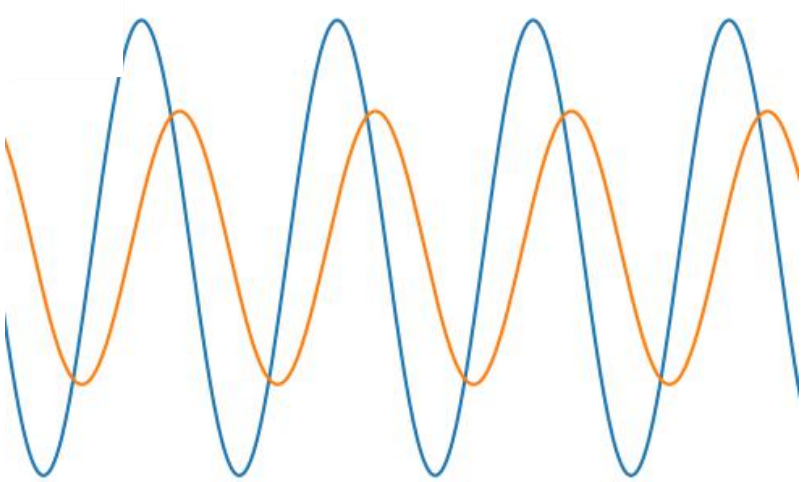
Contrôle actif du bruit acoustique



Acoustique linéaire non dissipative des milieux stationnaires

Equation des ondes : $\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} - \Delta p = \rho_0 \frac{\partial q}{\partial t}$

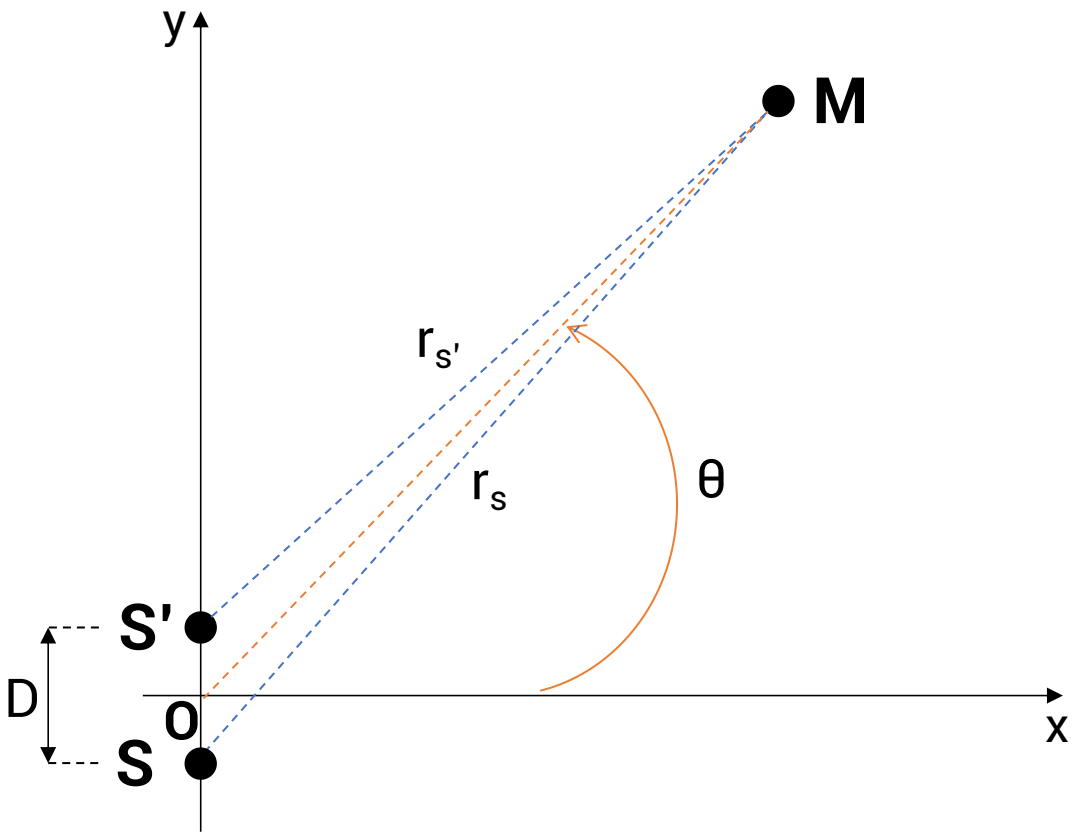
Equation de Helmholtz : $\Delta \underline{p_m} + k^2 \underline{p_m} = -j\omega\rho_0 \underline{q_m}$ où $\underline{p} = \underline{p_m} e^{j\omega t}$ $\underline{q} = \underline{q_m} e^{j\omega t}$

Solution pour une source ponctuelle (omnidirectionnelle) :

$$p(M) = \frac{j\omega\rho_0 q}{4\pi}\frac{e^{-jkr}}{r}$$

$$p(M) = \frac{j\omega\rho_0 q_m}{4\pi}\frac{e^{j(\omega t - kr)}}{r}$$

Dipôle acoustique



$$p_s(M) = \frac{j\omega\rho_0 q_s}{4\pi}\frac{e^{-jkr_s}}{r_s} \qquad q_s = Q_s e^{j\omega t}$$

$$p_{s'}(M) = \frac{j\omega\rho_0 q_{s'}}{4\pi}\frac{e^{-jkr_{s'}}}{r_{s'}} \qquad q_{s'} = Q_{s'} e^{j(\omega t+\phi)}$$

$$p(M) = p_s(M) + p_{s'}(M)$$

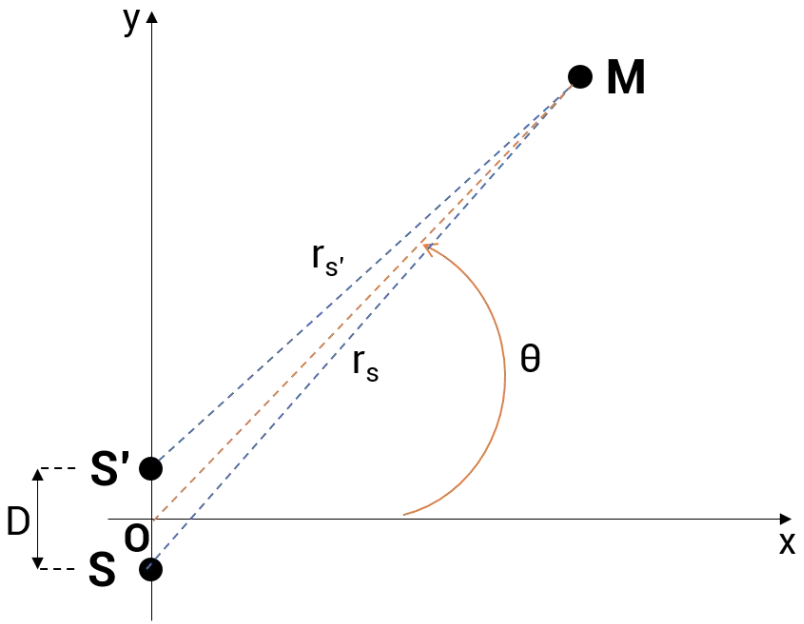
$$H(M) = \frac{p(M)}{p_s(M)} = 1 + \frac{p_{s'}(M)}{p_s(M)}$$

$$H(M) = 1 + \frac{q_{s'}}{q_s}\frac{r_s}{r_{s'}} e^{jk(r_s - r_{s'})}$$

$H(M) = 0$ *lorsque* $\frac{q_{s'}}{q_s}\frac{r_s}{r_{s'}} e^{jk(r_s - r_{s'})} = -1$

i.e. $\frac{Q_{s'}}{Q_s}\frac{r_s}{r_{s'}} e^{j\phi} e^{jk(r_s - r_{s'})} = -1$

i.e. $\begin{cases} Q_{s'} = \dfrac{Q_s r_{s'}}{r_s} \\ \phi = -k(r_s - r_{s'}) \end{cases}$



$$\overrightarrow{SM} = \overrightarrow{SO} + \overrightarrow{OM}$$

$$\overrightarrow{SM}^2 = \overrightarrow{SO}^2 + \overrightarrow{OM}^2 + 2\,\overrightarrow{SO}\cdot\overrightarrow{OM}$$

$$= \frac{D^2}{4} + r^2 + 2\cdot\frac{D}{2} r\cos\left(\frac{\pi}{2} - \theta\right)$$

$$= \frac{D^2}{4} + r^2 + Dr\sin\theta$$

$$= r^2\left(1 + \frac{D}{r}\sin\theta + \frac{D^2}{4r^2}\right)$$

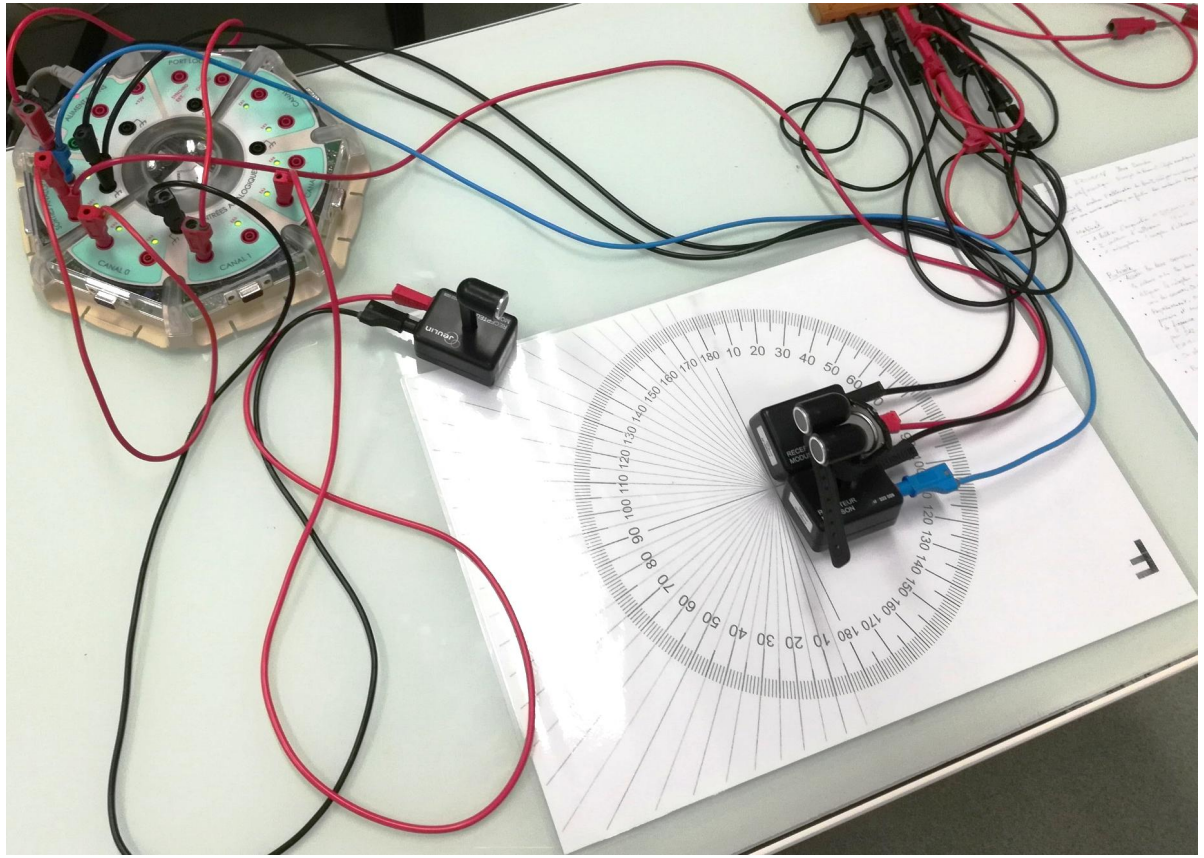
$$r_s = r\sqrt{1 + \frac{D}{r}\sin\theta + \frac{D^2}{4r^2}}$$

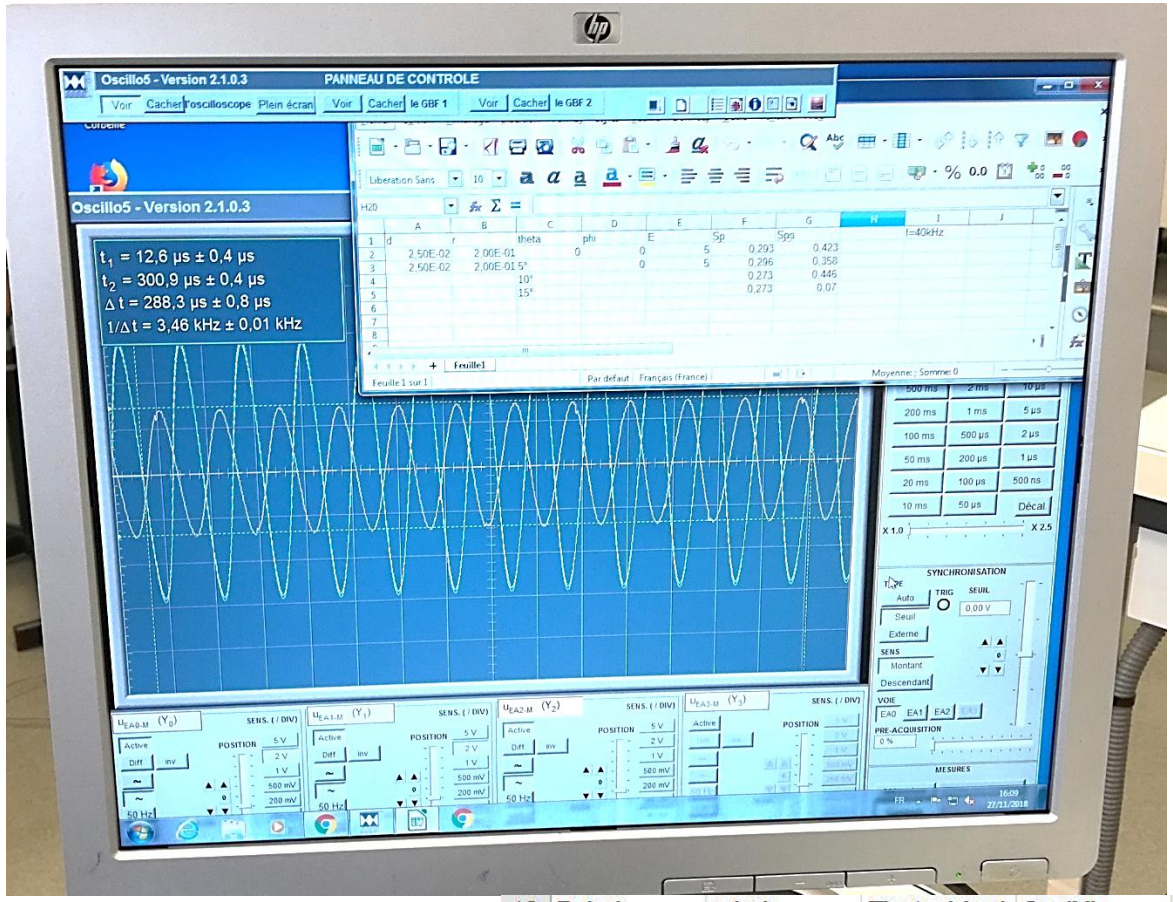
$$r \gg D \Rightarrow r_s = r\left(1 + \frac{D}{2r}\sin\theta\right)$$

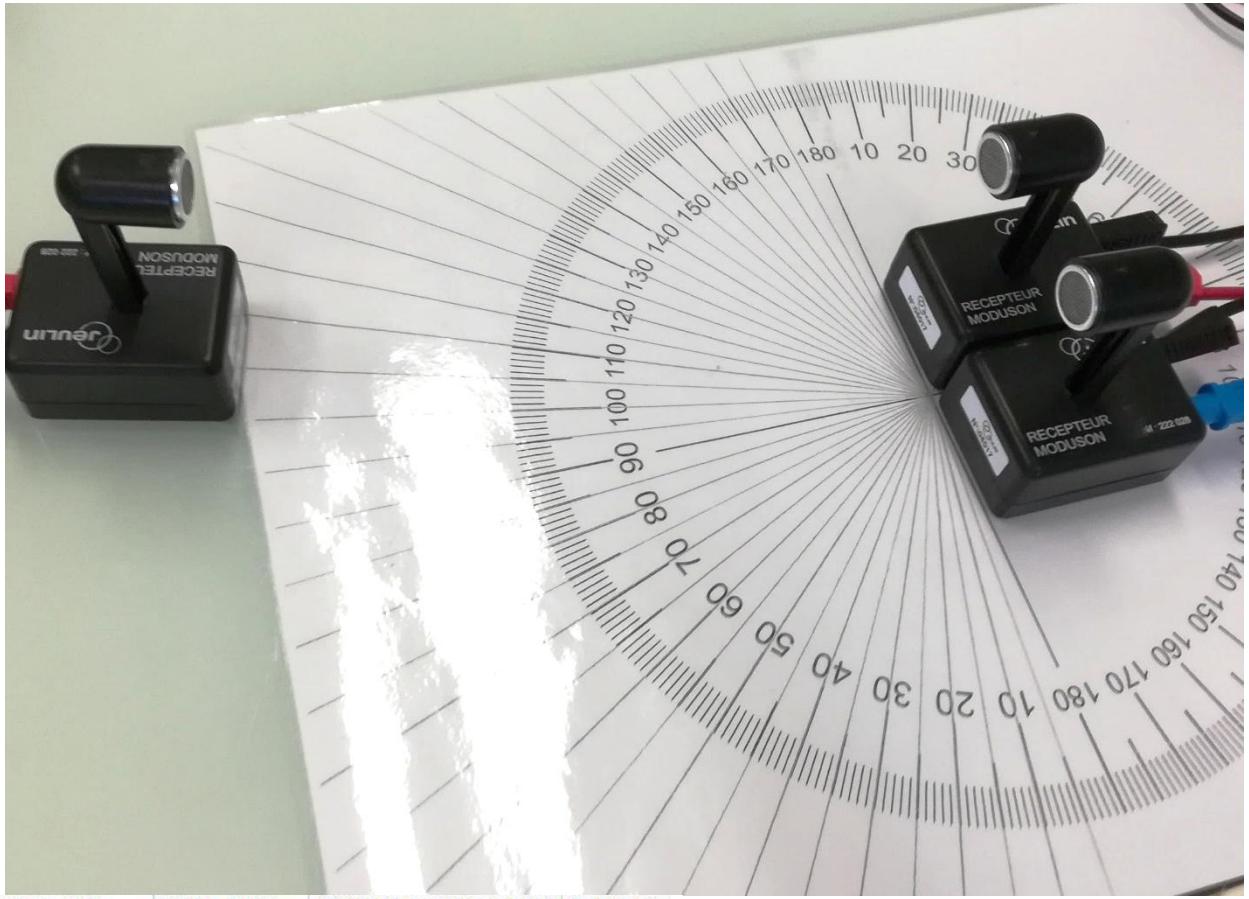
$$r_{s'} = r\left(1 - \frac{D}{2r}\sin\theta\right)$$

$$\begin{cases} Q_{s'} = \dfrac{Q_s r_{s'}}{r_s} \\ \phi = -k(r_s - r_{s'}) \end{cases} \Leftrightarrow \begin{cases} Q_{s'} \approx Q_s \\ \phi = -kD\sin\theta \end{cases}$$

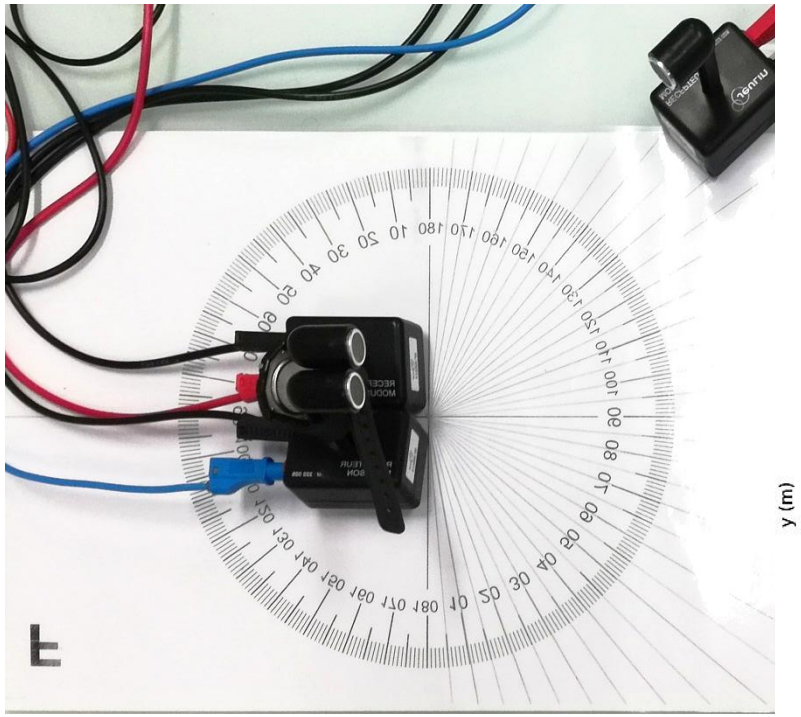
Expérience du dipôle acoustique

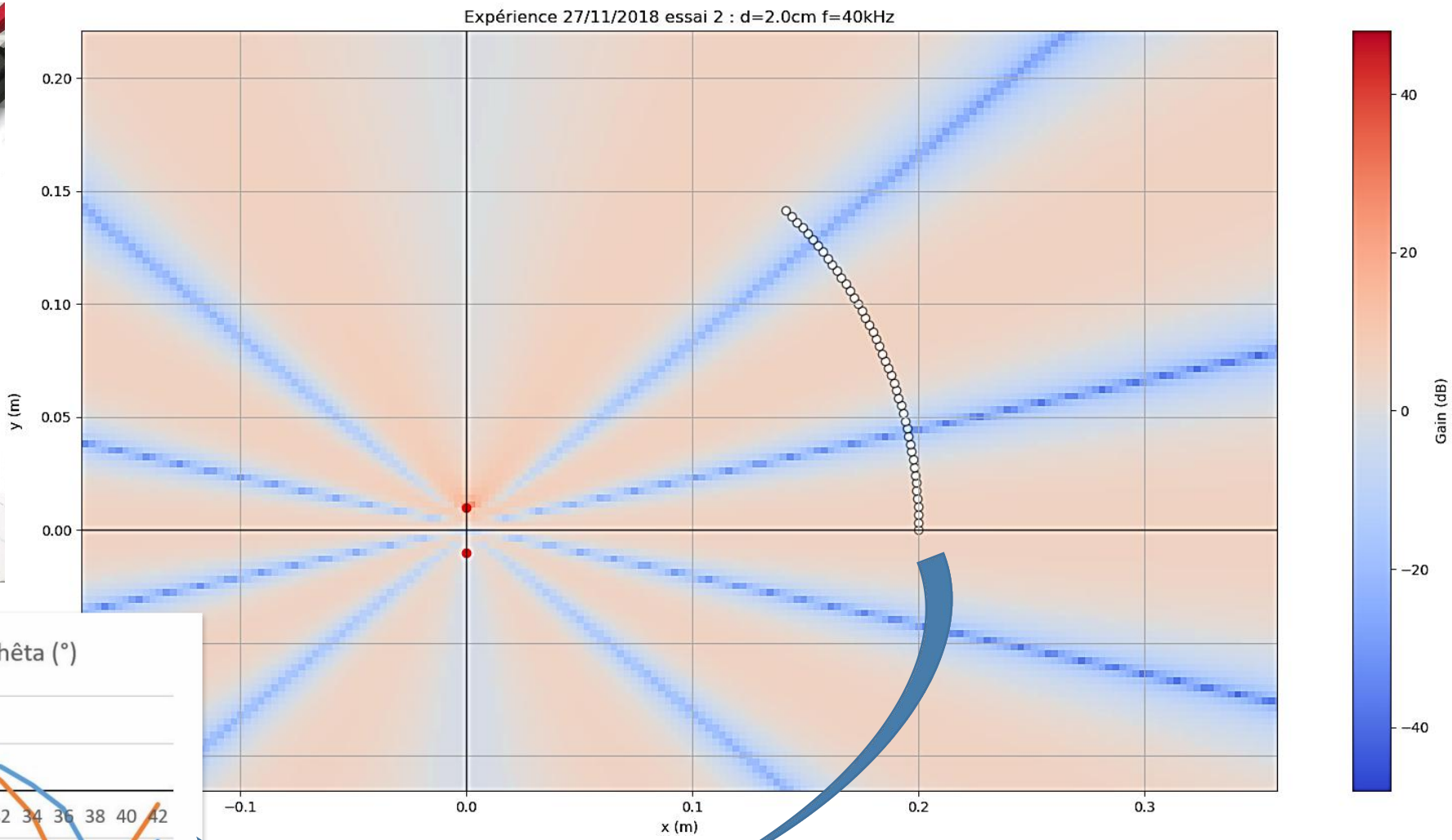


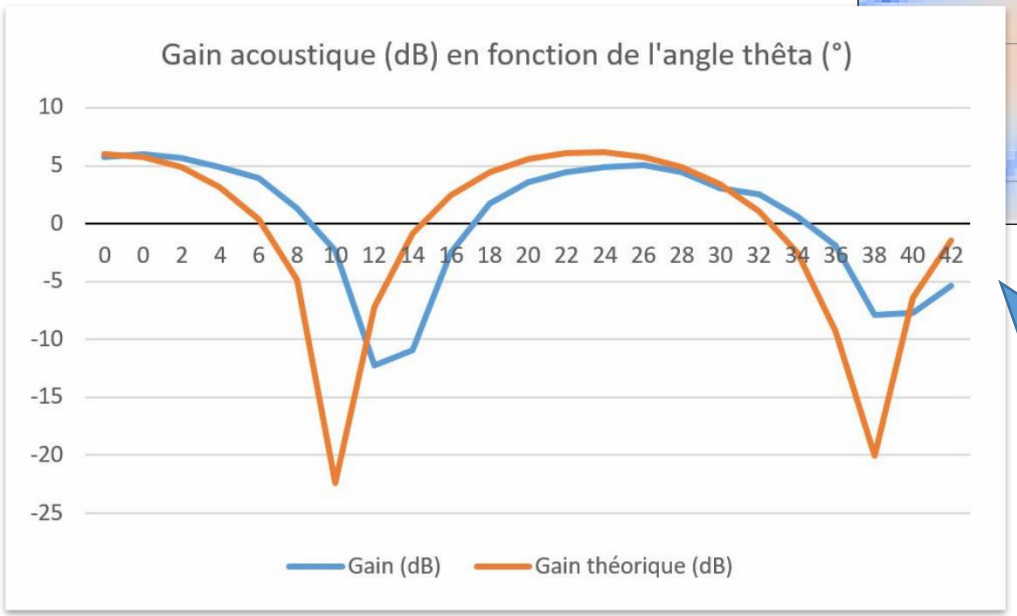


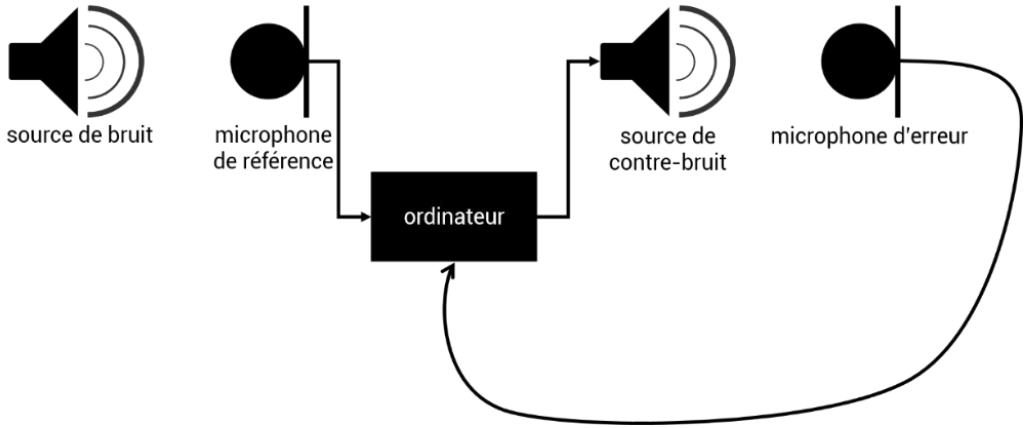


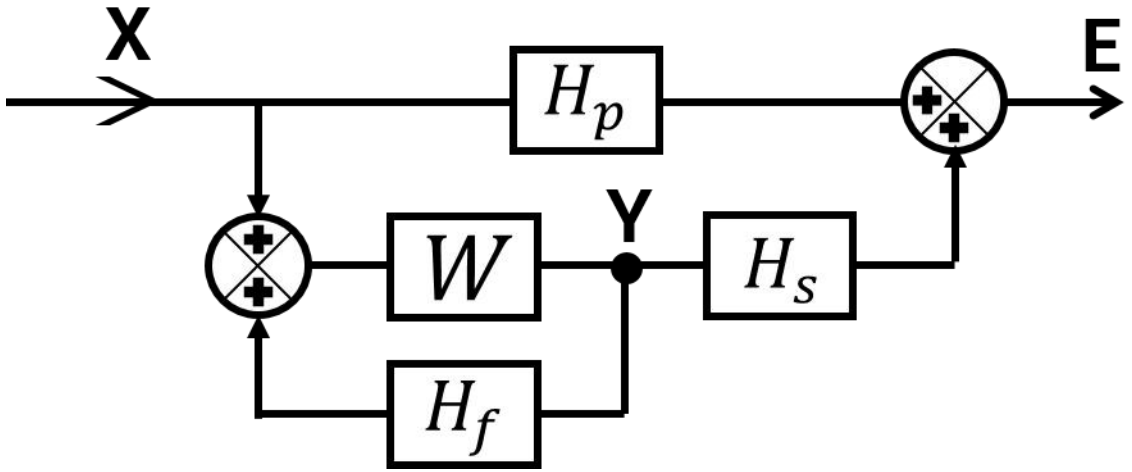
13	D (m)	r (m)	Theta (deg)	Sp (V)	Sps (V)	Gain (dB)	Gain théorique (dB)	f=40kHz
14	2.00E-02	2.00E-01	0	0.295	0.571	5.73628185	3.0103	
15			2	0.287	0.573	6.0054545	2.8721	
16			4	0.296	0.565	5.61513474	2.4221	
17			6	0.278	0.488	4.88750052	1.5865	
18			8	0.276	0.433	3.91157629	0.1754	
19			10	0.284	0.331	1.33019307	-2.4054	
20			12	0.276	0.211	-2.3325325	-11.2371	
21			14	0.265	0.065	-12.20665	-3.6319	
34			40	0.062	0.025	-7.8890336	-10.0437	
35			42	0.056	0.023	-7.7292038	-3.2122	
36			44	0.05	0.027	-5.3521248	-0.727	




Expérience 27/11/2018 essai 2 : d=2.0cm f=40kHz
y (m)
x (m)
Gain (dB)

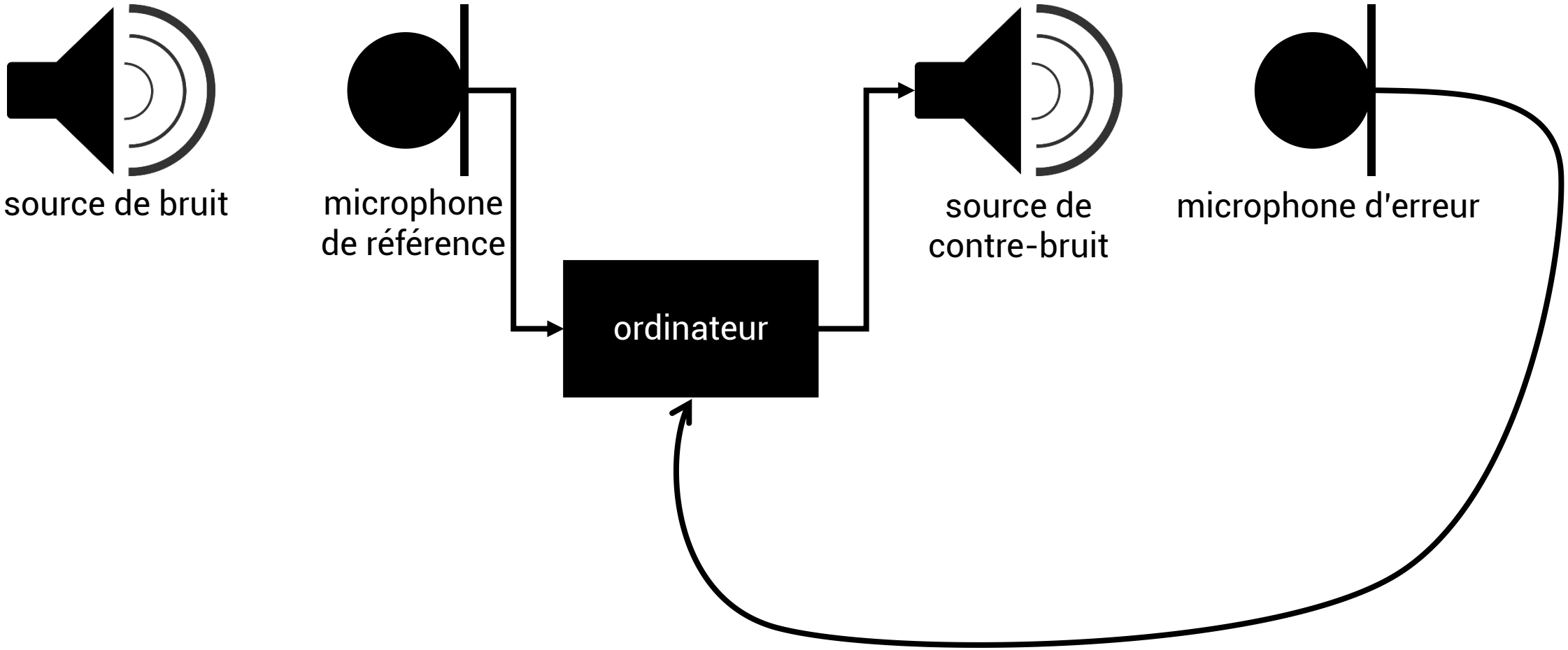

Gain acoustique (dB) en fonction de l'angle thêta (°)
Gain (dB)
Gain théorique (dB)





Contrôle actif et automatique

Contrôle monovoie



Transformation de Fourier discrète

Transformée de Fourier discrète de s[t], signal de N échantillons :

$$S(k) = \sum_{n=0}^{N-1} s(n) \mathrm{e}^{-2\mathrm{i}\pi k \frac{n}{N}} \qquad \text{pour} \qquad 0 \leqslant k < N$$

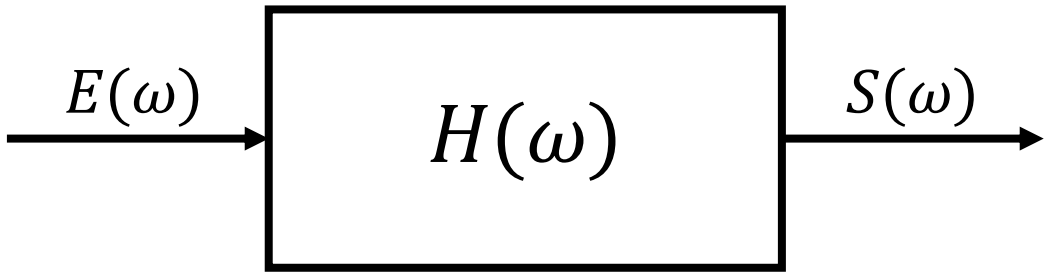
$$s(n) = \frac{1}{N} \sum_{k=0}^{N-1} S(k) \mathrm{e}^{2\mathrm{i}\pi n \frac{k}{N}}$$

Pour un signal réel, N/2+1 fréquences positives :

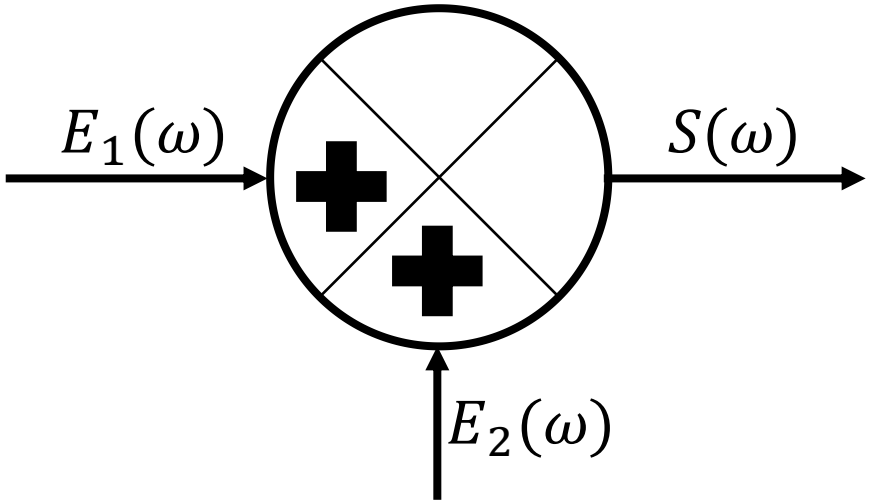
$$\forall k \in \left[\!\left[0, \frac{N}{2}\right]\!\right], \; f_k = k \frac{44100}{N} \;\; (Hz)$$

Théorème de convolution : $f(t) * g(t) = \mathcal{F}^{-1}(F(\omega)G(\omega))$

Systèmes linéaires et invariants

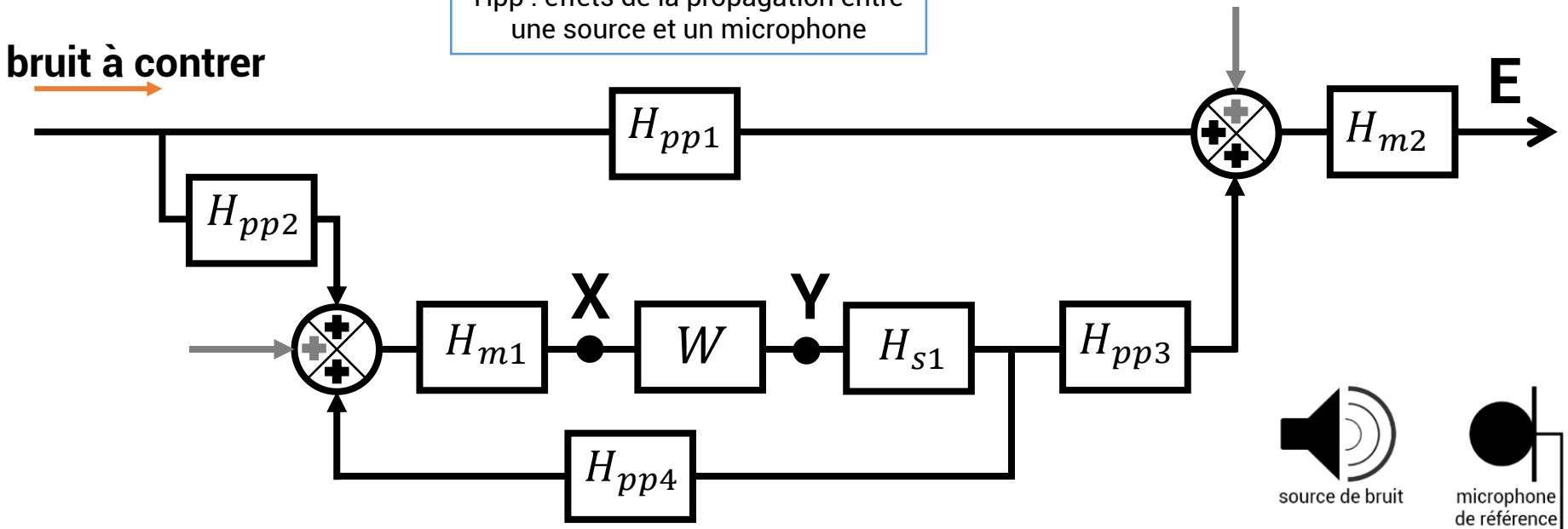


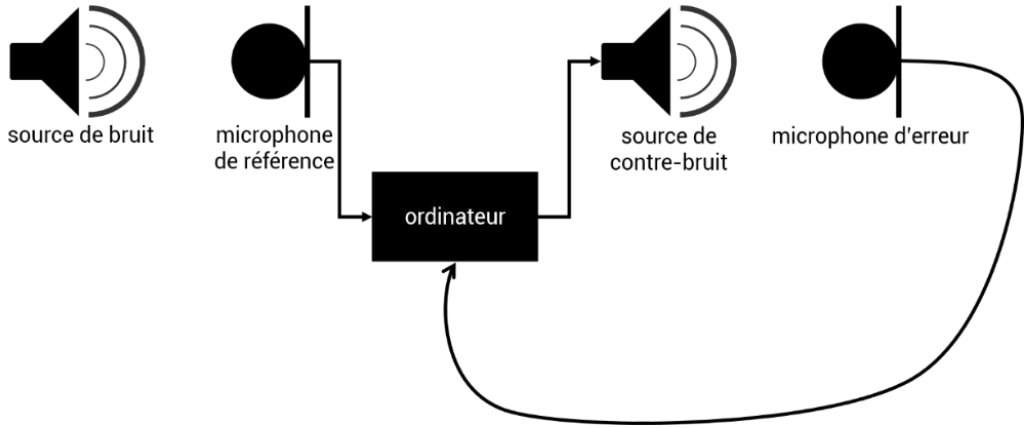
$$S(\omega) = H(\omega)E(\omega)$$

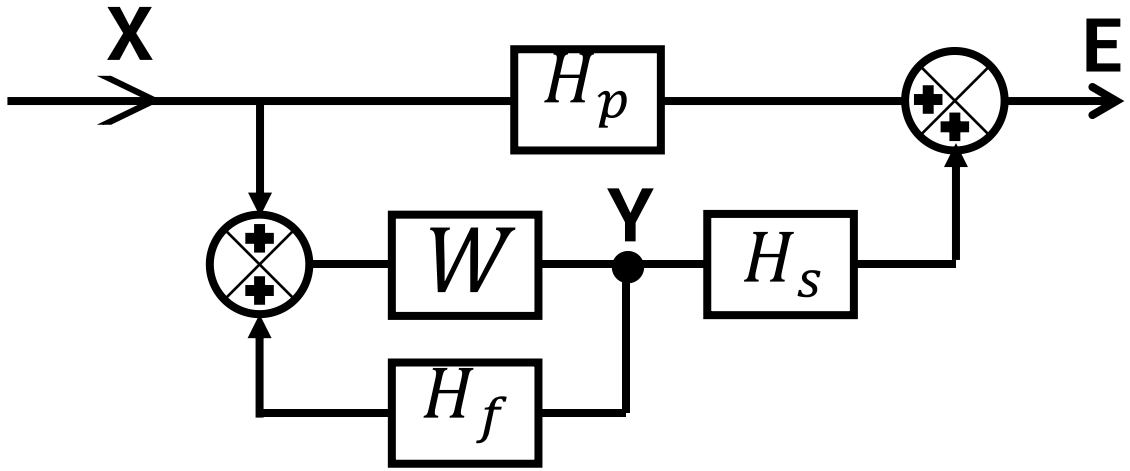


$$S(\omega) = E_1(\omega) + E_2(\omega)$$

Hpp : effets de la propagation entre une source et un microphone

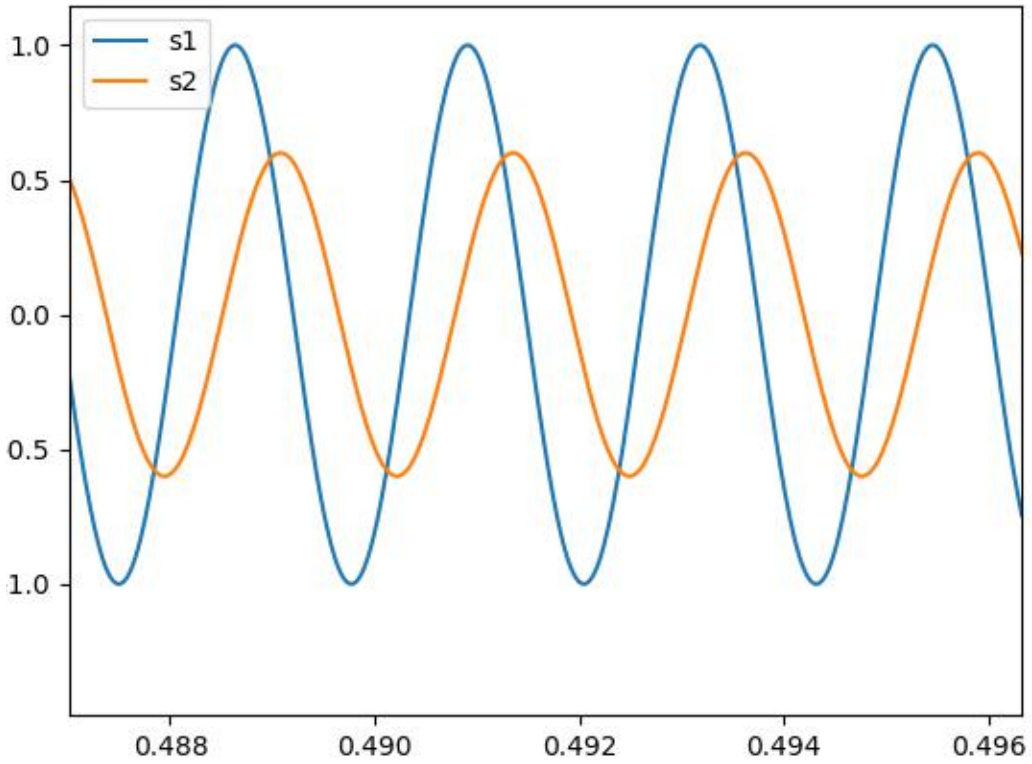






$$E = 0 \ \ lorsque \ W = \frac{1}{H_f - \frac{H_s}{H_p}}$$

Mesure de fonctions de transfert



$$s_1(t) = A_1 \cos(2\pi f t + \phi_1)$$
$$s_2(t) = A_2 \cos(2\pi f t + \phi_2)$$

$$H(f) = \frac{S_2(f)}{S_1(f)} = \frac{A_2}{A_1} e^{j(\phi_2 - \phi_1)}$$

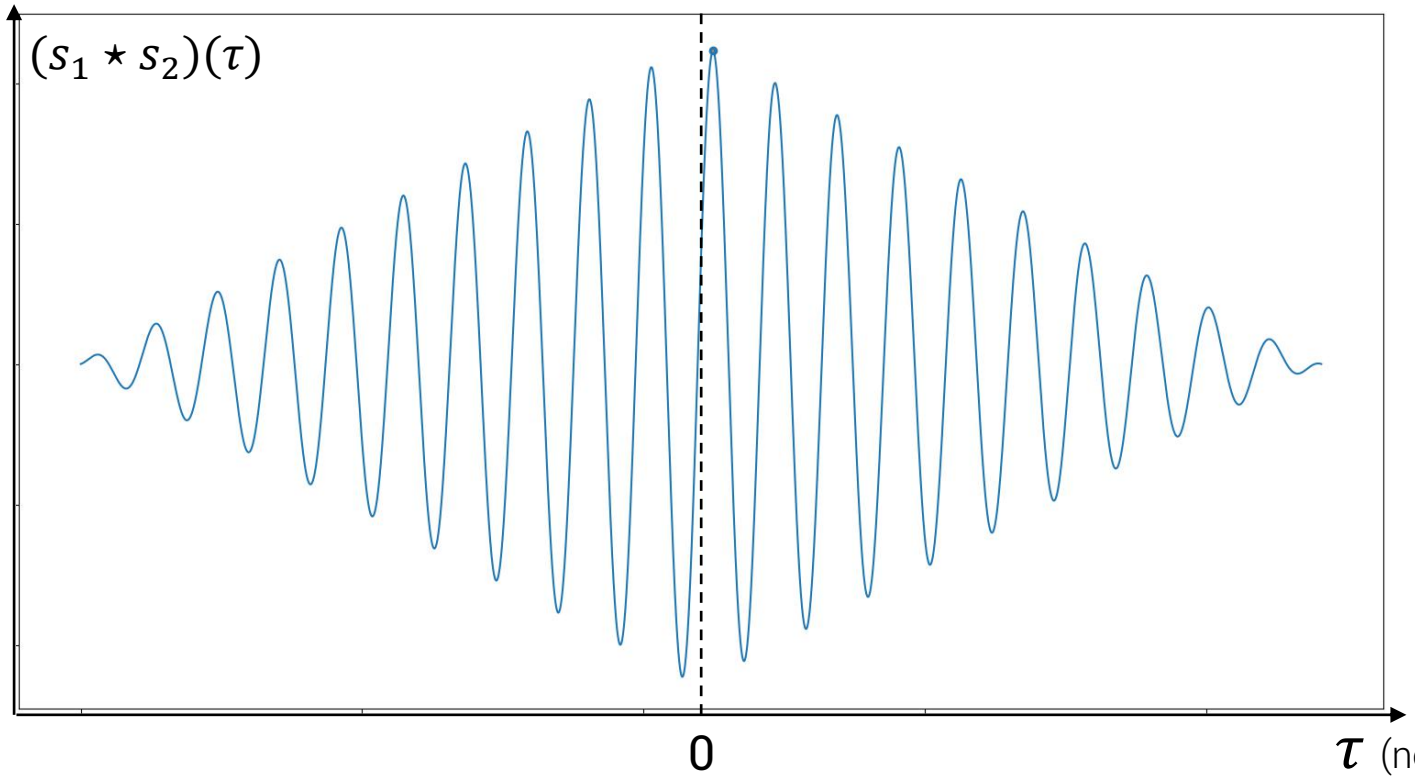
Mesure de $\frac{A_2}{A_1}$

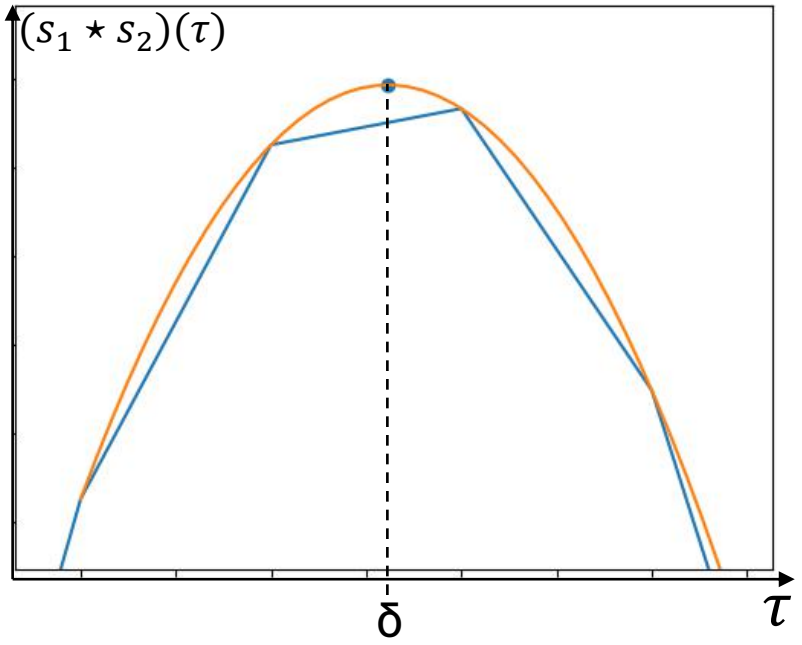
$$s(t) = A\cos(2\pi f t + \varphi)$$

$$Valeur\ efficace : \sqrt{< s^2(t) >} = \frac{A}{\sqrt{2}}$$

Mesure de ϕ par corrélation croisée

$$(f \star g)(\tau) = \int_{-\infty}^{\infty} f(t)g(t+\tau)\,dt$$

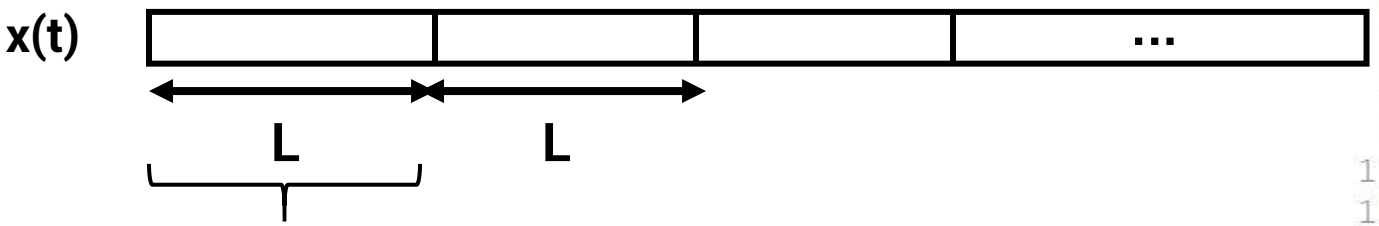




$$\phi = \frac{2\pi f \delta}{44100}$$

Gestion de l'audio avec Python

voir fonctions **engine**



`callback(indata, outdata, …)`

```python
import numpy as np
import sounddevice as sd
a=0
def callback(indata, outdata, frames, time, status):
    global a,s1_rec,s2_rec,s3_rec,s4_rec
    if status:
        print(status)

    s1 = np.array(indata).transpose()[0]
    s2 = np.array(indata).transpose()[1]

    t = np.linspace(a/SMPFREQ,(a+L)/SMPFREQ,num=L,endpoint=False)
    s3 = AMP_REC*np.cos(2*np.pi*f*t)

    s4 = np.zeros(L)

    a+=L

    s1_rec = np.concatenate([s1_rec[-30*SMPFREQ:],s1])
    s2_rec = np.concatenate([s2_rec[-30*SMPFREQ:],s2])
    s3_rec = np.concatenate([s3_rec[-30*SMPFREQ:],s3])
    s4_rec = np.concatenate([s4_rec[-30*SMPFREQ:],s4])

    outdata[:] = np.array([s3,s4]).transpose()

with sd.Stream(samplerate=SMPFREQ, blocksize=L, latency='low',
               channels=2, callback=callback):
    sd.sleep(int(duree*1000))

```

Filtrage en temps réel : méthode Overlap-Add (cf filter_OLA())

- On découpe x(t) en subdivisions $x_k(t)$ de taille L
- N=L+M-1
- $Y_k(\omega) = X_k^{(N)}(\omega) * W^{(N)}(\omega)$
- $y_k(t) = \mathcal{F}^{-1}(Y_k^{(N)}(\omega))$
- On somme les $y_k(t)$

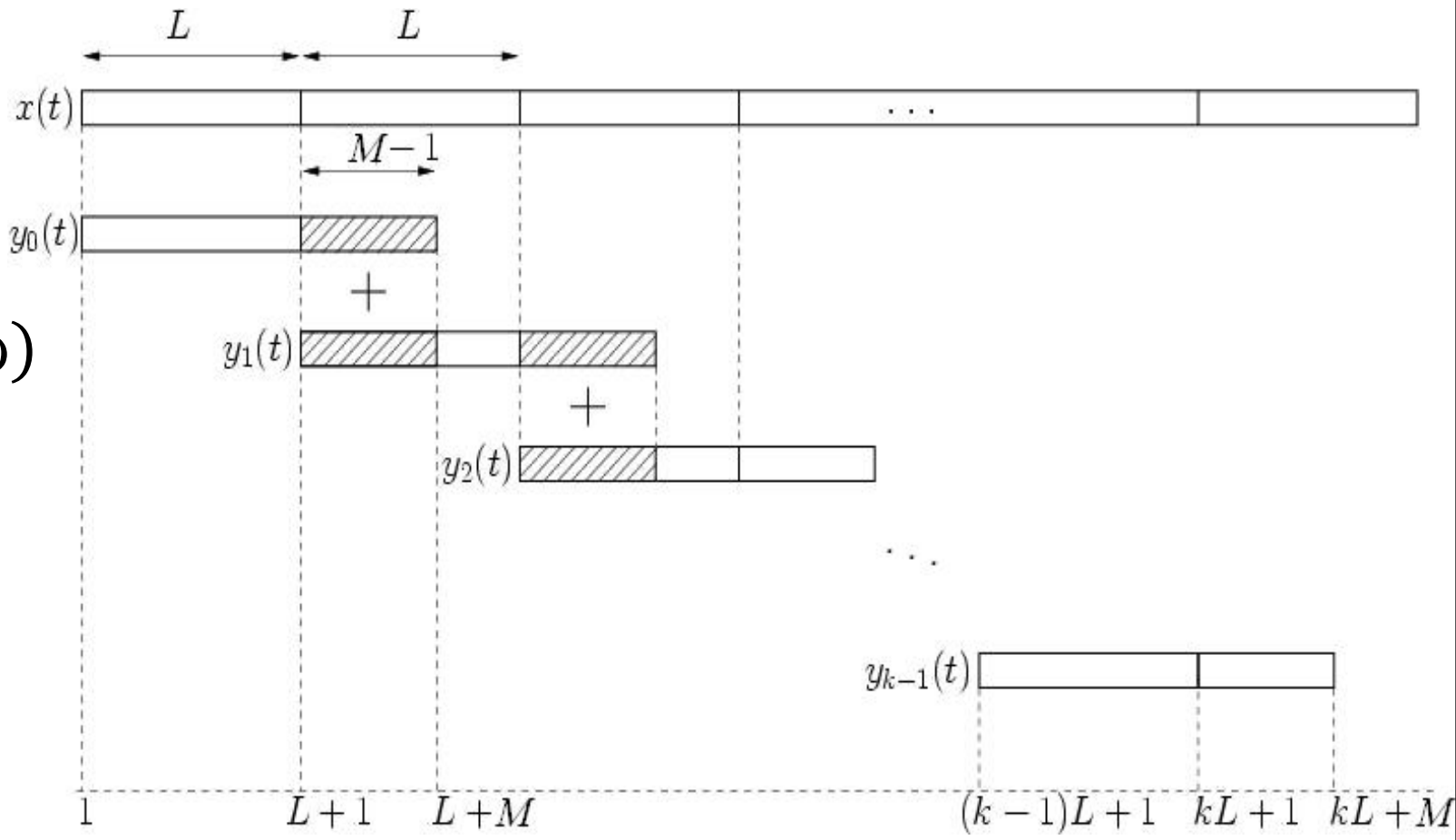
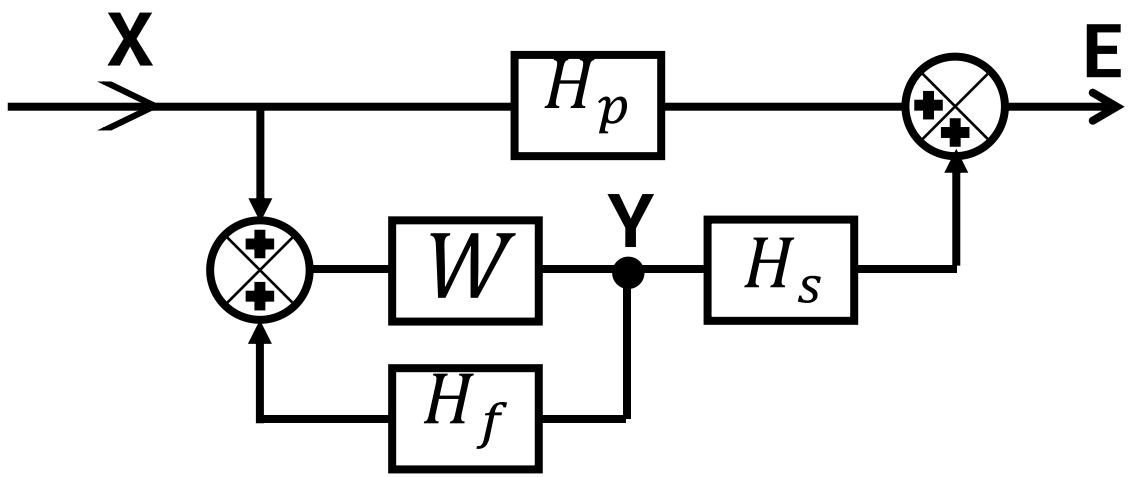


Illustration issue de l'article Wikipédia « Overlap-add method »



$$H_p(f) = \left.\frac{E(f)}{X(f)}\right|_{Y=0}$$

$$H_s(f) = \left.\frac{E(f)}{Y(f)}\right|_{X=0}$$

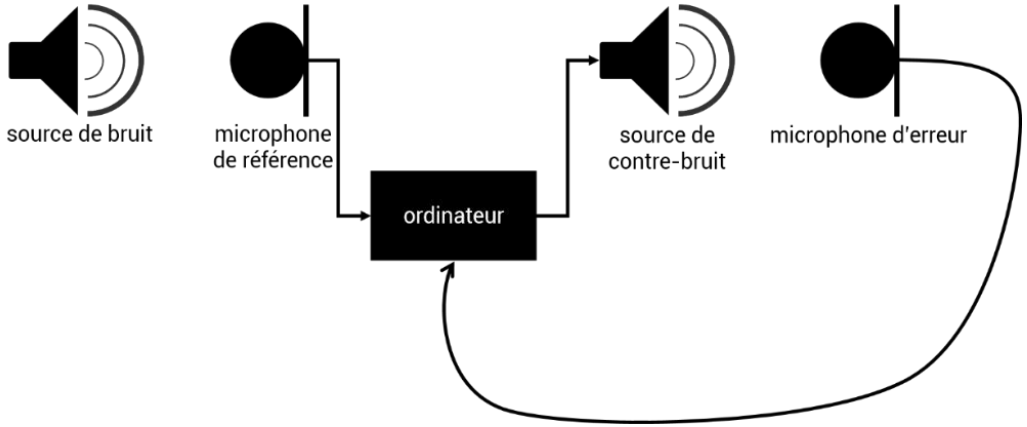
$$H_f(f) = \left.\frac{X(f)}{Y(f)}\right|_{\substack{X=0 \\ W=1}}$$

$$W(f) = \frac{1}{H_f(f) - \frac{H_s(f)}{H_p(f)}}$$

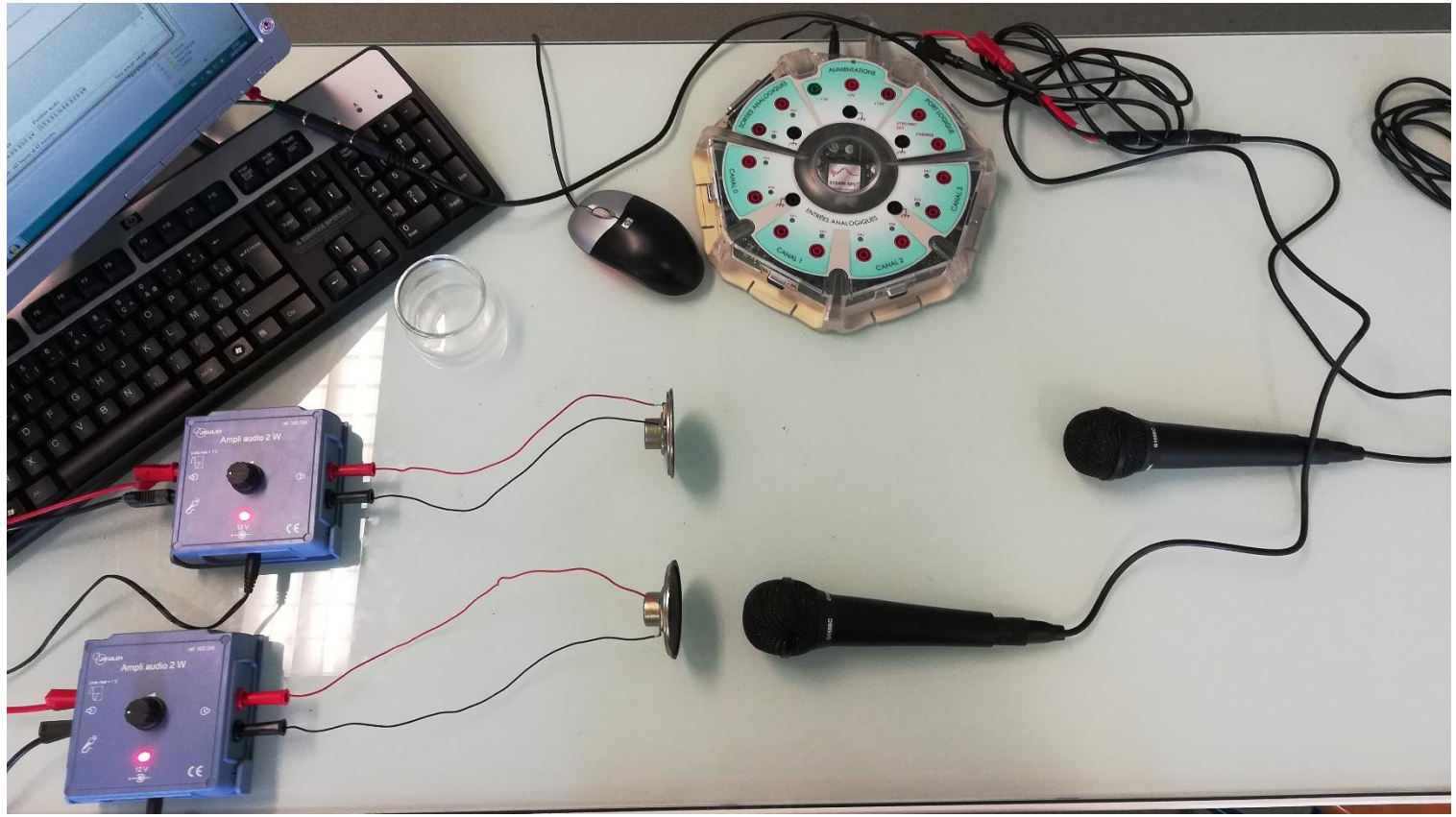
$$f \in \left\{0,\ 1 \cdot \frac{44100}{L},\ 2 \cdot \frac{44100}{L}, \ldots, \frac{L}{2} \cdot \frac{44100}{L}\right\} \text{(Hz)}$$

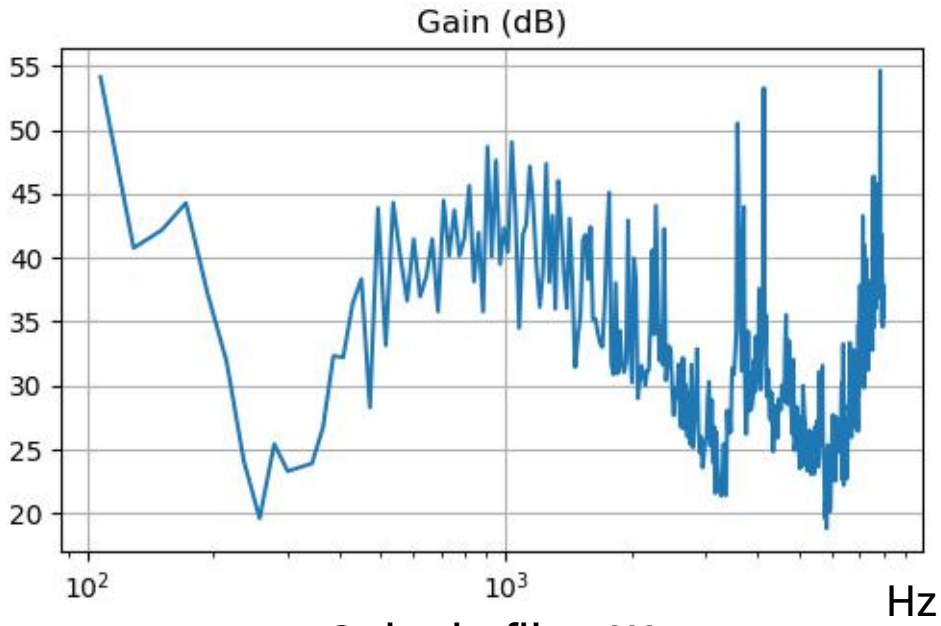
ici, avec L=2048 :

$$f \in \{0;\ 21{,}5;\ 43{,}1;\ \ldots;\ 22028;\ 22050\} \text{ (Hz)}$$

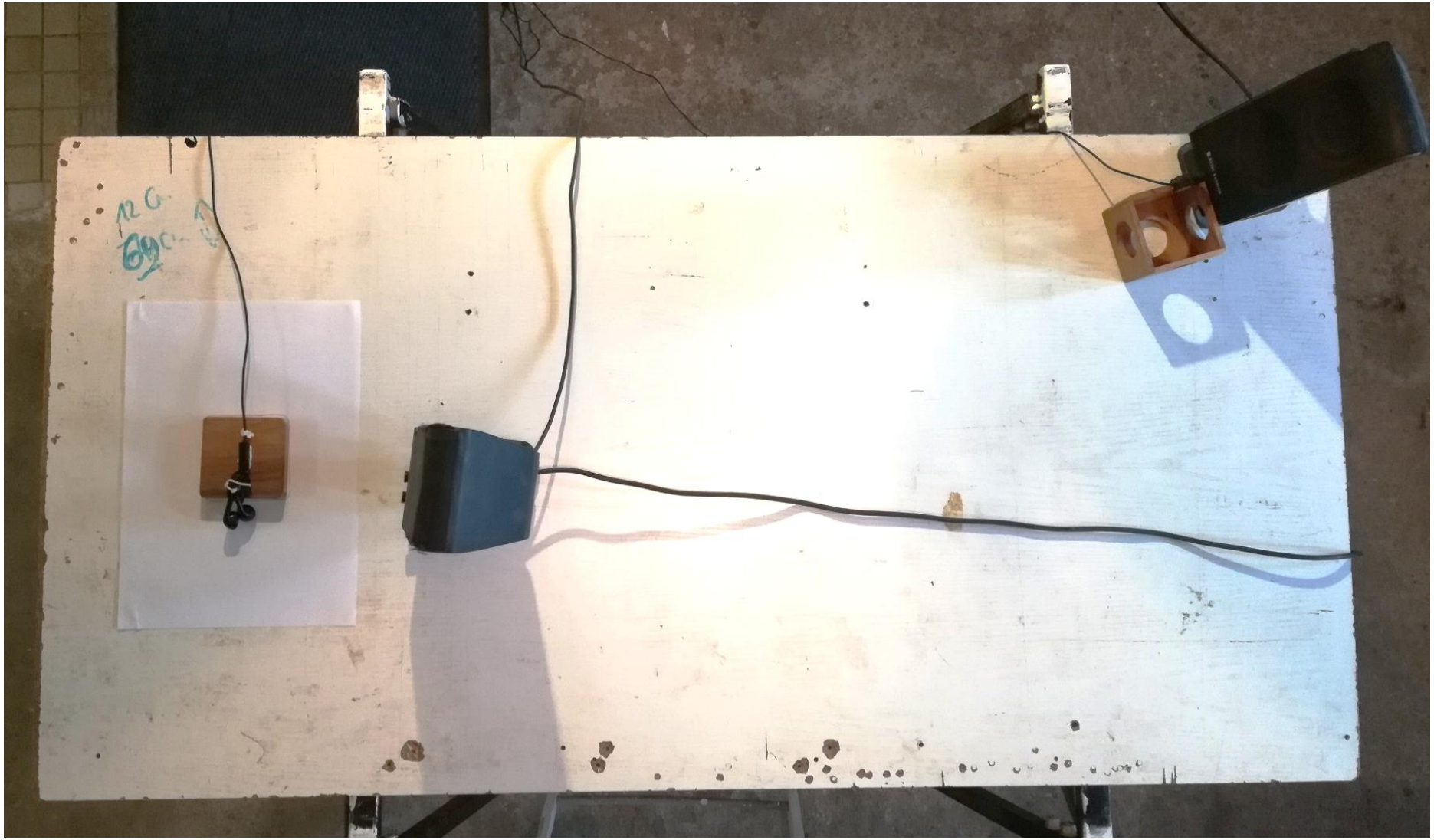


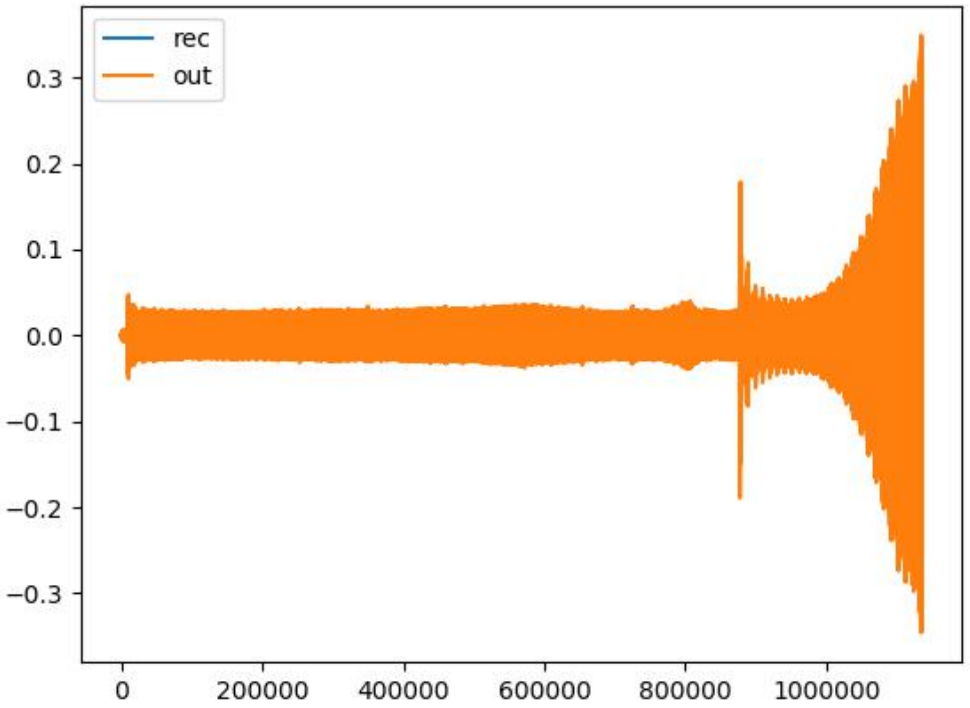
Expérimentations



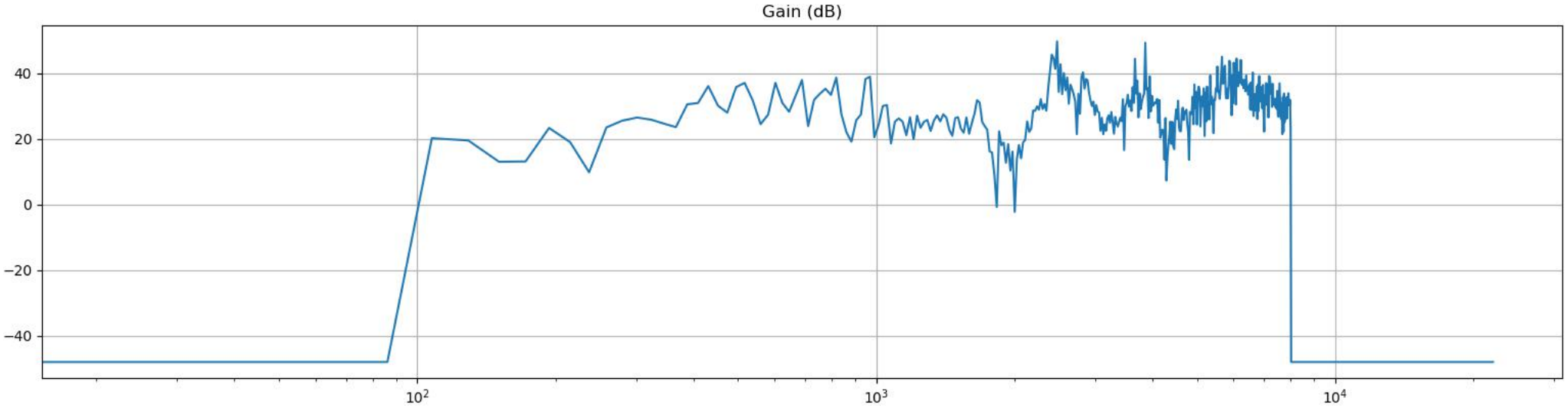


Gain du filtre W





effet larsen
(boucle instable)



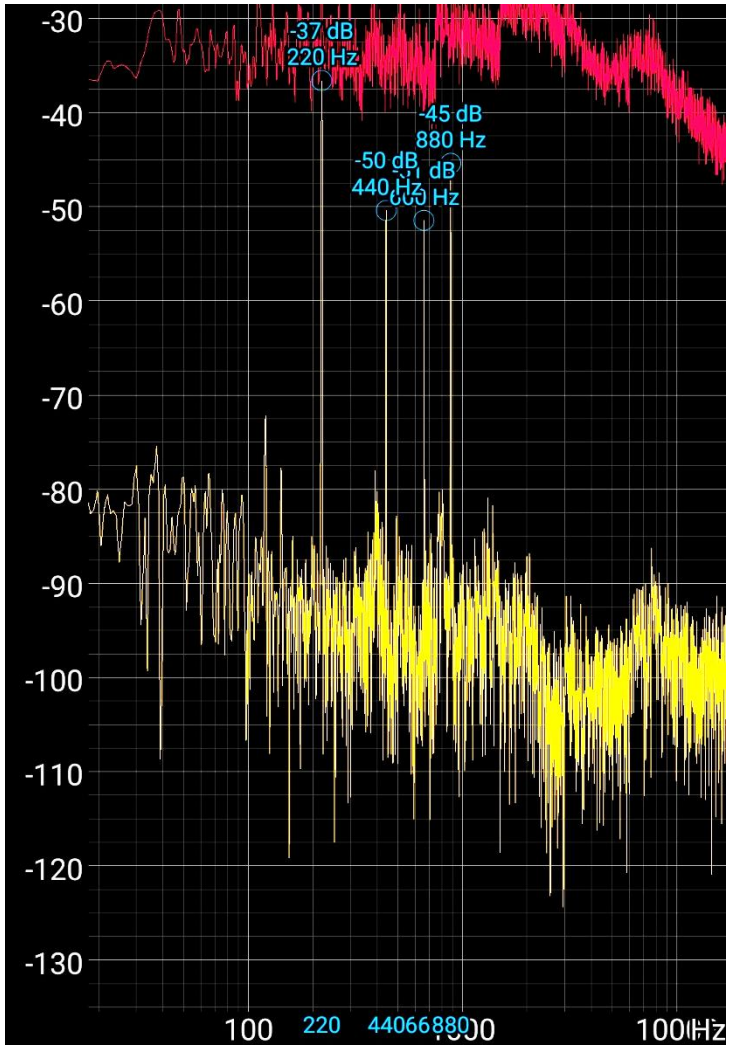
Cas d'un bruit pur (sinusoïdal)

```
>>> stats(n=7,f_list=[262,330,440,523,659,880,1046.5,1318.5,1760])
Gain moyen pour f= 262 : -5.417766635543484
Gain moyen pour f= 330 : -4.279523190942039
Gain moyen pour f= 440 : -2.74944354735907
Gain moyen pour f= 523 : -4.18458499648263
Gain moyen pour f= 659 : -1.9408427274437063
Gain moyen pour f= 880 : -1.1201043279370086
Gain moyen pour f= 1046.5 : 2.0494019620621944
Gain moyen pour f= 1318.5 : 1.083986158719984
Gain moyen pour f= 1760 : 3.2306335517360156
```

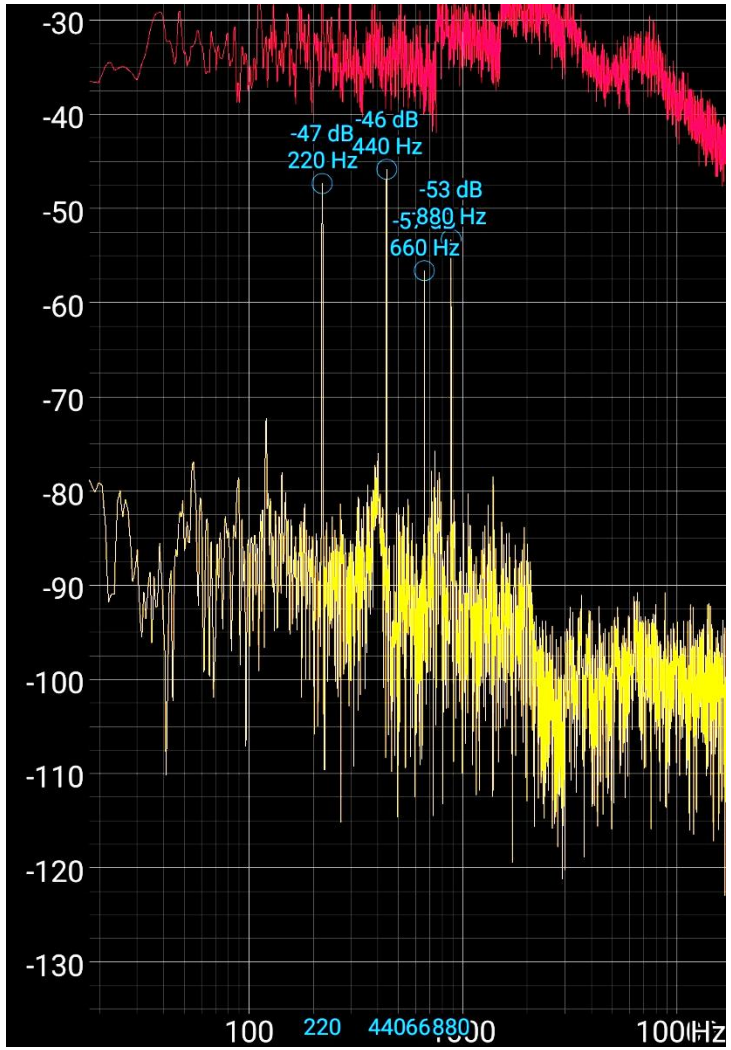
après avoir éloigné les sources, restreint le domaine fréquentiel (pour limiter l'effet larsen),
et réajusté les paramètres (pour minimiser les erreurs de mesure) :

sans contrôle actif

220 Hz	-37 dB
440 Hz	-50 dB
660 Hz	-51 dB
880 Hz	-45 dB

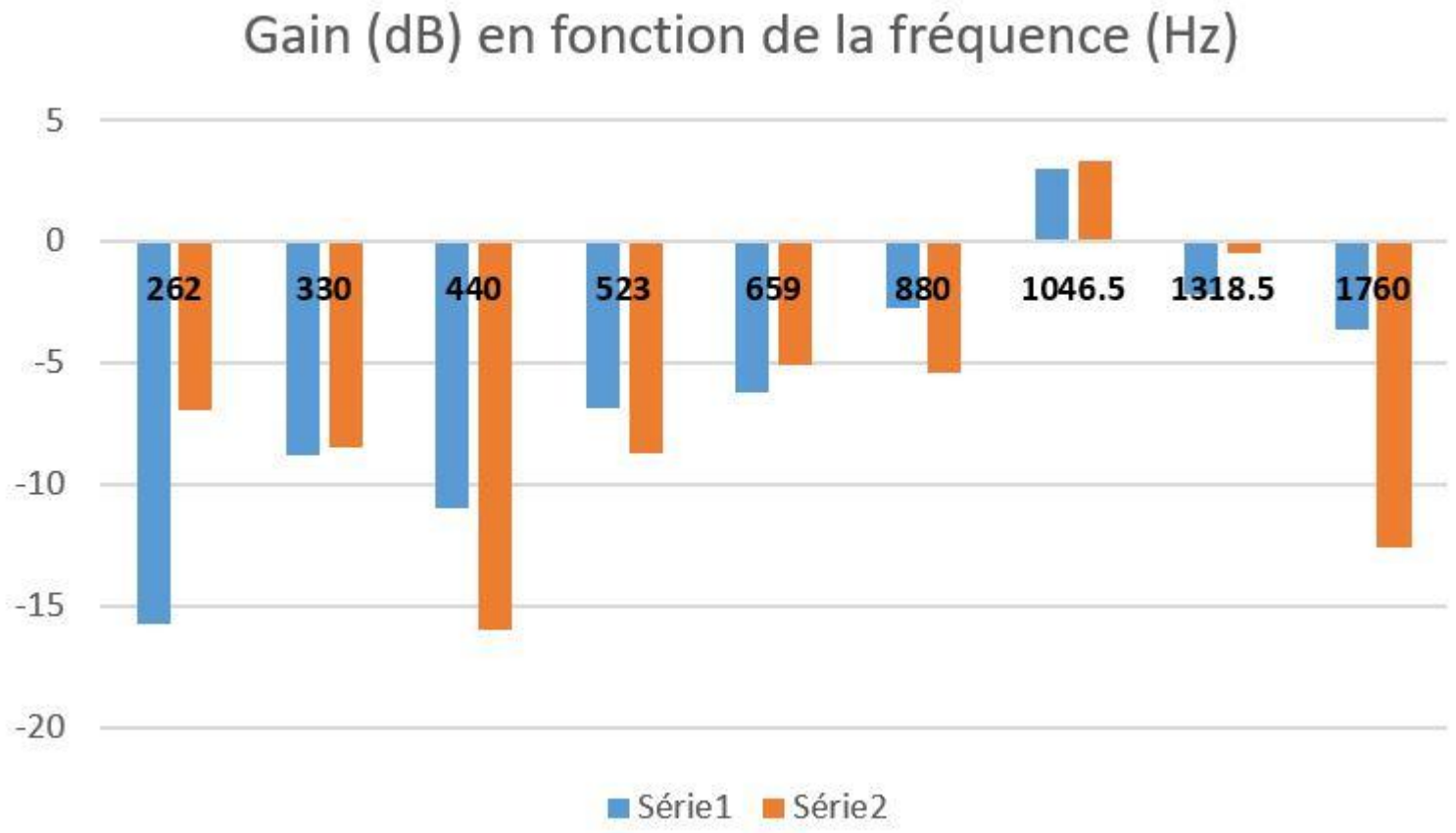


avec contrôle actif



220 Hz	-47 dB
440 Hz	-46 dB
660 Hz	-57 dB
880 Hz	-53 dB

et pour un bruit pur :



```
##################
## IMPORTS et variables globales
from scipy.io import wavfile as wav
import scipy.signal as sgnl
import numpy as np
import matplotlib.pyplot as plt
import os
from random import randint
import sounddevice as sd
from time import time

FFTSIZE = 2048    # nombre d'échantillons
SMPFREQ = 44100    # taux d'échantillonnage (44100 échantillons par seconde)
SUB = 16384    # nombre d'échantillons sur lesquels on mesure les amplitudes et les déphasages
AMP_REC = 0.2    # volume sonore
MARGE_REC = 30000
DELAI_ACTIVATION_CONTROLE_ACTIF = 1.0    # secondes
BUFFER = np.zeros(FFTSIZE)    # "tampon" utilisé pour le filtrage dans engine4()
s1_rec = np.array([])    # enregistrement du micro de réf
s2_rec = np.array([])    # enregistrement du micro d'erreur
s3_rec = np.array([])    # enregistrement de la source primaire
s4_rec = np.array([])    # enregistrement de la source secondaire

## Remarque : les fonctions utilisent les variables globales suivantes en écriture : BUFFER, a, rec, out, Hp, Hs, Hf

##
def transfer_sine(s1,s2,f,sampFreq):
    ## calcule la fonction de transfert S2/S1 à la fréquence f
    A1 = (2**.5)*np.mean(s1.astype(np.float)**2)**.5 # amplitude de s1
    A2 = (2**.5)*np.mean(s2.astype(np.float)**2)**.5 # amplitude de s2
    r = A2/A1
    xcorr = sgnl.correlate(r*s1, s2) # corrélation croisée de s1 et s2
    i_max = np.argmax(xcorr)

    ## Interpolation parabolique, détermination de l'abscisse alpha du sommet
    x1,x2,x3=i_max-1,i_max,i_max+1
    y1,y2,y3=xcorr[x1],xcorr[x2],xcorr[x3]
    a = y1/((x1-x2)*(x1-x3)) + y2/((x2-x1)*(x2-x3)) + y3/((x3-x1)*(x3-x2))
    b = -(x2+x3)*y1/((x1-x2)*(x1-x3)) - (x1+x3)*y2/((x2-x1)*(x2-x3)) - (x1+x2)*y3/((x3-x1)*(x3-x2))
    alpha = -b/(2*a)
    ##

    delta_t = alpha-(len(s2)-1)
    ps = 2*np.pi*f*delta_t/sampFreq    # déphasage de s2 par rapport à s1
    return r,ps

def test_sd():
    ## test des périphériques audio
    t = np.linspace(0,2,num=88200,endpoint=False)
```

```
    s1d=0*t
    s1g = AMP_REC*sgnl.tukey(88200,alpha=0.1)*np.cos(2*np.pi*440*t)
    myrecording = sd.playrec(np.array([s1g,s1d]).transpose(),44100,channels=2, blocking=True)
    plt.plot(myrecording)
    plt.show()


def filter_OLA(s, h):
    ## implémentation de la méthode Overlap-Add afin de filtrer s par le filtre h
    M = len(h)
    L = M
    N=M+L
    print("N=M+L=",N,"avec M=",M,"et L=",L)
    H = np.fft.rfft(h,n=N)
    Nx = len(s)
    a=0
    y=np.zeros(Nx+M)
    while a<Nx-L:
        b=a+L
        yt = np.fft.irfft( np.fft.rfft( s[a:b], n=N ) * H, n=N )
        c=a+N
        for i in range(N):
            y[a+i]+=yt[i]
        a+=L
    return y

def engine1(f,duree):
    ## inteface audio, utilisée pour mesurer Hp (dans transferHp())
    ## émet le bruit primaire (sinusoïde de freq f), pas de bruit secondaire
    global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec

    L = FFTSIZE    # taille d'un bloc
    BUFFER = np.zeros(2*L)
    a=0    # a est l'instant de début de chaque bloc (en échantillons)

    def callback(indata, outdata, frames, time, status):
        global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec
        if status:
            print(status)

        s1 = np.array(indata).transpose()[0]
        s2 = np.array(indata).transpose()[1]
        s3 = np.zeros(L)
        s4 = np.zeros(L)


        t = np.linspace(a/SMPFREQ,(a+L)/SMPFREQ,num=L,endpoint=False)
        s3 = AMP_REC*np.cos(2*np.pi*f*t)
        a+=L
```

```
99
100         s1_rec = np.concatenate([s1_rec,s1])
101         s2_rec = np.concatenate([s2_rec,s2])
102         s3_rec = np.concatenate([s3_rec,s3])
103         s4_rec = np.concatenate([s4_rec,s4])
104
105         outdata[:] = np.array([s3,s4]).transpose()
106
107     with sd.Stream(samplerate=SMPFREQ, blocksize=L, latency='low',
108                    channels=2, callback=callback):
109         #print("in stream! (engine1)")
110         sd.sleep(int(duree*1000))
111
112 def engine2(f,duree):
113     ## inteface audio, utilisée pour mesurer Hs (dans transferHs() et transferHf())
114     ## émet le bruit secondaire (sinusoïde de freq f), pas de bruit primaire
115     global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec
116
117     L = FFTSIZE    # taille d'un bloc
118     BUFFER = np.zeros(2*L)
119     a=0    # a est l'instant de début de chaque bloc (en échantillons)
120
121     def callback(indata, outdata, frames, time, status):
122         global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec
123         if status:
124             print(status)
125
126         s1 = np.array(indata).transpose()[0]
127         s2 = np.array(indata).transpose()[1]
128         s3 = np.zeros(L)
129         s4 = np.zeros(L)
130
131
132         t = np.linspace(a/SMPFREQ,(a+L)/SMPFREQ,num=L,endpoint=False)
133         s4 = AMP_REC*np.cos(2*np.pi*f*t)
134         a+=L
135
136         s1_rec = np.concatenate([s1_rec,s1])
137         s2_rec = np.concatenate([s2_rec,s2])
138         s3_rec = np.concatenate([s3_rec,s3])
139         s4_rec = np.concatenate([s4_rec,s4])
140
141         outdata[:] = np.array([s3,s4]).transpose()
142
143     with sd.Stream(samplerate=SMPFREQ, blocksize=L, latency='low',
144                    channels=2, callback=callback):
145         #print("in stream! (engine2)")
146         sd.sleep(int(duree*1000))
147
```

```
148
149 def engine4(w,f, duree=None, mesure=False):
150     ## interface audio de la fonction principale du contrôle actif (controleactif())
151     ## émet le bruit primaire (son composé de fréq fondamentale f), émet le bruit secondaire (qui est la signal du micro de réf filtré par w)
152     global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec,A1
153
154     L=len(w)    # taille d'un bloc
155     N=2*L
156     W=np.fft.rfft(w,n=N)
157     BUFFER = np.zeros(N)
158     a=0    # a est l'instant de début de chaque bloc (en échantillons)
159     A1=None
160
161
162     def callback(indata, outdata, frames, time, status):
163         global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec,A1
164         if status:
165             print(status)
166
167         s1 = np.array(indata).transpose()[0]
168         s2 = np.array(indata).transpose()[1]
169         s3 = np.zeros(L)
170         s4 = np.zeros(L)
171
172         ## après le délai d'activation du contrôle actif, s4 (bruit secondaire) est le signal s1 (réf) filtré par w
173         if a>=DELAI_ACTIVATION_CONTROLE_ACTIF*SMPFREQ :
174             if A1==None:A1 = (2**.5)*np.mean(s2.astype(np.float)**2)**.5 # amplitude avant contrôle actif
175             yt = np.fft.irfft( np.fft.rfft( s1, n=N ) * W, n=N )
176             y=np.zeros(N)
177             for i in range(L):
178                 y[i]+=BUFFER[L+i]+yt[i]
179             for i in range(L,N):
180                 y[i]+=yt[i]
181             s4=y[0:L]
182             BUFFER = y
183
184         t = np.linspace(a/SMPFREQ,(a+L)/SMPFREQ,num=L,endpoint=False)
185         s3 = AMP_REC*(0.5*np.cos(2*np.pi*f*t)+0.25*np.cos(4*np.pi*f*t)+0.17*np.cos(6*np.pi*f*t)+0.13*np.cos(8*np.pi*f*t))   # bruit primaire
186
187         a+=L
188
189         s1_rec = np.concatenate([s1_rec[-30*SMPFREQ:],s1])
190         s2_rec = np.concatenate([s2_rec[-30*SMPFREQ:],s2])
191         s3_rec = np.concatenate([s3_rec[-30*SMPFREQ:],s3])
192         s4_rec = np.concatenate([s4_rec[-30*SMPFREQ:],s4])
193
194         outdata[:] = np.array([s3,s4]).transpose()
195
196     with sd.Stream(samplerate=SMPFREQ, blocksize=L, latency='low',
```

```
                    channels=2, callback=callback):
        if mesure:
            while True:
                if a>=DELAI_ACTIVATION_CONTROLE_ACTIF*SMPFREQ and A1!=None :
                    A2 = (2**.5)*np.mean(s2_rec[-L:].astype(np.float)**2)**.5
                    print("Gain instantané :",20*np.log10(A2/A1))
                sd.sleep(1000)
        elif duree==None:
            print('#' * 80)
            print('Appuyer sur Entrée pour quitter')
            print('#' * 80)
            input()
        else:
            sd.sleep(int(duree*1000))

def controleactif_sine(f, duree=None, mesure=False):
    ## FONCTION PRINCIPALE DU CONTROLE ACTIF DANS LE CADRE D'UN BRUIT SINUSOIDAL PUR
    global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec,r1,phi1,A1
    BUFFER = np.zeros(FFTSIZE*2)
    a=0

    ## mesure de Hp(f)
    clear_rec()
    nsamples = MARGE_REC+3*SUB
    engine1(f,nsamples/SMPFREQ)
    r_p, phi_p = transfer_sine(s1_rec[MARGE_REC:MARGE_REC+SUB],s2_rec[MARGE_REC:MARGE_REC+SUB],f,SMPFREQ)
    #print("Hp(f) OK")

    ## mesure de Hs(f)
    clear_rec()
    nsamples = MARGE_REC+3*SUB
    engine2(f,nsamples/SMPFREQ)
    r_s, phi_s = transfer_sine(s4_rec[MARGE_REC:MARGE_REC+SUB],s2_rec[MARGE_REC:MARGE_REC+SUB],f,SMPFREQ)
    #print("Hs(f) OK")

    '''## mesure de Hf(f)
    clear_rec()
    nsamples = MARGE_REC+3*SUB
    engine3(f,nsamples/SMPFREQ)
    r_f, phi_f = transfer_sine(s4_rec[MARGE_REC:MARGE_REC+SUB],s1_rec[MARGE_REC:MARGE_REC+SUB],f,SMPFREQ)
    #print("Hf(f) OK")'''
    # remarque : on ne mesure pas Hf ici : le retour du bruit secondaire dans le micro de réf n'est pas pris en compte

    ## calcul du gain et du déphasage (par rapport au signal de réf) à appliquer au contre-bruit
    r_f,phi_f=0,0
    z = r_f*np.exp(phi_f*1j) - (r_s/r_p)*np.exp((phi_s-phi_p)*1j)
    R = 1/np.abs(z)
    PHI = - np.angle(z)

```

```
## c'est parti
L=FFTSIZE   # taille d'un bloc
BUFFER = np.zeros(2*L)
a=0
clear_rec()
r1,phi1,A1=None,None,None

def callback(indata, outdata, frames, time, status):
    global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec,r1,phi1,A1
    if status:
        print(status)

    s1 = np.array(indata).transpose()[0]
    s2 = np.array(indata).transpose()[1]
    s3 = np.zeros(L)
    s4 = np.zeros(L)


    t = np.linspace(a/SMPFREQ,(a+L)/SMPFREQ,num=L,endpoint=False)
    s3 = AMP_REC*np.cos(2*np.pi*f*t)
    if a>=DELAI_ACTIVATION_CONTROLE_ACTIF*SMPFREQ :
        if A1==None:A1 = (2**.5)*np.mean(s2.astype(np.float)**2)**.5 # amplitude avant contrôle actif
        if r1==None:r1,phi1 = transfer_sine(np.cos(2*np.pi*f*t), s1, f, SMPFREQ) # calcul de l'amplitude et de la phase de s1
        s4 = R*r1*np.cos(2*np.pi*f*t + phi1+PHI) # S4 = W*S1
    a+=L

    s1_rec = np.concatenate([s1_rec[-30*SMPFREQ:],s1])
    s2_rec = np.concatenate([s2_rec[-30*SMPFREQ:],s2])
    s3_rec = np.concatenate([s3_rec[-30*SMPFREQ:],s3])
    s4_rec = np.concatenate([s4_rec[-30*SMPFREQ:],s4])


    outdata[:] = np.array([s3,s4]).transpose()

with sd.Stream(samplerate=SMPFREQ, blocksize=L, latency='low',
               channels=2, callback=callback):
    if mesure:
        while True:
            if a>=DELAI_ACTIVATION_CONTROLE_ACTIF*SMPFREQ and A1!=None :
                A2 = (2**.5)*np.mean(s2_rec[-L:].astype(np.float)**2)**.5
                print("Gain instantané :",20*np.log10(A2/A1))
            sd.sleep(1000)
    elif duree==None:
        print('#' * 80)
        print('Appuyer sur Entrée pour quitter')
        print('#' * 80)
        input()
    else:
        sd.sleep(int(duree*1000))
```

```
295
296 def controleactif(f=220, duree=None, mesure=False):
297     ## FONCTION PRINCIPALE DU CONTROLE ACTIF POUR UN BRUIT COMPOSE
298     try: W
299     except NameError:
300         print("erreur : contrôle non initialisé (init_controleactif())")
301         return
302
303
304     w = np.fft.irfft(W)
305     engine4(w,f, duree=duree, mesure=mesure)
306
307
308 def init_controleactif():
309     ## initialise Hp, Hs et Hf
310     global Hp,Hs,Hf,W
311     global s1_rec,s2_rec,s3_rec,s4_rec
312     fmin,fmax=100,2000
313     fftfreq = np.fft.rfftfreq(FFTSIZE,1/SMPFREQ)    # fréquences (Hz) de la FFT
314     nsamples = MARGE_REC+3*SUB    # durée (échantillons)
315
316     ## mesure de Hp
317     Hp = np.zeros(len(fftfreq), dtype=complex)
318     for i in range(len(fftfreq)):
319         f=fftfreq[i]
320         if fmin<=f<=fmax:
321             clear_rec()
322             engine1(f,duree=nsamples/SMPFREQ)
323             r,phi = transfer_sine(s1_rec[MARGE_REC:MARGE_REC+SUB],s2_rec[MARGE_REC:MARGE_REC+SUB],f,SMPFREQ)
324             Hp[i] = r*np.exp(phi*1j)
325
326     ## mesure de Hs et de Hf
327     Hs = np.zeros(len(fftfreq), dtype=complex)
328     Hf = np.zeros(len(fftfreq), dtype=complex)
329     for i in range(len(fftfreq)):
330         f=fftfreq[i]
331         if fmin<=f<=fmax:
332             clear_rec()
333             engine2(f,duree=nsamples/SMPFREQ)
334
335             r,phi = transfer_sine(s4_rec[MARGE_REC:MARGE_REC+SUB],s2_rec[MARGE_REC:MARGE_REC+SUB],f,SMPFREQ)
336             Hs[i] = r*np.exp(phi*1j)
337
338             r,phi = transfer_sine(s4_rec[MARGE_REC:MARGE_REC+SUB],s1_rec[MARGE_REC:MARGE_REC+SUB],f,SMPFREQ)
339             Hf[i] = r*np.exp(phi*1j)
340
341     ## calcul du filtre
342     W = [1/(Hf[i]-Hs[i]/Hp[i]) if Hp[i]!=0 else 10**(-48/20) for i in range(len(Hp))] # W = 1/(Hf-Hs/Hp)    # 10**(-48/20) vaut -48dB
343
```

```
344  def bode(H):
345      ## diagramme de Bode de H
346      F = np.fft.rfftfreq(FFTSIZE,1/SMPFREQ)
347      G = 20*np.log10(np.abs(H))
348      P = np.angle(H, deg=True)
349      plt.subplot(211)
350      plt.semilogx(F,G)
351      plt.title("Gain (dB)")
352      plt.grid()
353      plt.subplot(212)
354      plt.semilogx(F,P)
355      plt.title("Phase (°)")
356      plt.grid()
357      plt.show()
358
359  def stats(n=10,f_list=[440]):
360      ## mesure les gains obtenus pour des bruits primaires purs de différentes fréquences
361      G_list = [ [] for f in f_list ]
362      for j in range(len(f_list)):
363          f=f_list[j]
364          for i in range(n):
365              clear_rec()
366              controleactif_sine(f,duree=DELAI_ACTIVATION_CONTROLE_ACTIF+4*SUB/SMPFREQ)
367              A1 = (2**.5)*np.mean(s2_rec[MARGE_REC:MARGE_REC+SUB].astype(np.float)**2)**.5 # amplitude avant contrôle actif
368              A2 = (2**.5)*np.mean(s2_rec[-SUB:].astype(np.float)**2)**.5 # amplitude après contrôle actif
369              G = 20*np.log10(A2/A1)
370              G_list[j].append(G)
371          print("Gain moyen pour f=",f,":",np.mean(G_list[j]))
372
373  def clear_rec():
374      global s1_rec,s2_rec,s3_rec,s4_rec
375      s1_rec = np.array([])
376      s2_rec = np.array([])
377      s3_rec = np.array([])
378      s4_rec = np.array([])
379
380  ## EXEMPLE D'UTILISATION :
381  #init_controleactif()
382  #bode(W)
383  #controleactif(f=440,mesure=True)
384  #stats(n=6,f_list=[262,330,440,523,659,880,1046.5,1318.5,1760])
```

```
'''
Simulation sources ponctuelles primaires et secondaires
calcule gain pour angles théta à r=20cm pour expérience
'''
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.animation as animation


cst_c = 340.0
cst_minfloat = 10**(-48)
cst_debit = 1
cst_masvol = 1.225
cst_p0 = 20*10**(-6)

def clamp(x,m,M):
    if x<m: return m
    if x>M: return M
    return x

class Source:
    def __init__(self,x,y,pulsation,phase,debit):
        self.__x = x
        self.__y = y
        self.__pulsation = pulsation
        self.__phase = phase
        self.__debit = debit
    def dist(self,x,y):
        return ((self.__x-x)**2+(self.__y-y)**2)**.5
    def champ(self,x,y,t=0):
        r = self.dist(x,y)
        return self.__pulsation*cst_masvol*self.__debit*np.exp(1j*(self.__pulsation*t+self.__phase))*np.exp(-1j*self.__pulsation*r/cst_c)/(4*np.pi*r)


class Simulation:
    def __init__(self, nom=""):
        self.__sources_primaires = []
        self.__sources_secondaires = []

        self.nom = nom

    def ajouterSource(self,x,y,pulsation,phase=0,debit=cst_debit,primaire=False):
        S = Source(x,y,pulsation,phase,debit)
        if primaire :
            self.__sources_primaires.append(S)
        else :
            self.__sources_secondaires.append(S)

    def transfert(self,x,y,t=0):
```

```
        A = sum([s.champ(x,y,t) for s in self.__sources_primaires])
        if A==0: A=cst_minfloat
        B = A+sum([s.champ(x,y,t) for s in self.__sources_secondaires])
        return B/A

    def champ(self,x,y,t=0):
        return sum([s.champ(x,y,t) for s in self.__sources_primaires])+sum([s.champ(x,y,t) for s in self.__sources_secondaires])

    def graph(self,xmin=-3,xmax=3,ymin=-3,ymax=3,pas=0.05,titre=""):
        if titre=="" : titre = self.nom
        x = np.arange(xmin,xmax,pas)
        y = np.arange(ymin,ymax,pas)
        X,Y = np.meshgrid(x,y)
        #Z = np.array([ [10*np.log(self.transfert(x[i],y[j])**2) for i in range(len(x))] for j in range(len(y))])
        #Z = np.array([ [10*np.log((self.champ(x[i],y[j])/cst_p0)**2) for i in range(len(x))] for j in range(len(y))])
        Z = np.array([ [ clamp(20*np.log10(abs(self.transfert(x[i],y[j]))),-48,48) for i in range(len(x))] for j in range(len(y))])
        im = plt.pcolormesh(X,Y,Z, cmap="coolwarm", vmin=-48, vmax=48)
        plt.xlabel("x (m)")
        plt.ylabel("y (m)")
        plt.colorbar(im, label="Gain (dB)")
        if titre!=None : plt.title(titre)
        plt.axis('equal')
        #plt.show()


if __name__ == '__main__':
    simulation = Simulation("Expérience 27/11/2018 essai 2 : d=2.0cm f=40kHz")
    d = 0.02
    simulation.ajouterSource(0,-d/2,2*np.pi*40000,phase=0,primaire=True)
    simulation.ajouterSource(0,d/2,2*np.pi*40000,phase=0)
    simulation.graph(pas=0.003,xmin=-0.5,xmax=0.5,ymin=-0.5,ymax=0.5)
    plt.axhline(0, color='black',lw=1)
    plt.axvline(0, color='black',lw=1)
    plt.scatter([0,0],[-d/2,d/2],c="red")
    for theta in range(46):
        x = 0.20*np.cos(theta*np.pi/180)
        y = 0.20*np.sin(theta*np.pi/180)
        plt.scatter(x,y,c="white",alpha=0.5,edgecolors="black")
        print("angle",theta,":",20*np.log10(abs(simulation.transfert(x,y))))
    plt.grid()
    plt.show()
```