Atténuation active du bruit urbain

GODEY Louis - Numéro de candidat : 20218

Membre du groupe : TRIBOUT Rémy - Numéro de candidat : 20004

TIPE 2023

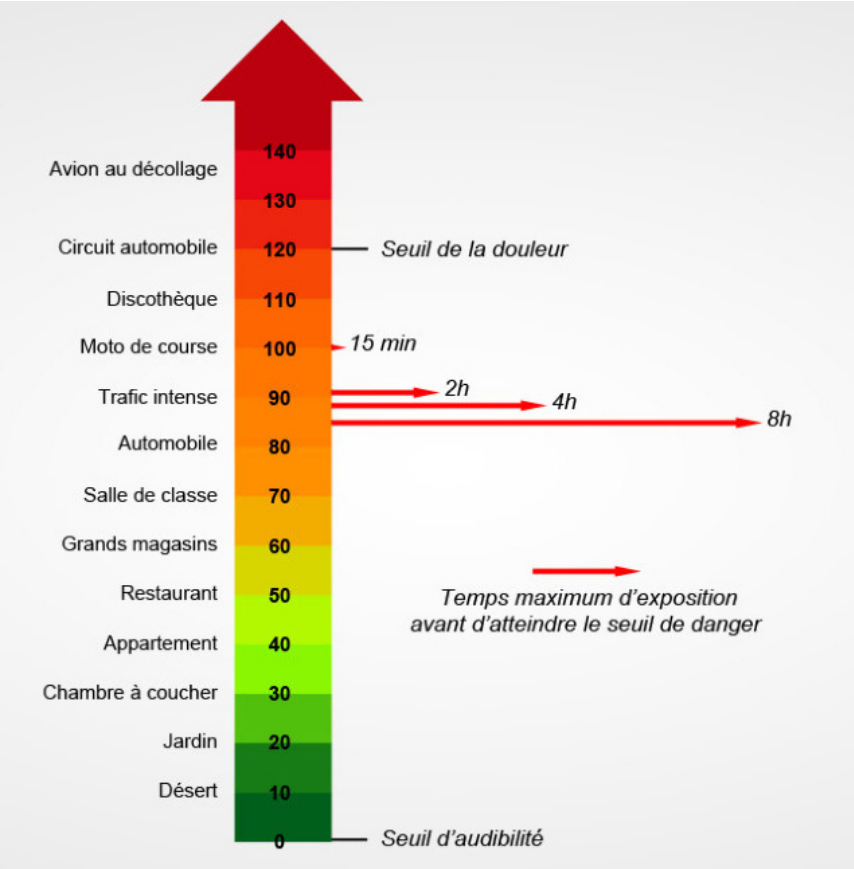
Thème : La ville

Table des matières

Comment identifier les paramètres optimaux d'un système, numérique ou analogique, de réduction active du bruit ?

Problématique du bruit urbain

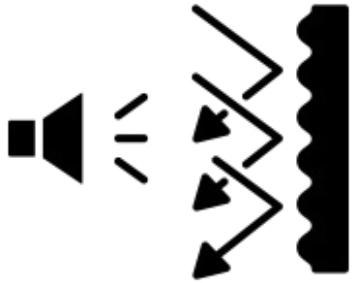
Échelle du bruit



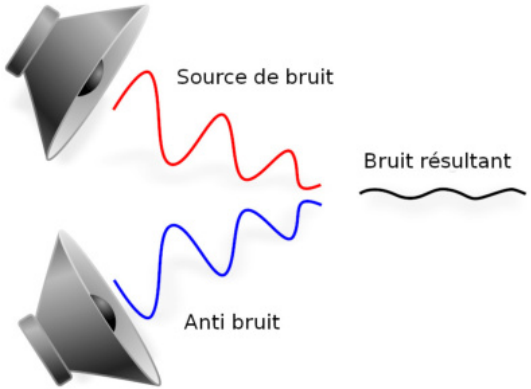
source : Haut Conseil de la Santé publique

Nécessité d'atténuer le bruit urbain :

- Par méthode passive

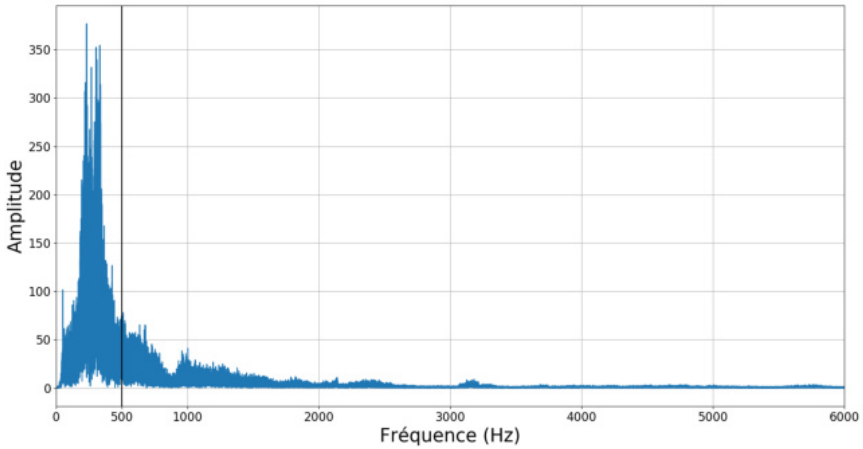


- Par méthode active

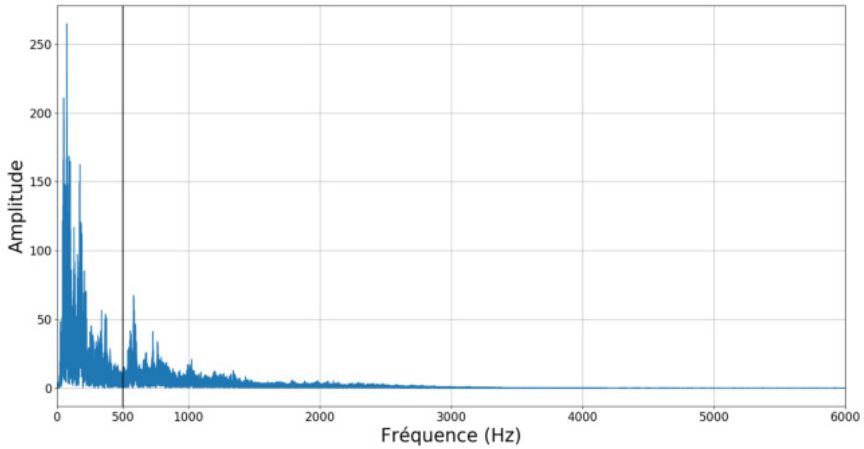


Fréquences caractéristiques du bruit urbain

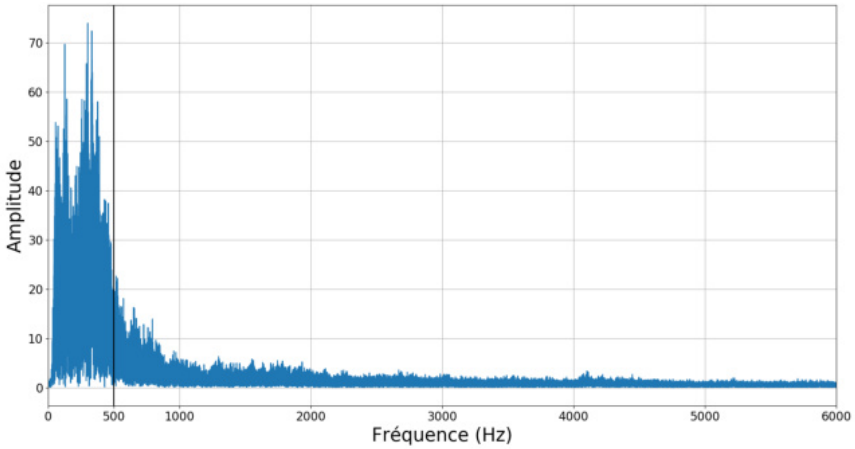
Passage d'un tramway



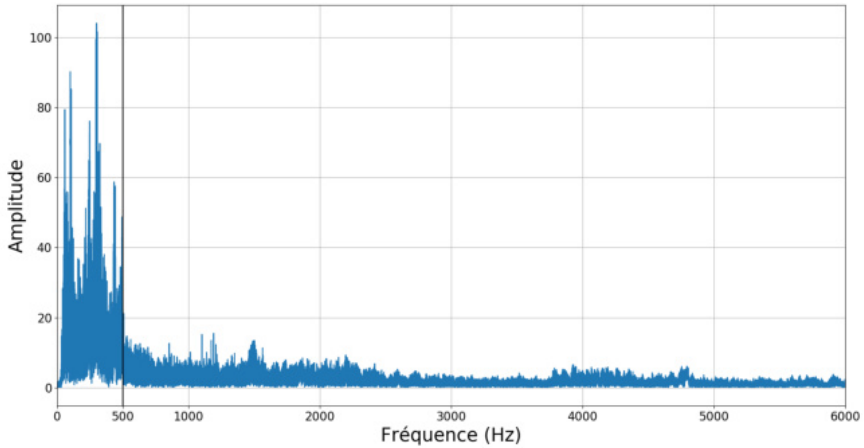
Passage d'un bus



Rue passante

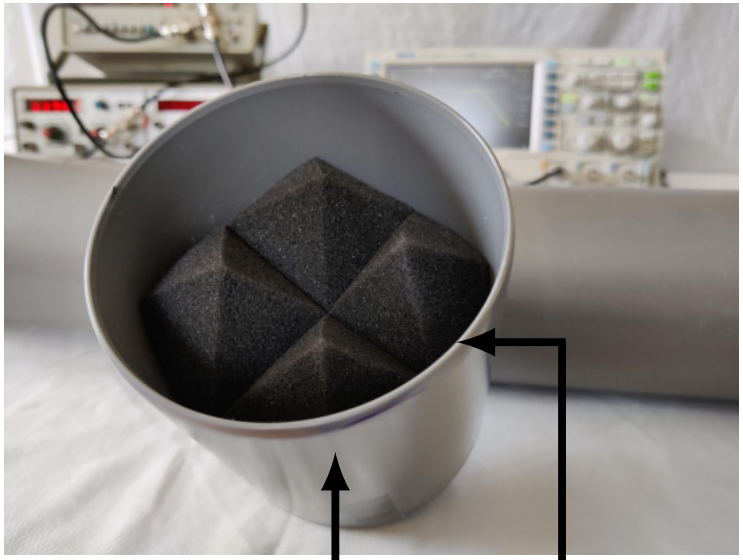


Marché



Fréquences inférieures à 500 Hz

Étude d'un dispositif passif



Caractéristiques de la mousse :

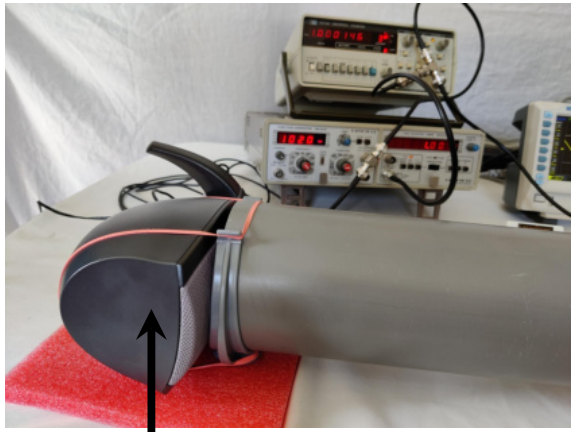
Matériau : Polyuréthane flexible à cellules ouvertes

Épaisseur : 5,0 cm

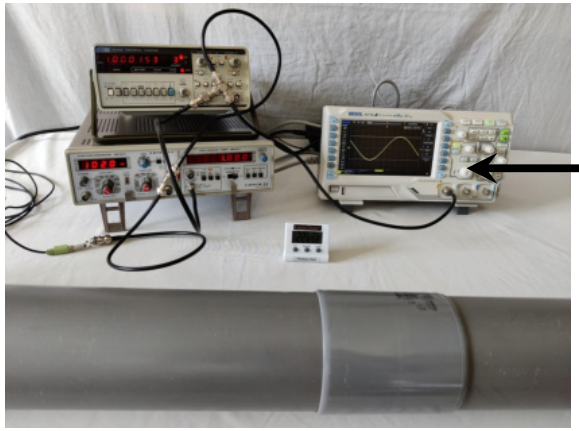
Densité : 25 $kg \cdot m^{-3}$

Norme NF EN ISO 10534-2

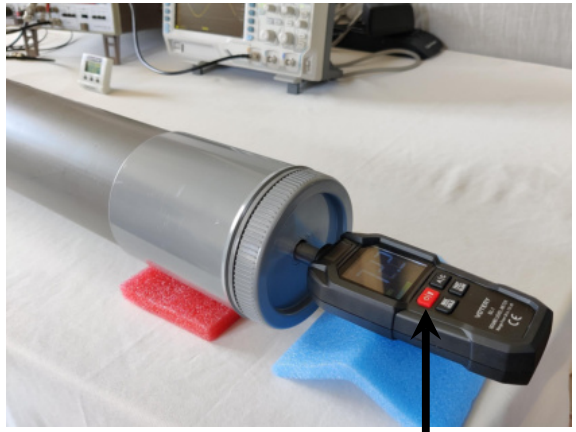
Pr Jean-François Fontaine
Université de Bourgogne



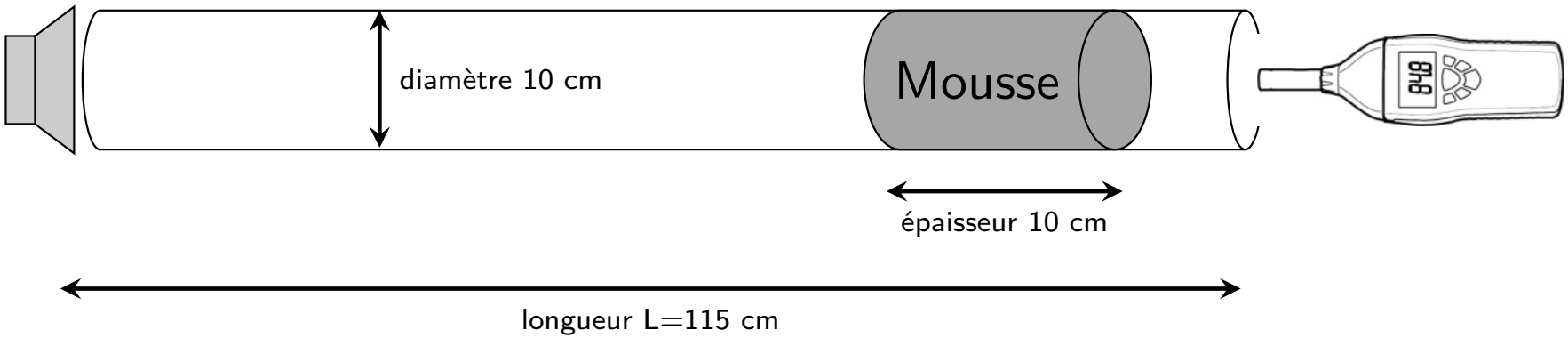
Source sonore
Signal sinusoïdal



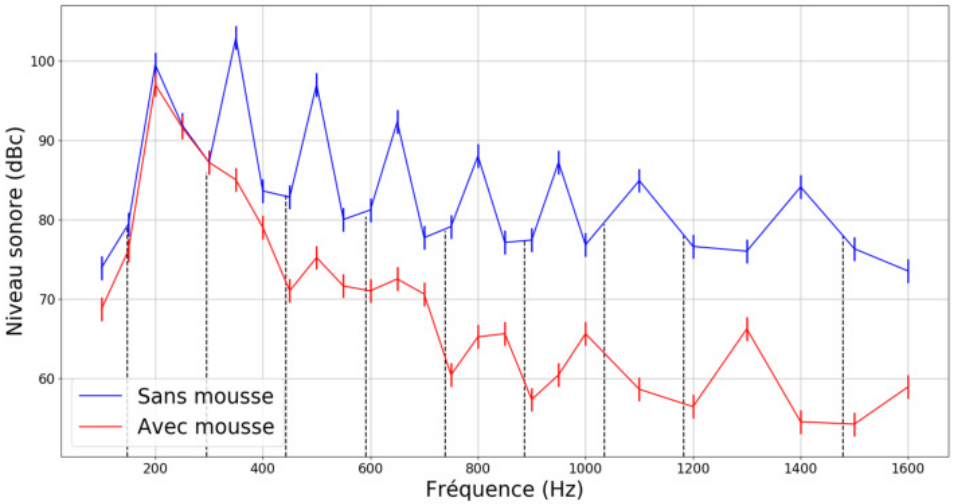
Oscilloscope

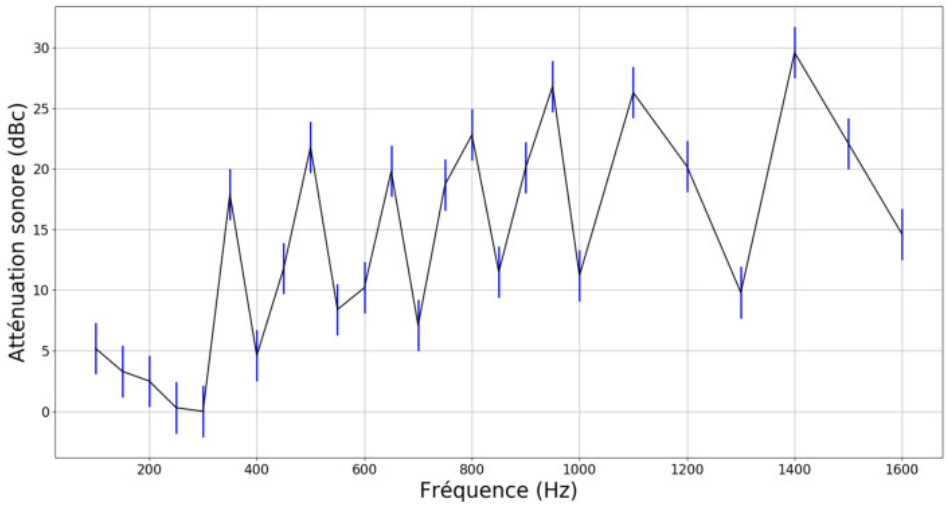


Sonomètre
Plage de mesure : 35 à 130 dBc
Précision : ± 1,5 dBc à 1 kHz



Courbes d'atténuation expérimentales





Pour cette cavité résonnante, les fréquences pouvant exister sont :

$$f_n = \frac{nc}{2L}, n \in \mathbb{N}^*$$

Pour $c = 340\ \mathrm{m} \cdot \mathrm{s}^{-1}$:

$$f_1 = 148\ \mathrm{Hz}$$

$$f_2 = 296\ \mathrm{Hz}$$

...

$$f_{10} = 1,48\ \mathrm{kHz}$$

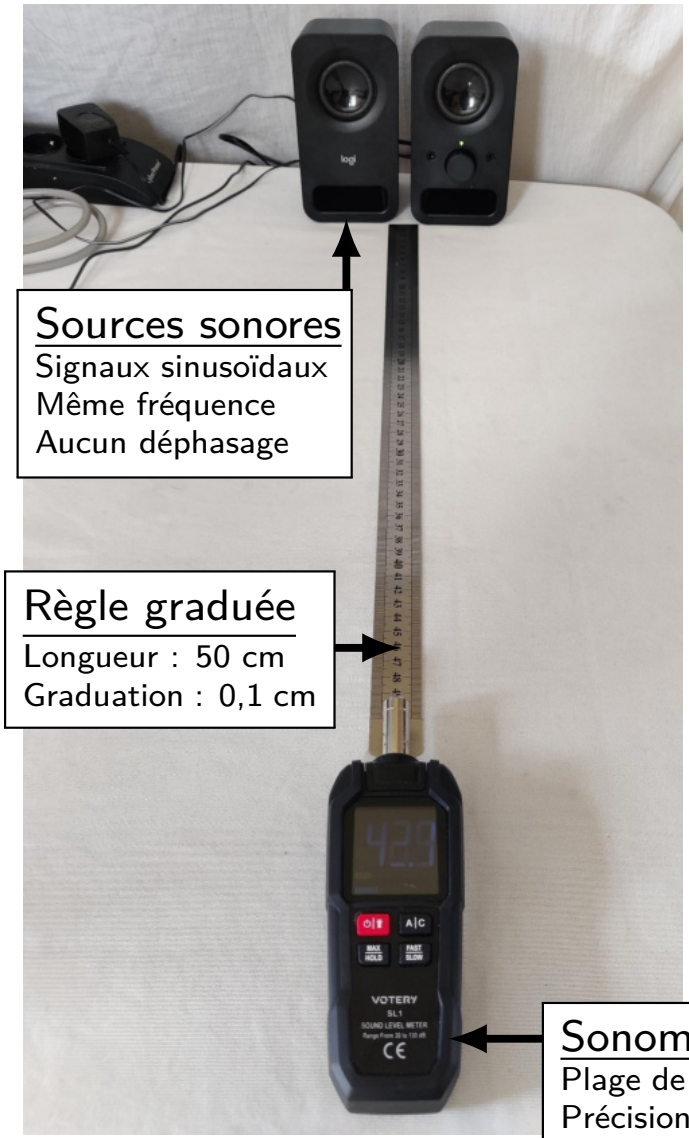
Courbe d'atténuation fournie par le constructeur

Conclusion de l'expérience

Mousse acoustique efficace à hautes fréquences ($\geqslant 1,5$ kHz).

Non adapté au bruit urbain ($\leqslant 500$ Hz).

Mise en évidence du phénomène d'interférences



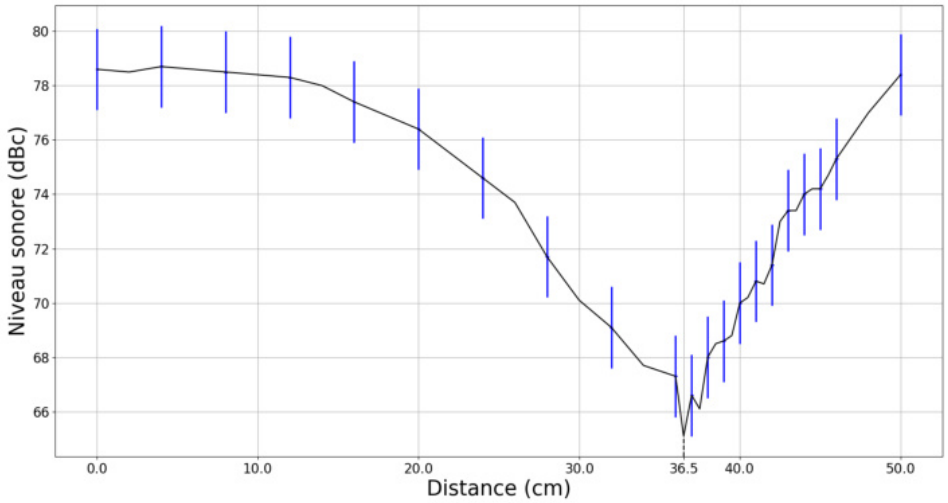
Approximation unidimensionnelle : minimum de niveau sonore pour $L = \lambda/2$.

$$\lambda = \frac{c}{f}$$

Pour $f = 440$ Hz : $\dfrac{\lambda}{2} = 38,5$ cm

Pour $f = 1,00$ kHz : $\dfrac{\lambda}{2} = 17,0$ cm

Niveau sonore en fonction de la distance (440 Hz)

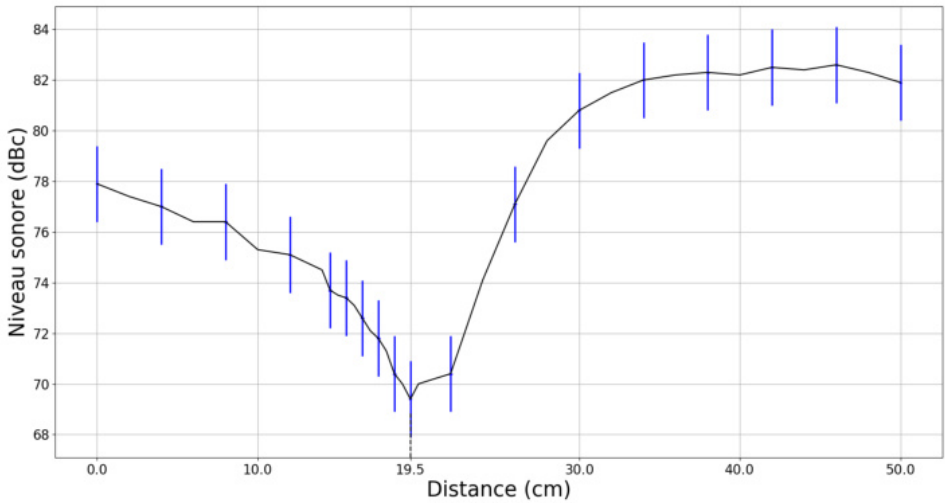


Niveau sonore minimum atteint pour :

$$L = 37 \text{ cm} \pm 1 \text{ cm}$$

Z-score : $Z = 1,5$

Niveau sonore en fonction de la distance (1,0 kHz)

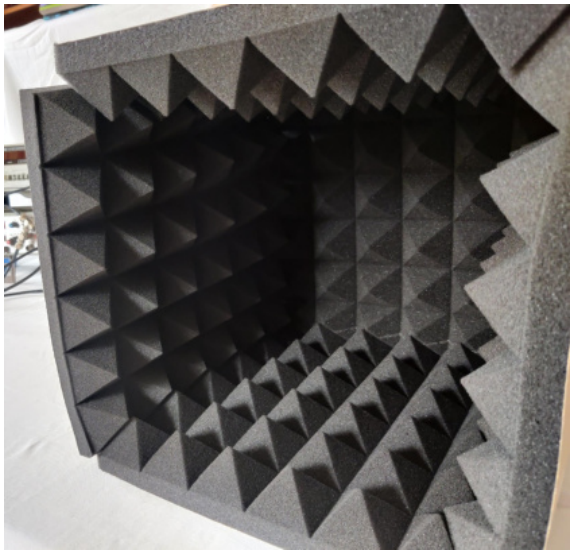


$$L = 20 \text{ cm} \pm 1 \text{ cm}$$

Z-score : $Z = 3,0$

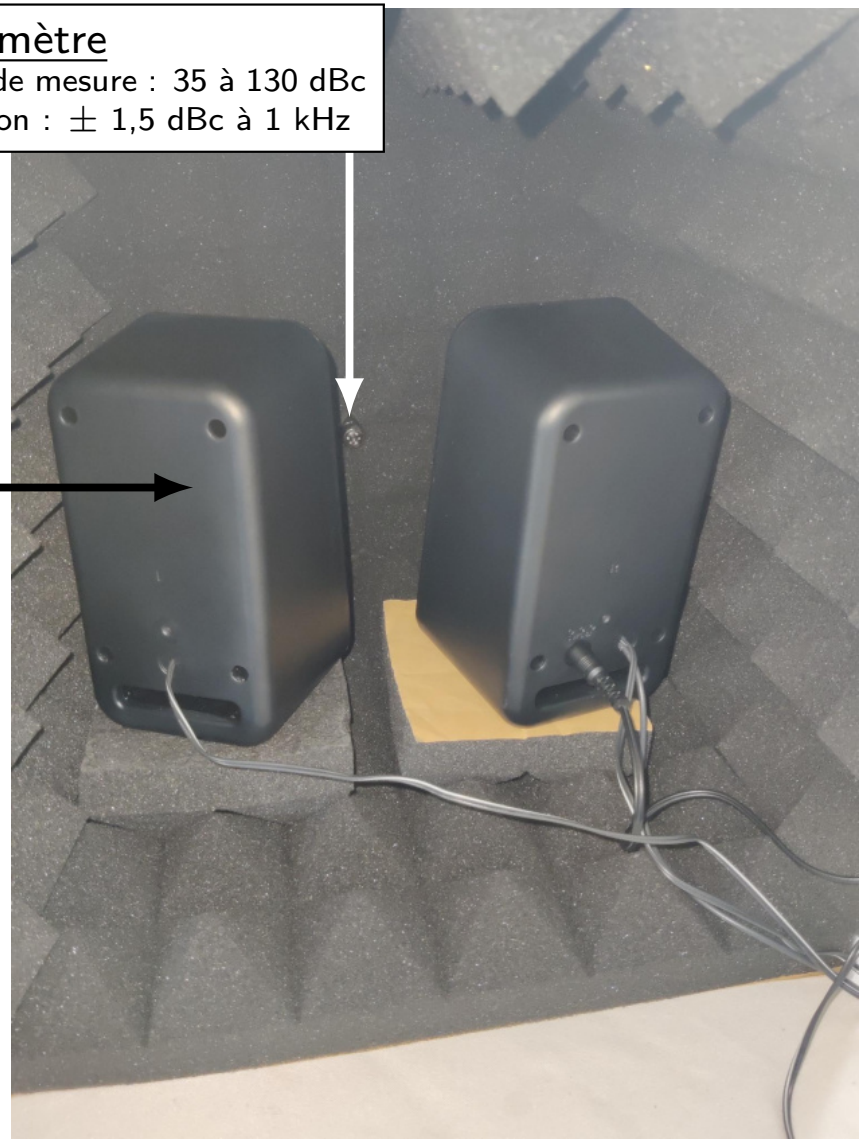
Étude d'un dispositif actif

Chambre anéchoïque

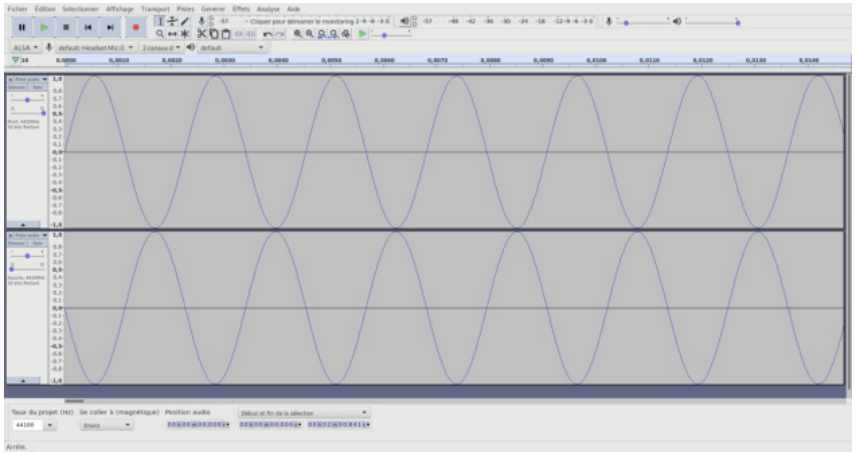


Sonomètre
Plage de mesure : 35 à 130 dBc
Précision : ± 1,5 dBc à 1 kHz

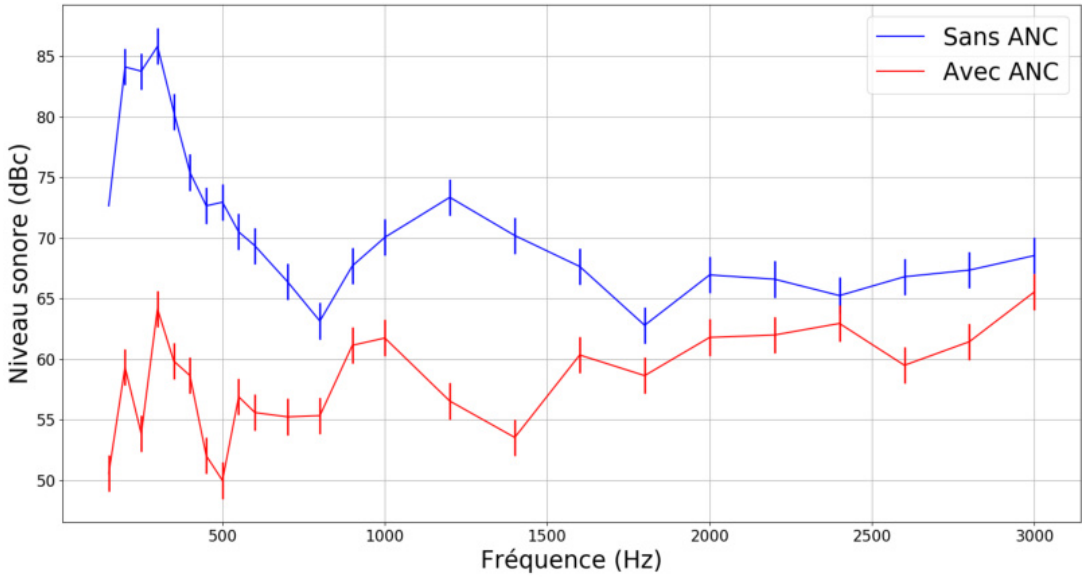
Sources sonores
Signaux sinusoïdaux
Même fréquence
Opposition de phase

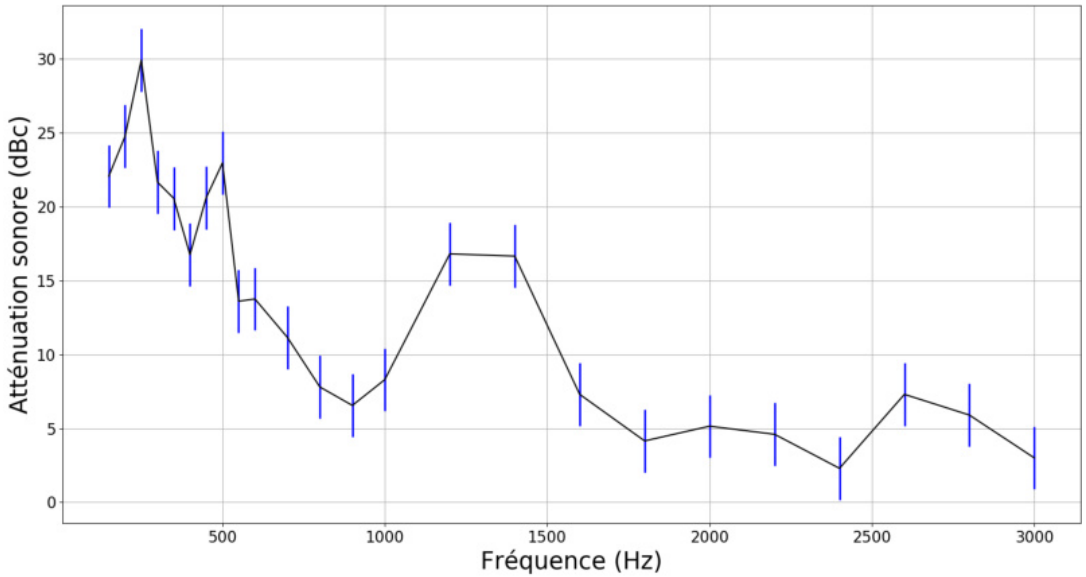


Contrôle du signal par Audacity



Courbes d'atténuation expérimentales



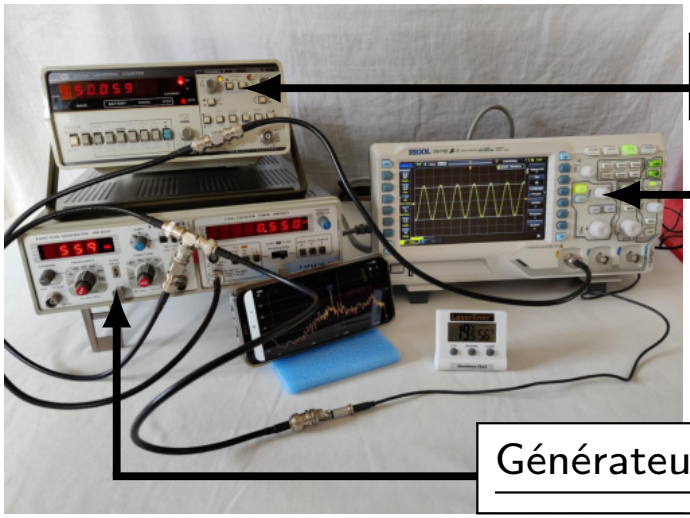


Conclusion de l'expérience

Meilleure efficacité à basses fréquences ($\leqslant$ 700 Hz).

Système plus adapté en milieu urbain ($\leqslant$ 500 Hz).

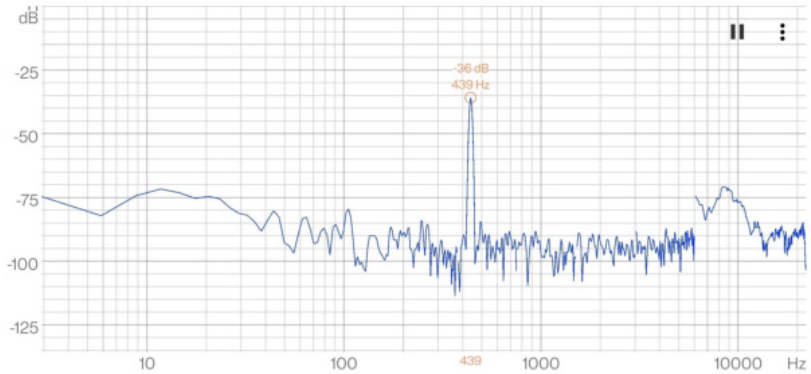
Étude d'un casque du commerce



Fréquencemètre

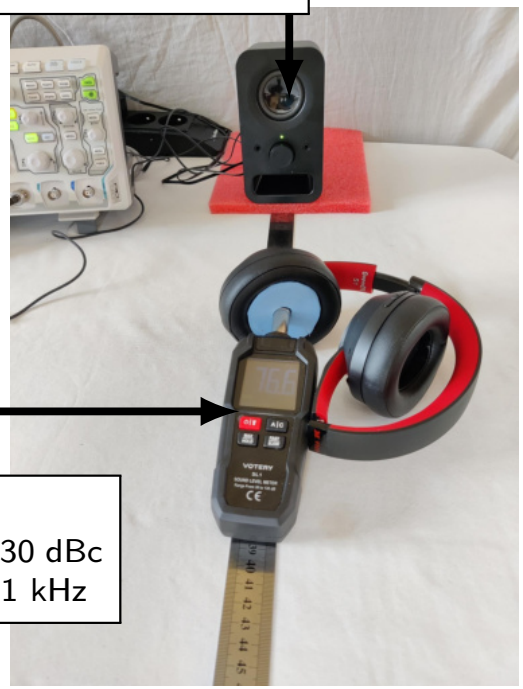
Oscilloscope

Générateur de Basses Fréquences (G.B.F.)



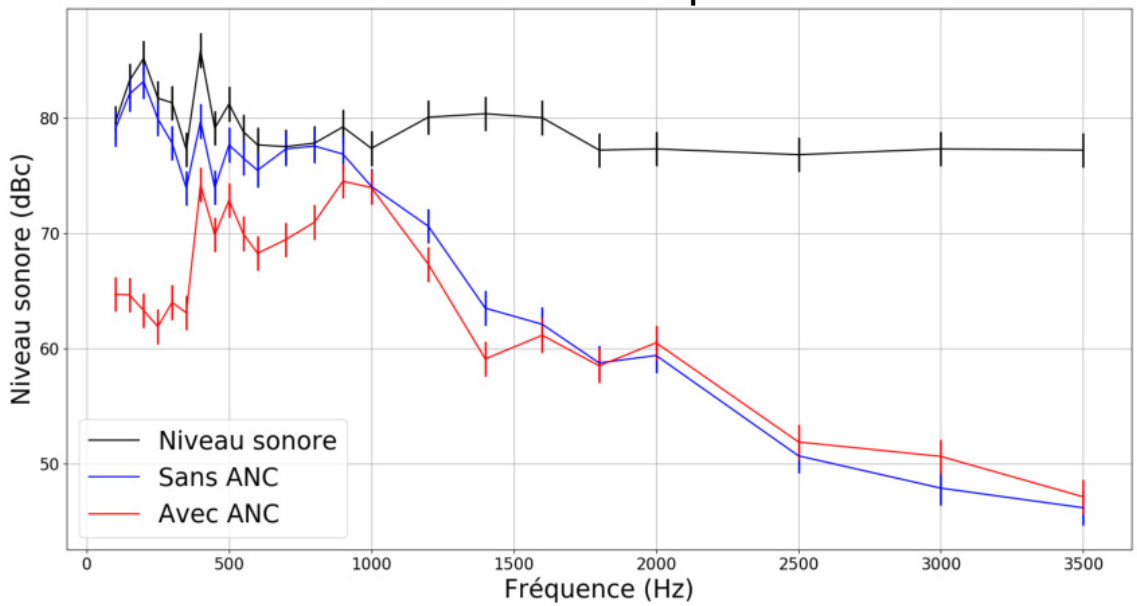
Source sonore
Signal sinusoïdal

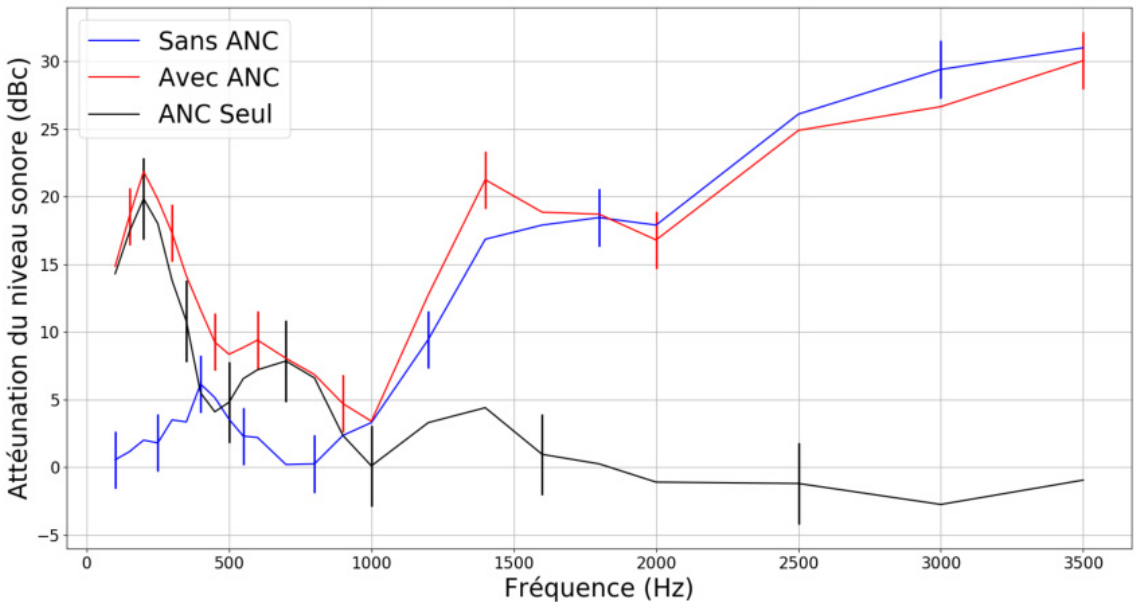
Oreille de substitution
Matériau : polyéthylène (Plastazote)
Permet de fermer la cavité du casque



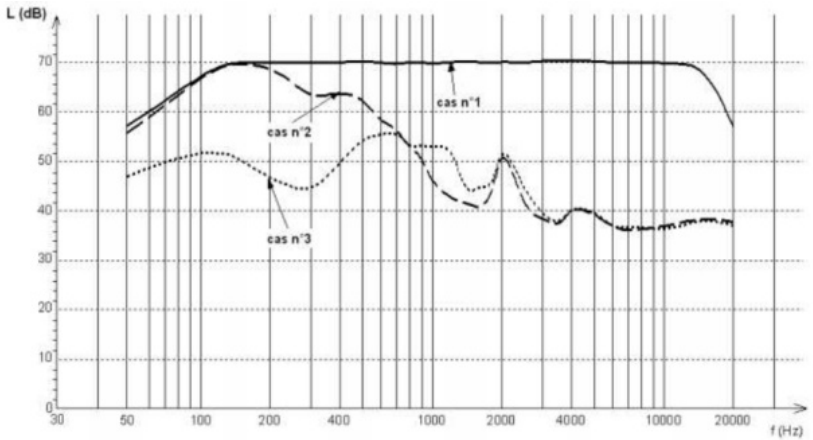
Sonomètre
Plage de mesure : 35 à 130 dBc
Précision : $\pm$ 1,5 dBc à 1 kHz

Courbes d'atténuation expérimentales



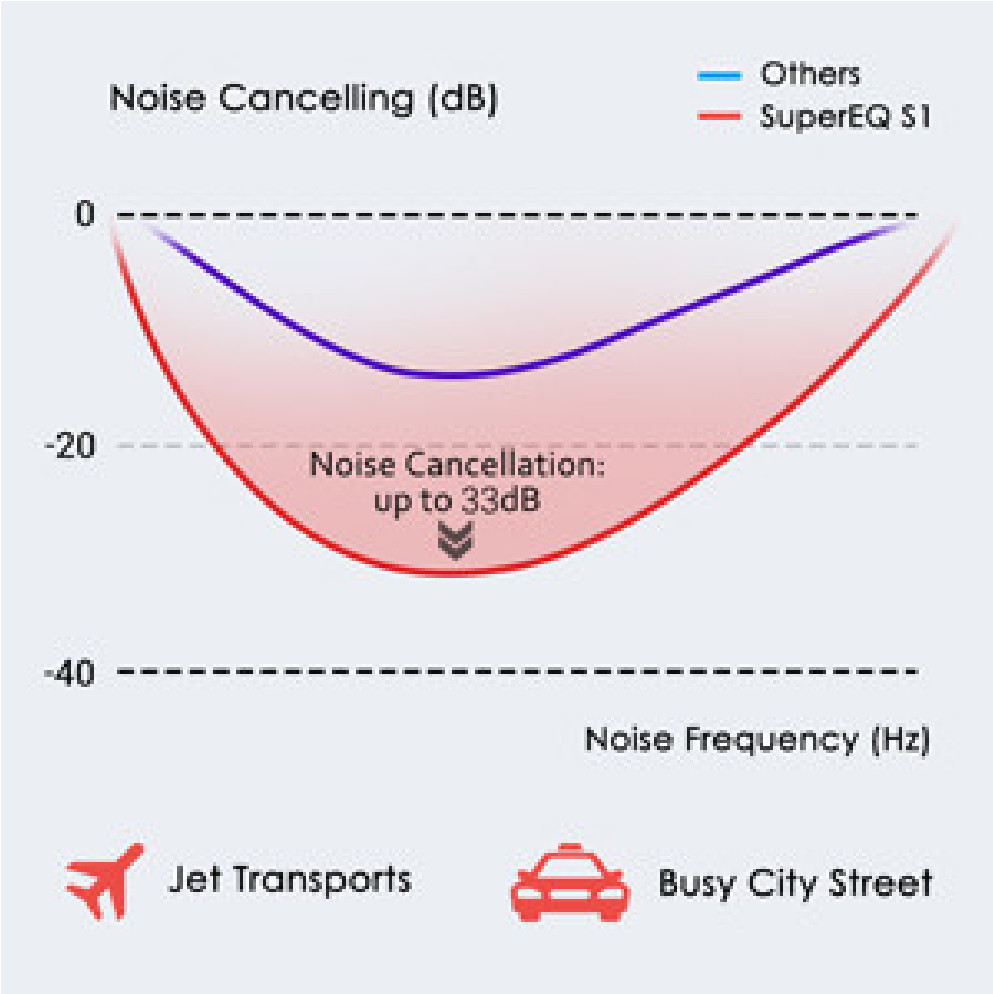


cas 1 : niveau sonore ambiant
cas 2 : Sans ANC
cas 3 : Avec ANC



source : BAC S métropole septembre 2014

Courbe commerciale de performance



Atténuation moyenne (0 - 3,5 kHz)

$15,1 \text{ dBc} \pm 2,1 \text{ dBc}$

Atténuation maximale

$30,1 \text{ dBc} \pm 2,1 \text{ dBc}$

Z-score : $Z = 1,4$

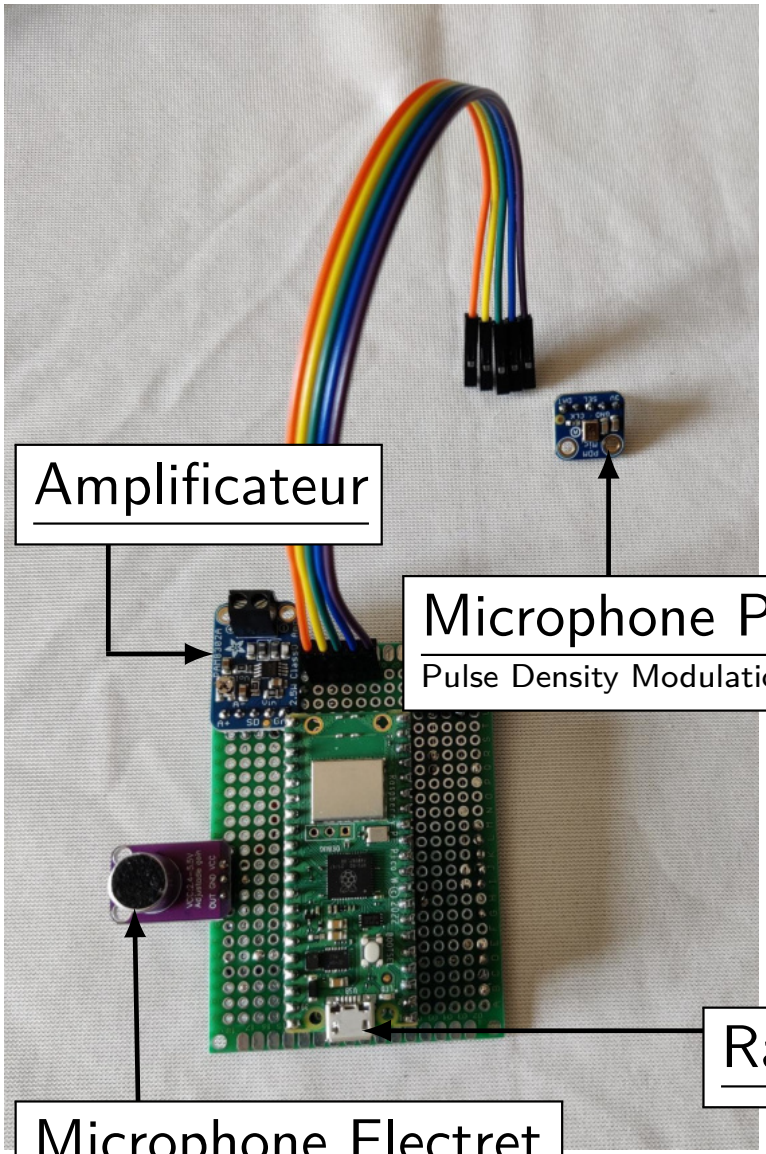
Conclusion de l'expérience

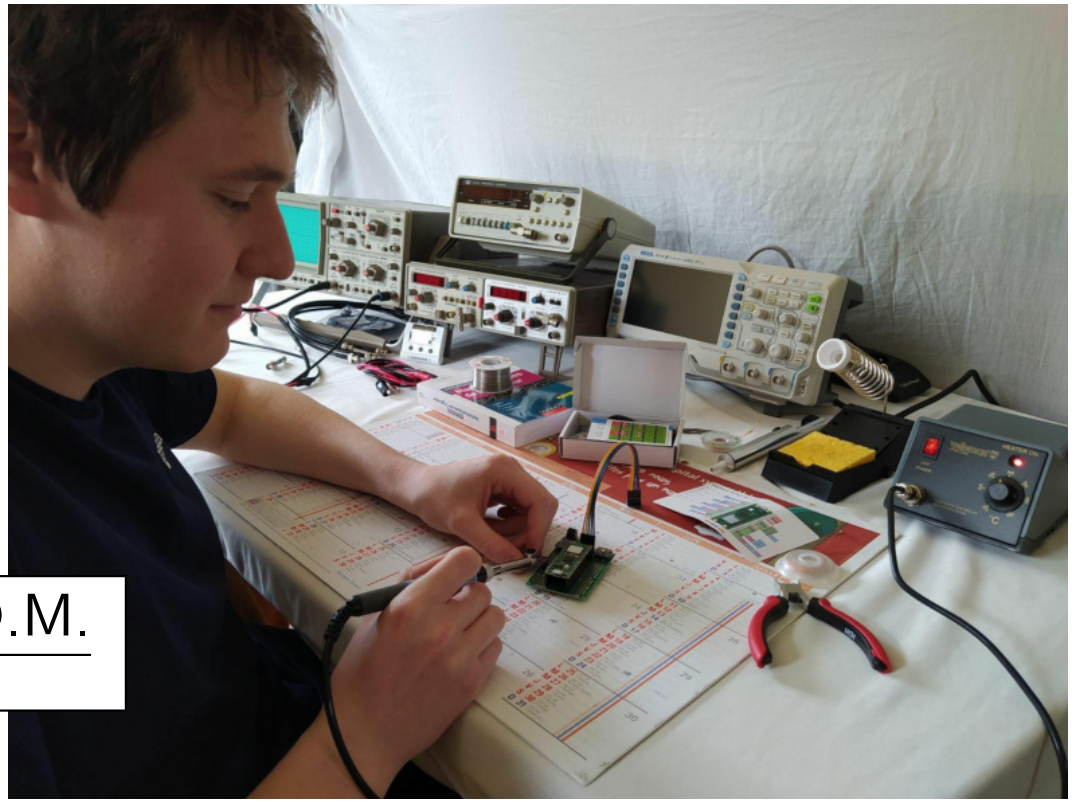
Complémentarité des solutions passives et actives.

Atténuation moyenne (0 - 500 Hz)

$15,3 \text{ dBc} \pm 2,1 \text{ dBc}$

Montage réalisé





Identification de la fréquence dominante

Implémentation sur carte programmable Raspberry Pi Pico

Fréquence d'échantillonnage
$f_e = 32$ kHz

signal acoustique

Microphone

signal électrique

Raspberry Pi Pico

signal échantillonné

Transformée de Fourier

Respect du critère de Nyquist-Shannon

Implémentation numérique d'un filtre passe-bas

Équation différentielle d'un filtre passe-bas d'ordre 1 :

$$\frac{ds}{dt} + \frac{s}{\tau} = \frac{e}{\tau} \text{ avec } \tau = \frac{1}{2\pi f_c}$$

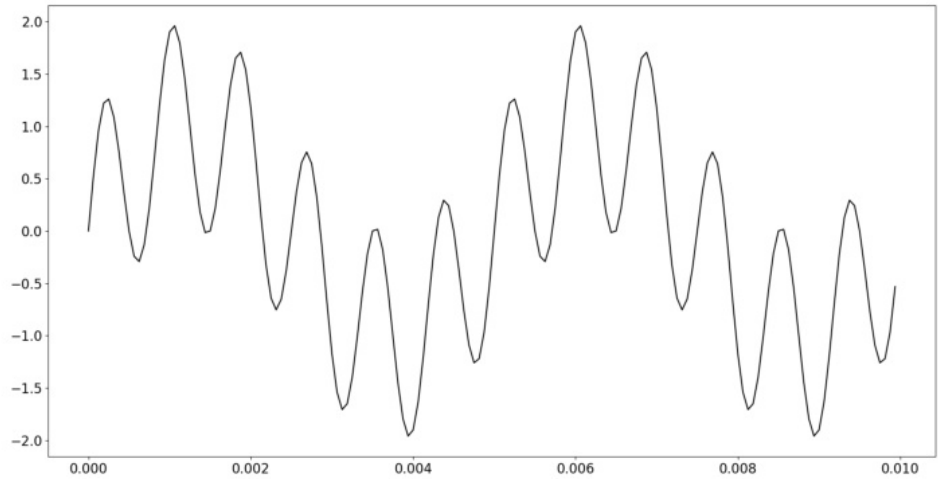
On discrétise avec $T_e = \frac{1}{32000}$ la période d'échantillonnage

$$\frac{s_{k+1} - s_k}{T_e} + \frac{s_k}{\tau} = \frac{e_k}{\tau}$$

D'où

$$\boxed{\forall k \in \mathbb{N}^* \ , \ s_{k+1} = s_k + \frac{T_e}{\tau}(e_k - s_k)}$$

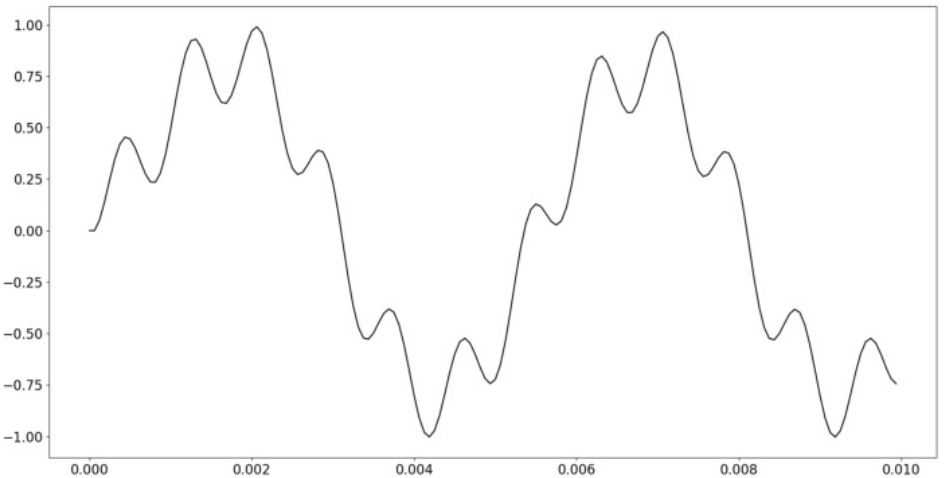
Sans filtrage



$$t \longmapsto \sin(2\pi 200t) + \sin(2\pi 1200t)$$

Fréquence de coupure :
$f_c = 250$ Hz

Avec filtrage

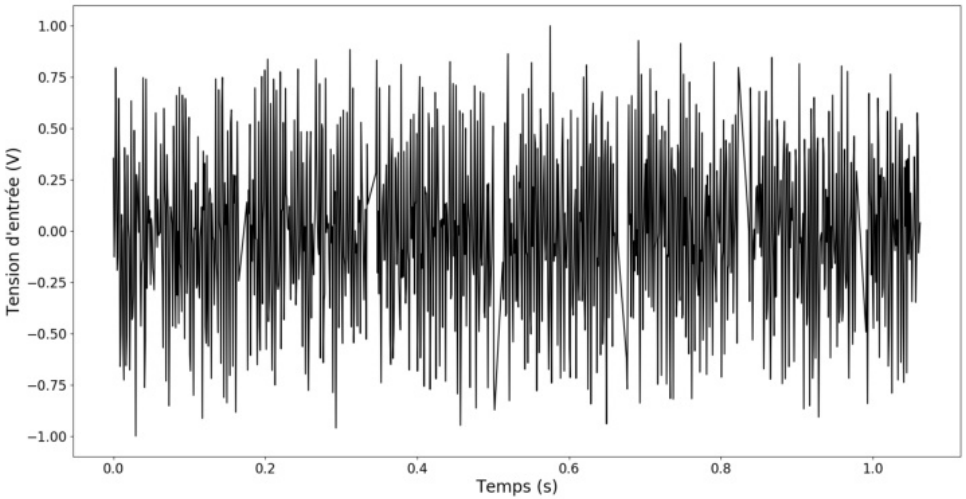


Respect du critère de Nyquist-Shannon :
$f_e > 2f_{max}$

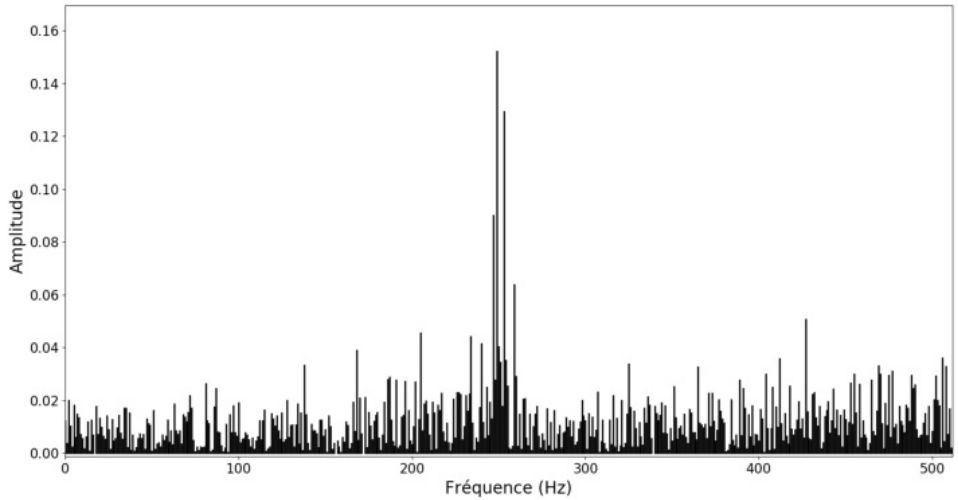
avec $f_c = 5$ kHz

Exemple d'échantillonnage

Signal électrique reçu par la carte



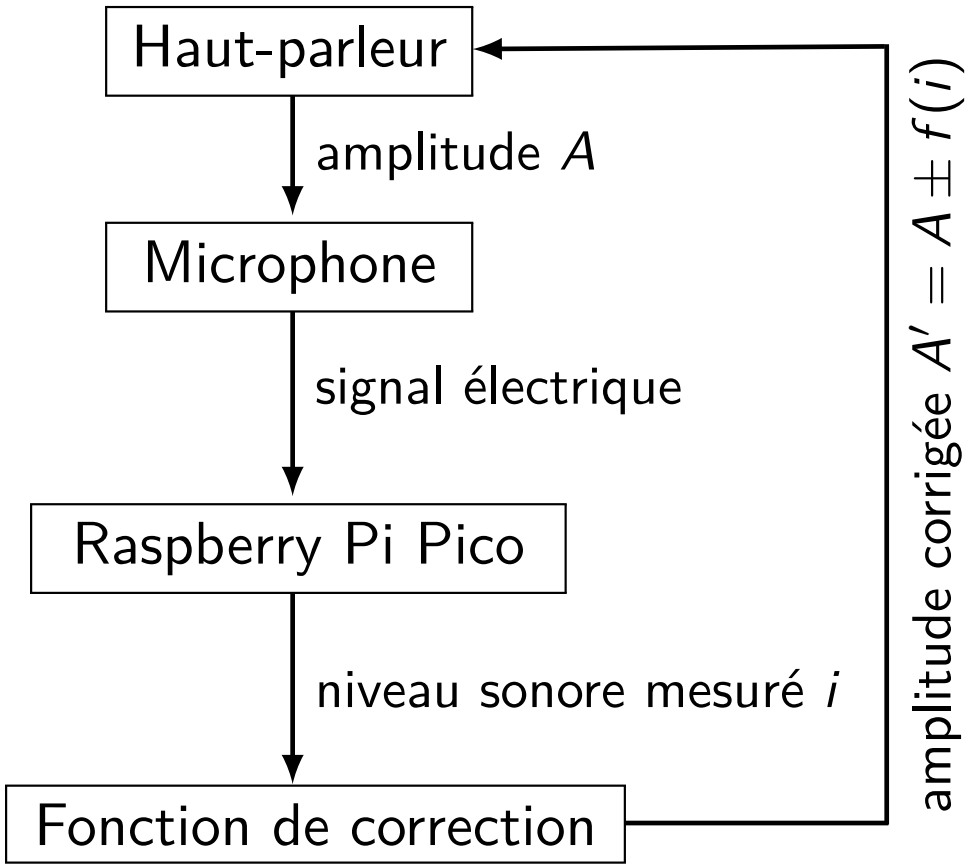
Spectre obtenu par F.F.T.



Fréquence perçue : 250 Hz

Rétroaction sur l'amplitude

Rétroaction sur l'amplitude

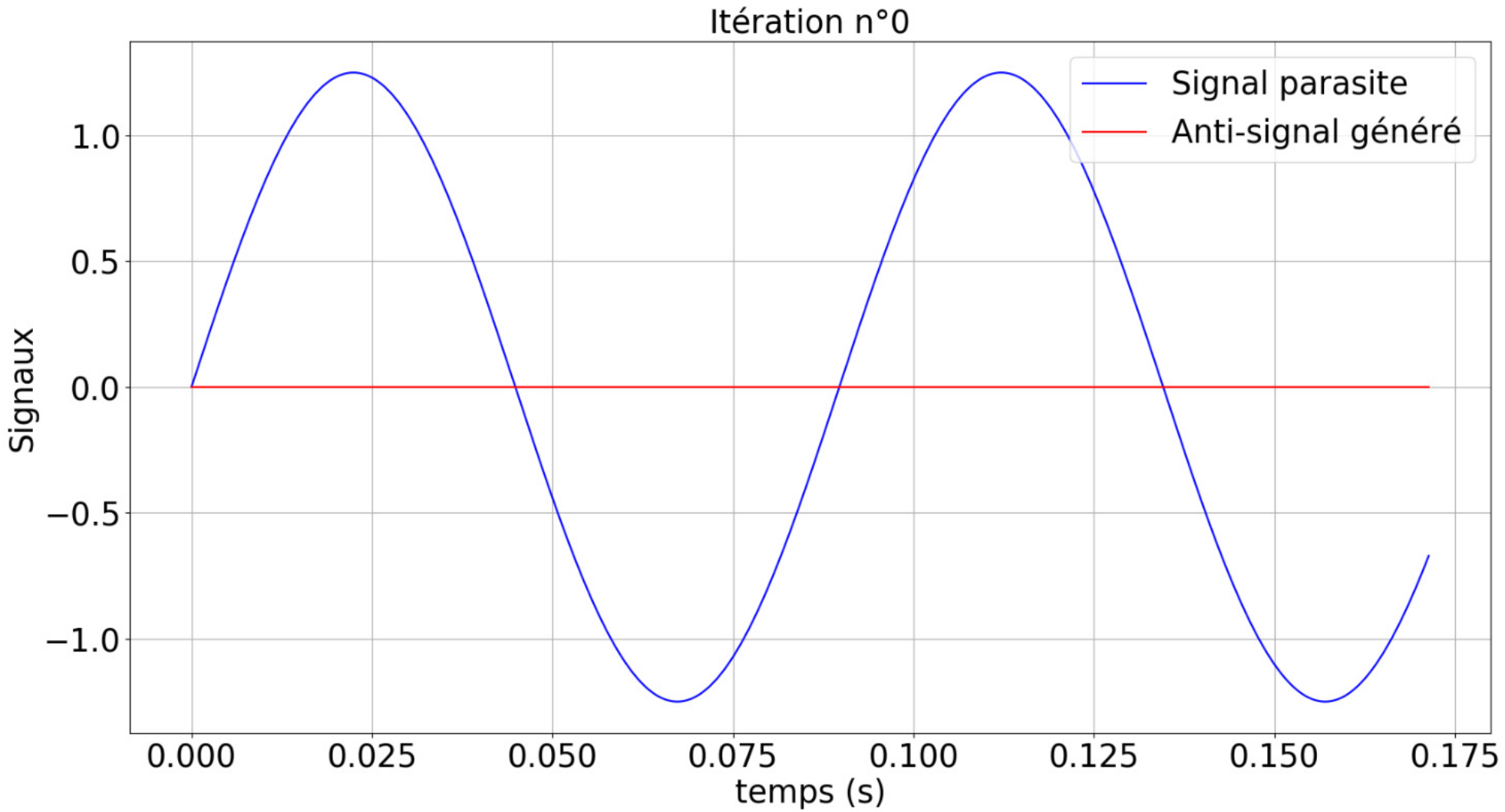


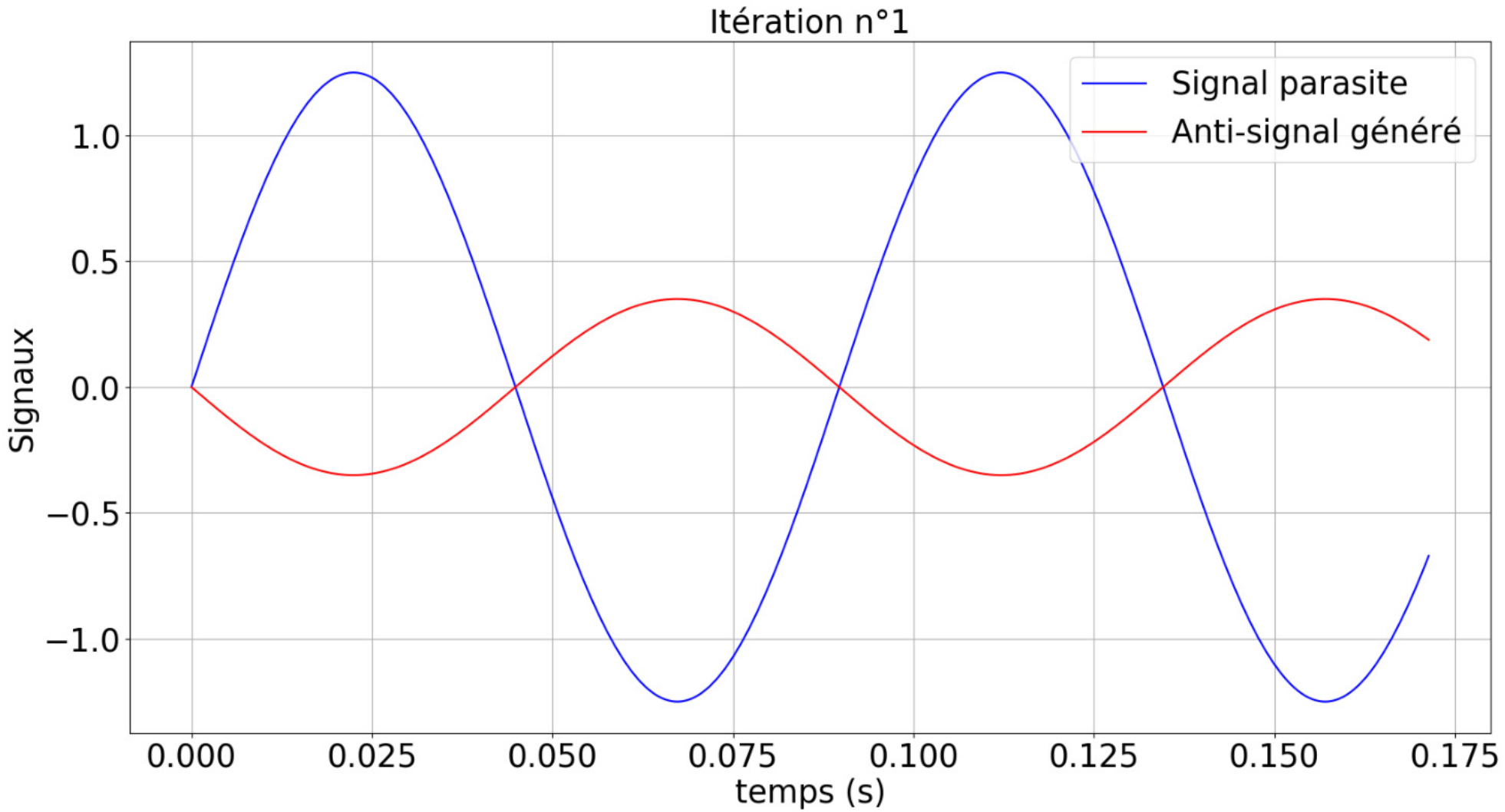
Fonction de correction

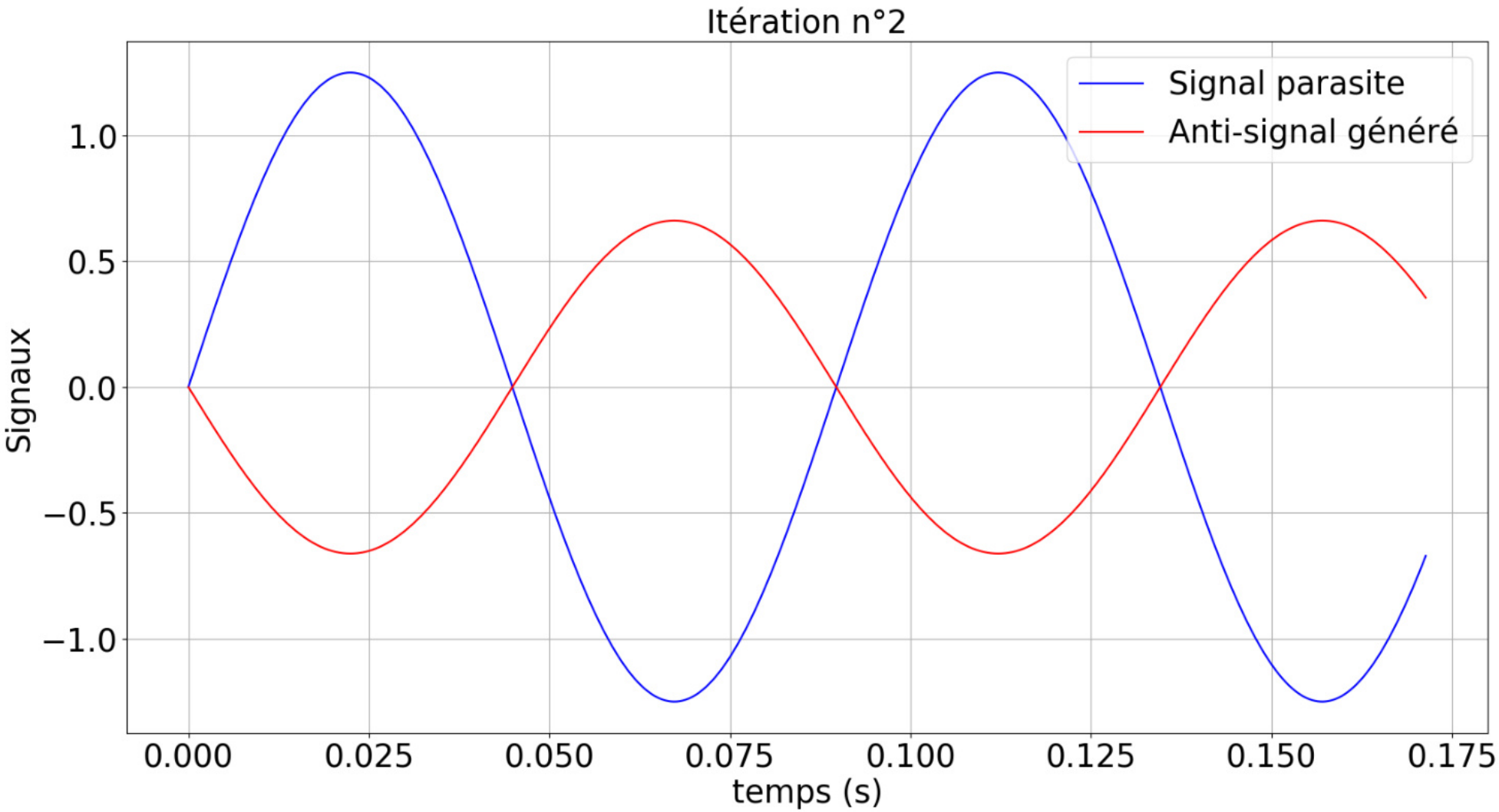
i_0 : niveau sonore du bruit parasite

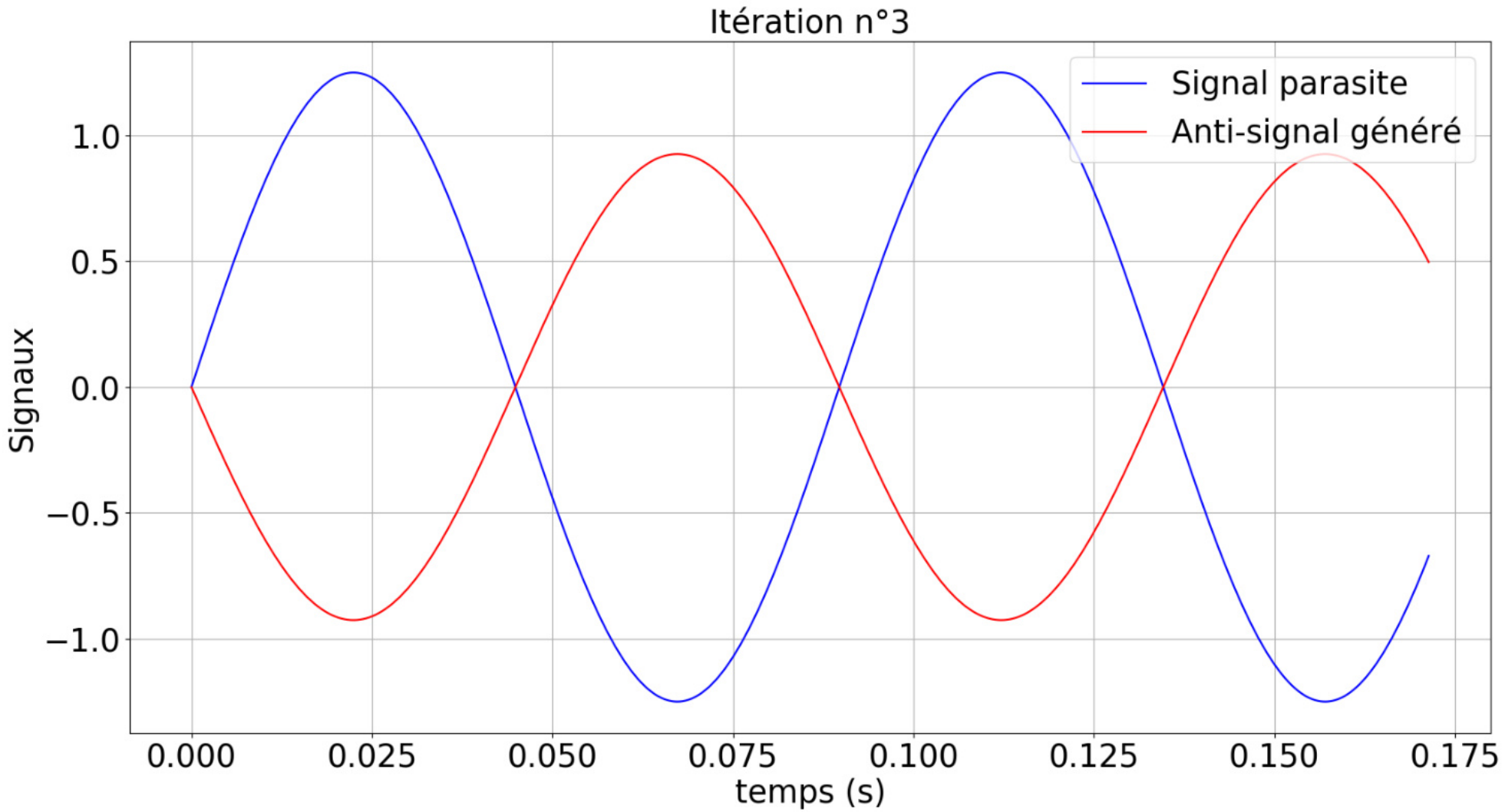
$c = i_0 - 30$: consigne

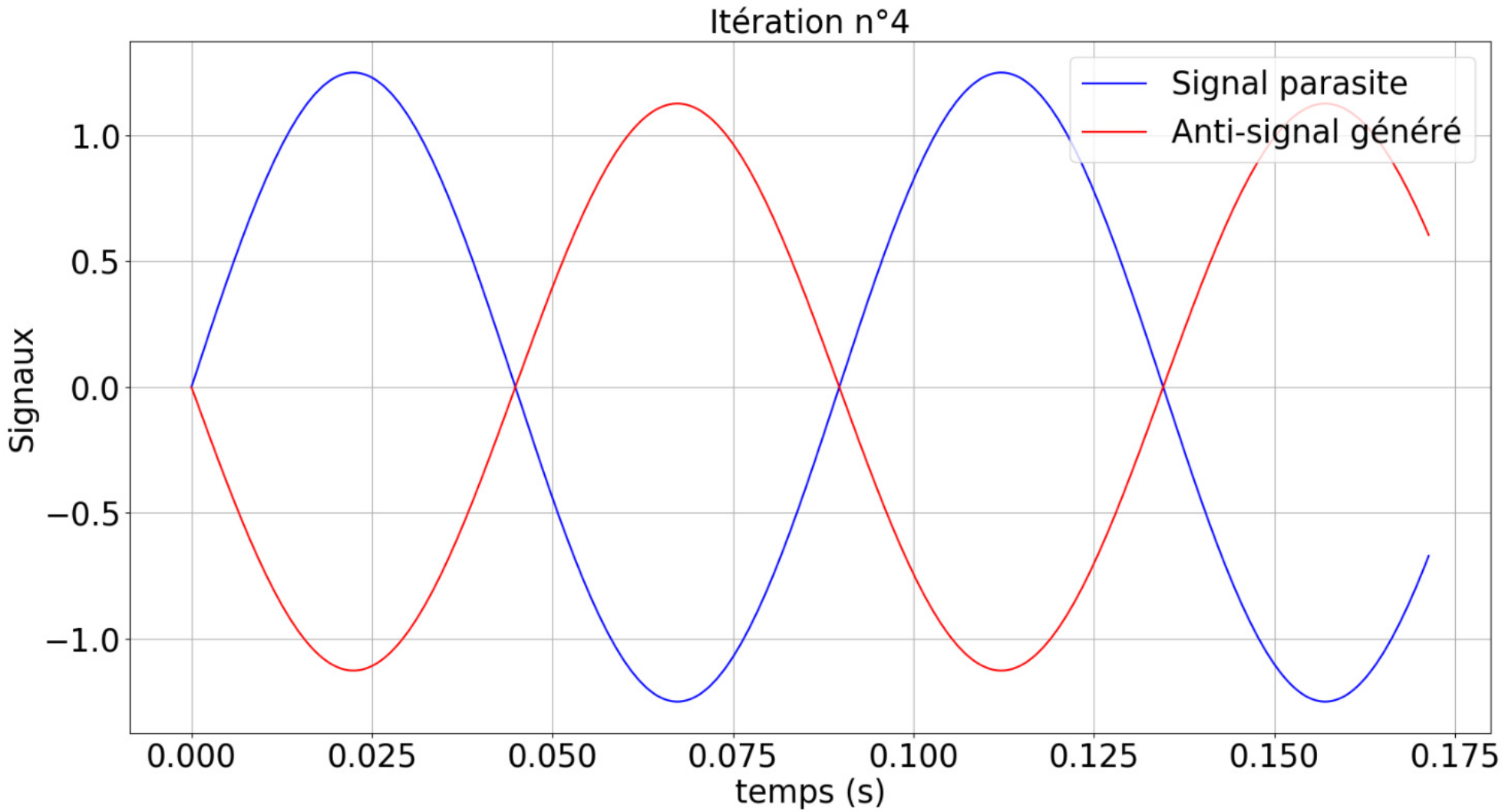
$$f : i \longmapsto \begin{cases} e^{\dfrac{i-c}{100}} & \text{si } i \leqslant c \\ 0 & \text{sinon} \end{cases}$$

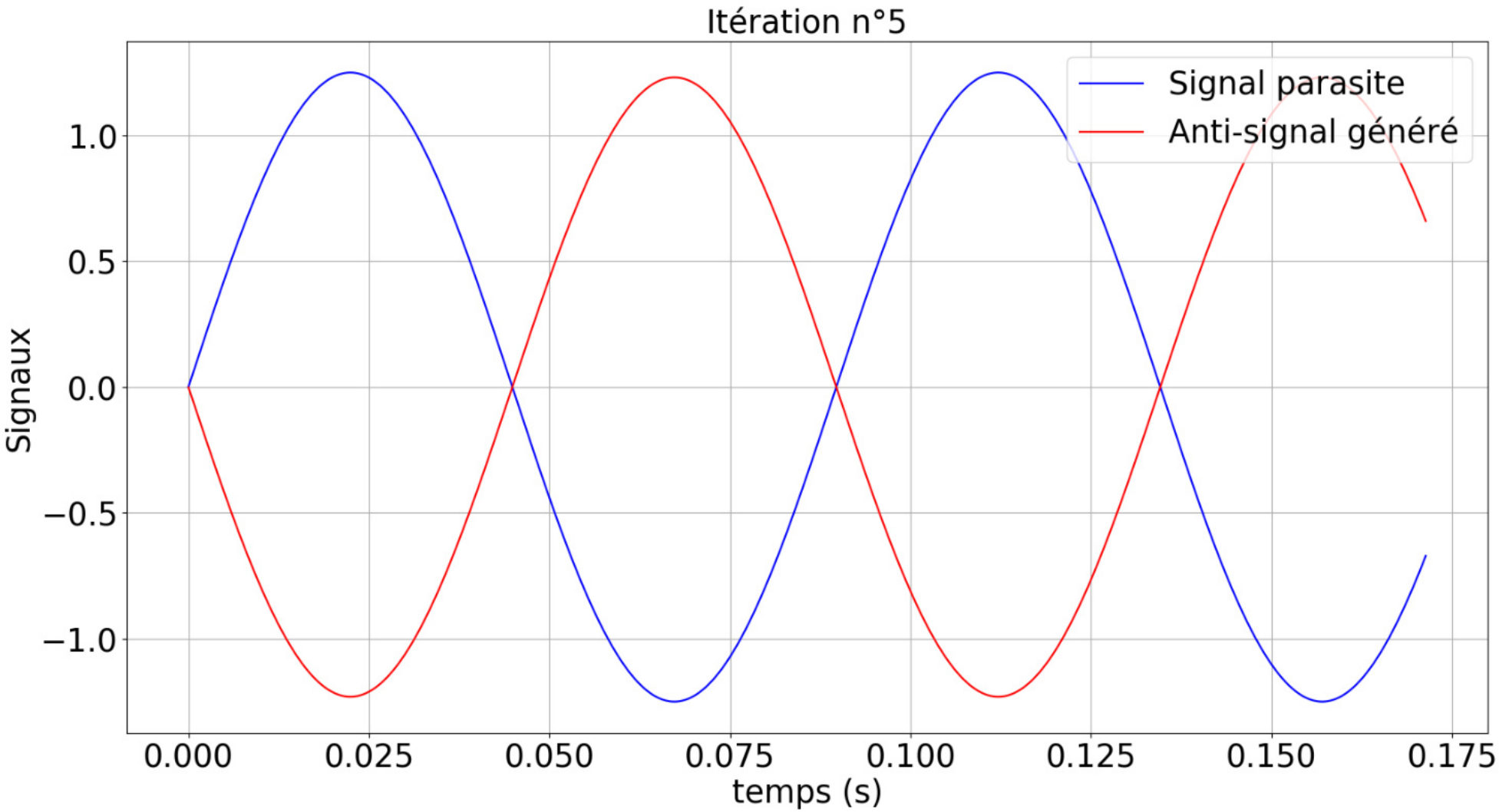

Itération n°0
Signal parasite
Anti-signal généré
Signaux
1.0
0.5
0.0
−0.5
−1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
temps (s)

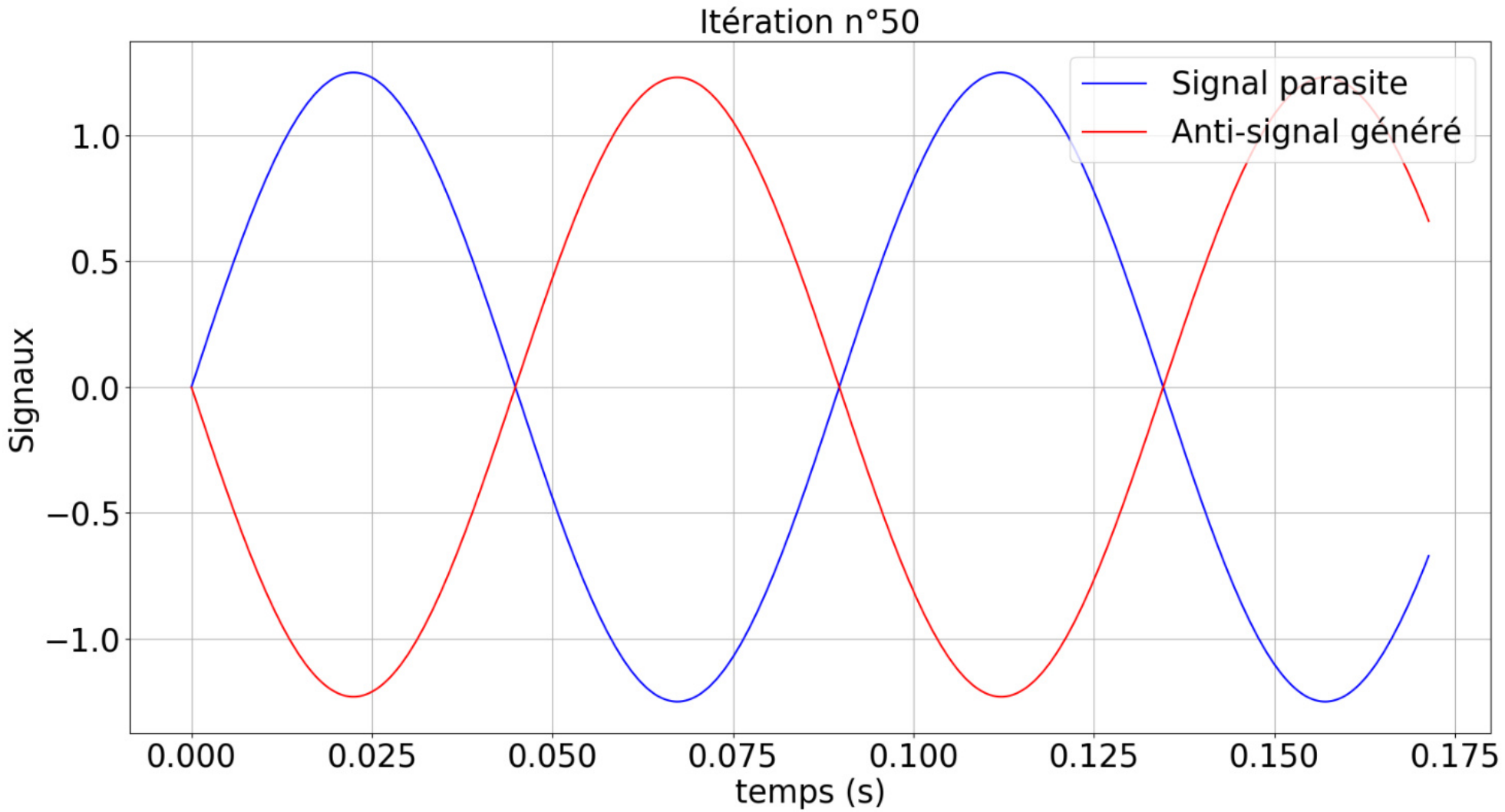

Itération n°1
Signal parasite
Anti-signal généré
Signaux
1.0
0.5
0.0
−0.5
−1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
temps (s)


Itération n°2
Signal parasite
Anti-signal généré
Signaux
1.0
0.5
0.0
−0.5
−1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
temps (s)


Itération n°3
Signal parasite
Anti-signal généré
Signaux
1.0
0.5
0.0
−0.5
−1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
temps (s)

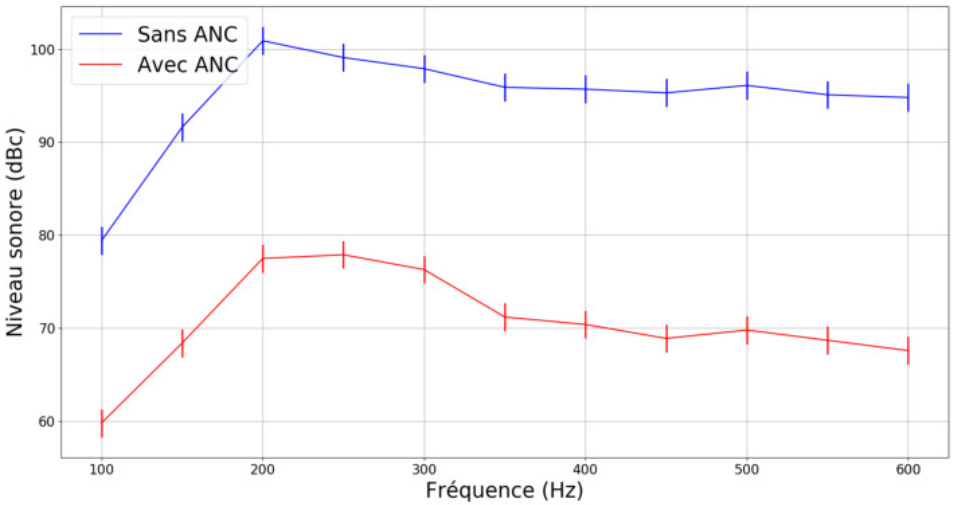

Itération n°4
Signal parasite
Anti-signal généré
Signaux
1.0
0.5
0.0
−0.5
−1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
temps (s)

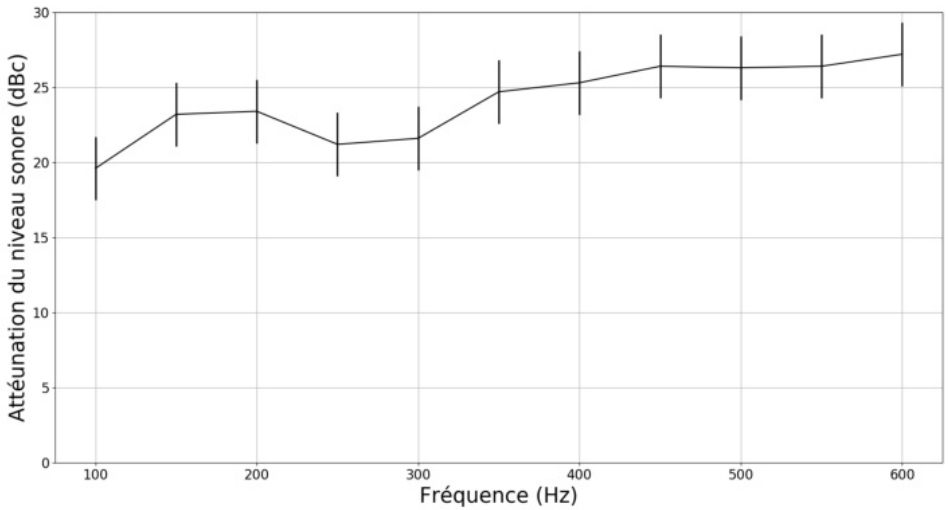

Itération n°5
Signal parasite
Anti-signal généré
Signaux
1.0
0.5
0.0
−0.5
−1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
temps (s)

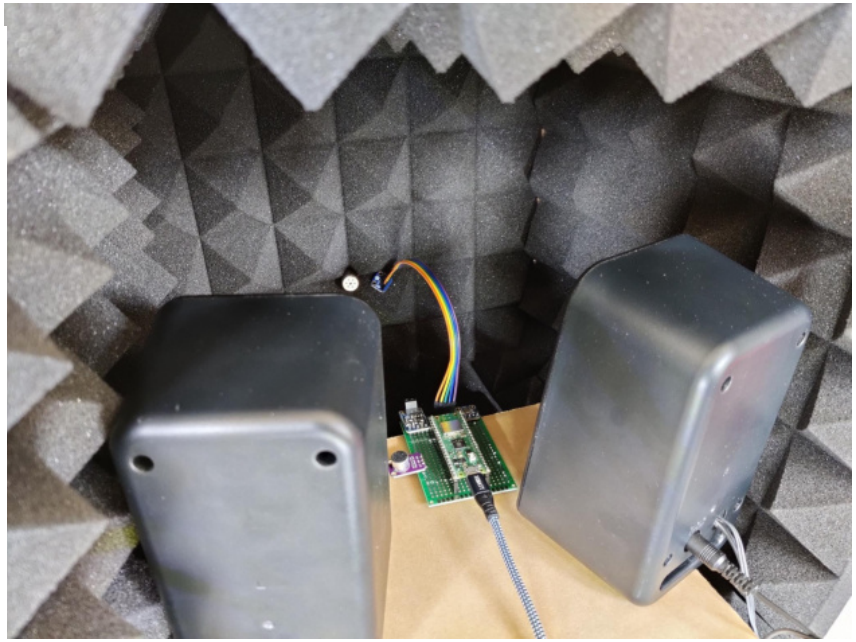

Itération n°50
Signal parasite
Anti-signal généré
Signaux
1.0
0.5
0.0
−0.5
−1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
temps (s)

Performances

Courbes d'atténuation expérimentales



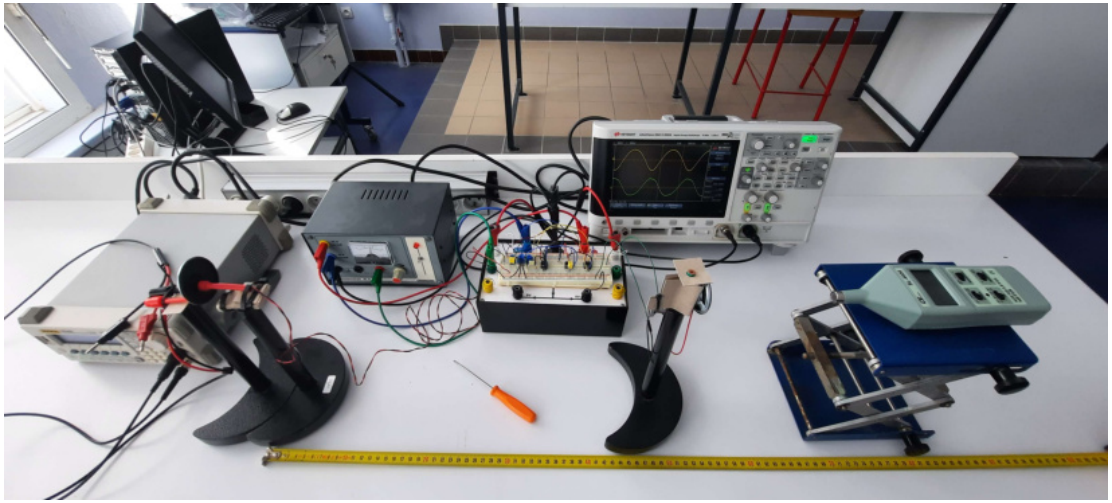




Atténuation moyenne

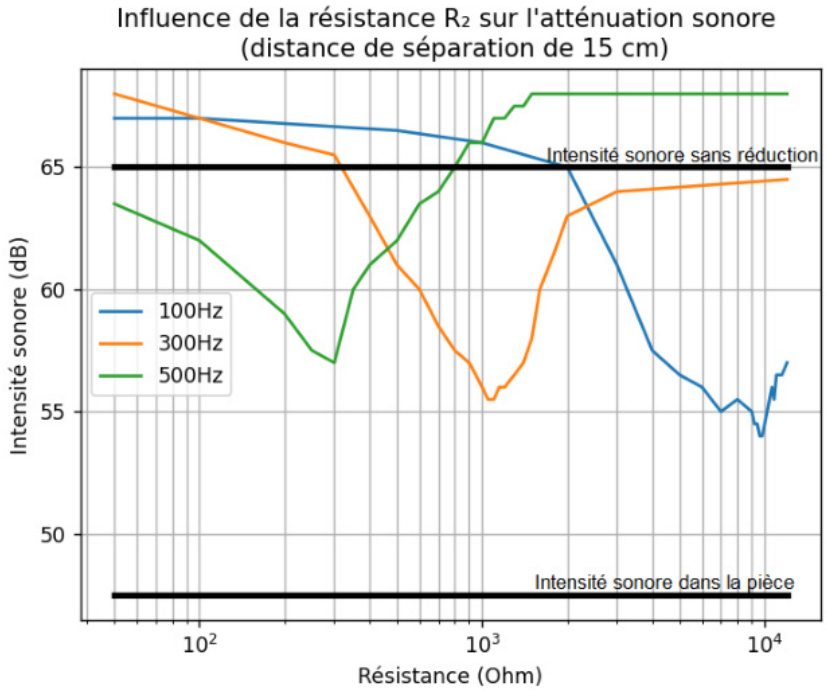
$24,1 \text{ dBc} \pm 2,1 \text{ dBc}$

Implémentation analogique (binôme)



Atténuation moyenne

$9,8 \text{ dBc} \pm 2,1 \text{ dBc}$



Comparaison sur la bande 0 - 500 Hz

Commerce	Numérique	Analogique
$15,3$ dBc $\pm 2,1$ dBc	$24,1$ dBc $\pm 2,1$ dBc	$9,8$ dBc $\pm 2,1$ dBc

Limites des systèmes

Analogique :

- Gestion amplitude

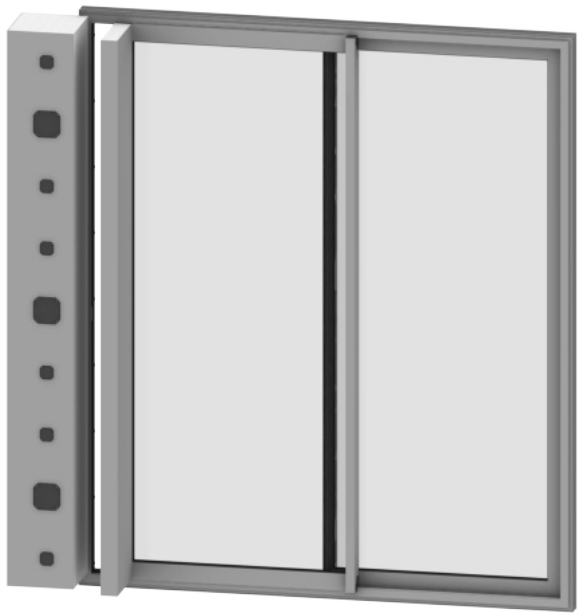
Numérique :

- Gestion déphasage
- Puissance de calcul

Applications

- Casques à réduction de bruit
- Processeur LAKE (Éric Dragon - ingénieur du son)

- Appuis-tête de véhicules
- Fenêtre à réduction de bruit (Patrick Lahbib - Technal)



Annexe 1.1 : Identification fréquence

```
1   import cmath
2   import time
3   import board # module de communication avec les broches du microcontrolleur
4   import analogio
5
6   def dftrapide(t, f):
7       '''Algorithme recursif de transformation de Fourier '''
8       Ni = len(t)
9       N = len(f)
10      x = cmath.exp(-2*cmath.pi*complex(0,1)/(1.*N))
11      if N == 2:
12          c = [(f[0] + (-1)**val*f[1])/2. for val in range(Ni)]
13      else:
14          fpaire = [f[i] for i in range(0, N, 2)]
15          fimpaire = [f[i] for i in range(1, N, 2)]
16          Pp = [val/2. for val in dftrapide(t, fpaire)]
17          Pi = [val/2. for val in dftrapide(t, fimpaire)]
18          c = []
19          for k in range(Ni):
20              c.append(Pp[k] + Pi[k] * x**k)
21      return c
```

Annexe 1.2 : Identification fréquence

```
22  def analyse(t, f):
23      c = dftrapide(t, f)
24      s = [val+val.conjugate() for val in c]
25      N = len(t)
26      T = []
27      for k in range( N // 2):
28          T.append(abs(s[k].real))
29      return T.index(max(T)) # on renvoie l'harmonique preponderante
30
31  def filtrage(e, Te, tau):
32      '''Filtre numerique passe-bas du premier ordre'''
33      s = [0]*len(e)
34      for i in range(len(s)-1):
35          s[i+1]=s[i] + (Te/tau)*(e[i]-s[i])
36      return s
37
38  def normalisation(e):
39      '''Normalise les valeurs d'une liste entre -1 et 1'''
40      for (i,x) in enumerate(e): # normalisation des donnees entre -1 et 1
41          e[i] = 2*(x-m)/(M-m) - 1
42      return e
```

Annexe 1.3 : Identification fréquence

```
43  microphone = analogio.AnalogIn(board.GP26)
44  echantillon = [0]*1024 # taille de l'echantillon a analyser
45  temps = [0.0]*1024 # liste des instants
46  T = 31968750 # duree d'echantillonnage (en ns)
47  Te = 1/32000 # periode d'echantillonnage (en s)
48  fc = 5000 # frequence de coupure a 5kHz
49  tau = 1/(2*cmath.pi*fc)
50  while True:
51      echantillon[0] = microphone.value
52      t1 = time.monotonic_ns()
53      for i in range(1, 1024): # remplissage de l'echantillon
54          k = t1 + i*T
55          t2 = time.monotonic_ns() - t1
56          while k < time.monotonic_ns() or t2 == temps[i-1]:
57              t2 = time.monotonic_ns() - t1
58          echantillon[i] = microphone.value # on enregistre la tension
59          temps[i] = t2/10**9 # on enregistre l'instant (en s)
60      echantillon = filtrage(echantillon, Te, tau) # application du filtre passe-bas
61      echantillon = normalisation(echantillon) # normalisation du signal
62      m, M = min(echantillon), max(echantillon)
63      analyse(temps, echantillon) # extraction de la frequence percue
```

Annexe 2.1 : Rétroaction d'amplitude

```
1   import numpy as np
2   import time
3   import math
4   import array
5   import board
6   import audiobusio
7   import audiopwmio
8
9   def stereo(gauche, droite):
10      '''Convertit deux sources mono en une source stereo'''
11      return np.ravel(np.vstack((gauche, droite)), order='F')
12
13  def sinus(haut_parleur, f, A, volume, duree):
14      '''Emet sur une paire de haut-parleurs un signal sinusoidal,
15      dephase avec des amplitudes differentes'''
16      fs = 32000
17      x = (1.25*np.sin((2 * np.pi * np.arange(fs * duree) * f) / fs)).astype(np.float32)
18      # signal simulant le bruit parasite
19      y = (-A*np.sin((2 * np.pi * np.arange(fs * duree) * f) / fs)).astype(np.float32)
20      # signal genere avec amplitude variable
21      signal = stereo(x,y)
22      haut_parleur.play(signal, loop=True)
23      time.sleep(duree)
24      haut_parleur.stop()
```

Annexe 2.2 : Rétroaction d'amplitude

```
25  def correction(n, i):
26      if n >= i - 30:
27          return np.exp((n - i + 30)/100) -1
28      return 0
29
30  def moyenne(L):
31      '''Renvoie la valeur moyenne d'une liste'''
32      return sum(L) / len(L)
33
34  def valeur_moyenne_normalisee(echantillon):
35      minbuf = int(moyenne(echantillon)) # soustraction de la composante continue
36      samples_sum = sum(
37          float(sample - minbuf) * (sample - minbuf)
38          for sample in echantillon)
39      return math.sqrt(samples_sum / len(echantillon))
40
41  def niveau_sonore(microphone):
42      '''Renvoie le niveau sonore en dB'''
43      echantillon = array.array('H', [0] * 320)
44      microphone.record(echantillon, len(echantillon))
45      amplitude = valeur_moyenne_normalisee(echantillon)
46      return 20 * math.log10(amplitude)
```

Annexe 2.3 : Rétroaction d'amplitude

```
47  haut_parleur = audiopwmio.PWMAudioOut(board.GP15)
48  # definition des broches des hauts parleurs
49  microphone_pdm = audiobusio.PDMIn(board.GP3, board.GP2, sample_rate=32000, bit_depth=16)
50  # definition des broches du microphone
51  f = 440 # frequence du bruit parasite
52  volume = 0.5
53  duration = 1.5
54  sinus(haut_parleur, f, 0, volume, 5)
55  A = 0.5
56  i0 = niveau_sonore(microphone_pdm) # niveau sonore de la source seule
57  while True:
58      sinus(haut_parleur, f, A, volume, duration) # emission du signal dephase
59      i = niveau_sonore(microphone_pdm) # mesure du nouveau niveau sonore
60      if (i > i0): # correction de l'amplitude
61          A -= correction(i, i0)
62      else:
63          A += correction(i, i0)
```