Échantillonnage et analyse spectrale

1. Introduction

Ce document présente des échantillonnages de signaux effectués avec un convertisseur analogique-numérique Eurosmart SysamSP5. Le spectre des signaux échantillonnés sera obtenu selon la méthode expliquée dans le document Introduction à l'analyse spectrale. On verra l'importance du respect de la condition de Nyquist-Shannon pour l'obtention d'un spectre correct, et l'influence de la largeur de la fenêtre d'analyse. Voir à ce sujet le document Échantillonnage et reconstruction d'un signal périodique.

Les signaux sont générés avec le logiciel Pure data, qui permet de synthétiser des sons en utilisant un langage de programmation visuel. La sortie de la carte son de l'ordinateur est envoyée sur l'entrée EA0 de la centrale SP5 (on utilise un seul canal audio).

L'acquisisition des signaux et leur traitement sont fait avec Python, en utilisant le module d'interface pour Sysam SP5 présenté dans CAN Eurosmart : interface pour Python.

2. Programme Python

Le programme effectue l'acquisition du signal et l'enregistre dans un fichier. Il effectue aussi le tracé de la forme d'onde et du spectre.

echantillonnage.py

```
import pycanum.main as pycan
from matplotlib.pyplot import *
import numpy
import math
import numpy.fft
import os

os.chdir("C:/Users/fred/Documents/electro/TP/EchantillonnageTFD")
```

On donne le nom de l'expérience, qui sera utilisé pour les fichiers :

```
nom = "sinus-1"
```

On comme par ouvrir l'interface avec la centrale SP5 :

```
can = pycan.Sysam("SP5")
```

On configure la voie 0 avec un calibre de 2 volts, car le maximum délivré par la sortie audio est d'environ 1.6 V.

```
can.config_entrees([0],[2.0])
```

On fixe la fréquence d'échantillonnage et la durée de l'acquisition, dont on déduit la période d'échantillonnage et le nombre de points :

```
fe=1000.0
T=2.0
te=1.0/fe
N = int(T/te)
print(N)
```

Avec la centrale SP5, le nombre de points lorsqu'on utilise une seule voie (et pas de sortie) peut aller jusqu'à 262142. C'est largement plus que la capacité mémoire des oscilloscopes numériques ordinaires (10000 points), ce qui rend la centrale SP5 intéressante pour faire des analyse spectrales très fines.

On configure la période d'échantillonnage (en microsecondes) et le nombre de points. La fréquence d'échantillonnage maximale est 10 MHz.

```
can.config_echantillon(te*10**6,N)
```

On effectue l'acquisition puis on récupère le tableau des instants et celui des tensions. Enfin, on ferme l'interface :

```
can.acquerir()
t0=can.temps()[0]
u0=can.entrees()[0]
can.fermer()
```

On enregistre ces données dans un fichier texte pour les traiter plus tard :

```
numpy.savetxt('%s.txt'%nom,[t0,u0])
```

On récupère la période d'échantillonnage et le nombre de points effectifs, car ceux-ci peuvent avoir été légèrement modifiés par le pilote :

```
te = t0[1]-t0[0]
fe = 1.0/te
N = t0.size
T = t0[N-1]-t0[0]
```

Pour finir, le programme trace le signal sur quelques périodes :

```
figure()
plot(t0,u0,'b')
xlabel("t (s)")
ylabel("u (V)")
axis([0,0.5,-2,2])
grid()
savefig("%s-signal.pdf"%nom)
```

puis calcule et trace le spectre :

```
tfd=numpy.fft.fft(u0)
a =numpy.absolute(tfd)/N
f=numpy.arange(N)*1.0/T
figure()
plot(f,a)
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,a.max()])
grid()
savefig("%s-spectre.pdf"%nom)

show()
```

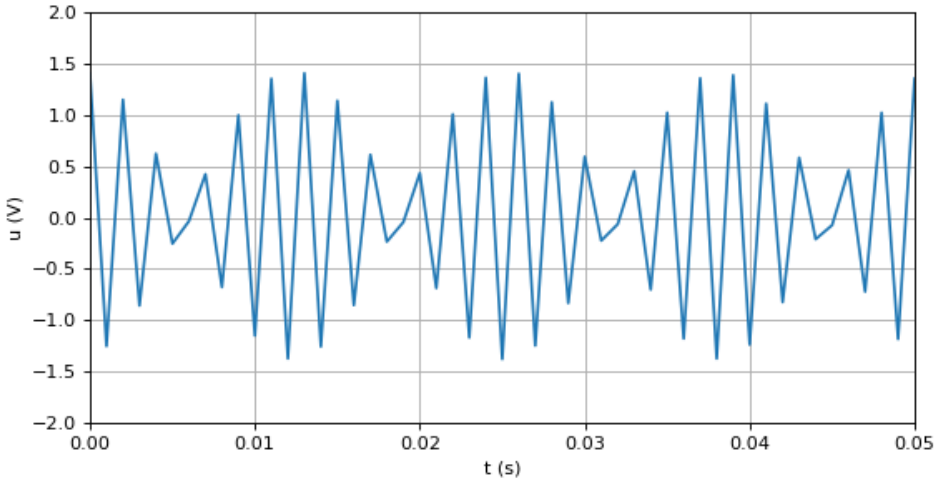
3. Sinusoïde

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

On commence par le cas le plus simple, un son sinusoïdal. Sa fréquence est 461,4 Hz. Pour respecter la condition de Nyquist-Shannon, il faut donc une fréquence d'échantillonnage d'au moins 923 Hz.

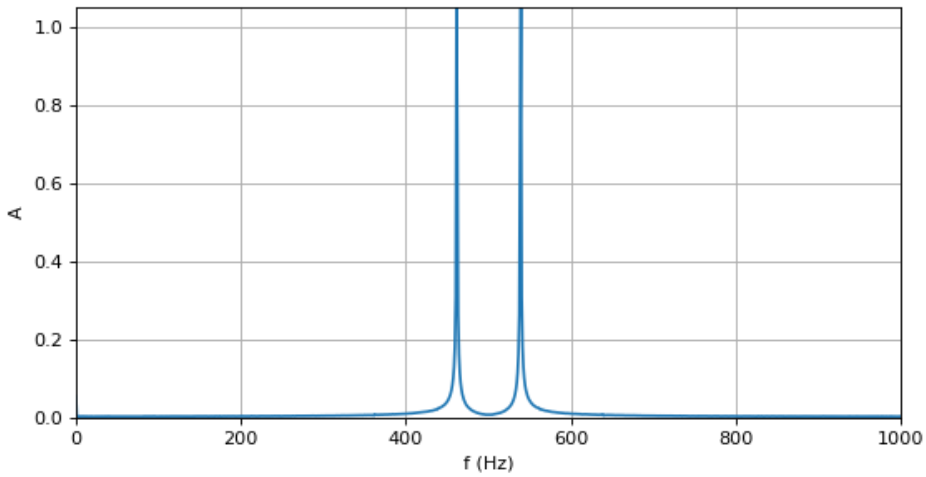
La durée de l'acquisistion est $T = 1\,\text{s}$, qui donnera une précision de 1 Hz sur le spectre. On commence par une fréquence d'échantillonnage juste au dessus de $2f_{max}$, égale à 1000 Hz :

```
[t,u] = numpy.loadtxt('sinus-1.txt')
figure(figsize=(8,4))
plot(t,u)
xlabel('t (s)')
ylabel('u (V)')
axis([0,0.05,-2,2])
grid()
```



Pour ce graphique, on s'est contenté de relier les échantillons par des segments, ce qui donne une très mauvaise représentation de la sinusoïde. Voyons le spectre du signal discret :

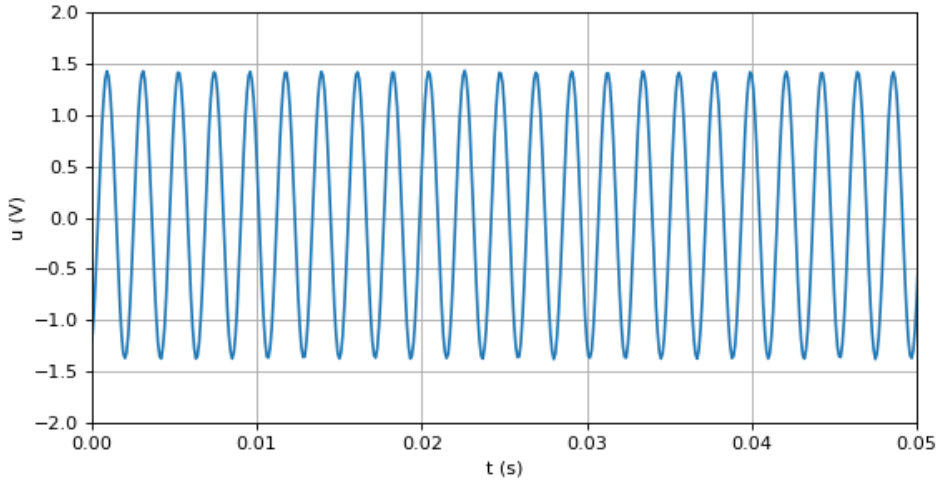
```
N = t.size
T = t[N-1]-t[0]
te = t[1]-t[0]
fe = 1.0/te
tfd=numpy.fft.fft(u)*2/N
a =numpy.absolute(tfd)
f=numpy.arange(N)*1.0/T
figure(figsize=(8,4))
plot(f,a)
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,a.max()])
grid()
```



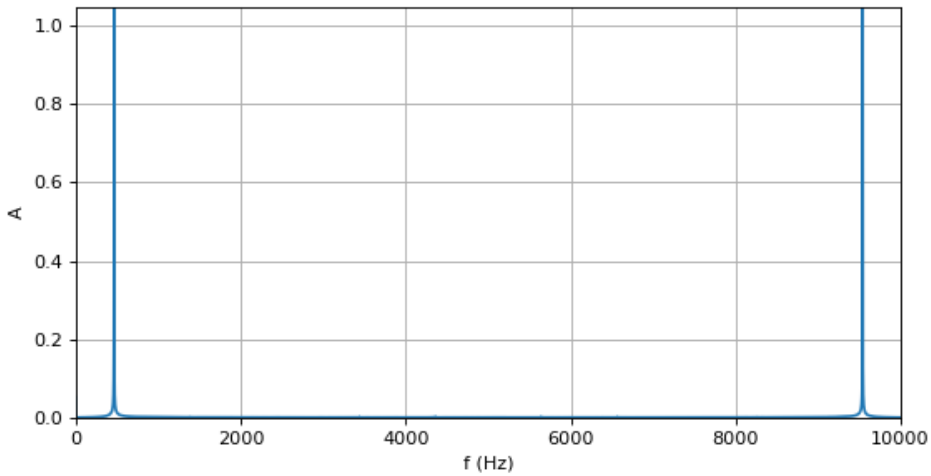
Il y a une raie à 461 Hz et son image à 1000-461=539 Hz, ce qui confirme que la condition de Nyquist-Shannon a bien été respectée. Ici la fréquence de Nyquist ($f_e/2$) est juste au dessus de la fréquence du signal. Même si cela rend difficile le tracé temporel, l'analyse spectrale est correcte.

Pour obtenir une représentation temporelle correcte, le plus simple est d'effectuer un sur-échantillonnage (lorsque cela est possible). Voyons le résultat pour une fréquence d'échantillonnage de 10000 Hz :

```
[t,u] = numpy.loadtxt('sinus-2.txt')
figure(figsize=(8,4))
plot(t,u)
xlabel('t (s)')
ylabel('u (V)')
axis([0,0.05,-2,2])
grid()
```



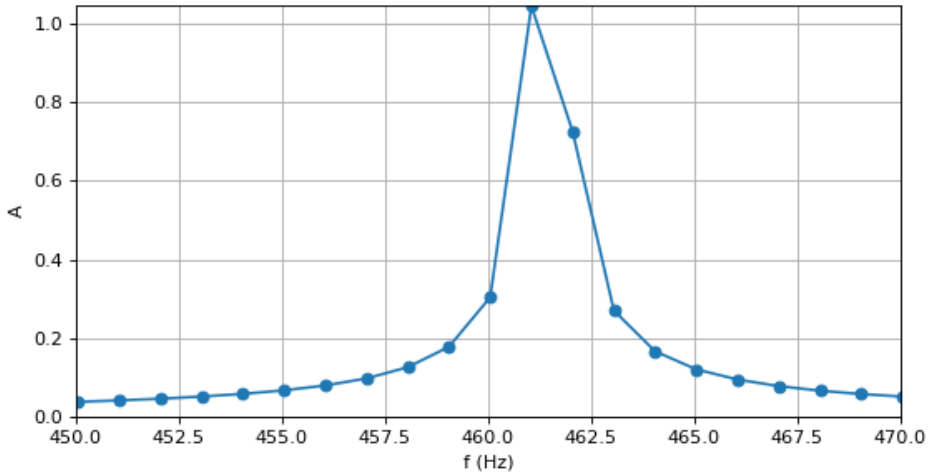

```
N = t.size
T = t[N-1]-t[0]
te = t[1]-t[0]
fe = 1.0/te
tfd=numpy.fft.fft(u)*2/N
a =numpy.absolute(tfd)
f=numpy.arange(N)*1.0/T
figure(figsize=(8,4))
plot(f,a)
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,a.max()])
grid()
```



Le sur-échantillonnage facilite le tracé de la forme d'onde mais n'apporte rien pour l'analyse spectrale (la résolution fréquentielle est toujours $1/T$).

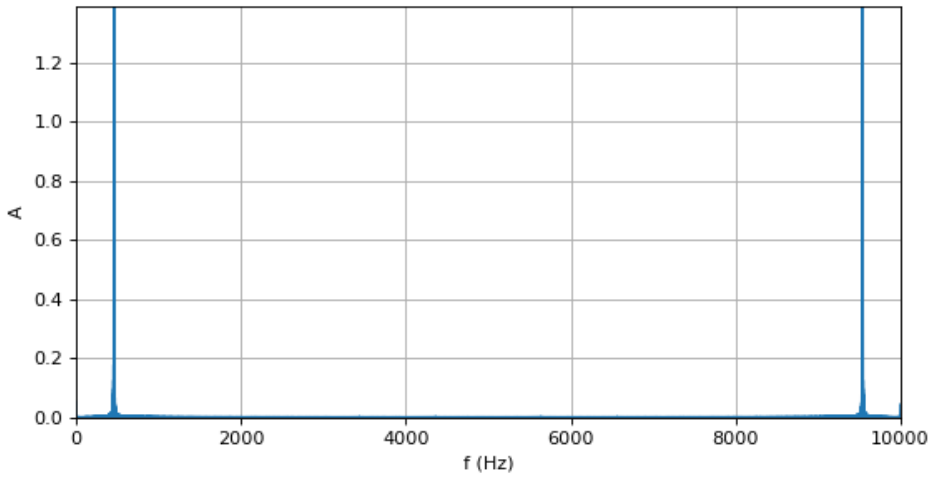
On remarque que la hauteur de la raie est inférieure à l'amplitude de la sinusoïde. Pour comprendre cette anomalie, examinons la raie en détail :

```
figure(figsize=(8,4))
plot(f,a,"-o")
xlabel("f (Hz)")
ylabel("A")
axis([450,470,0,a.max()])
grid()
```



Le maximum de la raie ne semble pas coïncider avec une des fréquences du spectre discret. Cela induit une erreur au plus égale à 1/T sur la fréquence de la raie, mais l'erreur sur son amplitude est très grande car les variations sont très rapides au voisinage du maximum. Pour un fenêtrage rectangulaire, la forme théorique de la raie est un sinus-cardinal. Un moyen très simple de reconstruire la forme complète des raies est d'ajouter des zéros au signal (technique du remplissage par des zéros), avant de calculer sa transformée de Fourier discrète :

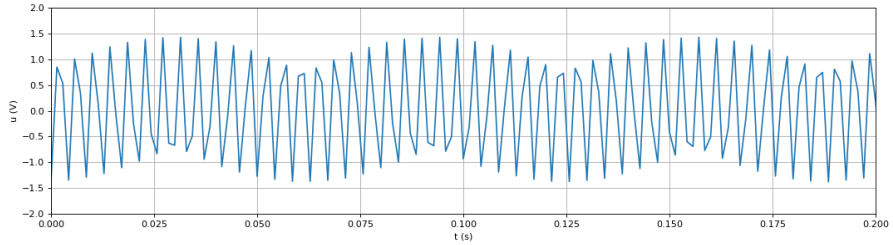
```
zeros=numpy.zeros(2*N)
u=numpy.concatenate((u,zeros))
Nz=len(u)
tfd=numpy.fft.fft(u)*2/N
a =numpy.absolute(tfd)
f=numpy.arange(Nz)*1.0/(Nz*te)
figure(figsize=(8,4))
plot(f,a)
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,a.max()])
grid()
```



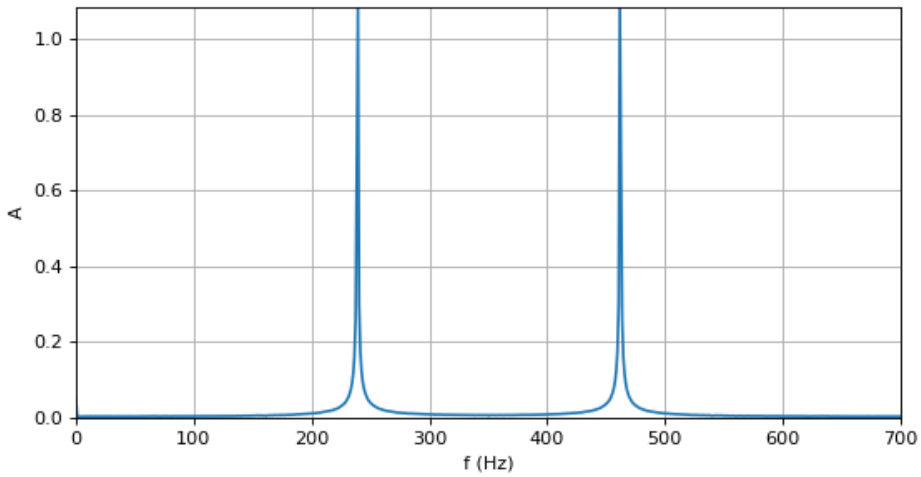
Avec la méthode de l'extension de la fenêtre avec remplissage par des zéros, la hauteur de la raie est correcte.

Le dernier cas est celui d'un sous-échantillonnage, c'est-à-dire une fréquence d'échantillonnage qui ne respecte pas la condition de Nyquist-Shannon, ici 700 Hz :

```
[t,u] = numpy.loadtxt('sinus-3.txt')
figure(figsize=(16,4))
plot(t,u)
xlabel('t (s)')
ylabel('u (V)')
axis([0,0.2,-2,2])
grid()
```



```
N = t.size
T = t[N-1]-t[0]
te = t[1]-t[0]
fe = 1.0/te
tfd=numpy.fft.fft(u)*2/N
a =numpy.absolute(tfd)
f=numpy.arange(N)*1.0/T
figure(figsize=(8,4))
plot(f,a)
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,a.max()])
grid()
```



La raie à 461 Hz est toujours présente mais se retrouve dans la partie droite du spectre, au dessus de la fréquence de Nyquist. La raie de gauche est en fait son image, de fréquence 700-461=239 Hz. La raie image s'est retrouvée dans la première moitié : c'est le phénomène de repliement de spectre. Cela signifie que si l'on tente de reconstituer le signal à partir de ces échantillons, on obtiendra une sinusoïde de fréquence 239 Hz, bien inférieure à celle du signal de départ. Cette ondulation de basse fréquence est visible sur la représentation temporelle.

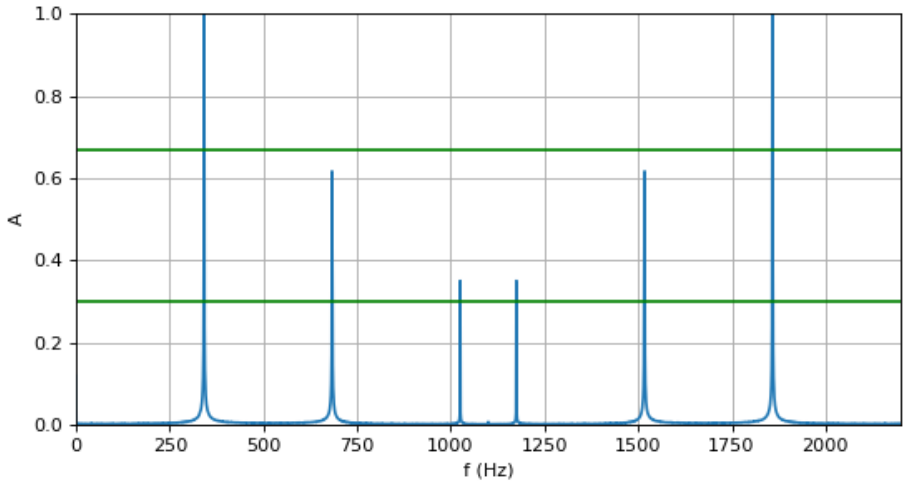
4. Signal périodique

Le signal audio est généré par le programme Pure Data syntheseHarmonique.pd. Il comporte 3 harmoniques, de rang 1,2 et 3. Sa fréquence fondamentale est 341,7 Hz. Les amplitudes appliquées aux harmoniques sont 0,314, 0,212 et 0,094.

La plus grande fréquence est celle de l'harmonique de rang 3, soit $f_{max} = 1025$ Hz. La fréquence d'échantillonnage doit donc être supérieure à 2050 Hz.

Voici le résultat pour $T = 1$ s et $f_e = 2200$ Hz. Pour faciliter la lecture du spectre, on le normalise par la valeur maximal.

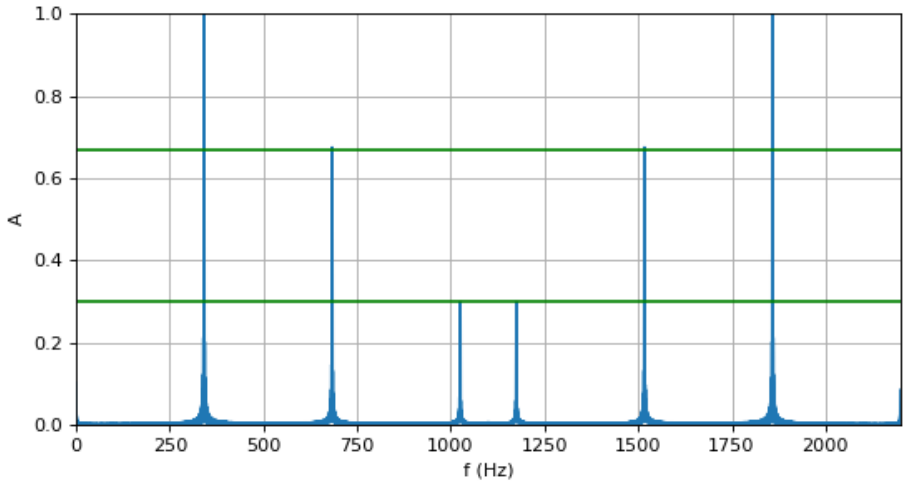
```
[t,u] = numpy.loadtxt('harmo-1.txt')
N = t.size
T = t[N-1]-t[0]
te = t[1]-t[0]
fe = 1.0/te
tfd=numpy.fft.fft(u)*2/N
a =numpy.absolute(tfd)
f=numpy.arange(N)*1.0/(N*te)
figure(figsize=(8,4))
plot(f,a/a.max())
plot([0,fe],[0.67,0.67],'g')
plot([0,fe],[0.30,0.30],'g')
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,1.0])
grid()
```



On obtient bien sur la première moitié ([0,1100] Hz) le spectre constitué de trois raies. La position des raies est correcte (au Hz près puisque $T = 1\,\text{s}$). Voyons la hauteur relative des raies. L'harmonique de rang 2 est attendu avec une hauteur relative au fondamental 0,212/0,314=0,67. Pour l'harmonique de rang 3, le rapport est 0,094/0,314=0,30. Ces valeurs ont été placées sur la figure sous forme d'un trait horizontal. On constate que la détermination de la hauteur des raies sur le spectre est grossière. Pour améliorer la précision de la hauteur des raies, on procède comme plus haut par ajout de zéros avant de calculer le spectre :

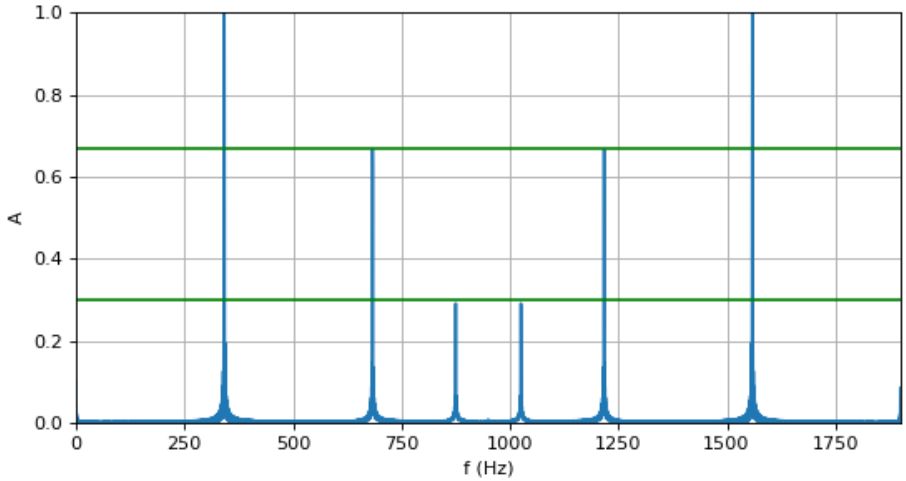
```
zeros=numpy.zeros(2*N)
u=numpy.concatenate((u,zeros))
Nz=len(u)
tfd=numpy.fft.fft(u)*2/N
a =numpy.absolute(tfd)
f=numpy.arange(Nz)*1.0/(Nz*te)
figure(figsize=(8,4))
```

```
plot(f,a/a.max())
plot([0,fe],[0.67,0.67],'g')
plot([0,fe],[0.30,0.30],'g')
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,1.0])
grid()
```



Voyons l'effet d'un sous-échantillonnage, avec $f_e = 1900\,\text{Hz}$ et $T = 1\,\text{s}$:

```
[t,u] = numpy.loadtxt('harmo-3.txt')
N = t.size
T = t[N-1]-t[0]
te = t[1]-t[0]
fe = 1.0/te
zeros=numpy.zeros(2*N)
u=numpy.concatenate((u,zeros))
Nz=len(u)
tfd=numpy.fft.fft(u)*2/N
a =numpy.absolute(tfd)
f=numpy.arange(Nz)*1.0/(Nz*te)
figure(figsize=(8,4))
plot(f,a/a.max())
plot([0,fe],[0.67,0.67],'g')
plot([0,fe],[0.30,0.30],'g')
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,1.0])
grid()
```



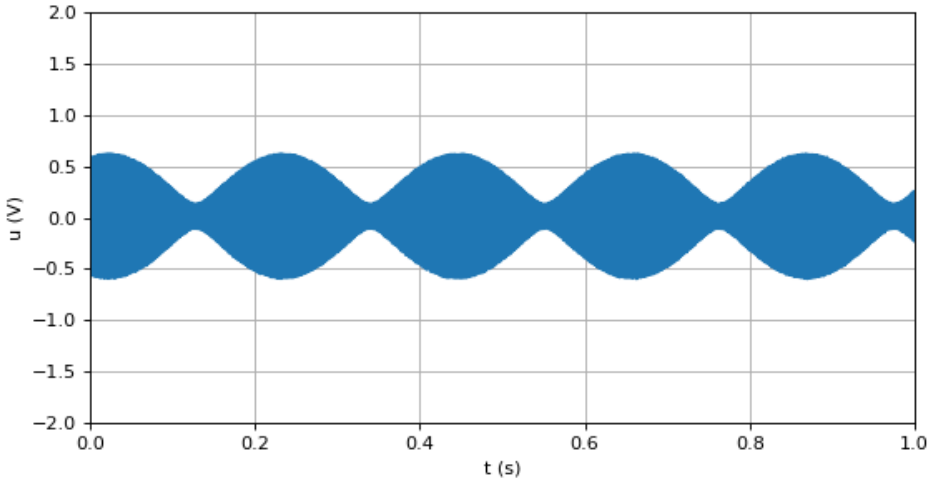
Les positions des raies 1 et 2 sont correctes. On voit immédiatement que la troisième raie n'est pas à la bonne fréquence. Il y a eu repliement de spectre : l'image de la troisième raie se retrouve dans la première moitié du spectre, à une fréquence 1900-1026=874 Hz au lieu de 1026 Hz. Dans le cas présent, le repliement de spectre affecte seulement du troisième harmonique. Le résultat est d'ailleurs un signal non périodique.

5. Superposition de deux fréquences voisines

Le signal est la somme de deux sinusoïdes de fréquences voisines. Le programme Pure Data battements.pdf génère ce signal. La première sinusoïde a une fréquence de 391,9 Hz et une amplitude de 0,377. La seconde a une fréquence de 396,6 Hz et une amplitude de 0,251. Le programme Pure Data permet de faire un calibrage de la tension en sortie, mais on s'intéresse surtout au rapport des amplitudes 0,251/0,377=0,666. L'écart entre les deux fréquences est 4,7 Hz.

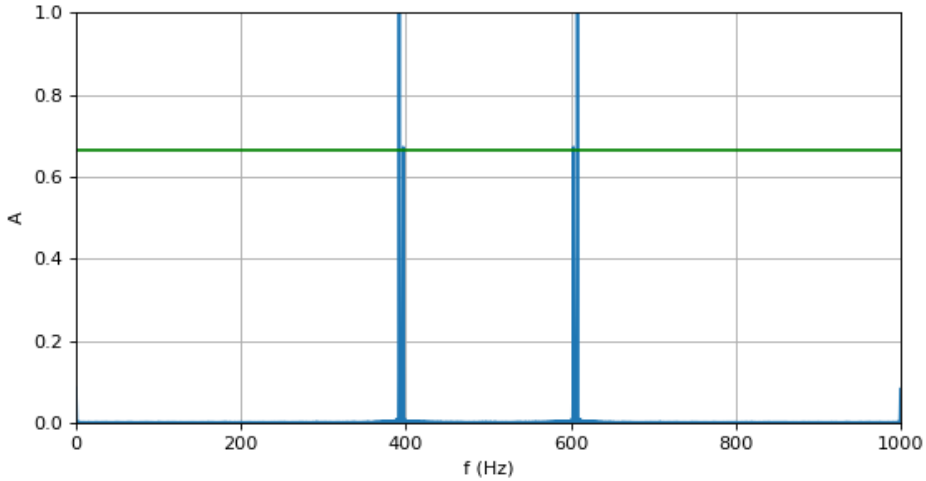
La fréquence d'échantillonnage doit être supérieure à 794 Hz. Voyons le résultat pour $f_e = 10000$ Hz (sur-échantillonnage) et $T = 2$ s :

```
[t,u] = numpy.loadtxt('batt-2.txt')
figure(figsize=(8,4))
plot(t,u)
xlabel('t (s)')
ylabel('u (V)')
axis([0,1.0,-2,2])
grid()
```



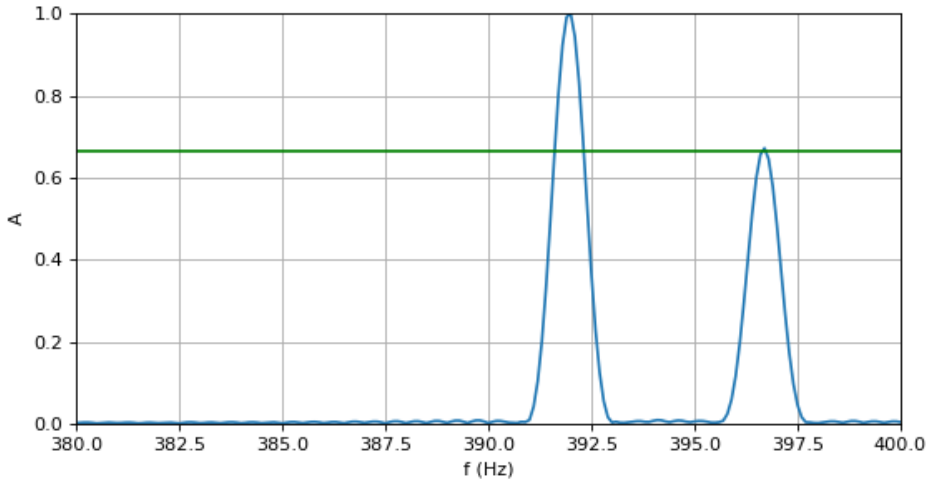
La superposition de deux fréquences voisines donne une oscillation dont l'amplitude est modulée : c'est le phénomène de battement. Voici le spectre, obtenu en appliquant un fenêtrage de Hamming puis en ajoutant des zéros :

```
[t,u] = numpy.loadtxt('batt-1.txt')
N = t.size
T = t[N-1]-t[0]
te = t[1]-t[0]
fe = 1.0/te
zeros=numpy.zeros(4*N)
u=numpy.concatenate((u*scipy.signal.hamming(N),zeros))
Nz=len(u)
tfd=numpy.fft.fft(u)*2/N/0.54
a =numpy.absolute(tfd)
f=numpy.arange(Nz)*1.0/(Nz*te)
figure(figsize=(8,4))
plot(f,a/a.max())
plot([0,fe],[0.666,0.666],'g')
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,1.0])
grid()
```



Les deux raies sont bien discernées, ce qui est logique puisque la durée T est supérieure à l'inverse de la différence de fréquence. Voyons le détail de ces deux raies :

```
figure(figsize=(8,4))
plot(f,a/a.max())
plot([0,fe],[0.666,0.666],'g')
xlabel("f (Hz)")
ylabel("A")
axis([380,400,0,1.0])
grid()
```

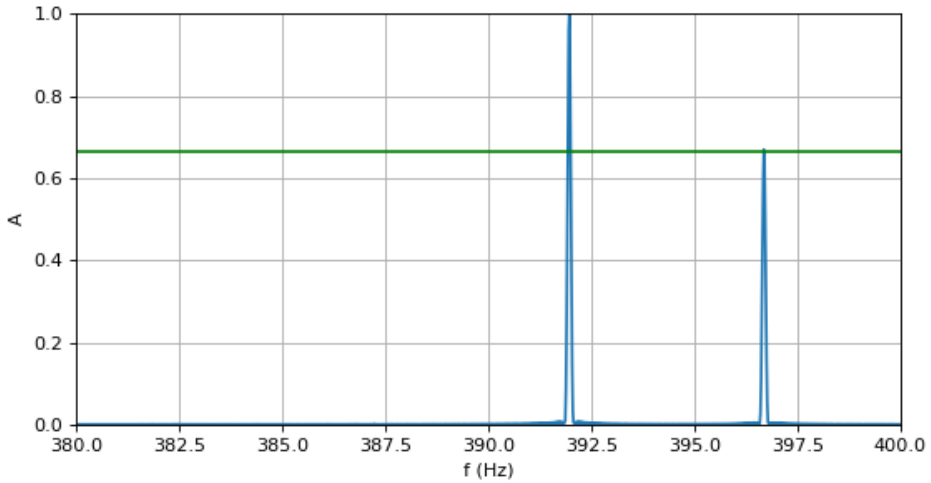


La résolution fréquentielle est de 1 Hz. Voyons le même spectre pour une durée $T = 20\ s$:

```
[t,u] = numpy.loadtxt('batt-6.txt')
N = t.size

T = t[N-1]-t[0]
te = t[1]-t[0]
```

```
fe = 1.0/te
zeros=numpy.zeros(2*N)
u=numpy.concatenate((u*scipy.signal.hamming(N),zeros))
Nz=len(u)
tfd=numpy.fft.fft(u)*2/N/0.54
a =numpy.absolute(tfd)
f=numpy.arange(Nz)*1.0/(Nz*te)
figure(figsize=(8,4))
plot(f,a/a.max())
plot([0,fe],[0.666,0.666],'g')
xlabel("f (Hz)")
ylabel("A")
axis([380,400,0,1.0])
grid()
```



La précision fréquentielle est à présent de 0,05 Hz.

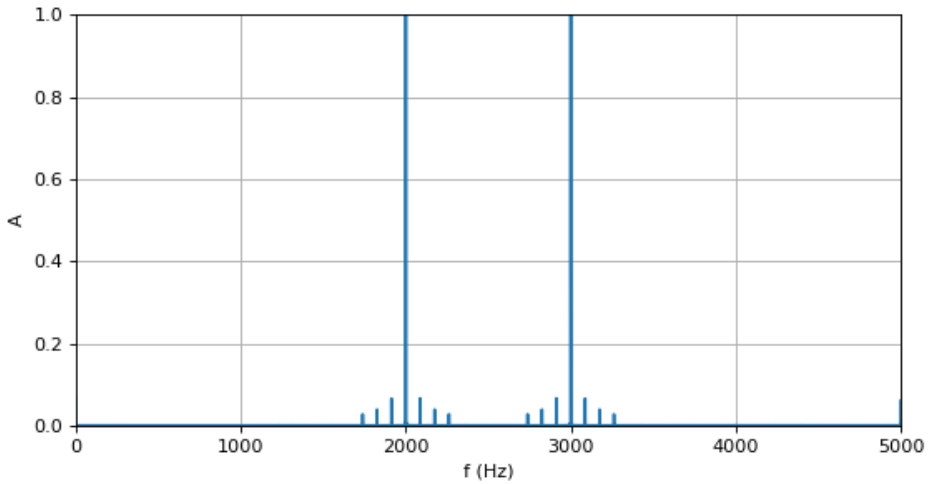
6. Modulation d'amplitude

Le programme Pure data modulationAmplitude.pd permet de générer une porteuse modulée en amplitude par un signal comportant jusqu'à 4 harmoniques. La porteuse a une fréquence de 2000 Hz. Le signal modulant a une fréquence fondamentale de 87.24 Hz. Il a 3 harmoniques, de rang 1,2 et 3, de rapports 1, 0,59, et 0,41. L'indice de modulation est 0,37 et l'amplitude de la porteuse 0,48.

La fréquence maximale est la somme de celle de la porteuse et de celle de l'harmonique le plus grand, soit 2000 + 262 Hz. Pour respecter la condition de Nyquist-Shannon, il faut donc une fréquence d'échantillonnage d'au moins 4520 Hz. Voyons le spectre avec $f_e = 5000$ Hz et $T = 20{,}0$ s, une durée assez longue pour avoir une bonne précision de la position des raies et de leur hauteur. Cela fait un total de 100000 points.

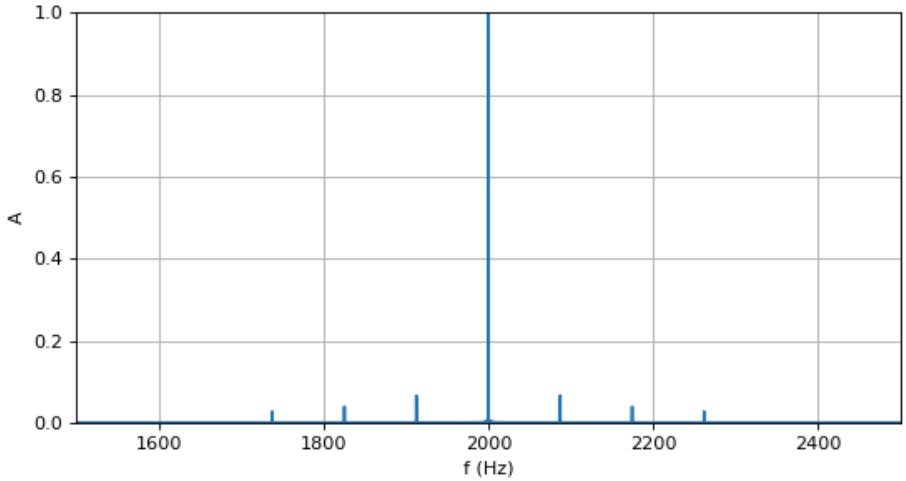
```
[t,u] = numpy.loadtxt('modAM-2.txt')
N = t.size
```

```
T = t[N-1]-t[0]
te = t[1]-t[0]
fe = 1.0/te
zeros=numpy.zeros(2*N)
u=numpy.concatenate((u*scipy.signal.hamming(N),zeros))
Nz=len(u)
tfd=numpy.fft.fft(u)*2/N/0.54
a =numpy.absolute(tfd)
f=numpy.arange(Nz)*1.0/(Nz*te)
figure(figsize=(8,4))
plot(f,a/a.max())
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,1.0])
grid()
```



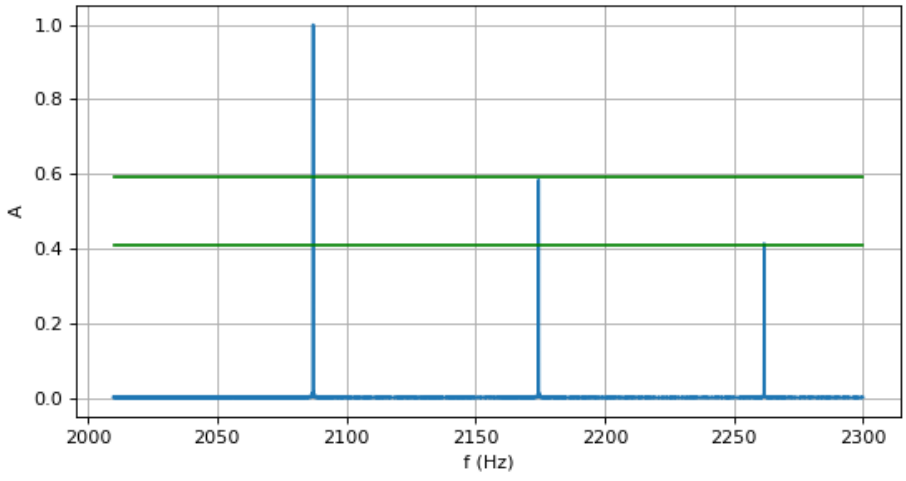
Le spectre du signal modulé en amplitude présente une raie centrale dont la fréquence est celle de la porteuse, et des raies latérales correspondant aux harmoniques du signal modulant. Voyons cela en détail :

```
figure(figsize=(8,4))
plot(f,a/a.max())
xlabel("f (Hz)")
ylabel("A")
axis([1500,2500,0,1.0])
grid()
```



On isole les trois raies à droite de la porteuse et on normalise par la raie du fondamental :

```
n1 = int(2010.0*Nz*te)
n2 = int(2300.0*Nz*te)
a2 = a[n1:n2]
f2 = f[n1:n2]
figure(figsize=(8,4))
plot(f2,a2/a2.max())
plot([2010,2300],[0.59,0.59],'g')
plot([2010,2300],[0.41,0.41],'g')
xlabel("f (Hz)")
ylabel("A")
grid()
```



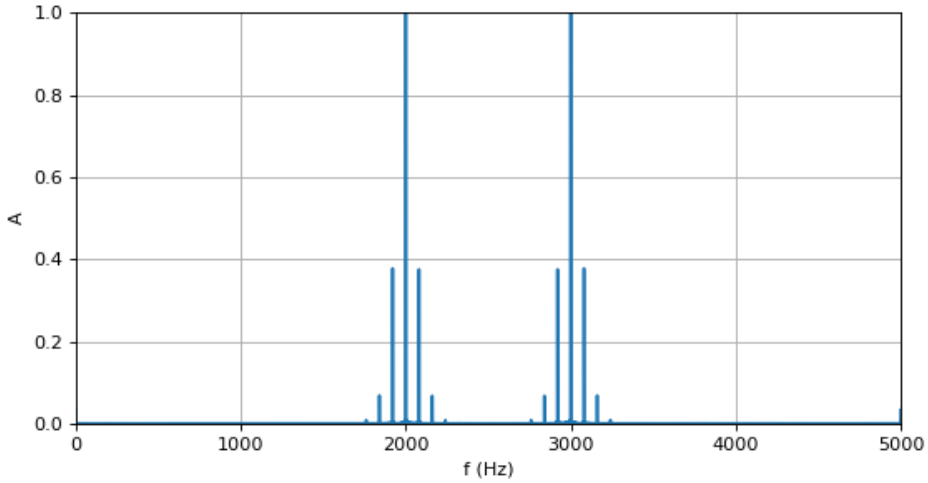
Les hauteurs relatives de ces raies sont bien celles des harmoniques définis au départ.

7. Modulation de fréquence

Le programme Pure data modulationFrequence.pd permet de générer une porteuse modulée en fréquence par un signal comportant jusqu'à 4 harmoniques. La porteuse a une fréquence de 2000 Hz. Le signal modulant est sinusoïdal, de fréquence 80,15 Hz, d'amplitude 0,448. On commence par un indice de modulation faible, égal à 0,06. Voici le résultat pour $f_e = 5000\,\text{Hz}$ et $T = 2{,}0\,\text{s}$:

```
[t,u] = numpy.loadtxt('modFM-1.txt')
N = t.size

T = t[N-1]-t[0]
te = t[1]-t[0]
fe = 1.0/te
zeros=numpy.zeros(2*N)
u=numpy.concatenate((u*scipy.signal.hamming(N),zeros))
Nz=len(u)
tfd=numpy.fft.fft(u)*2/N/0.54
a =numpy.absolute(tfd)
f=numpy.arange(Nz)*1.0/(Nz*te)
figure(figsize=(8,4))
plot(f,a/a.max())
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,1.0])
grid()
```

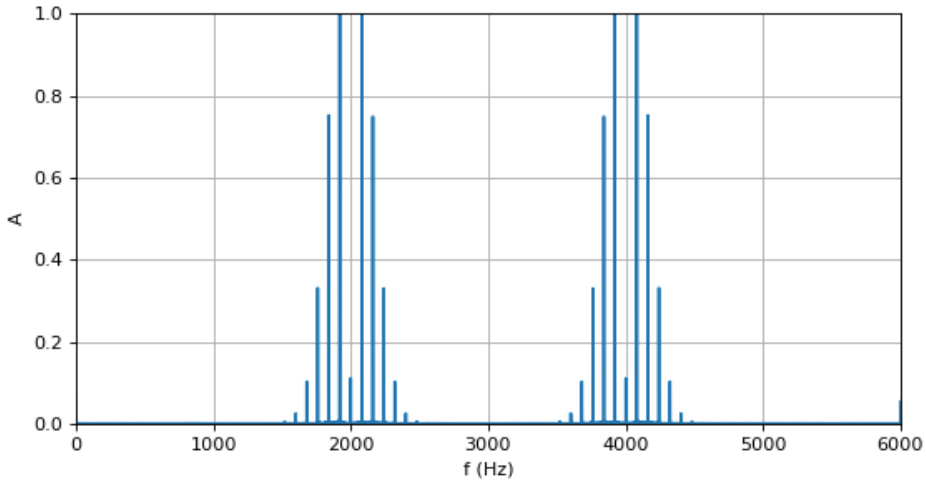


Le spectre est constitué d'une raie centrale dont la fréquence est celle de la porteuse, et de plusieurs raies latérales, bien que le signal modulant n'en comporte qu'une. Le spectre est donc plus complexe que celui d'une modulation d'amplitude. Voyons le spectre pour un indice de modulation de 0,204 et $f_e = 6000\,\text{Hz}$:

```
[t,u] = numpy.loadtxt('modFM-4.txt')
```

```
N = t.size

T = t[N-1]-t[0]
te = t[1]-t[0]
fe = 1.0/te
zeros=numpy.zeros(2*N)
u=numpy.concatenate((u*scipy.signal.hamming(N),zeros))
Nz=len(u)
tfd=numpy.fft.fft(u)*2/N/0.54
a =numpy.absolute(tfd)
f=numpy.arange(Nz)*1.0/(Nz*te)
figure(figsize=(8,4))
plot(f,a/a.max())
xlabel("f (Hz)")
ylabel("A")
axis([0,fe,0,1.0])
grid()
```



Il y a plus de raies, et la largeur de bande occupée par le signal est plus grande.