Audio and Acoustics Signal Processing, Mini-project report

Frequency dependent absorption for room acoustic simulation

Abstract

In this project, I implemented a way to use and simulate frequency dependent absorption coefficients for wall reflections in the Pyroomacoustics package. For that purpose, I modified the existing implementation to generate different room impulse responses (RIRs), based on frequency and then designed bandpass filters to apply to the RIRs to filter the input signal. The resulting simulation corresponds to what we could expect from a sound in a room with given absorption.

Introduction

Pyroomacoustics[1] is a python package that allows room acoustics simulation. You can design a room by defining the position and absorption of the walls and simulate the result of hearing a sound played anywhere in the room from any position. It is based on the Image Source Method (ISM). Reflections to the microphone or listener are computed relative to each wall with a symmetrical image of the source, up to a given order. Each of these reflection is attenuated due to the travel distance in air and to the wall that absorbs part of the sound. From that model, a room impulse response is generated with the delays and amplitude loss created by the reflections. The input sound is then filtered with this impulse response to create the output, as heard from the microphone.

This project addresses an enhancement proposal on the Pyroomacoustics GitHub page[2], that suggest that the wall absorption could be frequency dependent. At the present time, an assumption is made that sound hitting a wall is attenuated equally for all frequency, which is in practice is not exactly correct. Indeed, materials absorb sound differently based on frequency. For example, the absorption from a glass window goes from 0.35 at 125 Hz down to 0.04 at 4000 Hz[3]. Coefficients tables exist for all sorts of material and can be found online. This project shows an implementation of frequency dependent absorption in room acoustics with the Pyroomacoustics package.

Review of existing techniques

Some other room acoustics software packages exist in MATLAB, of which some also feature frequency dependent absorption coefficients[4,5]. These implementations are closed source and need a MATLAB license to be used, but they follow the same approach of image source method as in Pyroomacoustics. I was not able to find any other Python implementation of room acoustics that features frequency dependent absorption coefficients.

Implementation

The first part of the project consisted in getting familiar with the source code. The program pipeline has four parts. The first is the room creation (in Room.py), second, the image source model construction (also in Room.py), then the RIR computation (in Soundsource.py), and finally the simulation, filtering the signal with the computed RIR. These are the parts that I had to modify to allow frequency dependent absorption.

Room creation

First, I had to define a format to input the absorption coefficients. From diverse sources, the format seemed to be standard. Coefficients are measured for 125, 250, 500, 1000, 2000, 4000 Hz[6], and sometimes also for 8000 Hz[7]. As only some of the material have coefficients for 8000 Hz and that those which have are mostly the same as for 4000 Hz, I chose to limit the table length to six coefficients. These frequencies are defined in Parameters.py and can easily be changed, and further frequencies can also be added, as long as every frequency is two times the previous one. I also assumed that the remaining of the frequency range (up to the Nyquist frequency) had the same coefficient as the highest specified frequency.

Concerning the room creation, the package allows to input either a single coefficient to use for every wall of the room or a coefficient per wall. My implementation additionally allows to have a coefficient vector (of length equal to the number of frequencies in the table, 6 by default) per wall. This applies to the Shoebox room, to defining a room from its corners and to extrusion.

Image source model

The modifications to the image source model construction were mostly NumPy manipulations. As the absorption is now a 2D array (a row per wall and each column is a coefficient for a given frequency), this new structure needed to be forwarded as a 2D damping. Damping is defined as the succession of amplification coefficients to be applied to the signal. The damping is first 1 for direct sound, then takes values 1-a_i, with a_i being the absorption coefficient of wall I, and then further reduction by multiplying with the damping coefficients of the walls hit. For example, if each wall has absorption 0.2 (hence damping 0.8), a sound wave will first have damping 1 before hitting a wall, then 0.8 when hitting the first wall, and 0.64 after the second reflection. Hence, if we have for walls with absorption a_1, a_2, a_3, a_4 (and damping d_i = 1-a_i), the damping looks like $[1, d_1, d_2, d_3, d_4, d_1d_2, \dots]$. The damping now must be two-dimensional, having one row per frequency bin. To do so, we first take a unit column vector (of length equal to the number of frequencies in the table), then append the transpose of 1-A, where A is the 2D absorption array computed in the room creation, and then iteratively append the further reflections if needed. This structure will simplify the changes in RIR computation.

RIR computation

The room impulse response from source s_0 to microphone r, at instant n is computed with the following formula:

$$a_{\boldsymbol{r}}(\boldsymbol{s}_0, n) = \sum_{\boldsymbol{s} \in \mathcal{V}_{\boldsymbol{r}}(\boldsymbol{s}_0)} \frac{(1-\alpha)^{\text{gen}(\boldsymbol{s})}}{4\pi \|\boldsymbol{r}-\boldsymbol{s}\|} \delta_{\text{LP}} \left(n - F_s \frac{\|\boldsymbol{r}-\boldsymbol{s}\|}{c} \right)$$

Equation 1: formula of impulse response from source s_0 to microphone r (from the Pyroomacoustics original paper)

Where, in the sum, the numerator is the damping coefficient, the denominator is the impact of distance and δ_{LP} is a fractional delay filter, creating interpolation between the integer samples n. There are multiple ways of implementing multiple frequency bins support at this point. The first option is to add another summation over the rows of the damping (i.e. the frequency bins) and replace the fractional delay filter (that is an all-pass filter from 0 to Nyquist frequency) with band-pass filters, having a passband for each frequency bin. However, the fractional delay filters are useful for other modules of the package and should not be replaced. Hence, I decided to postpone the band-pass filtering to the simulation part and create a separate impulse response for each frequency bin. The room has then n RIRs, where n is the length of the frequency table, that can be plotted separately for a better visualization of the impact of the frequency dependent coefficients on the impulse response.

Simulation

This is the part where we use the impulse response (now impulse responses) to compute the output signal. We want to convolve each impulse response with a band-pass filter that will select the frequency range associated with the selected frequency. I defined the frequency ranges such that they span all the frequencies. Since each frequency in the table is two times the previous one, I chose to define the passband such that each band goes from halfway from the previous frequency to halfway to the next one. That means that the frequencies in the absorption coefficients table are not exactly the center frequencies, but it allows each filter to have a similar shape.

I had two options to design such filters. First is using the *scipy.signal.firwin*[8] method. This method from the SciPy library creates a FIR filter with a given window method. The second is *scipy.signal.butter*[9] and creates a Butterworth filter, that aims at being maximally flat at passband.

FIRwin

The firwin method takes as argument a filter length, which I chose to be 512, because it is the same length as the one used by default in the butter method. It then takes the boundaries of the bandpass filter, which is the range previously defined, and the window method to use, which must be one of the defined window method in the SciPy library[10]. After some research and experimentation, I chose the hamming window as it seemed to be the most commonly used. This parameter can easily be changed.

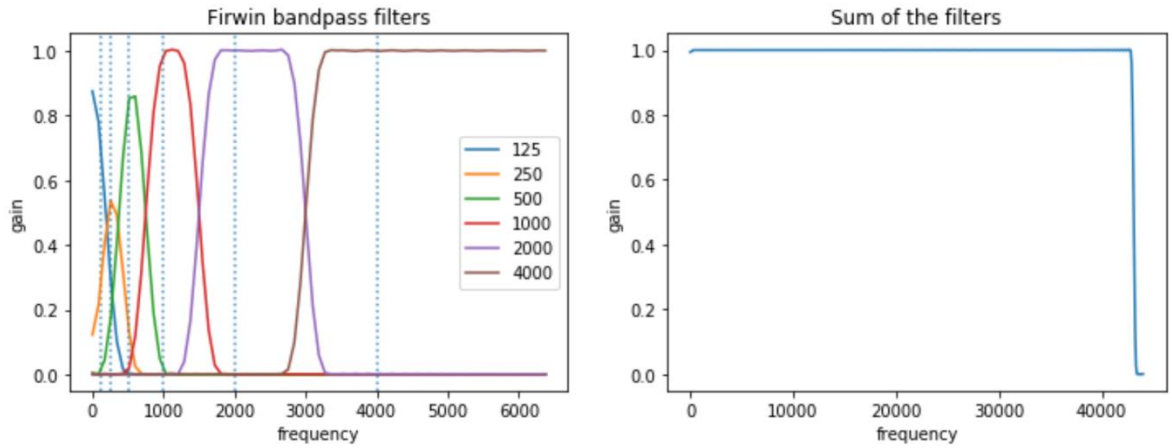

Figure 1: FIRwin bandpass filters, separated (a) and summed together (b)

In the first graph, the last (brown) filter goes all the way up to the Nyquist frequency (44100 Hz in this example. We can observe that some of the first filters do not have time to reach a gain of 1, because of resolution constraints, but that once summed together, we have a perfect all-pass filter from 0 to 44100 Hz. In this example, all filters are set to have gain 1, but we can simulate absorption by scaling the gain of the filters, based on absorption coefficients.

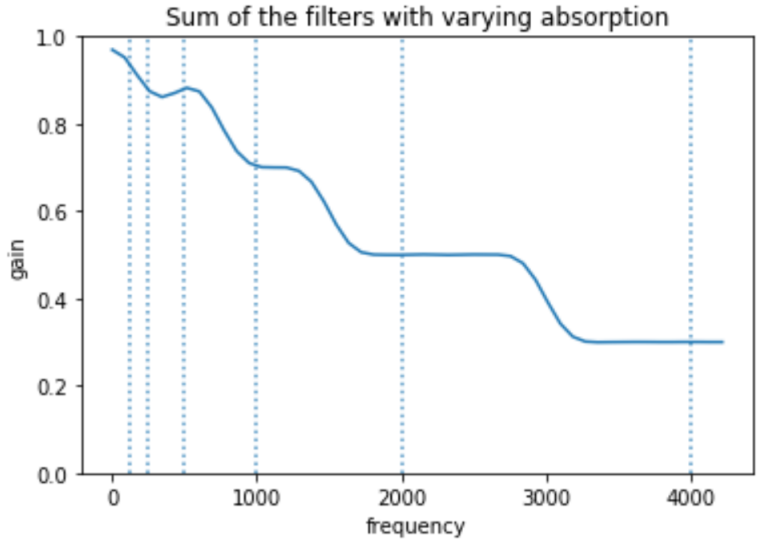

Figure 2: FIRwin filters with absorption [0,0.2,0.1,0.3,0.5,0.7]

Note that again, the graph continues up to the Nyquist frequency.

Butterworth

The butter method takes as argument a filter order, which I chose to be 5, because it is a commonly used value in practice. It then takes the boundaries of the bandpass filter, which is the range previously defined, and an output format, which can be 'ba' (first order, these are respectively the zeros and poles of the filter) or 'sos' (second order)

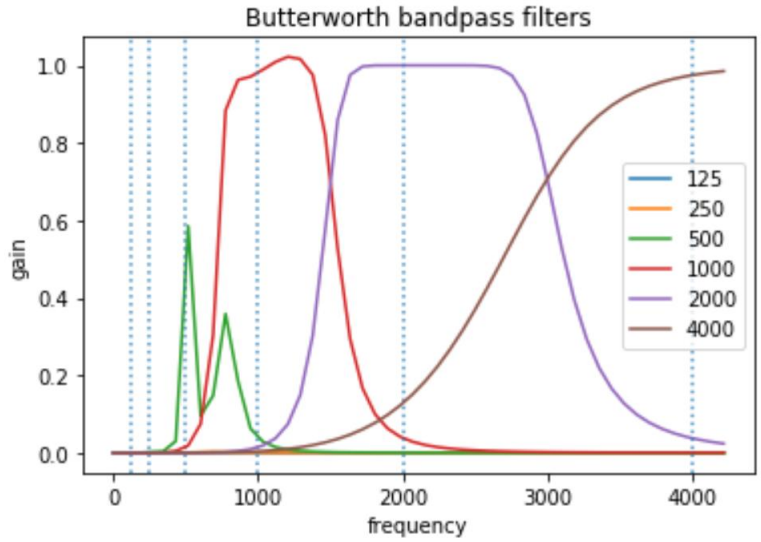

Figure 3: Butterworth filters with ba format

We can observe that the filters perform very poorly in the low frequencies. This is a known issue and is due to an instability related to the format. This unwanted behavior is solved by using the second-order output instead.

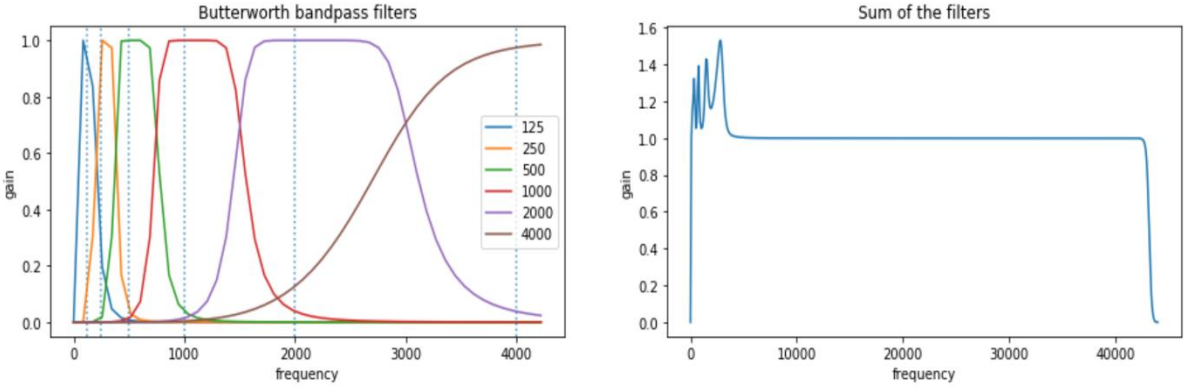

Figure 4: Butterworth filters with sos output format

We can observe that, with this method, all filters reach a gain of 1, which is nice, but once summed together, we have some residue. The summed filter is not a perfect all-pass filter and seem to accentuate some of the frequencies at the boundaries of the ranges. Once again, we can simulate absorption by scaling the gain of the filters, based on absorption coefficients.

Both firwin and butter method have advantages and downsides. Once applied to a signal, both seem to behave equally, so I decided to allow both methods in the simulate function. The filter method can be set as an argument and chosen between 'firwin' and 'butter'.

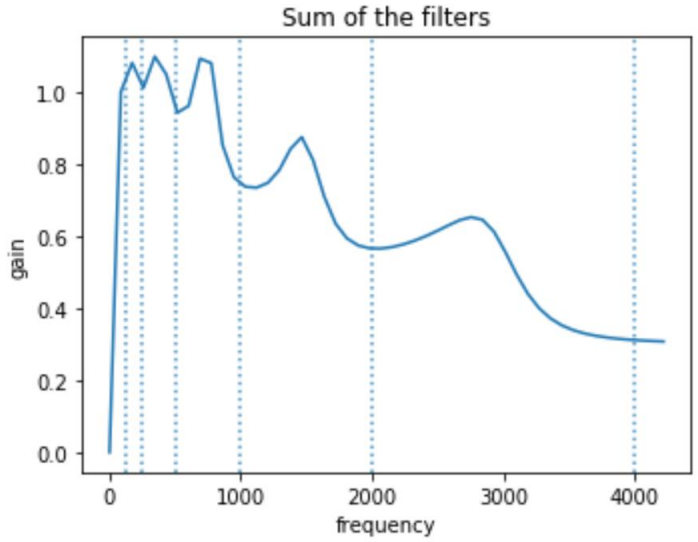

Figure 5: Butterworth filters with absorption [0,0.2,0.1,0.3,0.5,0.7]

Results and discussion

The results correctness cannot be quantified, but the output is convincing. It is now possible to create a room with frequency dependent absorption coefficients and simulate the room acoustics for a given input. The different impulse responses for each frequency band can be plotted. You can find an execution example in the *frequency_dependent_absorption_demo* notebook. I define a shoebox room with walls that are entirely reflective (absorption 0) at frequencies up to 2000 Hz and maximally absorbent (absorption 0.99) after 4000 Hz

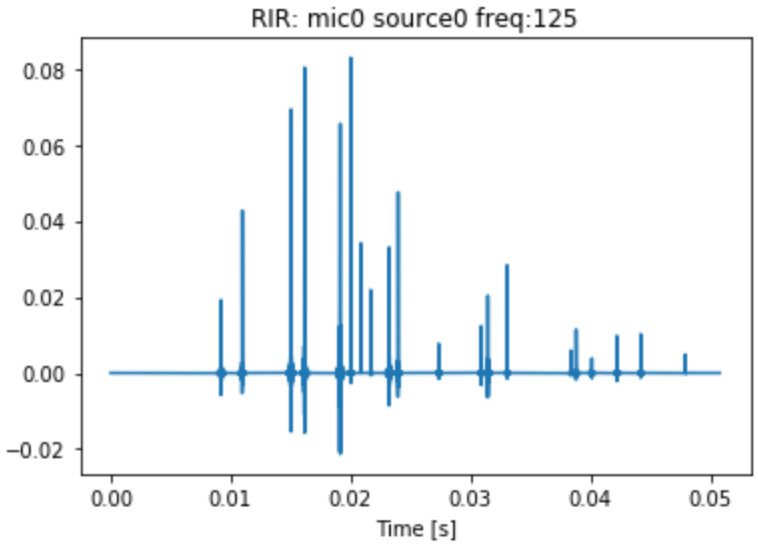

Figure 6: Room impulse response for the first frequency band (reflective)

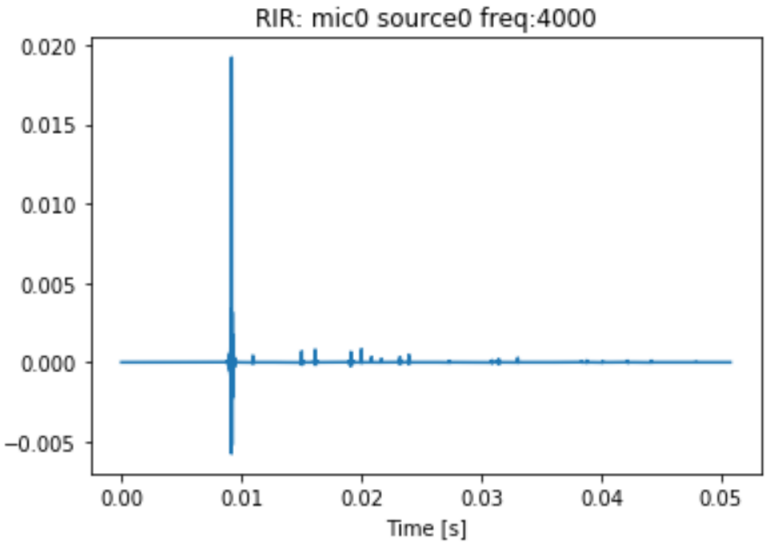

Figure 7: Room impulse response for the last frequency band (absorbent)

We can observe that the impulse response for 4000 Hz (and more) is very dry, due to the maximal absorption of the walls. On the contrary, it has a richer tail for the frequencies at which the material is reflective. Note that the impulse responses of frequencies bands up to 2000 Hz are the same as the first one. To test the implementation, I used a simple sound signal that is a linear frequency sweep from 0 to 44100 Hz. For that simulation, I used a maximal reflection order of 3, i.e. sound can hit at most 3 walls before reaching the microphone. The resulting output corresponds to the expectations. The first part of the sweep up to 2000 Hz has a lot of reverb and becomes dry after 4000 Hz. The result can be heard by running the notebook. Moreover, the two methods (firwin vs butterworth) can be compared. I was personally not able to tell a difference when listening to the outputs, hence I chose to allow the use of the two methods.

In terms of runtime, the image source model construction takes a bit more time, because of the extra dimension in the damping array, causing more memory usage and more multiplications. The filters creation and convolution take an extra 1-2 seconds compared to the algorithm without frequency dependent absorption, but it is an absolute time that does not increase with reflection order of room complexity. I chose to build the filters at simulation time instead of doing it at the start of the program. The reason is that if we run the simulation once, we compute only the user-selected filter, but we would have had to compute both filters if it was done at the start of the program, as we don’t know at this point what filter the user will use. It would only become more efficient to pre-compute the filters if the simulation is done more than two times.

Conclusion

The goal of this project was to show an implementation of frequency dependent absorption in the Pyroomacoustics package. I achieved that by creating different impulse responses for each frequency bin and then filtering them with a bandpass filter before summing them together. The results show that the output is acoustically coherent with the chosen absorption coefficients.

One simplification was made and could be the object of further research, is that I assume that the absorption due to the air is only due to the travel distance and is not related to the frequency, which is not the case in practice[11].

Bibliography

[1] Robin Scheibler, Eric Bezzam, Ivan Dokmanić. Pyroomacoustics: A Python package for audio room simulations and array processing algorithms. https://arxiv.org/abs/1710.04196, https://github.com/LCAV/pyroomacoustics

[2] https://github.com/LCAV/pyroomacoustics/issues/48

[3] http://www.sengpielaudio.com/calculator-RT60Coeff.htm

[4] Douglas R. Campbell, Kalle J. Palomäki and Guy J. Brown. Roomsim, a MATLAB Simulation of "Shoebox" Room Acoustics for use in Teaching and Research. http://users.spa.aalto.fi/kpalomak/pubs/Campbell_Palomaki_Brown.PDF

[5] Andrew Wabnitz, Nicolas Epain, Craig Jin and André van Schaik. Room acoustics simulation for multichannel microphone arrays. https://pdfs.semanticscholar.org/3a63/d433cb178a78a11a1f109a11de8693c1bab9.pdf

[6] http://www.sengpielaudio.com/calculator-RT60Coeff.htm

[7] https://cds.cern.ch/record/1251519/files/978-3-540-48830-9_BookBackMatter.pdf

[8] https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.firwin.html

[9] https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.butter.html

[10] https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.get_window.html#scipy.signal.get_window

[11] http://www.sengpielaudio.com/calculator-air.htm