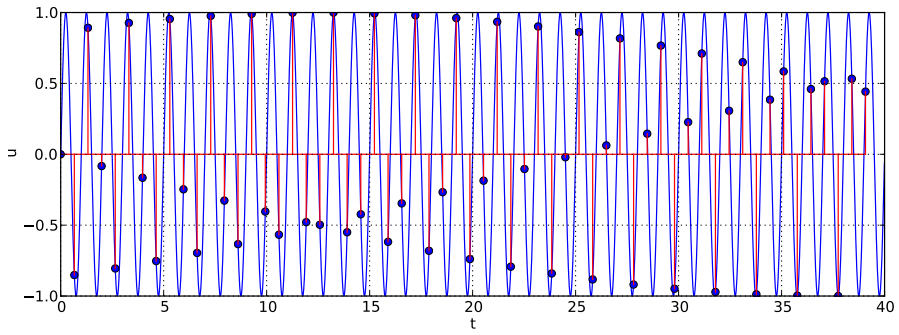
On voit sur ce spectre que la condition de Nyquist-Shannon est bien respectée : le spectre du signal continu, constitué des trois raies de fréquences 1,3 et 5, est bien obtenu sur la première moitié. Autrement dit, le spectre du signal et son image ne se chevauchent pas. Comme pour la sinusoïde, il est possible de reconstituer complètement le signal à partir des échantillons.

2.d. Repliement de bande

Le repliement de bande se produit lorsque la condition de Nyquist-Shannon n'est pas respectée. Voici un exemple de sinusoïde sous-échantillonnée :

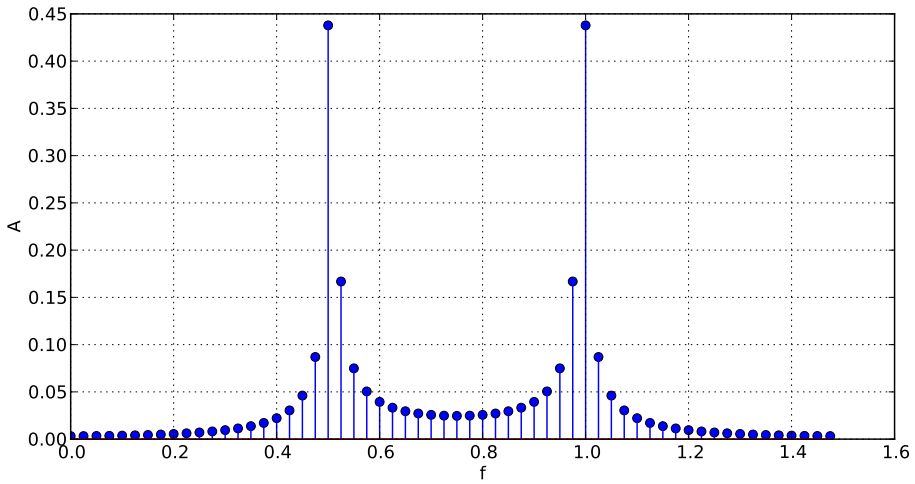
```
def u(t):
    return math.sin(2*math.pi*t)
T=40.0
fe1 = 100.0
te1 = 1.0/fe1
N1 = int(T*fe1)
t1 = numpy.arange(0,N1)*te1
x1 = numpy.zeros(N1)
for k in range(N1):
    x1[k] = u(t1[k])
fe2 = 1.51
te2 = 1.0/fe2
N2 = int(T*fe2)
t2 = numpy.arange(0,N2)*te2
x2 = numpy.zeros(N2)
for k in range(N2):
    x2[k] = u(t2[k])
```

```
figure(figsize=(12,4))
plot(t1,x1,"b")
stem(t2,x2,"r")
xlabel("t")
ylabel("u")
grid()
```



et le spectre du signal discret :

```
tfd = numpy.fft.fft(x2)
f = numpy.arange(0,N2)*1.0/T
figure(figsize=(10,5))
stem(f,numpy.absolute(tfd)/N2)
xlabel('f')
ylabel('A')
grid()
```



Le spectre obtenu est toujours symétrique par rapport à la fréquence de Nyquist, mais la partie de gauche ne correspond pas du tout au spectre du signal continu, puisque le maximum se trouve à 0.5 au lieu de 1. Ce spectre comporte en fait une raie à la fréquence $f = 1$ du signal et une autre à la fréquence $f_e - f$. Si on cherche à reconstituer le signal continu à partir de ces échantillons, on obtient une sinusoïde de fréquence $f_e - f = 0.51$, de

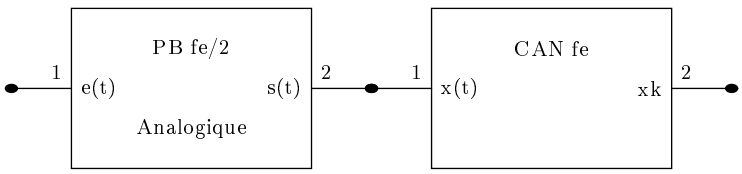
plus basse fréquence que la sinusoïde initiale. L'apparition de basses fréquences parasites est une conséquence du sous-échantillonnage qui peut être très gênante. Non seulement il y a perte d'information, mais il apparaît des informations non présentes dans le signal continu d'origine.

Le spectre obtenu peut s'interpréter en remarquant que le spectre d'une sinusoïde échantillonnée comporte toujours deux raies de fréquences f et $f_e - f$. Lorsque la condition de Nyquist-Shannon est respectée $f < f_e - f$. Lorsqu'elle ne l'est pas $f_e - f < f$. On dit que l'image du spectre (la raie $f_e - f$) s'est repliée dans la bande de fréquence $[0, f_e/2]$, c'est pourquoi on parle de repliement de bande.

2.e. Filtrage anti-repliement

Ce qui précède montre que le sous-échantillonnage doit être absolument évité. Dans le cas d'une conversion analogique-numérique, par exemple lors de la numérisation du son, la fréquence maximale f_{max} du signal peut être assez grande, alors que la fréquence d'échantillonnage f_e est limitée par la cadence de travail du circuit électronique de numérisation.

Si la fréquence d'échantillonnage maximale praticable est inférieure à $2f_{max}$, une solution consiste à effectuer un filtrage passe-bas analogique du signal avant sa numérisation, de manière à enlever de son spectre les fréquences supérieures à $f_e/2$. Ce type de filtre est appelé filtre anti-repliement. Idéalement, un filtre anti-repliement doit avoir un gain de 1 dans la bande passante $[0, f_e/2]$, nul en dehors. La figure suivante montre le schéma bloc du dispositif de numérisation comportant le filtre anti-repliement et le convertisseur analogique-numérique :



En réalité, le filtre anti-repliement est difficile à réaliser. On préfère donc, lorsque c'est possible, augmenter la fréquence d'échantillonnage.

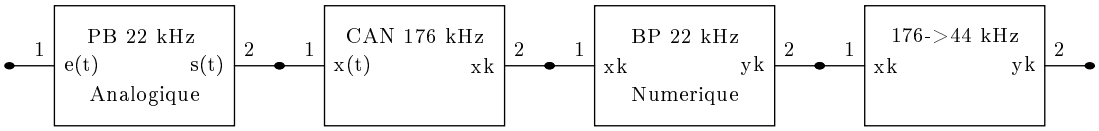
Prenons l'exemple de la numérisation du son. L'oreille humaine perçoit les sons jusqu'à 20 kHz. Pour numériser le son en vue d'une restitution haute-fidélité, il faut donc procéder à une fréquence d'au moins 40 kHz. Les sons de fréquence supérieure à 20 kHz sont inaudibles mais ils peuvent se retrouver dans la bande audible par le phénomène de repliement de bande. Il faut appliquer un filtrage passe-bas qui enlève les fréquences au delà de 20 kHz. Considérons par exemple un filtre passe-bas du premier ordre de fréquence de coupure $f_c = 20\ kHz$. Le gain a la forme suivante :

$$G(f) = \frac{1}{\sqrt{1 + \left(\frac{f}{f_c}\right)^2}} \tag{6}$$

Ce filtre a une pente de -20 decibel par décade dans la bande atténuée, ce qui n'est pas suffisant pour enlever les fréquences situées juste au dessus de 20 kHz, par exemple

25 kHz. Si on abaisse la fréquence de coupure, on risque d'introduire une distorsion du son. Il faut donc utiliser un filtre beaucoup plus sélectif, plus difficile à réaliser, surtout s'il faut minimiser la distorsion dans la bande passante.

La solution adoptée aujourd'hui pour la numérisation du son est celle du sur-échantillonnage. Par exemple, avec une fréquence d'échantillonnage de 176 kHz, le filtre devra avoir un gain de 1 dans la bande $[0, 20\ kHz]$ mais n'aura pas besoin d'être très sélectif. Il suffira que son gain à 96 kHz soit assez faible pour éliminer les fréquences au delà. En fait, le filtre anti-repliement n'est même plus nécessaire car les microphones effectuent naturellement ce filtrage. L'enregistrement du son se fait à une fréquence de 44 kHz (par exemple sur CD audio). Le passage de 192 kHz à 44 kHz se fait après avoir appliqué un filtre anti-repliement numérique très sélectif, bien plus simple à réaliser qu'un filtre analogique. Voici le schéma bloc du dispositif complet :

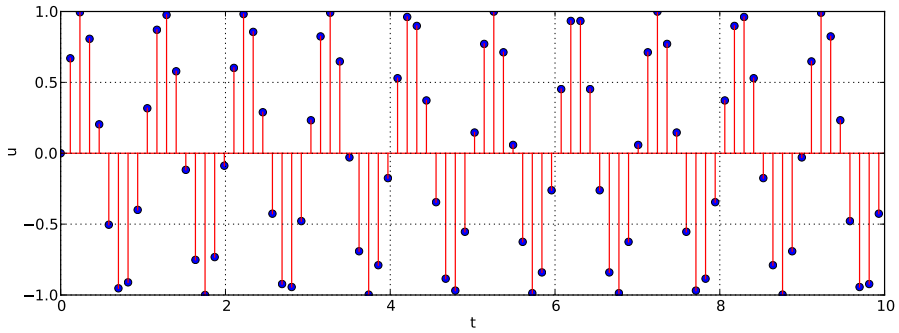


3. Reconstruction

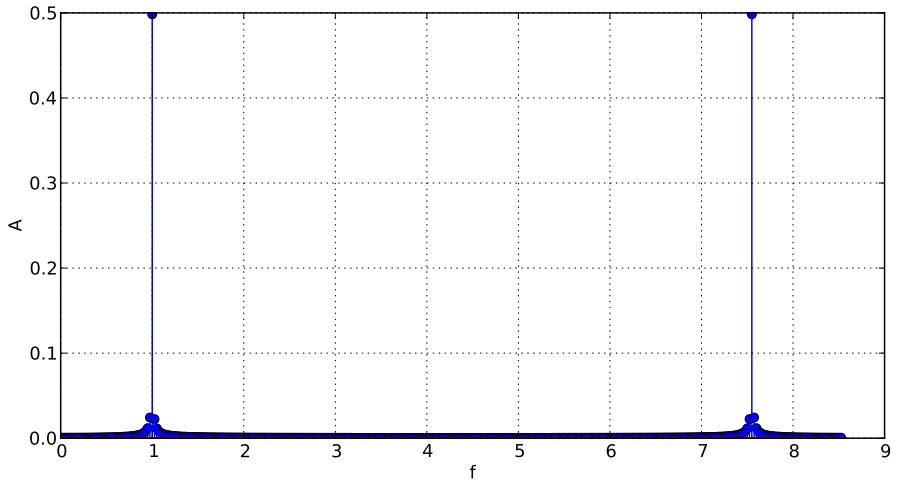
3.a. Filtre analogique de lissage

La reconstruction du signal se fait lors de la conversion numérique-analogique, par exemple dans un lecteur de CD audio. L'objectif est de reconstruire un signal continu (analogique) le plus proche possible du signal dont le spectre est celui de la bande $[0, f_e/2]$. Voyons cela sur l'exemple d'une sinusoïde de période 1, que l'on échantillonne à une fréquence supérieure à 2.

```
def u(t):
    return math.sin(2*math.pi*t)
T=40.0
fe2 = 8.56
te2 = 1.0/fe2
N2 = int(T*fe2)
t2 = numpy.arange(0,N2)*te2
x2 = numpy.zeros(N2)
for k in range(N2):
    x2[k] = u(t2[k])
figure(figsize=(12,4))
stem(t2,x2,"r")
xlabel("t")
ylabel("u")
axis([0,10,-1,1])
grid()
```



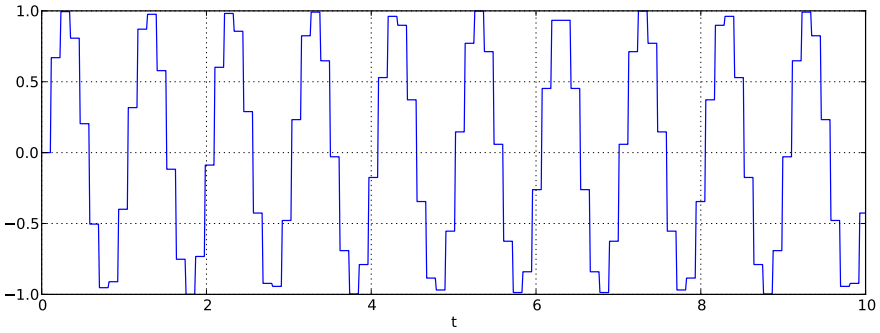

```
tfd = numpy.fft.fft(x2)
f = numpy.arange(0,N2)*1.0/T
figure(figsize=(10,5))
stem(f,numpy.absolute(tfd)/N2)
xlabel('f')
ylabel('A')
grid()
```



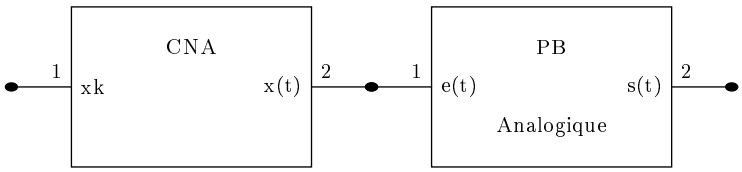
La sortie d'un convertisseur numérique-analogique n'est pas constituée de points comme le signal discret mais de paliers. En effet un circuit appelé échantillonneur-bloqueur maintient la tension de sortie constante entre deux échantillons. On peut simuler l'effet de l'échantillonneur-bloqueur. Pour cela, il faut augmenter la fréquence d'échantillonnage d'un facteur n :

```
def echantBloqueur(x,fe,n):
    N = x.size
    y = numpy.zeros(0)
    t = numpy.arange(0,N*n)*1.0/(fe*n)
    for k in range(N):
        y = numpy.concatenate((y,numpy.ones(n)*x[k]))
    return (t,y)
n=10
N3=N2*n
```

```
fe3 = fe2*n
(t,x3) = echantBloqueur(x2,fe2,n)
figure(figsize=(12,4))
plot(t,x3)
xlabel('t')
axis([0,10,-1,1])
grid()
```



Pour reconstruire la sinusoïde d'origine à partir de ce signal, il faut utiliser un filtre de lissage. Il s'agit d'un filtre analogique placé après le convertisseur numérique-analogique. D'un point de vue fréquentiel, la fonction de ce filtre est d'enlever les fréquences de la bande $[f_e/2, f_e]$, c'est-à-dire les fréquences de l'image du spectre du signal analogique. Pour cette raison, le filtre de lissage est aussi appelé filtre anti-image. En toute rigueur, il faudrait tenir compte de la modification du spectre apportée par l'échantillonneur-bloqueur ([2]), ce que nous ne ferons pas ici.



Idéalement, le filtre de lissage est un filtre passe-bas dont le gain vaut 1 dans la bande $[0, f_{max}]$ (avec une phase variant linéairement avec la fréquence), 0 dans la bande $[f_e/2, fe]$. Si la fréquence f_{max} est proche de la fréquence de Nyquist (la moitié de f_e), le filtre de lissage est très difficile à réaliser (comme le filtre anti-repliement). Au contraire, si f_{max} est faible devant la fréquence de Nyquist (sur-échantillonnage), le filtre de lissage est très facile à réaliser (un simple filtre RC suffit).

On peut simuler l'effet du filtre de lissage avec un filtre numérique RIF. Voir à ce sujet le document Exemples de filtres RIF. On choisit la fréquence de coupure égale à la moitié de la fréquence d'échantillonnage (avant l'augmentation du facteur n). Voici la réponse impulsionnelle (infinie) du filtre passe-bas idéal est :

$$g_k = 2a\, sinc(k2a) \tag{7}$$

où $a = f_c/f_e = 0.5$ et la fonction sinus cardinale a été définie plus haut (5). On a donc :

$$g_k = sinc(k) \tag{8}$$

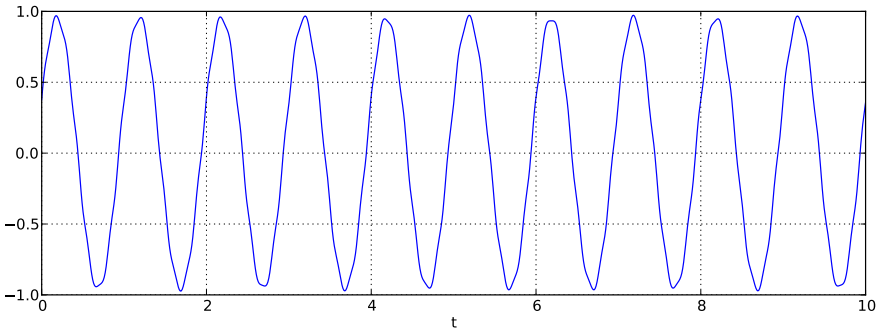
On obtient précisément le sinus cardinal qui apparaît dans la formule de Shannon (4). En pratique, il faut tronquer la réponse impulsionnelle au rang P pour la rendre finie. Pour filtrer le signal, il faut par ailleurs diviser la fréquence de coupure par n. Pour générer la réponse impulsionnelle finie, on utilise la fonction `scipy.signal.firwin`, en utilisant un fenêtrage de Hann pour réduire les ondulations dans la bande passante :

```
import scipy.signal
P = 10
h = scipy.signal.firwin(numtaps=2*P+1,cutoff=[0.5/n],nyq=0.5,window='hann')
```

P est l'indice de troncature de la réponse impulsionnelle, qu'il faut augmenter pour rendre le filtre plus sélectif.

Voici le résultat du filtrage :

```
y3 = scipy.signal.convolve(x3,h,mode='valid')
t3 = numpy.arange(0,y3.size)*1.0/fe3
figure(figsize=(12,4))
plot(t3,y3)
xlabel('t')
axis([0,10,-1,1])
grid()
```



3.b. Filtre numérique d'interpolation

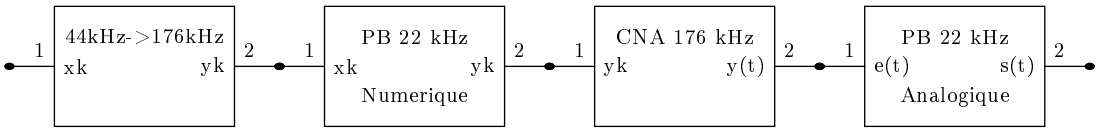
La technique précédente, consistant à utiliser un filtre de lissage analogique pour reconstituer le signal, est difficile à mettre en œuvre, surtout lorsque la fréquence de Nyquist est juste supérieure à f_{max}. Comme pour le filtre anti-repliement, on se heurte à la difficulté qu'il y a à réaliser un filtre analogique très sélectif sans distorsion dans la bande passante.

Une autre solution est d'augmenter la fréquence d'échantillonnage de manière à effectuer un lissage numériquement, avant la conversion numérique-analogique. Le filtre de lissage analogique est alors beaucoup plus simple à réaliser car la fréquence de Nyquist est plus élevée. Le filtre de lissage numérique est appelé filtre d'interpolation ; il s'agit

d'un filtre passe-bas dont la fréquence de coupure est la moitiée de la fréquence d'échantillonnage avant la multiplication. La réalisation d'un filtre passe-bas numérique très sélectif ne pose pas de difficulté. C'est exactement ce que nous avons fait dans l'exemple précédent, où la fréquence d'échantillonnage a été augmenté d'un facteur 10 avant d'appliquer le filtrage passe-bas numérique.

Le filtre d'interpolation réalise ainsi la convolution exprimée par la formule de Shannon (4), convolution entre les échantillons et un sinus cardinal. En pratique, la reconstitution est imparfaite car le sinus cardinal doit être tronqué pour obtenir une réponse impulsionnelle finie.

Cette technique est utilisée dans les lecteurs CD audio, où la fréquence de base de 44 kHz est augmentée d'un facteur 4 avant d'appliquer le filtre numérique d'interpolation (passe-bas 22 kHz). Comme la nouvelle fréquence du convertisseur numérique-analogique est 176 kHz, la fréquence de Nyquist (88 kHz) est bien plus grande que $f_{max} = 20\ kHz$, ce qui permet d'utiliser un simple filtre du premier ordre pour le lissage analogique. La figure suivante montre le schéma bloc de la chaîne complète :



Références

[1] Gasquet C., Witomski P., *Analyse de Fourier et applications*, (Masson, 1995)

[2] Tan Li, Jiang Jean, *Digital signal processing : fundamentals and applications*, (Elsevier, 2013)