Audio Acquisition and Basic Spectrum Analysis of the Signal.

*1st Lab Report Computational Audio, DEEC-FEUP

André de Azevedo Barata
Department of Electrical and Computer Engineering
Faculty of Engineering of the University of Porto
Porto, Portugal
up201907705@edu.fe.up.pt

André Nogueira Soares
Department of Electrical and Computer Engineering
Faculty of Engineering University of the Porto
Porto, Portugal
up201905318@edu.fe.up.pt

***Abstract*—This work presents an overview of two big acoustic phenomena: harmonics and Acoustic Beat. In order to explain, the authors start by analysing the Fourier Transform and spectrogram of a C scale generated by sine waves and then replicate the experience of the same scale but played on a Piano. Afterwards, the authors play a single note on the piano and explain the relationship between the different frequencies present on the sound and the acoustic perception of the human ear. Overall, the results provide insights into the realm of acoustic and aim to explain musical phenomena.**

***Index Terms*—Signal processing, acoustic, spectral analysis**

I. Introduction

The realm of acoustics is a rich and intricate domain that continues to captivate scientists, musicians, and enthusiasts alike. This work embarks on an exploration of these phenomena, guided by an analysis of the Fourier Transform and spectrogram, by analysing the C major scale generated by sine waves and the same scale played on a piano, we'll explain the enrichment that acoustic instruments bring to music at a mathematics level. During this work we'll compute the Fast Fourier Transform and the spectogram of tree different cases: C_scale built with sine waves, C_scale played in a piano and a long note played in the piano. The objective is to observe the influence of the harmonics in the fft and analyse of case of an acoustic beat.

II. Overview of the "Continuous Audio Data Acquisition" Demonstration

This program's main task is to capture continuous audio data in real-time and use Fast Fourier Transform (FFT) graphs to display its frequency content. Here is a brief overview of the main features:

- **Audio Device Setup:** The program begins by identifying and selecting an audio input device, a microphone.
- **Audio Data Acquisition Session:** An audio acquisition session is created. This session is responsible for interfacing with the audio hardware and continuously collecting audio data.
- **Real-time FFT Visualization:** The program initializes a plot to display the FFT (Fast Fourier Transform) of the live audio input signal.
- **Event Handling:** It sets up an event listener for the 'DataAvailable' event. When new audio data is available, the listener triggers a callback function (helper_continuous_fft) to process the data and update the FFT plot in real-time.
- **Data Acquisition Start and Duration:** The program initiates the data acquisition process. It's possible to observe the FFT plot updating dynamically as audio data is acquired. It continues acquiring data for a specified duration, 10 seconds in this case.
- **Data Acquisition Stop and Cleanup:** After the 10 seconds, the program stops the data acquisition process, sets the session for non-continuous operation, and removes the event listener to clean up resources.

III. Exploring Properties and Functions:

A. Device Discovery: daq.getDevices

This is a command used to retrieve information about the available data acquisition (DAQ) devices connected to our computer.

It returns a list of available devices along with their properties such as index, device ID, Description.

```
Data acquisition devices:

index   Vendor    Device ID                    Description
-----  ----------  ---------  ------------------------------------------------------
1      directsound Audio0     DirectSound Controlador de captura de som principal
2      directsound Audio1     DirectSound Conjunto de microfones (Realtek(R) Audio)
3      directsound Audio2     DirectSound Mistura estéreo (Realtek(R) Audio)
4      directsound Audio3     DirectSound Controlador de som principal
5      directsound Audio4     DirectSound Altifalantes (Realtek(R) Audio)
```

Fig. 1: Function daq.getDevices

B. Device Selection: dev

The output provided for the variable dev contains essential information regarding a specific Data Acquisition (DAQ) device.

```
directsound: DirectSound Conjunto de microfones (Realtek(R) Audio) (Device ID: 'Audio1')
   Audio input supports:
      -1.0 to +1.0 range
      Rates from 80.0 to 1000000.0 scans/sec
      2 channels ('1','2')
      'Audio' measurement type
```

Fig. 2: Function dev

Below is an elucidation of the details encompassed within the dev structure:

- **Device Name and Type:**
  Device Name: DirectSound Conjunto de microfones (Realtek(R) Audio)
  Device ID: 'Audio1'
  This line conveys vital information about the DAQ device in question. It is identified as a "DirectSound" device with the name "Conjunto de microfones (Realtek(R) Audio)" and is assigned the unique device identifier 'Audio1'.
- **Audio Input Capabilities:**
  Input Range: -1.0 to +1.0
  This section furnishes insights into the capabilities of the audio input offered by the DAQ device. It specifies the permissible input value range, which extends from -1.0 to +1.0.
- **Supported Sampling Rates:**
  Sampling Rate Range: 80.0 to 1000000.0 scans/second
  The DAQ device is equipped to operate at various sampling rates, facilitating data acquisition at rates spanning from 80.0 to 1000000.0 scans/second.
- **Number of Input Channels:**
  Channels: '1' and '2'
  This entry highlights that the DAQ device has the capacity to handle data acquisition from two separate input channels, designated as '1' and '2'. This capability allows for the simultaneous acquisition of data from these two distinct channels.
- **Measurement Type:**
  Supported Measurement Type: 'Audio'
  This specification delineates the type of measurement supported by the device, which, in this instance, is categorized as 'Audio'. It signifies that the device is optimized for audio data acquisition applications.
  In summation, the dev structure encapsulates comprehensive information about a DAQ device tailored for audio input.

C. Creating a Data Acquisition Session: daq.createSession

To commence data acquisition with the aforementioned DAQ device, a session object is created using the following line of code:

s = daq.createSession('directsound');

This command initializes a data acquisition session named "s" configured to interface with the 'directsound' device. Subsequently, the "s" session object becomes the conduit through which data acquisition operations can be meticulously configured and executed.

```
s =

Data acquisition session using DirectSound hardware:
   Will run for 1 second (44100 scans) at 44100 scans/second.
   No channels have been added.
```

Fig. 3: Function daq.createSession

For example, we use it here:

addAudioInputChannel(s, dev.ID, 1:1);

This code augments the data acquisition session, denoted by the "s" session object, by adding an audio input channel from the specified device with the identifier "dev.ID". The "1:1" argument indicates that a single audio input channel is added, and it designates channel '1' for acquisition. This configuration prepares the session to capture audio data from the selected input channel for subsequent processing or analysis.

D. Utilizing Function Handles (See: Creating Function Handles, Anonymous Functions)

In MATLAB, a handle function is defined using the @ symbol, followed by a list of input arguments and an enclosed expression or block of code. Here's how to discern the expression within a handle function:

- Locate the @ Symbol: Handle functions are created using the @ symbol, signifying the beginning of the function definition.
- Review the Input Arguments: The input arguments of the handle function are listed within parentheses immediately following the @ symbol. These arguments dictate what data the handle function can accept when it is invoked.
- Inspect the Enclosed Code: After the input arguments, it's find the expression or code block encapsulated within curly braces . This code represents the actual computation executed when the handle function is invoked.

For example, in our case:

plotFFT = @(src, event) helper_continuous_fft(event.Data, src.Rate, hp);

Breaking Down the Handle Function:

- Function Name: plotFFT is the name of the handle function, allowing us to reference and call this function later in our code.
- Input Arguments: (src, event) specifies the input arguments. In this case, src represents the source object that triggered the event, and event contains information about the event itself.
- Enclosed Expression: helper_continuous_fft(event.Data, src.Rate, hp) represents the expression in this handle function. It performs continuous Fast Fourier Transform (FFT) analysis.

E. Real-time Fast Fourier Transform: continuous_fft

This function serves as a valuable tool for the continuous computation of the Fast Fourier Transform (FFT) magnitude of a given input signal. This function is particularly useful for analyzing the frequency content of time-domain signals. Below, we provide a detailed breakdown of this function's inputs:

function continuous_fft(data, Fs, plotHandle)

Input Arguments:

- data: The input signal for which the FFT magnitude is to be computed.
- Fs: The sampling frequency of the input signal.
- plotHandle: A handle to the plot that will be updated with the FFT magnitude results.

F. Event Handling with addlistener

In MATLAB, addlistener is a powerful function that allows you to establish event listeners associated with objects. In the context of data acquisition, addlistener is often used to respond to events generated by data acquisition hardware. For example, you can use it to capture and process data as it becomes available from sensors, devices, or instruments in real-time.

In our case, we use it like this:

hl = addlistener(s, 'DataAvailable', plotFFT);

- s: The object to which the event listener is attached. In this example, s represents the data acquisition session, which interacts with data acquisition hardware.
- 'DataAvailable': The event name. This is the specific event the listener is designed to capture. In this case, it's set to 'DataAvailable,' indicating that the listener will respond when new data is available for processing.
- plotFFT: The callback function. When the 'DataAvailable' event occurs (i.e., new data is available for acquisition), the designated callback function, plotFFT, is invoked. This function is responsible for processing and visualizing the incoming data in real-time. It takes arguments such as event.Data (the acquired data), src.Rate (the sampling rate), and hp (a handle to a plot), allowing for dynamic updates of plots or data analysis.

```
listener with properties:

    Source: {[1×1 daq.audio.Session]}
 EventName: 'DataAvailable'
  Callback: @(src,event)helper_continuous_fft(event.Data,src.Rate,hp)
   Enabled: 1
 Recursive: 0
```

Fig. 4: Function addlistener

G. Source and Event Information: src, event

In MATLAB, event-driven programming is vital for real-time data acquisition and analysis, and "src" and "event" are key components in this process.

- "src" is the source object responsible for triggering an event. It identifies where the event originated, crucial when handling multiple event sources. In our case, "src" points to the data acquisition session object "s". When new data arrives, "s" triggers the 'DataAvailable' event
- "event" contains event-specific data, such as acquired data and sampling rate, offering context for event handling. For 'DataAvailable' events, event.Data holds the newly acquired data, and src.Rate provides the sampling rate.

IV. Harmonic and acoustic beat concepts

A. Harmonic in acoustics

According to [1], Harmonics are a series of distinct frequencies that occur above the fundamental frequency when a sound is produced. This phenomenon is vital for understanding the rich and complex sounds we encounter in music, speech, and everyday life. The frequency of the harmonics are given by $f_h = n_i \times f_f$, where f_h is the harmonic frequency, n_i is the number of the harmonic and f_f the fundamental frequency.

In musical instruments, harmonics are responsible for the timbral richness and diversity of sounds. For example, when you pluck a guitar string, it not only produces the fundamental pitch but also a series of harmonics. These harmonics give each musical instrument its unique character and enable us to distinguish between different instruments, even when they play the same note.

B. Acoustic beat

Acoustic beats are a fascinating auditory phenomenon that occurs when two sound waves with slightly different frequencies interact. These interactions lead to a periodic variation in the loudness or intensity of the combined sound, resulting in what we perceive as a "beat." Acoustic beats are commonly heard in music and serve as a crucial tool for tuning musical instruments and exploring the properties of sound. As described in [1], the frequency of the beat (f_beat) can be calculated as the absolute difference between the frequencies of the two interacting waves: $f_{beat} = f_1 - f_2$, where f_1. and f_2 are the two involved frequencies.

Acoustic beats have significant musical applications. such as tuning method, for instance when tuning a guitar, a musician listens for the beats produced between two strings: if the beats are too slow, it indicates the strings are out of tune, and adjustments are made until the beats disappear, indicating that the two strings are in tune with each other. they are also used in other instruments in order to make the sound more robust. We advise you to try to listen to this by producing the file "Acustic_piano_batimento_record_MacOS.wav"

V. Analysing the spectrum of the audio

A. C scale

We started by acquiring the audio data by using the *Data Acquisition Toolbox Support Package for Windows Sound*

Cards. The code used to obtain the data and compute its fft is presented in figure 5, yet we opted for a simpler approach that abstracts some point of the data acquisition process such as selecting the device and works on a variaty of Operative Systems. Therefore, we used the function *audioread* from Matlab, which the code used is presented in figure 6.

```
function continuousFFT(daqHandle, plotHandle)
global accquiredData
% Calculate FFT(data) and update plot with it.
data = read(daqHandle, daqHandle.ScansAvailableFcnCount, "OutputFormat", "Matrix");

Fs = daqHandle.Rate;
lengthOfData = length(data);
lenghOfData_full = length(accquiredData);

% next closest power of 2 to the length
nextPowerOfTwo = 2 ^ nextpow2(lengthOfData);
nextPowerOfTwo_full = 2 ^ nextpow2(lenghOfData_full);

fprintf("tamanha da variavel global: %d\n", length(accquiredData))
accquiredData = [accquiredData;data];
%audiowrite("./test_sound.mp3", accquiredData, Fs);

plotScaleFactor = 4;
% plot is symmetric about n/2
plotRange = nextPowerOfTwo / 2;
plotRange = floor(plotRange / plotScaleFactor);

plotRange_full = nextPowerOfTwo_full / 2;
plotRange_full = floor(plotRange_full / plotScaleFactor);

yDFT = fft(data, nextPowerOfTwo);
yDFT_full = fft(accquiredData, nextPowerOfTwo_full);

espetro = spectrogram(accquiredData);
spectrogram(espetro, "yaxis")

h = yDFT(1:plotRange);
abs_h = abs(h);

h_full = yDFT_full(1:plotRange_full);
abs_h_full = abs(h_full);

% Frequency range
freqRange = (0:nextPowerOfTwo-1) * (Fs / nextPowerOfTwo);
freqRange = (0:nextPowerOfTwo_full-1) * (Fs / nextPowerOfTwo_full);
% Only plot up to n/2 (as other half is the mirror image)
gfreq = freqRange(1:plotRange);
gfreq_full = freqRange(1:plotRange_full);


% Take the logarithm of both x and y data
log_xdata = log10(gfreq);
log_ydata = 20* log10(abs_h);%to put in dB

log_xdata_full = log10(gfreq_full);
log_ydata_full = 20* log10(abs_h_full);%to put in dB

% Update the plot
set(plotHandle, 'ydata', abs_h, 'xdata', gfreq);
set(plotHandle, 'ydata', log_ydata, 'xdata', log_xdata);
drawnow
end
```

Fig. 5: Acquisition function for windows

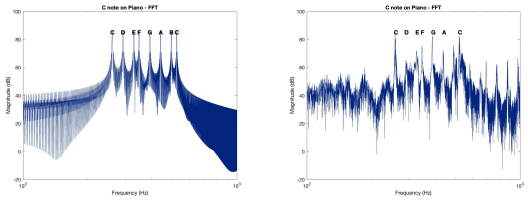
```
audio_name = "./audios/Acoustic_piano_Batimento_record_Mac05.wav";
% Read an MP3 file
[y, fs] = audioread(audio_name);
sound(y, fs);

plot_fft(y, fs)
plot_spectrogram(y, fs)

function plot_fft(data, fs)
    dark_green = 1/255 * [0,100,0];
    dark_blue = 1/255 * [3,37,126];

    N = length(data);
    Y = fft(data);

    f = (0:N-1) * fs / N;

    C_notes = [261.63, 293.66, 329.63, 349.23, 392, 440, 493.88, 523.25];
    G = 392*2;

    % Plot the waveform
    figure(1); plot(f,(20*log10(abs(Y))), 'color', dark_blue); xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); title('C note on Piano - FFT'); set(gca, 'XScale', 'log'); xlim([10^2, 10^3]);
    text(C_notes(1), 10*log10(5000), 'C', 'FontSize', 12, 'FontWeight', 'bold', 'HorizontalAlignment', 'center');
    text(C_notes(2), 10*log10(5000), 'D', 'FontSize', 12, 'FontWeight', 'bold', 'HorizontalAlignment', 'center');
    text(C_notes(3), 10*log10(5000), 'E', 'FontSize', 12, 'FontWeight', 'bold', 'HorizontalAlignment', 'center');
    text(C_notes(4), 10*log10(5000), 'F', 'FontSize', 12, 'FontWeight', 'bold', 'HorizontalAlignment', 'center');
    text(C_notes(5), 10*log10(5000), 'G', 'FontSize', 12, 'FontWeight', 'bold', 'HorizontalAlignment', 'center');
    text(C_notes(6), 10*log10(5000), 'A', 'FontSize', 12, 'FontWeight', 'bold', 'HorizontalAlignment', 'center');
    text(C_notes(7), 10*log10(5000), 'B', 'FontSize', 12, 'FontWeight', 'bold', 'HorizontalAlignment', 'center');
    text(C_notes(8), 10*log10(5000), 'C', 'FontSize', 12, 'FontWeight', 'bold', 'HorizontalAlignment', 'center');
    hlines(15, 10*log10(5000), 'G', 'FontSize', 12, 'FontWeight', 'bold', 'HorizontalAlignment', 'center');
end

function plot_spectrogram(data, fs, window_size, overlap, nfft)
    % Para parametros opcionais,
    if ~exist('window_size', 'var') window_size = 1024; end
    if ~exist('overlap', 'var') overlap = 512; end
    if ~exist('nfft', 'var') nfft = 1024; end
    y = reshape(data, 2, []);
    [S, f, t] = spectrogram(y, window_size, overlap, nfft, fs);
    figure(2); imagesc(t, f, 10*log10(abs(S(:,:)))); axis xy; colormap(jet); colorbar; xlabel('Time (s)'); ylabel('Frequency (Hz)'); title('C note on Piano'); ylim([10, 10^3]);
end
```

Fig. 6: Acquisition function for mac

In figure 7(a) we have the fft the C major scale produced using sine waves generated in python and in figure 7(b) the same scale played on a piano. Both audio files are present in the submitted folder, as well as the source code. Notice that in both figures the frequencies of the notes of the C major scales are marked according to their standard value (check table I). As you might tell, the peeks in the sinusoidal wave are much more clear, that has to do with the robustness of the piano sound (the fact that the piano audio is recorded and the sine generated scale is not also has influence on the noise but does no interfere with the magnitude of the peeks). Another interesting detail is that the sinusoidal scale doesn't present any peek beyond the highest C, while the piano scale does. that's caused by all the harmonics present in every note played by the piano. Notice that the peeks appear in the same frequency in both case, as they should, and the frequency values corresponds to the ones presented in table I which are the standard frequency values for this C scale.

Note	C	D	E	F	G	A	B	C
F. (Hz)	262	294	330	350	392	440	494	523

TABLE I: Fundamental frequencies of the note from C major scale



(a) FFT sinusoidal C scale (b) FFT piano C scale

Fig. 7: FFT of sinusoidal and piano C scale

B. *Long C note*

In figure 8 we can observe the fft of a long C note played on the Piano. Note that in the graph we have other peeks besides the 261.63 Hz (C note fundamental frequency) that's due to the presence of harmonics, as such, it's easy to note peek at the double of the fundamental and at 784 Hz, which corresponds to the first two harmonics of the harmonic sequence as mention in subsection IV-A. To notice that the predominant frequency is the fundamental frequency and the harmonics intensity decrease with their frequency.

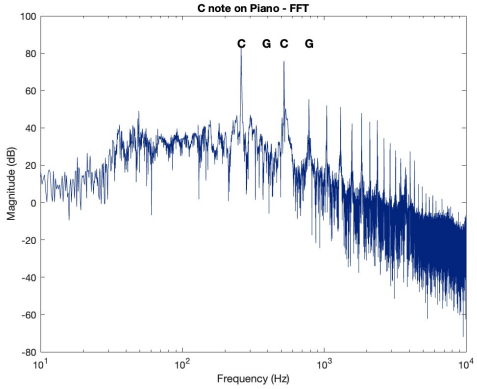


Fig. 8: spectrum of a long piano C note

Now let's explore the acoustic beat. in figure 9 we can see a zoom of the peek of 261.63 Hz at figure 8 (corresponds to frequency of C very zoomed). Those three peeks represent the three strings associated with the C key of the piano. You

might think that those strings should be tuned at the exact same frequency, yet this tuning is purposeful in order to create a sensation of movement in the note, i.e, creates the felling that the magnitude of the sound is periodically increasing and decreasing with the frequency equal to difference of the string frequency associated with the key, as mention in IV-B

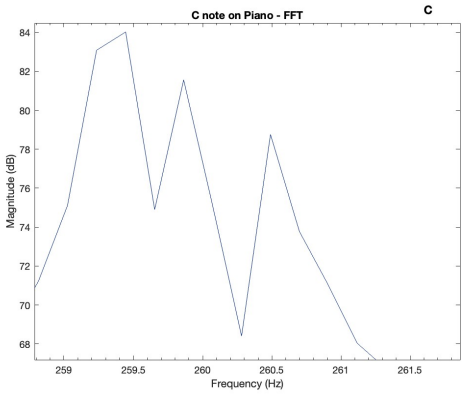
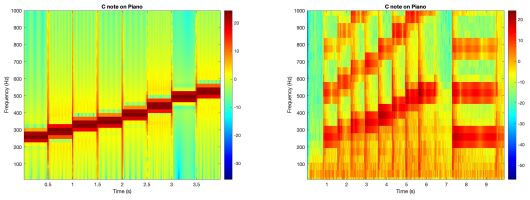


Fig. 9: Zoom of the fist fundamental frequency of the C note

Furthermore, we computed the spectrograms for both sinusoidal based C scale and piano played C scale (figure 10. The presence of the harmonics is clear in the Piano C scale (figure 10 b), yet the increasing of frequency over time is much more evident in the sinusoidal based C scale (figure 10 a), this increase represent the progressive higher pitch present at the C scale. In figure 11 we have the spectrogram relative to the sinusoidal based C scale generated in PRAAT software, which is very similar to the one generated in Matlab but it also presents the derivative pitch. Using praat is much easier, yet it's not as flexible as Matlab.



(a) Sinusoidal based C scale (b) Piano C scale

Fig. 10: spectrograms of sinusoidal and piano C scale in Matlab

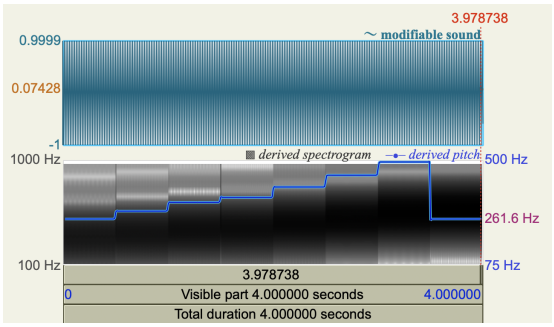


Fig. 11: spectrograms of sinusoidal and piano C scale in Praat

Lastly we also compared the sound recorded in the disk and the one produced by Matlab and they look the same. An exepected result since the aquisition method was the the same for both cases.

VI. Final Remarks

A. Code Overview

Regarding the code, mentioned in section V-A figure 5, we start the recording with a given duration, and call the event handler (explained in section III-D that will plot the fft and spectogram and save the record every time there is new information. This differs the code in 6 in the acquisition part, since we start by defining an audio acquisition device, and then then the acquisition is made every time there is new audio detected. This second way involves the use of global variable to save information and a vector that is constantly being updated with new acquisitions.

The plot of the fft is very simple: make the transformation using the matlab function fft() then the plot the data with a logarithmic scale on the x-axis (but the values are still in Hertz) and a linear scale in y axis (with the value in dB: magnitude = 10log(value)). It also marks some important frequencies, which in this case are the notes from the C scale. The spectogram plot allows the user to chose the parameters or to use the default values for overlap, window size and nfft and the reshapes the data in order to used the built in matlab function spectogram().

B. Folder Organization

All the audio files mentioned in this document are included in the submitted folder in *audios*, as well as all the code and images in *figures*. The file *work1* is the incomplete Windows version, and *Alternativa_MacOS.m* is the alternative made to work with all operating systems. We have also developed another similar code (*fft_and_specto_analyzer.m*) that plots the FFT and spectrogram of a given saved file. The files *demo_continuous_audio_acquisition_using_session.m* and *helper_continuous_fft.m* are just some templates that were not used to generate results for this document.

References

[1] Rossing, Thomas D., Moore, F. Richard, Wheeler, Paul A, The Science of Sound, Pearson Education, 2022.