Échantillonnage d'un signal

1. Introduction

L'échantillonnage est une opération courante non seulement en conversion analogique-numérique, mais aussi dans tout calcul numérique consistant à générer des valeurs discrètes à partir d'une fonction continue (échantillonnage de fonctions, synthèse d'images, etc).

On verra la nécessité d'une fréquence d'échantillonnage suffisante pour éviter le phénomène d'aliasing. Le principe de l'échantillonnage stochastique est aussi présenté.

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

2. Échantillonnage périodique

Soit une fonction à une variable réelle $u(t)$. La variable t représente le temps ou tout autre variable, par exemple une variable d'espace. L'échantillonnage périodique consiste à fixer une période d'échantillonnage T_e et à obtenir une suite de nombres définie par :

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

où l'indice k est entier. La fréquence d'échantillonnage est $f_e = 1/T_e$.

Dans certains cas (systèmes électroniques temps-réel), l'indice k n'est pas borné. Dans d'autres cas, on prélève un nombre fini N d'échantillons et l'indice k varie de 0 à $N-1$. Nous allons considérer ce cas. L'intervalle sur lequel la fonction est échantillonnée est $[0, T]$.

La fonction qui servira d'exemple est la suivante :

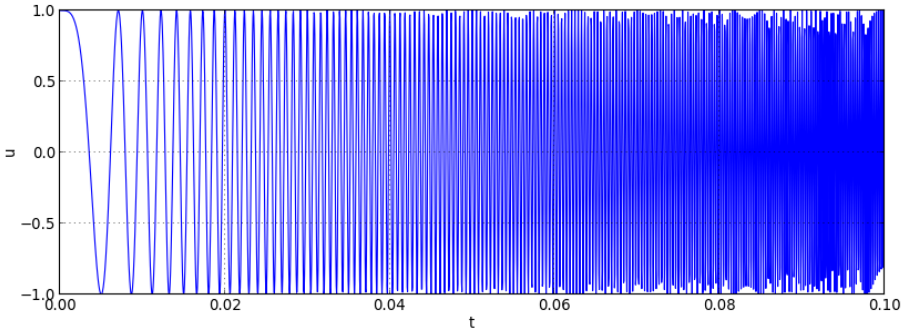
```
a = 20000
def u(t):
    return math.cos(2*math.pi*a*(1-t*t))
```

que l'on échantillonne sur l'intervalle $[0, 0.1]$:

```
T=0.1
N=2000
Te=T/N
fe=1.0/Te
t = numpy.zeros(N)
echant = numpy.zeros(N)
for k in range(N):
    t[k] = k*Te
    echant[k] = u(t[k])
```

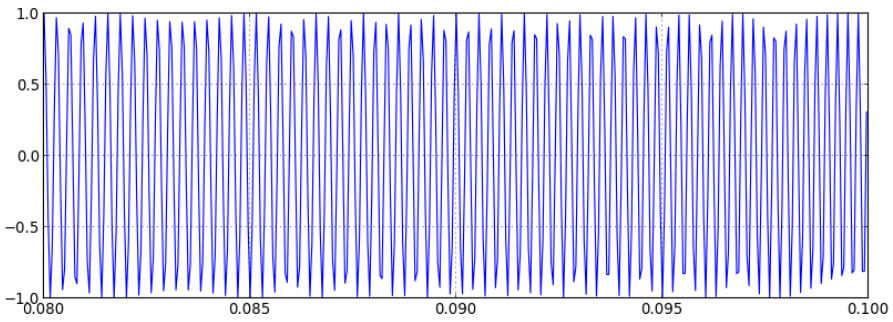
Pour représenter graphiquement les échantillons, on peut relier les points par des segments de droite, ce que l'on fait habituellement lorsqu'on trace une fonction :

```
figure(figsize=(12,4))
plot(t,echant)
xlabel('t')
ylabel('u')
grid()
```



Voyons aussi un détail de la zone de droite :

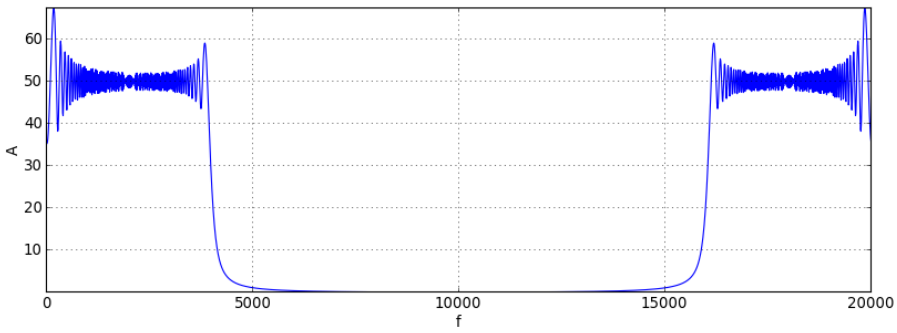
```
figure(figsize=(12,4))
plot(t,echant)
grid()
axis([0.08,0.1,-1,1])
```



Cette fonction présente à la fois des variations lentes et des variations rapides. En modifiant le paramètre a, on peut augmenter ou réduire la présence des variations rapides. La transformée de Fourier discrète permet d'obtenir le spectre de ces échantillons. La résolution du spectre est l'inverse de la durée T.

```
from numpy.fft import fft
tfd = fft(echant)
spectre = numpy.absolute(tfd)
freq = numpy.zeros(N)
for k in range(N):
    freq[k] = k*1.0/T
```

```
figure(figsize=(12,4))
plot(freq,spectre)
axis([0,fe,spectre.min(),spectre.max()])
xlabel('f')
ylabel('A')
grid()
```

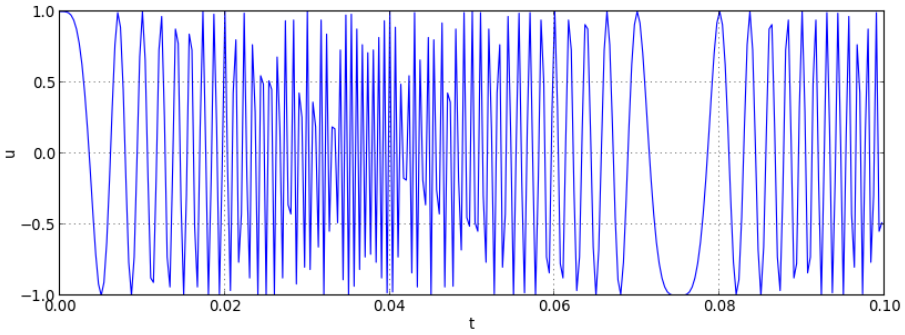


Le spectre s'étend de la fréquence nulle jusqu'à la fréquence d'échantillonnage f_e. La moitié de la fréquence d'échantillonnage est appelée la fréquence de Nyquist, $f_n = f_e/2$. On remarque que le spectre est symétrique par rapport à cette fréquence. La partie de droite est l'image du spectre de la fonction continue.

D'après de théorème de Shannon, les échantillons conservent l'information présente dans le signal d'origine si la fréquence d'échantillonnage est supérieure au double de la plus grande fréquence présente dans le spectre du signal. Notons f_{max} cette plus grande fréquence. Dans le cas présent, on peut l'estimer à environ 6000. La condition de Nyquist-Shannon s'écrit $f_n > f_{max}$. Sur cet exemple, cette condition semble vérifiée, ce qui signifie que la fréquence d'échantillonnage est assez grande. On constate néanmoins sur le tracé des échantillons dans l'intervalle $[0.08, 0.1]$ que la forme des oscillations n'est pas restituée parfaitement (bien que la période soit correcte).

On parle de sous-échantillonnage lorsque le critère de Nyquist-Shannon n'est pas vérifiée. Nous allons nous placer dans ce cas en réduisant la fréquence d'échantillonnage :

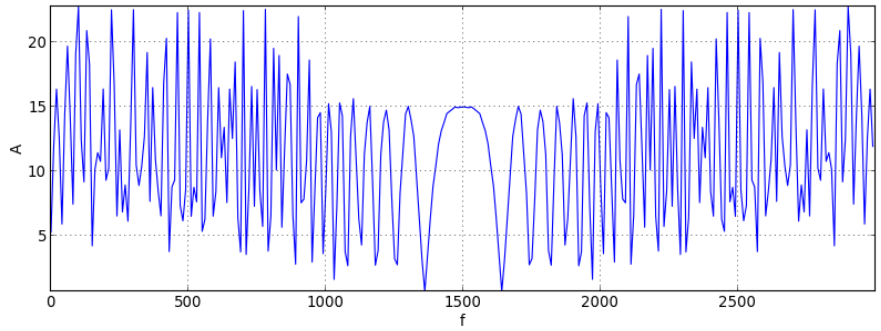
```
N=300
Te=T/N
fe=1.0/Te
t = numpy.zeros(N)
echant = numpy.zeros(N)
for k in range(N):
    t[k] = k*Te
    echant[k] = u(t[k])
figure(figsize=(12,4))
plot(t,echant)
xlabel('t')
ylabel('u')
grid()
```



La fonction d'origine n'est pas restituée correctement dans les zones de haute fréquence. On remarque aussi la présence d'une variation basse-fréquence (un peut avant 0.08) qui n'est pas du tout présente dans le signal d'origine. L'apparition de basses fréquences dans le signal échantillonné est la conséquence la plus gênante du sous-échantillonnage. Le mot anglais pour désigner ce phénomène est aliasing (du latin alias). Dans certains cas comme celui-ci, on observe en effet une réplique du signal de basse-fréquence.

Voyons le spectre des échantillons :

```
tfd = fft(echant)
spectre = numpy.absolute(tfd)
freq = numpy.zeros(N)
for k in range(N):
    freq[k] = k*1.0/T
figure(figsize=(12,4))
plot(freq,spectre)
axis([0,fe,spectre.min(),spectre.max()])
xlabel('f')
ylabel('A')
grid()
```



Les deux parties symétriques du spectre se sont mélangées. C'est évidemment une conséquence du non respect de la condition $f_n > f_{max}$. Le résultat est un spectre complètement modifié. Certaines parties du spectre image se sont repliées sur le spectre. Ce phénomène est appelé le repliement de bande. Il se produit en cas de sous-échantillonnage. Repliement de bande et aliasing sont deux manifestations (l'une fréquentielle et l'autre temporelle) du même phénomène.

Pour éviter le repliement de bande, il faut augmenter la fréquence d'échantillonnage à un niveau suffisant pour respecter la condition $f_e > 2f_{max}$. Dans le domaine de la numérisation des signaux temporels (par exemple la numérisation du son), cette condition est systématiquement réalisée. On procède même dans ce cas à un sur-échantillonnage très important au moment de la numérisation, car l'électronique d'aujourd'hui le permet sans problème. Dans d'autres domaines, comme la numérisation des images, le sur-échantillonnage n'est pas toujours possible. Par exemple, un capteur d'appareil photo numérique a une fréquence d'échantillonnage fixée par le nombre de pixels. Il faut alors procéder à un filtrage analogique passe-bas avant d'effectuer la numérisation (filtre anti-aliasing).

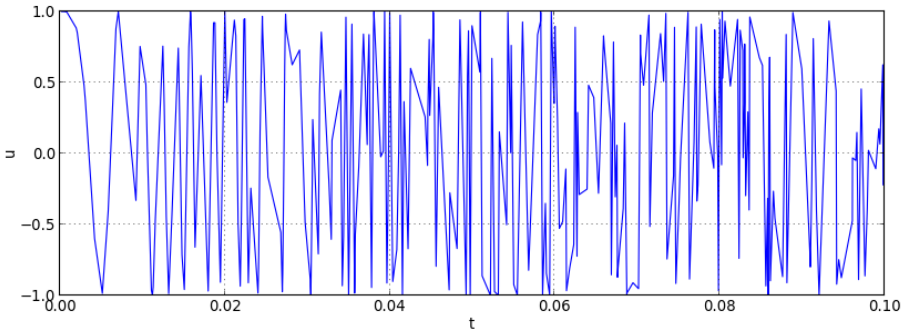
Dans le domaine purement numérique (échantillonnage de fonctions, synthèse d'images), le filtrage anti-aliasing n'est pas réalisable. Nous allons voir d'autres techniques d'échantillonnage qui permettent de réduire les effets d'aliasing.

3. Échantillonnage stochastique

L'échantillonnage stochastique, ou échantillonnage de Monte-Carlo, peut être utilisé dans le domaine de la synthèse de signaux, par exemple la synthèse d'image par tracé de rayons. Il consiste à choisir les points à échantillonner de manière plus ou moins aléatoire.

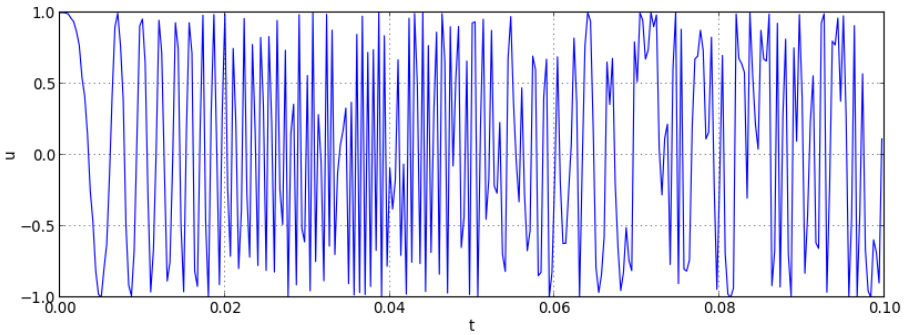
On peut commencer par choisir les N points à échantillonner aléatoirement sur l'intervalle $[0, T]$, avec une densité de probabilité uniforme. Il faut bien sûr ordonner les échantillons par t croissant avant de les tracer.

```
from random import random
N=300
table = numpy.zeros((N,2))
for k in range(N):
    table[k][0] = random()*T
    table[k][1] = u(table[k][0])
def key(elem):
    return elem[0]
table = sorted(table,key=key)
table=numpy.transpose(table)
t = table[0]
echant = table[1]
figure(figsize=(12,4))
plot(t,echant)
xlabel('t')
ylabel('u')
grid()
```



Le phénomène d'aliasing a disparu. Cette méthode est toutefois peu satisfaisante car les échantillons peuvent se concentrer à certains endroits ou au contraire manquer à d'autres endroits. Pour remedier à cela, on effectue un échantillonnage stochastique stratifié. Il s'agit de découper l'intervalle $[0, T]$ en N parties égales (comme pour l'échantillonnage périodique), et à placer l'instant t aléatoirement dans chaque partie. On garde cependant une répartition périodique des instants dans le tableau final.

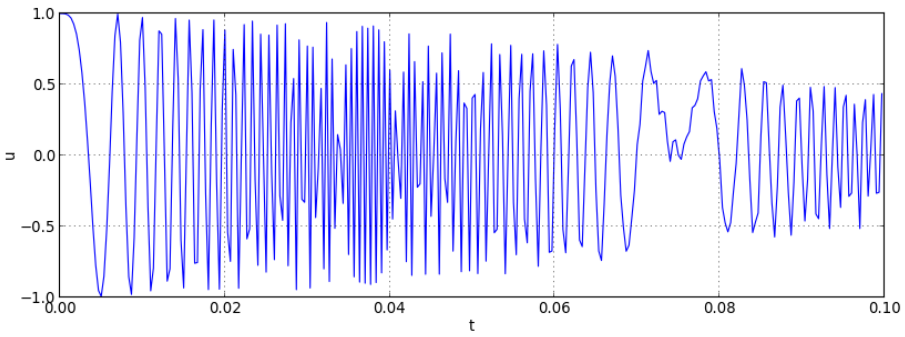
```
N=300
t = numpy.zeros(N)
echant = numpy.zeros(N)
Te=T/N
for k in range(N):
    t[k] = k*Te
    echant[k] = u(t[k]+Te*0.5*(random()))
figure(figsize=(12,4))
plot(t,echant)
xlabel('t')
ylabel('u')
grid()
```



Cette méthode permet de restituer correctement les bases fréquences du signal, tout en réduisant fortement le phénomène d'aliasing. Bien sûr, il est préférable, si on le peut, d'augmenter la fréquence d'échantillonnage.

Une variante, utilisée en synthèse d'images, consiste à choisir plusieurs échantillons aléatoirement sur chaque intervalle de longueur T_e, et à retenir la moyenne arithmétique des échantillons.

```
N=300
t = numpy.zeros(N)
echant = numpy.zeros(N)
Te=T/N
ne = 100
for k in range(N):
    t[k] = k*Te
    moy = 0.0
    for i in range(ne):
        moy += u(t[k]+Te*0.5*(random()))
    moy /= ne
    echant[k] = moy
figure(figsize=(12,4))
plot(t,echant)
xlabel('t')
ylabel('u')
grid()
```



Ce moyennage fait revenir le phénomène d'aliasing. Cette méthode est plutôt utilisée pour réduire l'effet de crénelage au voisinage des bords nets dans une image.

4. Sur-échantillonnage

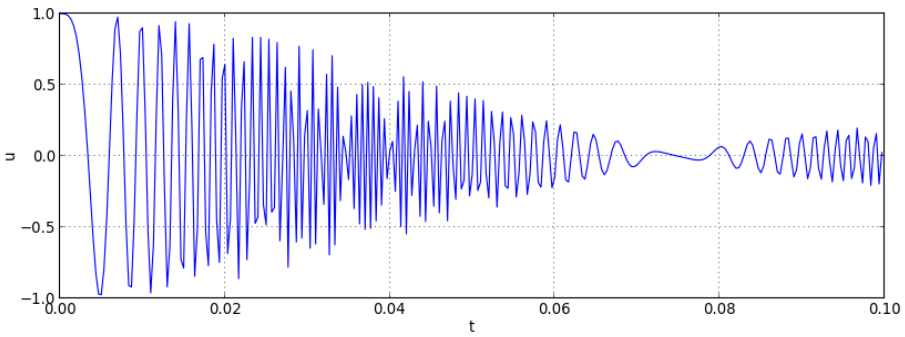
La meilleure méthode, lorsqu'elle est possible, est le sur-échantillonnage. Une fois l'échantillonnage à très hautre fréquence effectuée, il peut être nécessaire de réduire le nombre d'échantillons afin de limiter la quantité de données à stocker. Par exemple dans le cas d'une image, la résolution finale de l'image peut être bien inférieure à la résolution d'échantillonnage.

Il faut donc établir une méthode pour réduire le nombre d'échantillons, sans introduire à nouveau de phénomène d'aliasing. Pour y parvenir, il faut d'abord appliquer un filtrage passe-bas aux échantillons avant de réduire leur nombre.

Le filtre passe-bas le plus simple est le filtre moyenneur. Supposons que l'on échantillonne $10N$ points et que l'on souhaite garder N points. Pour chaque groupe de 10 points consécutifs, on garde la valeur moyenne.

```
N=300
Te=T/N
```

```
n = 10
dT = Te/n
t=numpy.zeros(N)
echant=numpy.zeros(N)
for k in range(N):
    t[k] = k*Te
    moy = 0.0
    for i in range(n):
        moy += u(t[k]+i*dT)
    moy /= n
    echant[k] = moy
figure(figsize=(12,4))
plot(t,echant)
xlabel('t')
ylabel('u')
grid()
```



Le résultat est un signal dans lequel les hautes fréquences ont été atténuées (par le filtrage). L'aliasing est présent mais il est consérablement atténué. Le filtre moyenneur n'est pas le meilleur filtre passe-bas. Il est préférable d'utiliser un filtre passe-bas gaussien, comme expliqué dans Introduction aux filtres numériques..