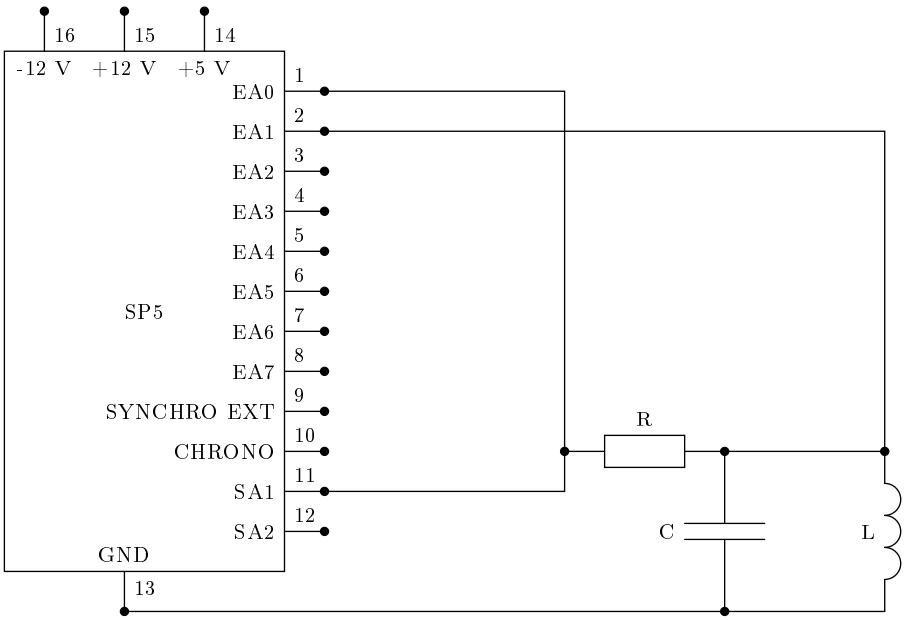
Filtre et oscillateur LC

1. Filtre LC passe-bande

1.a. Circuit d'étude

Pour obtenir le diagramme de Bode de la réponse fréquentielle, le filtre LC est branché sur le convertisseur analogique/numérique SysamSP5.



Les valeurs utilisées sont $R = 1.0\ k\Omega$, $C = 1.0\ \mu F$ et $L = 47\ mH$.

1.b. Programme d'acquisition

Le convertisseur Eurosmart SysamSP5 est piloté avec le module pycan.

```
import pycan.main as pycan
import numpy
import math
import time
```

La fonction suivante effectue l'analyse du filtre à une fréquence donnée, pour une amplitude choisie de la sinusoïde d'entrée. Elle renvoit le gain en décibel, le cosinus du déphasage et le déphasage.

```
def analyser(sys,f,a):
    print("Frequence = %f"%f)
    ne = 50
    T=1.0/f
```

```
    te=T/ne
    v1 = numpy.zeros(ne,numpy.double)
    for k in range(ne):
        phi = 2*math.pi/ne*k
        v1[k] = a*math.sin(phi)
    sys.config_sortie(1,te*10**6,v1,-1)
    sys.config_entrees([0,1],[10.0,10.0])
    np=100 # nombre de périodes de l'acquisition
    nea = ne*np
    duree = nea*te
    sys.config_echantillon(te*10**6,nea)
    sys.declencher_sorties(1,0)
    time.sleep(1) # régime transitoire
    sys.acquerir() # première acquisition pour connaitre l'amplitude de la sortie
    sys.stopper_sorties(1,0)
    t=sys.temps()
    u=sys.entrees()
    max1 = u[1].max()
    sys.config_entrees([0,1],[10.0,max1]) # ajustement du gain pour la sortie
    sys.config_echantillon(te*10**6,nea)
    sys.declencher_sorties(1,0)
    time.sleep(1) # régime transitoire
    sys.acquerir()
    sys.stopper_sorties(1,0)
    t=sys.temps()
    u=sys.entrees()
    mu0 = numpy.mean(u[0])
    mu1 = numpy.mean(u[1])
    rms0 = numpy.std(u[0])
    rms1 = numpy.std(u[1])
    g = rms1/rms0
    gdb = 20.0*math.log10(g)
    print("GdB = %f"%(gdb))
    cosphi = numpy.mean((u[0]-mu0)*(u[1]-mu1))/(rms0*rms1)
    print("cosphi = %f"%cosphi)
    phi = math.acos(cosphi)*180.0/math.pi
    print("phi = %f deg"%phi)
    return gdb,cosphi,phi
```

On ouvre l'interface avec le SysamSP5, on génère le tableau des fréquences et on effectue la boucle de calcul. Les résultats sont sauvegardés dans un fichier.

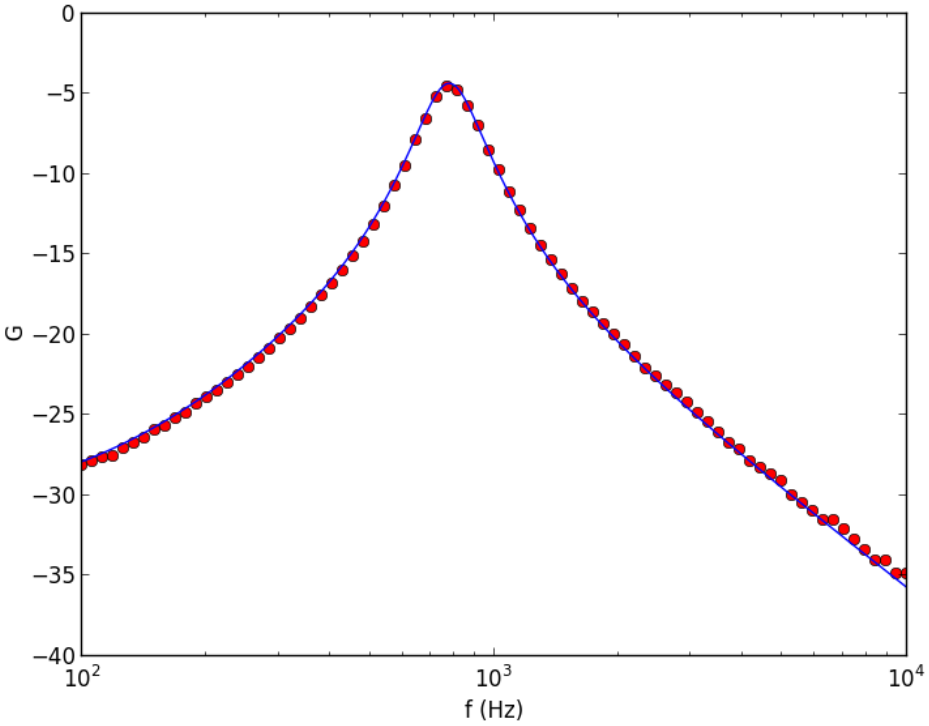
```
sys = pycan.Sysam("SP5")
freq = numpy.logspace(start=2,stop=4,num=80)
gdb = numpy.zeros(freq.size,dtype=numpy.float32)
cosphi = numpy.zeros(freq.size,dtype=numpy.float32)
phi = numpy.zeros(freq.size,dtype=numpy.float32)
amplitude = 1.0 # à choisir en fonction du filtre, pour ne pas dépasser 10 V en sorti
```

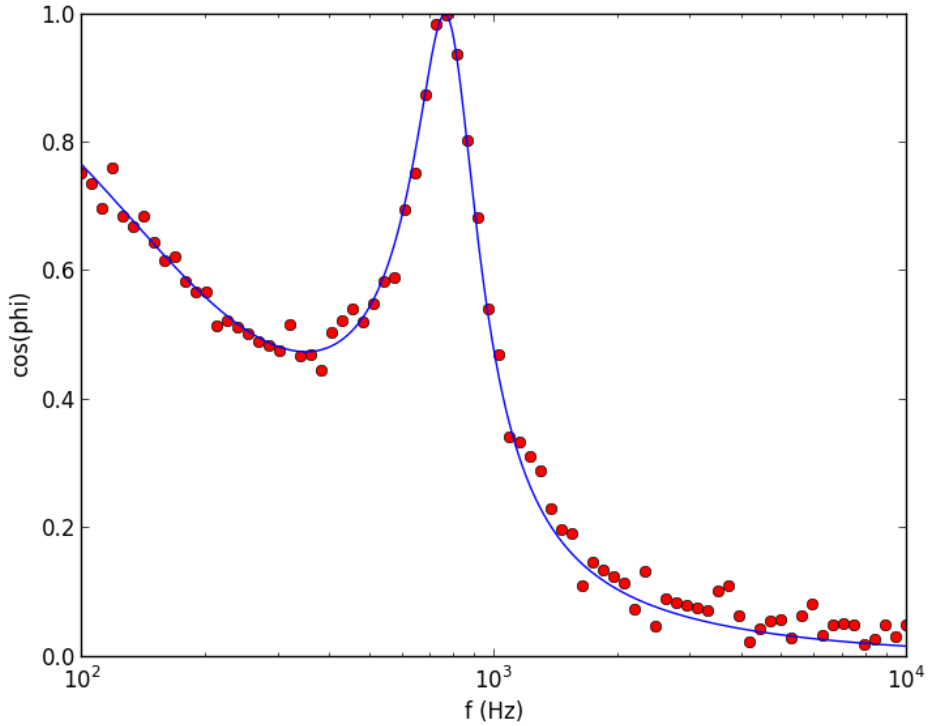
```
for k in range(freq.size):
    gdb[k],cosphi[k],phi[k] = analyser(sys,freq[k],amplitude)
sys.fermer()
numpy.savetxt("filtrePasseBande.txt",[freq,gdb,cosphi])
```

1.c. Diagramme de Bode

Les points expérimentaux sont représentés sous forme d'un diagramme de Bode pour le gain en décibel et le cosinus du déphasage. Une fonction de transfert est définie en ajoutant une résistance interne r à l'inductance. Les paramètres L,C et r sont ajustées pour obtenir la superposition des courbes avec les points expérimentaux.

```
import numpy as np
import math
import matplotlib.pyplot as plt
import cmath

[freq,gdb,cosphi]=np.loadtxt("filtrePasseBande.txt")
R=1000
C=0.98e-6
L=44e-3
r=30
def H(f): # fonction de transfert du filtre
    w=2*math.pi*f
    z=1.0/(1.0/(1j*L*w+r)+1j*C*w)
    return z/(R+z)
def GdB(f):
    return 20*np.log10(np.absolute(H(f)))
def CosPhi(f):
    return math.cos(cmath.phase(H(f)))
ufunc_cosphi = np.frompyfunc(CosPhi,1,1)
freq_theo = np.logspace(start=2,stop=4,num=1000)
gdb_theo = GdB(freq_theo)
cosphi_theo = ufunc_cosphi(freq_theo)
plt.figure()
plt.semilogx(freq,gdb,'ro')
plt.semilogx(freq_theo,gdb_theo)
plt.xlabel("f (Hz)")
plt.ylabel("G")
plt.figure()
plt.semilogx(freq,cosphi,'ro')
plt.semilogx(freq_theo,cosphi_theo)
plt.xlabel("f (Hz)")
plt.ylabel("cos(phi)")
plt.show()
```

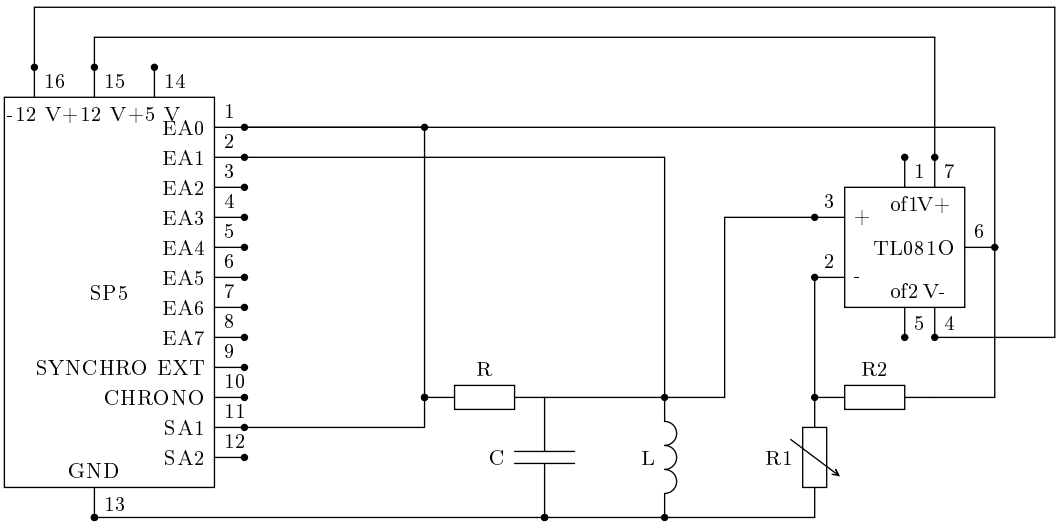

0
−5
−10
−15
−20
−25
−30
−35
−40
G
10^2
10^3
10^4
f (Hz)


1.0
0.8
0.6
0.4
0.2
0.0
cos(phi)
10^2
10^3
10^4
f (Hz)

2. Oscillateur LC

2.a. Circuit d'étude

L'oscillateur est construit à partir du filtre précédent. Pour cela, la sortie du filtre est amplifiée pour être appliquée à l'entrée. Le gain de l'amplificateur doit compenser le gain maximal du filtre, soit -4 dB. Il doit donc avoir un gain G=1.6 environ. Un potentiomètre permet de faire un réglage fin du gain.



2.b. Programme d'analyse

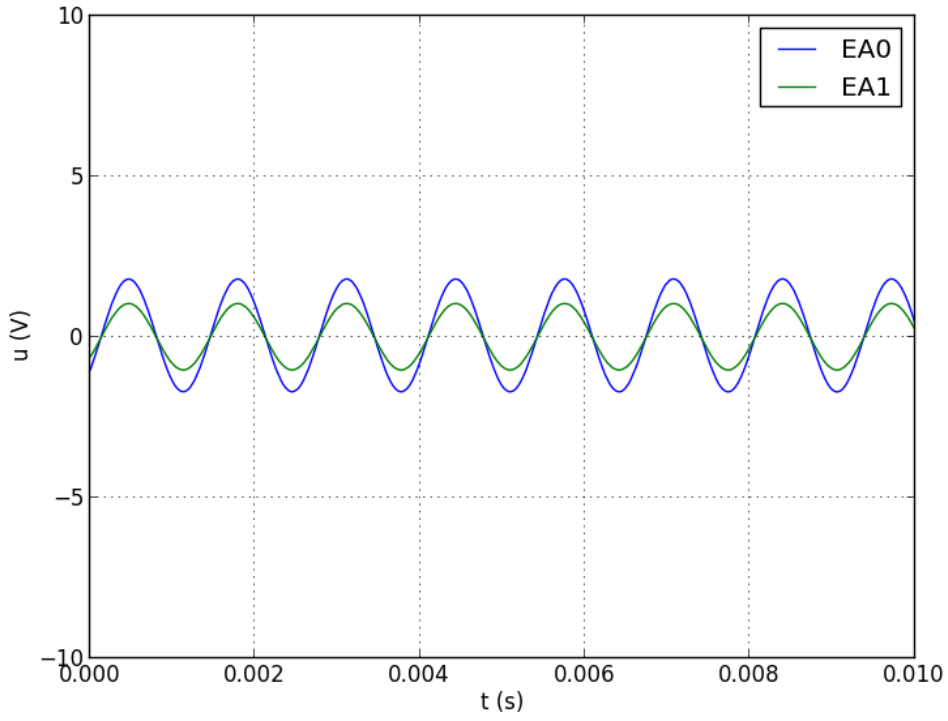
Le programme effectue une acquisition de l'entrée (voie EA0) et de la sortie (voieEA1) du filtre. L'analyse spectrale de la sortie est effectuée.

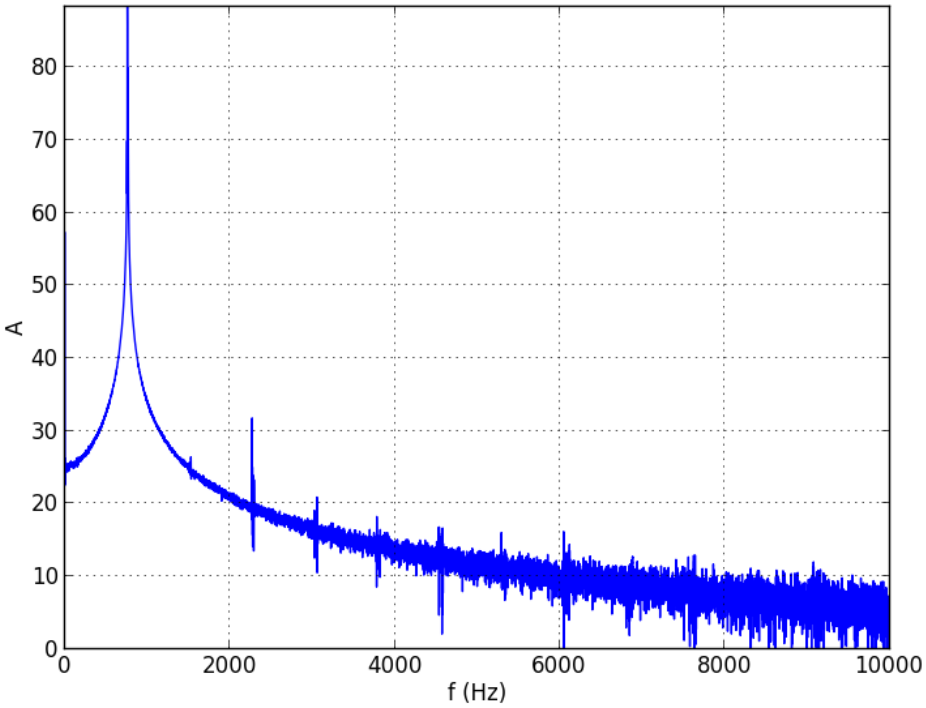
```
import pycan.main as pycan
import numpy
import math
import time
import matplotlib.pyplot as plt
import numpy.fft

sys = pycan.Sysam("SP5")
te=10.0e-6
ne=100000
duree=te*ne
sys.config_entrees([0,1],[10.0,10.0])
sys.config_echantillon(te*1e6,ne)
sys.acquerir()
t=sys.temps()
u=sys.entrees()
sys.fermer()
tfd=numpy.fft.fft(u[1])
f=numpy.arange(tfd.size)
for k in range(f.size):
```

```
    f[k] = 1.0*k/duree
spectre=20*numpy.log10(numpy.absolute(tfd))
plt.figure()
plt.plot(t[0],u[0],label="EA0")
plt.plot(t[1],u[1],label="EA1")
plt.xlabel("t (s)")
plt.ylabel("u (V)")
plt.axis([0,0.01,-10.0,10.0])
plt.grid()
plt.legend()
plt.figure()
plt.plot(f,spectre)
plt.axis([0,10000.0,0,numpy.amax(spectre)])
plt.grid()
plt.xlabel("f (Hz)")
plt.ylabel("A")
plt.show()
```

2.c. Analyse des oscillations

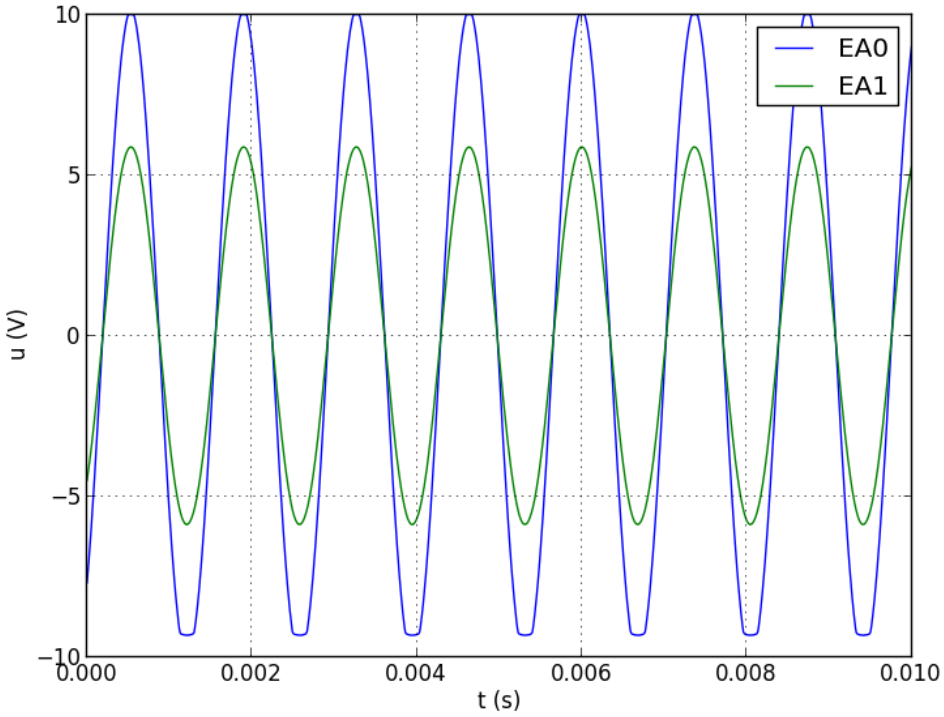
Dans le premier exemple, on ajuste le gain de l'amplificateur pour obtenir des oscillations sinusoïdales en sortie de l'amplificateur (en entrée du filtre).

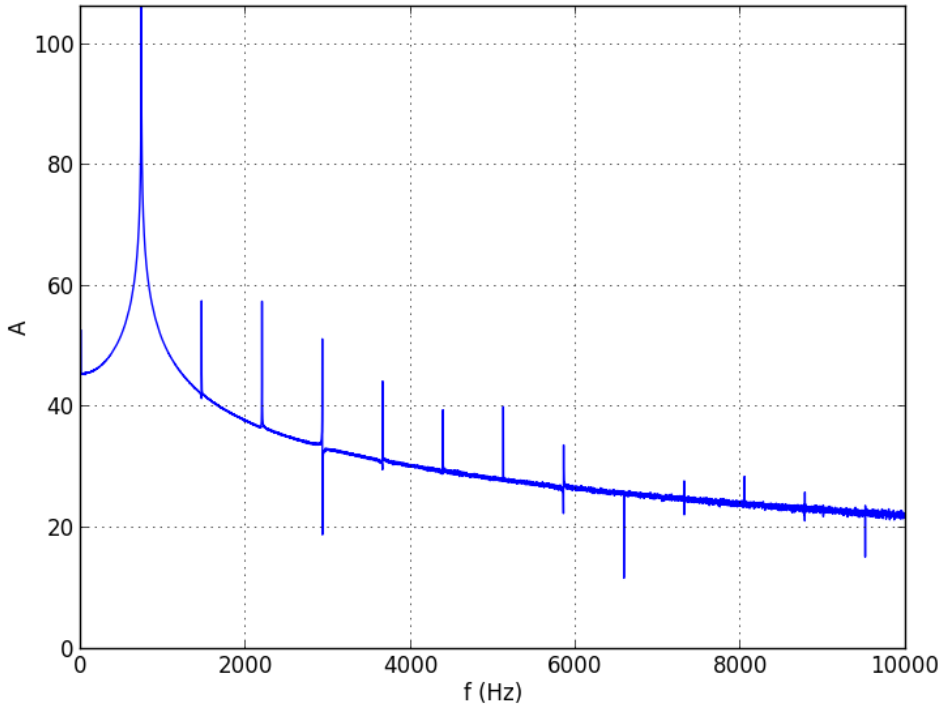




Le spectre montre une oscillation pratiquement harmonique. On observe une harmonique de rang 3 d'amplitude -60 dB par rapport au fondamental.

Dans le deuxième exemple, on ajuste le gain pour obtenir une saturation en sortie de l'amplificateur.





Ce réglage a l'avantage d'être plus stable que le précédent. Le facteur de qualité relativement élevé du filtre (environ 4,5) permet d'obtenir une oscillation pratiquement sinusoïdale en sortie du filtre, avec un taux de distortion inférieur à -50 dB.