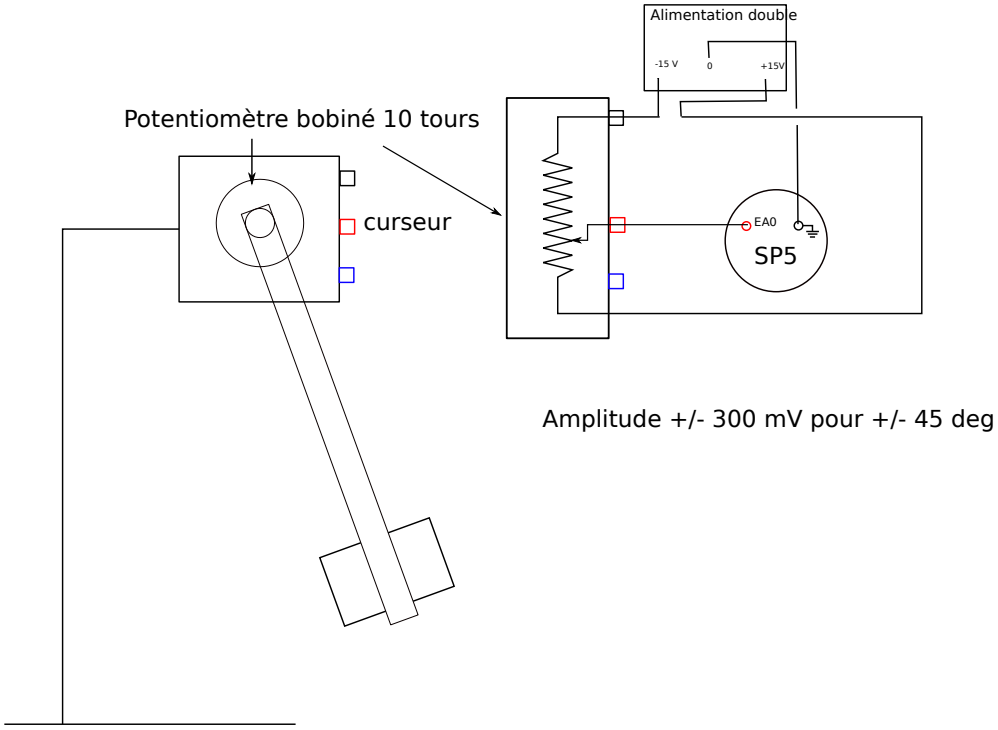
Étude expérimentale d'un pendule pesant

1. Dispositif expérimental

La liaison pivot du pendule pesant est réalisée avec un potentiomètre bobiné multi-tours de 10 $k\Omega$. On choisit pour cela un modèle qui présente très peu de frottement.



Une alimentation double fournit une tension de 30 V aux bornes du potentiomètre. La position angulaire du pendule est alors proportionnelle à la tension $u(t)$ entre le curseur du potentiomètre et la masse de l'alimentation. Lorsque le pendule est au repos, le curseur est d'abord placé approximativement à mi-course. Le support du potentiomètre est tourné de manière à obtenir une tension u à l'équilibre inférieure à 10 mV. L'échelle de tension est de 300 mV pour 45 deg.

L'acquision de la tension $u(t)$ est faite avec la centrale de conversion Eurosmart SysamSP5, pilotée par l'interface python présentée dans CAN Eurosmart : interface python. L'alimentation double $-12/0/12$ V de la centrale est d'ailleurs utilisée pour alimenter le potentiomètre.

2. Expérience et acquisition du signal

Le programme python ci-dessous effectue une acquisition de la tension $u(t)$ avec une période d'échantillonnage de 10 ms pendant 70 , durée qui permet de saisir l'oscillation

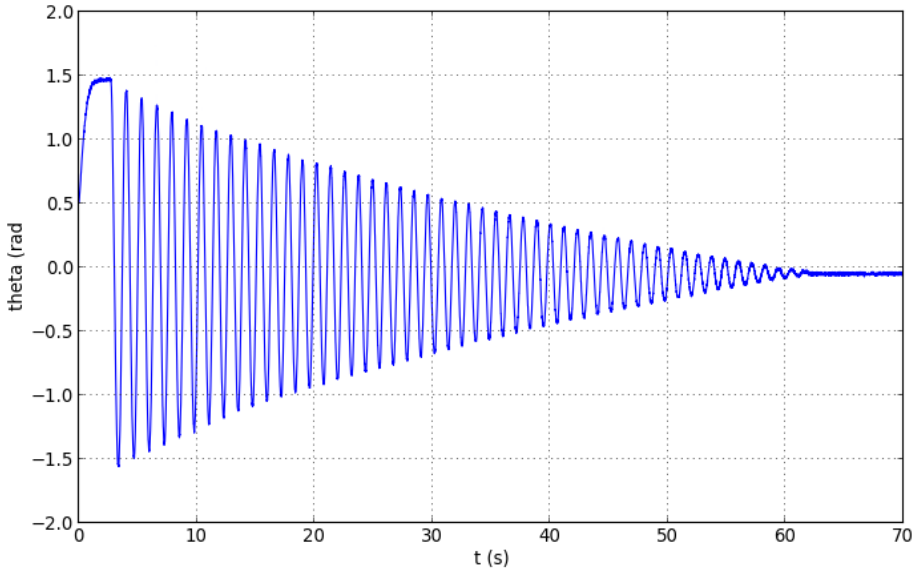
amortie complète. À la fin de l'acquisition, les données (temps et angle) sont sauvegardées dans un fichier texte puis l'angle est tracé en fonction du temps.

Lorsque l'acquisition est lancée le pendule est à l'équilibre ; on l'écarte de cette position puis on le lâche.

```
import pycan.main as pycan
import matplotlib.pyplot as plt

import numpy
import math


sys = pycan.Sysam("SP5")
sys.config_entrees([0],[1])
te=1e-2 # période d'échantillonnage
ne=7000 # nombre d'échantillons
duree=te*ne
sys.config_echantillon(te*10**6,ne)
sys.acquerir()
t=sys.temps()
u=sys.entrees()
echelle = 0.6/(math.pi/2) # conversion tension->angle
angle=u[0]/echelle
temps=t[0]
numpy.savetxt("pendule-6.txt",[temps,angle])
sys.fermer()
plt.figure(figsize=(18,6))

plt.plot(temps,angle)
plt.xlabel("t (s)")
plt.ylabel("theta (rad)")
plt.axis([0,duree,-math.pi/2,math.pi/2])
plt.show()
```

Voici la lecture du fichier texte et le tracé de l'angle en fonction du temps :

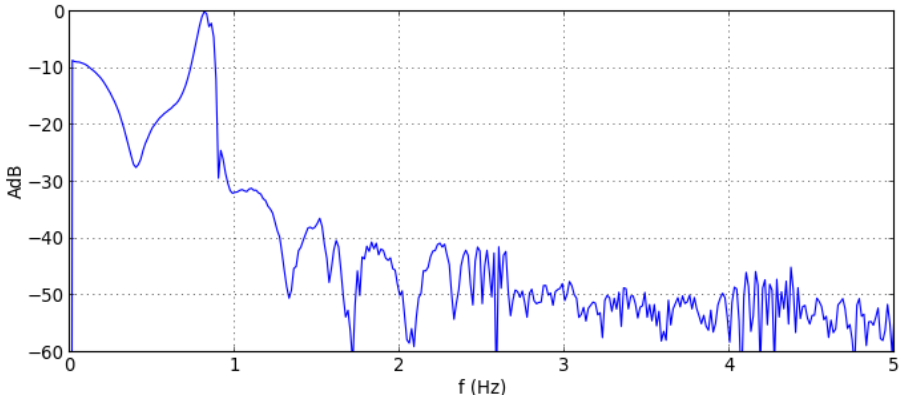
```
from matplotlib.pyplot import *
import numpy as np
import math
import scipy.signal

[temps,angle] = np.loadtxt("pendule-6.txt")
figure(figsize=(10,6))
plot(temps,angle)
xlabel("t (s)")
ylabel("theta (rad")
axis([0,70,-2,2])
grid()
```



3. Analyse spectrale

L'analyse spectrale d'un signal numérique est obtenue avec sa transformée de Fourier discrète :

```
from numpy.fft import fft

tfd = fft(angle)
N=angle.size
te=temps[1]-temps[0]
fe = 1.0/te
T = N*te
freq = np.zeros(N)
for k in range(N):
    freq[k] = 1.0/T*k
spectre = np.absolute(tfd)
spectre=spectre/spectre.max()
spectre_db = 20*np.log10(spectre)
figure(figsize=(10,4))
plot(freq,spectre_db,'b')
xlabel('f (Hz)')
ylabel('AdB')
axis([0,5,-60,0])
grid()
```



Ce spectre permet d'obtenir la fréquence des oscillations $(0,8\ Hz)$.

4. Calcul de la vitesse angulaire

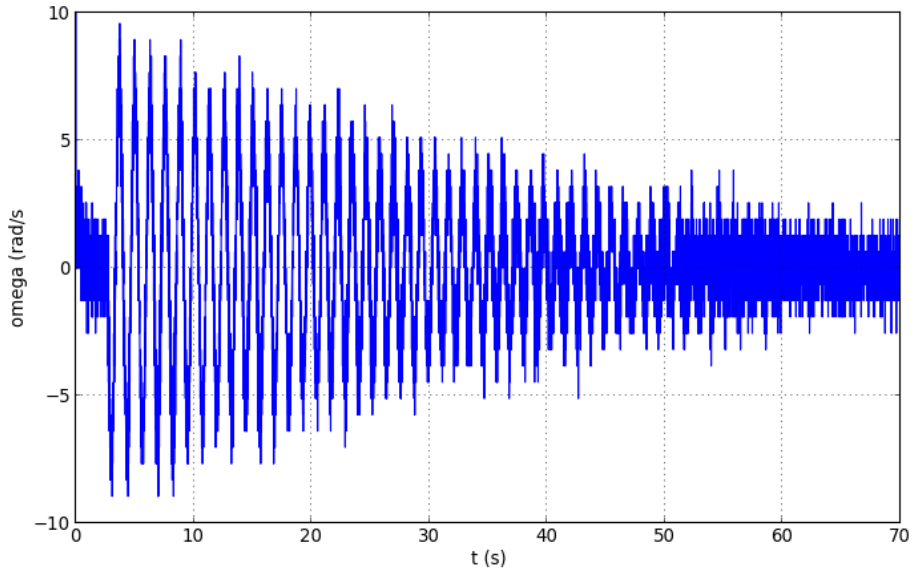
La dérivation d'un signal échantillonné se fait avec le filtre suivant :

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

où T_e est la période d'échantillonnage.

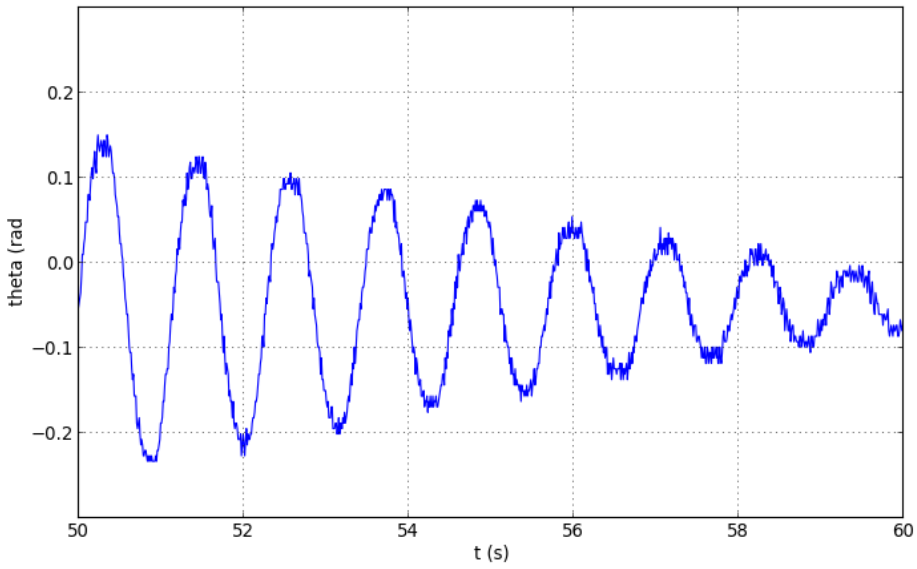
Voyons le résultat obtenu avec le signal précédent. Le filtre dérivateur est défini par ses coefficients comme expliqué dans la page Filtres intégrateur et dérivateur.

```
a=[te]
b=[1,-1]
omega = scipy.signal.lfilter(b,a,angle)
figure(figsize=(10,6))
plot(temps,omega)
xlabel("t (s)")
ylabel("omega (rad/s")
axis([0,70,-10,10])
grid()
```



Le signal obtenu est très bruité. Cela est dû au fait que le filtre dérivateur amplifie le bruit du signal de départ. Ce bruit est visible lorsqu'on observe en détail le signal :

```
figure(figsize=(10,6))
plot(temps,angle)
xlabel("t (s)")
ylabel("theta (rad")
axis([50,60,-0.3,0.3])
grid()
```



Il faut donc effectuer un filtrage passe-bas sur le signal de l'angle avant de calculer la dérivée. On choisit pour ce faire un filtre à réponse impulsionnelle finie. Ce type de filtre

passe-bas fonctionne très bien lorsque la fréquence d'échantillonnage (ici 100 Hz) est grande devant la fréquence utile du signal (ici de l'ordre de 1 Hz).

On commence par choisir le rang P de troncature de la réponse impulsionnelle ($2P+1$ est le nombre de coefficients du filtre). Plus P est grand, plus le filtre est sélectif mais on perd P points au début et P points à la fin du signal. Dans le cas présent, cette perte de points est sans importance. On choisit ensuite la fréquence de coupure relativement à la fréquence d'échantillonnage, la plus petite possible pour enlever le maximum de bruit sans altérer la forme du signal utile (faire des essais) :

```
P=20
fc=0.05
```

On calcule la réponse impulsionnelle du filtre passe-bas, avec une fenêtre de Hamming, puis on effectue le filtrage par convolution :

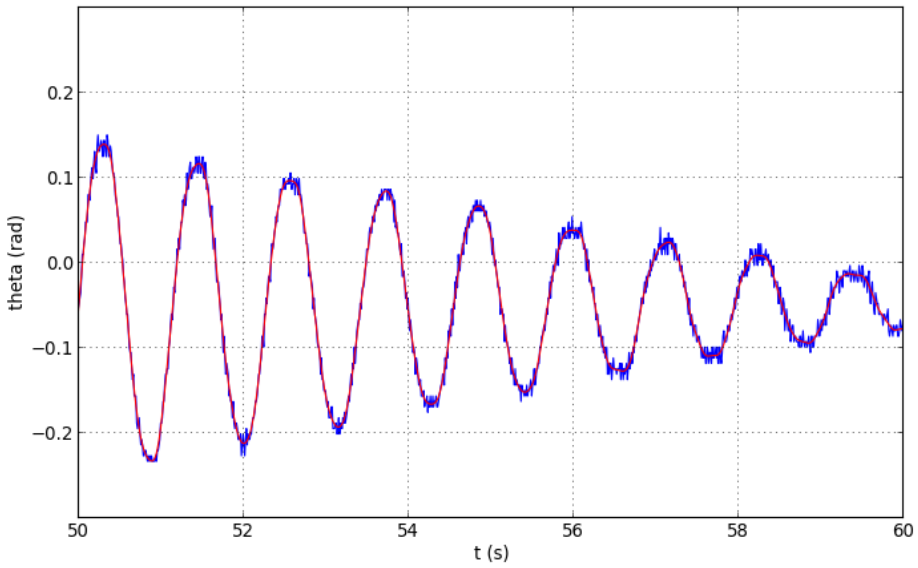
```
h = scipy.signal.firwin(numtaps=2*P+1,cutoff=[fc],nyq=0.5,window='hamming')
y = scipy.signal.convolve(angle,h,mode='valid')
```

En raison de la perte de points, il faut calculer une nouvelle liste de temps avant d'effectuer le tracé. Afin d'annuler le décalage entre le signal d'entrée et de sortie du filtre, cette liste débute à l'instant PT_e, comme expliqué dans la page filtre à réponse impulsionnelle finie.

```
ny = y.size
ty = np.zeros(ny)
for k in range(ny):
    ty[k] = P*te+te*k
```

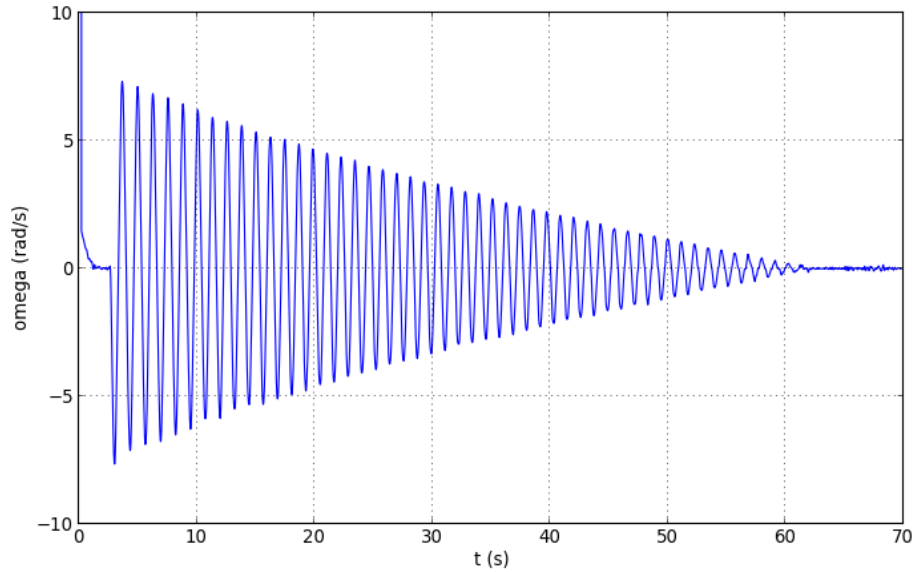
Voici le tracé du signal filtré avec le signal d'origine (vue de détail) :

```
figure(figsize=(10,6))
plot(temps,angle,"b")
plot(ty,y,"r")
xlabel("t (s)")
ylabel("theta (rad)")
axis([50,60,-0.3,0.3])
grid()
```



Nous pouvons à présent calculer la vitesse angulaire à partir du signal filtré :

```
a=[te]
b=[1,-1]
omega = scipy.signal.lfilter(b,a,y)
figure(figsize=(10,6))
plot(ty,omega)
xlabel("t (s)")
ylabel("omega (rad/s)")
axis([0,70,-10,10])
grid()
```



5. Portrait de phase

Pour tracer le portrait de phase, on enlève les points du début (avant le lâché du pendule) et on divise la vitesse angulaire par 2π :

```
debut = int(1.0/te)
y = np.delete(y,range(debut))
ty = np.delete(ty,range(debut))
omega = np.delete(omega,range(debut))/(2*math.pi)
figure(figsize=(6,6))
plot(y,omega)
axis([-math.pi/2,math.pi/2,-math.pi/2,math.pi/2])
xlabel("theta")
ylabel("omega")
grid()
```