



Technologie de l'Informatique

Avenue du Ciseau 15

1348 Ottignies

---

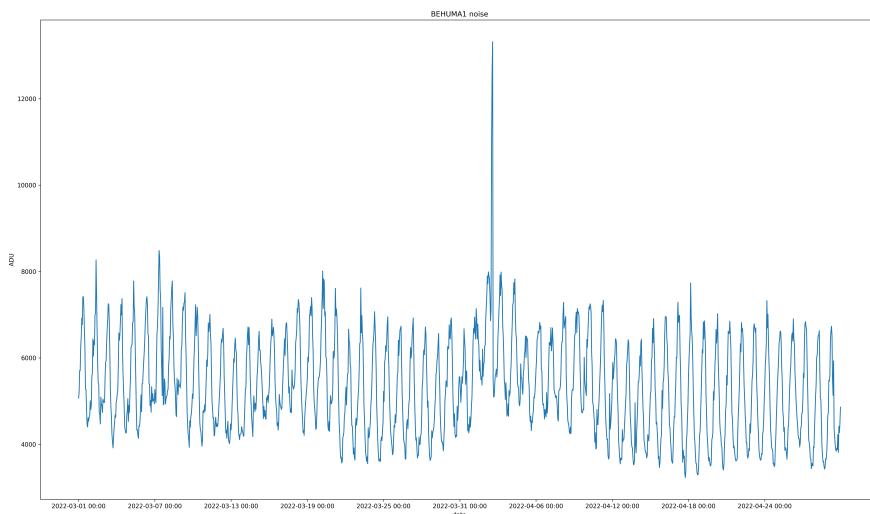
## Monitoring des données BRAMS et déttection automatique des échos de météore

---

*Miguel Antoons*

Rapporteur

*Monsieur Arnaud Dewulf*



2021-2022

## **Remerciements**

*Je tiens tout d'abord à remercier toutes les personnes qui m'ont aidé à réaliser ce projet de fin d'études.*

*En commençant par mon professeur rapporteur, à qui j'ai pu poser mes questions en cas de besoin et qui s'est assuré que tout se passe bien tout au long du projet.*

*Ensuite, je voudrai remercier Mr Hervé Lamy pour avoir proposé ce sujet de fin d'études, mais également pour m'avoir expliqué, de façon claire et précise, toutes les notions qui nécessitaient des explications.*

*Je tiens également à remercier Mrs Antoine Calegaro et Michel Anciaux, qui m'ont guidé quand c'était nécessaire et qui m'ont conseillé durant le projet de fin d'études.*

*Enfin, je voudrais exprimer ma reconnaissance envers toutes les personnes qui m'ont conseillé sur, et ont relu ce rapport de projet de fin d'études.*

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Détection des météores par ondes radios</b>	<b>4</b>
2.1	La Réflexion Spéculaire . . . . .	4
<b>3</b>	<b>Le projet BRAMS</b>	<b>5</b>
3.1	L'Émetteur . . . . .	5
3.2	Les Stations de Réception . . . . .	5
3.3	Les Données BRAMS . . . . .	6
3.4	L'Archive BRAMS . . . . .	9
<b>4</b>	<b>Problématique liée à l'étude des données BRAMS</b>	<b>11</b>
<b>5</b>	<b>Méthodologie</b>	<b>12</b>
<b>6</b>	<b>Technologies utilisées</b>	<b>13</b>
6.1	Python . . . . .	13
6.1.1	Librairies Utilisées . . . . .	13
6.2	MariaDB . . . . .	16
<b>7</b>	<b>Lecture des Fichiers BRAMS</b>	<b>17</b>
7.1	Fonctionnement du Module . . . . .	17
7.1.1	La méthode FFT . . . . .	19
<b>8</b>	<b>Monitoring des Données BRAMS</b>	<b>21</b>
8.1	Fonctionnement du Programme . . . . .	21
8.2	Résultats . . . . .	23
<b>9</b>	<b>Détection des Météores</b>	<b>25</b>
9.1	Analyse . . . . .	25
9.2	Fonctionnement du Programme . . . . .	28
<b>10</b>	<b>Pistes d'Améliorations</b>	<b>33</b>
<b>11</b>	<b>Conclusion</b>	<b>34</b>
<b>12</b>	<b>Annexes</b>	<b>38</b>

# 1 Introduction

Chaque jour, des milliers d'objets passent tout près de l'atmosphère terrestre. Parmi ces objets, on retrouve les météoroïdes : un corps pierreux ou métallique d'une largeur pouvant varier de quelques millimètres à un mètre. Un météoroïde, une fois rentré dans l'atmosphère, devient un météore et peut créer un phénomène lumineux connu sous le nom de "Étoile filante". Contrairement à ce que l'on peut penser, un météoroïde qui rentre dans l'atmosphère terrestre est un événement qui se passe des milliers de fois par jour.

L'étude de ces météores permet de retrouver différentes informations telles que la masse ou encore la trajectoire de ceux-ci. Mais, afin de pouvoir les étudier, il est nécessaire de les détecter avant. Actuellement, différentes techniques existent pour détecter des météores dans l'atmosphère.

## 2 Détection des météores par ondes radios

### 2.1 La Réflexion Spéculaire

Pour comprendre le fonctionnement du réseau BRAMS, la première chose à savoir est : lorsqu'un météore passe dans l'atmosphère, il laisse derrière lui une trainée ionisée. Cette trainée à la propriété de réfléchir les ondes radio. Dans la plus grande partie des cas cette réflexion se fait en un seul point le long de la trajectoire, phénomène qui s'appelle la réflexion spéculaire (illustré à la figure 3). La position de ce point, nommé le point de réflexion spéculaire, dépend notamment du positionnement de l'émetteur, du positionnement du récepteur et de la trajectoire du météore.

Pour exploiter le phénomène de la réflexion spéculaire et réussir à détecter les météores à l'aide d'ondes radios, le réseau BRAMS fonctionne de la façon suivante :

1. Un émetteur, situé à Dourbes, transmet de façon continue un signal à une fréquence de 49.97 MHz. Ce signal est émis en direction du ciel et peut être réfléchi sur des trainées ionisées dans le sillage des météores.
2. Lorsque le signal est réfléchi, il peut être détecté par une ou plusieurs stations réceptrices faisant partie du réseau BRAMS. Un signal calibreur est alors additionné au signal venant du ciel. Ce signal est injecté à 49.9705 MHz, c'est-à-dire 500 Hz plus haut que le signal direct. Il dispose d'une amplitude fixe et sert de référence d'amplitude pour le reste du signal. La station réceptrice décale ensuite le signal direct de 49.97 MHz vers une fréquence de 1 kHz.
3. Ensuite, la station réceptrice enregistre l'ensemble du signal dans un fichier audio de type WAV. Le signal est échantillonné à une fréquence de 5512,5 Hz pour les stations ICOM et 6048 Hz pour les stations RSP2. Si tout se passe bien, un fichier est généré toutes les cinq minutes et chaque fichier devrait commencer et terminer à un temps prédéfini (par exemple : 16 h 00 à 16 h 05, 16 h 05 à 16 h 10, etc.).
4. Tous les fichiers générés par les stations réceptrices sont envoyés à intervalle régulier aux serveurs utilisés pour le projet BRAMS par internet.
5. Une fois sur le serveur, les fichiers WAV sont archivés, manipulés et étudiés par les scientifiques du projet BRAMS afin d'en extraire les informations utiles.

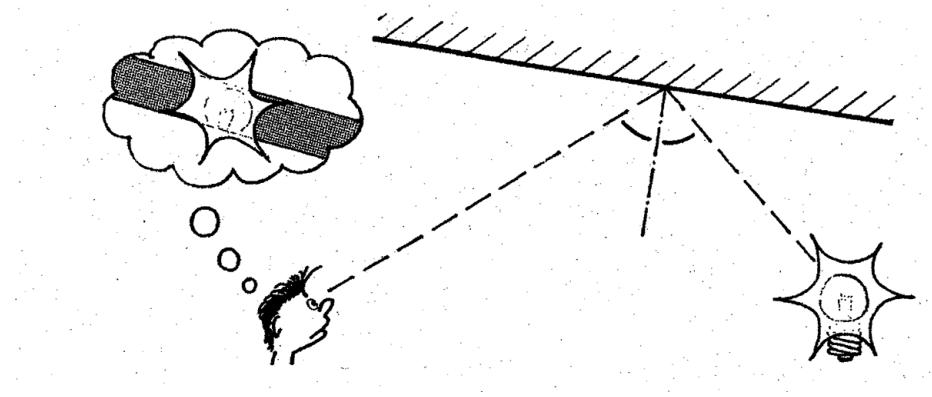


FIGURE 1 – Représentation de la réflexion spéculaire.

### 3 Le projet BRAMS

Lancé en 2010 par Monsieur Hervé Lamy à l’Institut Royal d’Aéronomie Spatiale de Belgique, le projet BRAMS (Belgian RAdio Meteor Stations) a pour but de collecter et stocker des données relatives à des objets rentrant dans l’atmosphère belge et, plus précisément, des météores. Ces données pourront ensuite être analysés afin de retrouver la trajectoire, la vitesse ou encore la masse d’un ou de plusieurs météores.

#### 3.1 L’Émetteur

#### 3.2 Les Stations de Réception

Afin d’accomplir ce but, le projet BRAMS dispose d’un réseau de stations nommé le réseau BRAMS. Ce réseau est composé d’un ensemble de quarante-deux stations émettrices, situées en Belgique ou dans les pays avoisinants, et d’un émetteur dédié situé à Dourbes, dans le sud de la Belgique.

Il existe 2 types de station de réception :

- Il y a d’abord les stations avec des récepteurs Icom IC-R75, que j’appellerai les récepteurs ICOM dans ce document. Ce sont les premières stations réceptrices mises en service pour le réseau BRAMS. Actuellement, ces stations ne sont plus utilisées suite à l’arrêt de la commercialisation du récepteur Icom IC-R75. De plus, une variation de

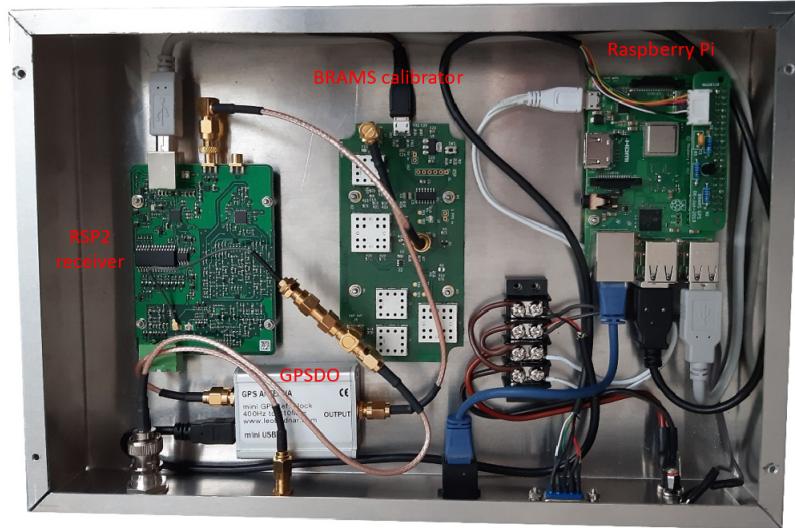


FIGURE 2 – Station RSP 2 du réseau BRAMS

température pouvait causer une légère déstabilisation en fréquence.

- C'est alors que les stations utilisant le récepteur RSP 2 ont été développés. Ces stations n'éliminaient pas seulement en grande partie les problèmes des anciennes stations, mais sont également plus compacts et plus faciles à installer. Une image des stations RSP 2 peut être trouvé à la figure 1.

### 3.3 Les Données BRAMS

Un fichier WAV venant d'une station réceptrice contient donc le signal capté par l'antenne ensemble avec un signal calibrateur. Il est composé d'une seule piste audio, échantillonnée à une fréquence de 5512.5 Hz ou 6048 Hz. Cette fréquence permet d'enregistrer des données dans une bande de fréquences allant jusqu'à 2756.25 Hz ou 3024 Hz (ou la moitié de la fréquence d'échantillonnage), selon le théorème de Nyquist. Sachant que les échos de météores apparaissent typiquement dans une bande de fréquence de 100 Hz autour du signal de l'émetteur à 1000 Hz, cette bande de fréquence couvre l'ensemble des signaux utiles à l'étude des météores.

À chaque fichier WAV est rajouté un bloc de données (data chunk) conçu

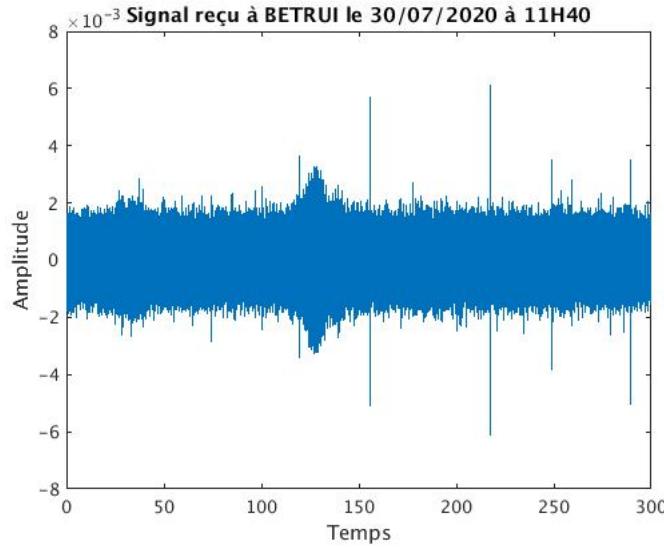


FIGURE 3 – Données brutes venant d'un fichier audio d'une station réceptrice.

pour faciliter l'étude des données. Dans ce bloc, on retrouve quelques informations relatives à la station de réception, la station émettrice, le signal GPS<sup>1</sup> ou encore, la fréquence d'échantillonnage. Ce bloc de données se trouve entre le bloc FMT et le bloc de données.

Ces fichiers audio, comme montrée à la figure 4, sont très difficiles à interpréter. Ils sont composés de bruits et de parasites sur l'entièreté de leur bande de fréquence. Par contre, lorsque l'on calcule le spectrogramme du fichier WAV, les données deviennent beaucoup plus simples à lire.

Un spectrogramme est une représentation différente des données contenues dans un fichier audio. Au lieu de représenter la puissance sur l'ordonnée et le temps sur l'abscisse, un spectrogramme représente la répartition des fréquences au cours du temps. Il contient donc trois axes :

- Sur l'abscisse on retrouve toujours le temps.
- Sur l'ordonnée on retrouve la fréquence.

---

1. Global Positioning System

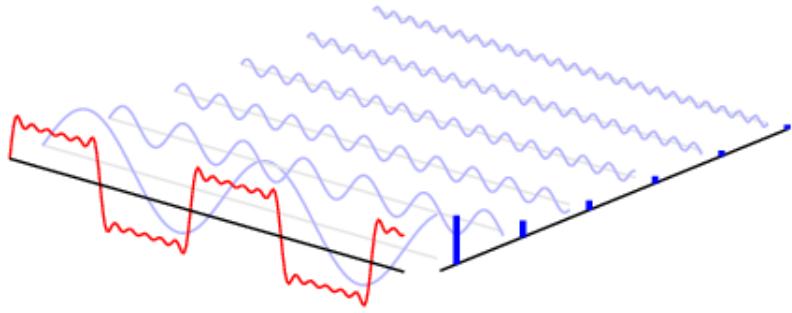


FIGURE 4 – Représentation visuelle de la Transformée de Fourier.

- Sur la cote, on retrouve la puissance. Cet axe est souvent représenté par un code couleur.

Pour générer un spectrogramme, il faut calculer un ensemble de Transformées de Fourier. La Transformée de Fourier permet de représenter la répartition de puissance entre les fréquences contenues dans un signal ou une partie d'un signal temporel. Elle produit donc ce qu'on appelle le spectre du signal. Elle se calcule sur un nombre quelconque d'échantillons qui se suivent dans un signal temporel. Dans l'informatique, elle est souvent calculée à l'aide de la FFT<sup>2</sup>. C'est un algorithme qui permet de calculer une Transformée de Fourier plus rapidement, mais qui exige un nombre d'échantillons  $2^n$ .

Sur la figure 5, on peut voir en rouge un signal temporel et le spectre de ce même signal en bleu. Les signaux en mauve claire sont les signaux à une fréquence, dont le signal rouge est composé.

Dans le cas des spectrogrammes pour le projet BRAMS, les FFT contenus dans un spectrogramme sont calculés sur 16384 échantillons. Si on suppose qu'un fichier WAV venant d'une station réceptrice dure en moyenne trois cents secondes (ou cinq minutes), un spectrogramme est composé d'environ 101 spectres pour les stations ICOM et 111 pour les stations RSP2. Ce nombre est obtenu avec la formule ci-dessous, où  $F_s$  est la fréquence d'échantillonnage,  $T$  la durée en secondes du signal et  $n_{fft}$  le nombre d'échantillons

---

2. Faste Fourier transform

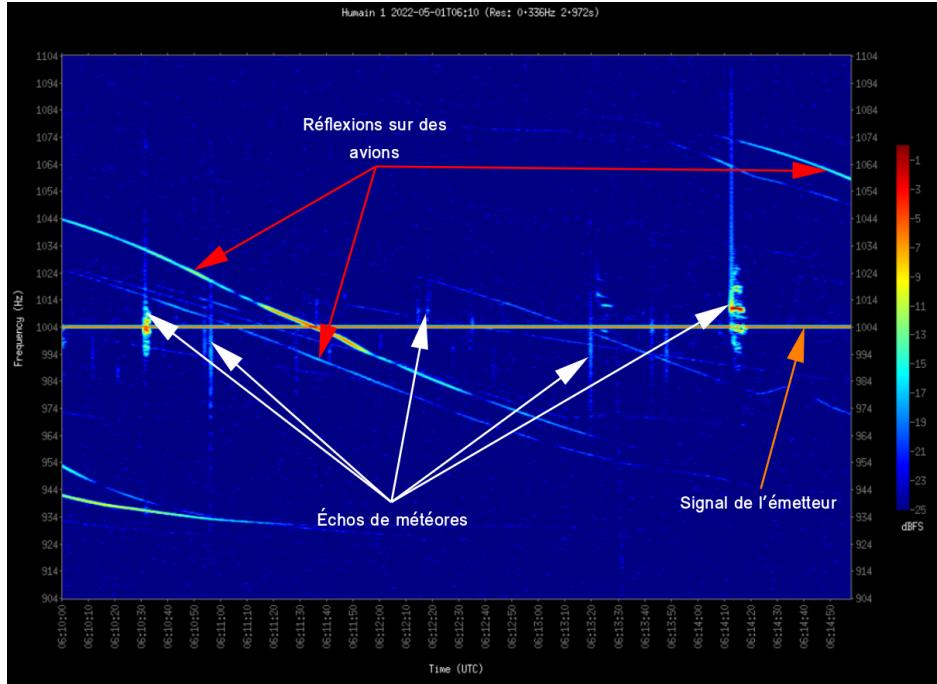


FIGURE 5 – Exemple d'un spectrogramme du projet BRAMS.

utilisés pour générer un spectre.

$$\frac{Fs * T}{nfft}$$

Le spectrogramme généré aura une résolution fréquentielle, donnée par la formule  $Fs/nfft$ , de 0.34 Hz pour les stations ICOM et 0.37 Hz pour les stations RSP2. Sa résolution temporelle, donnée par la formule  $nfft/Fs$ , est de 2.97 secondes pour les stations ICOM et de 2.7 secondes pour les stations RSP2. Un exemple de spectrogramme venant d'une station de réception est affiché à la figure 6.

### 3.4 L'Archive BRAMS

Quand les fichiers venant des stations de réception sont archivés, ils sont placés dans une structure de répertoires spécifique. Cette structure dépend de la date de début d'enregistrement des données contenues dans le fichier.

Dans la première couche de l'archive, on retrouve un répertoire par station BRAMS. Chaque nom de répertoire est le code de location de la station dont elle contient les données. Ensuite, on retrouve un nombre de répertoires, chacun contenant les fichiers d'une année spécifique. Le nom de ces répertoires est donc l'année des données qu'elles contiennent. Si on rentre dans le répertoire d'une année, on se retrouve dans la troisième couche de l'archive BRAMS. Ici, les données sont encore une fois séparées selon le mois d'enregistrement du fichier. Finalement, dans la quatrième couche de l'archive, on retrouve un répertoire par jour d'enregistrement d'un fichier.

Dans chaque répertoire représentant un jour spécifique, on retrouve les données BRAMS organisés par heure dans des fichiers de format TAR. Ce format permet de regrouper plusieurs fichiers dans un même fichier, qui prend alors le format TAR. Le nom des fichiers TAR et les fichiers WAV qui se trouvent à l'intérieur des fichiers TAR suivent également une structure fixe. Il est composé de six informations ordonnées de la manière suivante :

1. Les trois lettres **RAD**.
2. Le code de la station émettrice. Comme il n'y a qu'une station réceptrice à présent, la valeur sera toujours **BEDOUR**.
3. La date du début de l'enregistrement du fichier. Elle respecte le format YYYYMMDD.
4. Le temps du début de l'enregistrement du fichier, précis à la minute. Elle respecte le format HHMM.
5. Le code de location de la station émettrice. Un code de location est composé de deux informations. Les deux premiers caractères représentent le code du pays où se trouve la station (e.g. BE, FR, ...) et les quatre lettres restantes représentent typiquement les quatre premières lettres de la ville où est situé la station (e.g. HUMA pour Humain, UCCL pour Uccle, ...).
6. L'alias de la station. Elle est composée des trois caractères 'SYS', suivie du numéro de l'antenne qu'a capté les données. Le numéro de l'antenne est toujours écrit sur trois chiffres (e.g. 001, 002, ...).

Toutes ses valeurs sont séparées par un tiret bas. Un exemple de nom de fichier est affichée à la figure 7.

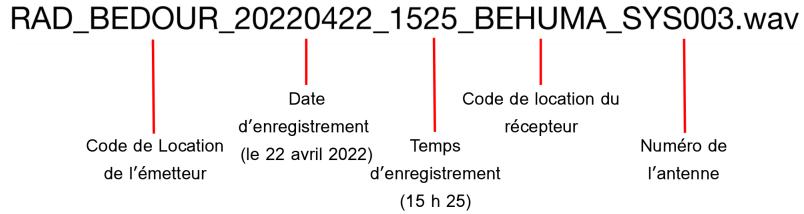


FIGURE 6 – Exemple de nom de fichier BRAMS.

## 4 Problématique liée à l'étude des données BRAMS

Afin de pouvoir trouver la trajectoire d'un météore dans l'atmosphère, il est nécessaire de rechercher par quelles stations réceptrices le météore est détecté. Ceci est une procédure qui prend un temps non négligeable pour les scientifiques du projet. En effet, toutes les stations de réception ne détectent pas tous les météores. De plus, certaines données peuvent être corrompues ou encore, non-existantes.

De plus, savoir, ou au moins avoir une idée, si un météore détecté par une station réceptrice est le même qu'un météore détecté sur une autre station réceptrice serait une grosse économie de temps.

C'est pour cette raison que les membres du projet BRAMS souhaitent automatiser deux procédures. D'une part, il faut un logiciel capable de vérifier si un fichier contient des données non-corrompues et utilisables. Dans le cas où les données sont inutilisables, le programme doit pouvoir aider l'utilisateur à retrouver la cause du problème. D'autre part, il faut un programme qui, sur base d'un écho de météore d'une station de réception, retrouve les échos pouvant venir de ce même météore sur des fichiers d'autres stations de réception. Ce dernier logiciel doit également donner le plus d'informations possibles pouvant aider à confirmer si un écho correspond à un autre.

## 5 Méthodologie

Afin d'arriver à un résultat final de qualité et qui est conforme aux requis des membres du projet BRAMS, il est important d'utiliser une bonne méthodologie.

La première chose que j'ai fait est : comprendre quoi exactement le programme devrait faire. Pour ce faire, plusieurs réunions ont eu place avant la réalisation du travail. Durant ces réunions, j'ai pu poser mes questions et demander des explications sur les concepts à connaître pour pouvoir réaliser le travail. Ayant reçu des documents qui expliquent de façon claire et précise le fonctionnement du réseau BRAMS, j'ai pu me préparer avant de m'attaquer à l'analyse et la réalisation du travail.

Durant la période d'analyse et de réalisation du TFE, j'ai pu régulièrement demander validation quant à la direction que je prenais pour mon travail. Je présentais régulièrement mes réalisations aux scientifiques du projet BRAMS afin d'avoir un feedback régulier et de pouvoir perfectionner mon programme un maximum.

Dans le but d'être plus efficace lors de l'écriture des fonctionnalités pour le programme, je décrivais avant ce que cette fonctionnalité devait faire. Ensuite, je la divisais en tâches techniques afin de pouvoir m'organiser plus facilement. Chaque tâche technique contenait et expliquait les étapes que le programme devrait exécuter pour accomplir cette tâche technique. Les étapes étaient décrites textuellement, par pseudo-code, par schéma ou encore, par le mélange de deux ou trois des moyens cités.

Durant l'entièreté du projet de fin d'études, j'ai tenté de garder un rythme de travail régulier. Je me suis organisé de telle façon de pouvoir travailler une moyenne de 23 heures par semaine. Une grande partie de ces heures se sont déroulés lorsque je rentrais de mon stage ou pendant le week-end.

## 6 Technologies utilisées

### 6.1 Python

Le programme réalisé est entièrement écrit en Python. Ce choix a été pris premièrement pour les librairies performantes et open-source qu'offre Python. En effet, des librairies comme numpy, scipy ou encore matplotlib ont été très utiles pour arriver à un résultat performant et fonctionnelle. Ces librairies offrent beaucoup de flexibilité et offrent une haute performance grâce à leur écriture en C, C++ et même en Fortran.

Un autre avantage du Python est sa portabilité. Puisque c'est un langage interprété, largement répandu, quasiment tous les systèmes d'exploitation le supportent. Ceci a facilité aussi bien la phase de développement que la phase de testing du projet de fin d'études.

Un des désavantages souvent évoqués pour Python, est que le langage est lent pour des gros traitement de données. Cependant, dans le cas de ce travail, ceci ne fut pas un problème et les librairies utilisées étaient largement assez rapide. Ceci est dû, entre autres, grâce à la nature open-source de ces librairies ainsi que leur âge. Suite à ces deux facteurs, des milliers de personnes travaillent sur ces librairies depuis plusieurs années dans le but d'optimiser le plus possible ses fonctions et méthodes.

#### 6.1.1 Librairies Utilisées

Dans cette section sont listées toutes les librairies Python utilisés pour la réalisation du travail de fin d'études.

##### Argparse

Argparse est une librairie Python qui permet de facilement créer une interface ergonomique en ligne de commande. On spécifie les paramètres nécessaires et optionnels pour le programme qu'on crée, et lorsqu'on lance le programme, Argparse pourra retrouver les paramètres entrés par l'utilisateur. Il indiquera également une erreur si un paramètre obligatoire manque, ou qu'un type de paramètre n'est pas respecté. Argparse est une librairie installée par défaut avec Python.

##### CSV

La librairie CSV offre la possibilité d'écrire et de lire des fichiers CSV à l'aide d'un programme en Python.

### **Datetime**

La librairie Datetime offre une suite de fonctions et de classes permettant la manipulation et la conversion de temps et de dates. Cette librairie fait partie des modules standards de Python et ne doit donc pas être installé.

### **Dotenv**

La librairie Dotenv permet de lire les fichiers et les variables d'environnement.

### **GeoPy**

GeoPy est une librairie Python qui facilite la recherche de coordonnées d'adresses, de villes ou encore de lieux connus. Elle permet également de trouver la distance entre deux lieux en donnant les coordonnées de ces endroits.

### **Math**

Le module math offre toute une série de fonctions mathématique de base. Elle est installée par défaut avec Python.

### **Matplotlib**

La librairie Python Matplotlib permet de générer des graphiques et de les enregistrer en tant qu'image.

### **Mysql.connector**

Cette librairie permet d'établir une connexion à une base de données MySQL ou MariaDB. Avec cette connexion, un programme Python peut faire des requêtes à cette base de données.

### **NumPy**

NumPy est une librairie Python utilisée lorsqu'on travaille avec des listes très larges. Il contient de nombreuses fonctions pour l'algèbre linéaire, les Transformées de Fourier et les matrices. Par rapport aux listes classiques de Python, les listes NumPy (aussi appelés des ndarray) sont beaucoup plus rapides.

Ceci est dû à trois raisons principales. La première est que les listes NumPy sont stockés à un endroit continu dans la mémoire RAM<sup>3</sup>, ce qui permet un

---

3. Random Access Memory

accès et une manipulation plus efficace des données. Ensuite, comme NumPy est une librairie open-source, il est continuellement adapté à des nouvelles architectures de processeurs. Finalement, une grande partie de la librairie NumPy est écrit dans des langages de bas niveau tels que le C, le C++ ou encore le Fortran. Ceci résulte à un code beaucoup plus proche du langage machine et qui est donc plus optimisé.

## Os

La librairie os de Python contient une suite de fonctions permettant d’interagir avec le système d’exploitation. Cette librairie vient par défaut avec une installation de Python, ce qui évite de devoir l’installer lorsque nécessaire.

## SciPy

Scipy est une librairie qui est basé sur la librairie NumPy. Elle partage donc de nombreuses fonctions et méthodes avec cette dernière et fonctionne également avec des ndarray. Cependant, cette librairie ajoute des fonctions pour la science des données. De plus, Scipy optimise des fonctions déjà présentes dans la librairie NumPy. Ceci permet de construire un programme qui demande moins de ressources machine et qui est donc plus rapide.

## Simplejson

Simplejson est une librairie Python offrant la possibilité de facilement écrire et lire des fichiers JSON<sup>4</sup>. Elle vient par défaut avec Python sous le nom ‘json’. Cependant, installer Simplejson séparément ajoute l’avantage d’avoir des mises à jour plus régulières et de disposer des dernières fonctionnalités plus rapidement.

## Tarfile

Le module Python Tarfile permet de lire et écrire des fichiers du format TAR. Cette librairie est incluse par défaut avec Python et ne doit donc pas être installé séparément.

## Tqdm

Tqdm est une librairie Python permettant d’afficher des barres de chargements en ligne de commande. Elle permet également de donner une estimation de temps de chargement restant.

---

<sup>4</sup>. JavaScript Object Notation

## **6.2 MariaDB**

Afin de sauvegarder les données produites par le programme, une base de données est requise. Les données seront enregistrées dans la base de données existent du projet BRAMS. Le type de base de données du projet BRAMS et MariaDB.

## 7 Lecture des Fichiers BRAMS

Avant de commencer à développer des outils pour la détection des météores ou pour l'analyse des fichiers BRAMS, il a fallu développer un logiciel permettant de lire ces fichiers. Ce logiciel sera implémenté dans un module Python contenant une classe nommée `BramsWavFile`. Ceci veut dire qu'il pourra facilement être utilisé par d'autres fichiers Python et qu'il doit donc être flexible. Dans cette section, le fonctionnement de ce module est expliqué en détail.

### 7.1 Fonctionnement du Module

Le contenu principal du module est donc la classe `BramsWavFile`. Cette classe permet de lire et de récupérer des informations sur les fichiers WAV du réseau BRAMS de manière efficace. Elle est basée sur un code écrit par Michel Anciaux, membre du projet BRAMS. Cependant, elle a fortement été retravaillé afin de l'adapter aux besoins du programme et de l'optimiser. La classe prend six paramètres en entrée :

- Le paramètre **date\_time** indique le temps et la date du fichier recherché. Ce paramètre est obligatoire et doit être de type `datetime`, type qui est défini dans le module `datetime`.
- Le paramètre **station** indique le code de location de la station recherchée (e.g. BEHUMA, BEHAAC, BEUCCL). Les deux premières lettres de ce code représentent le code pays (e.g. BE, FR). Typiquement, les quatre lettres suivantes sont les quatre premières lettres de la ville où se trouve la station. Ce paramètre est obligatoire et doit être de type `str` (chaîne de caractères).
- Le paramètre **alias** indique l'antenne, se trouvant à la location spécifiée, dont on veut récupérer un fichier. En effet, certaines stations disposent de plusieurs antennes. Les antennes sont spécifiées à l'aide de six caractères : les trois premiers contiennent les lettres 'SYS' et les trois suivants contiennent le numéro de l'antenne (e.g. 001, 002, ...). La valeur par défaut de ce paramètre est 'SYS001', c'est donc un paramètre optionnel. Le type du paramètre est `str` (chaîne de caractères).
- Le paramètre **is\_wav** est un booléen indiquant à la classe si le fichier recherché se trouve dans un fichier de format TAR (dans quel cas la valeur devrait être 'faux') ou pas (dans quel cas la valeur devrait être 'vrai'). Ce paramètre est optionnel et prend la valeur 'faux' par défaut.

- Le paramètre **parent\_directory** indique le répertoire parent ou il faudra commencer la recherche du fichier demandé. C'est un paramètre optionnel de type str (chaine de caractère) qui prend le répertoire de l'archive BRAMS comme valeur par défaut.
- Le paramètre **from\_archive** est un booléen indiquant si le fichier recherché se trouve dans un répertoire respectant la structure de l'archive BRAMS (dans quel cas la valeur est 'vrai') ou non (dans quel cas la valeur est 'faux'). Elle prend la valeur 'vrai' par défaut et est donc optionnel.

Lorsque la classe est initialisée, elle vérifie d'abord si le fichier recherché existe. Si le paramètre 'from\_archive' contient la valeur 'vrai', la classe rajoute, au chemin spécifié par le paramètre 'parent\_directory', les dossiers nécessaires pour que le chemin respecte la structure de l'archive BRAMS. Ensuite, le programme essaye de lister les contenus du dossier. Dans le cas où le dossier n'est pas trouvé, l'initialisation de la classe est interrompue et l'exception 'DirectoryNotFoundError' est levée.

La classe recherche ensuite pour le bon fichier dans le contenu du répertoire en comparant la date et le temps dans le nom du fichier avec la date et le temps reçu comme paramètre. Quand les fichiers WAV sont contenus dans une archive TAR, l'étape précédente est répétée pour trouver le bon fichier dans l'archive. Si aucun fichier correspondant à la date et la station reçue en paramètres est trouvé, l'initialisation de la classe est interrompue et l'exception 'BramsError' est levée. Une fois le bon fichier trouvé, toutes les données brutes sont lues et enregistrées dans un buffer (variable).

La prochaine étape est la séparation des différents blocs de données (ou chunk) contenus dans le fichier WAV. Pour ça, la classe parcourt le buffer contenant les données du fichier WAV. Il utilise un pointeur de lecture indiquant la position en octets du prochain bloc de données dans le buffer. Ce pointeur est calculé à partir du champ 'subChunkSize' indiquant la longueur du bloc de données courant.

Lorsque le programme retrouve un bloc de données avec un champ ID correspondant à 'fmt', 'BRA1' ou 'data', il extrait toutes ces données et les place dans un ndarray. Ceci est fait à l'aide de la fonction 'frombuffer' venant de la librairie NumPy. Cette fonction offre la possibilité de convertir, de façon très rapide, des données vers un ndarray. Une des contraintes pour utiliser cette fonction est l'indication du type de données qu'on veut placer

dans un ndarray.

Les données contenues dans la classe seront la fréquence d'échantillonnage, extrait du bloc de données 'BRA1', et la piste audio du fichier WAV, extrait du bloc de données 'data'.

### 7.1.1 La méthode FFT

La classe BramsWavFile dispose d'une méthode publique. Cette méthode se nomme 'FFT' et permet, comme son nom l'indique, de récupérer la Transformée de Fourier de l'entièreté de la piste audio du fichier WAV. Ceci est fait à l'aide de la fonction 'rfft' de la librairie Scipy, multipliée par une fenêtre de Hann. La fonction 'rfft' existe également dans la librairie NumPy, cependant dans ce cas de figure la librairie SciPy permettait de calculer la Transformée de Fourier plus rapidement. Enfin, la fonction FFT retourne la Transformée de Fourier normalisée, son abscisse et sa résolution fréquentielle.

Le code de cette méthode peut être visualisé à la figure 8.

```

1 def FFT(self , Isamples , force_new=False):
2     if (
3         self.fft is not None
4         and self.fft_fbin is not None
5         and self.fft_freq is not None
6         and not force_new
7     ):
8         return self.fft_freq , self.fft , self.fft_fbin
9
10    # get the length of all the audio samples
11    nsamples = Isamples.size
12
13    # create a window function
14    w = windows.hann(nsamples)
15    w_scale = 1 / w.mean()
16
17    # apply that window on all the audio samples
18    Isamples = Isamples * w * w_scale
19
20    # get the Fourier Tranform and normalize it
21    S = rfft(Isamples) / nsamples
22    S[1:-1] *= 2
23
24    self.fft = S
25    self.fft_fbin = self.fs / nsamples
26    self.fft_freq = rfftfreq(nsamples , 1 / self.fs)
27
28    return self.fft_freq , S , self.fft_fbin
29

```

FIGURE 7 – Code de la méthode qui génère une Transformée de Fourier sur base de données audio.

## 8 Monitoring des Données BRAMS

Comme indiqué précédemment, les fichiers ne contiennent pas toujours des données utilisables. Pour des diverses raisons, il peut parfois arriver que le bruit est trop fort pour pouvoir détecter des météores. Ou encore, lorsque les stations ICOM arrivent en fin de vie, la puissance des données détectées décroît de telle sorte qu'on détecte du bruit seulement. Bref, de nombreux problèmes peuvent survenir avec les données. Pour optimiser la détection de météores, les données corrompus doivent être détectés avant celle-ci. C'est pourquoi, un outil permettant le monitoring des données a été développé.

Cet outil se base sur deux éléments dans les données BRAMS indiquant si un fichier est utile ou non. Le premier est le bruit. En effet, lorsque le bruit augmente ou diminue beaucoup d'un coup ou graduellement sans revenir à un niveau normal, il est fort probable qu'il y ait un problème avec la station réceptrice ou son environnement.

Le deuxième élément permettant de détecter un problème est le signal du calibreur. Le signal calibreur devrait toujours être présent dans un fichier produit par une station réceptrice. Typiquement elle devrait se trouver autour de la fréquence 1500 Hz. Cependant, avec les anciennes stations, elle peut varier jusqu'à 250 Hz dépendant de la chaleur. Dans le cas où elle n'est pas aux alentours de 1500 Hz, la probabilité qu'il y ait une erreur avec la station est grande.

### 8.1 Fonctionnement du Programme

Le programme permet donc de mesurer le niveau du bruit et le signal calibreur d'un ou de plusieurs fichiers WAV du projet BRAMS. Il fait ceci en calculant la densité spectrale de puissance (dsp) de ceux-ci. La densité spectrale de puissance montre la distribution des puissances à travers les différentes fréquences contenues dans le signal de base. Ceci permet de comparer ces différentes valeurs et de remarquer des variations non typiques.

Lorsqu'on lance le programme, trois paramètres peuvent être ajoutés :

1. Le paramètre **START DATE** indique la date à partir de laquelle on veut mesurer la dsp des fichiers. Par défaut, elle prendra la date d'un jour avant le lancement du programme.
2. Le paramètre **END DATE** indique la date avant laquelle on veut mesurer la dsp des fichiers. Dans le cas où le paramètre 'START DA-

```

1 def get_psd(f, flow=800, fhigh=900):
2     # get fourier tranform from BramsWavFile class
3     freq, S, fbin = f.FFT(f.Isamples)
4     idx = (freq >= flow) * (freq < fhigh)
5
6     # calculate the total power of the wanted frequencies
7     p = (S[idx] * S[idx].conj()).real / 2
8
9     # get a mean normalized to 1Hz
10    psd = p.mean() / fbin
11
12    return psd
13

```

FIGURE 8 – Code de la fonction permettant de calculer la dsp.

TE' n'est pas donné, elle prend la valeur de la date de lancement du programme.

3. Le paramètre **STATIONS** indique les stations pour lesquelles on veut mesurer la dsp des fichiers. Si ce paramètre n'est pas donné, le programme mesurera la dsp des fichiers de toutes les stations.

Le programme calculera donc la dsp des fichiers de tous les fichiers générés entre **START DATE** et **END DATE** venant des stations **STATIONS**.

La première va donc, pour chaque fichier demandé, performer une suite d'étapes. Premièrement, elle va chercher le fichier à l'aide de la classe 'Bram'sWavFile'. Si une erreur se produit lors de la recherche du fichier, on considère que le fichier n'existe pas et le programme passe au fichier suivant. Lorsqu'on a le fichier, on peut calculer la dsp du bruit et du signal calibreur. La fonction permettant de calculer la dsp, peut être visualisée à la figure 9.

Dans le cas du bruit, on prend toujours la moyenne de la dsp entre 800 Hz et 900 Hz. Entre ces deux fréquences on retrouve normalement uniquement du bruit. Dans les cas où il se trouverait qu'il y ait un signal autre que le bruit, il ne devrait pas avoir grand impact sur la dsp puisque cette dernière se calcule sur 100 Hz, sur l'entièreté d'un fichier.

Pour le signal calibreur, in ne suffit pas de calculer la moyenne de la dsp à 1500 Hz. En effet, la fréquence du calibreur peut varier jusqu'à 250 Hz, autour de la fréquence de 1500 Hz, d'un fichier à l'autre. Comme le signal

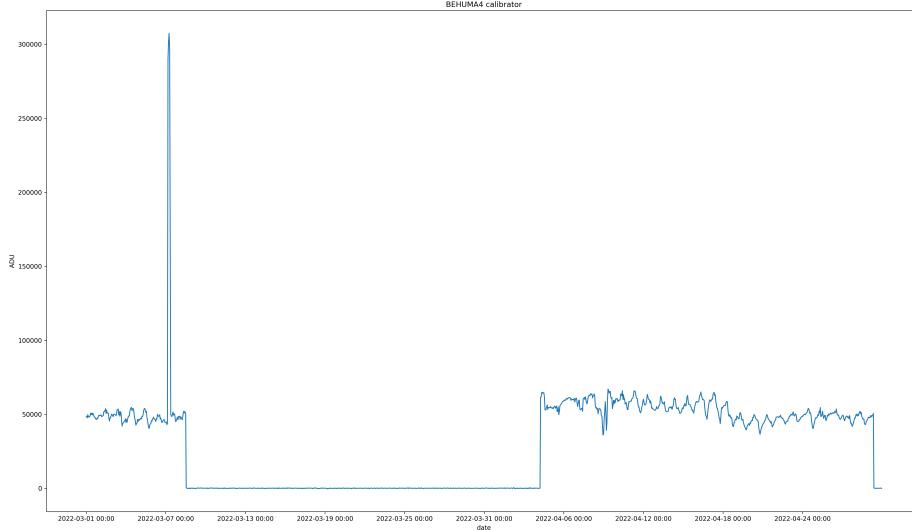


FIGURE 9 – Signal calibreur de la station BEHUMA, antenne 4 entre le 1<sup>er</sup> mars 2022 et le 30 avril 2022.

calibreur est le seul signal se trouvant entre 1350 Hz et 1750 Hz, il suffit de rechercher la fréquence, entre ces deux valeurs, avec la valeur de dsp maximale.

Enfin, le programme enregistre toutes les données de dsp dans la base de données BRAMS existante. Chaque valeur de dsp est alors lié à son fichier. Si l’option **-p** (**-plot**) a été donné au lancement du programme, le programme génère également un graphique représentant l’évolution de la dsp au cours du temps.

## 8.2 Résultats

Lors des tests, quelques résultats ont déjà montrés des informations intéressantes. Un bon exemple est l’antenne quatre de la station à Humain entre le 1<sup>er</sup> mars 2022 et le 30 avril 2022, à la figure 10. On remarque qu’à partir 8 mars 2022, le signal calibreur diminue fortement d’intensité. Après une étude en profondeur par les scientifiques du projet BRAMS, ils ont trouvé que ceci était dû à un problème avec l’oscillateur local. Ce problème causait une translation de 500 Hz vers le bas de tout signal. Ceci veut dire que tous les fichiers générés alors qu’il y avait ce problème sont inutilisables.

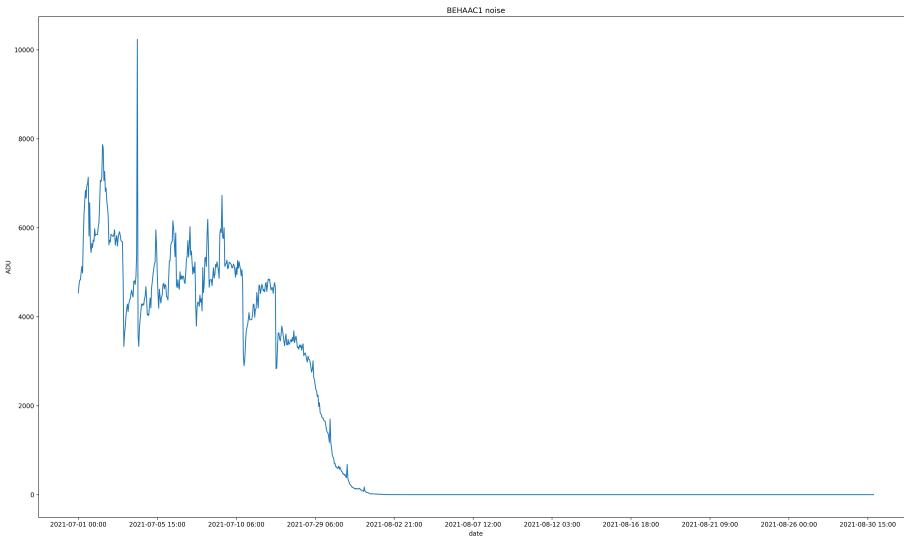


FIGURE 10 – Intensité du bruit de la station BEHAAC, antenne 1 entre le 1<sup>er</sup> juillet 2021 et le 31 août 2021.

Un autre exemple est la station à Haacht entre le 1<sup>er</sup> juillet 2021 et le 31 août 2021 affichée à la figure 11. Ici, on constate une diminution graduelle de l'intensité du bruit, jusqu'à ce que le récepteur ne capte plus rien. Ceci est causé suite au vieillissement du récepteur ICOM. Cette station a donc dû être remplacée.

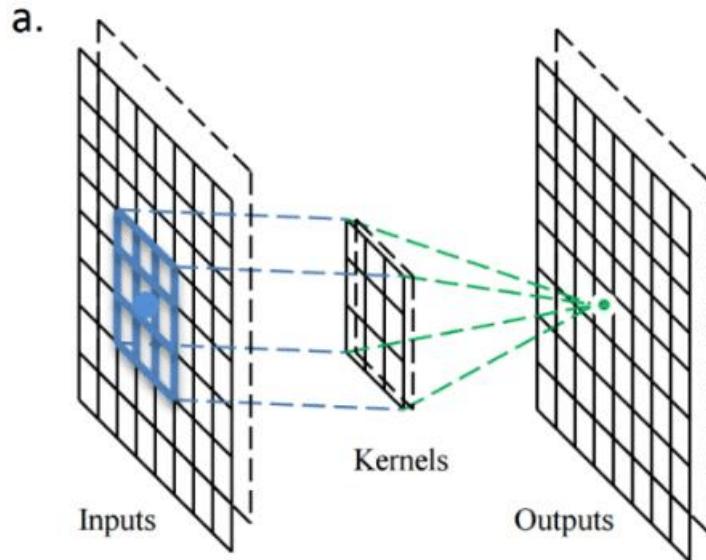


FIGURE 11 – Illustration de la convolution.

## 9 Détection des Météores

Finalement, on arrive à la détection des météores. Ce programme effectue deux tâches principales. La première tâche consiste à, à partir des coordonnées d'un météore donné, détecté sur une station donnée, trouver tous les échos qui viennent potentiellement du même météore. Ensuite, la deuxième tâche du programme est de donner l'utilisateur le plus d'informations possibles pour confirmer qu'un écho correspond à l'écho de météore reçu en entrée ou non.

### 9.1 Analyse

C'est ici la première étape qui pose la plus grande difficulté. En effet, la détection de météores consiste à retrouver des objets d'une forme spécifique dans une image. Cette image est, dans ce cas si, un spectrogramme calculé à partir d'un fichier WAV venant d'une station de réception BRAMS. De plus, on retrouve sur ce spectrogramme un nombre d'autres parasites que le programme doit pouvoir différencier d'un vrai météore tel que les échos d'avions qui passent, ou encore le signal direct de l'émetteur.

Afin de faciliter la recherche d'échos de météores dans un spectrogramme, l'amplification de ces échos peut faciliter la tâche. Un écho de météore est, dans un spectrogramme, caractérisé par un signal qui s'étale sur minimum 10 Hz et qui dure typiquement entre une et trois secondes. Ceci est possible en effectuant une convolution sur le spectrogramme. Une convolution est une opération mathématique qui prend deux fonctions en entrée et qui produit un nouveau signal. Dans ce cas si, le signal en entrée sera le spectrogramme, ou une partie du spectrogramme, et le deuxième signal est le filtre. Une illustration de la convolution peut être trouvée à la figure 12. Afin d'arriver à un bon résultat, il a fallu tester plusieurs filtres. Le filtre offrant les meilleurs résultats est illustré à la figure 13. Elle permet de faire deux choses :

- Premièrement, elle amplifie les éléments qui durent plus de 10 Hz. En effet, pour avoir un résultat d'une valeur maximale il faut que le haut et le bas du filtre soient multipliées par des grandes valeurs.
- Deuxièmement, elle permet de diminuer l'intensité des éléments qui ont une longue forme horizontale (typiquement des échos d'avions).

Suit alors la suppression du signal direct de l'émetteur. En effet, pour détecter les échos de météores, le signal direct nous complique la tâche. Pour l'éliminer, le programme recherche d'abord sa fréquence entre 800 Hz et 1200 Hz, en prenant la valeur maximale sur l'entièreté de la durée du fichier, entre ces deux fréquences. S'il n'est pas retrouvé, on considère qu'il n'est pas détecté (ce qui est le cas sur certaines stations réceptrices). Par contre, dans le cas où il est détecté, pour chaque seconde, on remplace la valeur du signal direct par la moyenne des valeurs se trouvant autour de la fréquence du signal direct.

Ensuite, il faut trouver une méthode permettant de séparer, dans le spectrogramme, les objets du bruit. La première méthode testée se basait sur un seuil fixe. Toutes les valeurs qui se trouvaient en dessous de ce seuil étaient considérées comme du bruit et leur valeur était mise à zéro. Bien que cette méthode fonctionnait bien sur un fichier spécifique, elle posait problème si on voulait la lancer sur plusieurs fichiers différents. En effet, comme toutes les stations de réception se trouvent dans des environnements différents et disposent parfois même de matériel différent, différents fichiers ont des niveaux de bruits différents.

Pour la deuxième méthode, le programme essayait de trouver un seuil qui se base sur la valeur du bruit sur l'entièreté d'un spectrogramme. Pour ce faire, la méthode divisait le spectrogramme, entre les fréquences de 600 Hz et 1400 Hz, en trente zones. Elle mesurait ensuite, à l'aide de la variance

de chaque zone, laquelle contenait le moins d'objets possibles. Finalement, la méthode calculait la 95e percentile du bloc contenant le moins d'objet possibles. Toutes les valeurs du spectrogramme se trouvant en dessous de cette valeur seraient considérés comme du bruit et mises à zéro. Malgré que cette méthode produisait des résultats marginalement meilleurs que celle à seuil fixe, elle n'était toujours pas parfaite étant donné qu'elle se basait sur le bruit d'une partie du spectrogramme, or le bruit peut varier au cours du temps.

Finalement, la méthode qui fonctionnait le mieux calculait la 95e percentile pour chaque colonne du spectrogramme. Ensuite, elle remplace toutes les valeurs de cette colonne se trouvant en dessous de la 95e percentile avec zéro. Les résidus de bruits qui ne sont pas encore éliminés sont négligeables puisqu'ils s'étalent sur moins de 10 Hz et ne sont donc pas considérés comme des météores.

Il faut maintenant éliminer tous les objets qui ne sont pas des échos de météores, mais qui s'étalent sur plus de 10 Hz. Ceci n'est pas une tâche facile puisque des parasites peuvent venir se mélanger à des échos de météore formant un objet. La première chose à faire est : nettoyer le spectrogramme des petits parasites, ou encore de résidus de bruits. Le programme fait cela en labellisant le spectrogramme et en éliminant tout objet s'étalant sur moins de 10 Hz. Ensuite, on récupère tous les objets restants. Pour chacun de ces objets, on vérifie s'il n'est pas composé de plusieurs parasites se superposant sur plus de 10 Hz. Ceci est fait en allant vérifier sur une longueur d'environ treize secondes autour de l'objet détecté, si on retrouve des parasites qui s'étalent sur la même fréquence que cet objet. Lorsque c'est le cas, l'objet n'est pas considéré comme un météore et est ignoré. À partir de ce stade, tous les objets détectés sont considérés comme étant des échos de météores.

Pour la deuxième étape, le programme récolte le plus d'informations possibles à propos des échos pouvant potentiellement correspondre à l'écho de météore reçu en entrée. Parmi ces informations, on retrouve tout d'abord le temps où un écho s'est produit. Ceci est important pour pouvoir situer l'écho si on veut l'étudier en profondeur.

Ensuite, le programme calcule la distance entre la station qui a détecté l'écho reçu en entrée et la station dont un écho pourrait correspondre à celui en entrée. En effet, un météore n'est pas détecté par toutes les stations, et dans le cas où il est détecté par plusieurs stations, ces stations se trouvent souvent proches les uns des autres.

La troisième information donnée indique le nombre d'échos trouvés pour

```

1   [[ 0.    0.    0.   50.    0.    0.    0.    ]
2   [ 0.    0.    0.   50.    0.    0.    0.    ]
3   [ 0.    0.    0.    0.    0.    0.    0.    ]
4   [ 0.    0.    0.    0.    0.    0.    0.    ]
5   [ 0.    0.    0.    0.    0.    0.    0.    ]
6   [ 0.    0.    0.    0.    0.    0.    0.    ]
7   [ 0.    0.    0.    0.    0.    0.    0.    ]
8   [ 0.    0.    0.    0.    0.    0.    0.    ]
9   [ 0.    0.    0.    0.    0.    0.    0.    ]
10  [ 0.    0.    0.    0.    0.    0.    0.    ]
11  [ 0.    0.    0.    0.    0.    0.    0.    ]
12  [ 0.    0.    0.    0.    0.    0.    0.    ]
13  [-1.5  0.    0.    0.    0.    0.   -1.5]
14  [-1.5  0.    0.    0.    0.    0.   -1.5]
15  [-1.5  0.    0.    0.    0.    0.   -1.5]
16  [ 0.    0.    0.    0.    0.    0.    0.    ]
17  [ 0.    0.    0.    0.    0.    0.    0.    ]
18  [ 0.    0.    0.    0.    0.    0.    0.    ]
19  [ 0.    0.    0.    0.    0.    0.    0.    ]
20  [ 0.    0.    0.    0.    0.    0.    0.    ]
21  [ 0.    0.    0.    0.    0.    0.    0.    ]
22  [ 0.    0.    0.    0.    0.    0.    0.    ]
23  [ 0.    0.    0.    0.    0.    0.    0.    ]
24  [ 0.    0.    0.    0.    0.    0.    0.    ]
25  [ 0.    0.    0.    0.    0.    0.    0.    ]
26  [ 0.    0.    0.   50.    0.    0.    0.    ]
27  [ 0.    0.    0.   50.    0.    0.    0.    ]
28

```

FIGURE 12 – Filtre utilisé pour amplifier les échos de météores.

une station, pouvant correspondre à l'écho en entrée. Dans ce cas, il est important d'en informer l'utilisateur, puisque ces échos viennent chacun d'un météore différent et ne peuvent donc pas tous venir de cet écho reçu en entrée.

La dernière information donnée par le programme sont la fréquence minimum et la fréquence maximum des échos détectés. Ces deux fréquences permettront, ensemble avec le temps où l'écho s'est produit, de faciliter la recherche de cet écho pour l'utilisateur.

## 9.2 Fonctionnement du Programme

Le programme prend en entrée le temps et la date de l'écho du météore recherché ainsi que le code de location de la station qui l'a détecté. À partir

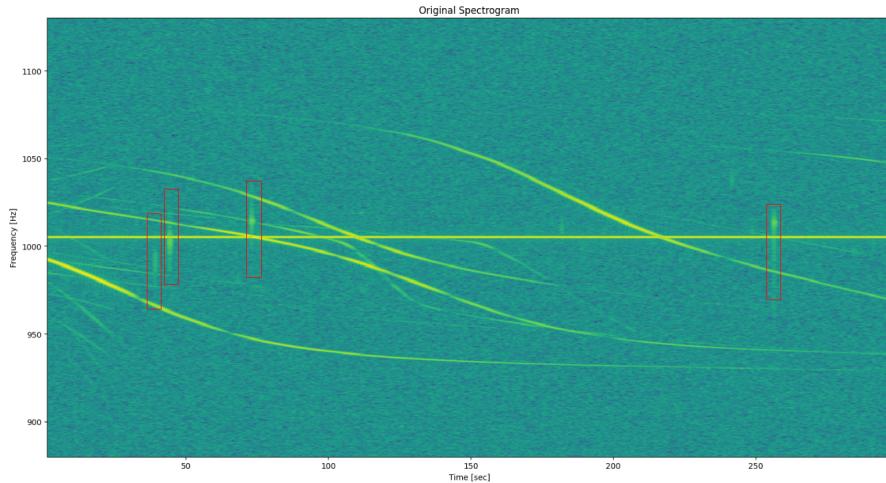


FIGURE 13 – Spectrogram originale, sans modifications. Les échos de météores sont entourés en rouge.

du temps et de la date reçue en entrée, le programme calcule l'intervalle de temps dans laquelle elle doit chercher des échos pour les autres stations réceptrices. Cet intervalle est toujours de six secondes autour du temps reçu en entrée. Cette valeur a été choisie puisque toute détection en dehors de ces six secondes a très peu de probabilité de venir du même météore que l'écho en entrée. Toutes les actions sur le spectrogramme s'appliqueront uniquement à l'intérieur de cet intervalle afin d'optimiser le programme.

Le programme va ensuite chercher, pour toutes les stations, les informations des fichiers contenant cet intervalle. Il fait pour ça une requête à la base de données BRAMS. Puis, pour chaque station, le programme calcule la distance entre la station et la location où l'écho de météore reçu en entrée a été détecté.

Quatrièmement, pour tous les fichiers dont on a récolté des informations, on va rechercher les météores dans l'intervalle calculé précédemment. Pour cette étape, le programme fait d'abord appelle à la classe BramsWavFile, qui ira chercher le fichier et extraire les données audio. L'étape suivante consiste à générer le spectrogramme à partir de ces données audio. Afin de générer ce spectrogramme, il faut faire appelle à la classe Spectrogram. Ceci est une classe spécialement faite pour ce logiciel, qui permet de créer un

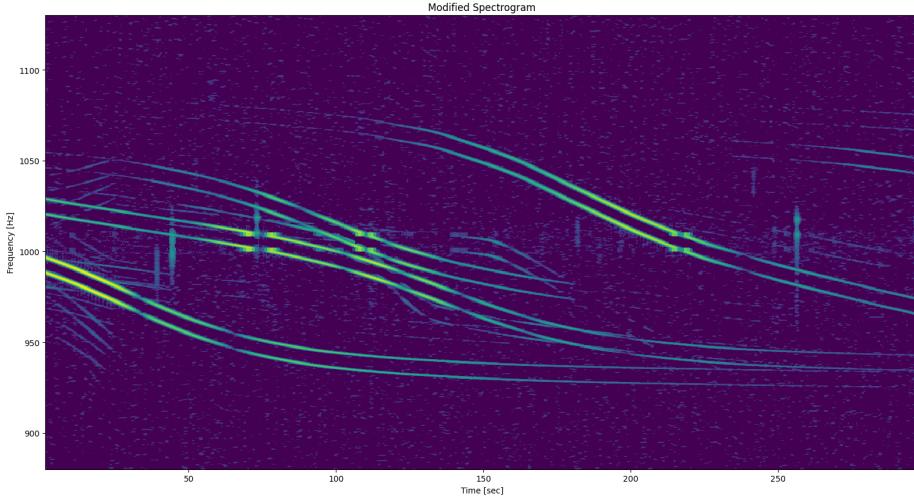


FIGURE 14 – Spectrogramme après l'amplification des échos de météores et la suppression de la majorité du bruit.

spectrogramme et de le modifier à l'aide d'une suite de méthodes. Lorsqu'on appelle cette classe, il faut passer comme paramètre les données audio. À partir de ces données, la classe va créer le spectrogramme du fichier avec la fonction "`scipy.signal.spectrogram()`", un exemple du spectrogramme original peut être visualisé à la figure 14. La classe gardera une copie originale du spectrogramme et une copie pour les modifications, dont elle supprime, dès l'initialisation, le signal direct de l'émetteur.

Suit alors une convolution du spectrogramme avec le filtre permettant d'amplifier les échos de météores, cette convolution est faite à l'aide de la fonction '`scipy.ndimage.convolve()`'. Ensuite, le filtre supprimant toutes les valeurs en dessous de la 95e percentile sera appliquée. Un spectrogramme à ce stade est illustré à la figure 15, ce spectrogramme est le même que celui à la figure 14. Afin d'éliminer tous les résidus de bruits, le programme labellise le spectrogramme et supprime tout objet s'étalant sur plus de 6 Hz. Dans le but de reconnecter les météores qui auraient été divisés en deux parties différentes, on applique un filtre moyenneur sur le spectrogramme. Puis, le logiciel cherche les coordonnées des échos de météore sur le spectrogramme. Les météores trouvés sur le spectrogramme de la figure 14 et 15 sont affichées à la figure 16. En comparant les météores mise en évidence à la figure 14 et les échos trouvés par le programme à la figure 16, on constate que le

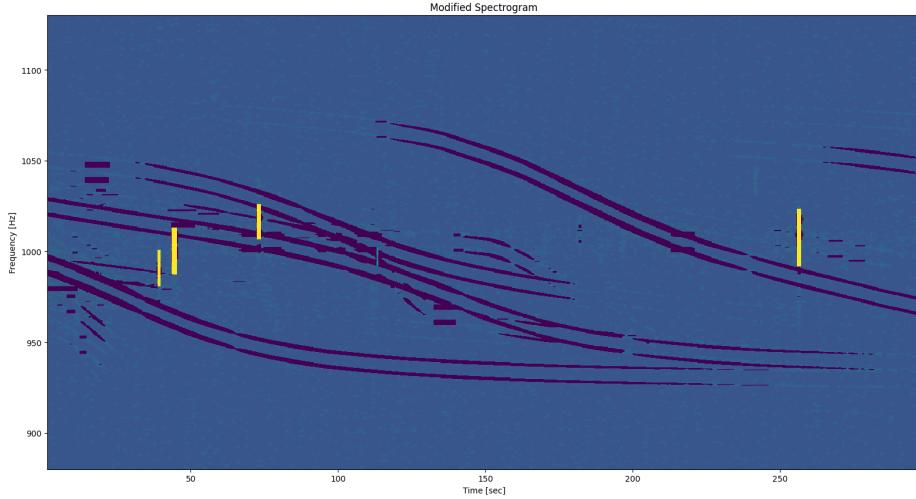


FIGURE 15 – Spectrogramme après l'amplification des échos de météores et la suppression de la majorité du bruit.

programme arrive bien à retrouver les échos des météores. Avec ces coordonnées, on pourra alors retrouver la fréquence maximale et la fréquence minimale de l'écho.

Enfin, le programme crée un fichier CSV<sup>5</sup> (figure 17) qui contiendra une ligne par écho de météore détecté. Chaque ligne contiendra :

1. Le code de location de la station où est détecté l'écho.
  2. Le numéro de l'antenne qui a détecté l'écho.
  3. La date et le temps de début d'enregistrement du fichier d'où vient l'écho.
  4. Le nombres d'écho comptés dans le même fichier que celui d'où vient l'écho.
  5. Le temps, précis à la microseconde, de détection de l'écho.
  6. La fréquence minimum de l'écho.
  7. La fréquence maximum de l'écho.
  8. La distance entre la station où est détecté l'écho et la station de l'écho reçu en entrée.
- 
5. Comma Separated Values

```

1 location_code,antenna_id,file_start,meteor_count,meteor_time,fmin,fmax,distance_km
2 BEHUMA,1,202204230000,2,2022-04-23T00:02:13:801995,1000.2845764160156,1015.7615661621094,0.0 km
3 BEHUMA,1,202204230000,2,2022-04-23T00:02:17:585397,987.835693359375,1010.7147216796875,0.0 km
4 BEHUMA,3,202204230000,2,2022-04-23T00:02:17:585397,987.4992370605469,1009.0324401855469,0.0 km
5 BEHUMA,3,202204230000,2,2022-04-23T00:02:13:801995,987.1627807617188,1012.7334594726562,0.0 km
6 BEHUMA,4,202204230000,2,2022-04-23T00:02:17:585397,988.5086059570312,1010.0418090820312,0.0 km
7 BEHUMA,4,202204230000,2,2022-04-23T00:02:13:801995,989.8544311523438,1012.7334594726562,0.0 km
8 BEHUMA,5,202204230000,2,2022-04-23T00:02:13:801995,987.835693359375,1012.7334594726562,0.0 km
9 BEHUMA,5,202204230000,2,2022-04-23T00:02:17:241451,988.1721496582031,1011.7240905761719,0.0 km
10 BEHUMA,6,202204230000,2,2022-04-23T00:02:13:801995,990.52734375,1002.6397705078125,0.0 km
11 BEHUMA,6,202204230000,2,2022-04-23T00:02:17:585397,990.52734375,1009.368896484375,0.0 km

```

FIGURE 16 – Exemple de fichier CSV généré par le programme de détection de météores.

De cette façon, les données pourront facilement être lues et réutilisées pour étudier ces météores.

## 10 Pistes d'Améliorations

Suite à une réflexion sur les programmes réalisés, j'ai pu trouver quelques pistes d'améliorations qui sont intéressantes pour le futur :

- Malgré que l'algorithme permettant de détecter des échos de météore fonctionne assez bien, il y a des cas où il détecte des échos là où il n'y en a pas et inversement. En effet, lors de cas extrêmes (échos de météore faibles, superposition de nombreux parasites), le programme a parfois du mal à séparer les échos de météores du reste des objets. Peaufiner cet algorithme permettrait d'éviter ces problèmes.
- Lors de la détection des échos de météore, l'élimination du signal direct de l'émetteur est également un domaine qui peut être amélioré. Actuellement, il est remplacé par la moyenne des valeurs situées autour de la fréquence du signal directe. Cependant, ceci n'est pas l'idéal puisqu'on élimine une partie du signal utile avec cette méthode. Une meilleure façon de l'éliminer est la reconstruction du signal directe, suivi de la soustraction.
- Afin d'améliorer l'expérience utilisateur du programme, on pourrait ajouter ces deux programmes au site web BRAMS. Ceci permet d'avoir un programme avec une interface plutôt que d'avoir un programme en ligne de commande.
- Bien que ce n'est pas un point important, l'optimisation du programme permettrait d'avoir un temps d'exécution plus court. Cela peut être achevé en essayant des librairies alternatives à NumPy et SciPy, ou encore à l'aide du développement de ses propres fonctions et méthodes.

## 11 Conclusion

Le but du travail était d'automatiser deux procédures du projet BRAMS. Premièrement elle devait permettre aux scientifiques d'identifier les données corrompues et aider à retrouver la cause de ceux-ci. Ceci a été accompli à l'aide du logiciel de monitoring qui analyse les fichiers du projet BRAMS.

Deuxièmement, le travail devait automatiser la recherche de tous les échos de météores pouvant correspondre à un météore spécifique. Cette tâche peut maintenant être complétée à l'aide du programme de détection de météore.

Ce travail m'a permis d'étendre mes connaissances dans le domaine du traitement du signal en mettant en pratique plusieurs notions théoriques vues durant les cours. Le mélange de ces connaissances avec des nouvelles notions, que je n'avais pas encore appris, afin d'arriver à un résultat final a été un vrai défi pour moi.

Enfin, ce projet m'a permis de contribuer à un projet scientifique de grande échelle.

## Références

- [1] *Site internet du projet BRAMS*, consulté en janvier 2022  
<https://brams.aeronomie.be/>
- [2] *Documentation de la fonction scipy.signal.spectrogram*, consulté en février 2022  
<https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.spectrogram.html>
- [3] *Hands-On Tutorial on Visualizing Spectrograms in Python*, consulté en février 2022  
<https://analyticsindiamag.com/hands-on-tutorial-on-visualizing-spectrograms-in-python/>
- [4] *Cutting unused frequencies with specgram, matplotlib*, consulté en février 2022  
<https://stackoverflow.com/questions/19468923/cutting-of-unused-frequencies-in-specgram-matplotlib>
- [5] *Documentation Matplotlib*, consulté en février 2022  
[https://matplotlib.org/stable/api/mlab\\_api.html#matplotlib.mlab.specgram](https://matplotlib.org/stable/api/mlab_api.html#matplotlib.mlab.specgram)
- [6] *Set spectrogram Parameters*, consulté en février 2022  
<https://stackoverflow.com/questions/29321696/what-is-a-spectrogram-and-how-do-i-set-its-parameters>
- [7] *Documentation de la fonction numpy.convolve*, consulté en mars 2022  
<https://numpy.org/doc/stable/reference/generated/numpy.convolve.html>
- [8] *How to convolve two 2-dimensional matrices in python with scipy*, consulté en mars 2022  
<https://moonbooks.org/Articles/How-to-do-a-simple-2D-convolution-between-a-kernel-and-an-image-in-python-with-scipy-/>
- [9] *Slice 2D array in smaller 2D arrays*, consulté en mars 2022  
<https://stackoverflow.com/questions/16856788/slice-2d-array-into-smaller-2d-arrays>
- [10] *Compute a confidence interval from sample data*, consulté en mars 2022  
<https://stackoverflow.com/questions/15033511/compute-a-confidence-interval-from-sample-data>

- [11] *Microsoft WAVE soundfile format*, consulté en mars 2022  
<http://soundfile.sapp.org/doc/WaveFormat/>
- [12] *Python - Sending Email using SMTP*, consulté en mars 2022  
[https://www.tutorialspoint.com/python/python\\_sending\\_email.htm](https://www.tutorialspoint.com/python/python_sending_email.htm)
- [13] *The fastest 2D convolution in the world*, consulté en mars 2022  
<https://laurentperrinet.github.io/sciblog/posts/2017-09-20-the-fastest-2d-convolution-in-the-world.html>
- [14] *Documentation de la fonction scipy.ndimage.convolve*, consulté en mars 2022  
<https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.convolve.html>
- [15] *Power Spectral Density - an overview*, consulté en avril 2022  
<https://www.sciencedirect.com/topics/computer-science/power-spectral-density>
- [16] *What is a Power Spectral Density*, consulté en avril 2022  
<https://community.sw.siemens.com/s/article/what-is-a-power-spectral-density-psd>
- [17] *How to add time onto a datetime object in Python*, consulté en avril 2022  
<https://www.adamsmith.haus/python/answers/how-to-add-time-onto-a-datetime-object-in-python>
- [18] *Fourier Transforms With scipy.fft : Python Signal Processing*, consulté en avril 2022  
<https://realpython.com/python-scipy-fft/#why-would-you-need-the-fourier-transform>
- [19] *Documentation de la fonction scipy.fft.rfft*, consulté en avril 2022  
<https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.rfft.html>
- [20] *Reading and Writing CSV Files in Python - Real Python*, consulté en mai 2022  
<https://realpython.com/python-csv/>
- [21] *A Fast, Extensible Progress Bar for Python and CLI - TQDM*, consulté en mai 2022  
<https://github.com/tqdm/tqdm>
- [22] *Documentation du module tarfile*, consulté en mai 2022  
<https://docs.python.org/3/library/tarfile.html>

- [23] *Documentation de la librairie simplejson*, consulté en mai 2022  
<https://pypi.org/project/simplejson/>
- [24] *Python Command Line Arguments*, consulté en mai 2022  
<https://realpython.com/python-command-line-arguments/>
- [25] *Documentation du module argparse*, consulté en mai 2022  
<https://docs.python.org/3/library/argparse.html>
- [26] *Artificial neural network for bubbles pattern recognition on the images - Scientific Figure on ResearchGate*, consulté en mai 2022  
[https://www.researchgate.net/figure/a-Illustration-of-the-operation-principle-of-the-convolution-kernel-convolutional-layer\\_fig2\\_309487032](https://www.researchgate.net/figure/a-Illustration-of-the-operation-principle-of-the-convolution-kernel-convolutional-layer_fig2_309487032)

## **12 Annexes**