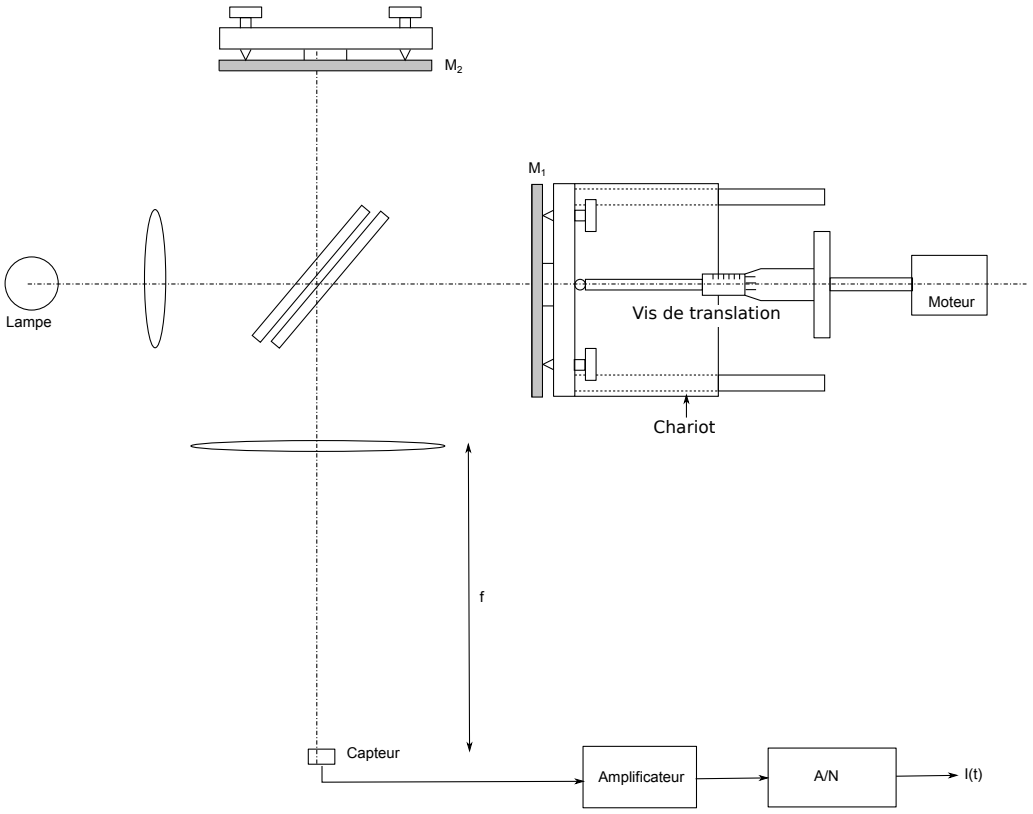
Spectrométrie interférentielle

1. Principe

La *spectrométrie interférentielle* est une technique qui consiste à obtenir le spectre d'une lumière à partir d'interférences. Par exemple, l'expérience des fentes d'Young permet de déterminer la longueur d'onde d'une lumière monochromatique par mesure de l'interfrange.

On considère ici l'utilisation d'un interféromètre de Michelson, pour montrer le principe d'un spectromètre à transformée de Fourier, utilisé couramment en spectroscopie moléculaire dans le domaine de l'infrarouge. L'interféromètre est réglé en lame d'air. Un capteur de lumière est placé au centre des anneaux, au foyer d'une lentille convergente de focale 1 mètre. Le chariot portant le miroir M_1 est entraîné en translation par un moteur dont la vitesse est de un tour par 15 minutes, ce qui fait une variation de différence de marche au centre des anneaux de 10/9 micromètres par seconde. Pour une longueur d'onde de 400 nm, cela fait une variation périodique de l'intensité de fréquence $2,7\ Hz$.



Le capteur a une surface de l'ordre du millimètre carré, très petite devant celle du disque central. Pour une longueur d'onde comprise entre λ et $\lambda + d\lambda$, l'intensité délivrée par le capteur est donc pratiquement l'intensité au centre des anneaux :

$$dI(\delta, \lambda) = S(\lambda)\left(1 + \cos\left(\frac{2\pi}{\lambda}\delta\right)\right) d\lambda \qquad (1)$$

$S(\lambda)$ est une fonction de la longueur d'onde qui prend en compte à la fois le spectre

de la lumière et la réponse spectrale du capteur. $\delta = 2e$ est la différence de marche à l'emplacement du capteur, qui varie à la vitesse de 10/9 micromètres par seconde.

On introduit l'inverse de la longueur d'onde $\sigma = 1/\lambda$ (le nombre d'onde), ce qui permet d'écrire :

$$dI(\delta, \sigma) = S(\sigma)\left(1 + \cos\left(2\pi\sigma\delta\right)\right) d\sigma \tag{2}$$

Pour une différence de marche δ, l'intensité délivrée par le capteur est la somme de ces contributions élémentaires, car les différentes longueurs d'onde sont incohérentes entre elles :

$$I(\delta) = \int_0^\infty S(\sigma)\left(1 + \cos\left(2\pi\sigma\delta\right)\right) d\sigma \tag{3}$$

La fonction $S(\sigma)$ est non nulle sur l'intervalle de nombre d'onde de sensibilité du capteur.

Cette intensité s'écrit :

$$I(\delta) = \int_0^\infty S(\sigma)\, d\sigma + Re\left[\int_0^\infty S(\sigma) e^{-i2\pi\sigma\delta}\, d\sigma\right] \tag{4}$$

Le premier terme est une constante indépendante de δ. On s'intéresse au second terme, appelé *interférogramme*. La fonction $S(\sigma)$ étant réelle et paire, il s'écrit :

$$J(\delta) = \frac{1}{2}\int_{-\infty}^\infty S(\sigma) e^{-i2\pi\sigma\delta}\, d\sigma \tag{5}$$

Il s'agit de la transformée de Fourier de la fonction spectrale $S(\sigma)$. La méthode de *spectrométrie de Fourier* consiste à obtenir expérimentalement la fonction $J(\delta)$, et à en déduire la fonction spectrale par la transformée de Fourier inverse :

$$S(\sigma) = 2\int_{-\infty}^\infty J(\delta) e^{i2\pi\sigma\delta}\, d\delta \tag{6}$$

Les fonction $J(\delta)$ et $I(\delta)$ ne diffèrent que d'une constante additive. On peut donc tout aussi bien calculer la transformée de Fourier inverse de $I(\delta)$, dont seule la composante de fréquence nulle est différente.

Le balayage de δ permettant d'échantillonner la fonction $I(\delta)$ est obtenu par la translation du miroir entraîné par le moteur. Soit v la vitesse de variation de la différence de marche (égale à 10/9 micromètres par seconde). En supposant pour simplifier que la différence de marche est nulle à l'instant zéro, on a :

$$\delta = vt \tag{7}$$

Le signal est numérisé à une fréquence d'échantillonnage f_e. Si N est le nombre total d'échantillons obtenus, la différence de marche maximale est :

$$L = v\frac{N}{f_e} \tag{8}$$

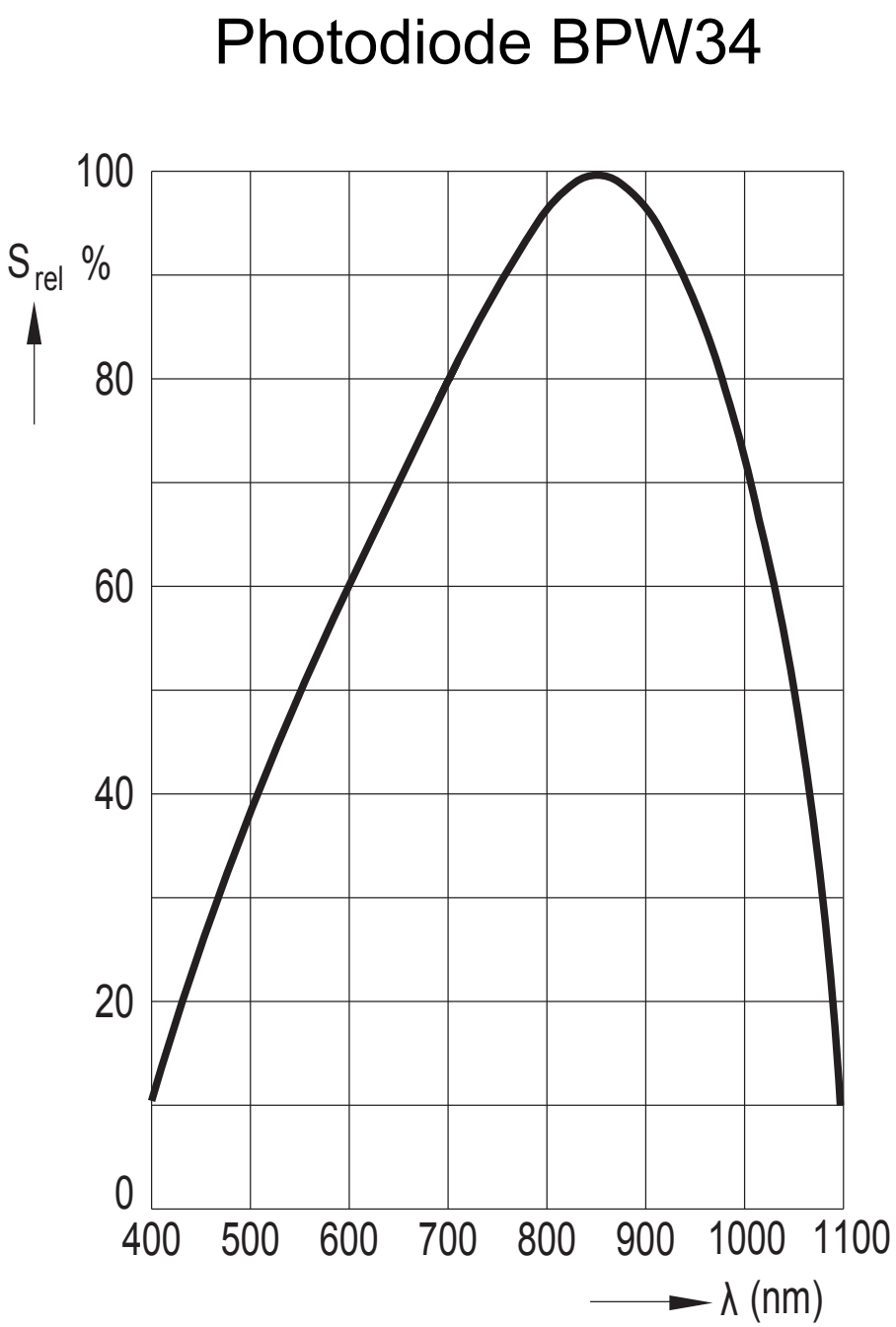
Une transformée de Fourier discrète est appliquée à ces N échantillons, ce qui donne les valeurs de la fonction $S(\sigma)$ pour N nombres d'onde allant de 0 à f_e/v et espacés de $1/L$. La résolution du spectre obtenu est donc en principe $1/L$, l'inverse de la différence de marche maximale. Cette résolution théorique maximale ne peut être obtenue que si le système d'entraînement est assez régulier.

La numérisation est faite avec la carte SysamSP5. Le programme d'acquisition est donné en annexe.

2. Capteur et traitement du signal

2.a. Photodiode et circuit électronique

Le capteur utilisé est une photodiode dont la réponse spectrale est donnée par le constructeur :

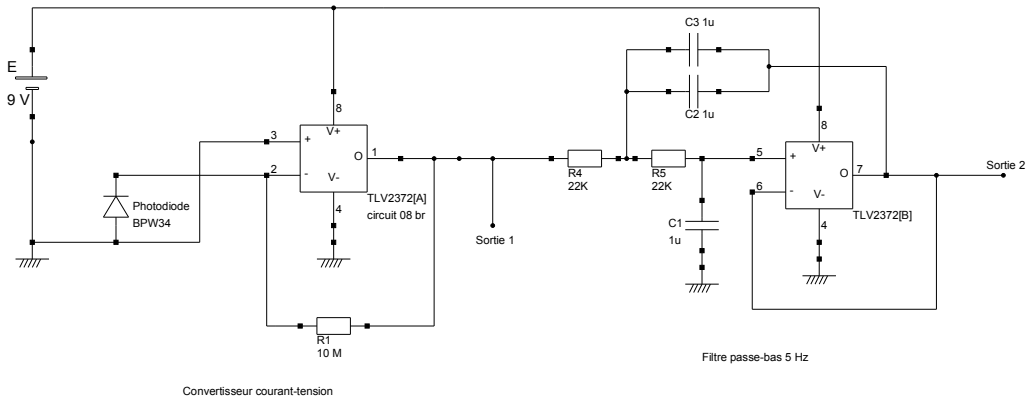


Le maximum de sensibilité se trouve dans l'infrarouge proche. La fonction spectrale utilisée pour le calcul de l'intensité lumineuse s'écrit :

$$S(\sigma) = S_p(\sigma)S_l(\sigma) \tag{9}$$

où $S_p(\sigma)$ est la réponse spectrale de la diode et $S_l(\sigma)$ le spectre de la lumière.

L'intensité du courant électrique délivré par la photodiode est supposée proportionnelle à l'intensité lumineuse $I(\delta)$ définie plus haut. Le circuit électronique comporte un convertisseur courant-tension, qui convertit ce courant en tension en le faisant passer dans une résistance de 10 $M\Omega$. Les tensions obtenues sur la sortie 1 sont de l'ordre du volt, ce qui correspond à des courants dans la photodiode de l'ordre de $0,1$ μA. Voici le schéma du circuit utilisé, qui comporte aussi un filtre passe-bas dont la fonction sera expliquée plus loin :



Les amplificateurs sont alimentés par une pile 9 V et sont de type rail-to-rail, ce qui permet à leur tension de sortie d'atteindre des valeurs proches de zéro. Le modèle utilisé, TLV2372, a des entrées CMOS dont le courant est au maximum de 60 pA, ce qui est très faible comparé au courant délivré par la photodiode.

Le circuit complet comporte en fait un commutateur qui permet de choisir une des 4 résistances suivantes pour la conversion courant-tension : 10 $M\Omega$, 1 $M\Omega$, 100 $k\Omega$ et 10 $k\Omega$. On choisit la résistance pour que l'intensité maximale ne donne pas de saturation de l'amplificateur. Avec une lampe spectrale, la plus grande doit être utilisée. Avec une diode laser, il faut utiliser la plus petite.

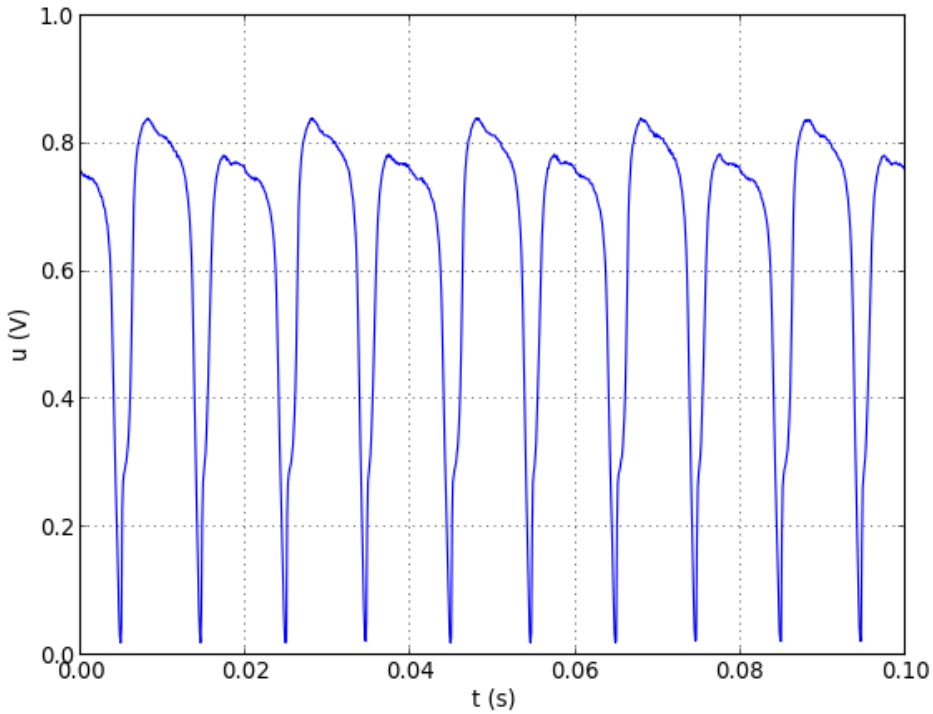
2.b. Étude du signal

On commence par un enregistrement du signal obtenu sur la sortie 1. La lumière est délivrée par une lampe à décharge au sodium. La différence de marche est constante (moteur arrêté). La fréquence d'échantillonnage est de 10 kHz.

```
import numpy
from matplotlib.pyplot import *

data = numpy.loadtxt("lampeNa-7-10kHz.txt")
t = data[0]
u = data[1]
te = t[1]-t[0]
fe = 1.0/te
N = t.size
```

```
figure()
plot(t,u)
xlabel("t (s)")
ylabel("u (V)")
grid()
axis([0,0.1,0,1])
```



On observe des variations cycliques de l'intensité, caractéristiques des lampes à décharge alimentées par une tension alternative de 50 Hz. La valeur qui nous intéresse est la valeur moyenne :

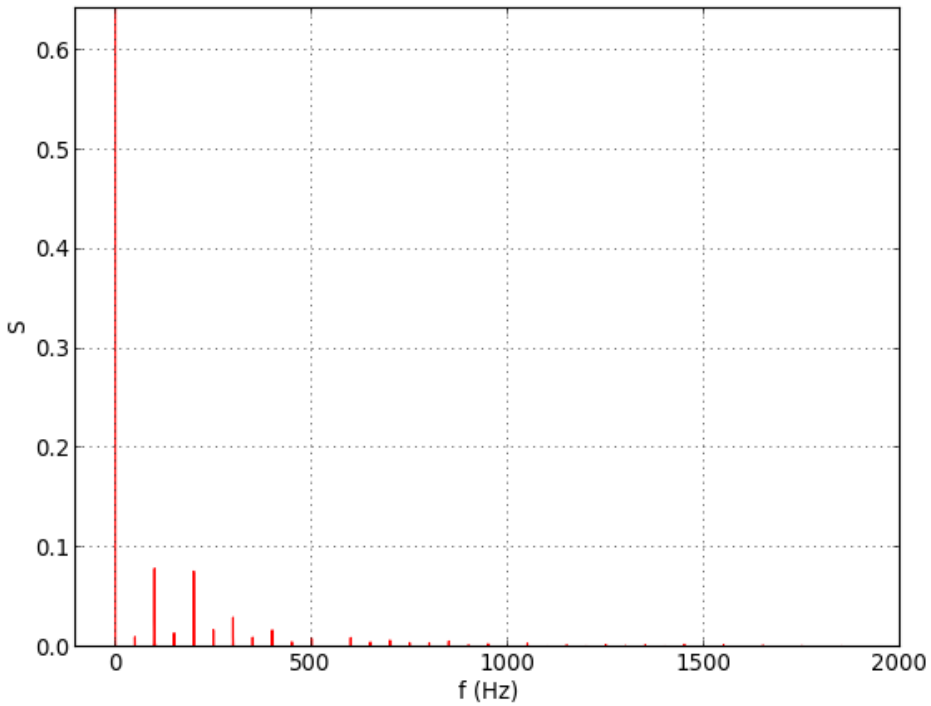
```
print(numpy.mean(u))
--> 0.64262937011718746
```

Voyons le spectre de ce signal :

```
import numpy.fft


spectre = numpy.absolute(numpy.fft.fft(u))/N
f = numpy.arange(N)*fe/N
figure()
plot(f,spectre,'r')
xlabel("f (Hz)")
ylabel("S")
```

```
grid()
axis([-100,2000,0,spectre.max()])
```



La fréquence fondamentale est de 50 Hz, avec des harmoniques de rang pair prépondérantes (100 Hz, 200 Hz). Pour échantillonner correctement ce signal, la fréquence d'échantillonnage doit être au minimun de 1 kHz. On voit aussi sur ce spectre la composante de fréquence nulle, qui correspond à la valeur moyenne de l'intensité.

Le signal qui nous intéresse dépend du temps à cause de la variation de la différence de marche :

$$i(t) = aI(vt) \tag{10}$$

où a est un facteur numérique qui dépend de la photodiode. On suppose que la conversion de l'intensité lumineuse en intensité électrique est linéaire. Le signal $u(t)$ reçu sur la sortie de l'amplificateur est donc la somme du signal utile $i(t)$ est de la variation due aux décharges cycliques dans la lampe. Le spectre du signal utile a une fréquence maximale d'environ 3 Hz (pour une radiation bleue). Il faut donc échantillonner le signal de manière à éviter le repliement des harmoniques de haute fréquence dans la bande utile $[0, 3]$ Hz. Il y a deux manières de procéder : effectuer un sur-échantillonnage ou bien appliquer un filtre passe-bas analogique avant l'échantillonnage.

2.c. Sur-échantillonnage et filtrage numérique

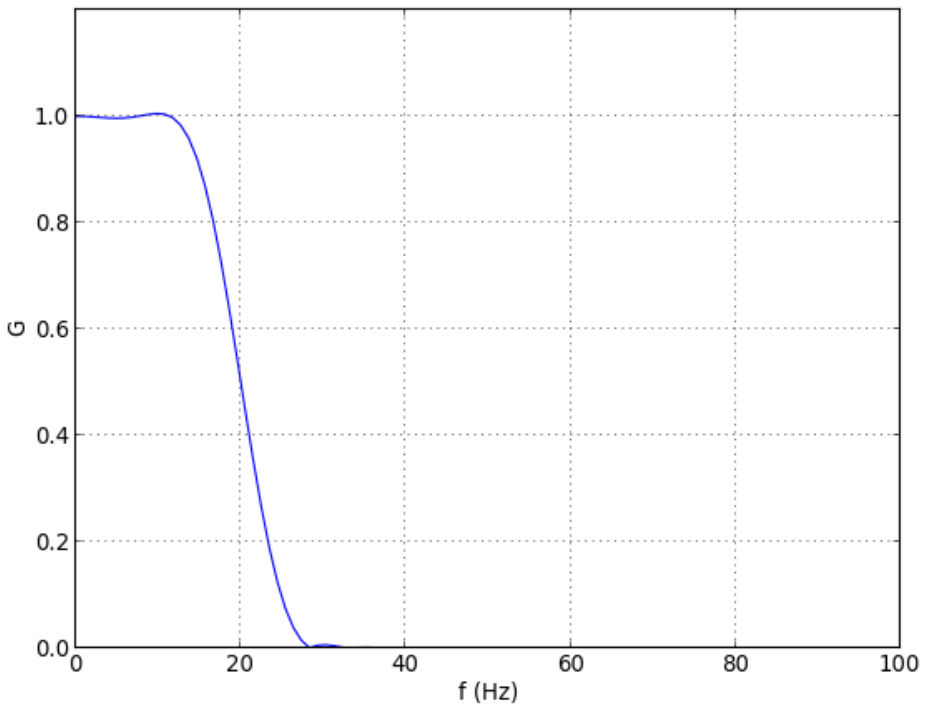
On échantillonne à une fréquence $f_e = 1$ kHz. Les harmoniques situées entre 500 et 1000 Hertz vont se replier dans la bande $[0, 500]$. Ces harmoniques sont faibles par rapport aux premières harmoniques du signal. Cependant, le signal utile $i(t)$ présente des variations plus faibles que les variations cycliques de l'éclairage, donc ce repliement

pourrait perturber le spectre dans la bande utile $[0,3]$ Hz. En choisissant une fréquence d'échantillonnage exactement égale à 1000 Hz, le seul repliement gênant se ferait sur la raie de fréquence nulle, qui de toute façon ne nous intéresse pas.

Pour une expérience de durée 1000 s, soit une variation de δ d'environ 1 mm, le nombre d'échantillons à acquérir est $N = 10^6$. Sachant que seules les basses fréquences (inférieures à 3 Hz) nous intéressent, on peut faire un filtrage passe-bas numérique avant de réduire la fréquence d'échantillonnage. Ce filtrage passe-bas nous permet d'obtenir le signal utile $i(t)$ et la réduction de la fréquence d'échantillonnage permet d'alléger les fichiers de stockage des données. Cependant, le filtrage n'est pas nécessaire pour l'obtention du spectre. Voici un filtre passe-bas RIF qui convient, avec une fréquence de coupure de 20 Hz :

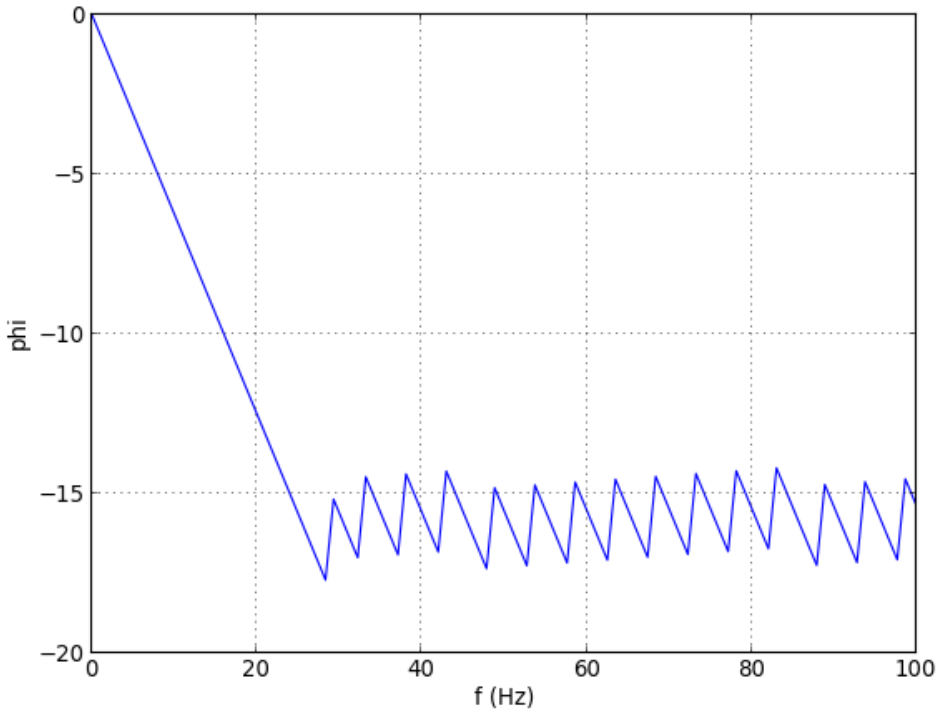
```
import scipy.signal

fe = 1000.0
fc = 20.0
P = 200
h = scipy.signal.firwin(numtaps=P,cutoff=[fc/fe],nyq=0.5,window='hann')
w,H = scipy.signal.freqz(h)
figure()
plot(w/(2*numpy.pi)*fe,numpy.absolute(H))
xlabel("f (Hz)")
ylabel("G")
grid()
axis([0,100,0,1.2])
```



Avec ce filtre, on a bien une très forte réduction des raies multiples de 50 Hz avec un gain constant dans la bande utile $[0, 3]$ Hz. Il faut aussi vérifier le déphasage :

```
figure()
plot(w/(2*numpy.pi)*fe,numpy.unwrap(numpy.angle(H)))
xlabel("f (Hz)")
ylabel("phi")
grid()
axis([0,100,-20,0])
```



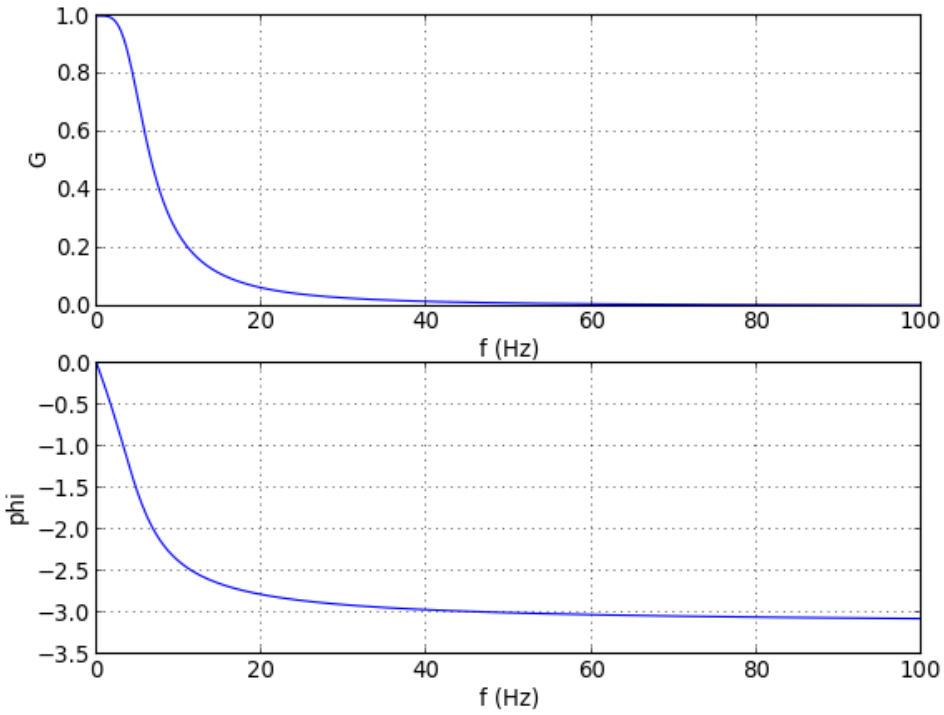
Le déphasage est parfaitement linéaire dans la bande passante, comme il se doit.

2.d. Filtre anti-repliement analogique

L'échantillonnage à une fréquence de 1000 Hz a pour inconvénient le grand nombre d'échantillons à manipuler, au moins avant le filtrage. Pour abaisser la fréquence d'échantillonnage, il faut utiliser un filtre passe-bas analogique. Le montage donné plus haut comporte un filtre passe-bas du second ordre (type Sallen-Key) dont la fréquence de coupure est de 5 Hz. Voici sa réponse fréquentielle :

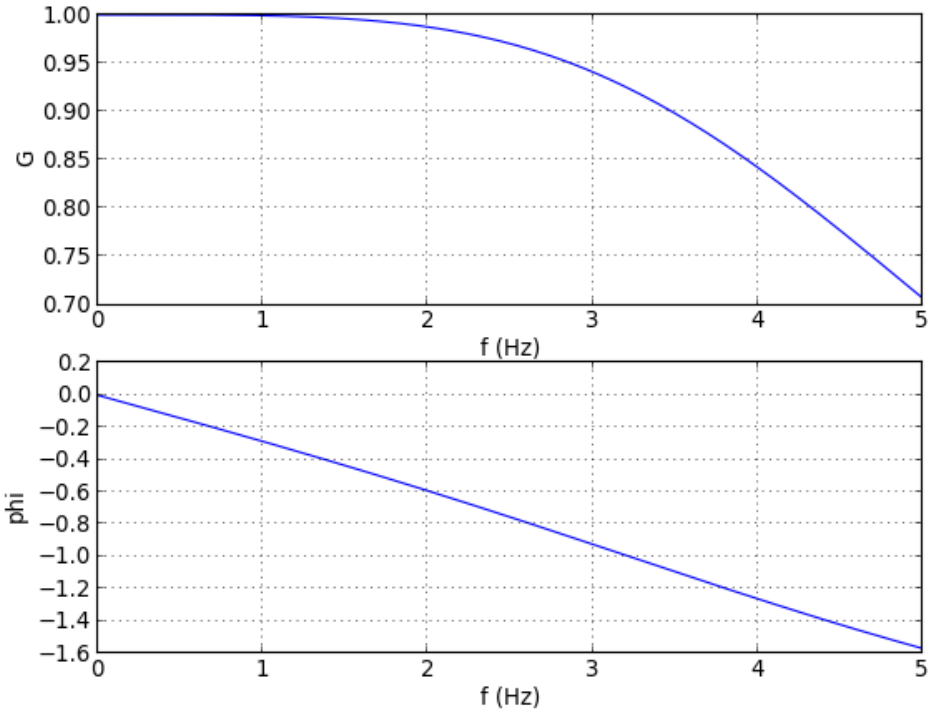
```
fc = 5.0
def H(f):
    return 1.0/(1.0+numpy.sqrt(2)*1j*f/fc-(f/fc)**2)
f = numpy.linspace(0,100,1000)
Ht = H(f)
figure()
subplot(211)
```

```
plot(f,numpy.absolute(Ht))
xlabel("f (Hz)")
ylabel("G")
grid()
subplot(212)
plot(f,numpy.angle(Ht))
xlabel("f (Hz)")
ylabel("phi")
grid()
```



Les raies multiples de 50 Hz sont bien éliminées. Voyons plus en détail la bande utile :

```
f = numpy.linspace(0,5,200)
Ht = H(f)
figure()
subplot(211)
plot(f,numpy.absolute(Ht))
xlabel("f (Hz)")
ylabel("G")
grid()
subplot(212)
plot(f,numpy.angle(Ht))
xlabel("f (Hz)")
ylabel("phi")
grid()
```



Dans la bande $[0,3]$ Hz qui nous intéresse, la phase est bien linéaire et le gain varie de moins de 5 pour cent. Cette variation n'est pas gênante ; elle se combine avec la réponse fréquentielle de la photodiode, pour donner de toute manière une faible sensibilité dans la partie bleue du spectre.

Avec ce filtre, on pourra abaisser la fréquence d'échantillonnage à 6 Hz (le double de la fréquence utile maximale). On gardera néanmoins une fréquence de 100 Hz, de manière à obtenir un tracé du signal $i(t)$. Pour l'analyse spectrale finale, une fréquence de 10 Hz serait largement suffisante.

3. Exemples

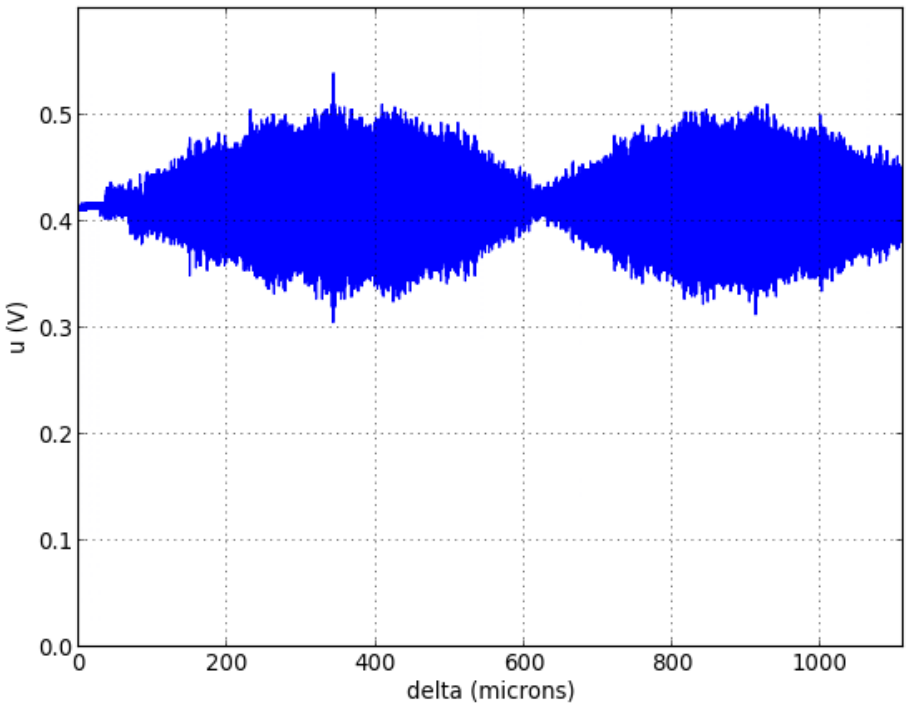
3.a. Lampe à décharge au sodium

Filtrage analogique

Voici l'enregistrement du signal en sortie du filtre passe-bas, avec une fréquence d'échantillonnage de 100 Hz, pour une durée totale de 1000 s :

```
data = numpy.loadtxt("Na-8.txt")
t = data[0]
u = data[1]
N = t.size
delta = t*10.0/9
figure()
plot(delta,u)
xlabel("delta (microns)")
ylabel("u (V)")
```

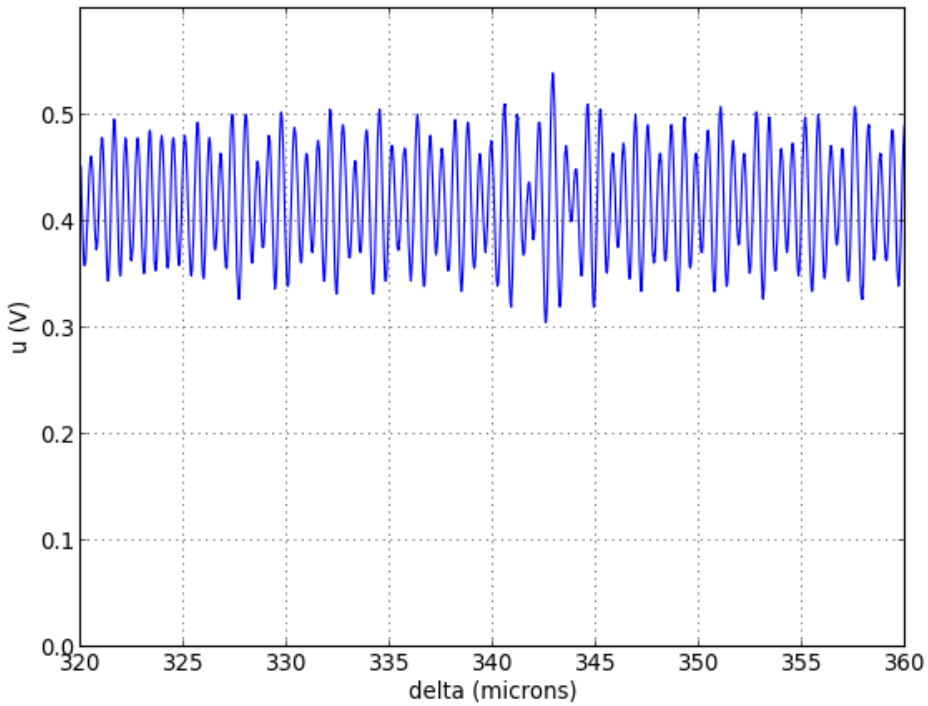
```
axis([0,delta.max(),0,0.6])
grid()
```



Le miroir passe par le contact optique au cours de l'expérience. Celui-ci est visible par la variation plus grande d'intensité vers 350 μm.

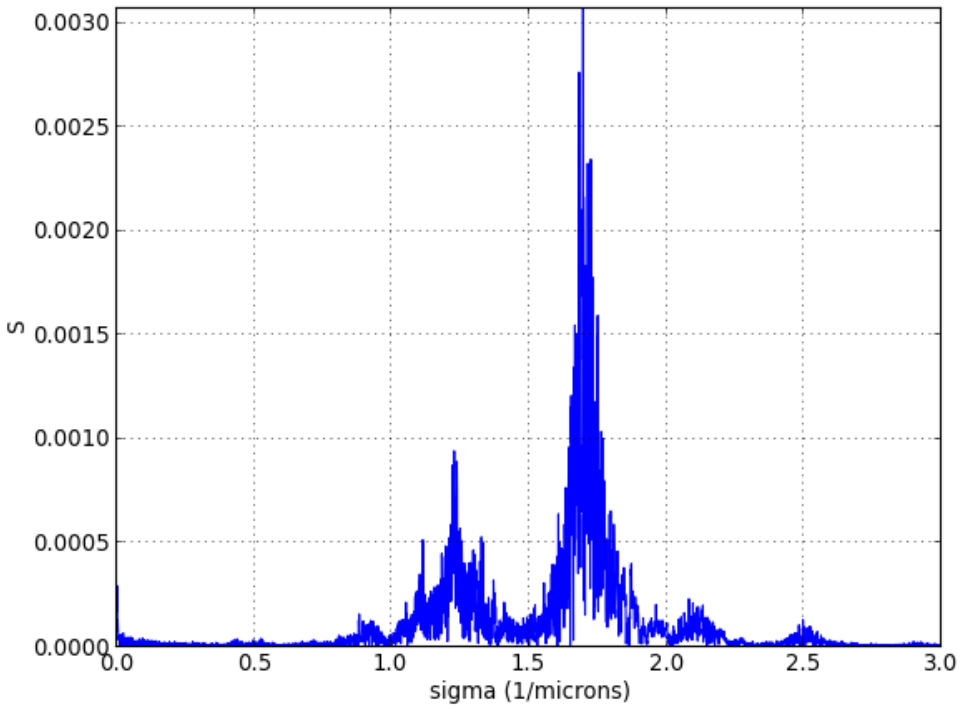
Voici un détail autour du contact optique

```
axis([320,360,0,0.6])
```



Voici l'analyse spectrale, avec un fenêtrage de Hamming :

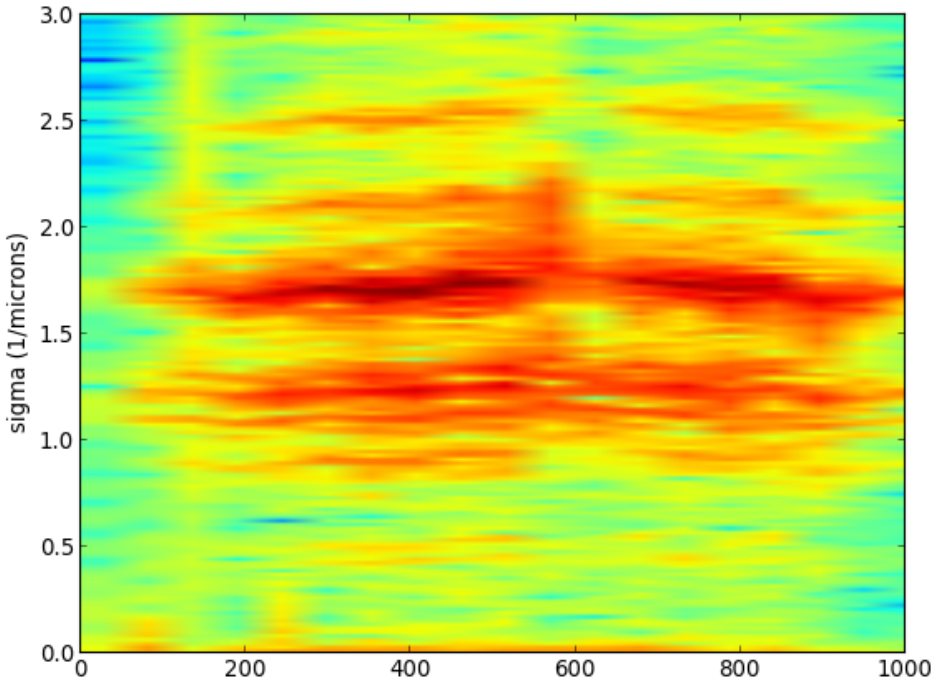
```
v = scipy.signal.hamming(N)*(u-u.mean())
spectre = numpy.absolute(numpy.fft.fft(v))/N
d = delta[1]-delta[0]
sigma = numpy.arange(N)*1.0/(N*d)
figure()
plot(sigma,spectre)
xlabel("sigma (1/microns)")
ylabel("S")
Smax = spectre.max()
axis([0,3,0,Smax])
grid()
```



Le spectre obtenu révèle la raie jaune à 590 nm et une raie infrarouge vers 800 nm.

Ces deux raies présentent une largeur bien trop grande, qui suggère un déplacement non régulier du chariot. Pour tester cela, on fait un spectrogramme, qui consiste à faire une analyse spectrale sur une fenêtre de taille réduite que l'on déplace. Considérons par exemple une fenêtre de 50 s, qui comporte 5000 échantillons. La résolution fréquentielle sur cette fenêtre est d'environ $0,02\ \mu m^{-1}$.

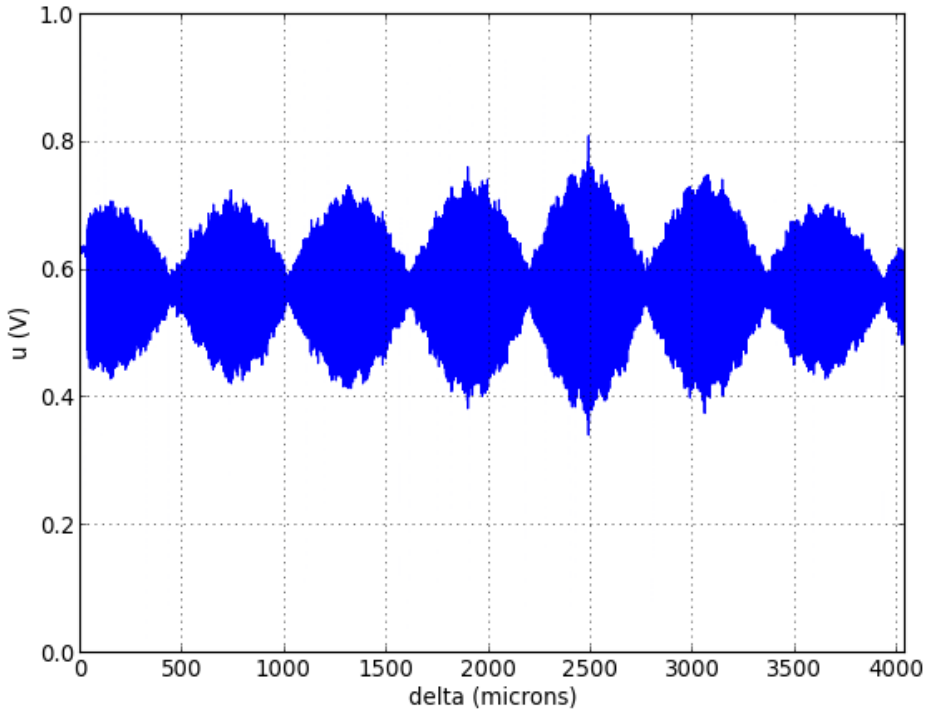
```
figure()
specgram(v,NFFT=5000,Fs=1.0/d,noverlap=0)
ylabel("sigma (1/microns)")
axis([0,1000,0,3])
```



Même à l'échelle de la minute, les deux raies ont une largeur bien plus grande que la résolution. La résolution est donc limitée par la régularité du moteur (ou par celle de la mécanique d'entraînement).

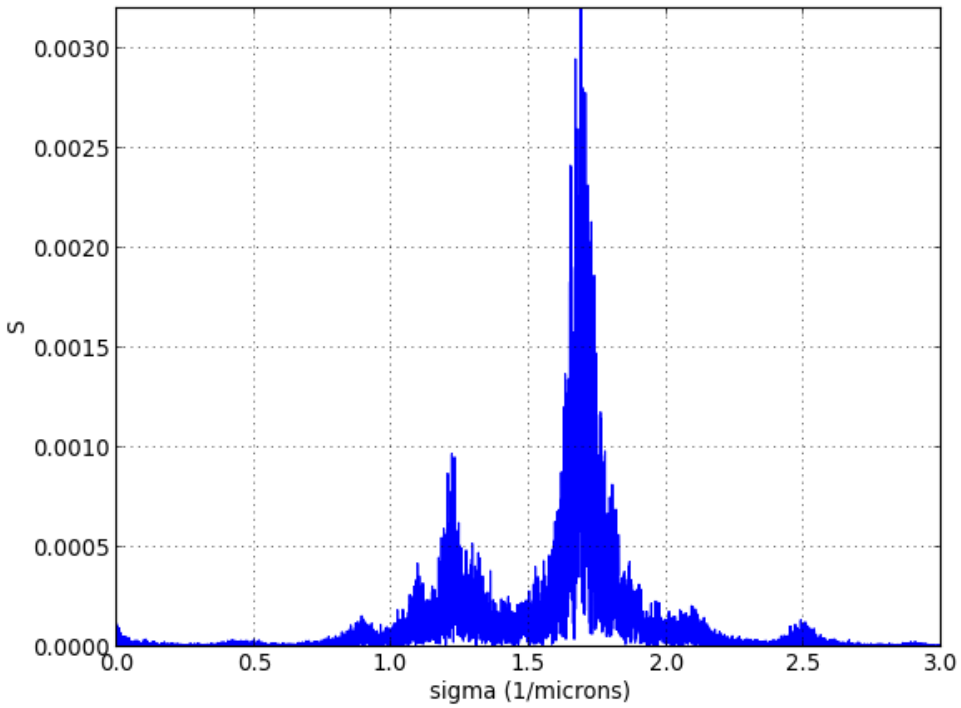
Voici un autre enregistrement sur une durée plus longue :

```
data = numpy.loadtxt("Na-9.txt")
t = data[0]
u = data[1]
N = t.size
delta = t*10.0/9
figure()
plot(delta,u)
xlabel("delta (microns)")
ylabel("u (V)")
axis([0,delta.max(),0,1.0])
grid()
```



Le contact optique est à 2500 μm. On voit bien les battements dûs au doublet du sodium.

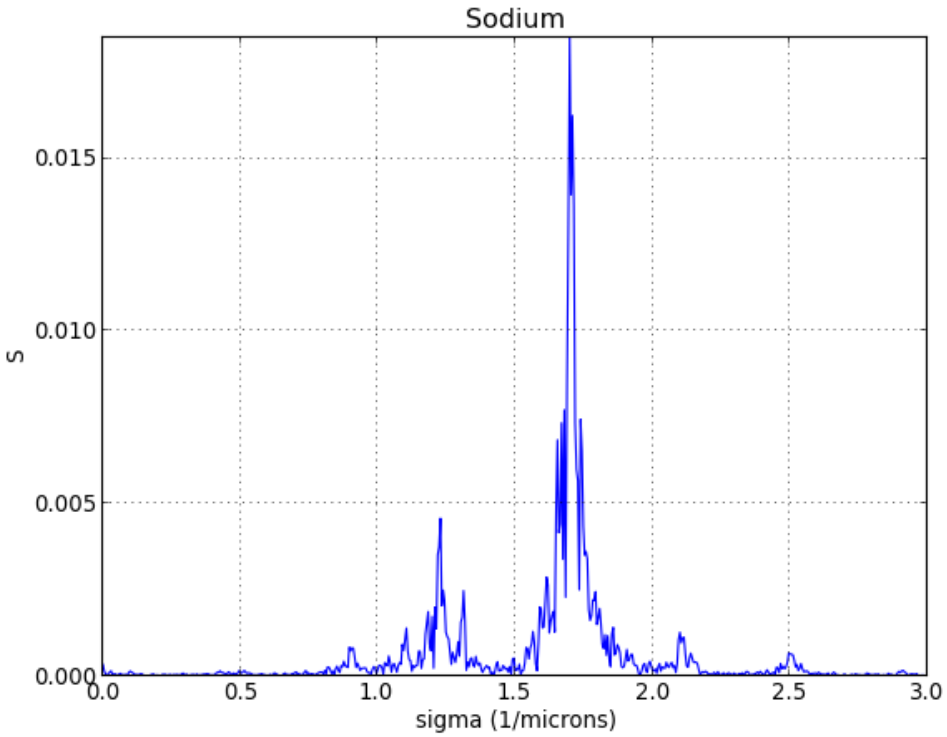
```
u=u-u.mean()
v = scipy.signal.hamming(N)*u
spectre = numpy.absolute(numpy.fft.fft(v))/N
d = delta[1]-delta[0]
sigma = numpy.arange(N)*1.0/(N*d)
figure()
plot(sigma,spectre)
xlabel("sigma (1/microns)")
ylabel("S")
Smax=spectre.max()
axis([0,3,0,Smax])
grid()
```



Le spectre n'est pas plus fin que précédemment, bien que la durée d'acquisition soit 4 fois plus grande. Cela confirme que la résolution est bien limitée par la régularité du moteur.

On peut d'ailleurs calculer le spectre sur une fenêtre autour du contact optique, de plus ou moins 100 μm :

```
N1 = int((2500.0-100.0)/4000*N)
N2 = int((2500.0+100.0)/4000*N)
v = scipy.signal.hamming(N2-N1)*u[N1:N2]
spectre = numpy.absolute(numpy.fft.fft(v))/(N2-N1)
sigma = numpy.arange(N2-N1)*1.0/((N2-N1)*d)
figure()
plot(sigma,spectre)
xlabel("sigma (1/microns)")
ylabel("S")
title("Sodium")
Smax = spectre.max()
axis([0,3,0,Smax])
grid()
```

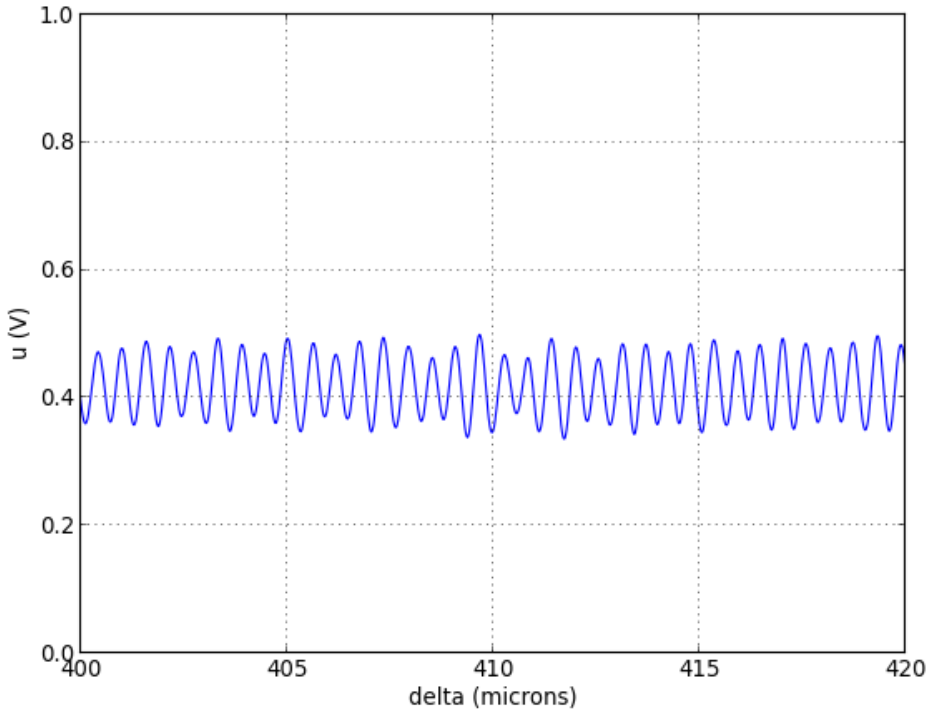


La définition des raies est meilleure sur cette durée plus courte.

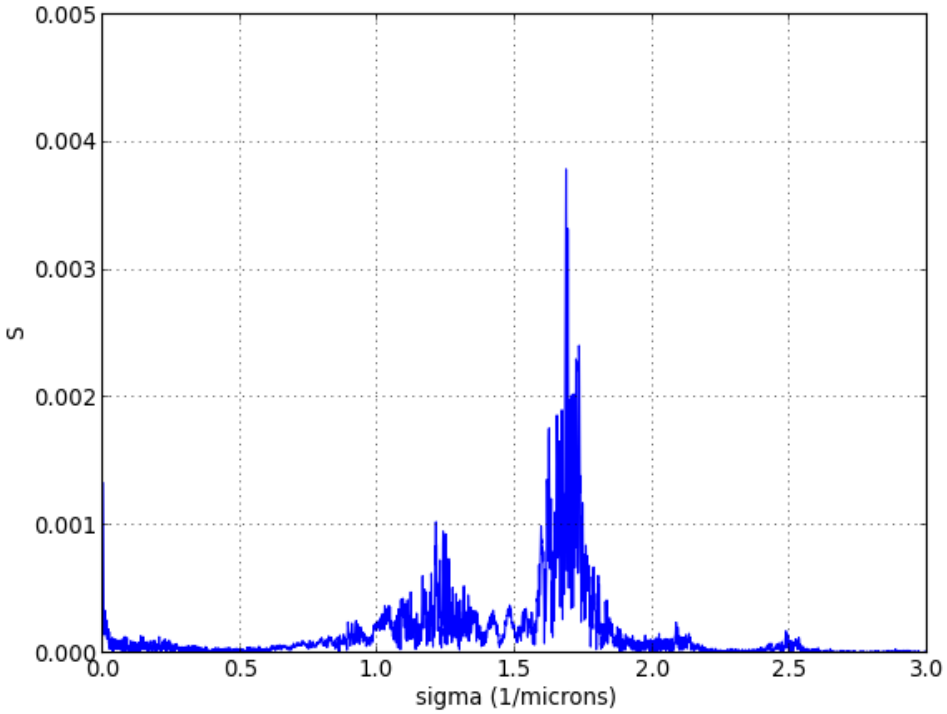
Sur-échantillonnage et filtrage numérique

Voici les résultats obtenus avec un sur-échantillonnage à 1 kHz suivi du filtrage numérique défini plus haut, et d'une réduction de la fréquence d'échantillonnage d'un facteur 20 :

```
data = numpy.loadtxt("Na-fast-7-filtre.txt")
t = data[0]
u = data[1]
N = t.size
delta = t*10.0/9
figure()
plot(delta,u)
xlabel("delta (microns)")
ylabel("u (V)")
axis([400,420,0,1.0])
grid()
```




```
delta = t*10.0/9
v = scipy.signal.hamming(N)*(u-u.mean())
spectre = numpy.absolute(numpy.fft.fft(v))/N
d = delta[1]-delta[0]
sigma = numpy.arange(N)*1.0/(N*d)
figure()
plot(sigma,spectre)
xlabel("sigma (1/microns)")
ylabel("S")
axis([0,3,0,0.005])
grid()
```

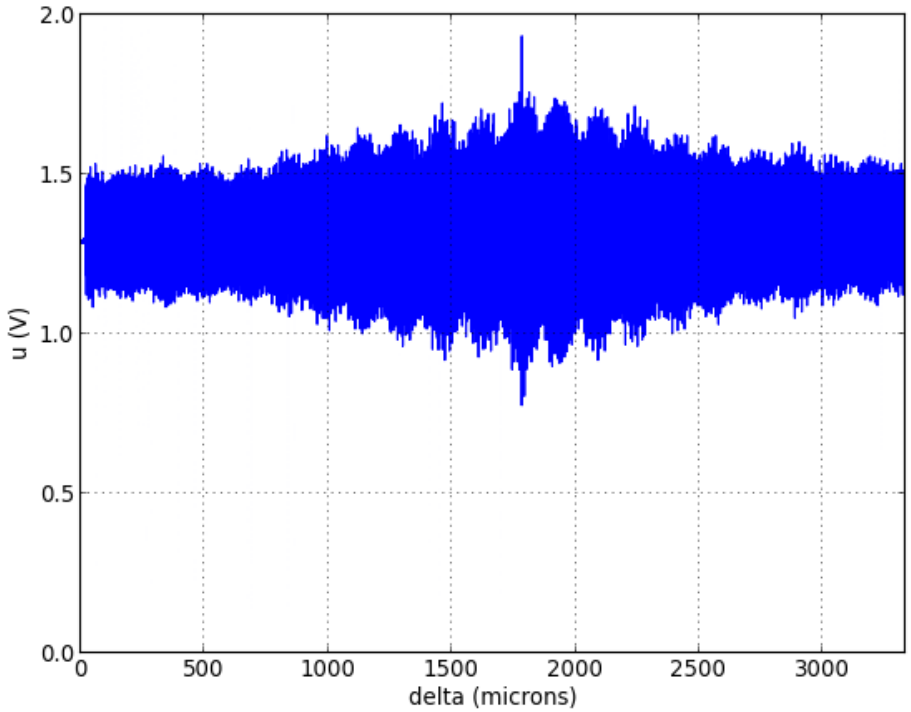


Les résultats sont similaires à ceux obtenus avec le filtre analogique.

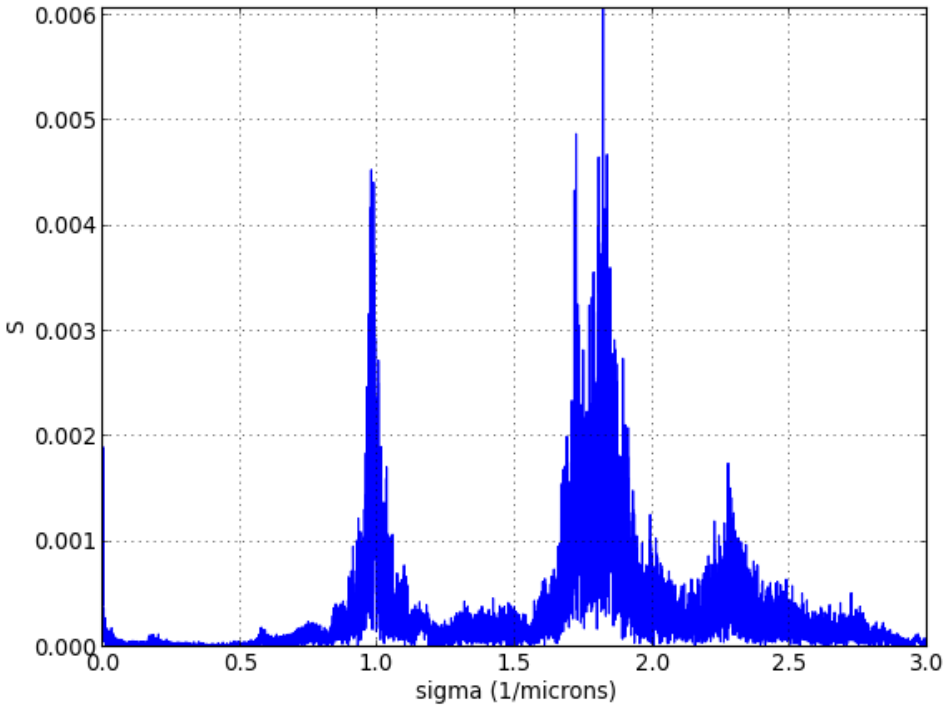
3.b. Lampe à décharge au mercure

Voici le résultat d'une acquisition d'environ 3000 s faite avec le filtre analogique et une fréquence d'échantillonnage de 100 Hz :

```
data = numpy.loadtxt("Hg-5.txt")
t = data[0]
u = data[1]
N = t.size
delta = t*10.0/9
figure()
plot(delta,u)
xlabel("delta (microns)")
ylabel("u (V)")
axis([0,delta.max(),0,2])
grid()
```

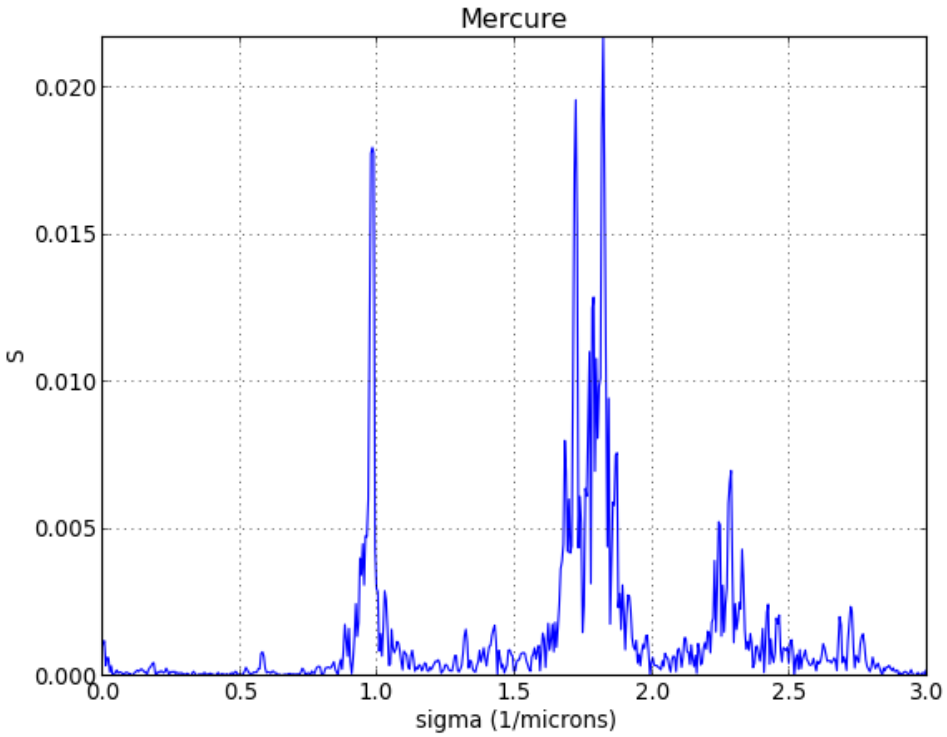



```
u = u-u.mean()
v = scipy.signal.hamming(N)*u
spectre = numpy.absolute(numpy.fft.fft(v))/N
d = delta[1]-delta[0]
sigma = numpy.arange(N)*1.0/(N*d)
figure()
plot(sigma,spectre)
xlabel("sigma (1/microns)")
ylabel("S")
Smax = spectre.max()
axis([0,3,0,Smax])
grid()
```



Là encore, on a intérêt à limiter le calcul du spectre à une fenêtre centrée sur le contact optique :

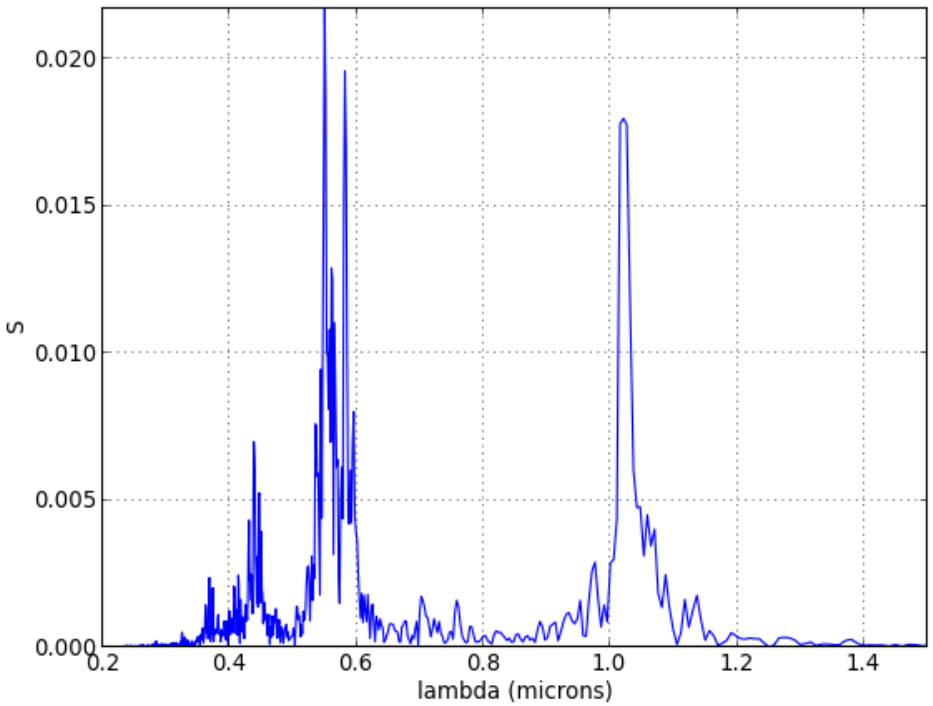
```
d_max = delta[N-1]
N1 = int((1780.0-100.0)/d_max*N)
N2 = int((1780.0+100)/d_max*N)
v = scipy.signal.hamming(N2-N1)*u[N1:N2]
spectre = numpy.absolute(numpy.fft.fft(v))/(N2-N1)
sigma = numpy.arange(N2-N1)*1.0/((N2-N1)*d)
figure()
plot(sigma,spectre)
xlabel("sigma (1/microns)")
ylabel("S")
title("Mercure")
Smax = spectre.max()
axis([0,3,0,Smax])
grid()
```



Pour cette fenêtre de largeur 200 μm, la résolution est de $0,005\ \mu m^{-1}$. Une augmentation de cette durée n'augmente pas la résolution à cause de l'irrégularité du moteur.

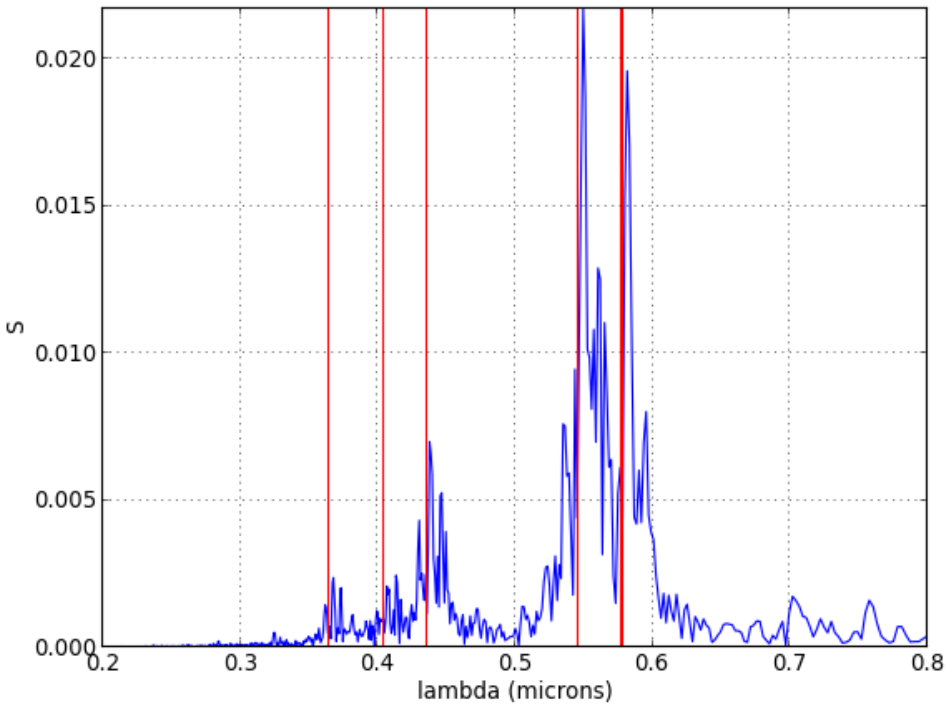
Pour peut aussi tracer le spectre en fonction de la longueur d'onde :

```
figure()
plot(1/sigma,spectre)
xlabel("lambda (microns)")
ylabel("S")
grid()
axis([0.2,1.5,0,Smax])
```



On repère les raies bien connues du mercure : une violette, une bleue, une verte et une jaune (le doublet est non résolu). On voit aussi une raie ultraviolette (très faible car la photodiode est peu sensible aux UV) et une raie infrarouge (là ou le capteur est le plus sensible). Plaçons les raies sur la figure :

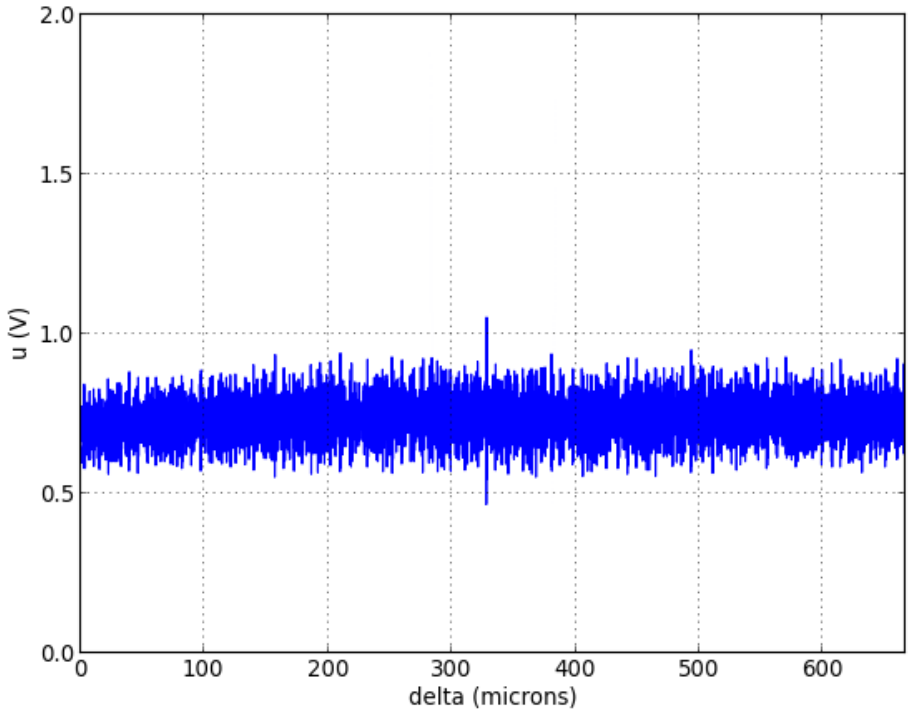
```
L = [0.365,0.405,0.436,0.546,0.577,0.579]
for l in L:
    plot([l,l],[0,Smax],'r')
axis([0.2,0.8,0,Smax])
```



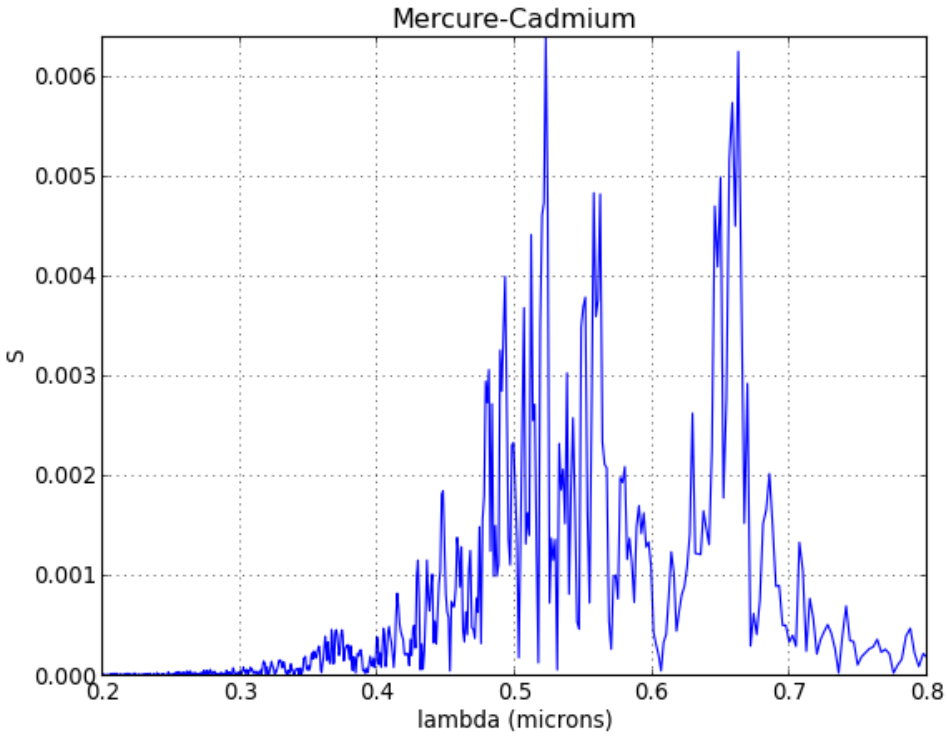
Les lampes à arc au mercure, très lumineuses, sont utilisées dans les microscopes.

3.c. Lampe à décharge mercure-cadmium

```
data = numpy.loadtxt("Hg-Cd-3.txt")
t = data[0]
u = data[1]
N = t.size
delta = t*10.0/9
figure()
plot(delta,u)
xlabel("delta (microns)")
ylabel("u (V)")
axis([0,delta.max(),0,2])
grid()
```



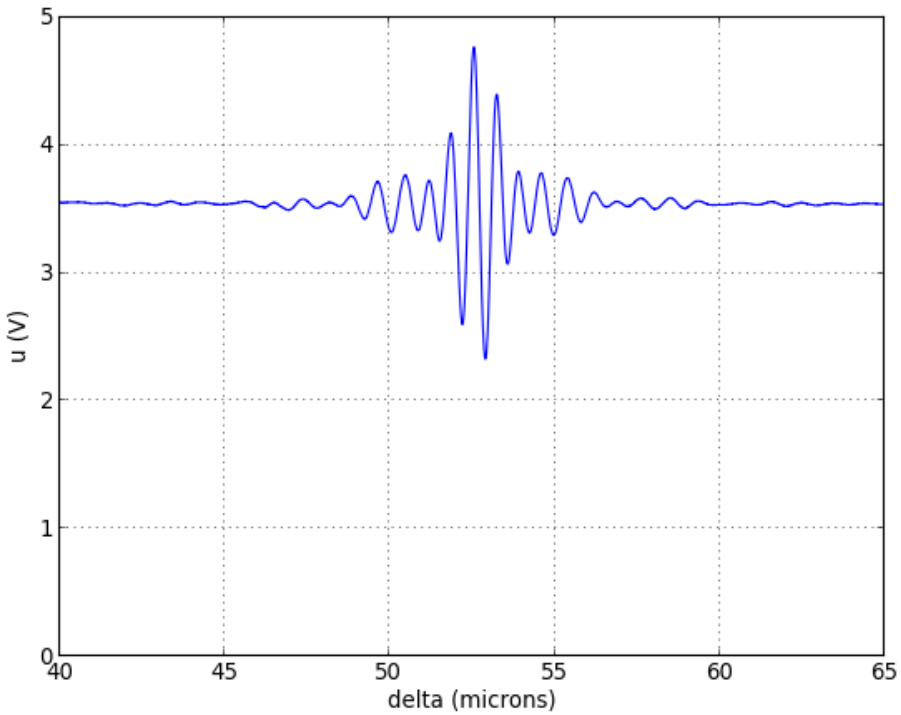

```
u=u-u.mean()
d_max = delta[N-1]
N1 = int((325.0-100.0)/d_max*N)
N2 = int((325.0+100)/d_max*N)
v = scipy.signal.hamming(N2-N1)*u[N1:N2]
spectre = numpy.absolute(numpy.fft.fft(v))/(N2-N1)
sigma = numpy.arange(N2-N1)*1.0/((N2-N1)*d)
figure()
plot(1.0/sigma,spectre)
xlabel("lambda (microns)")
ylabel("S")
title("Mercure-Cadmium")
Smax = spectre.max()
axis([0.2,0.8,0,Smax])
grid()
```



3.d. Lampe à incandescence

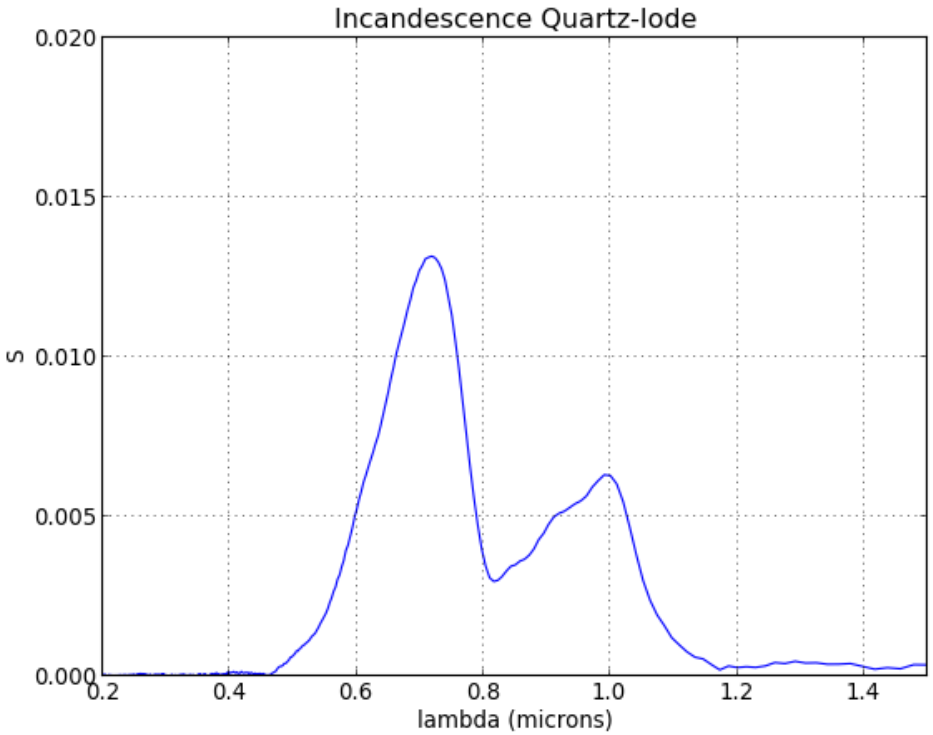
On utilise une lampe halogène de type quartz-iode. La résistance du convertisseur courant-tension est 1 $M\Omega$.

```
data = numpy.loadtxt("incandescence-QI-2.txt")
t = data[0]
u = data[1]
N = t.size
delta = t*10.0/9
figure()
plot(delta,u)
xlabel("delta (microns)")
ylabel("u (V)")
axis([40,65,0,5.0])
grid()
```



Voici le spectre :

```
u = u-u.mean()
spectre = numpy.absolute(numpy.fft.fft(u))/N
d = delta[1]-delta[0]
sigma = numpy.arange(N)*1.0/(N*d)
figure()
plot(1/sigma,spectre)
xlabel("lambda (microns)")
ylabel("S")
title("Incandescence Quartz-Iode")
axis([0.2,1.5,0,2e-2])
grid()
```



Il faut noter que ce spectre est le produit du spectre de la lumière émise par l'ampoule par le spectre de sensibilité de la photodiode, et par le spectre d'absorption des différents éléments optiques traversés par la lumière entre la lampe et le récepteur.

La décroissance au delà de 1 μm est due à la diode. Le minimum vers $0,8$ μm est dû soit à la lampe, soit à un élément traversé par la lumière (lentille en verre ou lame de l'interféromètre). La couche semi-réfléchissante de la lame séparatrice peut aussi avoir un effet sur le spectre.

4. Spectrométrie d'absorption

4.a. Principe

La spectrométrie d'absorption consiste à déterminer le spectre d'absorption d'un corps. Les spectres d'absorption dans le domaine infrarouge sont couramment obtenus avec des spectromètres à transformée de Fourier.

Pour obtenir un spectre d'absorption, il faut utiliser une source de lumière à large spectre. Le domaine de longueur d'onde analysé est aussi limité par la plage de réponse du capteur. Avec une lampe à incandecence et la photodiode utilisée ici, on peut accéder à l'intervalle de longueur d'onde entre $0,5$ μm et et $1,2$ μm.

Le corps absorbant, par exemple un liquide dans une cuve transparente, peut être placé soit devant la source de lumière, soit à la sortie de l'interféromètre, devant le capteur. Soit $S(\sigma)$ le spectre de référence, obtenu en l'absence du corps. Ce spectre résulte du spectre de la lampe, de la réponse spectrale du capteur, et de l'absorption des différents éléments traversés par la lumière. Si le corps étudié est une molécule en solution, on obtient ce spectre de référence avec la cuve remplie du solvant.

Soit $S_a(\sigma)$ le spectre obtenu avec le corps absorbant étudié. Le coefficient d'absorption A est défini par :

$$S_a(\sigma) = (1 - A(\sigma))S(\sigma) \tag{11}$$

4.b. Filtres colorés

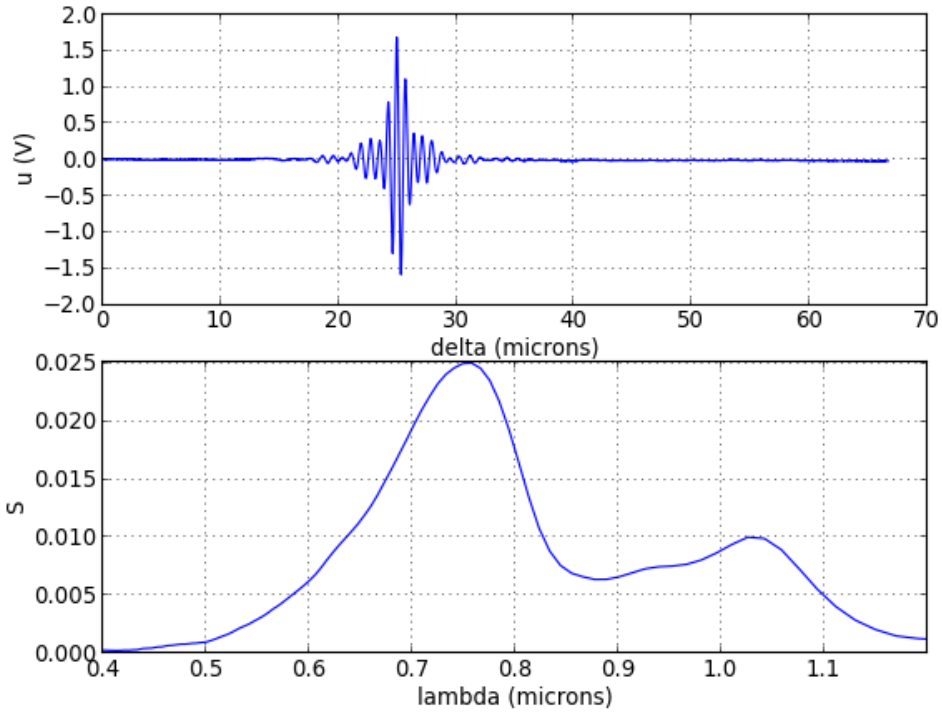
On cherche à obtenir le spectre d'absorption de filtres colorés. On enregistre l'interférogramme de référence puis l'interférogramme avec le filtre. Pour les filtres à large bande considérés ici, un déplacement du miroir de 20 μm de part et d'autre du contact optique est largement suffisant.

La fonction suivante trace l'interférogramme et calcule le spectre, pour un fichier donné. Elle renvoit le spectre.

```
def analyse(fichier):
    data = numpy.loadtxt(fichier)
    t = data[0]
    u = data[1]
    u = u-u.mean()
    N = t.size
    print(N)
    delta = t*10.0/9
    figure()
    subplot(211)
    plot(delta,u)
    xlabel("delta (microns)")
    ylabel("u (V)")
    grid()
    v = scipy.signal.hamming(N)*u
    spectre = numpy.absolute(numpy.fft.fft(v))/N
    d = delta[1]-delta[0]
    sigma = numpy.arange(N)*1.0/(N*d)
    subplot(212)
    plot(1/sigma,spectre)
    xlabel("lambda (microns)")
    ylabel("S")
    Smax = spectre.max()
    axis([0.4,1.2,0,Smax])
    grid()
    return (sigma,spectre)
```

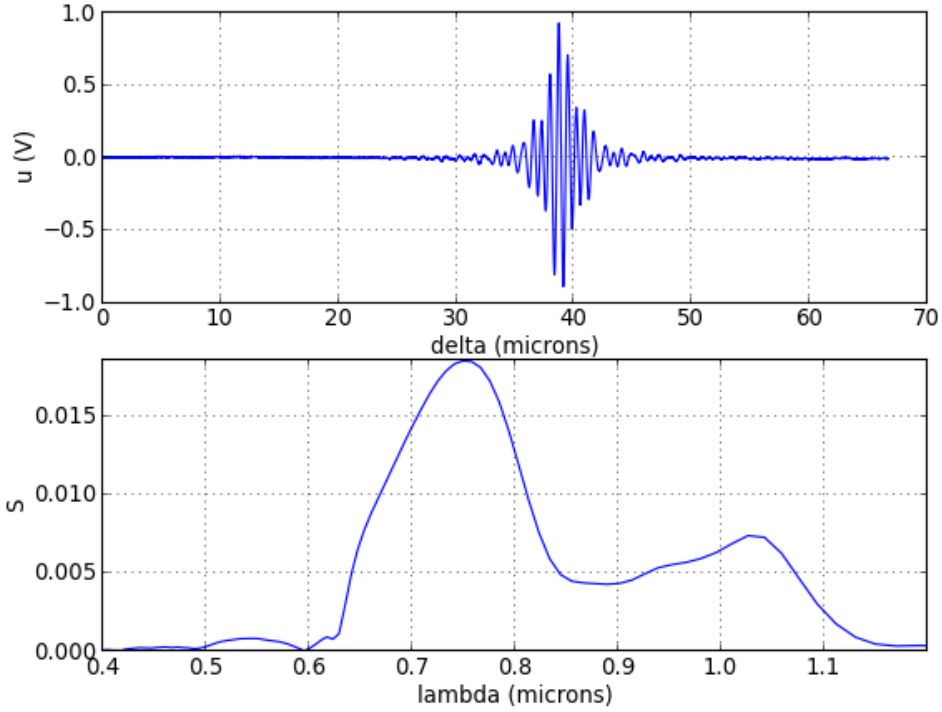
Voici tout d'abord le spectre de référence :

```
(sigma,S) = analyse("incandescence-QI-4.txt")
```



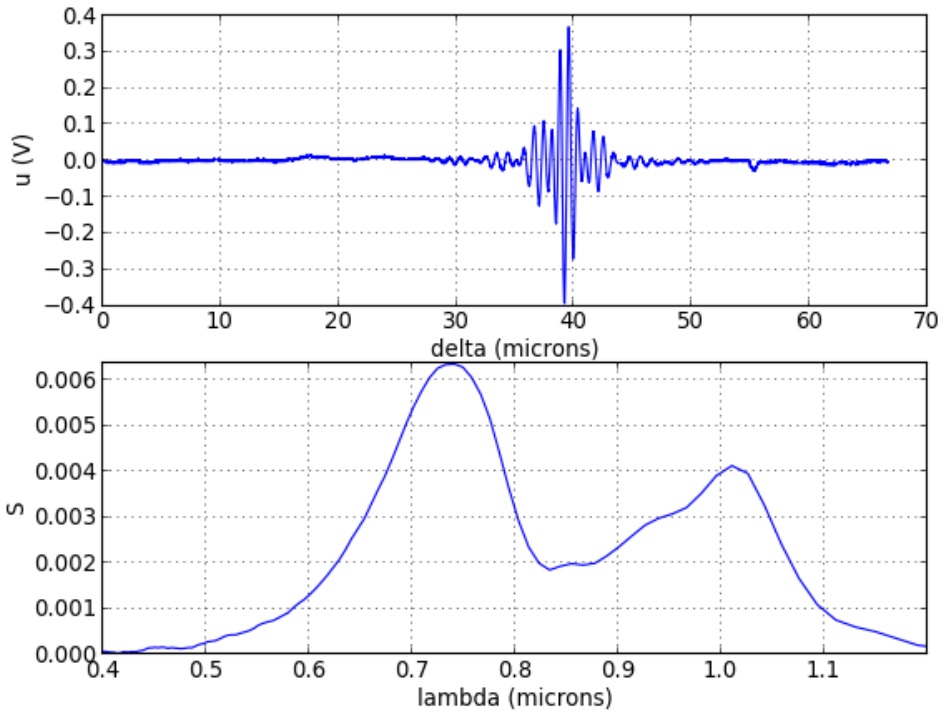
Voici le spectre obtenu avec un filtre rouge :

```
(sigma_rouge,S_rouge) = analyse("incandescence-QI-filtreRouge-2.txt")
```



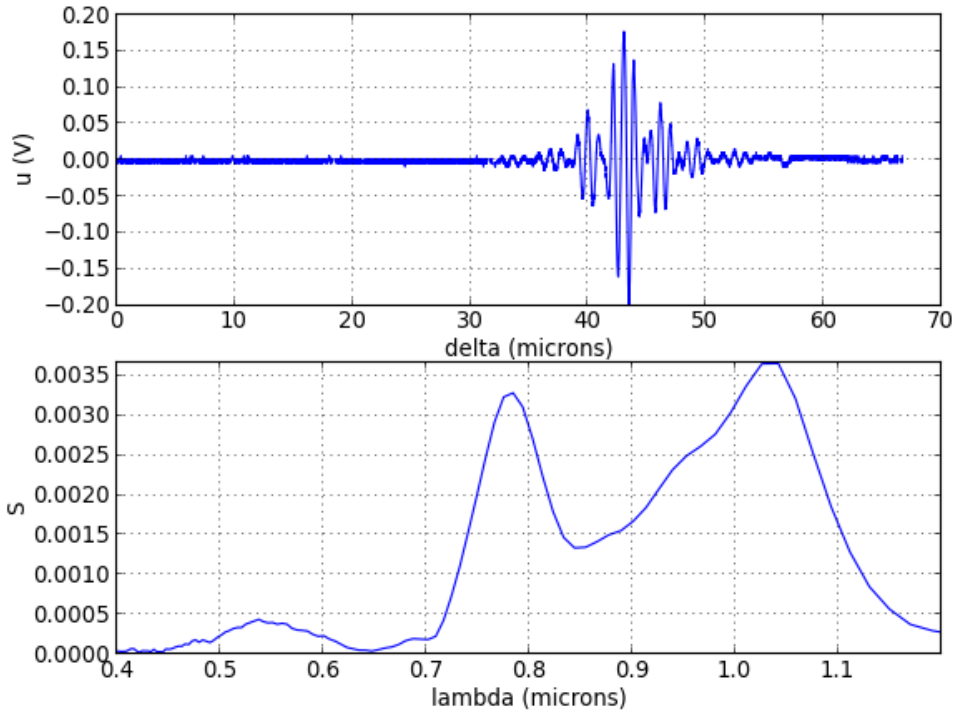
Voici le spectre obtenu avec un filtre jaune :

```
(sigma_jaune,S_jaune) = analyse("incandescence-QI-filtreJaune-1.txt")
```

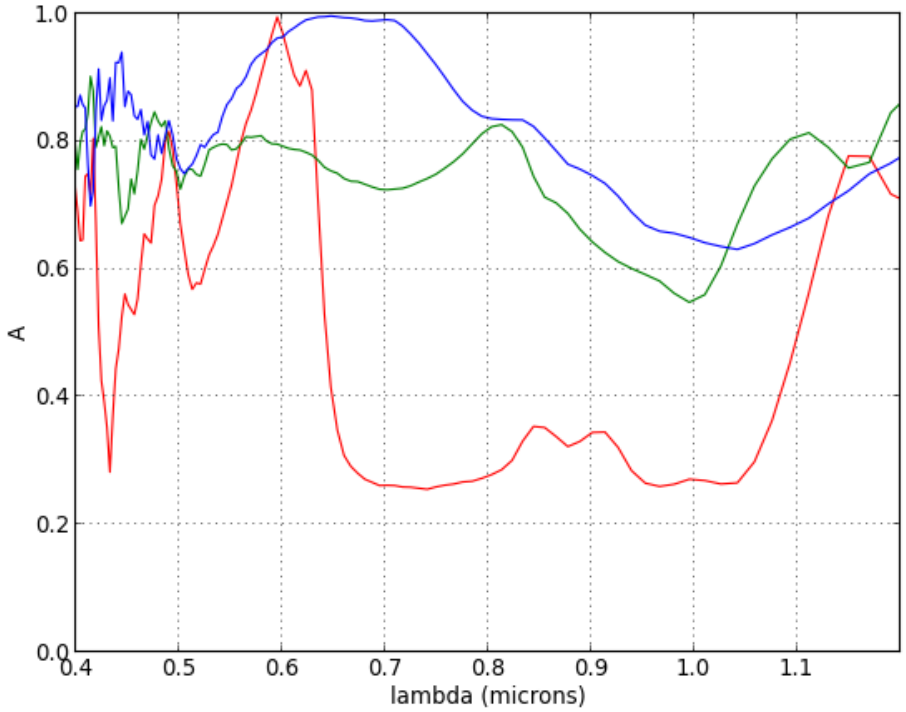


Le spectre pour un filtre cyan :

```
(sigma_cyan,S_cyan) = analyse("incandescence-QI-filtreCyan-1.txt")
```



```
A_rouge = 1.0-S_rouge/S
A_jaune = 1.0-S_jaune/S
A_cyan = 1.0-S_cyan/S
figure()
plot(1/sigma_rouge,A_rouge,'r')
plot(1/sigma_jaune,A_jaune,'g')
plot(1/sigma_cyan,A_cyan,'b')
xlabel("lambda (microns)")
ylabel("A")
grid()
axis([0.4,1.2,0,1])
```



Ces spectres ne sont pas exploitables en dessous de $0,5\ \mu$, à cause de la trop faible sensibilité de la diode dans ce domaine.

5. Annexe : programmes d'acquisition

Lorsque le nombre de points à acquérir est inférieur à $2^{18} = 262144$, on peut utiliser le logiciel LatisPro.

Le script python ci-dessous nécessite la version 4 du module pyCAN. Le nombre de points est illimité. Il trace le signal dans une fenêtre (par défaut sur une durée de 10 s). Lorsqu'on ferme la fenêtre, les données sont enregistrées dans un fichier texte `data.txt`, qu'il faut renommer à la fin de l'expérience. Pour la lecture de ce fichier, voir les codes python décrits plus haut.

acquisition.py

```
# -*- coding: utf-8 -*-

import pycan.main as pycan
import math
from matplotlib.pyplot import *
import matplotlib.animation as animation
import numpy
import scipy.signal

sys=pycan.Sysam("SP5")
Umax = 1 # valeur maximale de la tension, à changer en fonction de l'intensité de la
sys.config_entrees([0],[Umax])
fe=100.0 # fréquence d'échantillonnage
te=1.0/fe
N = 100000 # nombre d'échantillons total à acquérir
print(u"Durée de l'acquisition = %f s"%(N*te))
duree = 10.0 # durée des blocs
longueur_bloc = int(duree/te) # taille des blocs traités
nombre_blocs = int(N*te/duree)
nombre_echant = nombre_blocs*longueur_bloc

sys.config_echantillon_permanent(te*1e6,N)

n_tot = 0 # nombre d'échantillons acquis
n_bloc = 0 # nombre d'échantillons acquis dans un bloc

fig,ax = subplots()
t = numpy.arange(longueur_bloc)*te
u = numpy.zeros(longueur_bloc)

line0, = ax.plot(t,u)
ax.grid()
ax.axis([0,duree,-Umax,Umax])
ax.set_xlabel("t (s)")
u = numpy.array([],dtype=numpy.float32)

def animate(i):
    global sys,u,line0,n_tot,n_bloc,longueur_bloc
    data = sys.paquet(n_tot)
    u0 = data[1]
    n_tot += u0.size
    n_bloc += u0.size
    u = numpy.append(u,u0)
    if n_tot >= nombre_echant:
        print(u"Acquisition terminée")
    if n_bloc > longueur_bloc: # si on a assez d'échantillons pour constituer un bloc
        u = u[0:longueur_bloc]
        n_bloc = 0
```

```
        line0.set_ydata(u)
        u = numpy.array([],dtype=numpy.float32)

sys.lancer_permanent()

ani = animation.FuncAnimation(fig,animate,frames=nombre_blocs,repeat=False,interval=d
show()
data = sys.paquet(0)
numpy.savetxt("data.txt",data)
sys.fermer()
```

Le script suivant fait l'acquisition avec un filtrage numérique et applique un facteur de réduction de la fréquence d'échantillonnage :

acquisitionFiltrageReduction.py

```
# -*- coding: utf-8 -*-

import pycan.main as pycan
import math
from matplotlib.pyplot import *
import matplotlib.animation as animation
import numpy
import scipy.signal

sys=pycan.Sysam("SP5")
Umax = 1 # valeur maximale de la tension, à changer en fonction de l'intensité de la
sys.config_entrees([0],[Umax])
fe=1000.0 # fréquence d'échantillonnage
te=1.0/fe
N = 1000000 # nombre d'échantillons total à acquérir
print(u"Durée de l'acquisition = %f s"%(N*te))
duree = 10.0 # durée des blocs
reduction = 10 # réduction de la fréquence d'échantillonnage après le filtrage
fe_r = fe/reduction
te_r = te*reduction
longueur_bloc = int(duree/te_r) # taille des blocs traités
nombre_blocs = int(N*te/duree)
nombre_echant = nombre_blocs*longueur_bloc



#filtre passe-bas pour enlever les variations 50 Hz
fc = 20.0
P = 200
b = scipy.signal.firwin(numtaps=P,cutoff=[fc/fe],nyq=0.5,window='hann')
sys.config_filtre([1],b)
```

```
sys.config_echantillon_permanent(te*1e6,N)

n_tot = 0 # nombre d'échantillons acquis
n_bloc = 0 # nombre d'échantillons acquis dans un bloc

fig,ax = subplots()
t = numpy.arange(longueur_bloc)*te_r
u = numpy.zeros(longueur_bloc)

line0, = ax.plot(t,u)
ax.grid()
ax.axis([0,duree,-Umax,Umax])
ax.set_xlabel("t (s)")
u = numpy.array([],dtype=numpy.float32)

def animate(i):
    global sys,u,line0,n_tot,n_bloc,longueur_bloc
    data = sys.paquet_filtrees(n_tot,reduction)
    u0 = data[1]
    n_tot += u0.size
    n_bloc += u0.size
    u = numpy.append(u,u0)
    if n_tot >= nombre_echant:
        print(u"Acquisition terminée")
    if n_bloc > longueur_bloc: # si on a assez d'échantillons pour constituer un bloc
        u = u[0:longueur_bloc]
        n_bloc = 0
        line0.set_ydata(u)
        u = numpy.array([],dtype=numpy.float32)

sys.lancer_permanent()

ani = animation.FuncAnimation(fig,animate,frames=nombre_blocs,repeat=False,interval=d
show()
data = sys.paquet_filtrees(0,reduction)
numpy.savetxt("data.txt",data)
sys.fermer()
```