Analysis of respiratory sounds acquired with smartphone

João Pedro Sousa
Master's degree in Mathematical Engineering
Department of Mathematics
2023

Supervisor
Rute Almeida, CINTESIS@RISE, FMUP & CMUP

Co-supervisor
Cristina Jácome, CINTESIS@RISE, FMUP

Para ser grande, sê inteiro: nada

Teu exagera ou exclui.

Sê todo em cada coisa. Põe quanto és

No mínimo que fazes.

Assim em cada lago a lua toda

Brilha, porque alta vive.

Ricardo Reis, in "Odes"

Acknowledgements

Firstly, I would like to express my heartfelt gratitude to my supervisors, Dr. Rute Almeida, and Dr. Cristina Jácome, for their unwavering support, guidance, and mentorship throughout the journey of completing this Dissertation. Your expertise and dedication were fundamental in shaping my research and academic growth.

To my dearest friends, Andreia Batista, Gabriela Mendes, Juliana Nunes, Rui Miranda, and Sara Cunha, your friendship has been a constant source of inspiration and encouragement. Your belief in my abilities has kept me motivated during the most challenging times of this academic endeavour.

I would also like to acknowledge the four remarkable women who have played crucial roles in my life. To Inês, your wisdom and love have been a guiding light throughout the last six years. To my grandmother Glória your resilience and determination in life have taught me the importance of perseverance and following my dreams. To my mother Emília and sister Beatriz your kindness and compassion have shown me the beauty of empathy, and all the support and encouragement have made me the person I am today.

This Dissertation would not have been possible without the support and love of all these incredible individuals. Thank you for being a part of my journey.

Resumo

O estetoscópio é uma das mais essenciais ferramentas que os profissionais de saúde utilizam para detetar e diagnosticar doenças respiratórias, uma vez que pode rapidamente ajudar na deteção de algumas doenças letais, enquanto se torna acessível para os serviços de cuidados de saúde primários. Infelizmente, a necessidade de o ouvinte ser experiente é uma desvantagem, uma vez que pessoas com doenças respiratórias crónicas não conseguem auto-monitorizar a sua condição. Por esta razão, a identificação automática de sons respiratórios é um campo ativo de investigação em processamento de sinais médicos com o objetivo de detetar crepitações e sibilos.

Crepitações e sibilos são sons respiratórios adventícios comuns que podem indicar a presença de doença ou de distúrbio pulmonar. Investigações anteriores propuseram métodos para deteção automática de crepitações e sibilos em sons respiratórios adquiridos por um estetoscópio eletrónico. O desafio é adaptar algoritmos já desenvolvidos para gravações de sons respiratórios com smartphones.

O algoritmo de deteção de crepitação é baseado em duas etapas: (1) um valor de limiar é aplicado à derivada do primeiro valor absoluto do som respiratório para localizar a “zona de interesse” e (2) nesta zona, uma crepitação é detetada se alguma das condições forem verificadas. O algoritmo de deteção de sibilos envolve quatro etapas: (1) cálculo da representação da *time-frequency* do som gravado, (2) remoção do som respiratório básico do som respiratório total, (3) deteção de pico e (4) classificação final do som respiratório: os picos são classificados como sibilos ou não sibilos. Neste trabalho, estes algoritmos inicialmente desenvolvidos em MATLAB foram reimplementados em Python para permitir fácil inclusão num ecossistema que já recolhe sons respiratórios por meio do microfone dos smartphones.

Esses algoritmos foram aplicados a bases de dados de sons adquiridos em estetoscópios e em smartphones para avaliar o seu desempenho. A concordância entre os algoritmos de MATLAB e Python é razoável, tendo um coeficiente Kappa de Cohen de 0.629 atingindo uma exatidão balanceada de 81.7% entre algoritmos, 74.17% na identificação de sibilos pelo MATLAB e de 76.17% no Python.

Os dados de performance obtidos são promissores, no entanto, análises adicionais são necessárias para avaliar quais parâmetros e técnicas precisam de ajuste nas gravações do smartphone.

Palavras-chave: crepitações, sibilos, auscultação, sons respiratórios, deteção, smartphone, estetoscópio, algoritmo, *time-frequency*, picos, MATLAB, Python.

Abstract

The stethoscope is one of the essential tools physicians use to detect and diagnose lung diseases since it can help quickly diagnose some deadly maladies while making it affordable for primary health care services. Unfortunately, the need for an experienced listener is a drawback since people with chronic lung diseases cannot monitor their condition themselves. Therefore, automatic respiratory sound identification is an active field of medical signal processing research aiming to detect two sounds: crackles and wheezes.

Crackles and wheezes are common adventitious respiratory sounds that can indicate the presence of pulmonary disease or disorder. A previous work proposed methods for automatically detect crackles and wheezes over respiratory sounds acquired by an electronic stethoscope. The challenge is to adapt already developed algorithms to smartphone respiratory sound recordings.

The crackle detection algorithm is based on two steps: (1) a threshold value is applied to the first derivative absolute value of respiratory sound to locate the “zone of interest”, and (2) in this zone, a crackle is detected if some conditions are verified. The wheeze detection algorithm involves four steps: (1) calculation of the time-frequency representation of the recorded sound, (2) removal of the basic breath sound from the total breath sound, (3) peak detection and (4) final classification of the detected peaks as wheeze or non-wheeze.

In this work, these initially developed algorithms in MATLAB were reimplemented in Python to allow easy inclusion in an ecosystem that already collects respiratory sounds using mobile apps.

These algorithms were applied to databases of stethoscope-acquired and smartphone-acquired sounds to evaluate their performance. The agreement between MATLAB and Python algorithms is reasonable since the Kappa Cohen coefficient is 0.629, achieving a balanced accuracy of 81.7% between algorithms, 74.14% in wheeze detection in MATLAB and 76.17% in Python’s algorithm.

Despite, the performance evaluation results being promising, further analysis is required to assess what parameters and techniques need adjustment to smartphone recordings.

Keywords: crackles, wheezing, auscultation, respiratory sounds, detection, smartphone, stethoscope, algorithm, time-frequency, peaks, MATLAB, Python.

Table of Contents

List of Tables

List of Charts

List of Figures

List of Abbreviations

AIRDOC	APLICAÇÃO MÓVEL INTELIGENTE PARA SUPORTE INDIVIDUALIZADO E MONITORIZAÇÃO DA FUNÇÃO E SONS RESPIRATÓRIOS DE DOENTES OBSTRUTIVOS CRÓNICOS
AUC	AREA UNDER THE ROC CURVE
BA	BALANCED ACCURACY
CINTESIS	CENTER FOR HEALTH TECHNOLOGY AND SERVICES RESEARCH
CNN	CONVOLUTIONAL NEURAL NETWORK
COPD	CHRONIC OBSTRUCTIVE PULMONARY DISEASE
FCUP	FACULTY OF SCIENCES OF THE UNIVERSITY OF PORTO
FD	FIRST DERIVATIVE
FDAV	FIRST DERIVATIVE ABSOLUTE VALUE
FIR	FINITE IMPULSE RESPONSE
ICBHI	INTERNATIONAL CONFERENCE ON BIOMEDICAL AND HEALTH INFORMATICS
IDW	INITIAL DEFLECTION WIDTH
ISTFT	INVERSE SHORT-TIME FOURIER TRANSFORM
SG	SAVITZKY-GOLAY
STFT	SHORT-TIME FOURIER TRANSFORM
TF	TIME-FREQUENCY
TF-WD	TIME-FREQUENCY WHEEZE DETECTOR
UP	UNIVERSITY OF PORTO

1 Introduction

The year 1816 changed the way that physicians diagnose chest diseases. There was a need for an instrument capable of diagnosing chest conditions, mainly in stout individuals, where direct auscultation could be challenging and even bothersome. René Laënnec invented the stethoscope (= *stēthos* (chest) + *skopein* (to visualise, to see) [1]), using only a tube of paper tied with a string to auscultate the heartbeats of his patients. Later in that century, George P. Camman developed a Binaural stethoscope with two earpieces [2], with a similar shape to the modern one.

When conjugated with an experienced listener, this instrument has excellent value once it can quickly diagnose deadly disorders, making it affordable for primary health care services [3].

Despite all the benefits of performing auscultation as a diagnostic procedure, some disadvantages need to be considered: as the listener is a human, some random inaccuracies in identifying abnormal respiratory sounds may occur, possibly resulting in bad judgments [4]; it is required some silence to make the auscultation, what can be challenging to achieve in noisy clinical settings; the listener needs much training to make a good diagnosis, what means that the great majority of unhealthy people cannot monitor their disease by themselves.

Later in the twentieth century, the first electronic stethoscope was created, allowing sound to be louder and more precise. Nowadays, digital stethoscopes can filter the noise of the sound [5], surpassing some of the disadvantages pointed out earlier.

1.1 Motivation

The next big step in the evolution of the stethoscope is using all the scientific knowledge in signal processing to detect specific patterns that can lead to a diagnosis. This improvement can substantially reduce human error since clinical reasoning is interpreted into the detection algorithm [6]. As described in the next chapter, some algorithms were already developed with exciting results.

The reader may then ask how the stethoscope can be improved to make auscultation even more profitable. The answer is simple: making it available to everyone despite their level of knowledge in auscultation [7]. Nowadays, smartphones are accessible to everyone [8], so using their microphone and the enormous mathematical developments might solve this challenge [9].

This Dissertation explores the already-built algorithms with good results for the digital stethoscope. It adapts them to a database with recordings of respiratory sounds acquired with smartphones, aiming for the best results.

1.2 Adventitious respiratory sounds

Crackles and wheezes are familiar adventitious respiratory sounds that can indicate the presence of pulmonary disease or disorder [10].

Crackles are a type of lung sound characterised by a popping or crackling noise. They are caused by the opening and/or closing small airways and alveoli in the lungs [11]. Crackles can be heard with a stethoscope and are classified into fine and coarse: fine crackles are typically heard in the upper lobes of the lungs and are associated with conditions such as pneumonia and idiopathic pulmonary fibrosis; coarse crackles, on the other hand, are typically heard in the lower lobes of the lungs and are associated with conditions such as chronic bronchitis [12]. A possible explanation of crackles in chronic respiratory disease could be air bubbling in the secretion of the airways. The number of crackles is correlated with the severity of the underlying disease.

Wheezes are lung sounds characterised by a high-pitched whistling noise. They are caused by narrowing or obstructing the airways, which increases airflow resistance [13]. The bronchial obstruction causes harmonic oscillations of the airways due to Bernoulli’s principle, which can be heard as pitched noises on top of the physiological breath sounds. If there are multiple obstructions, the different wheezing sounds are overlapped, which results in polyphonic wheezing.

Wheezes can also be heard with a stethoscope and are associated with asthma and chronic obstructive lung disease [12]. In addition to helping with diagnosis, crackles and wheezes can also be used to monitor the progression and treatment of respiratory diseases [14]. For example, if a patient with chronic obstructive pulmonary disease (COPD) is experiencing an exacerbation, healthcare providers may hear an increase in the number and severity of crackles [15]. Similarly, if a patient with asthma is not responding well to treatment, healthcare providers may hear an increase in the severity of wheezes [16]. The normal auscultation performed with an analogic stethoscope typically cover all zones of the posterior and anterior parts of the chest, which might be challenging to achieve with a smartphone, especially in a self-monitoring set-up.

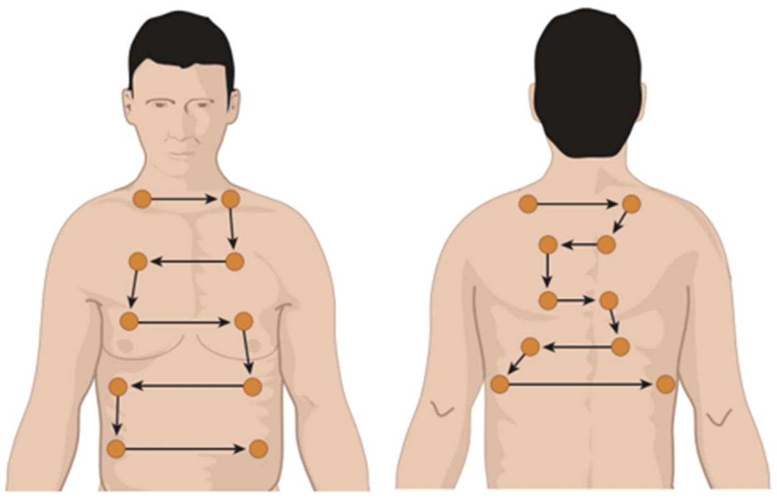
Figure 1 - Anterior and posterior auscultation schema.

1.3 Automatic respiratory sound detection

Automatic respiratory sound identification is an active area of research in medical signal processing. Intuitively, seeing some waveforms of wheeze, crackle and normal sounds, some patterns and differences are evident.

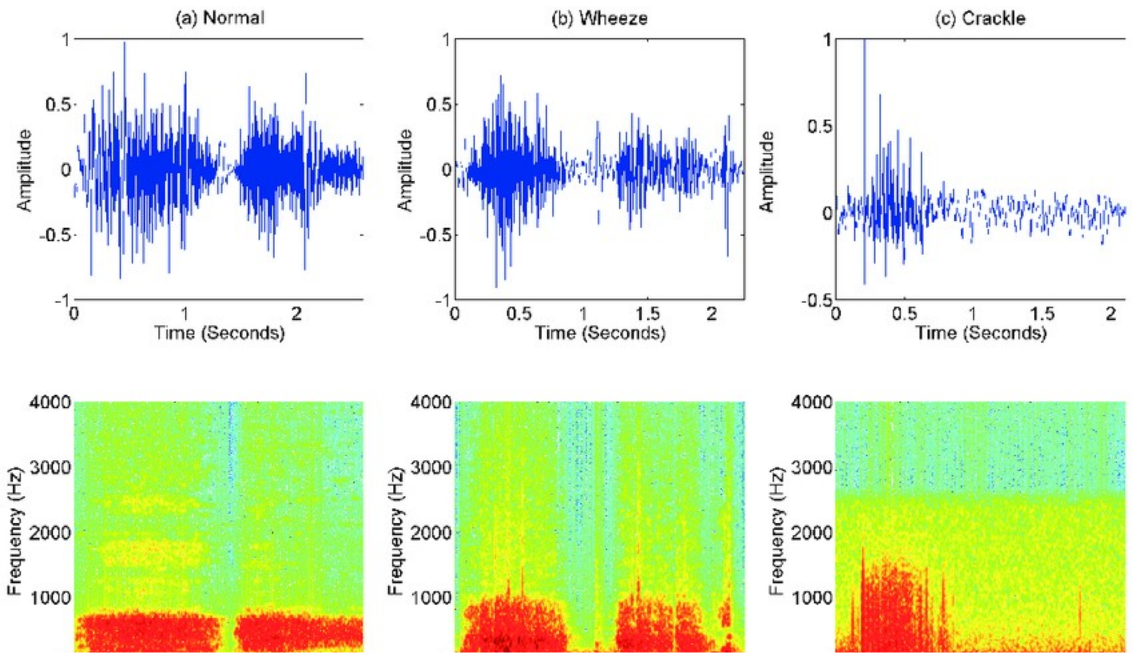

Figure 2 - Representation of different respiratory sounds and the respective spectrogram. [37]

Although the first algorithms used spectrogram analysis, fast Fourier transform and wavelet transform, nowadays, the state of the art in this field involves using machine learning techniques, such as deep neural networks, to classify respiratory sounds into different categories. Some relevant articles are reported below:

Petmezas et al. [17] proposed an algorithm that implements a trained focal loss function (FL) after features are extracted from a short-time Fourier transform (STFT) via a convolutional neural network (CNN). This algorithm aimed to detect four states for the sound: normal, crackle, wheeze, crackle and wheeze, and it was trained and tested on a respiratory sound database developed within the scope of the International Conference on Biomedical and Health Informatics (ICBHI) 2017.

The data was divided using three different splitting strategies with the following results: achieving sensitivity between 47.37% and 60.29%, specificity between 82.46% and 84.26%, a score between 64.92% and 68.52%, and accuracy between 73.69% and 76.39%.

Jizuo Li et al. [18] proposed a deep learning architecture algorithm to detect four states for the sound - normal, crackle, wheeze, crackle and wheeze - named LungAttn using a feature extraction method based on dual-tunable Q-factor wavelet transform and STFT. Implementing this algorithm in the ICBHI 2017 database achieved a sensitivity of 36.36%, a specificity of 71.44% and a score of 53.90%.

Yoonjoo Kim et al. [19] developed a predictive model of abnormal respiratory sounds with four categories: normal, crackles, wheezes, and rhonchi. The detection of adventitious sounds had an accuracy of 86.5% and an area under the ROC curve (AUC) of 0.93. The further classification into the four categories had an overall accuracy of 85.7% and a mean AUC of 0.92.

These algorithms were developed based on stethoscope-recorded sounds, and their accuracy dealing with smartphone-recorded sounds is still unknown.

The AIRDOC project aimed to develop an innovative solution for the early detection of adventitious sounds in the lungs using smartphones, developing a mobile application to record and analyse lung sounds [20]. The project's main objective was to create a low-cost, accessible, and easy-to-use tool that healthcare professionals and patients could use to detect potential lung problems. The project also focused on developing a cloud-based platform for storing and sharing lung sound recordings. This research has led to interesting advancements in the use of smartphones for the early detection of adventitious sounds in the lungs [21].

Respiratory Sound Assessment Toolkit was developed in University of Aveiro with promising results, with the development of a software application that could accurately detect adventitious sounds in the lungs, such as crackles and wheezes, and provide a visual representation of the sound for straightforward interpretation. The application also analysed the sound in real-time, providing immediate feedback to the user [22].

In this Dissertation, a wheeze detection algorithm will be addressed as well as further performance analysis to the implementation of that algorithm in a database with stethoscope and smartphone respiratory sounds recordings to deduce some conclusions of its applicability.

2 Signal Processing Methods

Mathematical methods are pivotal in implementing algorithms by providing a solid foundation and a deep understanding of the underlying principles. In this chapter, some definitions, and methods relevant for this Dissertation are briefly presented.

2.1 Window function

A window function is a mathematical function that is zero-valued outside of some chosen interval, usually symmetric around the middle where it achieves a maximum of the interval and also tapering away from the middle.

2.2 Hann function

The Hanning function is a window function is used to perform Hann smoothing. The function, with length L and amplitude $1/L$ defined as:

$$w_0(x) \triangleq \begin{cases} \frac{1}{L}\left(\frac{1}{2}+\frac{1}{2}\cos\left(\frac{2\pi x}{L}\right)\right) = \frac{1}{L}\cos^2\left(\frac{\pi x}{L}\right), \text{if } |x| \leq L/2 \\ 0, \text{if } |x| > L/2 \end{cases}$$

By applying the Hann window to a signal before computing its Fourier transform the signal is multiplied by the window function elementwise. This means the signal will be modulated with the shape of the Hann window. The result is that the signal smoothly tapers to zero at its edges, reducing the side lobes and improving the accuracy frequency component estimation in the Fourier domain. This phenomenon can be seen in the chart below.

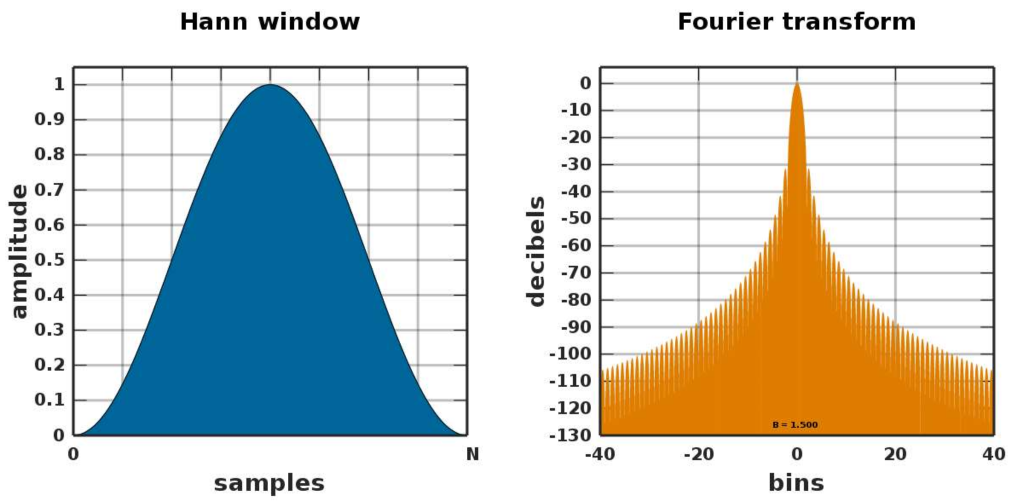

Chart 1 - Hann window and correspondent Fourier transform.

2.3 Box filtering

The box filter equally weights all samples within a square region of the image. Box filtering involves replacing each pixel of an image with the average in a box. When extended in several simple ways, it becomes an efficient general-purpose tool for image processing [23].

When applied to signal, a box slides over the signal, averaging the values within the box at each step. This process continues until the end of the signal.

The filtered signal is obtained by replacing the centre values in each box with the average of the values within that box.

The result is a smoothened signal that reduces the high-frequency noise, giving us a clearer representation of the underlying sinusoidal wave. An example of this method can be seen in Figure 3.

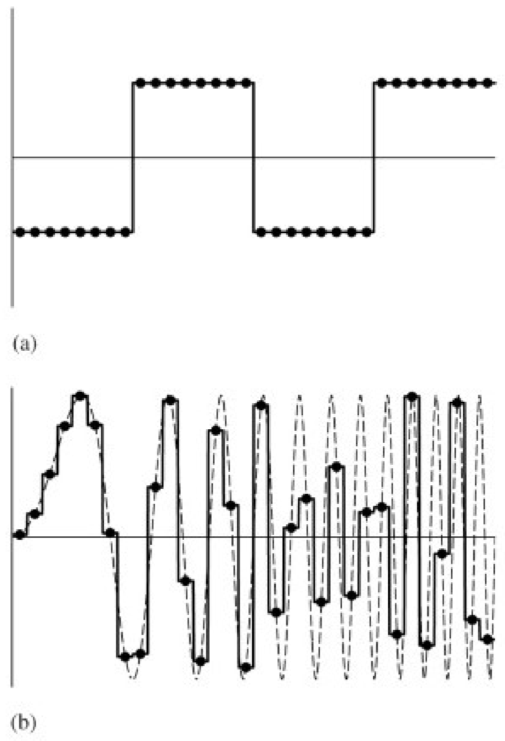

Figure 3 - Box filter applied (a) a step function and (b) a sinusoidal function. [24]

The box filter reconstructs (a) a step function and (b) a sinusoidal function with increasing frequency as x increases. As expected, this filter does well with the step function and does an inferior job with the sinusoidal function [24].

2.4 Convolution

Convolution is a mathematical way of combining two signals to form a third signal. It is the single most important technique in Digital Signal Processing. Convolution is essential because it relates the input, output, and impulse response of a system.

Abstractly, a convolution is defined as a product of functions f and g that are objects in the algebra of Schwartz functions in $\mathbb{R}^n$. The convolution of two functions f and g over a finite range $[0, t]$ is given by

$$[f * g] = \int_0^{\tau} f(\tau) g(t - \tau) d\tau$$

When taking over an infinite range,

$$f * g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

A visual way to understand the convolution is shown in Figure 4:

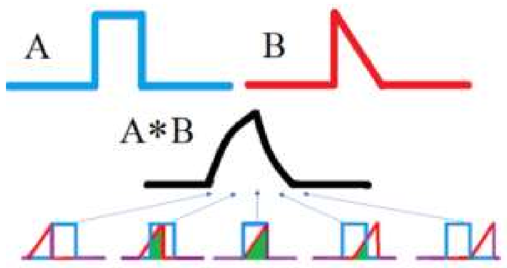

Figure 4 - Convolution of two functions A (red) and B (blue) produce a third function describing the overlap (green).

2.5 Savitzky-Golay filter

The Savitzky–Golay smoothing, and differentiation filter optimally fits a set of data points to a polynomial in the least-squares sense.

The smoothing is achieved by fitting successive subsets of adjacent data points with a low-degree polynomial by the method of linear least squares.

3 Adventitious respiratory sounds automatic detection

This chapter delves into advanced algorithms for identifying abnormal respiratory sounds. It will provide detailed insights into the underlying computational methods and signal processing techniques used to identify adventitious sounds.

3.1 Wheezes detection

The algorithm to detect wheezes was initially developed by Taplidou in 2007 [25]. It aims to locate wheezing episodes in breath sound recordings based on time-frequency (TF) analysis of the signal.

The TF representation is usually obtained using the STFT on a time signal $s(t)$:

$$F_{STFT}(\tau, f) = \int_{+\infty}^{-\infty} s(t) \cdot h^{*}(t-\tau) e^{-j2\pi f}\, dt$$

Equation 1

The use of the STFT for wheeze detection has been already proposed by M. Waris et al. [26] but it was insufficient in the presence of noise, such as cardiac sounds.

Therefore, Taplidou et al. developed this time-frequency wheeze detector (TF-WD) using five steps:

Step 1: Calculation of the TF representation of the recorded sound.

Step 2: The underlying basic breath sound is subtracted from the total breath sound.

Step 3: Peak detection in the TF plane.

Step 4: Classification of the detected peaks as wheeze and non-wheeze.

Step 5: Wheeze representation in the time domain.

Description of the wheezes detection algorithm

Step 1

Consider the signal $s(t)$ as

$$s(t) = w(t) + n(t)$$

where $w(t)$ and $n(t)$ correspond to wheezing and white noise sounds, respectively.

The $s(t)$ signal is analysed in the TF plane using **Equation 1**, with $h(t)$ being a 256-sample Hanning sliding window shifted by 128 overlapping samples in the time domain; at each window position, the fast Fourier transform of the windowed signal (extended to 2048 samples after zero-padding) is estimated, resulting in the $F_{STFT}(\tau, f)$.

Step 2

To remove the basic breath sound from the sample, a smoothing procedure is used to estimate the trend of the frequency content of the windowed signal at each time instance. This estimation is based on the Box filtering procedure, which reduces the variation of a signal, a commonly used technique to reduce white noise.

Step 3

The frequency axis is segmented into four frequency bands: B_1: 60–300 Hz, B_2: 300–600 Hz, B_3: 600–1400 Hz, and B_4: 1400–1900 Hz, so that different magnitude criteria can be defined for each one to facilitate the peak detection in the TF domain. This detection method involves a search for peaks that overpass a specific magnitude threshold, nT, (in dB) per frequency band: nT_1: 1.8, nT_2: 2, nT_3: 3 and nT_4: 3.

When these peaks are identified, their time and frequency location are marked. This procedure is repeated for each time window, resulting in $F_{STFT}^{SP}(\tau, f)$.

Step 4

A set of criteria is applied to examine the validity of the detected peaks at step 3. These criteria include:

1) Local maxima: the peaks should have the maximum magnitude of a 70Hz frequency window sliding over the frequency bands at each time-window position.
2) Peak coexistence: the number of peaks coexisting at each time instance should not be greater than four.
3) Continuity in time: peaks should have a duration greater than 150ms.
4) Grouping: peaks are part of the same wheeze when the frequency proximity of peaks that belong to successive time-window positions is no mere than 50 Hz;

their time proximity is no more than 23.2 ms, and the total duration of the gradually formed wheeze does not exceed 2.5 s.

The detected peaks in the $F_{STFT}^{SPC}(\tau, f)$ that satisfy the criteria described above correspond to wheezes; if not, they are considered non-wheezing sounds and are discarded.

Step 5

By applying the inverse short-time frequency transform (ISTFT), the $w(t)$ signal is estimated, showing the wheeze information in the time domain.

This algorithm was used to detect wheezes in stethoscope-recorded sounds from patients with stable and exacerbated COPD. [27], [28]

3.2 Crackles detection

The algorithm to detect crackles was initially developed by Laura Vannuccini in 1998 [29]. This automatic method detects peaks corresponding to the points of the maximum slope. The respiratory sound's first derivative (FD) is calculated to emphasise pitch and speed, and the crackle should be detected using the deflection property. So, this algorithm is generically based on two steps:

Step 1: Locate the zone of interest using the lung sound's first derivative absolute value (FDAV).

Step 2: Verify if the zones detected in the first step fulfil all requisites to consider it a crackle.

Description of the crackles detection algorithm

Step 1

Apply the FDAV to the sample and compare it to a predefined threshold, *T*. The intervals t_{on} and t_{off} are calculated using this comparison: t_{on} is the time during which FDAV remains over *T,* and t_{off} is the time it remains under *T*:

$$t_{\mathrm{on}} = \sum_i t_{\mathrm{on}_i} \quad , \qquad t_{\mathrm{off}} = \sum_i t_{\mathrm{off}_i}$$

This counting continues until t_{on} is greater than t_{off} and the sum of these two variables is less than the temporal window T_W. When the counting stops, a zone of interest is found.

<u>Step 2</u>

To verify if the zone of interest found contains a crackle, these conditions must be achieved:

1) At least three close peaks of FDAV in TW must exist. This condition is easy to verify since the number of peaks is equal to the number of T_{on_i}. The positions of these peaks correspond to the points of the maximum slope of the signal (t_1, t_2 and t_3).
2) FDAV(t_2) must be greater than FDAV(t_1) and FDAV(t_3).
3) The intervals $t_1 - t_S$, $t_2 - t_1$ and $t_3 - t_2$ must increase because:

$$t_1 - t_S > t_2 - t_1 > t_3 - t_2$$

where t_S is the starting point.

To know the variable t_S, some parameters of the crackle are helpful, such as the initial deflection width (IDW).

A finite impulse response (FIR) filter evaluates the FD. These kinds of filters belong to the Savitsky–Golay (SG) group and are low-pass filters, well adapted for smoothing and differentiation, whose properties are defined in the time domain rather than the Fourier domain. Specifications for an SG filter are the number of coefficient *n*, the degree of the fitting polynomial *p*, the order of derivation *d* and the order of the higher moment to preserve *m*.

Similar to wheezes detection algorithm, this crackles detection algorithm was used previously in stethoscope-recorded sounds from patients with stable and exacerbated COPD. [27], [28]

4 Database description

This database contains recordings from 26 young students with ages between 10 and 13 years. Recordings were acquired with a smartphone (iPhone 7, MN902ZD/A, iOS 12.1.2) and one stethoscope (Littmann® model 3200, 3M, Cerritos, CA, USA) at 7 locations simultaneously, totalling 415 respiratory sounds.

The characterisation of the sample is presented below:

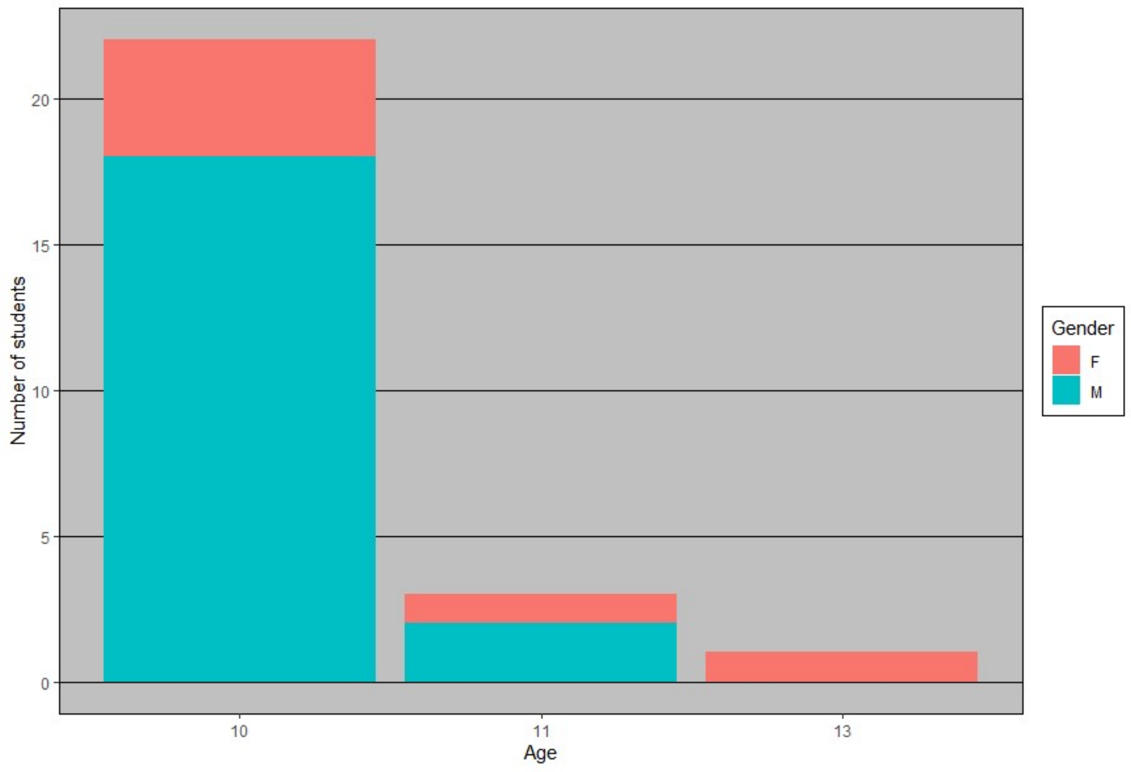

Chart 2 - Number of students by age and gender (n=26).

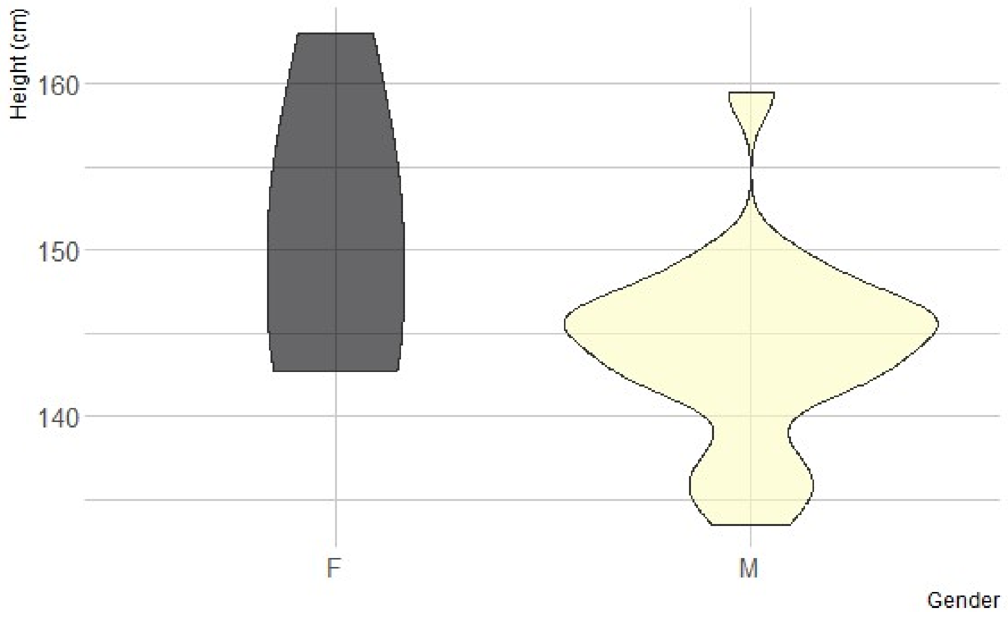

Chart 3 - Box plot (violin chart) of height in cm by gender.

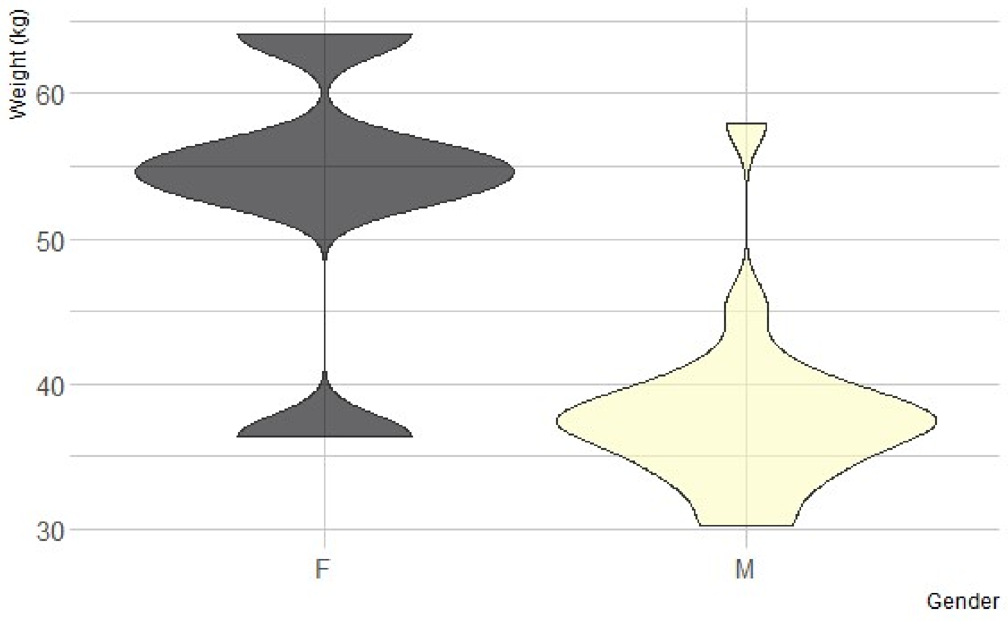

Chart 4 - Box plot (violin chart) of weight in kg by gender.

Regarding height and weight, the data shows some symmetry around the mean values for weight and height which indicates that the sample can be from a normal population.

Each sound file has a duration of thirty seconds and were obtained from 7 different locations:

- Anterior right lower lobe
- Anterior left lower lobe
- Posterior right lower lobe
- Posterior left lower lobe
- Posterior right upper lobe
- Posterior left upper lobe
- Trachea

Due to the simultaneous recording between the smartphone and stethoscope, the data was balanced regarding local and equipment.

Two researchers independently classified the recordings in terms of sound quality (yes/no) and presence of adventitious sounds (yes/no) was only assessed in the recordings with quality. This created a slight discrepancy between the number of recordings per location (Table 1). In the event of disagreement, the recordings were classified by a third investigator and the final decision was determined by the majority rule.

Table 1 - Number of recordings per location.

	Anterior		**Posterior**				**Trachea**
	Right Lower	**Left Lower**	**Right Lower**	**Left Lower**	**Right Upper**	**Left Upper**	
Smartphone	25	25	25	23	25	25	26
Stethoscope	24	25	24	24	21	22	25
Total	49	50	49	47	46	47	51

The presence of adventitious sounds, as annotated by the specialists, is presented in the table 2 below.

Table 2 - Distribution of Wheezes and Crackles presence in the sample.

	Smartphone	Stethoscope	Total
Wheezes	16	17	33
No-Wheezes	158	148	306
Crackles	0	1	1
No-Crackles	174	164	338

As the sounds with wheezes presence represent only 9.73% of the total sample, this database can be considered unbalanced, and the performance analysis must take it into consideration. Regarding the crackle’s presence, it only represents 0.29% of the sample, much less than wheezes. This data is not adequate to evaluate automatic crackle detection.

5 Python Implementation

MATLAB, short for MATrix LABoratory, is a powerful and versatile programming environment and numerical computing software widely used in academia, industry, and research, where all variables are multidimensional arrays, no matter what type of data.

In February 2014, both algorithms described in the previous chapter were implemented in MATLAB by a team in the University of Aveiro, taking advantage of digital signal processing packages in the platform. These algorithms were then integrated in a computer-assisted learning tool which simultaneously allows the recording and analyses of respiratory sounds to be used by healthcare providers. [30]

Later in this Dissertation, the results of applying the algorithms to sounds recorded from different equipments are discussed.

Python is a versatile and high-level programming language that has gained widespread popularity among developers and data scientists due to its simplicity, readability, and extensive libraries. Guido van Rossum initially created Python in the late 1980s [31], and it has since evolved into a powerful tool for various applications, from web development to scientific research and machine learning.

One of Python's standout features is its rich standard library, which includes modules for various tasks, from handling files and working with data to creating graphical user interfaces. Additionally, Python's active open-source community has contributed to an extensive ecosystem of third-party libraries and frameworks, such as NumPy for numerical computing [32] and TensorFlow for machine learning [33].

Targeting the implementation of these algorithms in a cloud server, Python was the chosen language to translate the previous MATLAB algorithm directly.

Due to the vast difference in syntax between both languages, this implementation showed lots of challenges to overcome, having also in interest the computational economy needed for this kind of project.

The Python version used in this implementation was 3.11. The packages used in the implementation are described in the table below.

Table 3 - Version of the packages used in Python's implementation.

Package	Version Control
`cv2`	4.8.0
`numpy`	1.25.2
`pandas`	2.1.0
`scipy`	1.11.2
`soundfile`	0.12.1

Strategy

Translating an algorithm from MATLAB to Python is a challenging task, as these two languages have significant differences in their syntax and operation. To bypass this problem, a proper strategy to approach this difficulty was made, making it possible to efficiently perform this conversion.

Step 1: Understand the MATLAB Algorithm

Before beginning Python's implementation, it's crucial to have a comprehensive understanding of the algorithm in MATLAB. This was made not only from the algorithm's documentation but also from knowledge of digital signal processing.

Step 2: Choose the right Python Libraries

Python has an extensive collection of libraries that can replace MATLAB's functions and features. Depending on the algorithm, some needed libraries might be for linear algebra (e.g., *NumPy*), data processing (*pandas*), visualisation (*Matplotlib*), among others.

Step 3: Map Variables and Data Types

In MATLAB, variables are dynamically typed, while in Python, they are statically typed. Mapping MATLAB variables to their corresponding data types in Python is essential to grant the biggest equivalence between implementations. For example, a matrix in MATLAB can be mapped to a list of lists in Python, where each list corresponds to a matrix row.

Step 4: Translate Flow Control Structures

Translation flow control structures like loops (for, while) and conditionals (if, else) from MATLAB to Python. A particularity is that Python uses indentation to define code blocks, unlike MATLAB, which uses "end" to delimit these blocks.

Step 5: Convert Functions and Operations

Translation of MATLAB-specific functions and operations to their Python equivalents. For example, if the *zeros* function in MATLAB is used to create an array of zeros, in Python, it can be used *numpy.zeros*.

Step 6: Recall Differences in Indexing and adjust parameters accordingly

One of the main difficulties is that in MATLAB, array indexing starts at 1, while in Python, it starts at 0. Also, when defining intervals, MATLAB assumes the input as a closed interval, while Python assumes the first input as belonging to the interval and the last as a majorant.

Step 7: Test and Debug

After the initial translation, it is essential to test the Python code to ensure it produces results equivalent to MATLAB. Using known test cases, it is possible to step by step verify the results it produces. Some Python debugging tools like the *pdb* module also help detect and fix errors and ensure the code works as expected.

Step 8: Code Optimisation

As the main goal is to implement the algorithm in a cloud-based server, optimising the Python code is essential to improve performance or leverage specific language features, using fewer resources and increasing the response time.

Step 9: Comments

Commenting on the Python code properly explains complex parts and provides information about input, output, and the algorithm's logic, making it easier to understand.

Step 10: PEP 8 – Style Guide for Python

PEP 8, "Python Enhancement Proposal 8," is a style guide for writing Python code. It provides a set of conventions and guidelines for formatting and structuring Python code to make it more readable and consistent. By following it, it is ensured that the code remains aligned with modern best practices.

These few steps were aligned to make an organised code translation, considering the particularities of each programming language.

Wheeze Detection Algorithm translation

Step 1

The first step, as mentioned before, calculates the TF of the recorded sound. The decimation frequency is set to 5512Hz, and a bandpass filter to the interval 60Hz – 2100Hz is done through the auxiliary function *butter_filter*, returning the filtered signal. This function made use of the package *signal* for the function *butter* and was defined as:

```
def butter_filter(x, n, fc, fs, t):
    # INPUT parameters:
    #   x - signal to be filtered
    #   n - order of the filter
    #   fc - cut frequency
    #   fs - taxa de amostragem do sinal
    #   type - "high" or "low"

    if t == "high":
        (b, a) = signal.butter(n, fc/(fs/2), "highpass")
    elif t == "low":
        (b, a) = signal.butter(n, fc/(fs/2), "lowpass")
    else:
        print("ERROR: Filter Type unknown.")
    return signal.lfilter(b, a, x)
```

Then, to downsample, the signal was decimated. Ideally, the package signal could be used again, as in the snippet below.

```
ratio = Fraction(Decimal(str(fs / fs_raw)))
y_filt = signal.resample_poly(y_filt, ratio.numerator,
ratio.denominator)
```

Due to computational memory usage, this code could not be used, and a decimation function needed to be defined. It took advantage of *numpy* properties. Please note that *numpy* functions are preceded by the prefix “np.”.

```
def decimation(signal, input_fs, output_fs):

    scale = output_fs / input_fs
    n = round(len(signal) * scale)

    resampled_signal = np.interp(
        np.linspace(0.0, 1.0, n, endpoint=False),
        np.linspace(0.0, 1.0, len(signal), endpoint=False),
        signal,
    )
    return resampled_signal
```

Finally, the STFT can be calculated applying the Hanning Sliding Window to the signal.

```
nfft = 512
noverlap = round(nfft * 0.9)
vf, vt, y_spec = signal.stft(y_filt, fs, "hann", noverlap=noverlap,
nperseg=nfft, nfft=nfft)
```

<u>Steps 2 and 3</u>

To estimate the wanted trend, it is used a box filtering procedure. The below code snippet shows the way it was implemented. Let *c* be the number of columns in the object *y_spec*.

```
for i in range(0, c-1, 2):
    t_window = [z[i:i + 2] for z in y_spec]
    t_mean = [statistics.mean(z.real) for z in t_window]

    # BOX FILTERING
    M_boxfilt = 10
    nn = M_boxfilt * 2
    tt_mean = [t_mean[0] for _ in range(nn)] + t_mean + [t_mean[-1]
for i in range(nn)]
```

```
    t_filt = signal.lfilter([1 for _ in range(M_boxfilt)], 1, 
tt_mean)

    D = round(M_boxfilt / 2)
    # Divide by  M_boxfilt to obtain the averaging filter
    tt_filt = [i / M_boxfilt for i in t_filt[D + nn:-1 - nn + D + 
1]]

    for z in range(len(M_det)):
        M_det[z][i] = y_spec[z, i] - tt_filt[z]
        M_det[z][i + 1] = y_spec[z, i+1] - tt_filt[z]

# DELETE VALUES less than 0
for i in range(len(M_det)):
    for j in range(len(M_det[0])):
        # Delete the null complex part of the cells.
        M_det[i][j] = M_det[i][j].real
        if M_det[i][j] < 0:
            M_det[i][j] = 0
```

The frequency axis is then segmented into four bands, and a vector of threshold values is created, as explained in the algorithm's third step. To detrend the bands, an auxiliary function is defined: *f_detrended*.

```
def f_detrended(detrend_1band, nThreshold):
    detrend_1band_temp = list(map(list, zip(*detrend_1band)))
    mean_band = [statistics.mean(x) for x in detrend_1band_temp]
    std_band = [statistics.stdev(x) for x in detrend_1band_temp]

    vThreshold = [x+y*nThreshold for x, y in zip(mean_band, 
std_band)]

    for j in range(len(detrend_1band[0])):
        for i in range(len(detrend_1band)):
            if detrend_1band[i][j] <= vThreshold[j]:
                detrend_1band[i][j] = 0

    detrend_1band_peak = detrend_1band

    return detrend_1band_peak
```

This function uses the package *statistics* to calculate the means and standard deviations needed.

Step 4

The detected peaks are stored in a data frame using the package *pandas*. The columns' names are: 'obj_sizes', 'tstart', 'tstop', 'duration', 'valid_obj', 'flag' and 'indices'.

In the last step, the identified peaks are evaluated to a set of criteria. If a peak does not fulfil the criteria, it is marked as 0 in the *valid_obj* columns, and then this row is eliminated. In the end, the rows of the data frame only contain valid wheezes.

One of the limitations of Python compared to MATLAB is the lack of matrix-ready packages and specific capabilities like finding indices of an element and the opposite: finding the row and column of and given a specific index. To bypass these difficulties, the approach was to maintain the original MATLAB design as a data frame and create two auxiliary functions.

```
def f_findIndices(Acc, N):
    temp = []
    for j in range(len(Acc[0])):
        for i in range(len(Acc)):
            temp.append(Acc[i][j])

    IND = np.nonzero(temp)[0].tolist()
    Acc_short = [temp[i] for i in IND]

    M_indices = []
    for k in range(1, N):
        temp2 = [i for i, x in enumerate(Acc_short) if x == k]
        M_indices.append([IND[i] for i in temp2])
    return M_indices
```

```
def ind2sub(size, ind):
    row = []
    col = []

    for z in ind:
        row.append(z % size[0])
        col.append(int(z/size[0]))

    return row, col
```

All these functions described in this chapter were essential to translate the code from MATLAB. It allowed us to maintain the same line of thought while making this algorithm available to be implemented in all platforms due to the versatility of Python.

6 Performance analysis

6.1 Preliminary Analysis

All the statistical analysis was done in R, version 4.3.1, using the package `caret` version 6.0-94.

To evaluate the performance of the algorithm over for smartphone recording adventitious sound detection, we first applied the MATLAB algorithm in the smartphone recordings database and compared it with the specialists' annotations.

Table 4 - Performance of Wheeze and Crackle Detection algorithms.

	Wheezes	**Crackles**
Accuracy	0.631	0.015
Kappa	0.187	1e-04
Sensitivity	0.879	1.000
Specificity	0.604	0.012
Precision	0.193	0.003
F_1 Score	0.317	0.006

The accuracy of the crackle's detection algorithm is much worse than the wheezes one. Cohen's Kappa also shows that the annotations are very different from the result of the implementation.

Considering the crackles' poor results and due to time limitations, only the wheeze detection algorithm was translated into Python language, aiming to achieve, in the worst-case scenario, an equivalent algorithm of this already satisfactory method.

Considering the actual values as the MATLAB output, the Python ones as the predicted class and implementing the metrics of Appendix I – Methodological Foundations, the results are shown in Table 5.

Table 5 - Python vs MATLAB.

Accuracy	0.817
Kappa	0.629
Sensitivity	0.787
Specificity	0.841
Precision	0.797
F_1 Score	0.792

Cohen’s Kappa of 0.629 shows a reasonable similarity between the MATLAB and Python algorithms. Due to the different syntax and filters used, it was expected not to have a completely equal algorithm, but one somehow equivalent.

With 81.7% accuracy, Python’s algorithm is equivalent to MATLAB results, as expected. Indeed, high accuracy requires high precision and trueness, following the International Organization for Standardization nomenclature. Even so, there are differences between both, and a deeper examination of the results was needed.

6.2 Performance against specialists' annotations

Considering specialists' annotations as ground truth, the predicted results from the two implementations were validated against them.

Table 6 - Comparison of performance metrics between MATLAB and Python.

	Annotation vs MATLAB	Annotation vs Python
Accuracy	0.631	0.643
Kappa	0.187	0.205
Sensitivity	0.879	0.909
Specificity	0.604	0.614
Precision	0.193	0.203
F_1 Score	0.317	0.331

As concluded from Table 2, the data is unbalanced, and precision and sensitivity might be biased because of that characteristic. On the other hand, the F_1 Score is more stable for this data set and should be the proper parameter to analyse. As the F_1 Score is greater for Python's implementation, we can conclude that it has better precision and recall, meaning fewer errors when detecting wheezes.

Accuracy is not the best parameter to consider because the sample is unbalanced. Alternatively, there is another approach using Balanced Accuracy.

Definition – Balanced Accuracy
Balanced accuracy, BA, is the arithmetic mean of sensitivity and specificity.

For Annotations *versus* MATLAB, the Balanced Accuracy is 74.17%, and Annotations *versus* Python is 76.17%. This result shows an increase of 2% in BA for the Python implementation, meaning that the number of correctly detected wheezes out of all recordings is more significant for Python.

The difference between equipments may also affect the detection of wheezes. In fact, considering that one equipment, the stethoscope, is made precisely to perform medical auscultation and smartphone not, it is expected to have noticeable differences. To enlighten possible differences, some more measurements were made.

Table 7 - Metrics for MATLAB and Python implementations *versus* Annotations taking the equipment into account.

		Sensitivity	Specificity	BA	Precision	F_1 Score
MATLAB	Smartphone	0.938	0.709	0.823	0.246	0.389
	Stethoscope	0.824	0.493	0.658	0.157	0.264
Python	Smartphone	0.938	0.722	0.829	0.254	0.400
	Stethoscope	0.882	0.500	0.691	0.168	0.283

Surprisingly, the implementation of the algorithms in both languages showed better results for the smartphone subset, and again, Python's implementation is statistically better than in MATLAB, reaching a 0.4 score in the F_1 Score metric. Also, the balanced accuracy for smartphone recordings is very satisfactory, 82% for MATLAB and 1% more in Python.

Another cause that can influence the results is the location where the samples were previously recorded. To address this possibility, some exploratory analysis of correlation was made.

Time of implementation

In MATLAB, the algorithm implementation for the smartphone dataset takes 9.97 minutes; for the stethoscope dataset, it takes 12.54 minutes.

In Python, the algorithm implementation for the smartphone dataset takes 7.55 minutes, as, for the stethoscope data set, it takes 9.49 minutes.

For both cases, Python implementation is around 25% faster than MATLAB's.

7 Concluding Remarks

The smartphone has emerged as a powerful tool in eHealth. Detecting adventitious sounds such as wheezes would revolutionise how we monitor and manage respiratory health outside medical facilities. Its portability, accessibility, and ever-evolving technological capabilities have made it an indispensable companion for better health and well-being.

Furthermore, the smartphone's role extends beyond mere detection; it empowers individuals by providing valuable information and resources for managing their respiratory conditions. From tracking symptoms and medication schedules to connecting with healthcare professionals remotely, smartphones have become an essential lifeline for those living with respiratory issues. By aggregating anonymised data from millions of users, researchers and healthcare providers can gain insights into patterns and trends in respiratory health, ultimately leading to improved treatments and preventive measures.

In this Dissertation, the objective was, as the title says, to analyse respiratory sounds acquired with a smartphone. With two already available algorithms implemented in MATLAB and previously used in clinical studies, we took an additional step to implement them in Python. As the crackles detection algorithm showed significant limitations, there was no time to assess that problem.

During this time, the knowledge of digital signal processing increased due to the contact with diverse methods.

The implemented algorithm in Python had a remarkable performance overall: from better accuracy to faster execution time, proving the value that it has and that with further work, better results may appear, encouraging to continue the research in this field.

7.1 Limitations and achievements

In the development of this Dissertation, some limitations conditioned the progress:

- Python's huge syntax difference compared to MATLAB and the lack of matrix-ready packages delayed the translation more than expected also making the debug period very complicated;
- available signal packages are not sufficiently optimized to these complex projects, being even more difficult to perform digital signal processing in Python;
- the database has some limitations as the same stethoscope and smartphone were used, created a small bias in the results interpretations;
- the sample was small and only included children, more specifically for recordings with crackles.

Despite these limitations, interesting achievements were made:

- A Python equivalent algorithm to MATLAB implementation was written;
- Statistical analysis shows even greater performance on Python's algorithm;
- When applied to smartphone data, both precision and balanced accuracy increases around 10%.
- Execution time in Python is 25% faster than in MATLAB.

These achievements increase expectations about the future of automatic detection of adventitious sounds as current algorithms already returns satisfactory results.

7.2 Future work

Despite the work developed in this Dissertation allowing an exciting discussion about automatic adventitious sound detection algorithms in stethoscopes and smartphone databases, this topic is far from being finished.

Wheezes detection algorithm needs to be tested in more extensive databases with different age groups to investigate more about its performance and allow, with that implementation, to tune this method.

The same should be done with the crackles detection algorithm, although the performance in MATLAB was inferior.

Once this research is done, it would be possible to implement these algorithms in a cloud-based server and make them available to the population.

8 References

[1] A. Roguin, "Rene Theophile Hyacinthe Laënnec (1781-1826): The man behind the stethoscope," *Clin Med Res*, vol. 4, no. 3, pp. 230–235, 2006, doi: 10.3121/cmr.4.3.230.

[2] F. Weinberg, "The history of the stethoscope," *Canadian Family Physican*, vol. 39, pp. 2223–2224, 1993.

[3] S. Núñez, A. Moreno, K. Green, and J. Villar, "The stethoscope in the emergency department: A vector of infection?," *Epidemiol Infect*, vol. 124, no. 2, pp. 233–237, 2000, doi: 10.1017/S0950268800003563.

[4] E. G. M. Cox *et al.*, "Should the ultrasound probe replace your stethoscope? A SICS-I sub-study comparing lung ultrasound and pulmonary auscultation in the critically ill," *Crit Care*, vol. 24, no. 1, pp. 1–7, 2020, doi: 10.1186/s13054-019-2719-8.

[5] F. Belloni, D. Della Giustina, M. Riva, and M. Malcangi, "A new digital stethoscope with environmental noise cancellation," *International Conference on Mathematical and Computational Methods in Science and Engineering - Proceedings*, no. May 2014, pp. 169–174, 2010.

[6] T. Grzywalski *et al.*, "Practical implementation of artificial intelligence algorithms in pulmonary auscultation examination," *Eur J Pediatr*, pp. 883–890, 2019, doi: 10.1007/s00431-019-03363-2.

[7] S. Dramburg, E. Dellbrügger, W. van Aalderen, and P. M. Matricardi, "The impact of a digital wheeze detector on parental disease management of pre-school children suffering from wheezing—a pilot study," *Pilot Feasibility Stud*, vol. 7, no. 1, pp. 1–11, 2021, doi: 10.1186/s40814-021-00917-w.

[8] M. A. Batista and S. M. Gaglani, "The future of smartphones in health care," *Virtual Mentor*, vol. 15, no. 11, pp. 947–950, 2013, doi: 10.1001/virtualmentor.2013.15.11.stas1-1311.

[9] H. Ferreira-Cardoso *et al.*, "Lung auscultation using the smartphone—feasibility study in real-world clinical practice," *Sensors*, vol. 21, no. 14, pp. 1–15, 2021, doi: 10.3390/s21144931.

[10] J. C. Aviles-Solis *et al.*, “Prevalence and clinical associations of wheezes and crackles in the general population: The Tromsø study,” *BMC Pulm Med*, vol. 19, no. 1, Sep. 2019, doi: 10.1186/s12890-019-0928-1.

[11] R. X. A. Pramono, S. Bowyer, and E. Rodriguez-Villegas, “Automatic adventitious respiratory sound analysis: A systematic review,” *PLoS One*, vol. 12, no. 5, May 2017, doi: 10.1371/journal.pone.0177926.

[12] A. Bohadana, G. Izbicki, and S. S. Kraman, “Fundamentals of Lung Auscultation,” *New England Journal of Medicine*, vol. 370, no. 8, pp. 744–751, Feb. 2014, doi: 10.1056/nejmra1302901.

[13] B. Zimmerman and D. Williams, *Lung Sounds*. Treasure Island (FL): StatPearls Publishing, 2022.

[14] S. Dramburg, E. Dellbrügger, W. van Aalderen, and P. M. Matricardi, “The impact of a digital wheeze detector on parental disease management of pre-school children suffering from wheezing—a pilot study,” *Pilot Feasibility Stud*, vol. 7, no. 1, pp. 1–11, 2021, doi: 10.1186/s40814-021-00917-w.

[15] U. Koehler *et al.*, “Die akute Exazerbation der COPD: Kann die Telemedizin einen Beitrag zur Früherkennung leisten?,” *Deutsche Medizinische Wochenschrift*, vol. 138, no. 16. pp. 837–841, 2013. doi: 10.1055/s-0032-1332982.

[16] J. E. Fergeson, S. S. Patel, and R. F. Lockey, “Acute asthma, prognosis, and treatment,” *Journal of Allergy and Clinical Immunology*, vol. 139, no. 2. Mosby Inc., pp. 438–447, Feb. 01, 2017. doi: 10.1016/j.jaci.2016.06.054.

[17] G. Petmezas *et al.*, “Automated Lung Sound Classification Using a Hybrid CNN-LSTM Network and Focal Loss Function,” *Sensors*, vol. 22, no. 3, Feb. 2022, doi: 10.3390/s22031232.

[18] J. Li *et al.*, “LungAttn: Advanced lung sound classification using attention mechanism with dual TQWT and triple STFT spectrogram,” *Physiol Meas*, vol. 42, no. 10, Oct. 2021, doi: 10.1088/1361-6579/ac27b9.

[19] Y. Kim *et al.*, “Respiratory sound classification for crackles, wheezes, and rhonchi in the clinical field using deep learning,” *Sci Rep*, vol. 11, no. 1, Dec. 2021, doi: 10.1038/s41598-021-96724-7.

[20] R. Almeida *et al.*, *AIRDOC: SMART MOBILE APPLICATION FOR INDIVIDUALIZED SUPPORT AND MONITORING OF RESPIRATORY FUNCTION AND SOUNDS OF PATIENTS WITH CHRONIC OBSTRUCTIVE DISEASE*. 2020.

[21] H. Ferreira-Cardoso *et al.*, "Lung auscultation using the smartphone—feasibility study in real-world clinical practice," *Sensors*, vol. 21, no. 14, pp. 1–15, 2021, doi: 10.3390/s21144931.

[22] J. Dinis, G. Campos, J. Rodrigues, and A. Marques, "RESPIRATORY SOUND ANNOTATION SOFTWARE," in *Proceedings of the International Conference on Health Informatics*, SciTePress - Science and and Technology Publications, 2012, pp. 183–188. doi: 10.5220/0003756301830188.

[23] M. J. McDonnell, "Box-filtering techniques," *Computer Graphics and Image Processing*, vol. 17, no. 1, pp. 65–70, Sep. 1981, doi: 10.1016/S0146-664X(81)80009-3.

[24] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering*. Elsevier, 2017. doi: 10.1016/C2013-0-15557-2.

[25] S. A. Taplidou and L. J. Hadjileontiadis, "Wheeze detection based on time-frequency analysis of breath sounds," *Comput Biol Med*, vol. 37, no. 8, pp. 1073–1083, Aug. 2007, doi: 10.1016/j.compbiomed.2006.09.007.

[26] M. Waris, P. Helistö, S. Haltsonen, A. Saarinen, and A. R. A. Sovijärvi, "A new method for automatic wheeze detection," IOS Press, 1998.

[27] C. Jácome, A. Oliveira, and A. Marques, "Computerized respiratory sounds: a comparison between patients with stable and exacerbated COPD.," *Clin Respir J*, vol. 11, no. 5, pp. 612–620, Sep. 2017, doi: 10.1111/crj.12392.

[28] C. Jácome and A. Marques, "Computerized Respiratory Sounds: Novel Outcomes for Pulmonary Rehabilitation in COPD," *Respir Care*, vol. 62, no. 2, pp. 199–208, Feb. 2017, doi: 10.4187/respcare.04987.

[29] L. Vannuccini, M. Rossi, and G. Pasquali, "A new method to detect crackles in respiratory sounds," *Technology and Health Care*, vol. 6, no. 1, pp. 75–79, Feb. 1998, doi: 10.3233/THC-1998-6109.

[30] J. Semedo *et al.*, “Computerised Lung Auscultation – Sound Software (CLASS),” *Procedia Comput Sci*, vol. 64, pp. 697–704, 2015, doi: 10.1016/j.procs.2015.08.589.

[31] C. Severance, “Guido van Rossum: The Early Years of Python,” *Computer (Long Beach Calif)*, vol. 48, no. 2, pp. 7–9, Feb. 2015, doi: 10.1109/MC.2015.45.

[32] C. R. Harris *et al.*, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020, doi: 10.1038/s41586-020-2649-2.

[33] B. Pang, E. Nijkamp, and Y. N. Wu, “Deep Learning With TensorFlow: A Review,” *Journal of Educational and Behavioral Statistics*, vol. 45, no. 2, pp. 227–248, Apr. 2020, doi: 10.3102/1076998619872761.

[34] A. Banerjee, U. Chitnis, S. Jadhav, J. Bhawalkar, and S. Chaudhury, “Hypothesis testing, type I and type II errors,” *Ind Psychiatry J*, vol. 18, no. 2, p. 127, 2009, doi: 10.4103/0972-6748.62274.

[35] B. Everitt and A. Skrondal, “The Cambridge Dictionary of Statistics,” 2010.

[36] D. Chicco, M. J. Warrens, and G. Jurman, “The Matthews Correlation Coefficient (MCC) is More Informative Than Cohen’s Kappa and Brier Score in Binary Classification Assessment,” *IEEE Access*, vol. 9, pp. 78368–78381, 2021, doi: 10.1109/ACCESS.2021.3084050.

[37] N. Sengupta, M. Sahidullah, and G. Saha, “Lung sound classification using cepstral-based statistical features,” *Comput Biol Med*, vol. 75, pp. 118–129, Aug. 2016, doi: 10.1016/j.compbiomed.2016.05.013.

9 Appendix I – Methodological Foundations

Observational error is the difference between a measured value of a quantity and its actual value. In statistics, an error is not necessarily a "mistake": variability is inherent in measurement results and the measurement process.

The standard statistical model used is that the error has two additive parts:

- Statistical bias always occurs with the same value when we use the instrument in the same way and in the same case.
- A random error may vary from one observation to another due to the independence of the subjects.

Definition – Confusion Matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes.

Example – Binary Confusion Matrix

		Prediction value	
		Yes (PP)	**No (NN)**
Actual value	**Yes (P)**	TP	FN
	No (N)	FP	TN

Where:

- TP: True Positives → hit
- FN: False Negatives → type II error
- FP: False Positives → type I error
- TN: True Negatives → correct rejection

Using the confusion matrix, there are two errors [34] to take some conclusions from:

- Type I error (α-error): occurs if an investigator rejects a null hypothesis that is true in the population → false positives.
- Type II error (β-error) occurs if the investigator fails to reject a false null hypothesis in the population → false negatives.

To analyse the performance of the algorithms, some metrics were used [35], based on the confusion matrix.

For the following definitions, consider NT as the number of elements in the sample.

Definition – Accuracy (ACC)
Accuracy is the degree of closeness of the measured value to the actual value. It considers the true positives and true negatives over the whole sample. $$ACC = \frac{TP + TN}{P + N}$$

Definition – Precision (Positive predictive value - PPV)
A term applied to the likely spread of parameter estimates in a statistical model and measured by the inverse of the standard deviation of the estimator. $$PPV = \frac{TP}{TP + FP}$$

Definition – Sensitivity (Recall / True Positive Rate - TPV)
An index of a diagnostic test's performance is calculated as the percentage of individuals with a disease who are correctly classified as having the disease. $$TPV = \frac{TP}{P}$$

Definition – Specificity (True Negative Rate - TNV)

An index of the performance of a diagnostic test, calculated as the percentage of individuals without the disease who are classified as not having the disease.

$$TNV = \frac{TN}{TN + FP}$$

Definition – F_1 Score

The F_1 Score combines the precision and sensitivity of a classifier into a single metric by taking their harmonic mean.

$$F_1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

Definition – Kappa coefficient

Cohen’s Kappa is a chance-corrected index of the agreement between datasets. Calculated as the ratio of the observed excess over the chance agreement to the maximum possible excess over chance, the coefficient takes the value one when there is perfect agreement and zero when the observed agreement is equal to the chance agreement. Can be written as [36]:

$$\kappa = \frac{2 \cdot (TP \times TN - FN \times FP)}{(TP + FP) \times (FP + TN) \times (TP + FN) \times (FN + TN)}$$

10 Appendix II – Main function

```
# WHEEZES DETECTION ALGORITHM

# UA, CP, Feb2014
# Wheeze detection based on paper Taplidou et al. 2007 (DOI:
10.1016/j.compbiomed.2006.09.007)
# Python implementation by Sousa, João (FCUP - Math Dept., Oct2023)

from fractions import Fraction
from decimal import Decimal
import math
import statistics
import AuxiliarWh
import cv2
import numpy as np
import pandas as pd
import scipy.signal as signal
import soundfile as sf


def taplidou_main(file_fullpath):
    # ~ PRE-PROCESSING ~

    # Decimation frequency
    fs = 5512  # 5012.5; 11025;

    # Read .wav file
    y_raw, fs_raw = sf.read(file_fullpath)

    # Band-pass filtering [60-2100] Hz
    fc1 = 60
    fc2 = 2100

    if fs_raw / 2 < fc2:
        fc2 = fs_raw / 2 - 1

    y_filt = AuxiliarWh.butter_filter(y_raw, 4, fc1, fs_raw, "high")
    y_filt = AuxiliarWh.butter_filter(y_filt, 4, fc2, fs_raw, "low")

    # Decimate the signal
    y_filt = AuxiliarWh.decimation(y_filt, fs_raw, fs)
    nfft = 512
    # window = signal.windows.hann(nfft)      # Hanning window
    # noverlap = [i*0.9 for i in window]      # Overlap (128 - 50%)

    noverlap = round(nfft * 0.9)
    vf, vt, y_spec = signal.stft(y_filt, fs, "hann",
noverlap=noverlap, nperseg=nfft, nfft=nfft)  # returns STFT

    # Select SFTF only until 2000 Hz
    n = next(x for x, val in enumerate(vf) if val >= 2000)
    y_spec = y_spec[:n]
    vf = vf[:n]

    for i in range(len(vf)):
        for j in range(len(vt)):
            y_spec[i][j] = 20 * math.log10(abs(y_spec[i][j]) ** 2)

    dt = vt[1] - vt[0]  # 23.2 ms   ->  Detrend (mean filter)
    df = vf[1] - vf[0]  # 0.25 Hz

    # to guarantee that y_spec has an even number of columns
    (l, c) = (len(y_spec), len(y_spec[0]))

    if c % 2 == 1:
        temp = y_spec[:, -1]
        y_spec = np.column_stack((y_spec, temp))

    (l, c) = (len(y_spec), len(y_spec[0]))

    # Matrix detrended
    M_det = [[0 for _ in range(c)] for _ in range(l)]

    for i in range(0, c-1, 2):
        t_window = [z[i:i + 2] for z in y_spec]
        t_mean = [statistics.mean(z.real) for z in t_window]

        # BOX FILTERING
        M_boxfilt = 10
        nn = M_boxfilt * 2
        tt_mean = [t_mean[0] for _ in range(nn)] + t_mean +
[t_mean[-1] for i in range(nn)]

        t_filt = signal.lfilter([1 for _ in range(M_boxfilt)], 1,
tt_mean)
```

```
        D = round(M_boxfilt / 2)
        # Divide by  M_boxfilt to obtain the averaging filter
        tt_filt = [i / M_boxfilt for i in t_filt[D + nn:-1 - nn + D
+ 1]]

        for z in range(len(M_det)):
            M_det[z][i] = y_spec[z, i] - tt_filt[z]
            M_det[z][i + 1] = y_spec[z, i+1] - tt_filt[z]

    # DELETE VALUES less than 0
    for i in range(len(M_det)):
        for j in range(len(M_det[0])):
            # Delete the null complex part of the cells.
            M_det[i][j] = M_det[i][j].real
            if M_det[i][j] < 0:
                M_det[i][j] = 0

    # SEGMENT 4 FREQUENCY BANDS
    # B1 = 60 - 300 Hz - STD(3)
    # B2 = 300 - 600 Hz - STD(3)
    # B3 = 600 - 1400 Hz - STD(2)
    # B4 = 1400 - 1900 Hz - STD(2)
    # alteration from 2000 to 1900 – due to some problems

    MB = [60, 300, 600, 1400, 1900]
    nMB = [0 for _ in range(len(MB))]

    for i in range(len(nMB)):
        nMB[i] = next(x for x, y in enumerate(vf) if y >= MB[i])

    if nMB[-1] > len(M_det):
        nMB[-1] = len(M_det)

    n1 = sum([x2 - x1 for x1, x2 in zip(nMB, nMB[1:])])
    n2 = c

    Mdet_total_peak = [[0 for _ in range(n2)] for _ in range(n1)]
    Nthreshold = [1.8, 2, 3, 3]

    for j in range(0, c, 2):
        Mdet_b1 = AuxiliarWh.f_detrended([z[j:j + 2] for z in
M_det[nMB[0]:nMB[1]]], Nthreshold[0])
        Mdet_b2 = AuxiliarWh.f_detrended([z[j:j + 2] for z in
M_det[nMB[1]:nMB[2]]], Nthreshold[1])
```

```
        Mdet_b3 = AuxiliarWh.f_detrended([z[j:j + 2] for z in
M_det[nMB[2]:nMB[3]]], Nthreshold[2])
        Mdet_b4 = AuxiliarWh.f_detrended([z[j:j + 2] for z in
M_det[nMB[3]:nMB[4]]], Nthreshold[3])

        Mdet_window_peak = Mdet_b1 + Mdet_b2 + Mdet_b3 + Mdet_b4

        for i in range(len(Mdet_window_peak)):
            Mdet_total_peak[i][j:j + 2] = Mdet_window_peak[i]

    Mdet_total_peak = [[0 for _ in range(c)] for _ in range(nMB[0] -
1)] + Mdet_total_peak  # add the frequency band


    # CLASSIFICATION OF DETECTED PEAKS

    img = np.array(Mdet_total_peak).astype(np.uint8)
    analysis = cv2.connectedComponentsWithStats(img, connectivity=8)
    (N, Acc, values, centroid) = analysis

    Acc_ind = AuxiliarWh.f_findIndices(Acc, N)

    (rA, cA) = (len(Acc), len(Acc[0]))

    min_duration = 0.12

    Mdata = pd.DataFrame(columns=['obj_sizes', 'tstart', 'tstop',
'duration', 'valid_obj', 'flag', 'indices'])

    for k in range(N - 1):
        ind = Acc_ind[k]
        (r, c) = AuxiliarWh.ind2sub([rA, cA], ind)

        if isinstance(ind, int):  # Garantir que todos os elementos
dos indices são adicionados como vetores.
            ind = [ind]

        Mdata.loc[len(Mdata)] = [len(r), min(c) * dt, max(c) * dt,
max(c) * dt - min(c) * dt, 0, k, ind]

        if Mdata.iloc[k, 3] < min_duration:  # Remove a sibilância
caso a duração seja inferior à mínima.
            for i, j in zip(r, c):
                Acc[i][j] = 0
        else:
            Mdata.iloc[k, 4] = 1  # O objeto é válido.

    # DELETE INVALID WHEEZES
    Mdata = Mdata[Mdata.valid_obj > 0]

    # REMOVING OVERLAPING WHEEZES & APPLYING RULES

    min_gap = 2 * dt

    D = [abs(a - b) for a, b in zip(Mdata.tstart.tolist()[1:],
Mdata.tstop.tolist()[:-1])]
    n = [i + 1 for i in range(len(D)) if D[i] <= min_gap]

    temp = []
    for i in range(len(Mdata)):
        temp.append(Mdata.iloc[i, 6])

    for k in n:
        Mdata.iloc[k, 0] += Mdata.iloc[k - 1, 0]  # junta os dois
objetos
        Mdata.iloc[k, 1] = min([Mdata.iloc[k - 1, 1], Mdata.iloc[k,
1]])  # t_start
        Mdata.iloc[k, 2] = max([Mdata.iloc[k - 1, 2], Mdata.iloc[k,
2]])  # t_stop
        Mdata.iloc[k, 3] = Mdata.iloc[k, 2] - Mdata.iloc[k, 1]  #
duração

        temp[k] = temp[k - 1] + temp[k]

        Mdata.iloc[k - 1, 4] = 0

        (r, c) = AuxiliarWh.ind2sub([rA, cA], list(Mdata.iloc[k - 1,
6]))
        for i, j in zip(r, c):
            Acc[i][j] = Mdata.iloc[k, 5]

    Mdata.drop("indices", axis=1)
    Mdata["indices"] = temp


    # DELETE INVALID WHEEZES
    Mdata = Mdata[Mdata.valid_obj > 0]

    N = len(Mdata.valid_obj)
    # RE-NUMBERING WHEEZES
    for k in range(N):
        (r, c) = AuxiliarWh.ind2sub([rA, cA], list(Mdata.iloc[k,
6]))
        for i, j in zip(r, c):
            Acc[i][j] = k

        Mdata.iloc[k, 5] = k

    # POLYPHONIC WHEEZES -> Deciding whether it is polyphonic or
monophonic
    # The last decision is based on the number of time slices where
there is a frequency component greater than 1.8 time
    # lowest frequency.
    # If there are 10% of such time slices, then the patch is said
to be polyphonic.

    Mdata["poly"] = ""
    Mdata["freq"] = ""
    Mdata["freq_max"] = ""
    Mdata["freq_min"] = ""

    for i in range(N):
        (r, c) = AuxiliarWh.ind2sub([rA, cA], Mdata.iloc[i, 6])
        k_min = [[] for _ in range(max(c) - min(c))]
        k_max = k_min
        k_peak = k_min
        poly_flag = [0 for _ in range(max(c) - min(c))]

        for j in range(min(c), max(c)):
            m = j - min(c)

            ind = [i for i, x in enumerate(c) if x == j]
            k_ind = [r[i] - 1 for i in ind]
            if len(ind) != 0:
                k_min[m] = min(k_ind)
                k_max[m] = max(k_ind)

                if k_max[m] > 1.8 * k_min[m]:
                    poly_flag[m] = 1

                pows = [y_spec[r[z]][j] for z in ind]
                pows_trans = np.array(pows).transpose()
                q2 = [np.argmax(i) for i in pows_trans]
                k_peak[m] = [r[z] for z in ind if ind.index(z) in
q2]

        k_peak = [x for xs in k_peak for x in xs]
        k_max = [x for xs in k_max for x in xs]
        k_min = [x for xs in k_min for x in xs]

        Mdata.iloc[i, Mdata.columns.get_loc("poly")] =
sum(poly_flag) > 0.1 * len(poly_flag)
        Mdata.iloc[i, Mdata.columns.get_loc("freq")] =
statistics.mean([i * df for i in k_peak])
        Mdata.iloc[i, Mdata.columns.get_loc("freq_max")] =
max(k_max) * df
        Mdata.iloc[i, Mdata.columns.get_loc("freq_min")] =
min(k_min) * df

        if 960 < Mdata["freq"].iloc[i] < 1100:
            Mdata.iloc[i, 4] = 0
        elif Mdata["freq"].iloc[i] < 120:
            Mdata.iloc[i, 4] = 0
        elif Mdata["freq"].iloc[i] > 1700:
            Mdata.iloc[i, 4] = 0
        elif Mdata["duration"].iloc[i] < 0.130:
            Mdata.iloc[i, 4] = 0
        else:
            Mdata.iloc[i, 4] = 1

        for z in range(len(k_peak)):
            if 0 == k_peak[z]:
                k_peak[z] = statistics.median(k_peak) * df

    # DELETE INVALID WHEEZES
    Mdata = Mdata[Mdata.valid_obj > 0]
    N = len(Mdata)

    Acc_f = [[0 for _ in range(len(Acc[0]))] for _ in
range(len(Acc))]
    (rA, cA) = (len(Acc), len(Acc[0]))

    # Re-numbering WHEEZES

    for k in range(N):
        (r, c) = AuxiliarWh.ind2sub([rA, cA], Mdata.iloc[k, 6])
        for i, j in zip(r, c):
            Acc_f[i][j] = k
        Mdata.iloc[k, Mdata.columns.get_loc("flag")] = k
        Mdata.iloc[k, Mdata.columns.get_loc("tstart")] -= 0.035
        Mdata.iloc[k, Mdata.columns.get_loc("tstop")] += 0.035
        Mdata.iloc[k, Mdata.columns.get_loc("duration")] += 0.07

    return Mdata
```

11 Appendix III – Auxiliary Functions

```
import scipy.signal as signal
import statistics
import numpy as np


def butter_filter(x, n, fc, fs, t):
    # INPUT parameters:
    #   x - signal to be filtered
    #   n - order of the filter
    #   fc - cut frequency
    #   fs - taxa de amostragem do sinal
    #   type - "high" or "low"

    if t == "high":
        (b, a) = signal.butter(n, fc/(fs/2), "highpass")
    elif t == "low":
        (b, a) = signal.butter(n, fc/(fs/2), "lowpass")
    else:
        print("ERROR: Filter Type unknown.")

    return signal.lfilter(b, a, x)


def f_detrended(detrend_1band, nThreshold):
    detrend_1band_temp = list(map(list, zip(*detrend_1band)))
    mean_band = [statistics.mean(x) for x in detrend_1band_temp]
    std_band = [statistics.stdev(x) for x in detrend_1band_temp]

    vThreshold = [x+y*nThreshold for x, y in zip(mean_band,
std_band)]

    for j in range(len(detrend_1band[0])):
        for i in range(len(detrend_1band)):
            if detrend_1band[i][j] <= vThreshold[j]:
                detrend_1band[i][j] = 0

    detrend_1band_peak = detrend_1band

    return detrend_1band_peak
```

```
def f_findIndices(Acc, N):
    temp = []
    for j in range(len(Acc[0])):
        for i in range(len(Acc)):
            temp.append(Acc[i][j])

    IND = np.nonzero(temp)[0].tolist()
    Acc_short = [temp[i] for i in IND]

    M_indices = []
    for k in range(1, N):
        temp2 = [i for i, x in enumerate(Acc_short) if x == k]
        M_indices.append([IND[i] for i in temp2])
    return M_indices


def ind2sub(size, ind):
    row = []
    col = []

    for z in ind:
        row.append(z % size[0])
        col.append(int(z/size[0]))

    return row, col

def decimation(signal, input_fs, output_fs):

    scale = output_fs / input_fs
    # calculate new length of sample
    n = round(len(signal) * scale)
    resampled_signal = np.interp(
        np.linspace(0.0, 1.0, n, endpoint=False),  # where to
interpret
        np.linspace(0.0, 1.0, len(signal), endpoint=False),  # known
positions
        signal,  # known data points
    )
    return resampled_signal
```