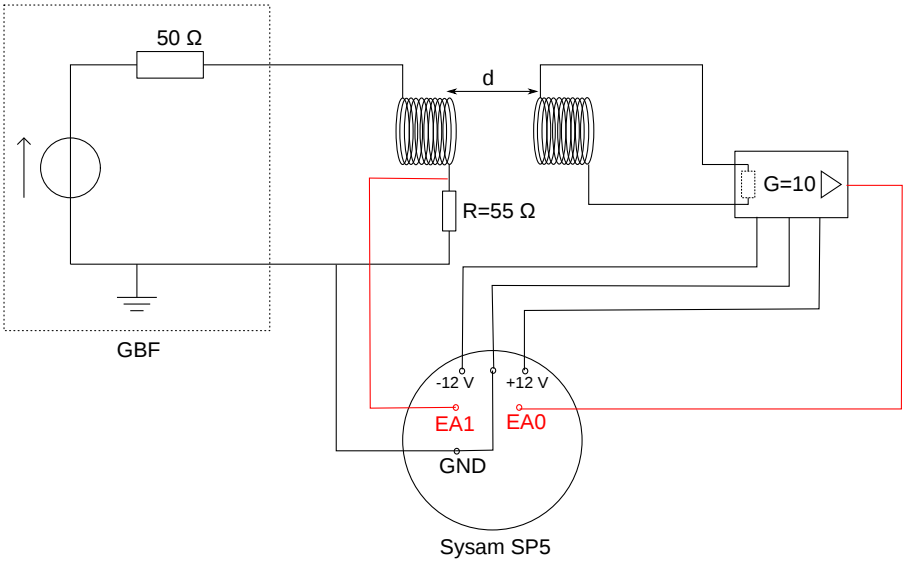
Mesure d'une inductance mutuelle

1. Dispositif expérimental

Le circuit primaire est constitué d'une bobine de 500 spires placée en série avec une résistance $R = 55\ \Omega$ et un générateur basse fréquence délivrant une tension sinusoïdale d'une fréquence de l'ordre de 100 Hz.

Le circuit secondaire est constitué d'une bobine identique à la première, placée sur le même axe à une distance variable. Le circuit est fermé par l'appareil de mesure, qui est un amplificateur inverseur de gain $G = 10$, dont la sortie est reliée à l'entrée EA0 de la carte d'acquisition SysamSP5. L'impédance d'entrée de l'amplificateur (10 $k\Omega$) étant beaucoup plus grande que l'impédance de la bobine (inférieure à 100 Ω à 1 kHz), on mesure pratiquement la force électromotrice $e(t)$ générée dans la bobine.

La tension aux bornes de la résistance R du circuit primaire est acquise sur la voie EA1.



La fréquence d'échantillonnage est fixée à 5000 Hz. Voici le script python qui effectue l'acquisition. Le temps, la force électromotrice (multipliée par G) et l'intensité dans le circuit primaire sont enregistrés dans un fichier texte. L'amplitude maximale de la voie 0 devra être ajustée en fonction de l'amplitude de la force électromotrice.

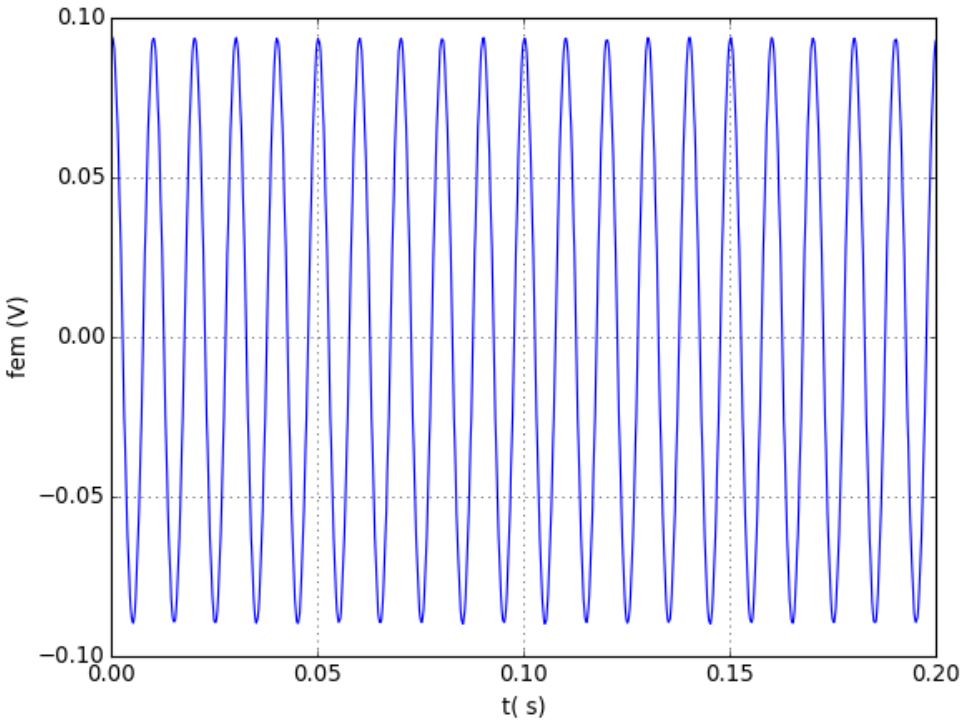
numerisation.py

```
# -*- coding: utf-8 -*-
R=55.0 # résistance du cicuit primaire
#branchement EA0 : fem secondaire, EA1 : Ri primaire
sys=pycan.Sysam("SP5")
fe=5000.0 # fréquence d'échantillonnage
te=1.0/fe
N=10000
Umax = 10.0 # à réduire lorsque la fem diminue
```

```
sys.config_entrees([0,1],[Umax,10])
sys.config_echantillon(te*1e6,N)
sys.acquerir()
t = sys.temps()
u = sys.entrees()
temps = t[0]
fem = u[0]
i1 = u[1]/R
sys.fermer()
numpy.savetxt("mutuelle-1.txt",[temps,fem,i1])
```

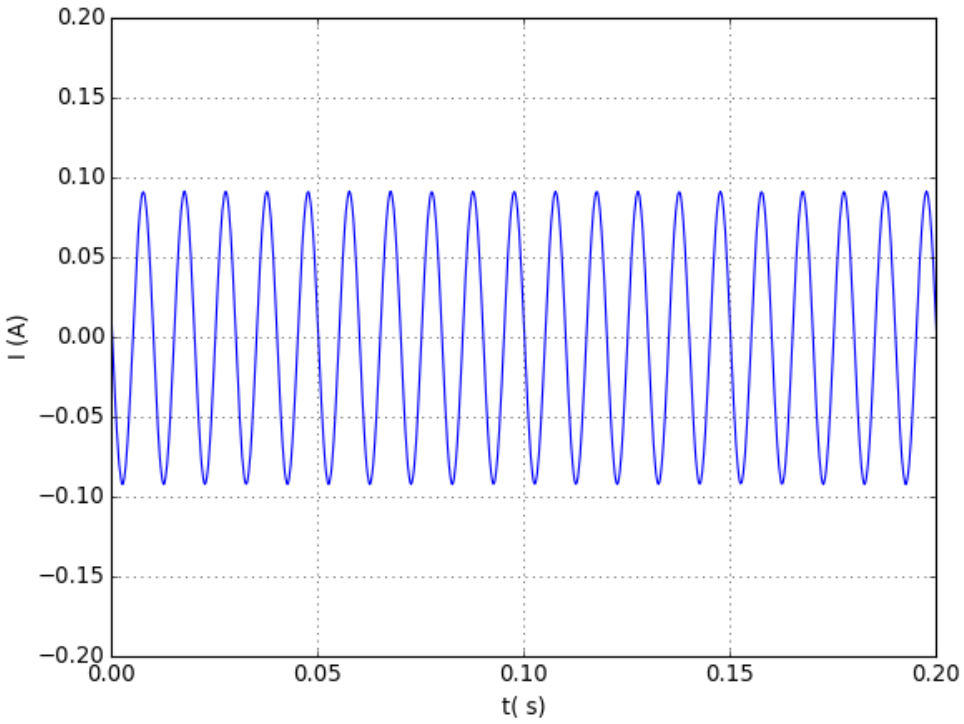
Voici un exemple d'enregistrement de la force électromotrice, réalisé pour une distance $d = 0\ cm$:

```
import numpy
from matplotlib.pyplot import *

[temps,fem,i1] = numpy.loadtxt("mutuelle-1.txt")
G = 10
fem = fem/G

figure()
plot(temps,fem)
xlabel("t( s)")
ylabel("fem (V)")
axis([0,0.2,-0.1,0.1])
grid()
```



Voici l'intensité du courant dans le circuit primaire :

```
figure()
plot(temps,i1)
xlabel("t( s)")
ylabel("I (A)")
axis([0,0.2,-0.2,0.2])
grid()
```



2. Calcul du flux magnétique

2.a. Intégrateur numérique

La loi de Faraday permet d'exprimer la force électromotrice en fonction du flux magnétique dans le circuit secondaire :

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

Dans le cas présent, on peut omettre le signe moins car la force électromotrice est déjà inversée (par l'ampli inverseur). De toute manière, son signe dépend du sens d'enroulement de la bobine secondaire. Il faut donc effectuer l'intégration suivante :

$$\Phi(t) = \int_0^t e(t)\,dt \tag{2}$$

L'intégrateur numérique parfait est défini par la relation de récurrence :

$$y_n = y_{n-1} + \frac{T_e}{2}(x_n + x_{n-1}) \tag{3}$$

qui correspond à la méthode des trapèzes pour le calcul numérique approchée d'une intégrale. Sa fonction de transfert en Z est :

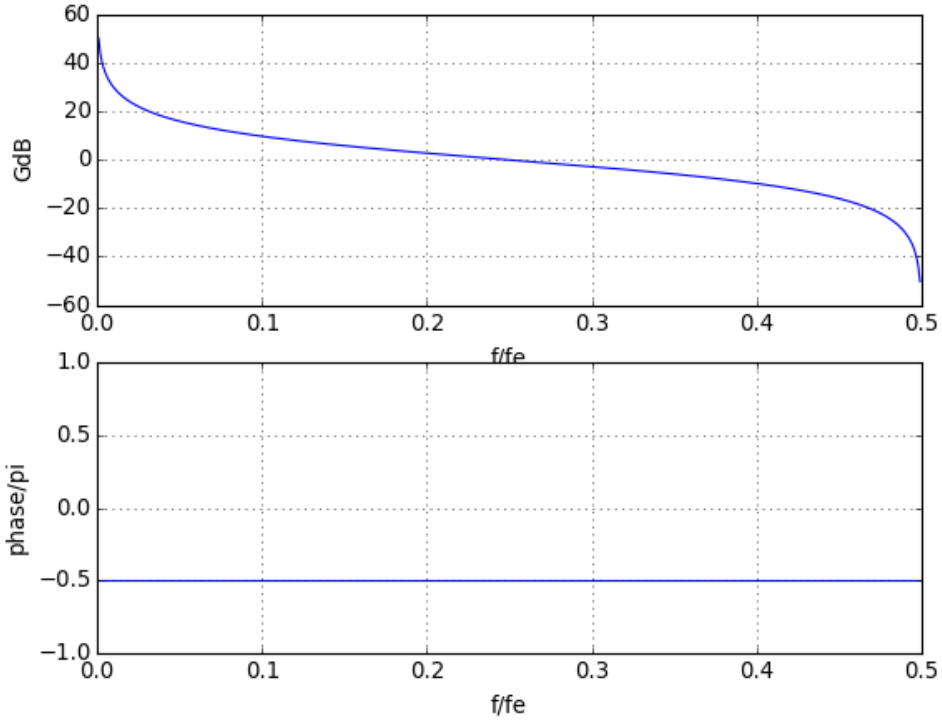
$$H(z) = \frac{T_e}{2}\frac{1+z^{-1}}{1-z^{-1}} \tag{4}$$

Sa réponse fréquentielle est obtenue avec :

$$Z = e^{i2\pi\frac{f}{f_e}} \tag{5}$$

Voici le tracé du gain et du déphasage en fonction de la fréquence :

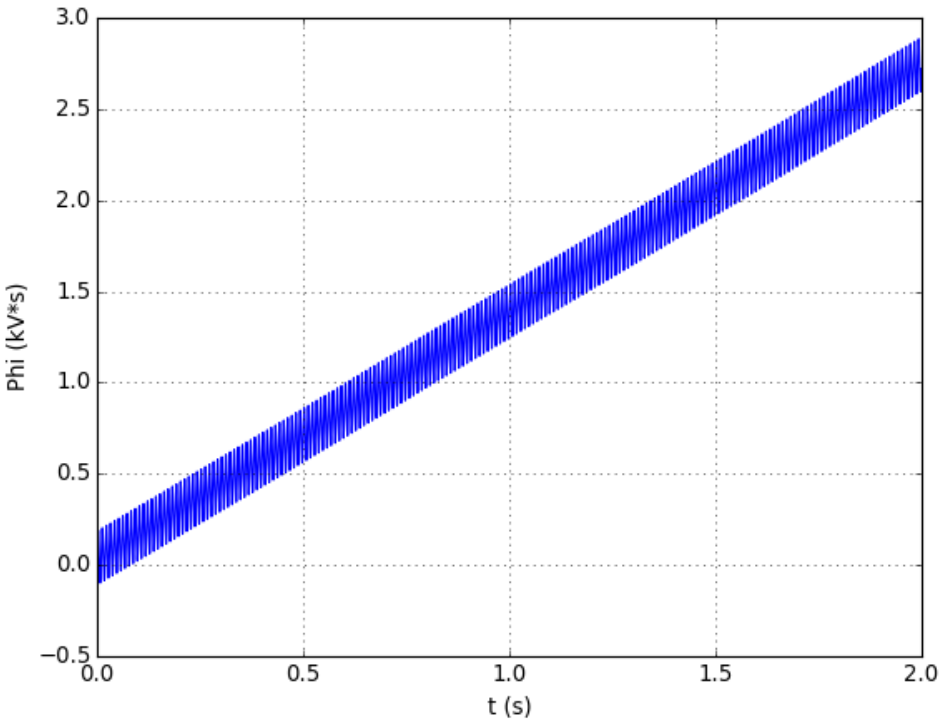
```
import scipy.signal
b=[1,1]
a=[1,-1]
w,h=scipy.signal.freqz(b,a)

figure()
subplot(211)
plot(w/(2*numpy.pi),20*numpy.log10(numpy.absolute(h)))
xlabel("f/fe")
ylabel("GdB")
grid()
subplot(212)
plot(w/(2*numpy.pi),numpy.angle(h)/numpy.pi)
xlabel("f/fe")
ylabel("phase/pi")
axis([0,0.5,-1,1])
grid()
```



Le déphasage égal à $-\pi/2$ sur toute la gamme de fréquence accessible confirme le caractère intégrateur parfait.

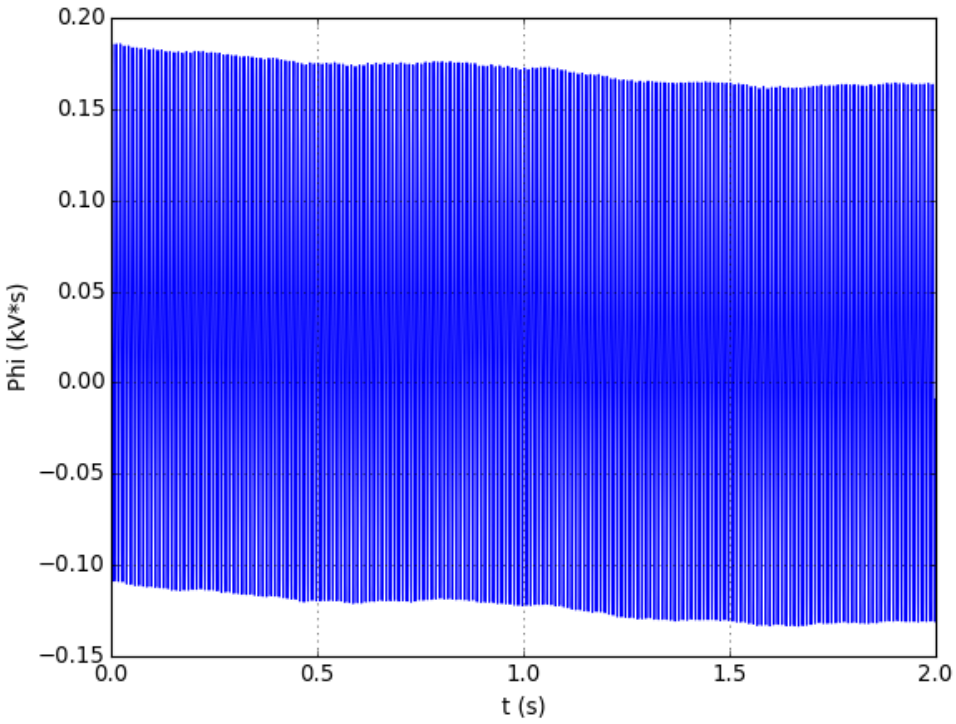
On applique cet intégrateur à la force électromotrice numérisée, en utilisant la fonction `scipy.signal.lfilter`. On peut ausi écrire une fonction pour calculer explicitement la récurrence dans une boucle.

```
te = temps[1]-temps[0]
zi1 = scipy.signal.lfiltic(b,a,y=[0],x=[0]) # condition initiale
[flux,zi1] = scipy.signal.lfilter(b,a,fem,zi=zi1)
flux *= te/2
figure()
plot(temps,flux*1000)
xlabel("t (s)")
ylabel("Phi (kV*s)")
grid()
```



On constate une dérive très importante, qui est la conséquence du gain infini à fréquence nulle. Il faut donc enlever la composante continue avant d'intégrer :

```
fem2 = fem-fem.mean()
zi1 = scipy.signal.lfiltic(b,a,y=[0],x=[0]) # condition initiale
[flux,zi1] = scipy.signal.lfilter(b,a,fem2,zi=zi1)
flux *= te/2
figure()
plot(temps,flux*1000)
xlabel("t (s)")
ylabel("Phi (kV*s)")
grid()
```



Le résultat n'est pas parfait, car la composante de fréquence nulle varie au cours de l'enregistrement. Nous allons voir comment améliorer l'intégrateur en réduisant le gain à fréquence nulle.

2.b. Filtre passe-bas intégrateur

Une première amélioration consiste à modifier la relation de récurrence de la manière suivante :

$$y_n = r y_{n-1} + \frac{T_e}{2}(x_n + x_{n-1}) \tag{6}$$

Le coefficient r doit être strictement inférieur à 1 mais proche de 1. Pour une justification de cette relation, voir Conception d'un filtre par placement des zéros et des pôles.

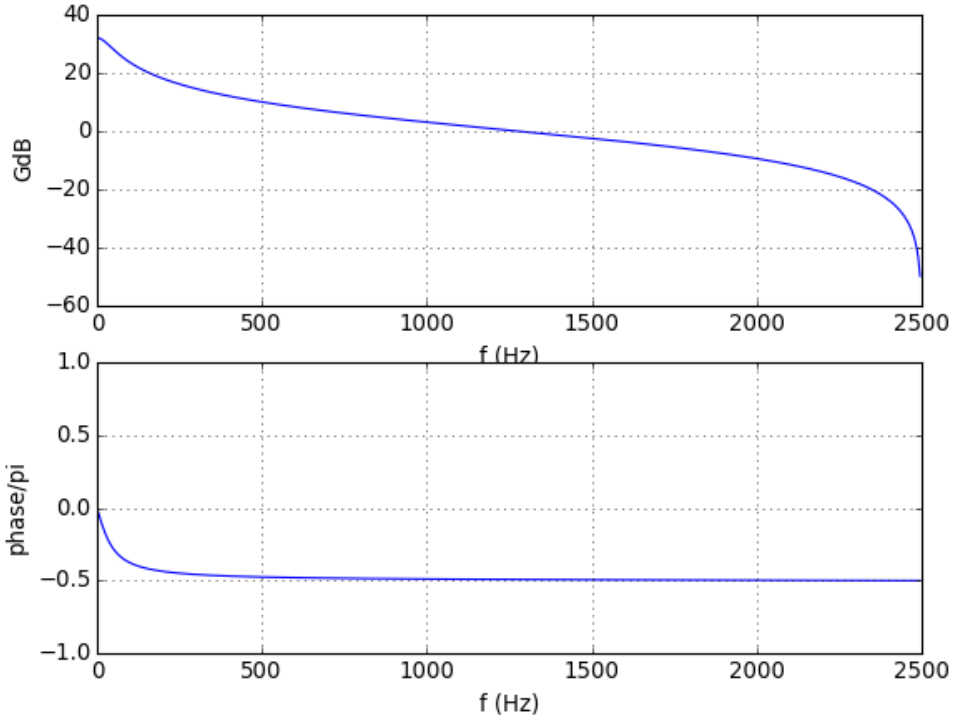
La fonction de transfert en Z est :

$$H(z) = \frac{T_e}{2}\frac{1+z^{-1}}{1-rz^{-1}} \tag{7}$$

On trace la réponse fréquentielle :

```
import scipy.signal
r = 0.95
b=[1,1]
a=[1,-r]
fe=1.0/te
w,h=scipy.signal.freqz(b,a)

figure()
```

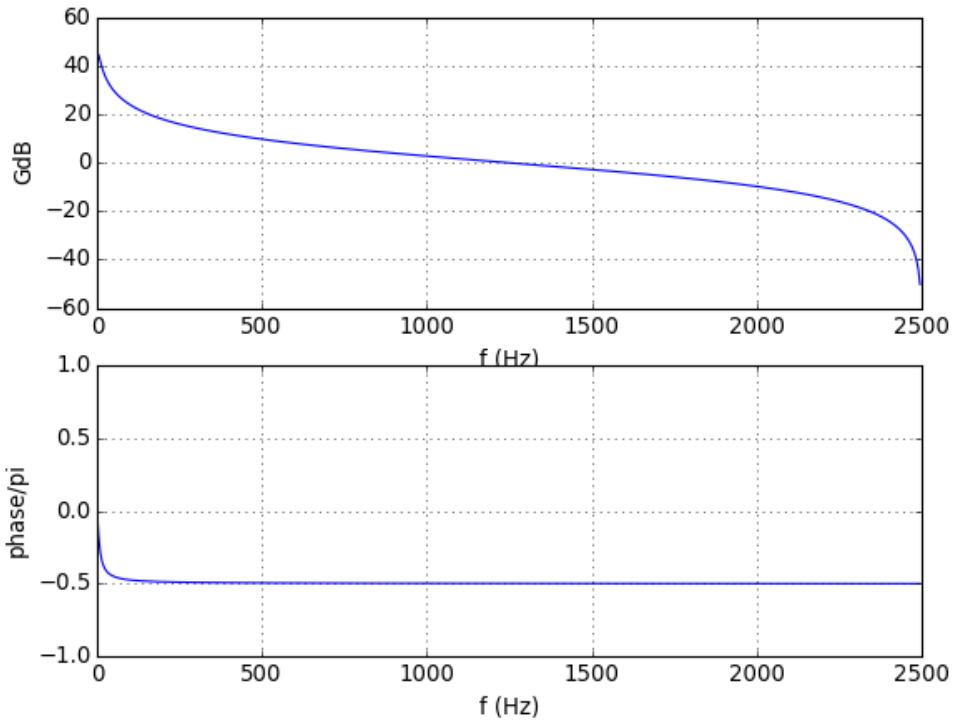
```
subplot(211)
plot(w/(2*numpy.pi)*fe,20*numpy.log10(numpy.absolute(h)))
xlabel("f (Hz)")
ylabel("GdB")
grid()
subplot(212)
plot(w/(2*numpy.pi)*fe,numpy.angle(h)/numpy.pi)
xlabel("f (Hz)")
ylabel("phase/pi")
axis([0,2500,-1,1])
grid()
```



Il s'agit d'un filtre passe-bas du premier ordre, similaire à un filtre RC analogique. Le gain à fréquence nulle est ramené à une valeur finie, ce qui permet d'éliminer la dérive. En revanche, l'intégration n'est réalisée qu'à partir de 500 Hz. Pour notre application, où la fréquence du signal est 100 Hz, il faut augmenter r :

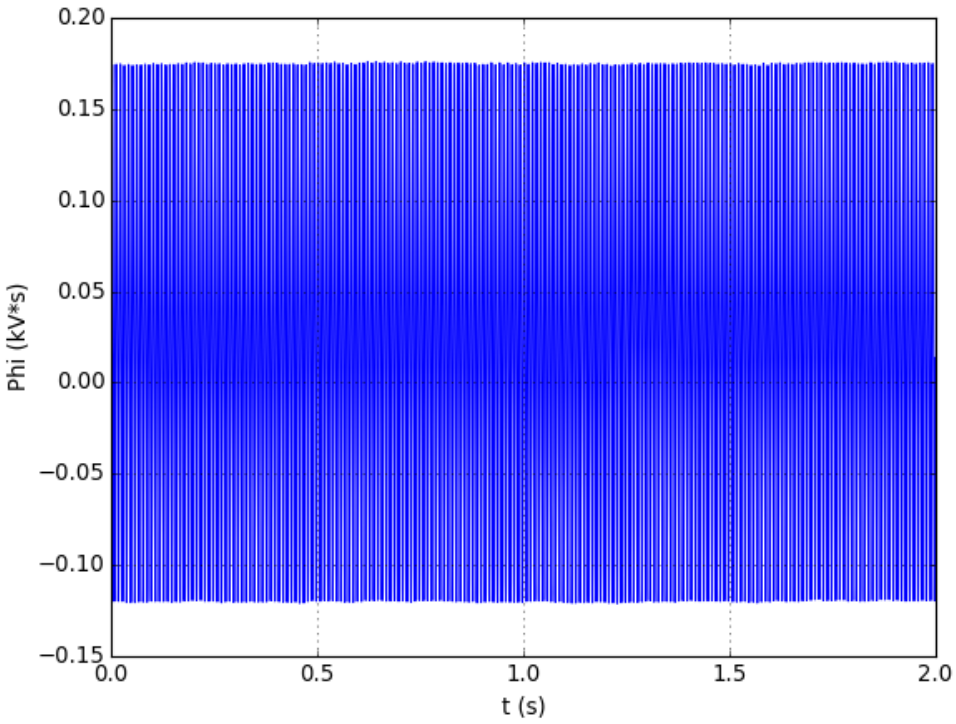
```
import scipy.signal
r = 0.99
b=[1,1]
a=[1,-r]
w,h=scipy.signal.freqz(b,a)

figure()
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.angle(h)/numpy.pi)
xlabel("f (Hz)")
ylabel("phase/pi")
axis([0,2500,-1,1])
grid()
```



On applique ce filtre à la force électromotrice :

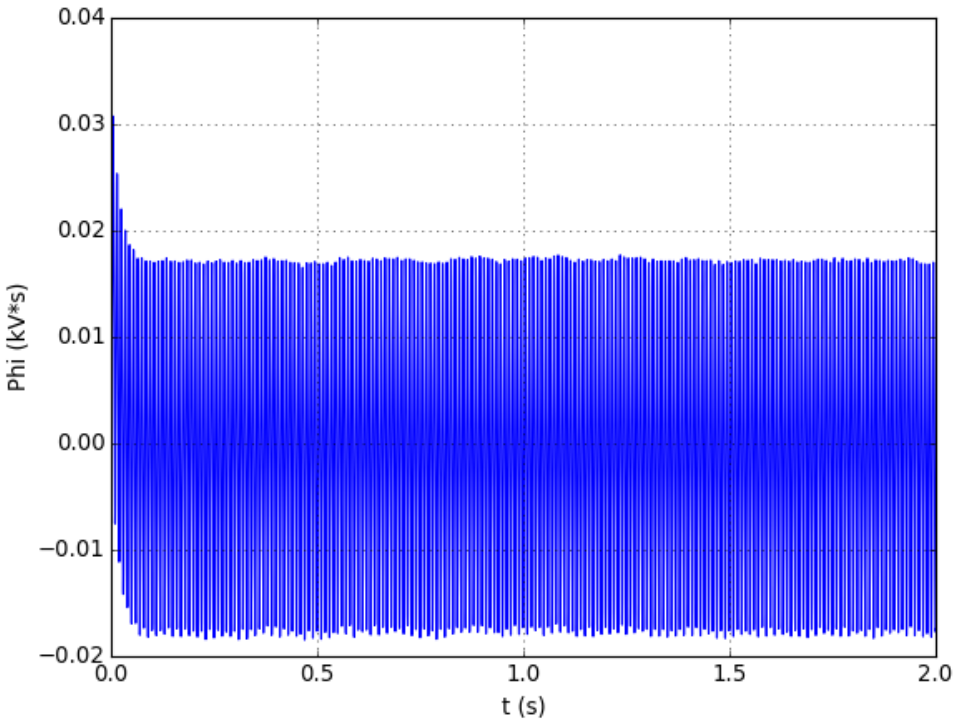
```
zi1 = scipy.signal.lfiltic(b,a,y=[0],x=[0]) # condition initiale
[flux,zi1] = scipy.signal.lfilter(b,a,fem,zi=zi1)
flux *= te/2
figure()
plot(temps,flux*1000)
xlabel("t (s)")
ylabel("Phi (kV*s)")
grid()
```



Après le régime transitoire, on obtient une oscillation avec un décalage constant (à cause du grand gain à fréquence nulle).

On vérifie le fonctionnement sur l'enregistrement obtenu avec une distance plus grande ($d = 9\ cm$) :

```
[temps,fem,i1] = numpy.loadtxt("mutuelle-10.txt")
fem = fem/G
zi1 = scipy.signal.lfiltic(b,a,y=[0],x=[0]) # condition initiale
[flux,zi1] = scipy.signal.lfilter(b,a,fem,zi=zi1)
flux *= te/2
figure()
plot(temps,flux*1000)
xlabel("t (s)")
ylabel("Phi (kV*s)")
grid()
```



La fin du traitement consiste à calculer la valeur efficace des oscillations du flux, en éliminant la zone transitoire. On calcule aussi la valeur efficace de l'oscillation du courant dans le circuit primaire. Le rapport donne l'inductance mutuelle. Voici la fonction qui lit un fichier et fait ces calculs :

```
def inductance(fichier):
    [temps,fem,i1] = numpy.loadtxt(fichier)
    fem = fem/G
    te = temps[1]-temps[0]
    fe = 1.0/te
    r = 0.99
    b=[1,1]
    a=[1,-r]
    zi1 = scipy.signal.lfiltic(b,a,y=[0],x=[0])
    [flux,zi1] = scipy.signal.lfilter(b,a,fem,zi=zi1)
    flux *= te/2
    N=len(flux)
    flux = flux-flux.mean()
    flux_efficace = numpy.std(flux[int(N/4):N-1])
    i1_efficace = numpy.std(i1[int(N/4):N-1])
    M = flux_efficace/i1_efficace
    return M
```

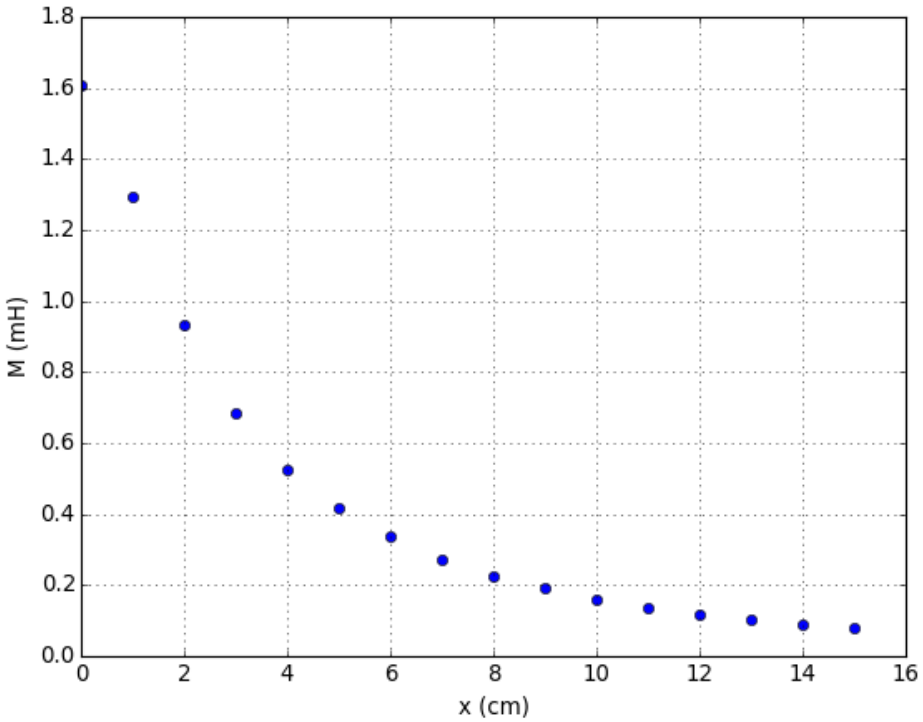
Voici par exemple l'inductance mutuelle pour $d = 0$:

```
M1 = inductance("mutuelle-1.txt")
```

```
print(M1)
--> 0.0016098347151059828
```

3. Inductance mutuelle en fonction de la distance

```
M = []
d = []
for i in range(16):
    d.append(i)
    M.append(1000*inductance("mutuelle-%d.txt"%(i+1)))
figure()
plot(d,M,"o")
xlabel("x (cm)")
ylabel("M (mH)")
grid()
```



4. Filtre passe-bande intégrateur

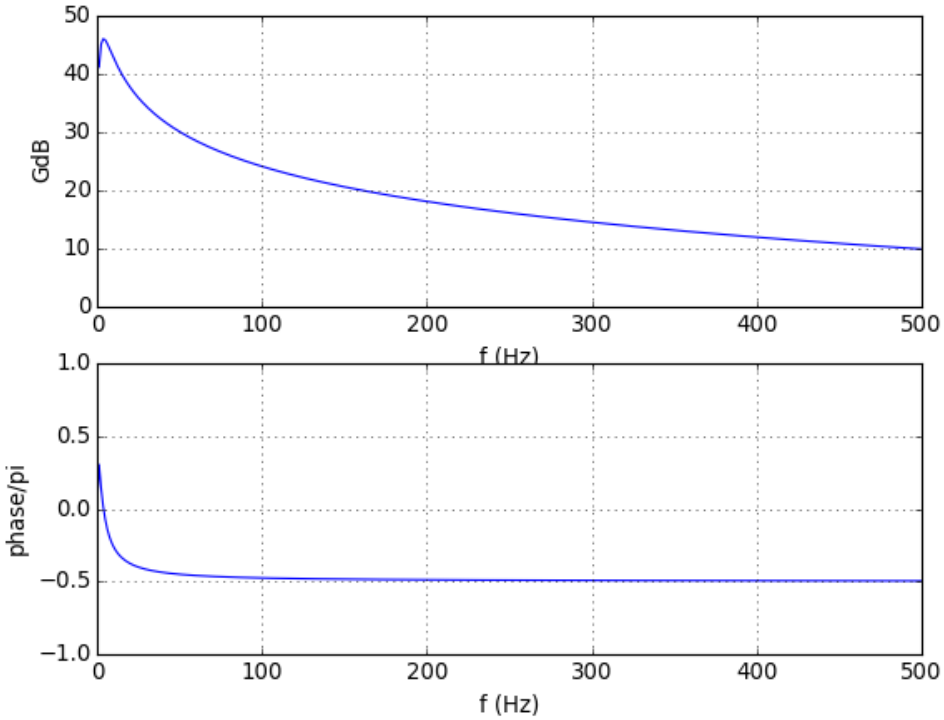
Le filtre passe-bas intégrateur a l'inconvénient d'avoir un gain important à fréquence nulle, qui amplifie le décalage présent dans le signal d'entrée. Pour un filtre fonctionnant en temps réel, il faut annuler le gain à fréquence nulle, ou du moins le réduire à un niveau très faible.

On considère le filtre défini par la fonction de transfert suivante, qui associe en série un filtre passe-bas et un filtre passe-haut :

$$H(z) = \frac{T_e}{2}\frac{z-1}{z-r_1}\frac{z+1}{z-r_2} = \frac{T_e}{2}\frac{1-z^{-2}}{1-(r_1+r_2)z^{-1}+r_1r_2z^{-2}} \tag{8}$$

Les valeurs de r_1 et r_2 doivent être proches de 1 mais inférieures à 1. Voici un exemple :

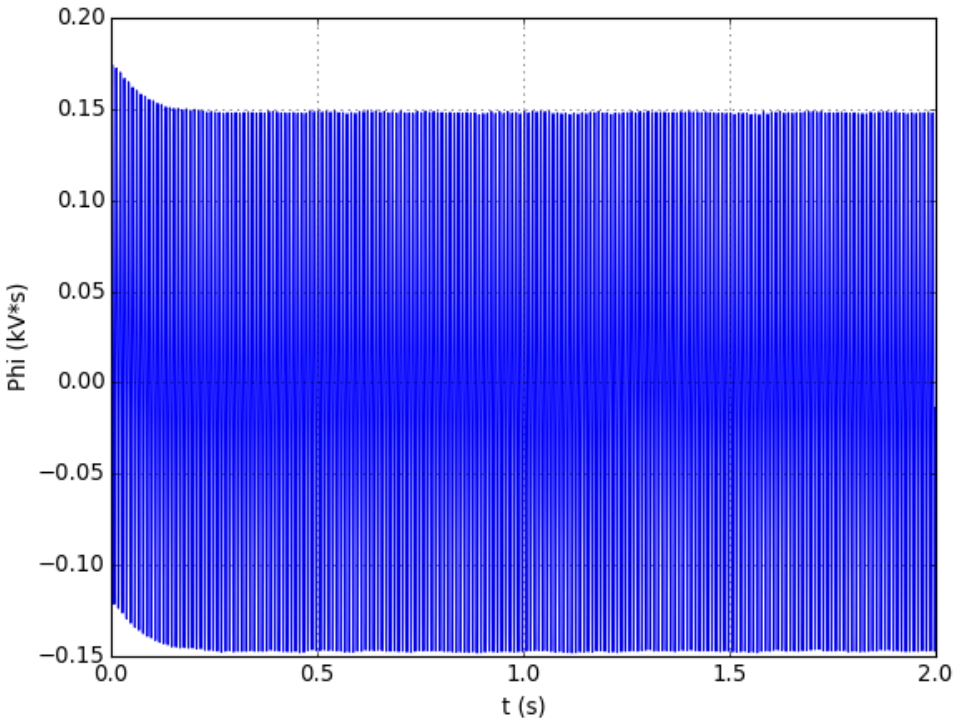
```
r1=0.995
r2=0.995
b=[1,0,-1]
a=[1,-(r1+r2),r1*r2]

w,h=scipy.signal.freqz(b,a,worN=2000)

figure()
subplot(211)
plot(w/(2*numpy.pi)*fe,20*numpy.log10(numpy.absolute(h)))
xlabel("f (Hz)")
ylabel("GdB")
axis([0,500,0,50])
grid()
subplot(212)
plot(w/(2*numpy.pi)*fe,numpy.angle(h)/numpy.pi)
xlabel("f (Hz)")
ylabel("phase/pi")
axis([0,500,-1,1])
grid()
```



On a bien un intégrateur à partir de 100 Hz, et le gain à fréquence nulle est nul.

On l'applique à la force électromotrice :

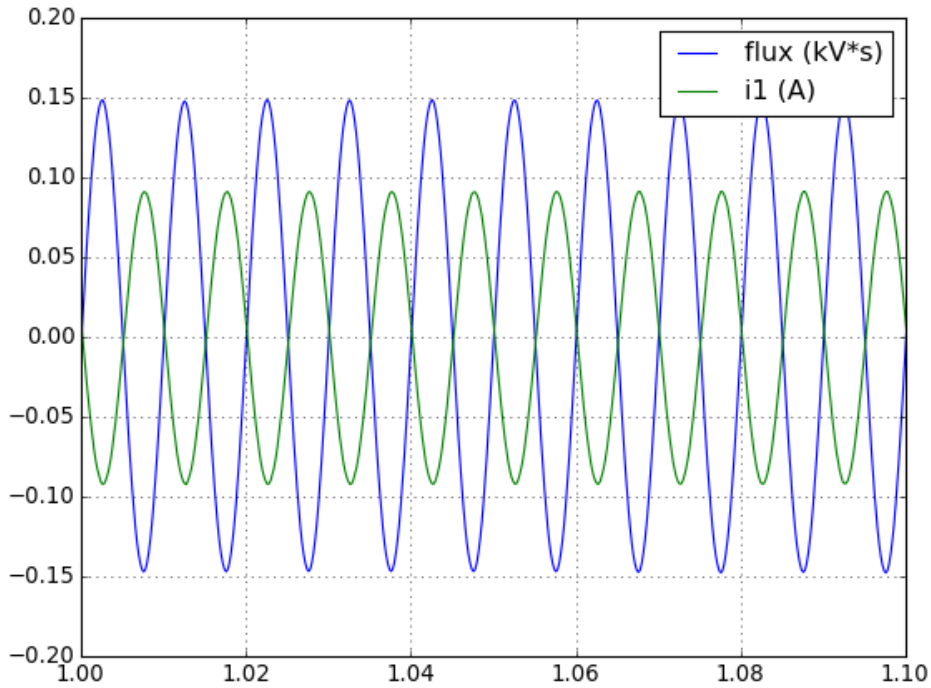
```
[temps,fem,i1] = numpy.loadtxt("mutuelle-1.txt")
fem = fem/G
zi1 = scipy.signal.lfiltic(b,a,y=[0],x=[0]) # condition initiale
[flux,zi1] = scipy.signal.lfilter(b,a,fem,zi=zi1)
flux *= te/2
figure()
plot(temps,flux*1000)
xlabel("t (s)")
ylabel("Phi (kV*s)")
grid()
```



On a bien en sortie une oscillation non décalée. Le temps de réponse du filtre est de l'ordre de $0,2$ s.

On peut aussi tracer le flux avec l'intensité du courant dans la bobine primaire :

```
figure()
plot(temps,flux*1000,label="flux (kV*s)")
plot(temps,i1,label="i1 (A)")
axis([1.0,1.1,-0.2,0.2])
legend(loc="upper right")
grid()
```



Le flux et le courant sont en phase ou en opposition de phase en fonction du sens de branchement des bobines. Dans le cas présent, l'inductance mutuelle est négative.

5. Filtrage et tracé en temps réel

Le script python ci-dessous met en œuvre le filtre passe-bande intégrateur. La fréquence d'échantillonnage est de 25 kHz. Un filtre RIF passe-bas est appliqué avant de réduire la fréquence d'échantillonnage à 5 kHz, après quoi le filtre intégrateur est appliqué. Les signaux suivants sont tracés en temps réel :

- ▷ La force éléctromotrice multipliée par G=10, enregistrée sur la voie EA0.
- ▷ L'intensité du courant dans la bobine primaire, multipliée par 10.
- ▷ Le flux magnétique dans la bobine secondaire, multipliée par 1000, et par un gain ajustable avec un curseur.

Les paramètres suivants peuvent être modifiés :

- ▷ `Umax` : la tension maximale sur la voie EA0.
- ▷ `duree_totale` : la durée totale de l'enregistrement en secondes.
- ▷ `duree` : la durée de la fenêtre affichée en secondes.
- ▷ Les paramètres `r1` et `r1` du filtre passe-bande.

La réponse fréquentielle du filtre est tracée dans une fenêtre. À la fin de l'acquisition, les signaux sont enregistrés dans un fichier texte.

inductanceMutuelleTempsReel.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
import time

G=10 # gain ampli
R=55.0 #résistance courant primaire
sys=pycan.Sysam("SP5")
Umax = 2.0
sys.config_entrees([0,1],[Umax,10])
reduction = 5 # réduction de la fréquence d'échantillonnage
fe=5000.0*reduction # fréquence d'échantillonnage
te=1.0/fe
duree_totale = 60.0
N = int(duree_totale*fe) # nombre d'échantillons total à acquerir
sys.config_echantillon_permanent(te*1e6,N)

fe_r = fe/reduction
te_r = te*reduction
duree = 0.2 # 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)

# filtre passe-bande ordre 1
r1=0.995
r2=0.995
b1=[te_r/2,0,-te_r/2]
a1=[1,-(r1+r2),r1*r2]
zi1 = scipy.signal.lfiltic(b1,a1,y=[0],x=[0]) # condition initiale

w,H=scipy.signal.freqz(b1,a1)

figure()
subplot(211)
plot(w/(2*numpy.pi)*fe,numpy.absolute(H))
xlabel("f (Hz)")
```

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



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

fig,ax = subplots()
subplots_adjust(left=0.1, bottom=0.3)
subplots_adjust(left=0.1, bottom=0.3)
line0, = ax.plot([0],[0],label="e*10 (V)")
line1, = ax.plot([0],[0],label="Phi*1000*gain (V*s)")
line2, = ax.plot([0],[0],label="i1*10 (A)")
ax.grid()
ax.axis([0,duree,-Umax,Umax])
ax.set_xlabel("t (s)")
ax.legend(loc="upper right")
axe_gain = axes([0.1,0.1,0.8,0.03])
gain = 1
slider_gain = Slider(axe_gain,"Gain flux (log)",-1,3,valinit=0)
def update_gain(val):
    global gain,ax,line1
    gain = 10**(slider_gain.val)
slider_gain.on_changed(update_gain)


fem = numpy.array([],dtype=numpy.float32)
flux = numpy.array([],dtype=numpy.float32)
t = numpy.array([],dtype=numpy.float32)
courant = numpy.array([],dtype=numpy.float32)

def animate(i):
    global ax,sys,fem,flux,courant,t,line0,n_tot,longueur_bloc,zi1,zi2,zi3,Umax,zero,
    data = sys.paquet(n_tot,reduction)
    u0 = data[4] # fem filtrée
    u2 = data[5] # courant primaire filtré
    [u1,zi1] = scipy.signal.lfilter(b1,a1,u0/G,zi=zi1)
    u1 = u1
    n_tot += u0.size
    fem = numpy.append(fem,u0)
    flux = numpy.append(flux,u1)
```

```
    courant = numpy.append(courant,u2/R)
    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],fem[i1:i2])
    line1.set_data(t[i1:i2],flux[i1:i2]*1000*gain)
    line2.set_data(t[i1:i2],courant[i1:i2]*10)
    ax.axis([t[i1],t[i2],-Umax,Umax])


ani = animation.FuncAnimation(fig,animate,frames=nombre_blocs,repeat=True,interval=du
show(block=True)
sys.stopper_acquisition()
time.sleep(1)
numpy.savetxt("data-1.txt",[t,fem,flux,courant])
sys.fermer()
```

Voici une copie de la fenêtre de tracé des signaux :