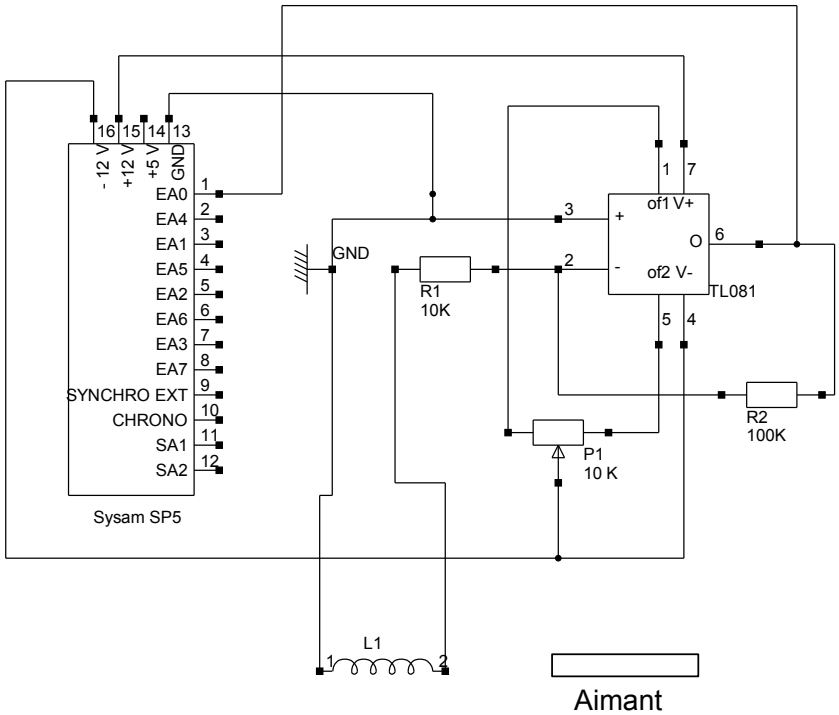
Mesure d'un flux magnétique

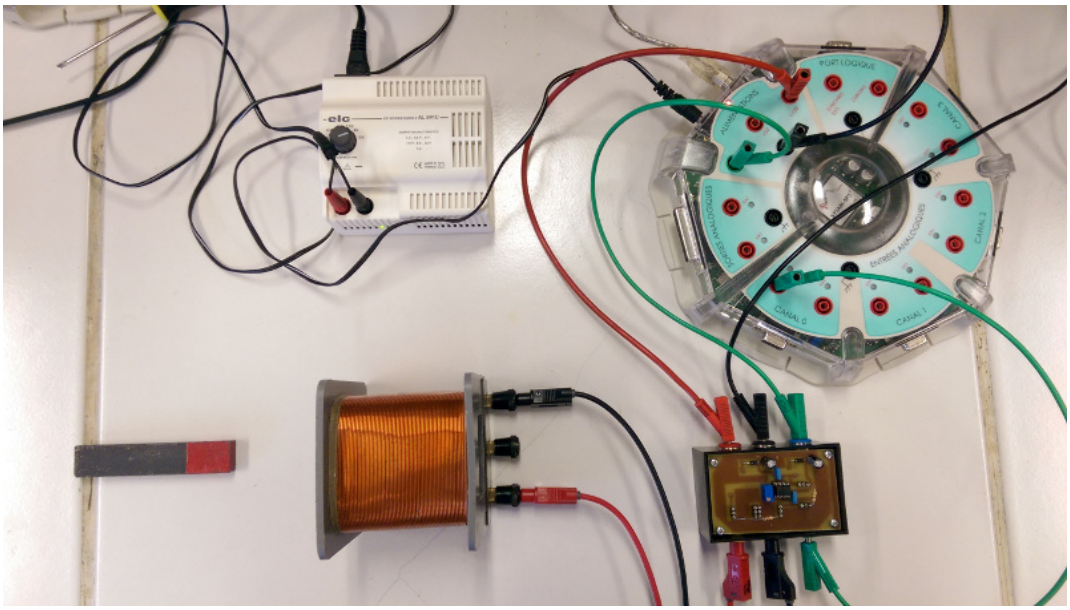
1. Introduction

Ce document montre comment mesurer un flux magnétique à travers une bobine en intégrant la force électromotrice. Celle-ci est numérisée, puis filtrée et intégrée numériquement.

2. Dispositif expérimental

Une bobine comportant 500 spires est branchée sur l'entrée d'un amplificateur inverseur de gain G=10. L'amplificateur est alimenté par l'alimentation -12V/12V disponible sur le boitier de la carte SysamSP5. Celle-ci est alimentée avec une alimentation linéaire 12 V, et non pas l'alimentation à découpage d'origine. L'amplificateur comporte un potentiomètre qui permet d'ajuster le décalage de la tension en sortie. La sortie de l'amplificateur est branchée sur l'entrée EA0 de la carte SysamSP5.





Pour la mesure du flux magnétique, l'aimant est déplacé à partir d'une position lointaine jusqu'à sa position finale. Le force électromotrice (le courant induit dans la résistance R1) est enregistrée au cours du temps.

3. Traitement du signal

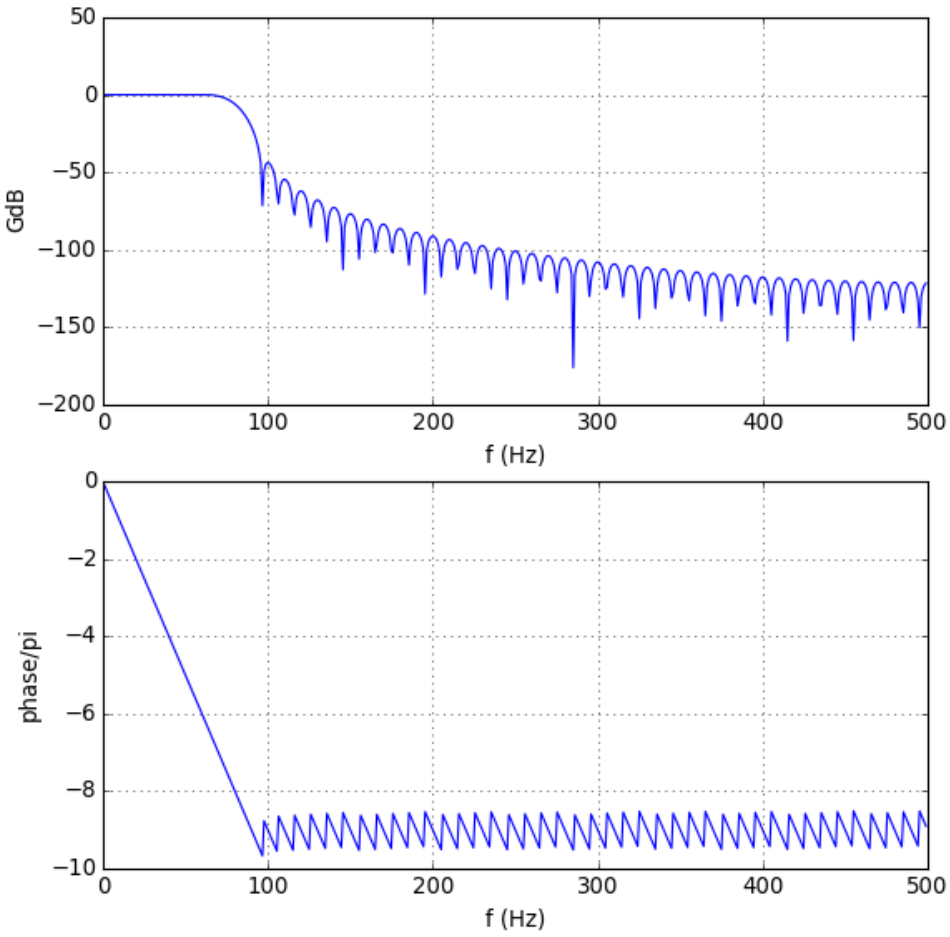
3.a. Numérisation

L'aimant étant déplacé à la main, la force électromotrice varie lentement. On procède néanmoins à un sur-échantillonnage à 1 kHz pour éviter le repliement dû au bruit. Juste après la numérisation, un filtrage passe-bas anti-repliement est appliqué avant une réduction de la fréquence d'échantillonnage d'un facteur 5, qui amène la fréquence finale à 200 Hz.

Le filtre anti-repliement est un filtre RIF à phase linéaire à 101 coefficients, dont la fréquence de coupure est légèrement inférieure à la moitié de la fréquence d'échantillonnage finale. Voici le calcul de ce filtre et sa réponse fréquentielle :

```
import numpy
from matplotlib.pyplot import *
import scipy.signal

fe = 1000.0
reduction = 5
fe_finale = fe/reduction
fc = 0.8*fe_finale*0.5
b = scipy.signal.firwin(numtaps=101,cutoff=[fc/fe],nyq=0.5,window='hann')

w,h = scipy.signal.freqz(b,[1.0])
figure(figsize=(8,8))
subplot(211)
plot(w/(2*numpy.pi)*fe,20*numpy.log10(numpy.absolute(h)))
xlabel("f (Hz)")
```

```
ylabel("GdB")
grid()
subplot(212)
plot(w/(2*numpy.pi)*fe,numpy.unwrap(numpy.angle(h))/numpy.pi)
xlabel("f (Hz)")
ylabel("phase/pi")
grid()
```



Les fréquences supérieures à la fréquence de Nyquist finale (100 Hz) sont atténuées à plus de 40 décibels.

3.b. Filtre intégrateur

La force électromotrice d'induction est donnée par la loi de Faraday :

$$e(t) = -\frac{d\Phi(t)}{dt} \tag{1}$$

où $\Phi(t)$ est le flux magnétique créé par l'aimant dans la bobine. La variation du flux est causée par le déplacement de l'aimant (on peut aussi déplacer la bobine).

En introduisant le gain $G = 10$ de l'amplificateur inverseur, la tension numérisée est :

$$u(t) = G\frac{d\Phi(t)}{dt} \tag{2}$$

Notons x_n le signal numérique issu de l'échantillonnage de $u(t)$, à la fréquence d'échantillonnage f'_e après réduction. L'intégration numérique est définie par la relation de récurrence suivante :

$$y_n = y_{n-1} + b(x_n + x_{n-1}) \tag{3}$$

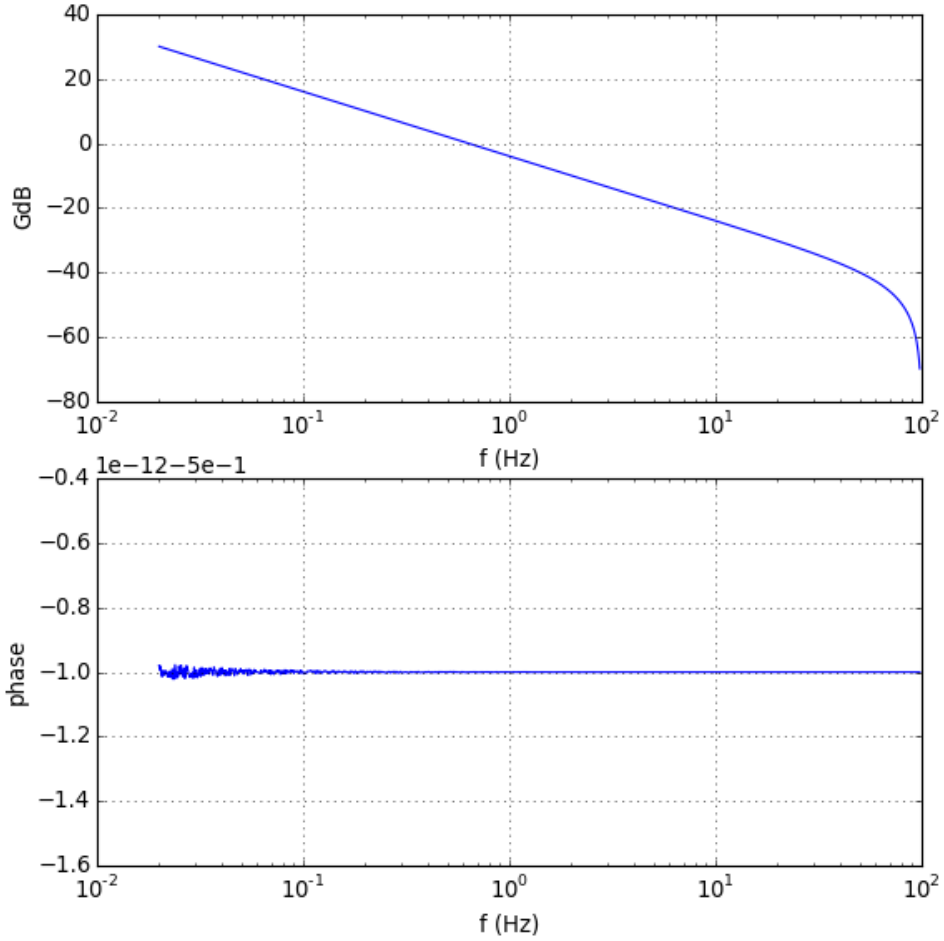
où b est un coefficient que l'on ajuste en fonction du gain souhaité. Pour obtenir des valeurs correspondant précisément à la dérivée par rapport au temps, il faut poser :

$$b = \frac{T'_e}{2} \tag{4}$$

Néanmoins, d'autres valeurs peuvent être utilisées pour avoir un signal de sortie dans la même gamme que le signal d'entrée. Dans ce cas, il faudra appliquer un facteur correctif pour obtenir le flux magnétique (en tenant compte aussi du gain G).

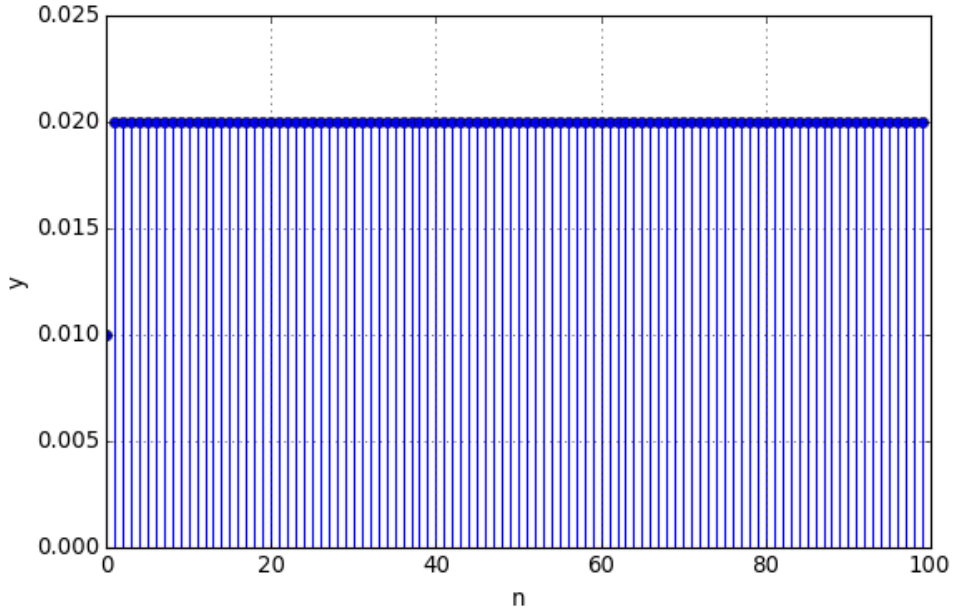
Voici la définition de ce filtre intégrateur et sa réponse fréquentielle :

```
b1 = [0.01,0.01]
a1 = [1.0,-1.0]
w,h = scipy.signal.freqz(b1,a1,worN=numpy.logspace(-4,-0.31,1000)*2*numpy.pi)
figure(figsize=(8,8))
subplot(211)
plot(w/(2*numpy.pi)*fe_finale,20*numpy.log10(numpy.absolute(h)))
xlabel("f (Hz)")
ylabel("GdB")
xscale("log")
grid()
subplot(212)
plot(w/(2*numpy.pi)*fe_finale,numpy.angle(h)/numpy.pi)
xlabel("f (Hz)")
ylabel("phase")
xscale("log")
grid()
```



Il s'agit d'un intégrateur parfait, avec un déphasage égal à $-\pi/2$ sur toute la gamme de fréquence. L'inconvénient d'un intégrateur parfait est son instabilité. En effet, un terme constant (de fréquence nulle) dans le signal conduit à une dérive de la tension en sortie, en raison du gain infini à fréquence nulle. On peut aussi calculer la réponse impulsionnelle de ce filtre :

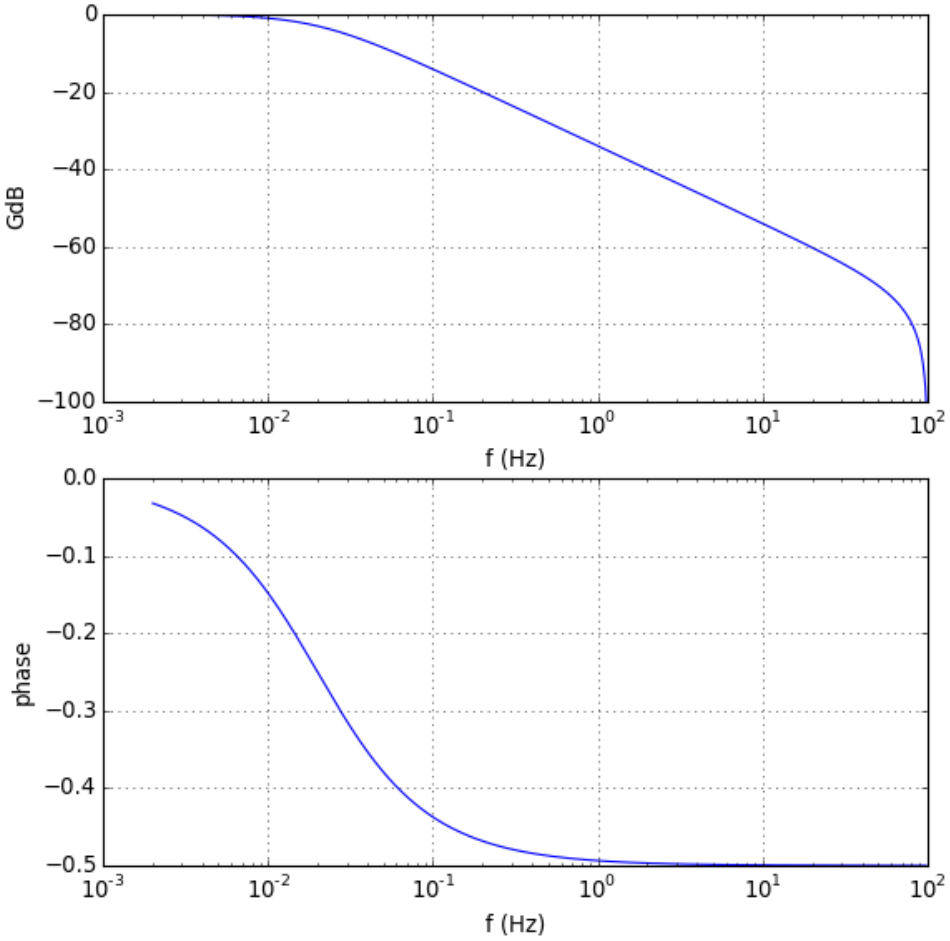
```
x = numpy.zeros(100)
x[0] = 1.0
y = scipy.signal.lfilter(b1,a1,x)
figure(figsize=(8,5))
stem(y)
xlabel("n")
ylabel("y")
grid()
```



La réponse impulsionnelle d'un intégrateur est un échelon. Elle ne tend pas vers zéro (elle reste même constante), ce qui montre l'instabilité du filtre. En pratique, le filtre intégrateur est utilisable sur une durée limitée si le décalage indésirable du signal d'entrée est très faible. C'est la raison pour laquelle nous avons ajouté une correction du décalage dans l'amplificateur inverseur, qui permet de corriger le décalage introduit par l'électronique, en particulier celle de la carte d'acquisition. On ajoutera également une correction logicielle dans le script python.

Pour les signaux alternatifs, un intégrateur stable peut être réalisé au moyen d'un filtre passe-bas du premier ordre. En voici un exemple :

```
b1,a1 = scipy.signal.iirfilter(N=1,Wn=[0.0001*2],btype="lowpass",ftype="butter")
w,h = scipy.signal.freqz(b1,a1,worN=numpy.logspace(-5,-0.31,1000)*2*numpy.pi)
figure(figsize=(8,8))
subplot(211)
plot(w/(2*numpy.pi)*fe_finale,20*numpy.log10(numpy.absolute(h)))
xlabel("f (Hz)")
ylabel("GdB")
xscale("log")
grid()
subplot(212)
plot(w/(2*numpy.pi)*fe_finale,numpy.angle(h)/numpy.pi)
xlabel("f (Hz)")
ylabel("phase")
xscale("log")
grid()
```



Le gain à fréquence nulle a une valeur finie, ce qui permet d'éviter la dérive à long terme. La fréquence de coupure est très basse, égale à $0,02\ Hz$, une valeur impossible à réaliser avec un filtre analogique RC.

Ce type de filtre ne convient pas pour notre expérience de déplacement de l'aimant, car le signal obtenu n'est pas alternatif : il comporte une composante constante qu'il faut inclure dans l'intégration. Nous verrons néanmoins son utilité dans une expérience de variation alternative du flux.

3.c. Programme python

Voici le programme python complet, qui effectue un tracé en temps réel du signal numérisé et du signal intégré. On a ajouté un curseur pour corriger le décalage de manière logicielle, un curseur pour modifier le gain de l'intégrateur, et un bouton RESET pour annuler le signal de sortie, ce qui peut être utile après une dérive trop importante. La variable `Umax` définie au début du script permet de choisir le calibre de la SysamSP5 et l'échelle de l'affichage. Le réglage matériel du décalage (avec le potentiomètre de l'amplificateur) doit être fait pour un calibre donné. Si l'on a pas le temps de faire ce réglage, on peut utiliser le décalage logiciel avec le curseur. La largeur de la fenêtre affichée est réglée par la variable `duree`. La durée totale de l'acquisition est déterminée par le nombre d'échantillons `N`. Lorsque l'acquisition est terminée, il faut fermer la fenêtre

graphique pour que l'enregistrement des données dans un fichier se fasse.
filtreIntegrateurSP5.py

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

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

sys=pycan.Sysam("SP5")
Umax = 5.0
sys.config_entrees([0],[Umax])
fe=1000.0 # fréquence de la numérisation
te=1.0/fe
duree_totale = 60.0
N = int(duree_totale*fe)
sys.config_echantillon_permanent(te*1e6,N)
reduction = 5 # réduction de la fréquence d'échantillonnage
fe_r = fe/reduction
te_r = te*reduction
duree = 10.0 # durée des blocs
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 anti-repliement
fc = fe/reduction/2*0.8 # fréquence de coupure
b = scipy.signal.firwin(numtaps=101,cutoff=[fc/fe],nyq=0.5,window='hann')
sys.config_filtre([1],b)

#intégrateur parfait
b1 = [0.01,0.01]
a1 = [1.0,-1.0]
zi1 = scipy.signal.lfiltic(b1,a1,y=[0],x=[0]) # condition initiale

sys.lancer_permanent()
n_tot = 0 # nombre d'échantillons acquis

fig,ax = subplots()
subplots_adjust(left=0.1, bottom=0.3)
line0, = ax.plot([0],[0])
line1, = ax.plot([0],[0])
ax.grid()
ax.axis([0,duree,-Umax,Umax])
ax.set_xlabel("t (s)")
```

```
axe_zero = axes([0.1,0.2,0.8,0.03])
zero=0.0
slider_zero = Slider(axe_zero,"Zero",-0.01,0.01,valinit=zero)
zero = 0.0
def update_zero(val):
    global zero
    zero = slider_zero.val
slider_zero.on_changed(update_zero)
axe_gain = axes([0.1,0.1,0.8,0.03])
gain = 1
slider_gain = Slider(axe_gain,"Gain",-1,3,valinit=0)
def update_gain(val):
    global gain
    gain = 10**(slider_gain.val)
slider_gain.on_changed(update_gain)

axe_reset = axes([0.8, 0.025, 0.1, 0.04])
button_reset = Button(axe_reset, 'Reset', hovercolor='0.975')
def reset(event):
    global zi1
    for k in range(len(zi1)):
        zi1[k] = 0.0
button_reset.on_clicked(reset)

x = numpy.array([],dtype=numpy.float32)
y = numpy.array([],dtype=numpy.float32)
t = numpy.array([],dtype=numpy.float32)

def animate(i):
    global ax,sys,x,y,t,line0,n_tot,longueur_bloc,zi1,zi2,zi3,Umax,zero
    data = sys.paquet(n_tot,reduction)
    u0 = data[2]-zero
    [u1,zi1] = scipy.signal.lfilter(b1,a1,u0,zi=zi1)
    u1 = u1*gain
    n_tot += u0.size
    x = numpy.append(x,u0)
    y = numpy.append(y,u1)
    t = numpy.append(t,data[0])
    if n_tot >= nombre_echant:
        print(u"Acquisition terminée")
    i2 = n_tot-1
    if n_tot == 0:
        return
    if n_tot > longueur_bloc:
        i1 = i2-longueur_bloc
    else:
        i1 = 0
    line0.set_data(t[i1:i2],x[i1:i2])
```

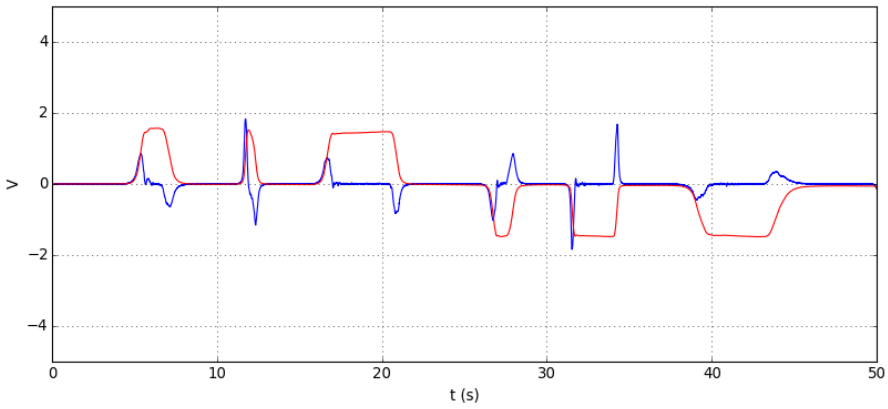
```
    line1.set_data(t[i1:i2],y[i1:i2])
    ax.axis([t[i1],t[i2],-Umax,Umax])


ani = animation.FuncAnimation(fig,animate,frames=nombre_blocs,repeat=True,interval=du
show(block=True)
numpy.savetxt("induction-1.txt",[t,x,y])
sys.fermer()
```

4. Résultats

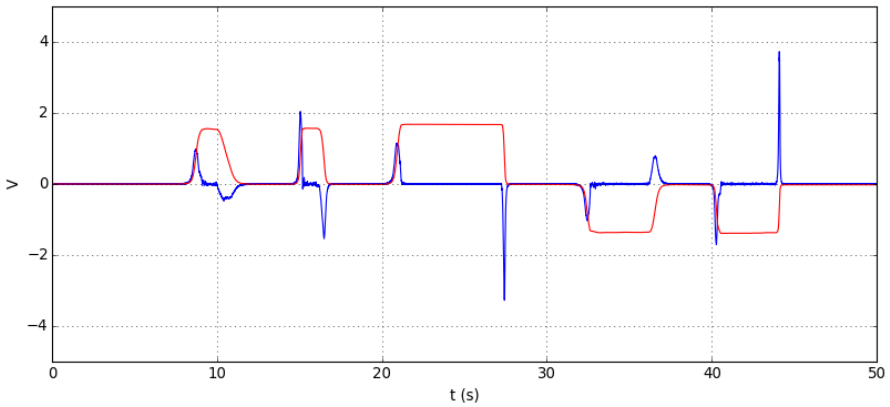
4.a. Mesure de flux magnétique

Voici les résultats d'une expérience consistant à introduire l'aimant dans la bobine en partant d'une position éloignée d'environ 50 *cm*. L'aimant est introduit puis retiré à différentes vitesses, puis en changeant son orientation.

```
[t,x,y] = numpy.loadtxt("induction-1.txt")
figure(figsize=(12,5))
plot(t,x,"b")
plot(t,y,"r")
xlabel("t (s)")
ylabel("V")
grid()
axis([0,50,-5,5])
```




```
[t,x,y] = numpy.loadtxt("induction-2.txt")
figure(figsize=(12,5))
plot(t,x,"b")
plot(t,y,"r")
xlabel("t (s)")
ylabel("V")
grid()
axis([0,50,-5,5])
```

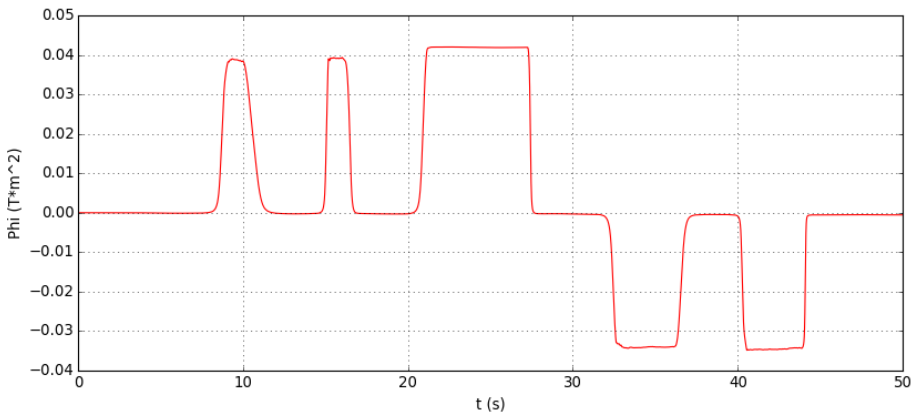


Le flux magnétique obtenu est celui de la position finale de l'aimant, qui est à peu près la même à chaque mouvement. La force électromotrice maximale est plus grande lorsque le déplacement est plus rapide. Cette expérience constitue une confirmation de la loi de Faraday, puisqu'elle montre que l'intégrale de la force électromotrice ne dépend que des positions initiale et finale de l'aimant.

On remarque que lorsque le mouvement est rapide la force électromotrice s'approche d'une impulsion et l'intégration donne un échelon, conformément à la réponse impulsionnelle d'un intégrateur.

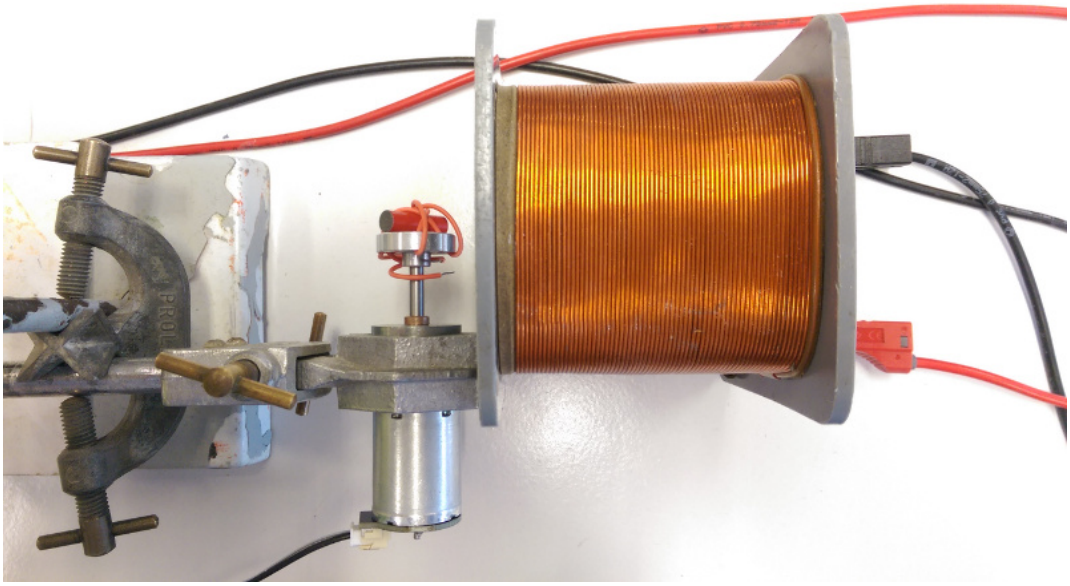
Pour calculer le flux magnétique, il faut tenir compte du gain $G = 10$ de l'amplificateur, du coefficient $b = 0,01$ de l'intégrateur, de la fréquence d'échantillonnage, et du gain supplémentaire g.

```
G = 10.0
g = 1.0
b = 0.01
fe = 200.0
facteur = 1.0/fe*0.5/b/(G*g)
flux = y*facteur
figure(figsize=(12,5))
plot(t,flux,"r")
xlabel("t (s)")
ylabel("Phi (T*m^2)")
grid()
```



4.b. Flux magnétique alternatif

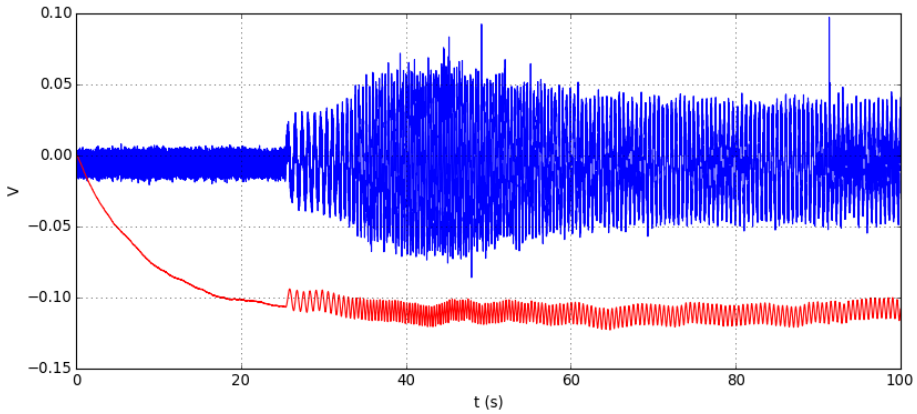
Un petit aimant placé à l'entrée de la bobine est entraîné en rotation par un moteur électrique à courant continu. L'axe de rotation est perpendiculaire à l'axe de la bobine.



La force électromotrice est beaucoup plus faible que dans l'expérience précédente, ce qui oblige à utiliser un gain plus fort pour l'amplification avant numérisation (calibre 0.2 V de la carte SysamSP5). En conséquence, le décalage relatif est plus grand et l'effet de dérive très important. La force électromotrice étant alternative, on peut utiliser le filtre passe-bas au lieu de l'intégrateur parfait. Cela a pour effet de ramener le gain à fréquence nulle à une valeur finie. Le décalage en entrée du filtre se traduit par un décalage important en sortie, mais il n'y a plus de dérive. La fréquence de coupure est $0,02\ Hz$, largement inférieure à la fréquence de rotation du moteur.

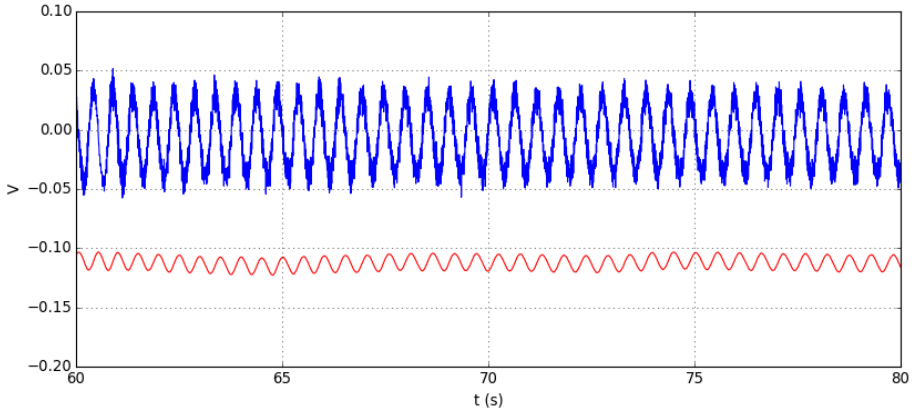
Voici un enregistrement complet de 100 s. On commence par attendre que le décalage en sortie atteigne sa valeur stationnaire, puis on démarre le moteur.

```
[t,x,y] = numpy.loadtxt("induction-3.txt")
figure(figsize=(12,5))
plot(t,x,"b")
plot(t,y,"r")
xlabel("t (s)")
ylabel("V")
grid()
```



Voici un détail :

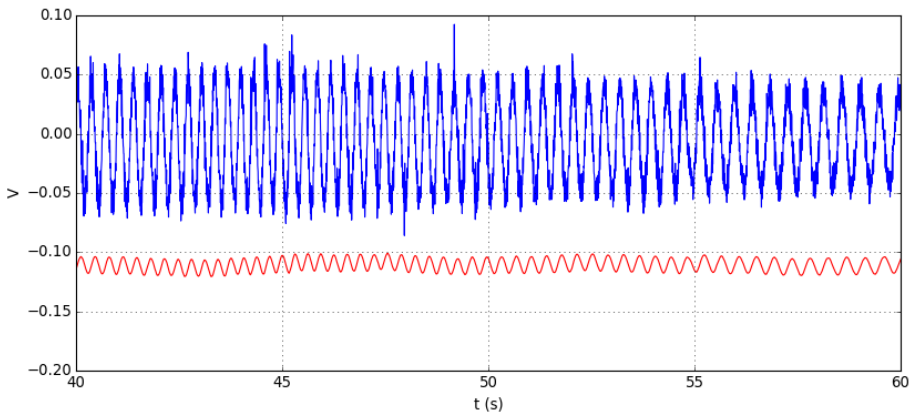
```
axis([60,80,-0.2,0.1])
```



On voit bien l'effet passe-bas du filtre intégrateur : alors que la force électromotrice est très bruitée, le flux magnétique est bien lisse. La grandeur pertinente est l'amplitude d'oscillation du flux de crête à crête, qui est égale au double du flux magnétique lorsque l'aimant est aligné avec la bobine. Cette technique de rotation est donc un bon moyen de mesurer le flux créé par un petit aimant.

Voyons une zone où la vitesse du moteur est plus grande :

```
axis([40,60,-0.2,0.1])
```



L'amplitude d'oscillation de la force électromotrice est plus grande (elle est proportionnelle à la fréquence), mais celle du flux est inchangée, conformément à la loi de Faraday.