



## Rapport de Projet

---

# Traitement différentiel pour une antenne circulaire de microphones

---

**Auteurs :** Tom AUBIER et Raphaël DUMAS

**Promoteurs :** M. Pierre LECOMTE, Chercheur Postdoctoral,

M. Manuel MELON, Professeur des Universités,

M. Laurent SIMON, Professeur des Universités.



## Résumé

Le traitement différentiel d'ordre 1 d'une antenne circulaire de microphones est étudié dans ce compte rendu de projet de 3<sup>ème</sup> année de Licence Sciences Pour l'Ingénieur, en champ lointain et libre pour des ondes planes arrivant sur l'antenne. Ce traitement est modélisé dans le domaine fréquentiel pour une fréquence donnée, puis pour plusieurs fréquences afin de réaliser un motif de faisceau cardioïde large bande. Il est montré expérimentalement que la distance entre les microphones et l'ordre de la méthode différentielle limitent le traitement en hautes fréquences pour cause de repliement spatial, et en basses fréquences pour cause d'amplification du bruit. Une estimation de la formation de voie à appliquer est alors donnée. L'expérimentation étant réalisée en salle anéchoïque et respectant au mieux la condition de champ libre, elle permet ensuite de confirmer les résultats obtenus lors de la modélisation. Enfin, un traitement de l'antenne en temps réel est mis en place, mettant ainsi en avant une utilisation de cette dernière dans des conditions de salle réverbérante avec du bruit.

**Mots clés** – antenne, traitement différentiel, formation de voie, temps réel

## Abstract

The first order circular differential microphone array is studied during this report of our third Bachelor's year in Science for the Engineer, for plane waves impinging on the array in the far and free field. Spatial filters are applied in the frequency domain to each microphone of the array for a given frequency, then for multiple frequencies to get a broadband cardioïd beamforming. Because of the distance between microphones and the order of the beampattern, it is shown experimentally that these spatial filters are limited in high frequency due to spatial aliasing and in low frequency due to noise amplification. An estimate of these filters is then given. The experimentation being realized in an anechoical room with respect to the free field condition, it can then confirm the results obtained previously. Finally, this beamforming is applied in real time, in a noisy and reverberant environment.

**Index Terms** – differential microphone array, broadband beamforming, real time

# Remerciements

Nous tenons vivement à remercier les promoteurs de ce projet, respectivement M. Pierre LECOMTE, Chercheur Postdoctoral ; M. Manuel MELON, Professeur des Universités ; et M. Laurent SIMON, Professeur des Universités ; pour leurs qualités de promoteur. Nous accordant les moyens nécessaires à la réalisation de ce projet et nous laissant une autonomie propre à stimuler nos intérêts, ils nous ont ainsi permis d'effectuer un travail qui nous a été d'une grande richesse.

Nous tenons à remercier le Laboratoire d'Acoustique de l'Université du Mans (LAUM), nous permettant de réaliser l'ensemble des mesures expérimentales au sein de leur salle anéchoïque.

Nous remercions également Mme. Catherine POTEL, Professeur d'Expression Scientifique et Technique, pour ses précieux conseils et analyses nous permettant d'améliorer nos qualités d'écriture, nos présentations manuscrites et nos prestations orales.

Nous remercions également Mr. Alann RENAULT, Ingénieur d'Études au LAUM, nous permettant la réalisation de l'impression 3D du support de l'antenne utilisée lors de ce projet dans les plus brefs délais.

# Table des Matières

<b>Introduction</b> .....	<b>1</b>
<b>1 Modélisation</b> .....	<b>2</b>
1.1 Champ de pression .....	2
1.2 Traitement de l'antenne pour une fréquence .....	4
1.3 Traitement de l'antenne large bande .....	7
<b>2 Expérimentation</b> .....	<b>11</b>
2.1 Antenne xCORE .....	11
2.2 Traitement de l'antenne .....	12
2.3 Implémentation en temps réel .....	15
Conclusion .....	18
<b>A Script Python : differentialarray.py</b> .....	<b>19</b>

# Liste des Figures

1.1	Représentation d'une antenne circulaire uniforme . . . . .	2
1.2	Représentation de la partie réelle du champ de pression au temps $t = 0$ pour $M = 3$ et $k_0R \approx 0.4$ . . . . .	3
1.3	Représentation de la pression $y_m[n]$ captée par les $M$ microphones (zoom sur une période $T_0$ ) . . . . .	4
1.4	Schéma représentant la méthode différentielle utilisée . . . . .	4
1.5	Exemple d'un motif de faisceau d'ordre 3 . . . . .	5
1.6	Représentation du motif de faisceau orienté vers $\phi_1 = 0$ rad (en noir) et modélisation de l'expérience (en rouge) . . . . .	7
1.7	Module des filtres $H_m[f_k]$ . . . . .	8
1.8	Représentation de l'interpolation effectuée sur un nombre de points $N_{interp}$ . . . . .	9
1.9	Représentation des réponses impulsionales $RI_m[n]$ . . . . .	9
1.10	Comparaison entre le signal de sortie $z[n]$ et le signal du microphone $y_1[n]$ . . . . .	10
2.1	Photographie [13] de l'antenne de microphones xCORE . . . . .	11
2.2	Modélisation 3D du support de l'antenne . . . . .	12
2.3	Représentation du dispositif expérimental vue de côté avec le haut-parleur (HP), l'antenne utilisée et la table tournante en bleu . . . . .	13
2.4	Comparaison du motif de faisceau théorique (en noir) et expérimental (en rouge) . . . . .	14
2.5	Angle $\theta$ en fonction de la fréquence . . . . .	14
2.6	Capture d'écran de l'interface graphique réalisée . . . . .	15
2.7	Schéma représentatif du traitement en temps réel . . . . .	16

# Introduction

L'analyse de champs acoustiques par des antennes de microphones est un sujet de recherche en plein essor depuis 1930, ces dernières permettant de résoudre nombre de problèmes tels que la réduction de bruit, la dé-réverbération, la séparation de sources ou encore la caractérisation de matériaux. Composée de plusieurs microphones répartis spatialement en une géométrie organisée, une antenne permet notamment de réaliser une forme de directivité cible à l'aide d'un traitement du signal adapté, en opposition à un microphone seul caractérisé par un filtrage spatial intrinsèque.

Deux types de traitement sont couramment utilisés : le *delay and sum*, dont le principe est d'appliquer un retard à chaque signal microphonique de l'antenne pour ensuite les sommer ; et le traitement *différentiel* dont le principe est de réaliser une antenne sensible à la dérivée d'ordre  $N$  du champ de pression par différence finie, en considérant que la distance entre les microphones  $\delta$  est petite devant la longueur d'onde  $\lambda$ , soit  $\delta \ll \lambda$ . L'avantage principal de cette dernière méthode sur la première est sa capacité inhérente à former un filtrage spatial invariant quelque soit la fréquence. De plus, étant de taille relativement petite, l'antenne peut être facilement intégrée dans des appareils de communication.

Par suite, l'objectif de cette étude consiste à appliquer un traitement différentiel à une antenne de géométrie circulaire. Dans le premier chapitre, la modélisation du champ de pression et des filtres à appliquer à chaque microphone est détaillée pour une fréquence donnée dans un premier temps, puis pour plusieurs fréquences avec une méthode différentielle. Les résultats de la modélisation sont alors comparés à l'expérience à l'aide d'une antenne microphonique conçue par l'entreprise XMOS au deuxième et dernier chapitre. L'ensemble des chapitres a pour objectif secondaire d'introduire les notions permettant de mettre en oeuvre un filtrage spatial en temps réel.

A noter que pour mener à bien le traitement d'une antenne circulaire, la méthode différentielle utilisée dans cette étude est décrite par Jacob BENESTY dans son livre *Design of Circular Differential Microphone Arrays* [6].

# Chapitre 1

## Modélisation

L'objectif de ce chapitre est tout d'abord de modéliser un champ de pression permettant de simuler les signaux microphoniques de l'antenne, puis d'appliquer à ces derniers un traitement pour une fréquence donnée avec une méthode différentielle du premier ordre. Enfin, la modélisation se porte sur le traitement de l'antenne pour des signaux large bande en fréquence.

### 1.1 Champ de pression

L'antenne circulaire considérée ici est constituée de  $M$  microphones, assimilés à des monopôles. La configuration étudiée ici est à 2 dimensions. Les  $M$  microphones de l'antenne circulaire, Figure 1.1, sont repérés par leur position  $\mathbf{M}_m$ , telle que

$$\mathbf{M}_m = (R \cos \phi_m \quad R \sin \phi_m)^T, \quad (1.1)$$

avec  $R$  le rayon de l'antenne,  $\phi_m$  les angles des microphones pour  $m$  allant de 1 à  $M$ , et  ${}^T$  l'opérateur transposé. L'antenne circulaire étant uniforme, les angles  $\phi_m$  peuvent donc s'écrire

$$\phi_m = (m - 1) \frac{2\pi}{M}. \quad (1.2)$$

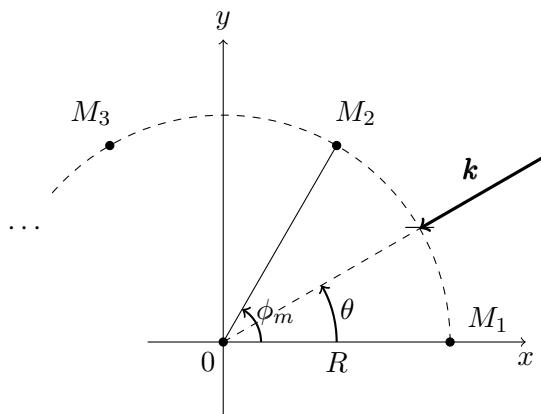


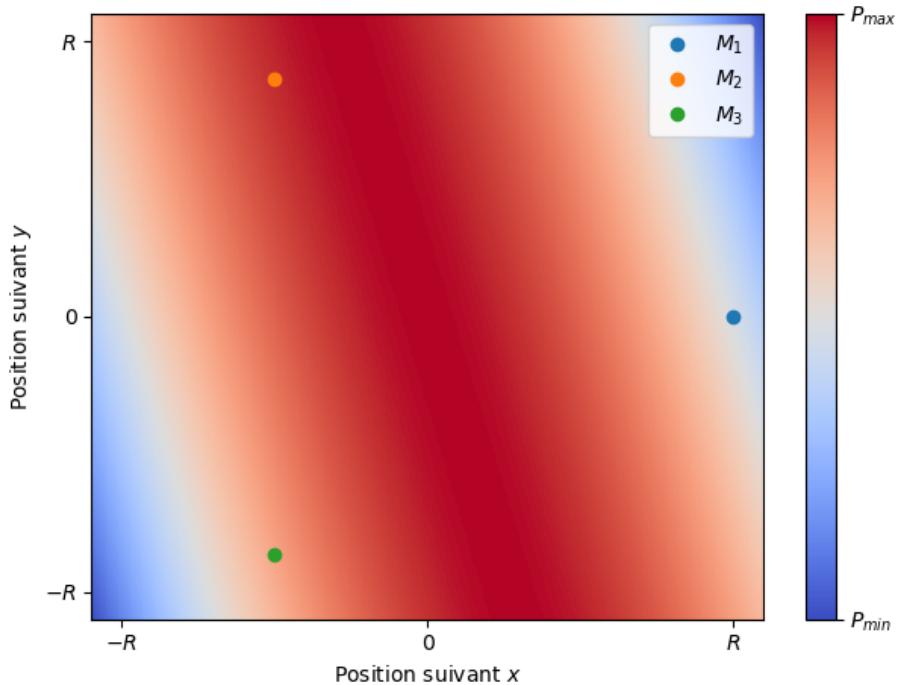
FIGURE 1.1 – Représentation d'une antenne circulaire uniforme

Le milieu de propagation est par ailleurs supposé homogène, isotrope et sans dispersion. La modélisation se limite en champ lointain au cas d'ondes planes en problème plan. En régime harmonique, la pression complexe d'une onde plane [5] arrivant sur l'antenne en champ libre

pour un microphone  $m$  s'écrit par rapport au centre de l'antenne

$$p(\mathbf{M}_m, t) = P e^{j(\omega t - \mathbf{k} \cdot \mathbf{M}_m)} ,$$

avec  $\mathbf{k} = (-k_0 \cos \theta \quad -k_0 \sin \theta)^T$  le vecteur nombre d'onde et  $\theta$  l'angle d'incidence de l'onde plane. Le nombre d'onde vérifie la relation de dispersion en milieu isotrope  $\|\mathbf{k}\| = k_0 = \omega/c$  avec  $\omega$  la pulsation et  $c$  la vitesse de propagation dans le milieu. Comme indiqué dans l'Introduction page 1, la distance  $\delta$  est petite devant la longueur d'onde  $\lambda$  : la pression complexe varie peu d'un microphone à un autre.

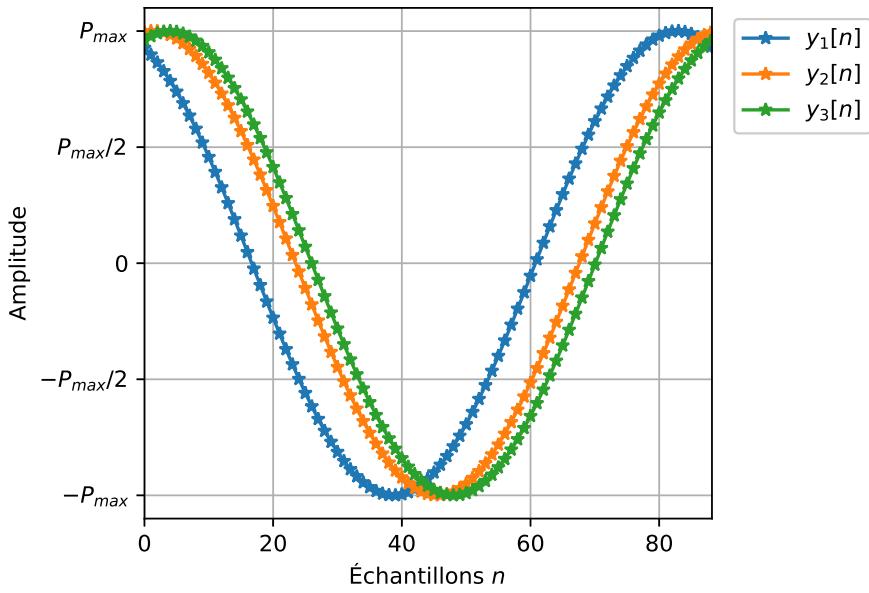


**FIGURE 1.2 – Représentation de la partie réelle du champ de pression au temps  $t = 0$  pour  $M = 3$  et  $k_0 R \approx 0.4$**

Une fois traduit en langage Python (Annexe A page 19), il est alors possible de simuler un champ de pression dans le milieu considéré, en particulier au niveau des microphones de l'antenne. La partie réelle du champ de pression est représentée Figure 1.2 pour une onde plane d'amplitude complexe  $P$  quelconque, de fréquence  $F_0 = 500$  Hz et d'angle d'incidence  $\theta = \pi/12$  rad. Les  $M$  microphones de l'antenne sont de même représentés par des points, avec ici  $M = 3$ .

La partie réelle de la pression captée par le microphone  $m$  (notée  $y_m[n]$  après échantillonnage à la fréquence  $F_s = 1/T_s$ ) est représentée Figure 1.3 page suivante. Le zoom est proposé sur une période du signal  $T_0 = 1/F_0$ . Les microphones sont supposés omnidirectionnels et de mêmes caractéristiques (réponse fréquentielle, gain, etc).

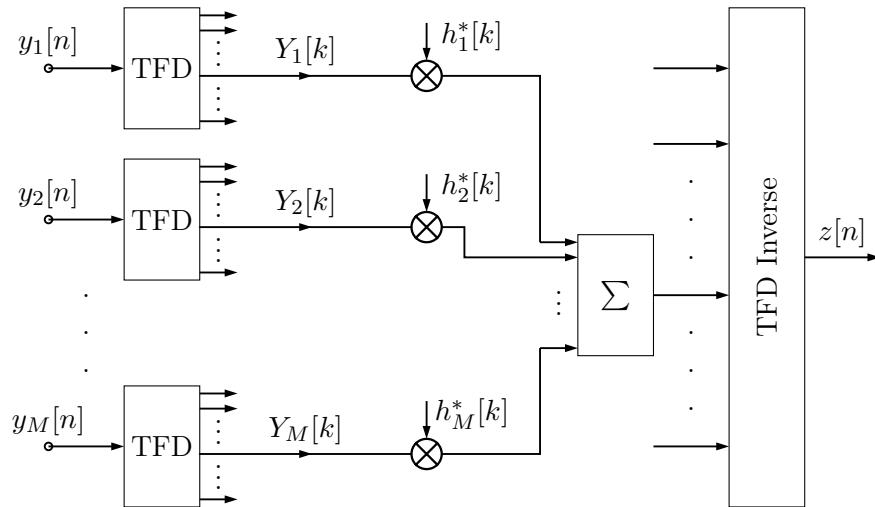
Il est maintenant possible de réaliser un filtrage spatial à l'aide d'une méthode différentielle, ce qui est détaillé dans la section suivante.



**FIGURE 1.3 – Représentation de la pression  $y_m[n]$  captée par les  $M$  microphones (zoom sur une période  $T_0$ )**

## 1.2 Traitement de l'antenne pour une fréquence

L'objectif de cette section est de réaliser un traitement d'antenne pour une fréquence donnée et de modéliser l'expérience qui sera réalisée par la suite.



**FIGURE 1.4 – Schéma représentant la méthode différentielle utilisée**

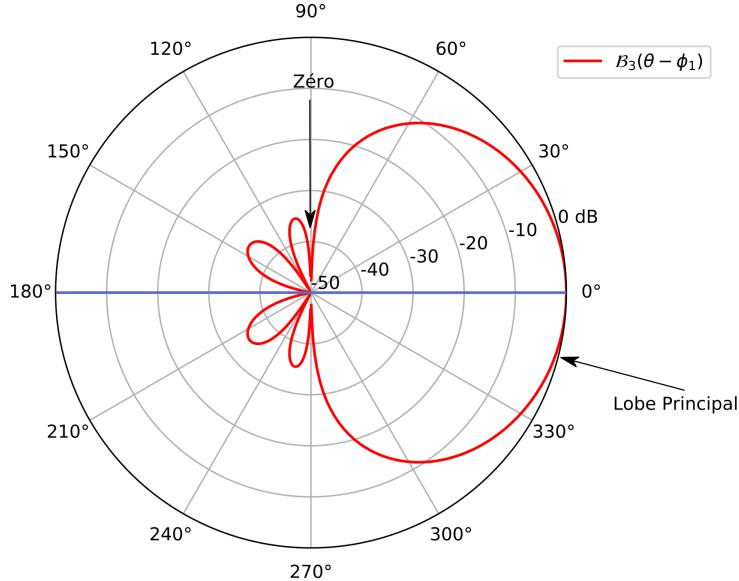
Avec la méthode différentielle décrite par Jacob BENESTY [6] représentée Figure 1.4, il faut dans un premier temps appliquer une Transformée de Fourier Discrète (TFD) sur  $N_{tfid}$  points pour chaque signal microphonique,

$$y_m[n] \xrightarrow{\text{TFD}} Y_m[k] \quad \text{avec } k \text{ entier, } k \in [0, N_{tfid} - 1] .$$

Pour une composante fréquentielle  $k$ , la valeur des TFD  $Y_m[k]$  s'écrit par la suite sous forme

vectorielle comme

$$\mathbf{y}[k] = (Y_1[k] \quad Y_2[k] \quad \dots \quad Y_M[k])^T .$$



**FIGURE 1.5 – Exemple d'un motif de faisceau d'ordre 3**

L'objectif est par la suite d'appliquer, pour chaque microphone  $m$  et pour chaque composante  $k$ , un poids  $h_m[k]$  à estimer en vue d'obtenir une directivité donnée de l'antenne. Les poids  $h_m[k]$  sont notés sous forme vectorielle comme

$$\mathbf{h}[k] = (h_1[k] \quad h_2[k] \quad \dots \quad h_M[k])^T .$$

Il est nécessaire de connaître le motif cible du faisceau à réaliser, noté  $\mathcal{B}_N(\theta - \theta_s)$  avec  $\theta_s$  l'angle pilotant le faisceau et  $\theta \in [0, 2\pi]$ . D'une manière générale, le motif de faisceau est appelé limaçon de Pascal [11]. Il se retrouve dans de nombreux domaines, notamment en mathématiques avec l'ensemble de Mandelbrot [12] et en physique avec par exemple la réflexion des rayons lumineux dans une tasse de café [7]. Par définition, dans le cadre de cette étude, on considère

$$\mathcal{B}_N(\theta - \theta_s) = \sum_n a_{N,n} \cos^n(\theta - \theta_s) , \text{ avec } \sum_n a_{N,n} = 1 , \quad (1.3)$$

avec  $N$  l'ordre de la méthode différentielle. Le motif du faisceau possède donc un axe de symétrie comme représenté en bleu Figure 1.5. Pour respecter la symétrie avec cette méthode, il faut que l'angle pilotant le faisceau  $\theta_s$  corresponde à l'angle d'un microphone de l'antenne  $\phi_m$  (voir [6]). De manière équivalente à l'équation (1.3), le motif de faisceau peut également être défini comme le module de la réponse en fréquence, auquel cas il est noté

$$\mathcal{B}_N[k, \theta] = |\mathbf{d}^H[k, \theta] \mathbf{h}[k]| , \quad (1.4)$$

avec  $\mathcal{B}_N[k, \theta = \theta_s] = 1$  la contrainte de distorsion et  $^H$  l'opérateur conjugué transposé, le vecteur de pilotage de l'antenne  $\mathbf{d}[k, \theta]$  étant défini comme

$$\mathbf{d}[k, \theta] = (e^{j\varpi_k \cos(\theta - \phi_1)} \quad e^{j\varpi_k \cos(\theta - \phi_2)} \quad \dots \quad e^{j\varpi_k \cos(\theta - \phi_M)})^T ,$$

avec  $\varpi_k = 2\pi k \Delta f \frac{R}{c}$  et  $\Delta f = F_s/N_{tfld}$ .

Il est maintenant possible de déterminer les gains complexes  $\mathbf{h}[k]$  permettant de réaliser le filtrage pour une composante  $k \in [0, N_{tf}/2 - 1]$ .

Pour 3 microphones, en supposant que la source soit située à un angle  $\theta = \phi_1$ , le lobe principal du faisceau est orienté suivant l'angle  $\phi_1$  du microphone  $M_1$ . La contrainte de distorsion de l'équation (1.4) devant être respectée, on a alors

$$\mathbf{d}^H[k, \phi_1] \mathbf{h}[k] = 1 . \quad (1.5)$$

Par ailleurs, le motif de faisceau possède un unique zéro pour une méthode différentielle d'ordre 1. Pour réaliser un motif de faisceau cardioïde, il est possible par exemple de placer le zéro en  $\phi_1 + \pi$  tel que

$$\mathbf{d}^H[k, \phi_1 + \pi] \mathbf{h}[k] = 0 . \quad (1.6)$$

Enfin, pour respecter la symétrie du motif de faisceau, les gains à appliquer aux microphones symétriques par rapport à l'axe ( $OM_1$ ) doivent être identiques, c'est à dire

$$\mathbf{c}^T \mathbf{h}[k] = 0 , \quad (1.7)$$

avec dans ce cas  $\mathbf{c} = (0 \ 1 \ -1)^T$ .

En écrivant les équations (1.5), (1.6) et (1.7) sous forme matricielle, avec la matrice

$$\mathbf{A}[k] = \begin{pmatrix} \mathbf{d}^H[k, \phi_1] \\ \mathbf{d}^H[k, \phi_1 + \pi] \\ \mathbf{c}^T \end{pmatrix} ,$$

et le vecteur

$$\mathbf{b} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} ,$$

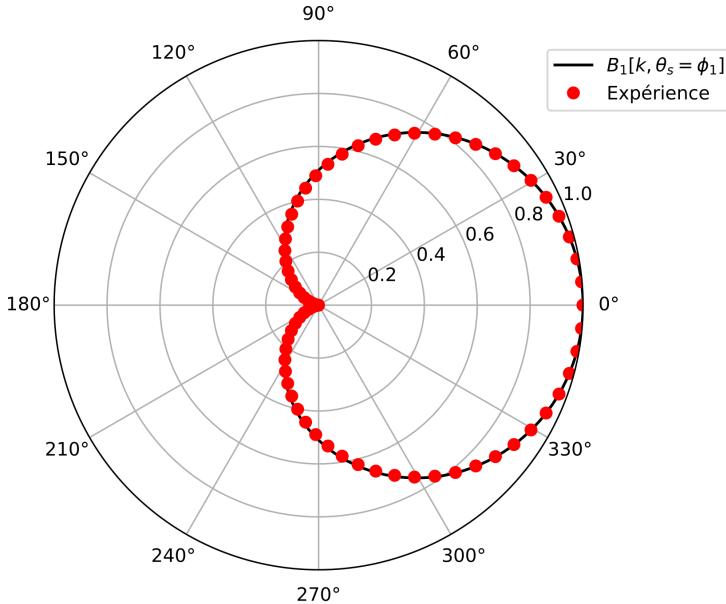
la valeur des gains complexes  $\mathbf{h}$  à appliquer à chaque microphone pour une composante  $k$  est alors

$$\mathbf{h}[k] = \mathbf{A}^{-1}[k] \mathbf{b} . \quad (1.8)$$

Finalement, la valeur  $Z[k]$  du signal de sortie après filtrage par  $h[k]$  est  $Z[k] = \mathbf{h}^H[k] \mathbf{y}[k]$ . Il faut appliquer une TFD Inverse sur l'ensemble des fréquences pour obtenir le signal temporel de sortie  $z[n]$ .

A l'aide d'un script Python et de la relation (1.4), le motif du faisceau obtenu est représenté en basse fréquence Figure 1.6 page suivante pour un lobe principal orienté vers  $\phi_1 = 0$  rad. Il est nécessaire de vérifier que le signal de sortie  $z[n]$  dépende bien de l'angle d'incidence de la source  $\theta$ . Le diagramme de directivité modélisant l'expérience est donné pour un angle d'incidence de la source variant par pas de 5 degrés.

La courbe obtenue Figure 1.6 page ci-contre est cohérente avec les prédictions du modèle, mais le traitement réalisé est cependant limité comme il est très peu probable en réalité que la source soit monochromatique. Un des avantages du traitement différentiel est que le motif de faisceau est large bande. Par conséquent, obtenir un motif de faisceau à une fréquence implique qu'il reste identique à plusieurs fréquences préalablement déterminées. La section suivante s'intéresse donc à un traitement pour plusieurs fréquences, comme c'est par exemple le cas d'un signal vocal.



**FIGURE 1.6** – Représentation du motif de faisceau orienté vers  $\phi_1 = 0$  rad (en noir) et modélisation de l'expérience (en rouge)

### 1.3 Traitement de l'antenne large bande

L'objectif de cette section est de réaliser le traitement sur une bande de fréquences préalablement déterminées. Enfin, les réponses impulsionales des filtres obtenus sont données pour une utilisation de l'antenne en temps réel.

Comme vu en cours de Traitement du Signal au semestre 5, la Transformée de Fourier Discrète  $X[k]$  d'un signal réel  $x[n]$  possède une symétrie Hermitienne, c'est à dire

$$X[k] = X^*[N_{tf} - k] , \quad (1.9)$$

avec \* l'opérateur conjugué et pour  $N_{tf}$  pair.

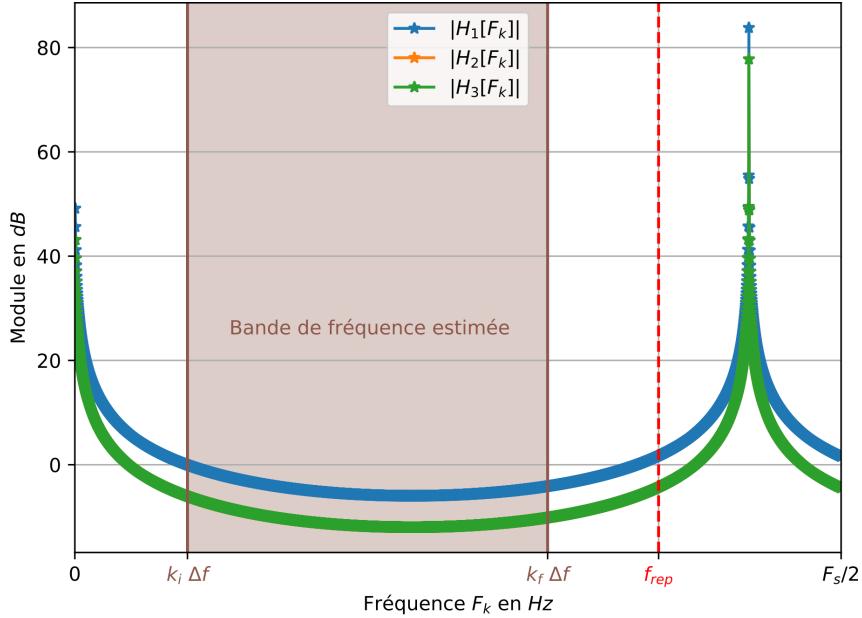
En plaçant l'ensemble des gains complexes dans une matrice  $\mathbf{H}$  de taille  $M \times N_{tf}$ , alors la relation (1.9) impose

$$\mathbf{H} = (\mathbf{h}[0] \quad \mathbf{h}[1] \quad \dots \quad \mathbf{h}^*[2] \quad \mathbf{h}^*[1]) .$$

Une représentation du module des filtres obtenus  $H_m[k]$ , correspondant aux lignes de la matrice  $\mathbf{H}$ , est donnée Figure 1.7 page suivante. Il est possible de constater que pour une fréquence donnée, le module des gains complexes à appliquer peut devenir très élevé. Cela s'explique par la prise en compte du repliement spatial dont la fréquence  $f_{rep}$  est estimée à partir de la relation  $\delta < \frac{\lambda}{2}$ , par suite,

$$f_{rep} = \frac{c}{2\delta} . \quad (1.10)$$

La relation précédente ne prenant pas en compte l'écart de phase entre les microphones, la fréquence de repliement représentée Figure 1.7 page suivante est sous-estimée quelque soit  $c$  et  $\delta$ . Cela n'a cependant pas d'importance car appliquer un traitement au-delà de cette fréquence n'a pas de sens. De plus, l'antenne amplifie le bruit en basses fréquences puisque la méthode différentielle est basée sur un schéma aux différences finies, comme expliqué dans l'introduction page 1.



**FIGURE 1.7 – Module des filtres  $H_m[f_k]$**

Appliquer un traitement sur l'ensemble des fréquences n'étant pas réalisable (les gains étant trop élevés), il est alors nécessaire d'imposer une matrice  $\mathbf{H}$  permettant d'assurer le traitement dans une bande  $k \in [k_i, k_f]$  avec  $k_f < N_{tfid}/2 - 1$ .

Les composantes  $k_i$  et  $k_f$  représentées Figure 1.7 sont déterminées pendant l'expérimentation Chapitre 2 page 11 en fonction de l'estimation de l'amplification du bruit en basses fréquences et de la fréquence de repliement (1.10) en hautes fréquences. Néanmoins, aucun traitement n'étant souhaité pour tout  $k \notin [k_i, k_f]$ , les gains sont nuls. La matrice  $\mathbf{H}$  peut donc s'écrire

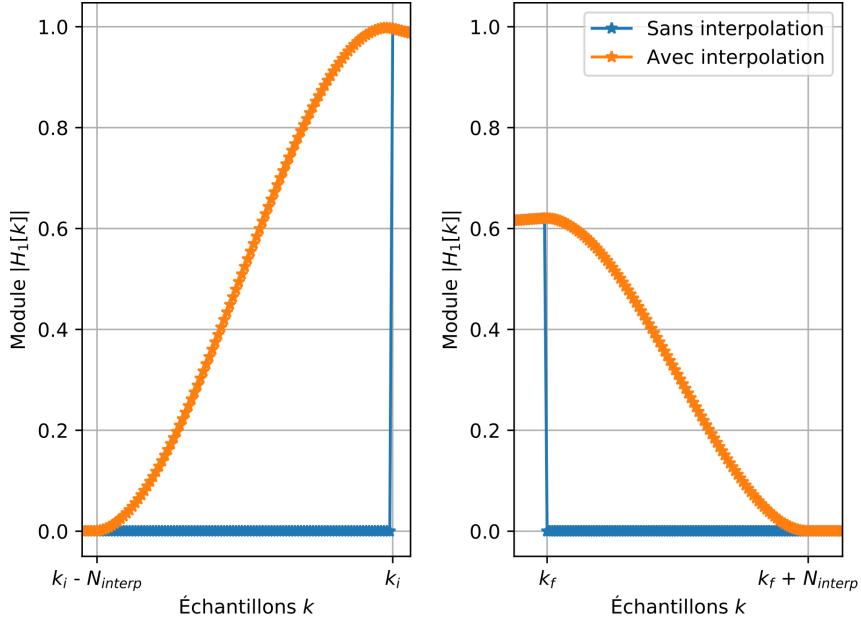
$$\mathbf{H} = (\mathbf{0} \quad \cdots \quad \mathbf{0} \quad \mathbf{h}[k_i] \quad \mathbf{h}[k_i + 1] \quad \cdots \quad \mathbf{h}[k_f] \quad \mathbf{0} \quad \cdots \quad \mathbf{0}) , \quad (1.11)$$

avec le vecteur  $\mathbf{0}$  de taille  $M$ .

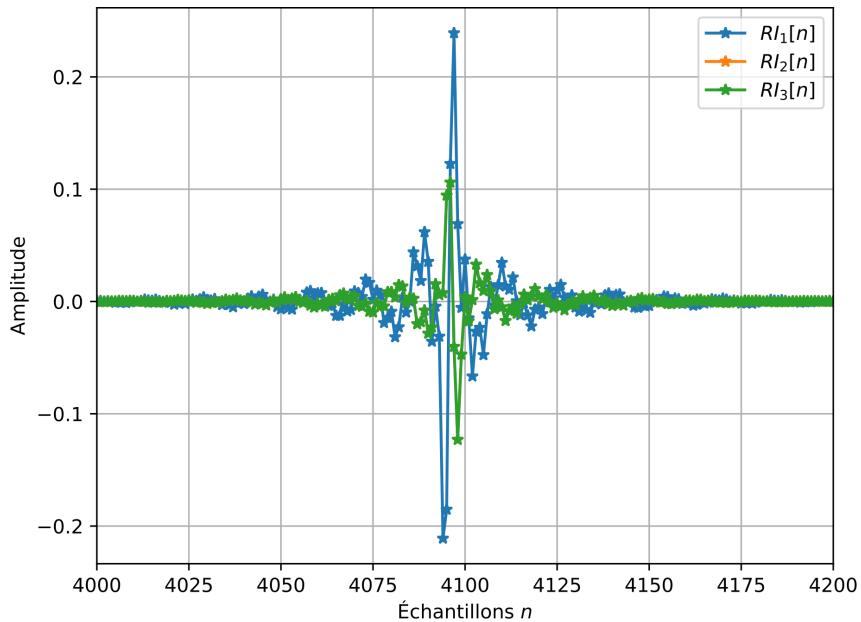
Cette solution a l'inconvénient d'ajouter des discontinuités et cela pose problème comme les effets de bords sont importants. Il est possible de contourner ce problème en lissant les effets des discontinuités comme présenté Figure 1.8 page ci-contre.

Le traitement pour tout  $k \in [0, N_{tfid}/2 - 1]$  une fois réalisé, il reste à appliquer la symétrie Hermitienne (1.9) pour que  $\mathbf{H}$  soit complètement déterminée, permettant ainsi une implémentation du filtrage en temps réel.

Les algorithmes temporels de convolution sous Python comme *lfilter*, *convolve1d* ou encore *fftconvolve* permettent une implémentation du filtrage en temps réel très rapide. C'est pourquoi une Transformée de Fourier Discrète Inverse est appliquée à la matrice  $\mathbf{H}^*$  afin d'obtenir les réponses impulsionales notées  $RI_m[n]$ . Étant non causales, il est nécessaire de leur appliquer un retard pour les rendre causales et permettre une implémentation en temps réel. Comme représenté Figure 1.9 page suivante, la contrainte de symétrie (1.7) étant bien respectée, il n'est par suite pas possible de distinguer les réponses impulsionales  $RI_2[n]$  et  $RI_3[n]$ .



**FIGURE 1.8 – Représentation de l’interpolation effectuée sur un nombre de points  $N_{\text{interp}}$**



**FIGURE 1.9 – Représentation des réponses impulsionales  $RI_m[n]$**

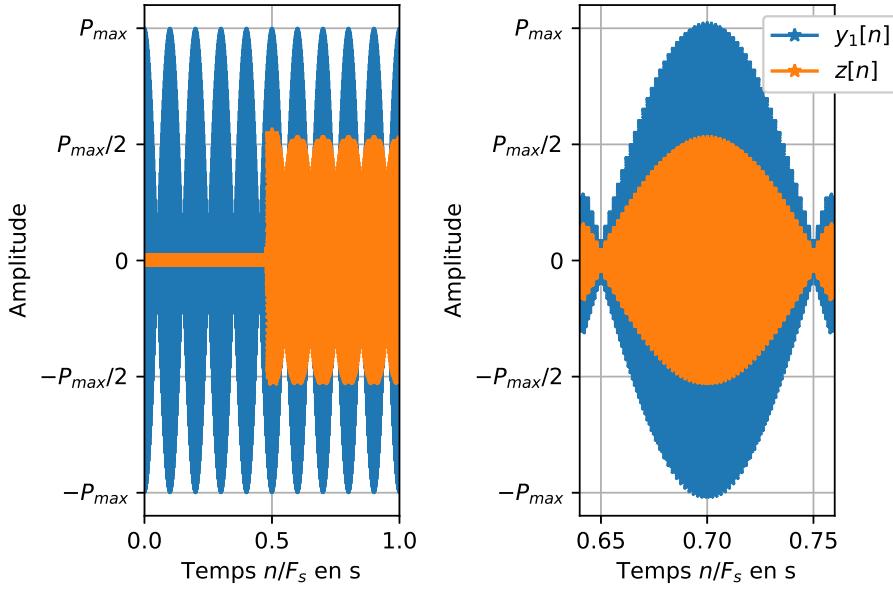
Finalement, le signal temporel de sortie  $z[n]$  s’écrit

$$z[n] = \sum_{m=1}^M y_m[n] * RI_m[n] , \quad (1.12)$$

avec  $*$  l’opérateur de convolution.

Dans le cadre de l’acoustique linéaire, il est possible de vérifier la modélisation grâce au principe de superposition. A partir de plusieurs ondes planes monochromatiques avec le même angle d’incidence  $\theta$ , il suffit d’additionner les champs de pression et de prendre la partie réelle

pour modéliser les signaux microphoniques. La représentation du signal microphonique  $y_1[n]$  et du signal de sortie  $z[n]$  est donnée Figure 1.10 pour une source possédant un angle d'incidence  $\theta = \pi/2$  rad et composée de deux fréquences quelconques situées en basse fréquence entre  $f_{k_i}$  et  $f_{k_f}$ . L'algorithme de convolution utilisé ici est la *fftconvolve* de la bibliothèque *Scipy* sous Python.



**FIGURE 1.10 – Comparaison entre le signal de sortie  $z[n]$  et le signal du microphone  $y_1[n]$**

Le fait qu'il y ait un décalage temporel entre le microphone et le signal de sortie n'est pas surprenant et est lié au traitement. Le contenu fréquentiel étant dans la bande de fréquences du traitement, le filtrage spatial s'effectue de la même manière. Enfin, il est normal que le signal de sortie  $z[n]$  ait une amplitude deux fois moins élevée que le signal microphonique  $y_1[n]$ , car le motif de faisceau Figure 1.6 page 7 atténue de  $-6$  dB en basses fréquences pour un angle d'incidence de la source  $\theta = \pi/2$ .

La modélisation est vérifiée et semble cohérente. Il est à noter que le traitement n'a de sens que si les fréquences de la source sont dans la bande de traitement. L'avantage de l'interpolation est d'éviter les effets de bords, mais utilisée de cette manière, elle possède un inconvénient : elle permet à certaines fréquences non filtrées d'être présentes dans le signal de sortie. Enfin, la matrice  $\mathbf{H}$  obtenue ici n'est qu'une matrice possible : dans un souci d'optimisation, il serait nécessaire de définir objectivement les bornes  $k_i$ ,  $k_f$  ainsi que l'interpolation effectuée.

La modélisation du traitement différentiel du premier ordre étant terminée, il est possible de passer à l'expérimentation pour ainsi confronter les résultats obtenus dans ce chapitre à l'expérience.

# Chapitre 2

## Expérimentation

L'objectif de ce chapitre est dans un premier temps d'introduire l'antenne utilisée, pour ensuite pouvoir confronter la modélisation à l'expérience et enfin implémenter le traitement de l'antenne en temps réel.

### 2.1 Antenne xCORE

L'ensemble des expériences est réalisé à l'aide de l'antenne xCORE du fabricant XMOS [13] représentée Figure 2.1. Cette antenne circulaire est composée de 7 microphones de *modulation de densité d'impulsions* [9] et de quatre boutons. Le bouton A permet d'allumer l'antenne, les boutons B et C permettent respectivement d'augmenter et de diminuer son gain, tandis que le dernier bouton D n'est pas défini. En ayant préalablement installé les pilotes nécessaires, il suffit de connecter l'antenne à un ordinateur via un câble USB pour qu'elle soit reconnue comme un périphérique audio et utilisable instantanément.



FIGURE 2.1 – Photographie [13] de l'antenne de microphones xCORE

Souhaitant réaliser un traitement différentiel du premier ordre, uniquement trois microphones respectant la condition d'une antenne circulaire uniforme (1.2) sont nécessaires. Il est choisi arbitrairement pendant l'expérience de prendre dans l'ordre les microphones MIC1, MIC5 et MIC3 représentés Figure 2.1. Par souci de notation, ils sont respectivement notés  $M_1$ ,  $M_2$  et  $M_3$ , et leurs signaux microphoniques temporels  $y_1[n]$ ,  $y_2[n]$  et  $y_3[n]$ . De plus, le microphone central MIC0 est utilisé comme référence, et est noté  $M_0$ .

## 2.2 Traitement de l'antenne

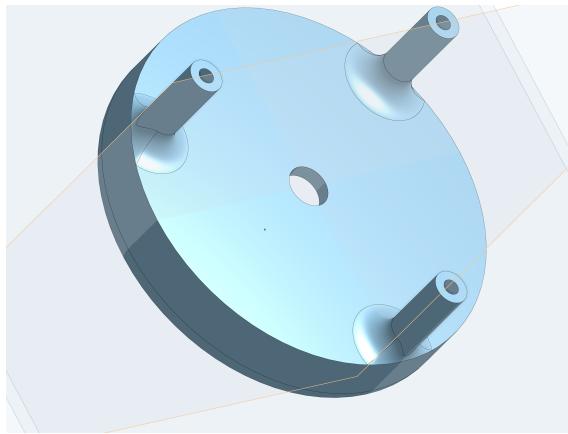
L'objectif de cette section est de définir l'ensemble des paramètres expérimentaux pour ensuite confronter les résultats de la modélisation en directivité et en traitement large bande.

Connaissant les angles  $\phi_m$  des microphones utilisés et la vitesse de propagation des ondes acoustiques dans l'air  $c \approx 342 \text{ m s}^{-1}$ , il suffit de mesurer directement le rayon  $R$  de l'antenne pour pouvoir déterminer indirectement la fréquence de repliement  $f_{rep}$ . Avec un pied à coulisse, on trouve

$$R \approx 4.32 \text{ cm} . \quad (2.1)$$

La distance entre deux microphones étant  $\|\mathbf{M}_2 - \mathbf{M}_1\|$ , équation (1.1), alors l'estimation de la fréquence de repliement  $f_{rep}$  de l'équation 1.10 page 7 est

$$f_{rep} \approx 2285 \text{ Hz} .$$

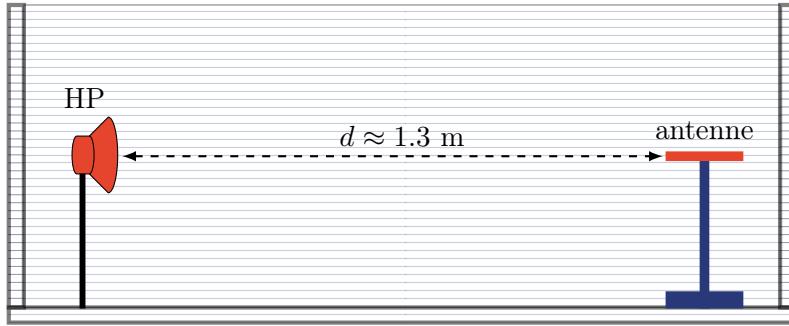


**FIGURE 2.2 – Modélisation 3D du support de l'antenne**

Il est maintenant possible de réaliser les mesures de directivité pour un motif de faisceau  $\theta_s$  orienté vers l'angle  $\phi_1$  du microphone  $M_1$  dans la chambre anéchoïque du LAUM. Les mesures sont réalisées en commun avec les auteurs du projet antennerie pour la détection d'arrivée d'une source avec l'algorithme SRP-PHAT, Juliette ALCARAZ et Pablo GIRAUD-GIRARD.

Il a été possible de concevoir et d'imprimer en 3D grâce à Mr. Alann RENAULT, Ingénieur d'Études au LAUM, un support détachable permettant de solidariser l'antenne à la table tournante Figure 2.2. Afin de simplifier cette mesure, la table tournante ET250-3D du fabricant Outline disponible en salle semi-anéchoïque est utilisée. Enfin, une interface graphique avec le langage de programmation Python a été réalisée par nos soins. Elle permet notamment l'acquisition des signaux microphoniques de l'antenne, la génération d'un signal d'excitation et l'asservissement de la table tournante via une carte Arduino Nano à l'aide d'un ordinateur. Dépassant le cadre de ce projet, une documentation détaillée ainsi que l'ensemble des scripts Python sont donnés en libre service sur la plateforme GitHub [1].

L'antenne étant reliée à la table tournante grâce au support, cette dernière est placée au milieu de la salle anéchoïque comme représenté Figure 2.3 page ci-contre. Un haut-parleur amplifié est placé à la même hauteur que l'antenne à une distance  $d \approx 1.3 \text{ m}$  permettant d'approximer la condition de champ lointain. La table est orientée pour que l'axe ( $OM_1$ ) de l'antenne coïncide avec l'axe du haut-parleur, puis initialisée à 0 degré pour un pas de 5 degrés. La quantification des signaux microphoniques étant codée en entier 16 bits signés [8] allant de  $-32768$  à  $32767$ ,



**FIGURE 2.3 – Représentation du dispositif expérimental vue de côté avec le haut-parleur (HP), l'antenne utilisée et la table tournante en bleu**

il paraît raisonnable pour éviter un écrêtage d'avoir une dynamique allant approximativement de -20000 à 20000. C'est pourquoi les gains de l'antenne et du haut-parleur sont réglés en même temps en générant un sinus de fréquence quelconque, vérifiant ainsi avec un signal microphonique que l'amplitude de la source ne soit pas trop élevée et qu'il n'y ait pas de non linéarités.

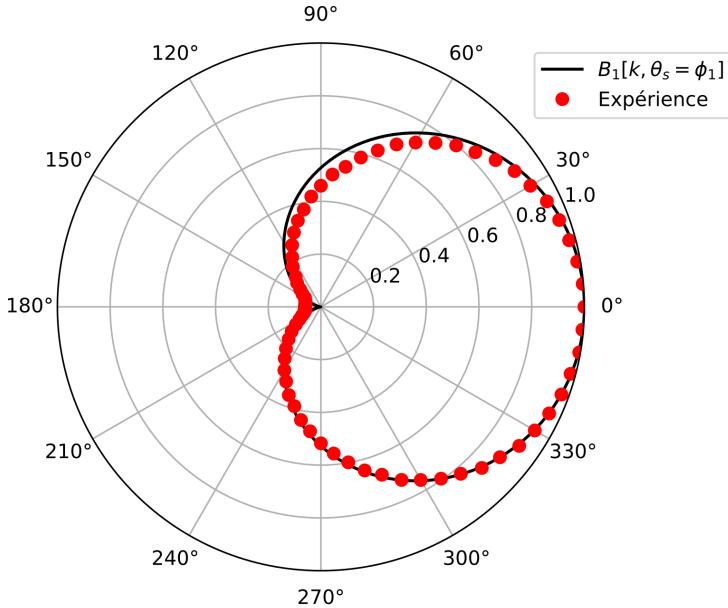
Le signal d'excitation  $s_{exc}[n]$  choisi est un sinus glissant exponentiel d'amplitude  $A$  quelconque et de la forme

$$s_{exc}[n] = A \cos \left( 2\pi f_{min} \frac{e^{\beta n T_s} - 1}{\beta} \right), \quad \text{où} \quad \beta = \frac{\log_2 f_{max} - \log_2 f_{min}}{n_{-1} T_s},$$

avec  $n_{-1}$  le dernier échantillon du vecteur temps. La fréquence minimale  $f_{min} = 60 \text{ Hz}$  est choisie en fonction de la gamme de fréquences du haut-parleur et la fréquence maximale  $f_{max} = 8 \text{ kHz}$  est choisie pour mettre en évidence la fréquence de repliement  $f_{rep}$ . La fréquence d'échantillonnage  $F_s = 44.1 \text{ kHz}$  et le temps d'acquisition de 5 s étant réglés, il est par suite possible de réaliser les mesures de directivité par pas de 5 degrés jusqu'à 360 degrés en enregistrant les signaux microphoniques à l'intérieur d'un fichier .wav différent pour chaque angle.

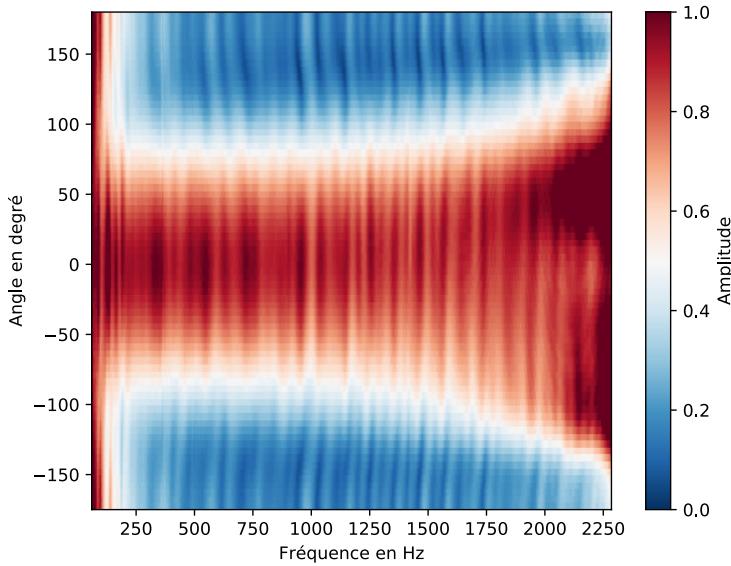
Pour éviter les problèmes de synchronisation entre l'enregistrement du signal source et l'enregistrement des signaux microphoniques, il est possible de choisir le microphone central de l'antenne comme référence et ainsi garder uniquement les voies des microphones  $M_0$ ,  $M_1$ ,  $M_2$  et  $M_3$  des fichiers audios de format .wav obtenus. Le traitement vu à la section 1.3 page 7 est appliqué aux trois microphones respectant la condition d'une antenne circulaire de  $f_{min}$  à  $f_{rep}$  dans le domaine fréquentiel. Puis, le résultat obtenu est divisé par la TFD du signal microphonique de référence pour obtenir la réponse en fréquence. A l'aide d'un script, le motif de faisceau est représenté pour une fréquence égale environ à 500 Hz Figure 2.4 page suivante en normalisant par le maximum sur l'ensemble des angles.

Le motif de faisceau correspond bien au motif de faisceau théorique souhaité pendant la modélisation section 1.2 page 4 en basses fréquences. Ses maximum et minimum correspondent respectivement aux angles  $\phi_1$  et  $\phi_1 + \pi$  et il est bien de forme cardioïde. Cependant, il n'est pas symétrique suivant l'axe ( $OM_1$ ). L'écart entre l'expérience et la modélisation provient probablement d'une erreur systématique : en effet, le haut-parleur n'était pas nécessairement exactement dans l'axe ( $OM_1$ ) de l'antenne au début de la mesure, et par suite le lobe principal du motif de faisceau est décalé. Il est à noter qu'en utilisant la table tournante, les erreurs aléatoires dues au changement d'angle par pas de 5 degrés sont supposées négligeables.



**FIGURE 2.4** – Comparaison du motif de faisceau théorique (en noir) et expérimental (en rouge)

La modélisation pour un traitement à une fréquence étant vérifiée, il est possible de tracer le module de la réponse en fréquence sur l'ensemble des angles Figure 2.5 en imposant une amplitude maximale égale à 1.



**FIGURE 2.5** – Angle  $\theta$  en fonction de la fréquence

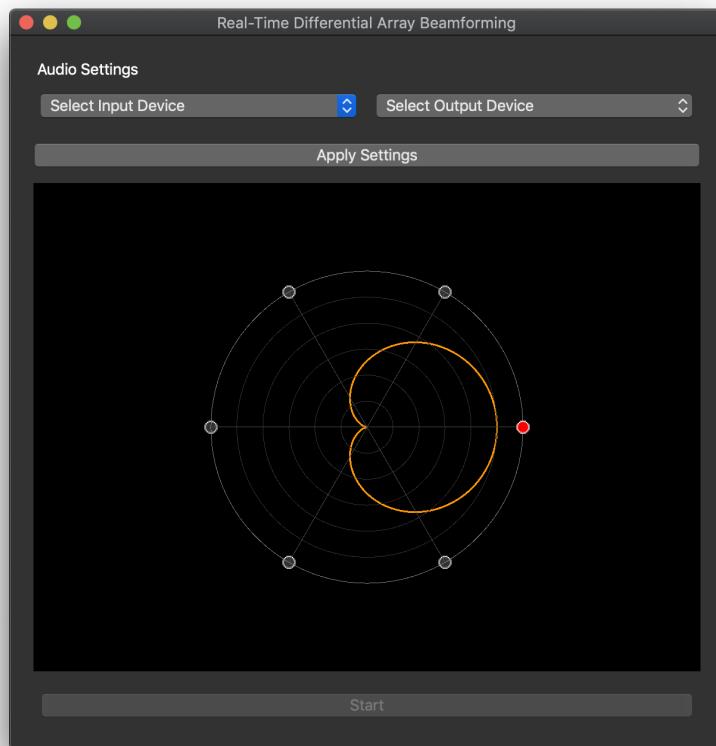
Cette représentation permet de vérifier que le traitement différentiel ne varie quasiment pas sur une large bande de fréquences. En basses fréquences, il existe une amplification quelque soit l'angle  $\theta$  jusqu'à environ 125 Hz, illustrant bien expérimentalement l'amplification du bruit avec la méthode différentielle utilisée. Il est à noter que les mesures étant réalisées en salle anéchoïque, leur confiance accordée est élevée : l'amplification du bruit provient bien de la méthode différentielle et non des conditions expérimentales. Il est de même possible de noter que, en hautes fréquences, le motif de faisceau atténue de moins en moins plus la fréquence se rapproche de la fréquence de repliement  $f_{rep}$ . Le motif de faisceau va jusqu'à amplifier et

ne plus respecter la contrainte de distorsion (1.5), primordiale dans cette étude pour pouvoir appliquer un traitement cohérent. De plus, la contrainte de symétrie (1.7) est globalement vérifiée quelque soit la fréquence dans la bande de fréquences. Finalement, à partir de l'équation (1.11), une estimation des bornes fréquentielles pour appliquer le traitement différentiel consiste à choisir  $k_i\Delta f \approx 250$  Hz en basses fréquences et  $k_f\Delta f \approx 1500$  Hz en hautes fréquences. Cela évite une amplification trop élevée du bruit ainsi qu'une trop grande variation du motif de faisceau.

Ainsi s'achève l'expérimentation se montrant concluante et vérifiant la modélisation précédemment réalisée. Il est par conséquent possible de réaliser une implémentation en temps réel de l'antenne dans la prochaine section afin de mettre en évidence une utilisation possible de l'antenne au quotidien dans des conditions réverbérantes avec du bruit.

### 2.3 Implémentation en temps réel

Pour réaliser le traitement en temps réel avec la méthode différentielle, il est nécessaire de calculer les réponses impulsionales  $RI_m[n]$  (Figure 1.9 page 9) à appliquer aux microphones  $M_1$ ,  $M_2$  et  $M_3$ . Le motif de faisceau est orienté à l'angle  $\phi_1$  du microphone  $M_1$  et le traitement s'effectue pour une bande de fréquences allant de 250 Hz à 1500 Hz, choisie par défaut en fonction de l'estimation donnée précédemment section 2.2 page 12. Un programme Python muni d'une interface graphique, représentée Figure 2.6, gère les calculs à effectuer en parallèle à l'aide d'un *multithreading* (filage multiple en français).



**FIGURE 2.6 – Capture d'écran de l'interface graphique réalisée**

Du point de vue de l'utilisateur, cette interface graphique permet de sélectionner une entrée et une sortie audio. Pour réaliser le traitement différentiel, il est cependant nécessaire que l'entrée soit l'antenne de microphones xCORE. La sortie peut être choisie par défaut avec un nombre de voies quelconque sous réserve qu'elle soit reliée à l'ordinateur utilisé. En cliquant sur les boutons *Apply Settings* puis *Start*, il est alors possible d'entendre la restitution du signal de sortie après filtrage. Enfin, le motif de faisceau étant représenté, il est de plus possible de modifier son orientation en cliquant sur la position voulue.

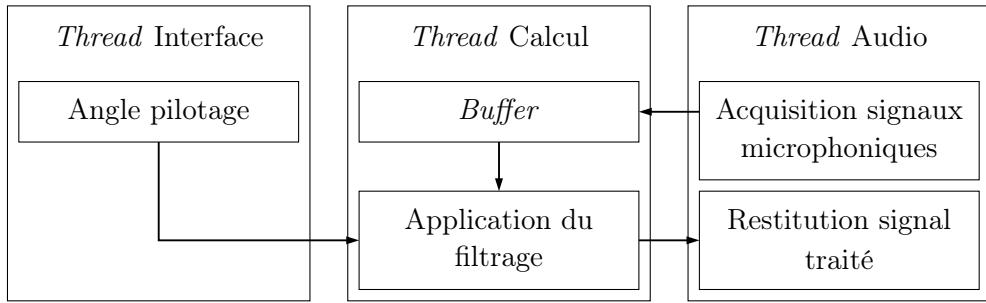


FIGURE 2.7 – Schéma représentatif du traitement en temps réel

En interne, l'interface est réalisée à l'aide de la bibliothèque Python *PyQt5*. L'ensemble des scripts permettant de la réaliser peut être trouvé en libre service sur la plateforme GitHub [2].

Comme représentés Figure 2.7, trois *threads* et un *buffer*, respectivement fils et tampon en français, sont initialisés au démarrage de l'application. Le rôle du *thread* Audio est de gérer automatiquement l'acquisition des signaux microphoniques provenant de l'entrée audio, ici l'antenne xCORE. L'ensemble des calculs du traitement est réalisé avec le *thread* Calcul à partir de la relation (1.12) section 1.3 page 7. Son rôle est d'appliquer le traitement différentiel pour chaque voie microphonique sur l'ensemble du *buffer* à l'aide de l'algorithme *fftconvolve* de la bibliothèque *Scipy* et des réponses impulsionales  $RI_m[n]$ . Une fois le filtrage réalisé, le *thread* Audio restitue le signal de sortie  $z[n]$  en sortie audio. De plus, le *thread* Interface permet de gérer le changement dans l'orientation du motif de faisceau. Enfin, les bornes du traitement différentiel ne peuvent être changées et sont réglées par défaut.

Une vidéo `demo.mp4` montrant l'interface et l'utilisation en temps réel de l'antenne de microphones xCORE dans une salle quelconque est disponible en libre service sur la plateforme GitHub [2] dans le dossier `Realtime_processing`. Il est possible de constater de manière qualitative que dans le cas d'un signal vocal, le signal de sortie est plus ou moins atténué selon l'angle d'incidence de la source et selon le motif de faisceau cible, illustrant bien une utilisation possible de l'antenne dans des conditions de salle réverbérantes avec du bruit.

Il est à noter que le terme "temps réel" avec le langage de programmation Python est un abus de langage. En effet, étant un langage interprété, il ne permet pas l'exécution de deux calculs instantanément en parallèles par mesure de sécurité, évitant ainsi d'accéder et de changer l'information contenue dans une case de mémoire au même moment. Cette sécurité s'appelle *Global Interpreter Lock (GIL)* [10] et permet la gestion automatique de la répartition des calculs à effectuer de manière *séquentielle*. Par suite, pour réaliser un traitement en temps réel de manière strictement rigoureuse, il serait nécessaire d'utiliser un langage de programmation compilé à typage statique comme le langage C par exemple.

La principale difficulté pouvant être rencontrée pour mener à bien le traitement en temps

réel est inhérente à la taille du *buffer*. En effet, il est possible que la gestion *séquentielle* des calculs par le GIL entraîne une instabilité dans le transfert des données entre l'antenne xCORE et l'interface. Il peut être vu rapidement avec de l'essai erreur qu'une taille trop petite entraîne un débordement dans le flux des données pour l'entrée audio, soit l'erreur

`IOError: [Errno Input overflowed] -9981 .`

Il est possible de choisir une taille de *buffer* plus large pour palier à ce problème.

Plusieurs choix sont possibles pour améliorer le travail réalisé par nos soins. En gardant cette méthode de temps réel, il est possible de l'optimiser pour réduire la complexité des calculs. Enfin, il est également possible de partitionner les réponses impulsionales  $RI_m[n]$  de manière uniforme ou non [3] pour avoir un délai de traitement plus faible.

# Conclusion

Ainsi, ce projet a permis dans un premier temps d'expliciter dans le domaine fréquentiel la méthode décrite par Jacob BENESTY dans son livre *Design of Circular Differential Microphone Arrays* [6] pour un traitement différentiel d'ordre 1 pour une et plusieurs fréquences, puis de montrer avec l'expérience que l'antenne peut être utilisée en temps réel et au quotidien, c'est à dire dans des conditions de salles réverbérantes avec du bruit.

Il a été vu que la principale contrainte pour réaliser un traitement différentiel est le rayon  $R$  de l'antenne circulaire limitant la bande fréquentielle  $f_{exp}$  du traitement pour cause de repliement spatial. Plusieurs choix sont néanmoins possibles pour permettre un traitement sur toute la bande de fréquences d'une voie humaine, allant approximativement de 0.3 kHz à 4 kHz. Une solution triviale consisterait à changer l'antenne et en choisir une de rayon suffisamment petit pour que la fréquence de repliement (1.10) soit au moins égale à 4 kHz. En conservant l'antenne utilisée ici, un autre choix serait de réaliser un traitement avec plus de microphones, diminuant ainsi la distance  $\delta$  et par suite augmentant la fréquence de repliement. Cela impliquerait une augmentation dans l'ordre  $N$  de la méthode différentielle et permettrait un avantage non négligeable : avoir un motif de faisceau plus directif que celui utilisé dans cette étude. Cependant, l'amplification du bruit en basses fréquences sera plus élevée.

L'avantage de cette méthode différentielle est qu'elle est flexible, changer de motif de faisceau consistant uniquement, avec trois microphones, à changer la relation (1.6) ; robuste, dans le sens où elle permet une utilisation de l'antenne au quotidien ; et simple à mettre en place avec le langage de programmation Python. Son seul inconvénient vient des réponses impulsionnelles, comme il est nécessaire de passer dans le domaine fréquentiel pour revenir dans le domaine temporel et étant non causales d'ajouter un terme de déphasage. Le plus simple pour une communication audio en temps réel serait d'utiliser directement des réponses impulsionnelles causales, et c'est pourquoi il serait intéressant dans le cadre d'un projet futur d'utiliser la méthode décrite par Jacob BENESTY [4], cette fois-ci dans le domaine temporel.

## Annexe A

### Script Python : differentialarray.py

```
1  from scipy.interpolate import interp1d
2  import scipy.signal as sg
3  import numpy.fft as fft
4  import numpy as np
5
6  """
7      Bibliothèque Python créée pour le traitement différentiel d'une antenne circulaire uniforme.
8      Réalisée par Tom Aubier et Raphaël Dumas.
9  """
10
11
12 def index(array, value):
13     """
14         Fonction permettant de trouver l'indice le plus proche à l'intérieur d'un vecteur en
15         fonction de la valeur.
16         Entrée :
17             array : 1D np.array
18             value : int
19         Sortie :
20             index : int
21     """
22     index = 0
23     while array[index] < value: index += 1
24     return index
25
26 class Medium:
27     """
28         Classe permettant de modéliser le milieu. Dans le cadre de ce projet, seule la vitesse de
29         propagation C dans le milieu est définie. Mais cela permettra dans le cadre d'un projet
30         futur de prendre en compte le fait que le milieu soit non homogène et/ou anisotrope.
31     """
32     C = 342
33
34 class PlaneWave(Medium):
35     """
36         Classe permettant de créer une onde plane arrivant sur l'antenne.
37         Entrée :
38             amp : float, amplitude de l'onde plane
39             f : float, fréquence de l'onde plane
40             ang : float, angle de l'onde plane
41     """
42     def __init__(self, amp=1, f=500, ang=0):
43         super(PlaneWave, self).__init__()
44         self.amp, self.f, self.ang = amp, f, ang
```

---

```

43     self.w = 2 * np.pi * self.f
44     self.k = self.w / Medium.C * np.array([np.cos(self.ang), np.sin(self.ang)])
45
46 def static_field(self, x, y):
47     """
48         Crée le champ de pression de l'onde plane à l'instant t=0.
49         Compatible uniquement avec x et y np.meshgrid
50         Entrée :
51             x : 2D np.array, matrice position suivant x
52             y : 2D np.array, matrice position suivant y
53         Sortie :
54             field : 2D np.array, champ de pression complexe
55     """
56     field = np.zeros((x.shape[0], x.shape[0]), dtype='cfloat')
57     for ii in range(x.shape[0]):
58         for jj in range(x.shape[0]):
59             field[ii, jj] = np.exp(1j*np.dot(self.k, [x[ii, jj], y[ii, jj]]))
60     return field
61
62 def field(self, x, y, t):
63     """
64         Crée le champ de pression de l'onde plane à tout instant aux points (x, y).
65         Entrée :
66             x : 1D np.array, vecteur position suivant x
67             y : 1D np.array, vecteur position suivant y
68             t : 1D np.array, vecteur temps
69         Sortie :
70             field : 1D np.array, champ de pression complexe
71     """
72     time = np.exp(1j*self.w*t)
73     temp = self.amp*np.exp(1j*np.dot(self.k, [x, y]))
74     field = np.dot(temp.reshape(x.size, 1), time.reshape(1, t.size))
75     return field
76
77
78 class CircularArray:
79     """
80         Classe permettant de créer une antenne circulaire uniforme.
81         Entrée :
82             M : int, nombre de microphones
83             R : float, fréquence de l'onde plane
84     """
85
86     def __init__(self, M=3, R=4.32e-2):
87         super(CircularArray, self).__init__()
88         self.M, self.R = M, R
89         self.phi = np.arange(0, 2 * np.pi, 2 * np.pi / self.M)
90         self.dphi = self.phi[1] - self.phi[0]
91
92     def coordinates(self):
93         """
94             Retourne les coordonnées cartésiennes des microphones.
95             Sortie :
96                 x, y : tuple, positions des microphones suivant x et y
97         """
98         x, y = self.R * np.cos(self.phi), self.R * np.sin(self.phi)
99         return x, y
100
101 class DifferentialArray(CircularArray, Medium):
102     """

```

```

103     Classe permettant de créer un traitement différentiel d'ordre 1 pour un motif de faisceau
104     ↵ cardioïde.
105     Entrée :
106         steering_mic : int, microphone pilotant le motif de faisceau
107         Fs : int, fréquence d'échantillonnage
108         Ntfd : int, nombre de points de la TFD
109         freqBand : tuple, bande de fréquence du traitement à appliquer
110         interp : tuple, si vrai, interpolation effectuée sur le nombre de points donné
111     """
112     def __init__(self, steering_mic=1, Fs=8000, Ntfd=2**13, freqBand=(500, 700), interp=(True,
113     ↵ 10)):
114         super(DifferentialArray, self).__init__()
115         self.Fs, self.Ntfd = Fs, Ntfd
116         self.Fk = np.arange(self.Ntfd)*self.Fs / self.Ntfd
117         self.tau = self.R/Medium.C
118         self.steering_mic = steering_mic - 1
119         self.interp, self.pt_interp = interp
120         self.ii_min, self.ii_max = index(self.Fk, freqBand[0]), index(self.Fk, freqBand[1])
121
122     def steering_vector(self, angle, f):
123         """
124             Retourne le soi-disant vecteur de pilotage de l'antenne pour des angles et une fréquence
125             donnée.
126             Entrée :
127                 angle : 1D np.array, vecteur angulaire en radian
128                 f : float, fréquence
129             Sortie :
130                 d : 1D np.array, vecteur de pilotage
131         """
132         d = [np.exp(-1j*2*np.pi*f*self.tau*np.cos(angle - phi)) for phi in self.phi]
133         return np.array(d, dtype='cfloat')
134
135     def filter_vector(self, f):
136         """
137             Retourne les gains complexes à appliquer à chaque microphone pour une fréquence f.
138             Entrée :
139                 f : float, fréquence
140             Sortie :
141                 h : 1D np.array, vecteur des filtres
142
143         A, b = np.zeros((self.M, self.M), dtype='cfloat'), np.zeros(self.M, dtype='cfloat')
144         b[0] = 1
145         A[-1] = np.roll(np.array([0, 1, -1]), dtype='cfloat'), self.steering_mic, None)
146         theta_c = np.array([0, np.pi]) + self.dphi*self.steering_mic
147         for ii, angle in enumerate(theta_c):
148             A[ii] = self.steering_vector(angle, f)
149         h = np.linalg.solve(A, b)
150         return h
151
152     def filter_matrix(self):
153         """
154             Retourne la matrice des gains complexes à appliquer à chaque microphone sur toute la
155             bande de fréquences freqBand.
156             Sortie :
157                 H : 2D np.array, matrice des filtres
158
159         H = np.zeros((self.M, self.Fk.size), dtype='cfloat')
160         for ii, freq in enumerate(self.Fk[self.ii_min:self.ii_max], self.ii_min):
161             H[:, ii] = self.filter_vector(freq)
162         if self.interp:
163             H = self.interpolation(H)

```

---

```

160     for ii, freq in enumerate(self.Fk[1:int(self.Fk.size/2)], 1):
161         H[:, -ii] = np.conj(H[:, ii])
162     return H
163
164     def interpolation(self, H):
165         """
166             Modifie la matrice des gains H pour réaliser une interpolation sur le nombre de points
167             donné.
168             Entrée :
169                 H : 2D np.array, matrice des filtres
170             Sortie :
171                 H : 2D np.array, matrice des filtres avec interpolation
172             """
173     del_bef = np.arange(self.ii_min - self.pt_interp, self.ii_min)
174     del_aft = np.arange(self.ii_max, self.ii_max + self.pt_interp)
175     del_elt = np.stack([del_bef, del_aft])
176     Fk_inter = np.delete(self.Fk, del_elt)
177     for mic in range(self.M):
178         H_inter = np.delete(H[mic], del_elt)
179         f = interp1d(Fk_inter, H_inter, kind='cubic')
180         H[mic, :int(self.Ntf/2)-1] = f(self.Fk)[:int(self.Ntf/2)-1]
181     return H
182
183     def beampattern(self, angle, f):
184         """
185             Retourne le motif de faisceau pour tous les angles à une fréquence f.
186             Entrée :
187                 theta : 1D np.array, vecteur angulaire en radian
188                 f : float, fréquence
189             Sortie :
190                 beam : 1D np.array, motif de faisceau
191             """
192     beam = np.dot(self.filter_vector(f), self.steering_vector(angle, f))
193     return np.abs(beam)
194
195     def impulse_responses(self):
196         """
197             Calcul les réponses impulsionales non causales en ajoutant un terme de déphasage.
198             Sortie :
199                 h : 2D np.array, matrice des réponses impulsionales
200             """
201     h = np.real(fft.ifft(np.conj(self.filter_matrix())))
202     h = np.roll(h, int(h.size/2), axis=1)
203     return h
204
205     def dma_output(self, signals):
206         """
207             Calcul le signal de sortie z du traitement en convoluant les signaux avec les réponses
208             impulsionales en utilisant l'algorithme fftconvolve de Scipy.
209             Entrée :
210                 signals : 2D np.array, matrice des signaux microphoniques
211             Sortie :
212                 z : 1D np.array, signal de sortie du traitement
213             """
214     z = sg.fftconvolve(signals, self.impulse_responses(), mode='same', axes=1)
215     return np.sum(z, axis=0)

```

# Bibliographie

- [1] Tom AUBIER et Raphaël DUMAS. *Acoustic Directivity Measures*. Page consultée le 9 mai 2019. 2019. URL : <https://github.com/Tomaubier/adm> (cf. page 12).
- [2] Tom AUBIER et Raphaël DUMAS. *Application Python differentialGUI.py*. Page consultée le 14 mai 2019. 2019. URL : [https://github.com/Tomaubier/projetAntenne/blob/master/Realtime\\_processing/differentialGUI.py](https://github.com/Tomaubier/projetAntenne/blob/master/Realtime_processing/differentialGUI.py) (cf. page 16).
- [3] Eric BATTEMBERG et Rimas AVIZIENIS. « Implementing real-time partitioned convolution algorithms on conventional operating systems ». In : (sept. 2011). Page consultée le 30 avril 2019, page 8. URL : <https://www.researchgate.net/publication/228842427> (cf. page 17).
- [4] Yaakov BUCHRIS, Israel COHEN et Jacob BENESTY. « On the design of time-domain differential microphone arrays ». In : *Applied Acoustics* 148 (2019). Page consultée le 30 avril 2019, pages 212-222. ISSN : 0003-682X. DOI : <https://doi.org/10.1016/j.apacoust.2018.12.013>. URL : <http://www.sciencedirect.com/science/article/pii/S0003682X18304870> (cf. page 18).
- [5] Potel CATHERINE et Bruneau MICHEL. *Acoustique générale - équations différentielles et intégrales, solutions en milieux fluides et solides, applications*. 2<sup>e</sup> édition. Technosup. Elipses, 2016. ISBN : 9782340014879 (cf. page 2).
- [6] Israel Cohen (auth.) JACOB BENESTY Jingdong Chen. *Design of Circular Differential Microphone Arrays*. 1<sup>re</sup> édition. Springer Topics in Signal Processing 12. Springer International Publishing, 2015. ISBN : 978-3-319-14841-0, 978-3-319-14842-7 (cf. pages 1, 4, 5, 18).
- [7] Simon PLOUFFE. *The reflection of light rays in a cup of coffee*. Page consultée le 13 mai 2019. 2019. URL : [http://xahlee.info/SpecialPlaneCurves\\_dir/Cardioid\\_dir/\\_p/LightsRaysReflections.pdf](http://xahlee.info/SpecialPlaneCurves_dir/Cardioid_dir/_p/LightsRaysReflections.pdf) (cf. page 5).
- [8] WIKIPEDIA. *Entier (informatique)*. [https://www.wikiwand.com/fr/Entier\\_\(informatique\)](https://www.wikiwand.com/fr/Entier_(informatique)). Page consultée le 30 avril 2019 (cf. page 12).
- [9] WIKIPEDIA. *Pulse Density Modulation*. [https://www.wikiwand.com/en/Pulse-density\\_modulation](https://www.wikiwand.com/en/Pulse-density_modulation). Page consultée le 15 mars 2019 (cf. page 11).
- [10] WIKIPEDIA CONTRIBUTORS. *Global Interpreter Lock — Wikipedia, The Free Encyclopedia*. Page consultée le 14 mai 2019. 2019. URL : <https://wiki.python.org/moin/GlobalInterpreterLock> (cf. page 16).
- [11] WIKIPEDIA CONTRIBUTORS. *Limaçon — Wikipedia, The Free Encyclopedia*. Page consultée le 13 mai 2019. 2019. URL : <https://www.wikiwand.com/en/Lima%C3%A7on> (cf. page 5).
- [12] WIKIPEDIA CONTRIBUTORS. *Mandelbrot set — Wikipedia, The Free Encyclopedia*. Page consultée le 13 mai 2019. 2019. URL : [https://www.wikiwand.com/en/Mandelbrot\\_set](https://www.wikiwand.com/en/Mandelbrot_set) (cf. page 5).
- [13] XMOS. *Photo de l'antenne circulaire xCORE*. <https://www.xmos.com/developer/kits/xcore-array-microphone>. Page consultée le 15 mars 2019 (cf. page 11).