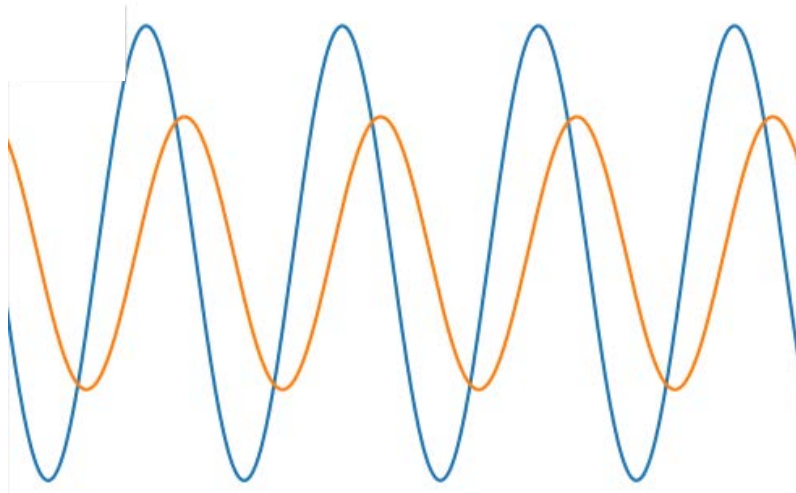


# Contrôle actif du bruit acoustique

candidat 1216



# Acoustique linéaire non dissipative des milieux stationnaires

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

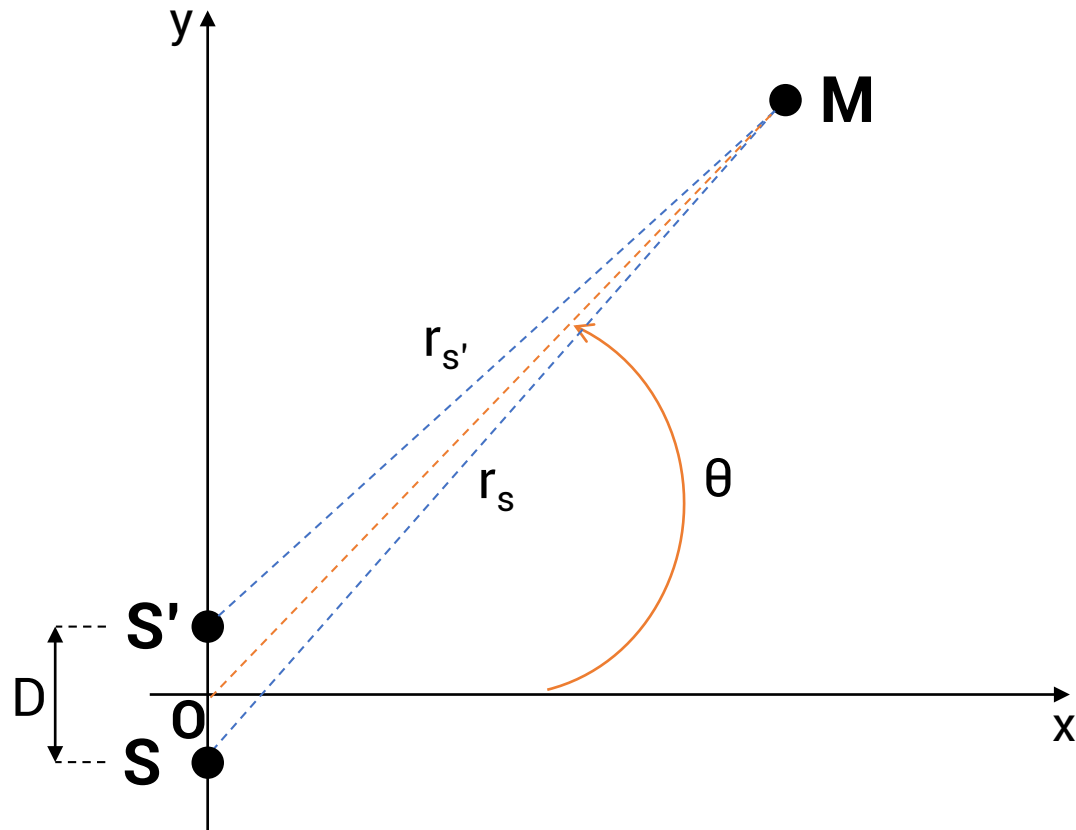
Equation de Helmholtz :  $\Delta \underline{p_m} + k^2 \underline{p_m} = -j\omega \rho_0 \underline{q_m}$  où  $\underline{p} = \underline{p_m} e^{j\omega t}$   
 $\underline{q} = \underline{q_m} e^{j\omega t}$

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



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

$$q_s = Q_s e^{j\omega t}$$

$$p_{s'}(M) = \frac{j\omega\rho_0 q_{s'}}{4\pi} \frac{e^{-jkr_{s'}}}{r_{s'}}$$

$$q_{s'} = Q_{s'} e^{j(\omega t + \phi)}$$

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

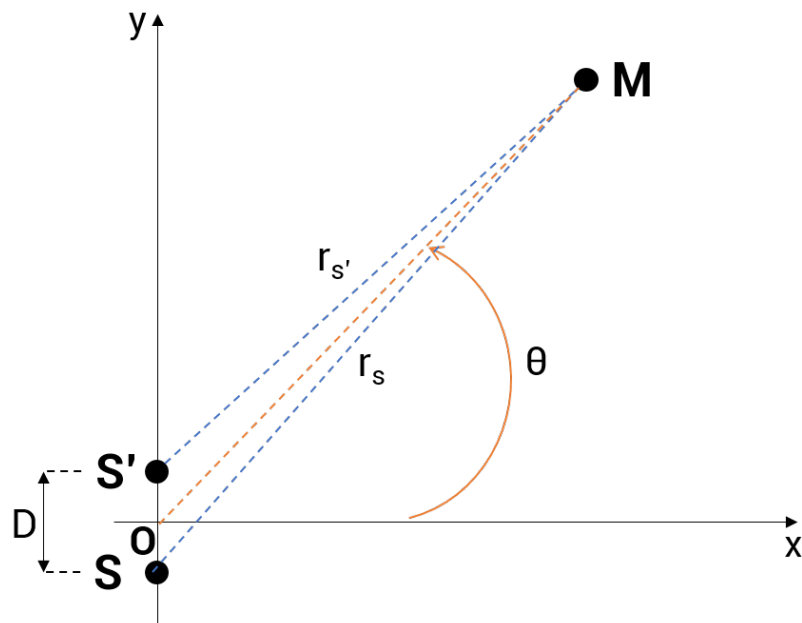
$$H(M) = \frac{p(M)}{p_s(M)} = 1 + \frac{p_{s'}(M)}{p_s(M)}$$

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

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

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

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



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

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

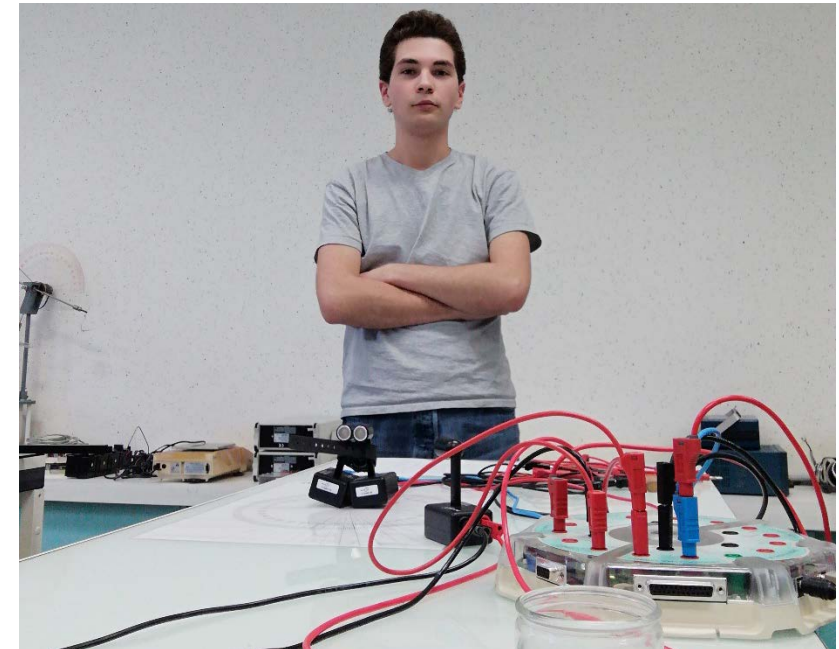
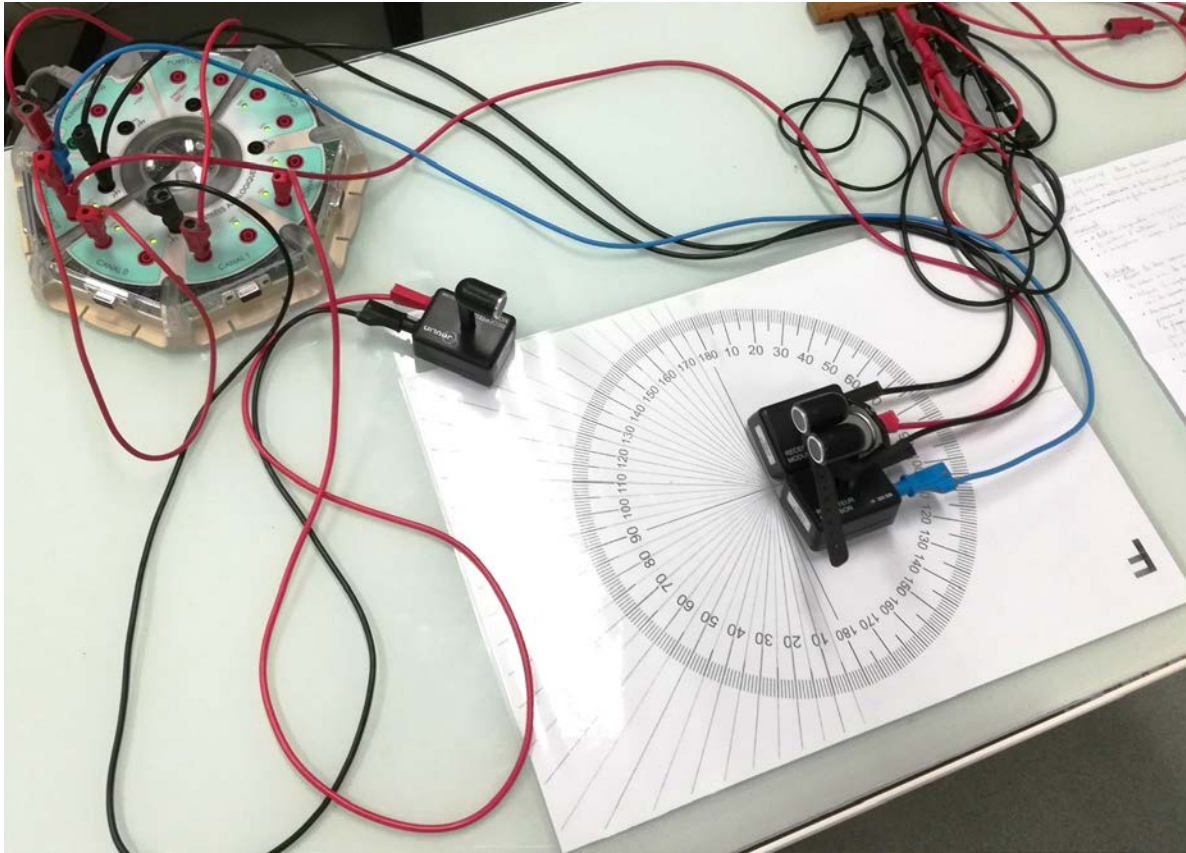
$$r_{s'} = r \left(1 - \frac{D}{2r} \sin \theta\right)$$

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

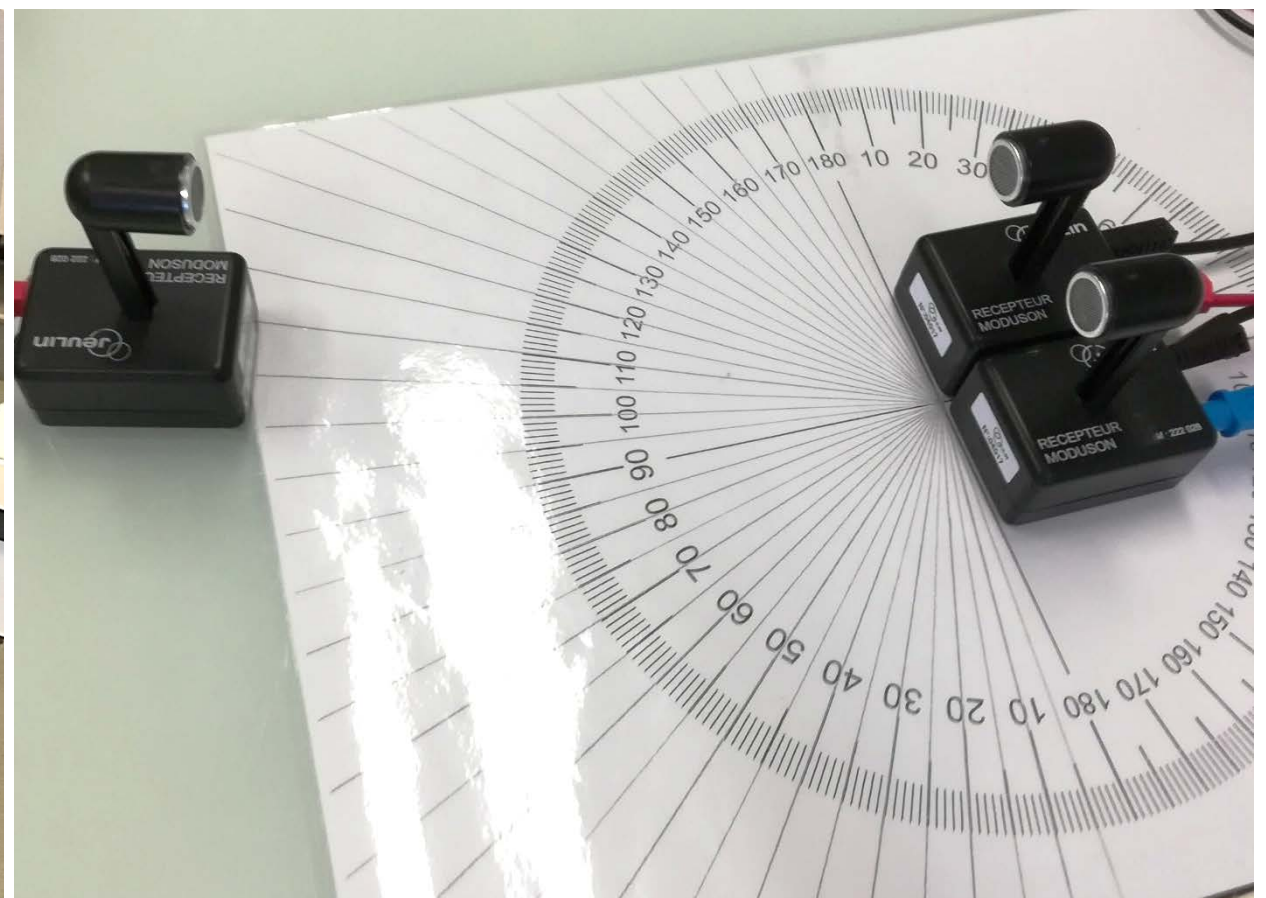
$$\overrightarrow{SM} = \overrightarrow{SO} + \overrightarrow{OM}$$

$$\begin{aligned} \overrightarrow{SM}^2 &= \overrightarrow{SO}^2 + \overrightarrow{OM}^2 + 2 \overrightarrow{SO} \cdot \overrightarrow{OM} \\ &= \frac{D^2}{4} + r^2 + 2 \cdot \frac{D}{2} r \cos\left(\frac{\pi}{2} - \theta\right) \\ &= \frac{D^2}{4} + r^2 + Dr \sin \theta \\ &= r^2 \left(1 + \frac{D}{r} \sin \theta + \frac{D^2}{4r^2}\right) \end{aligned}$$

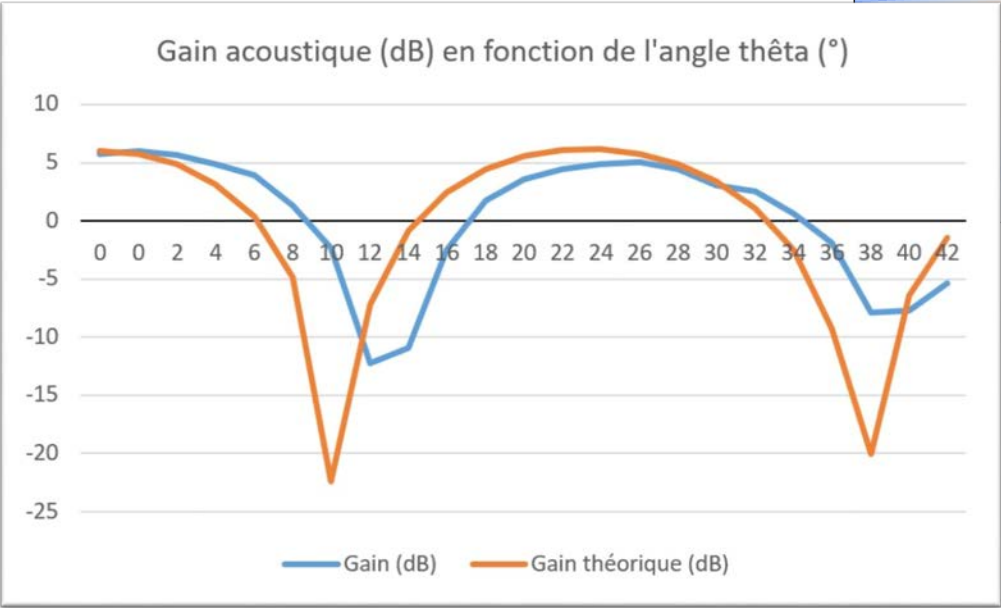
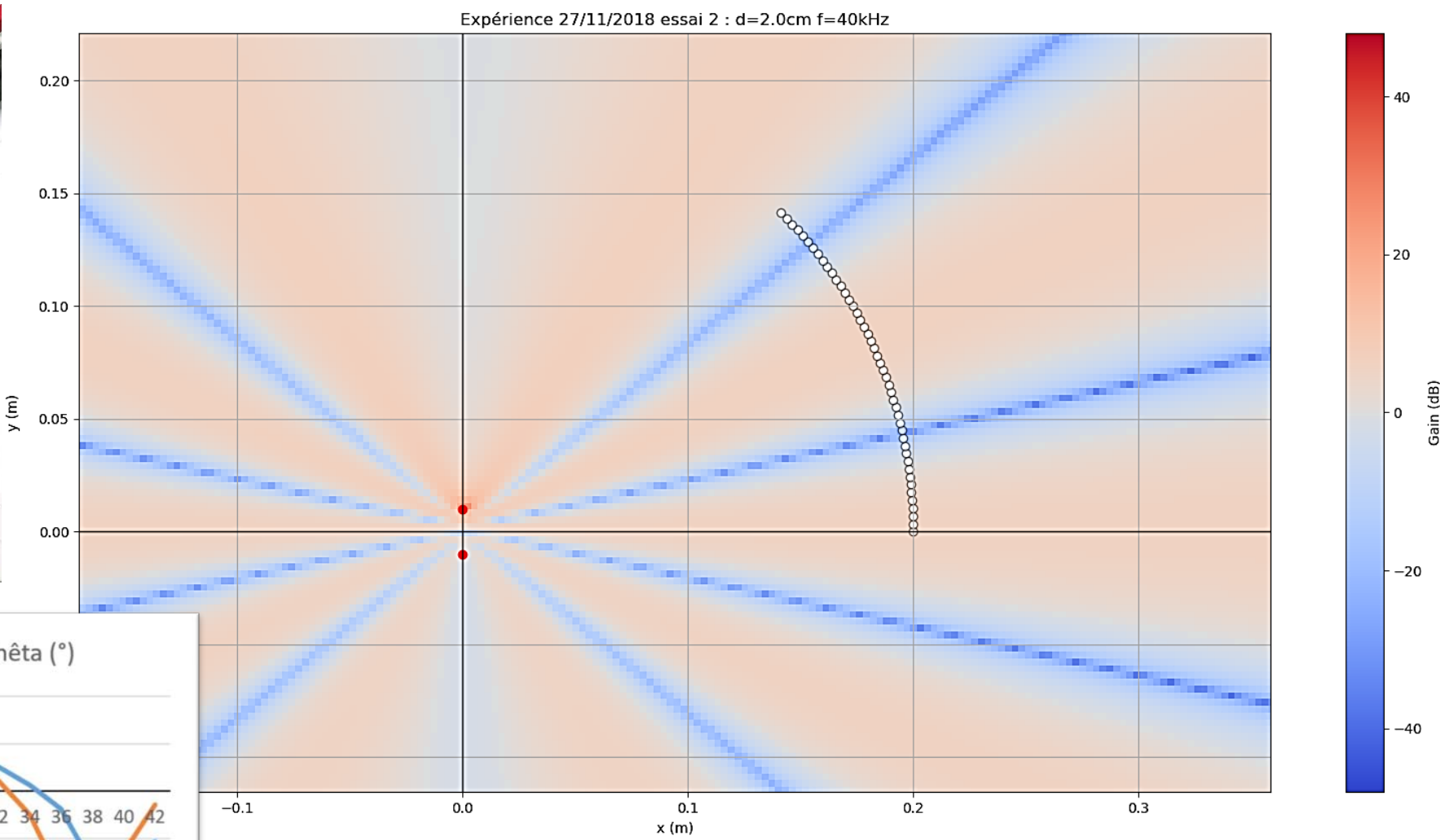
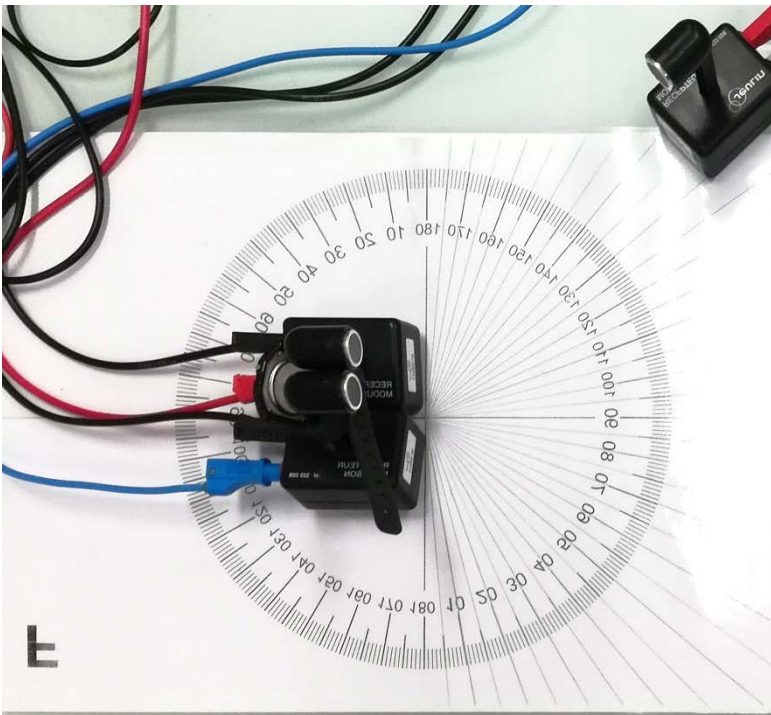
# Expérience du dipôle acoustique

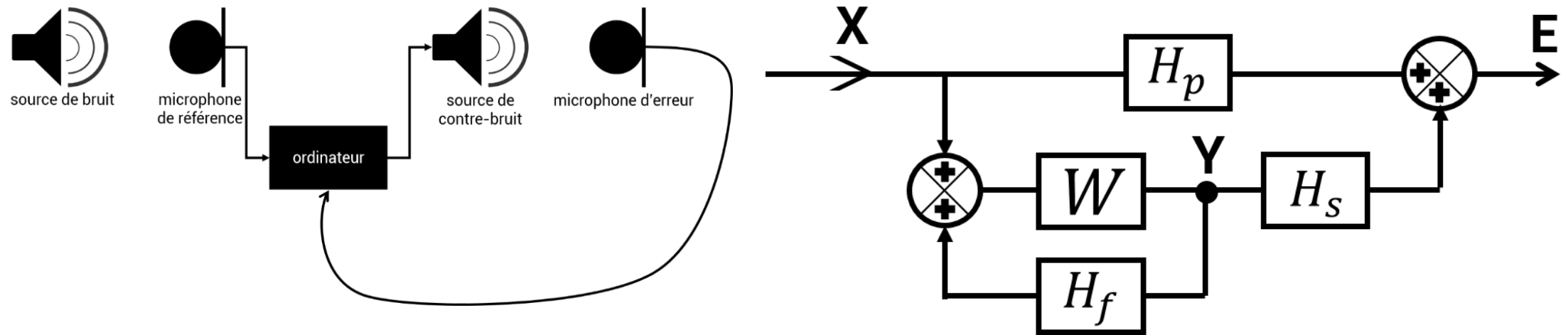






13	D (m)	r (m)	Theta (deg)	Sp (V)	Sps (V)	Gain (dB)	Gain théorique (dB) f=40kHz
14	2.00E-02	2.00E-01	0	0.295	0.571	5.73628185	3.0103
15			2	0.287	0.573	6.0054545	2.8721
16			4	0.296	0.565	5.61513474	2.4221
17			6	0.278	0.488	4.88750052	1.5865
18			8	0.276	0.433	3.91157629	0.1754
19			10	0.284	0.331	1.33019307	-2.4054
20			12	0.276	0.211	-2.3325325	-11.2371
21			14	0.265	0.065	-12.20665	-3.6319
22			16	0.257	0.022	-16.013007	-7.0272
23			18	0.247	0.007	-19.913007	-10.4272
24			20	0.237	0.002	-23.813007	-13.8272
25			22	0.227	0.001	-27.713007	-17.2272
26			24	0.217	0.000	-31.613007	-20.6272
27			26	0.207	0.000	-35.513007	-24.0272
28			28	0.197	0.000	-39.413007	-27.4272
29			30	0.187	0.000	-43.313007	-30.8272
30			32	0.177	0.000	-47.213007	-34.2272
31			34	0.167	0.000	-51.113007	-37.6272
32			36	0.157	0.000	-55.013007	-41.0272
33			38	0.147	0.000	-58.913007	-44.4272
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

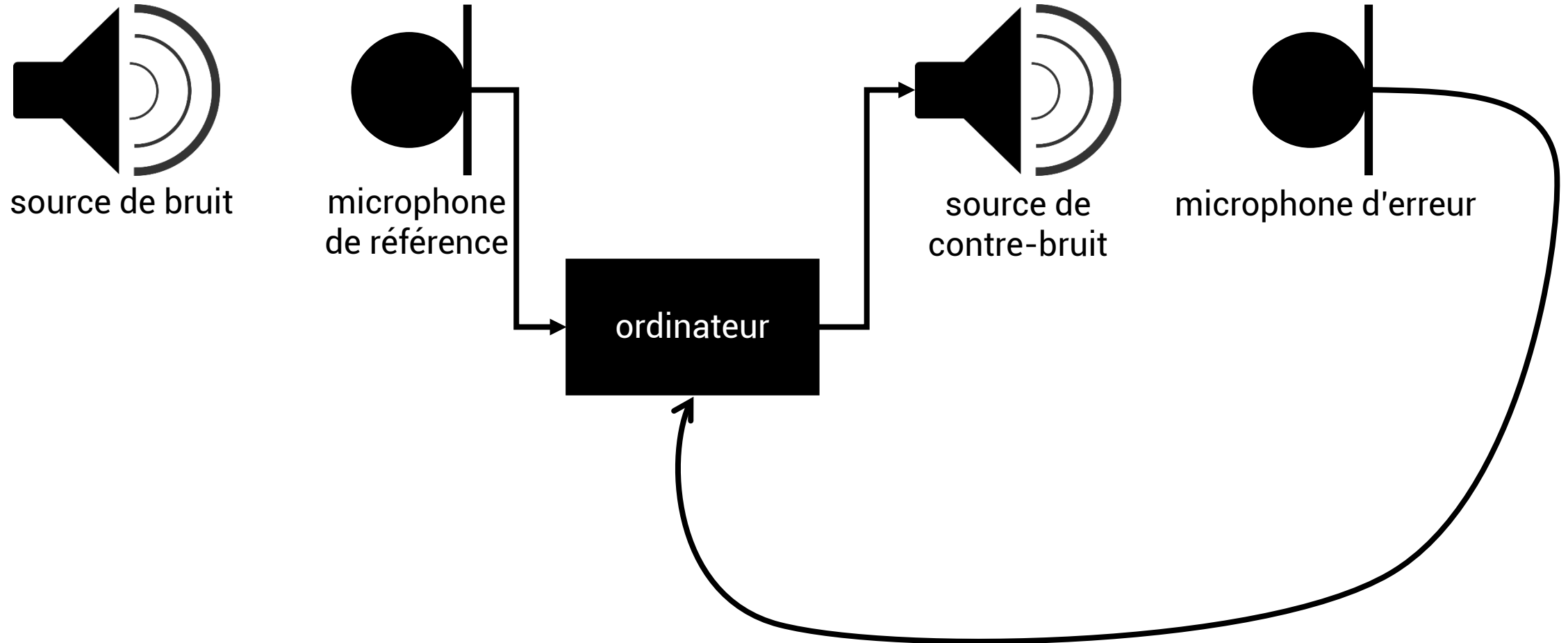




# Contrôle actif et automatique



# Contrôle monovoie



# Transformation de Fourier discrète

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

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

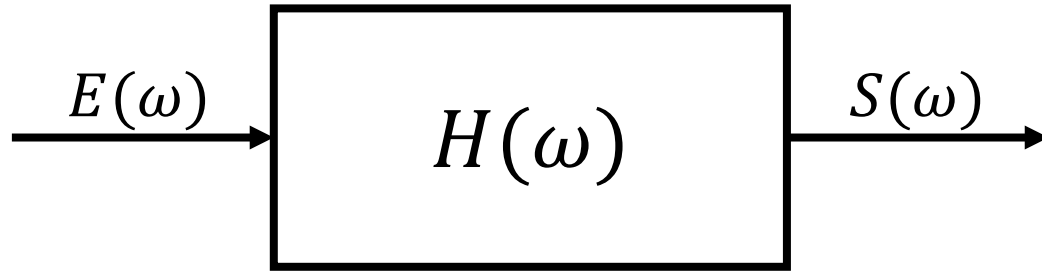
$$s(n) = \frac{1}{N} \sum_{k=0}^{N-1} S(k) e^{2i\pi n \frac{k}{N}}$$

Pour un signal réel,  $N/2+1$  « vraies » fréquences :

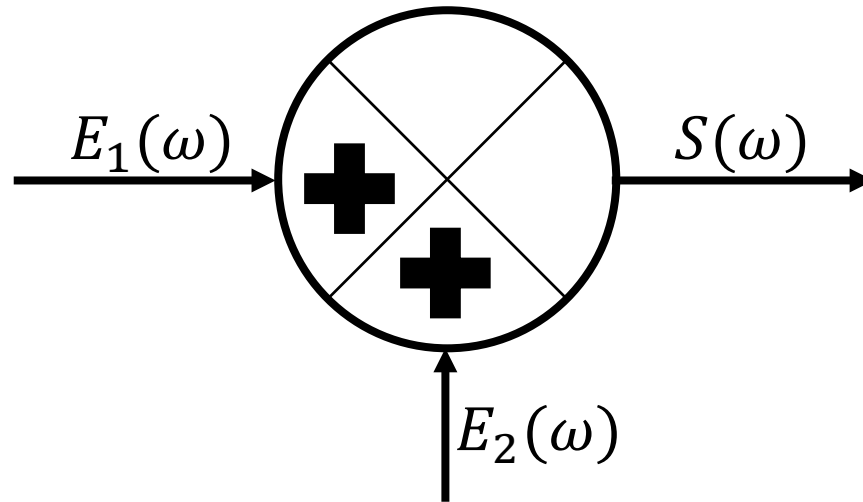
$$\forall k \in \left[0, \frac{N}{2}\right], f_k = k \frac{44100}{N} \text{ (Hz)}$$

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

# Systemes linéaires et invariants

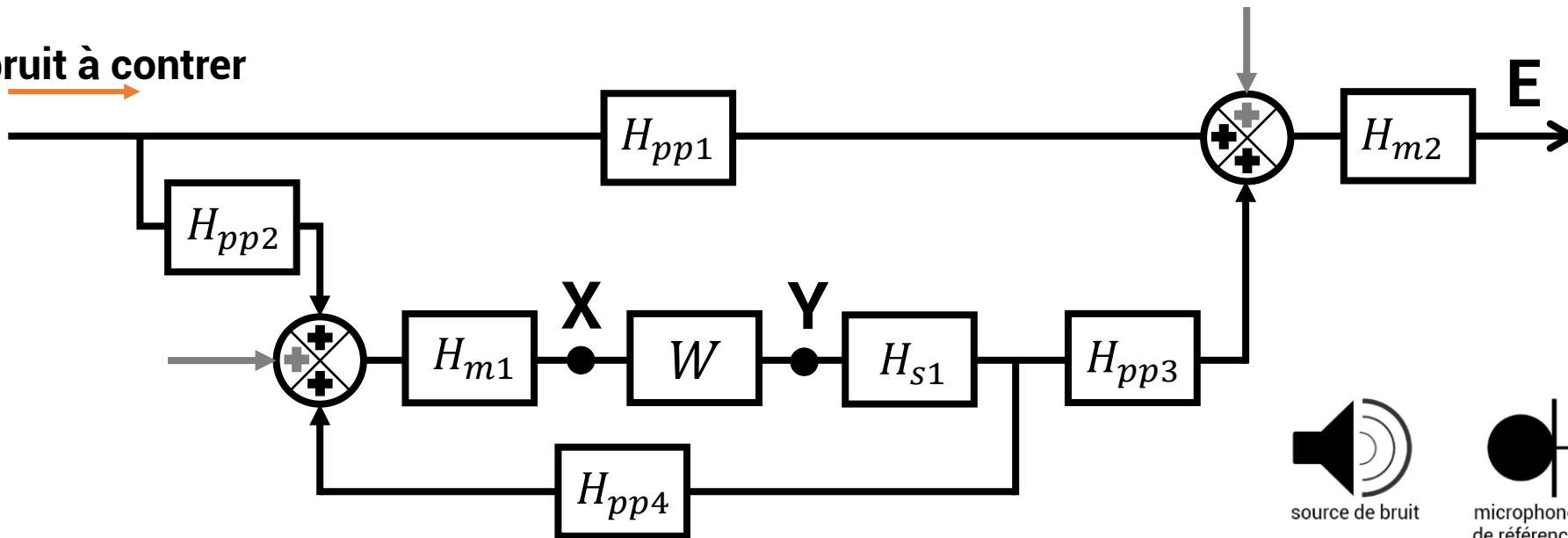


$$S(\omega) = H(\omega)E(\omega)$$



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

bruit à contrer



source de bruit



microphone de référence

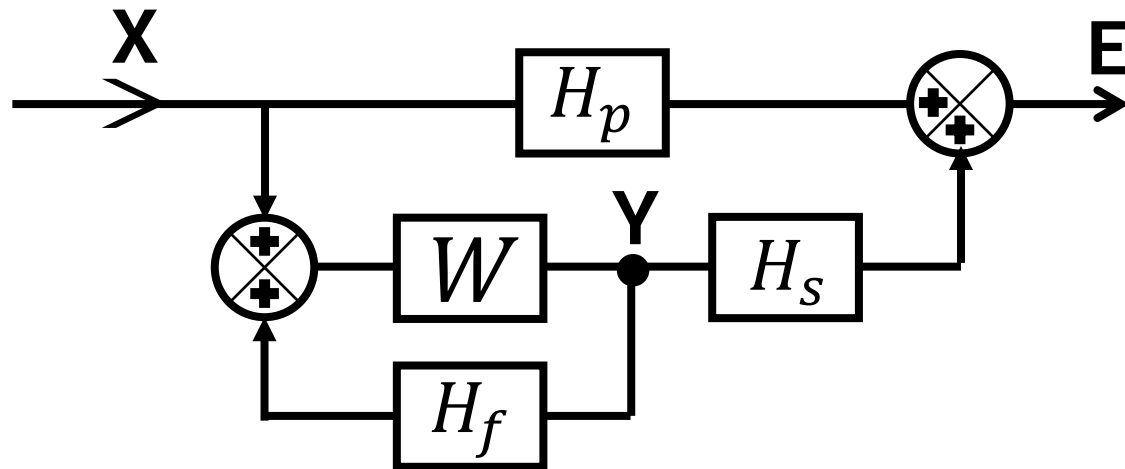


source de contre-bruit



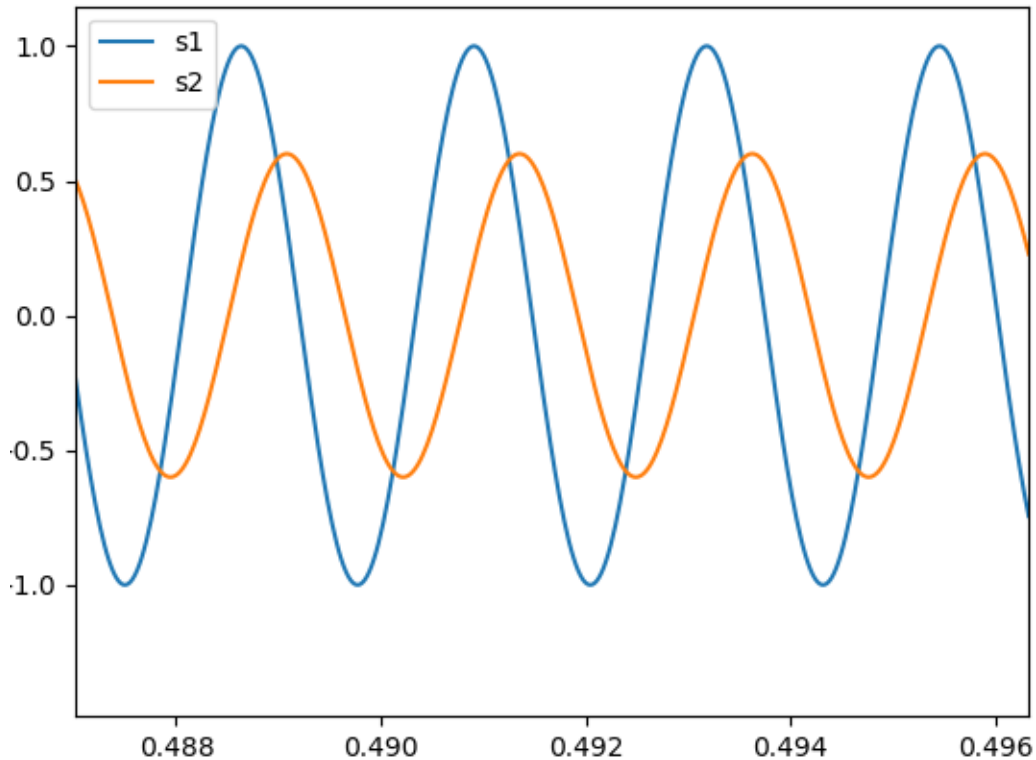
microphone d'erreur

ordinateur



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

# Mesure de fonctions de transfert



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

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

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



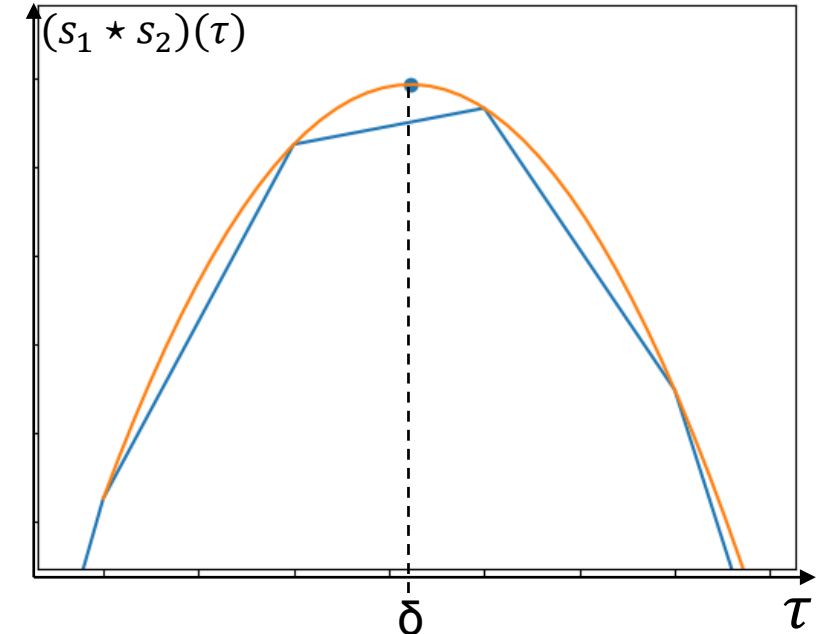
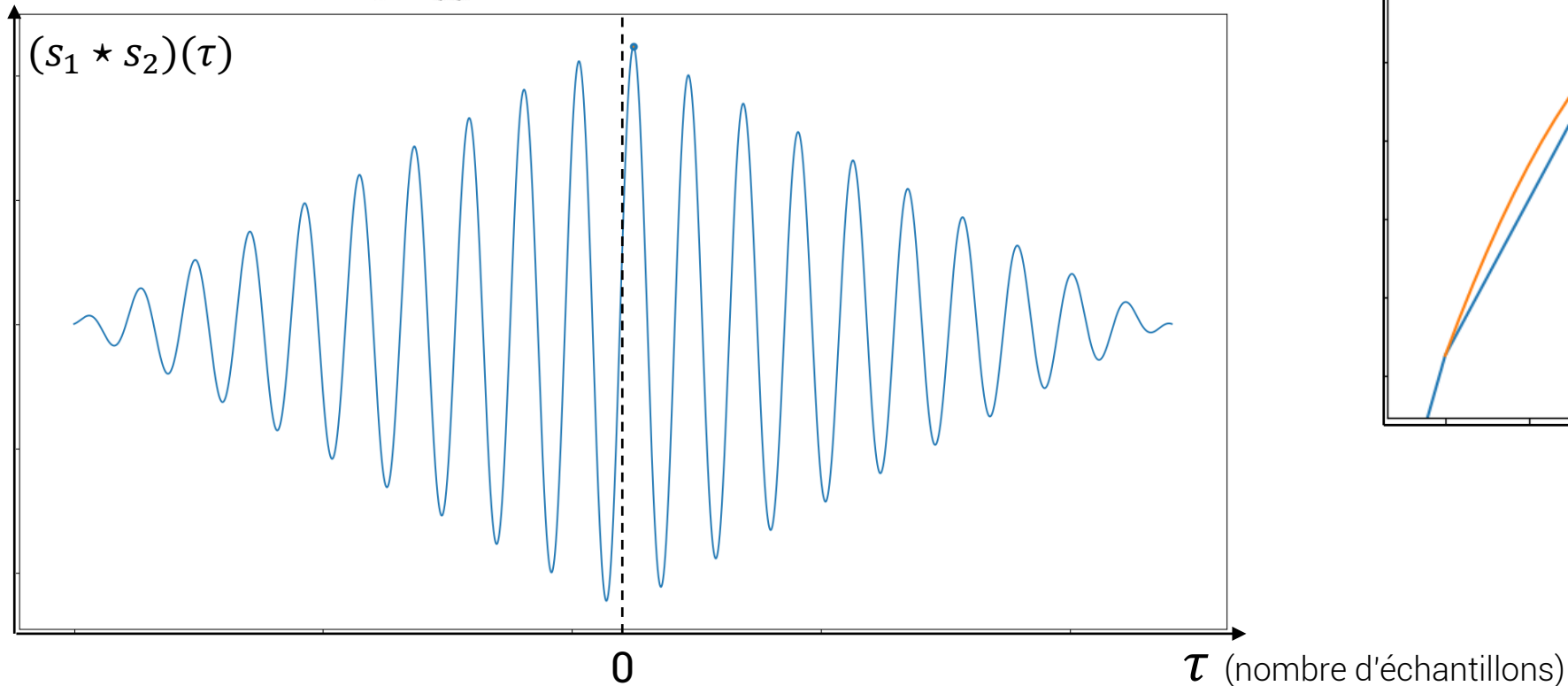
Mesure de  $\frac{A_2}{A_1}$

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

$$\text{Valeur efficace : } \sqrt{\langle s^2(t) \rangle} = \frac{A}{\sqrt{2}}$$

Mesure de  $\phi$  par corrélation croisée

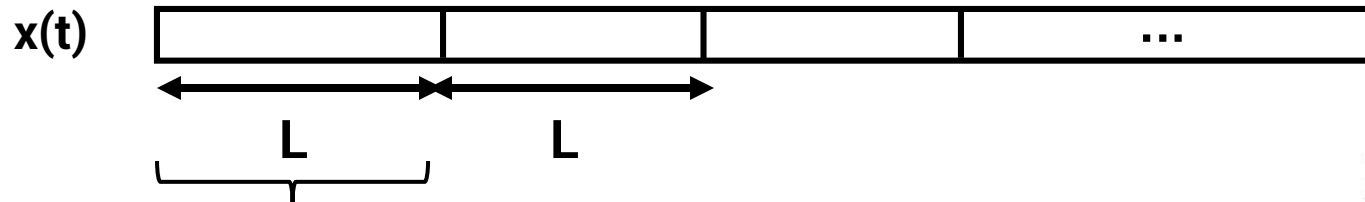
$$(f \star g)(\tau) = \int_{-\infty}^{\infty} f(t)g(t + \tau) dt$$



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

# Gestion de l'audio avec Python

voir fonctions engine



`callback(indata, outdata, ...)`

```
1  import numpy as np
2  import sounddevice as sd
3  a=0
4  def callback(indata, outdata, frames, time, status):
5      global a,s1_rec,s2_rec,s3_rec,s4_rec
6      if status:
7          print(status)
8
9      s1 = np.array(indata).transpose()[0]
10     s2 = np.array(indata).transpose()[1]
11
12     t = np.linspace(a/SMPFREQ, (a+L)/SMPFREQ,num=L,endpoint=False)
13     s3 = AMP_REC*np.cos(2*np.pi*f*t)
14
15     s4 = np.zeros(L)
16
17     a+=L
18
19     s1_rec = np.concatenate([s1_rec[-30*SMPFREQ:],s1])
20     s2_rec = np.concatenate([s2_rec[-30*SMPFREQ:],s2])
21     s3_rec = np.concatenate([s3_rec[-30*SMPFREQ:],s3])
22     s4_rec = np.concatenate([s4_rec[-30*SMPFREQ:],s4])
23
24     outdata[:] = np.array([s3,s4]).transpose()
25
26 with sd.Stream(samplerate=SMPFREQ, blocksize=L, latency='low',
27               channels=2, callback=callback):
28     sd.sleep(int(duree*1000))
29
```

# 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  $y_k(t)$

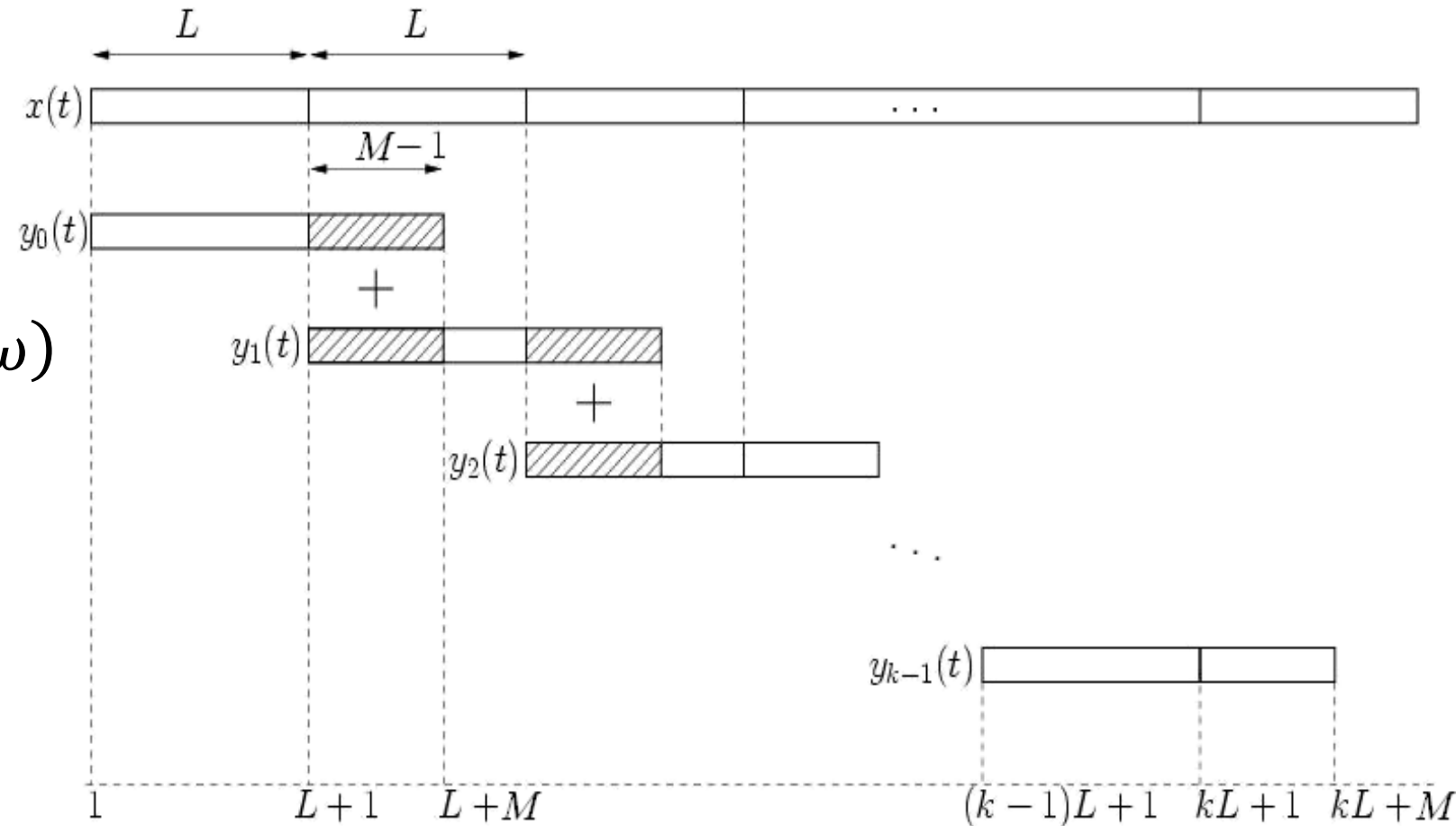
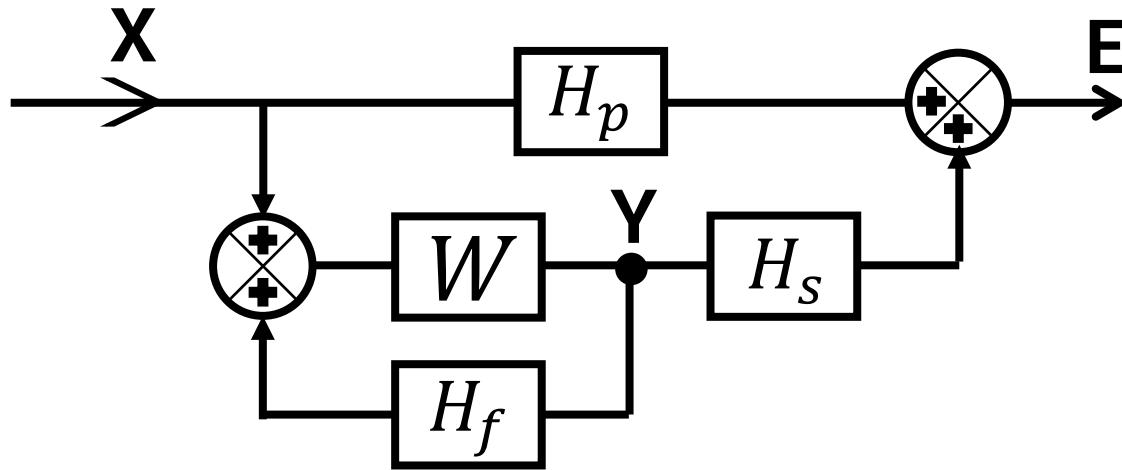


Illustration issue de l'article Wikipédia « Overlap-add method »



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

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

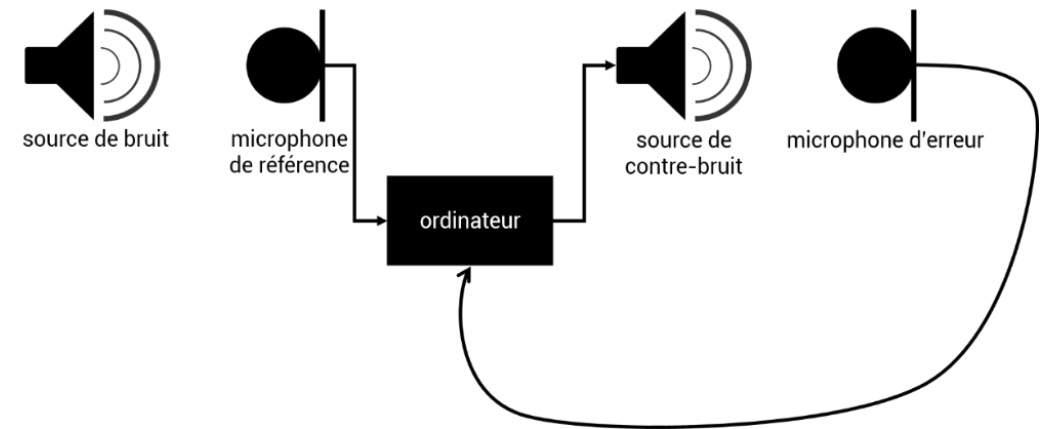
$$H_f(f) = \left. \frac{X(f)}{Y(f)} \right|_{\substack{X=0 \\ W=1}}$$

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

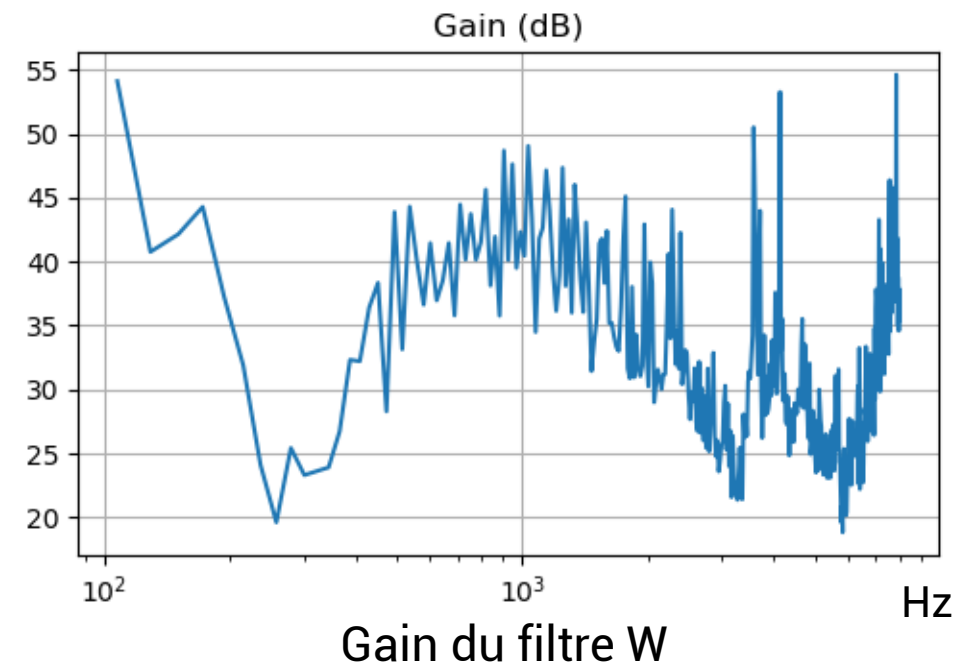
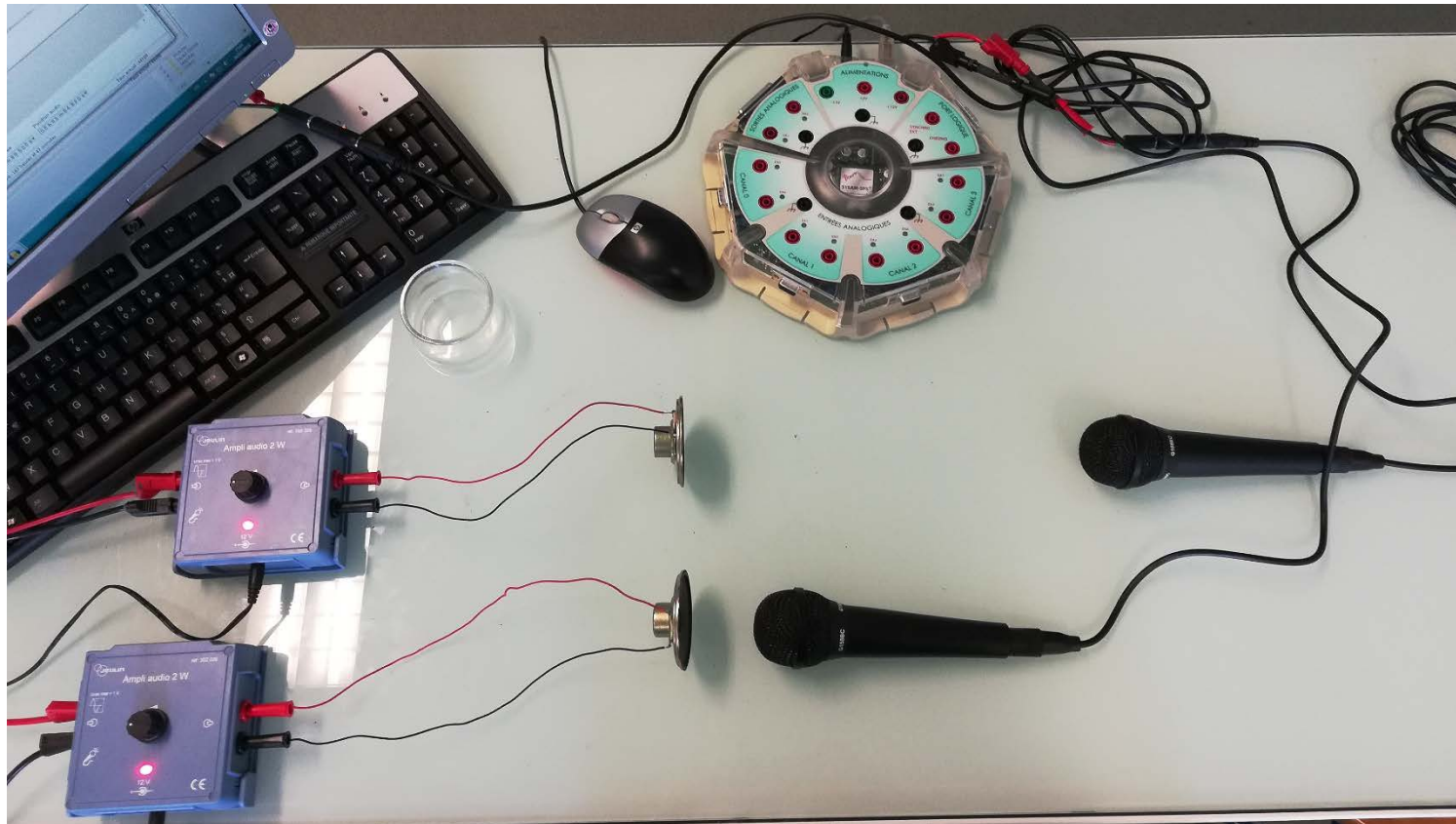
$$f \in \left\{ 0, 1 \cdot \frac{44100}{L}, 2 \cdot \frac{44100}{L}, \dots, \frac{L}{2} \cdot \frac{44100}{L} \right\} \text{ (Hz)}$$

ici, avec  $L=2048$  :

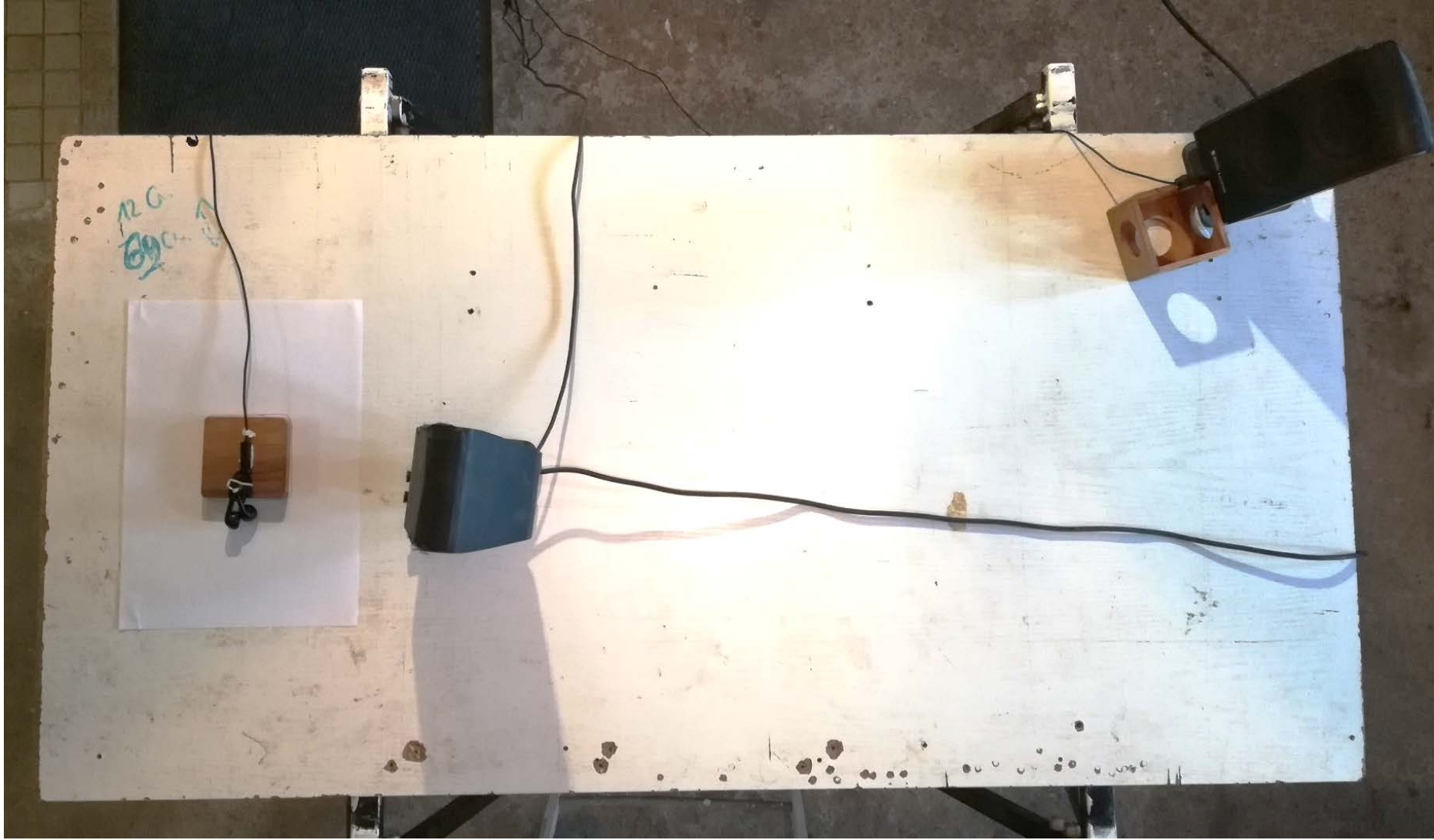
$$f \in \{0; 21,5; 43,1; \dots; 22028; 22050 \} \text{ (Hz)}$$

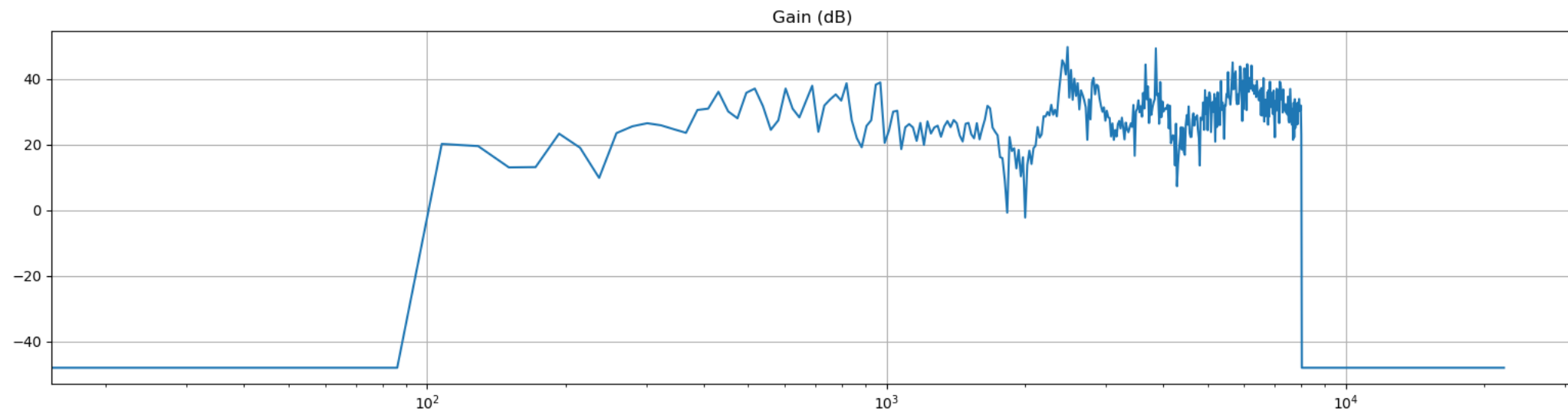
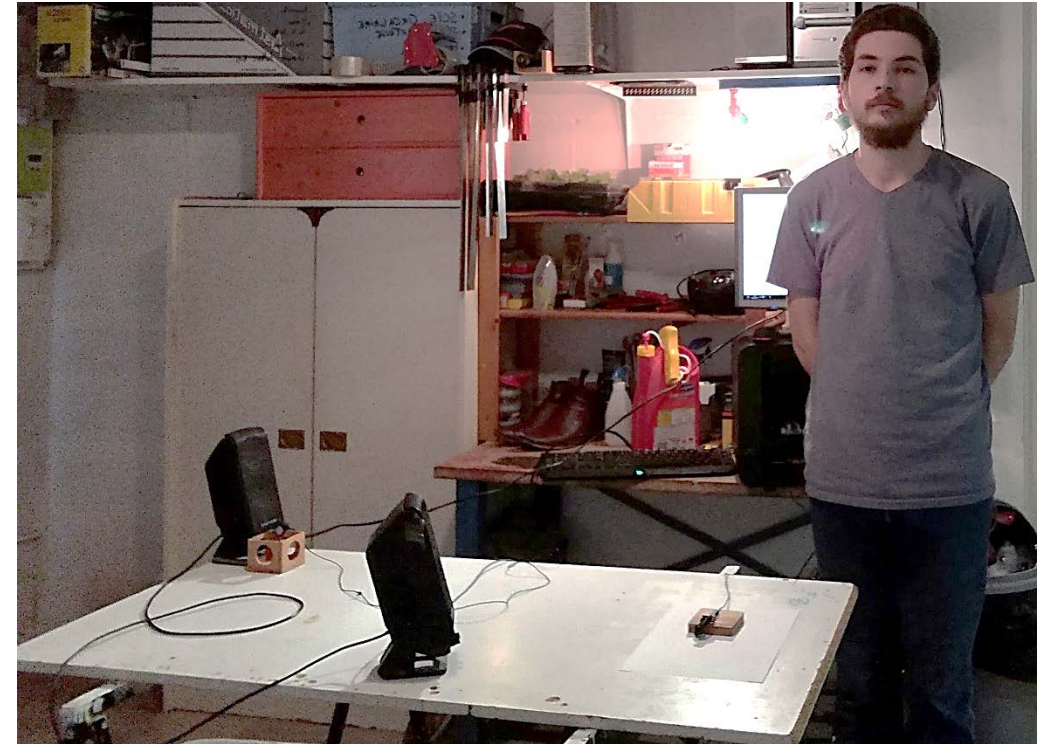
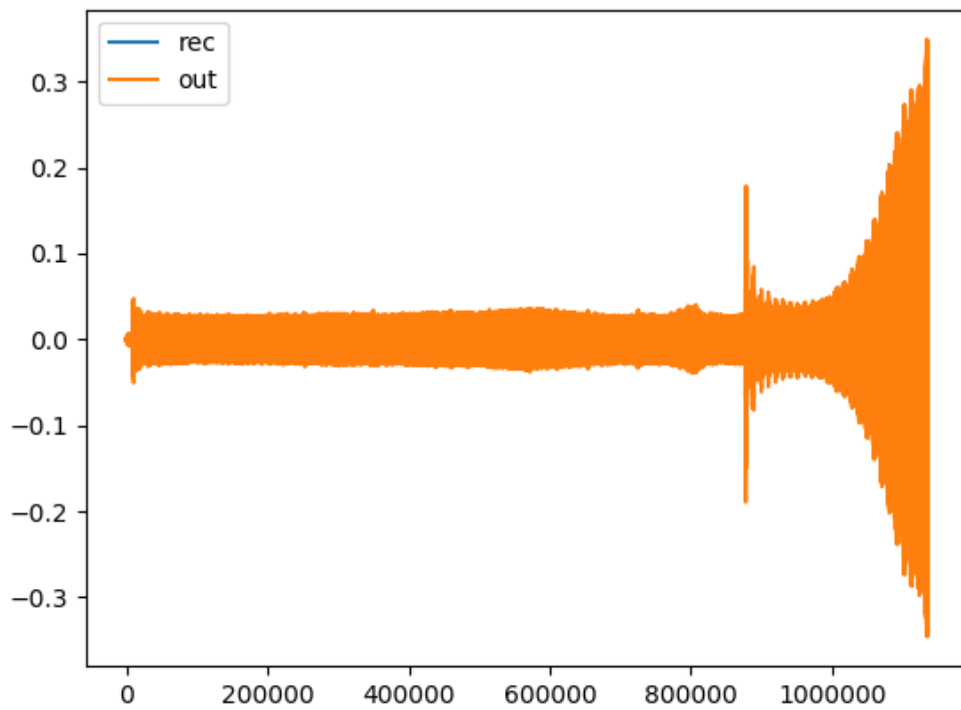


# Expérimentations









# Cas d'un bruit pur (sinusoïdal)



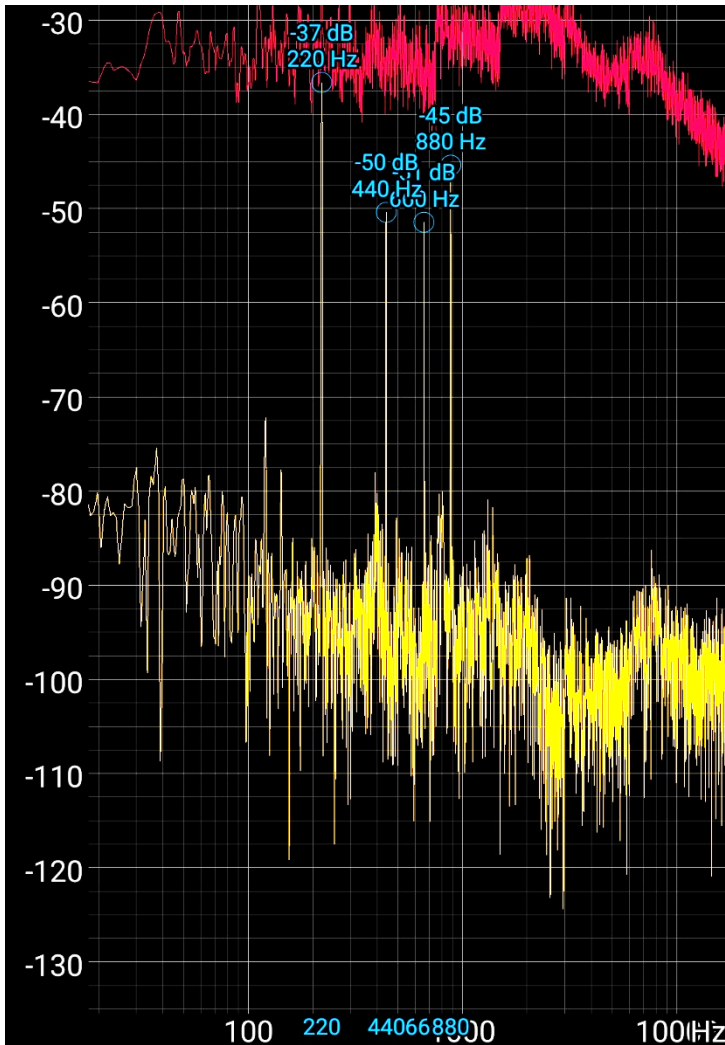
```
>>> stats(n=7,f_list=[262,330,440,523,659,880,1046.5,1318.5,1760])  
Gain moyen pour f= 262 : -5.417766635543484  
Gain moyen pour f= 330 : -4.279523190942039  
Gain moyen pour f= 440 : -2.74944354735907  
Gain moyen pour f= 523 : -4.18458499648263  
Gain moyen pour f= 659 : -1.9408427274437063 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

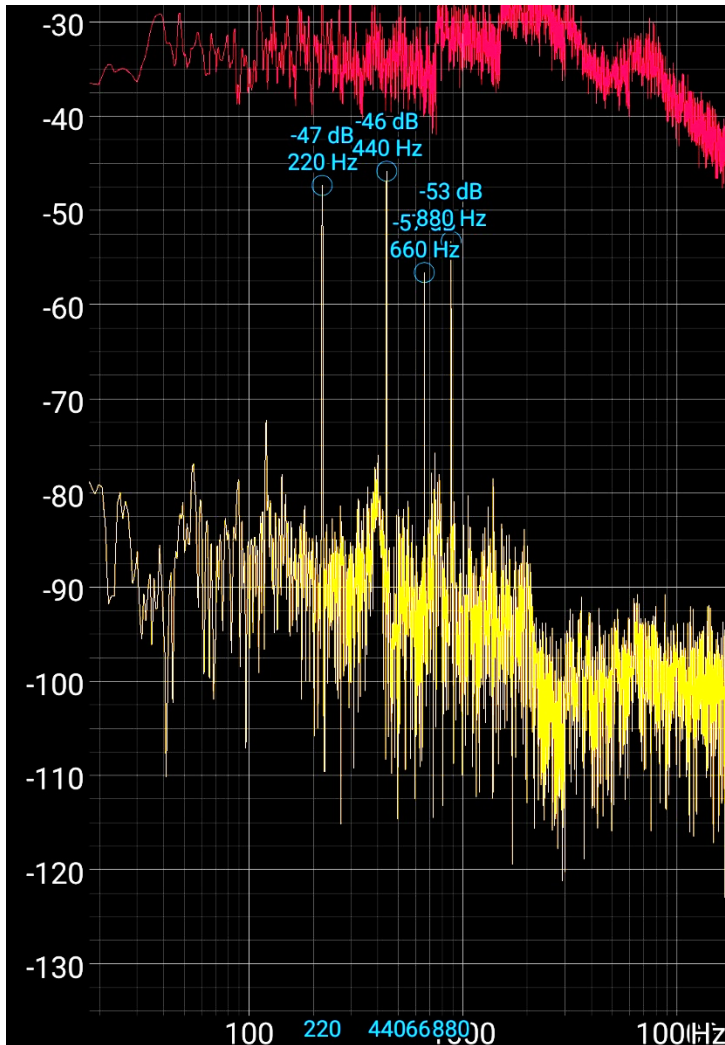
**sans contrôle actif**

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

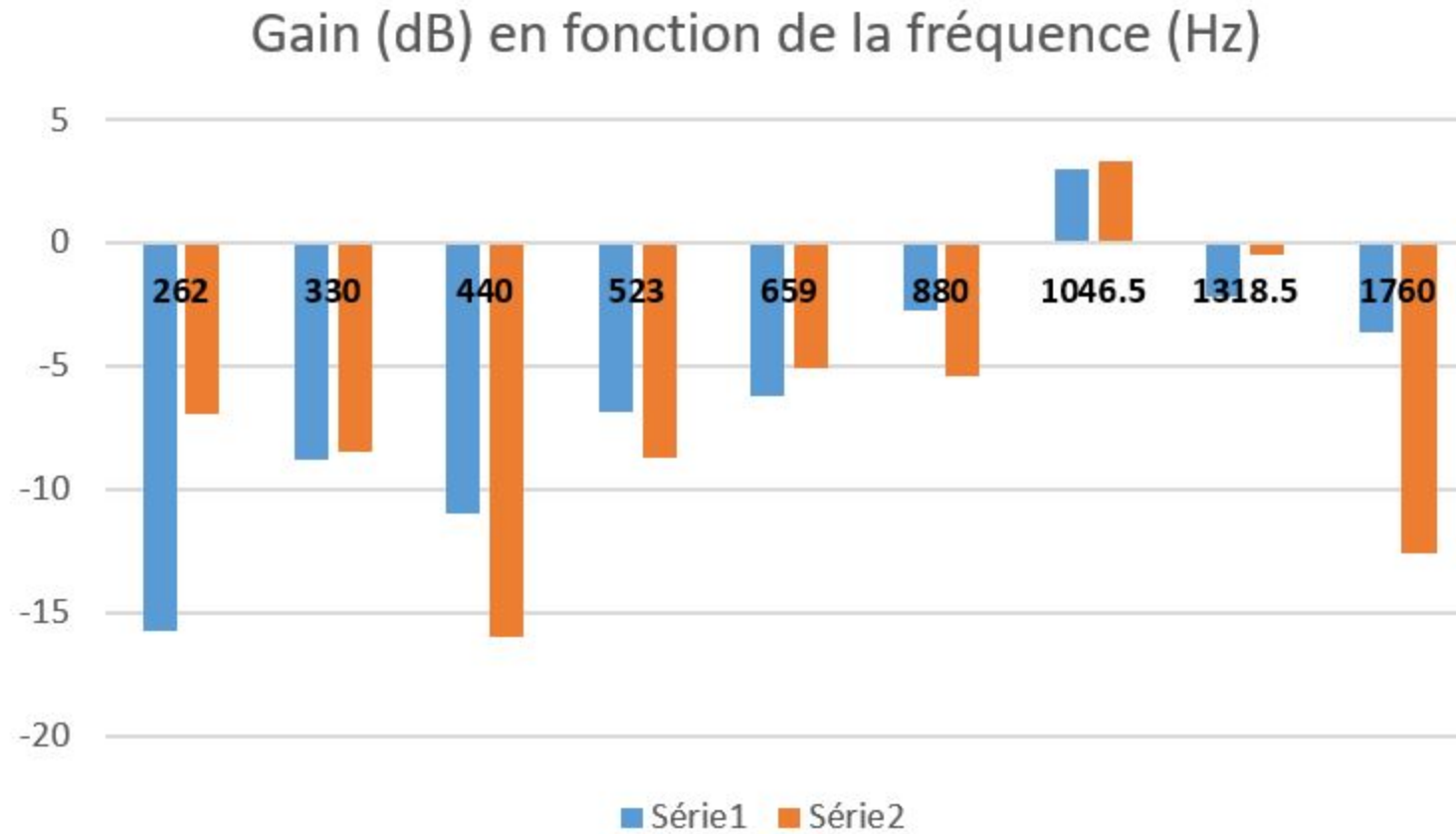


**avec contrôle actif**

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



et pour un bruit pur :





```

1 #####
2 ## IMPORTS et variables globales
3 from scipy.io import wavfile as wav
4 import scipy.signal as sgnl
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import os
8 from random import randint
9 import sounddevice as sd
10 from time import time
11
12 FFTSIZE = 2048 # nombre d'échantillons
13 SMPFREQ = 44100 # taux d'échantillonnage (44100 échantillons par seconde)
14 SUB = 16384 # nombre d'échantillons sur lesquels on mesure les amplitudes et les déphasages
15 AMP_REC = 0.2 # volume sonore
16 MARGE_REC = 30000
17 DELAI_ACTIVATION_CONTROLE_ACTIF = 1.0 # secondes
18 BUFFER = np.zeros(FFTSIZE) # "tampon" utilisé pour le filtrage dans engine4()
19 s1_rec = np.array([]) # enregistrement du micro de réf
20 s2_rec = np.array([]) # enregistrement du micro d'erreur
21 s3_rec = np.array([]) # enregistrement de la source primaire
22 s4_rec = np.array([]) # enregistrement de la source secondaire
23
24 ## Remarque : les fonctions utilisent les variables globales suivantes en écriture : BUFFER, a, rec, out, Hp, Hs, Hf
25
26 ##
27 def transfer_sine(s1,s2,f,sampFreq):
28     ## calcule la fonction de transfert S2/S1 à la fréquence f
29     A1 = (2**5)*np.mean(s1.astype(np.float)**2)**.5 # amplitude de s1
30     A2 = (2**5)*np.mean(s2.astype(np.float)**2)**.5 # amplitude de s2
31     r = A2/A1
32     xcorr = sgnl.correlate(r*s1, s2) # corrélation croisée de s1 et s2
33     i_max = np.argmax(xcorr)
34
35     ## Interpolation parabolique, détermination de l'abscisse alpha du sommet
36     x1,x2,x3=i_max-1,i_max,i_max+1
37     y1,y2,y3=xcorr[x1],xcorr[x2],xcorr[x3]
38     a = y1/((x1-x2)*(x1-x3)) + y2/((x2-x1)*(x2-x3)) + y3/((x3-x1)*(x3-x2))
39     b = -(x2+x3)*y1/((x1-x2)*(x1-x3)) - (x1+x3)*y2/((x2-x1)*(x2-x3)) - (x1+x2)*y3/((x3-x1)*(x3-x2))
40     alpha = -b/(2*a)
41     ##
42
43     delta_t = alpha-(len(s2)-1)
44     ps = 2*np.pi*f*delta_t/sampFreq # déphasage de s2 par rapport à s1
45     return r,ps
46
47 def test_sd():
48     ## test des périphériques audio
49     t = np.linspace(0,2,num=88200,endpoint=False)

```

```

50     s1d=0*t
51     s1g = AMP_REC*sgn1.tukey(88200,alpha=0.1)*np.cos(2*np.pi*440*t)
52     myrecording = sd.playrec(np.array([s1g,s1d]).transpose(),44100,channels=2, blocking=True)
53     plt.plot(myrecording)
54     plt.show()
55
56
57 def filter_OLA(s, h):
58     ## implémentation de la méthode Overlap-Add afin de filtrer s par le filtre h
59     M = len(h)
60     L = M
61     N=M+L
62     print("N=M+L=",N,"avec M=",M,"et L=",L)
63     H = np.fft.rfft(h,n=N)
64     Nx = len(s)
65     a=0
66     y=np.zeros(Nx+M)
67     while a<Nx-L:
68         b=a+L
69         yt = np.fft.irfft( np.fft.rfft( s[a:b], n=N ) * H, n=N )
70         c=a+N
71         for i in range(N):
72             y[a+i]+=yt[i]
73         a+=L
74     return y
75
76 def engine1(f,duree):
77     ## interface audio, utilisée pour mesurer Hp (dans transferHp())
78     ## émet le bruit primaire (sinusoïde de freq f), pas de bruit secondaire
79     global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec
80
81     L = FFTSIZE # taille d'un bloc
82     BUFFER = np.zeros(2*L)
83     a=0 # a est l'instant de début de chaque bloc (en échantillons)
84
85     def callback(indata, outdata, frames, time, status):
86         global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec
87         if status:
88             print(status)
89
90         s1 = np.array(indata).transpose()[0]
91         s2 = np.array(indata).transpose()[1]
92         s3 = np.zeros(L)
93         s4 = np.zeros(L)
94
95
96         t = np.linspace(a/SMPFREQ,(a+L)/SMPFREQ,num=L,endpoint=False)
97         s3 = AMP_REC*np.cos(2*np.pi*f*t)
98         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     ## interface 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 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_ACTIV*SMFPREQ :
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/SMFPREQ,(a+L)/SMFPREQ,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*SMFPREQ:],s1])
190             s2_rec = np.concatenate([s2_rec[-30*SMFPREQ:],s2])
191             s3_rec = np.concatenate([s3_rec[-30*SMFPREQ:],s3])
192             s4_rec = np.concatenate([s4_rec[-30*SMFPREQ:],s4])
193
194             outdata[:] = np.array([s3,s4]).transpose()
195
196     with sd.Stream(samplerate=SMFPREQ, blocksize=L, latency='low',

```

```

197         channels=2, callback=callback):
198     if mesure:
199         while True:
200             if a>=DELAI_ACTIVATION_CONTROLE_ACTIVIF*SMPFREQ and A1!=None :
201                 A2 = (2**.5)*np.mean(s2_rec[-L:].astype(np.float)**2)**.5
202                 print("Gain instantané :",20*np.log10(A2/A1))
203                 sd.sleep(1000)
204             elif duree==None:
205                 print('#' * 80)
206                 print('Appuyer sur Entrée pour quitter')
207                 print('#' * 80)
208                 input()
209             else:
210                 sd.sleep(int(duree*1000))
211
212 def controleactif_sine(f, duree=None, mesure=False):
213     ## FONCTION PRINCIPALE DU CONTROLE ACTIF DANS LE CADRE D'UN BRUIT SINUSOIDAL PUR
214     global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec,r1,phi1,A1
215     BUFFER = np.zeros(FFTSIZE*2)
216     a=0
217
218     ## mesure de Hp(f)
219     clear_rec()
220     nsamples = MARGE_REC+3*SUB
221     engine1(f,nsamples/SMPFREQ)
222     r_p, phi_p = transfer_sine(s1_rec[MARGE_REC:MARGE_REC+SUB], s2_rec[MARGE_REC:MARGE_REC+SUB], f, SMPFREQ)
223     #print("Hp(f) OK")
224
225     ## mesure de Hs(f)
226     clear_rec()
227     nsamples = MARGE_REC+3*SUB
228     engine2(f,nsamples/SMPFREQ)
229     r_s, phi_s = transfer_sine(s4_rec[MARGE_REC:MARGE_REC+SUB], s2_rec[MARGE_REC:MARGE_REC+SUB], f, SMPFREQ)
230     #print("Hs(f) OK")
231
232     '''## mesure de Hf(f)
233     clear_rec()
234     nsamples = MARGE_REC+3*SUB
235     engine3(f,nsamples/SMPFREQ)
236     r_f, phi_f = transfer_sine(s4_rec[MARGE_REC:MARGE_REC+SUB], s1_rec[MARGE_REC:MARGE_REC+SUB], f, SMPFREQ)
237     #print("Hf(f) OK)'''
238     # remarque : on ne mesure pas Hf ici : le retour du bruit secondaire dans le micro de réf n'est pas pris en compte
239
240     ## calcul du gain et du déphasage (par rapport au signal de réf) à appliquer au contre-bruit
241     r_f,phi_f=0,0
242     z = r_f*np.exp(phi_f*1j) - (r_s/r_p)*np.exp((phi_s-phi_p)*1j)
243     R = 1/np.abs(z)
244     PHI = - np.angle(z)
245

```



```

246     ## c'est parti
247     L=FFTSIZE # taille d'un bloc
248     BUFFER = np.zeros(2*L)
249     a=0
250     clear_rec()
251     r1,phi1,A1=None,None,None
252
253     def callback(indata, outdata, frames, time, status):
254         global BUFFER,a,s1_rec,s2_rec,s3_rec,s4_rec,r1,phi1,A1
255         if status:
256             print(status)
257
258         s1 = np.array(indata).transpose()[0]
259         s2 = np.array(indata).transpose()[1]
260         s3 = np.zeros(L)
261         s4 = np.zeros(L)
262
263
264         t = np.linspace(a/SMPFREQ,(a+L)/SMPFREQ,num=L,endpoint=False)
265         s3 = AMP_REC*np.cos(2*np.pi*f*t)
266         if a>=DELAI_ACTIVATION_CONTROLE_ACTIV*SMPFREQ :
267             if A1==None:A1 = (2**.5)*np.mean(s2.astype(np.float)**2)**.5 # amplitude avant contrôle actif
268             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
269             s4 = R*r1*np.cos(2*np.pi*f*t + phi1+PHI) # S4 = W*S1
270         a+=L
271
272         s1_rec = np.concatenate([s1_rec[-30*SMPFREQ:],s1])
273         s2_rec = np.concatenate([s2_rec[-30*SMPFREQ:],s2])
274         s3_rec = np.concatenate([s3_rec[-30*SMPFREQ:],s3])
275         s4_rec = np.concatenate([s4_rec[-30*SMPFREQ:],s4])
276
277
278         outdata[:] = np.array([s3,s4]).transpose()
279
280     with sd.Stream(samplerate=SMPFREQ, blocksize=L, latency='low',
281                  channels=2, callback=callback):
282         if mesure:
283             while True:
284                 if a>=DELAI_ACTIVATION_CONTROLE_ACTIV*SMPFREQ and A1!=None :
285                     A2 = (2**.5)*np.mean(s2_rec[-L:].astype(np.float)**2)**.5
286                     print("Gain instantané :",20*np.log10(A2/A1))
287                     sd.sleep(1000)
288             elif duree==None:
289                 print('#' * 80)
290                 print('Appuyer sur Entrée pour quitter')
291                 print('#' * 80)
292                 input()
293         else:
294             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])

```

```

1  ...
2  Simulation sources ponctuelles primaires et secondaires
3  calcule gain pour angles theta à r=20cm pour expérience
4  ...
5  import matplotlib.pyplot as plt
6  import numpy as np
7  import matplotlib.animation as animation
8
9
10 cst_c = 340.0
11 cst_minfloat = 10**(-48)
12 cst_debit = 1
13 cst_masvol = 1.225
14 cst_p0 = 20*10**(-6)
15
16 def clamp(x,m,M):
17     if x<m: return m
18     if x>M: return M
19     return x
20
21 class Source:
22     def __init__(self,x,y,pulsation,phase,debit):
23         self.__x = x
24         self.__y = y
25         self.__pulsation = pulsation
26         self.__phase = phase
27         self.__debit = debit
28     def dist(self,x,y):
29         return ((self.__x-x)**2+(self.__y-y)**2)**.5
30     def champ(self,x,y,t=0):
31         r = self.dist(x,y)
32         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)
33
34
35 class Simulation:
36     def __init__(self, nom=""):
37         self.__sources_primaires = []
38         self.__sources_secondaires = []
39
40         self.nom = nom
41
42     def ajouterSource(self,x,y,pulsation,phase=0,debit=cst_debit,primaire=False):
43         S = Source(x,y,pulsation,phase,debit)
44         if primaire :
45             self.__sources_primaires.append(S)
46         else :
47             self.__sources_secondaires.append(S)
48
49     def transfert(self,x,y,t=0):

```

```

50     A = sum([s.champ(x,y,t) for s in self.__sources_principales])
51     if A==0: A=cst_minfloat
52     B = A+sum([s.champ(x,y,t) for s in self.__sources_secondaires])
53     return B/A
54
55     def champ(self,x,y,t=0):
56         return sum([s.champ(x,y,t) for s in self.__sources_principales])+sum([s.champ(x,y,t) for s in self.__sources_secondaires])
57
58     def graph(self,xmin=-3,xmax=3,ymin=-3,ymax=3,pas=0.05,titre=""):
59         if titre=="": titre = self.nom
60         x = np.arange(xmin,xmax,pas)
61         y = np.arange(ymin,ymax,pas)
62         X,Y = np.meshgrid(x,y)
63         #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))])
64         #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))])
65         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))])
66         im = plt.pcolormesh(X,Y,Z, cmap="coolwarm", vmin=-48, vmax=48)
67         plt.xlabel("x (m)")
68         plt.ylabel("y (m)")
69         plt.colorbar(im, label="Gain (dB)")
70         if titre!=None : plt.title(titre)
71         plt.axis('equal')
72         #plt.show()
73
74
75     if __name__ == '__main__':
76         simulation = Simulation("Expérience 27/11/2018 essai 2 : d=2.0cm f=40kHz")
77         d = 0.02
78         simulation.ajouterSource(0,-d/2,2*np.pi*40000,phase=0,primaire=True)
79         simulation.ajouterSource(0,d/2,2*np.pi*40000,phase=0)
80         simulation.graph(pas=0.003,xmin=-0.5,xmax=0.5,ymin=-0.5,ymax=0.5)
81         plt.axhline(0, color='black',lw=1)
82         plt.axvline(0, color='black',lw=1)
83         plt.scatter([0,0],[-d/2,d/2],c="red")
84         for theta in range(46):
85             x = 0.20*np.cos(theta*np.pi/180)
86             y = 0.20*np.sin(theta*np.pi/180)
87             plt.scatter(x,y,c="white",alpha=0.5,edgecolors="black")
88             print("angle",theta," :",20*np.log10(abs(simulation.transfert(x,y))))
89         plt.grid()
90         plt.show()
91
92

```