Lab work nº1

Information and Coding

Universidade de Aveiro

João Su 112790, Isac Cruz 90513

2025/2026

Departamento de Eletrónica, Telecomunicações e Informática

19 October, 2022

Contents

Introduction

This report presents the results obtained in **Information and Coding – Lab Work nº 1**.

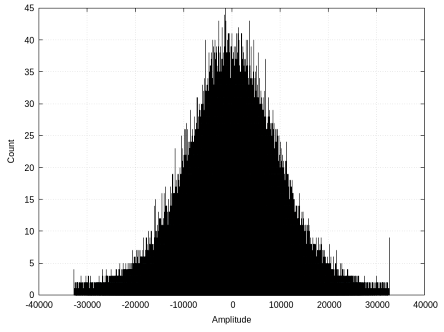
All the source code developed for this lab is available in the following repository: GitHub Repository. Detailed instructions on how to use each program and script, as well as the steps required to generate the corresponding graphs and results, are provided in the `README` file of the same repository.

Part I

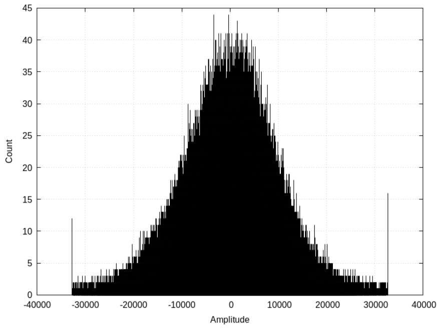
2.1 Exercise 1

In this exercise, the WAVHist class was modified to compute histograms for stereo audio, including the MID and SIDE channels, defined respectively as MID $= \frac{L+R}{2}$ and SIDE $= \frac{L-R}{2}$. Additionally, coarser bin options were implemented to group multiple sample values per bin, with each bin covering 2^n values.

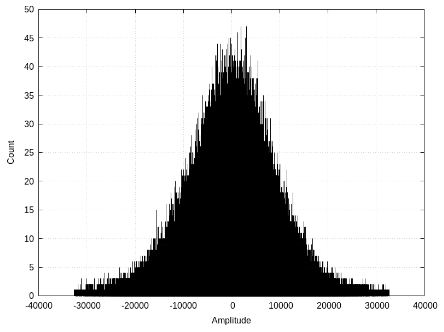
The results presented below were obtained using a sample of audio from *Dream Theater – The Alien*. Listening to the sample, we observe that the left and right channels are very similar, as can be seen in Figures 2.1a and 2.1b, where the histograms of the two channels closely resemble each other. Considering this similarity, it is possible to draw conclusions about the Mid and Side channels. For the Mid channel, being the average of the left and right channels, its histogram closely resembles those of the original channels, as can be seen in Figure 2.1c. For the Side channel knowing that the differences between the left and right channel are very small, resulting in values concentrated around the zero, which is reflected in Figure 2.1d, where the count values near the center are almost twice as high as in the individual channel histograms.



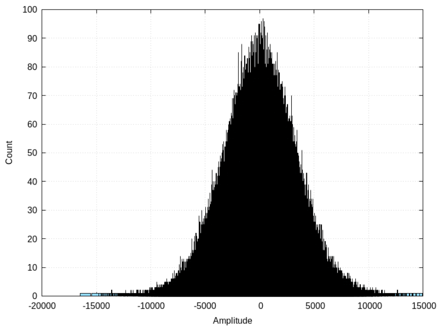
(a) Left channel



(b) Right channel



(c) Mid channel

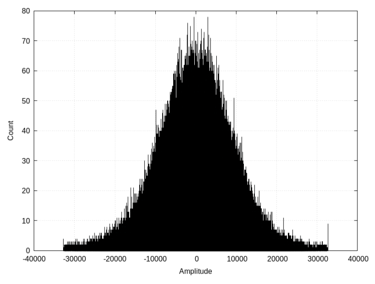


(d) Side channel

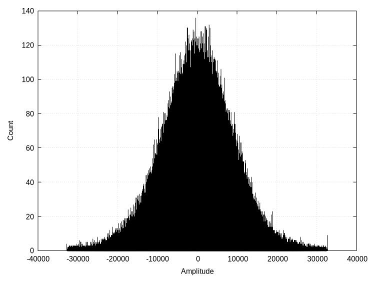
Figure 2.1: Histograms of Left, Right, Mid, and Side channels.

When using coarser bins, the audio amplitudes are grouped together, effectively reducing the number of quantization levels by a factor of 2^n. For example, grouping 65536 levels (16-bit audio) into bins of 1024 results in only 64 levels ($65536/1024 = 64$). This reduction in levels causes the number of counts in each bin to increase, since more samples fall into each coarser category. This effect is clearly visible in Figure 2.2, where the histograms for coarser bins show taller peaks compared to the finer-resolution histograms.

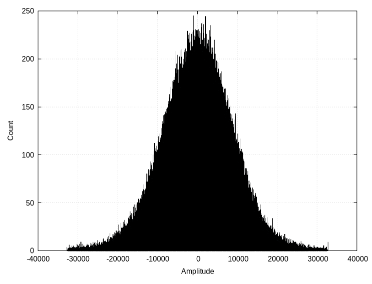
It is also important to note that in Figures 2.1a and 2.1b, distinct spikes can be observed at the extremes of the histograms. This behaviour occurs because this audio sample slightly exceeds the maximum representable amplitude for 16-bit audio. As a result, all sample values that surpass the 16-bit range are clipped to the limit, causing an accumulation of counts at the ends of the histogram and producing the visible spikes.



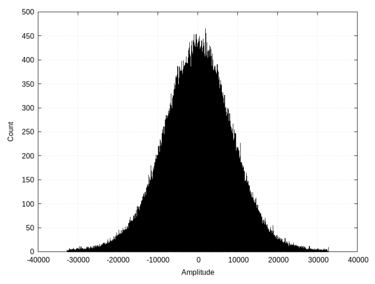
(a) Bin = 2



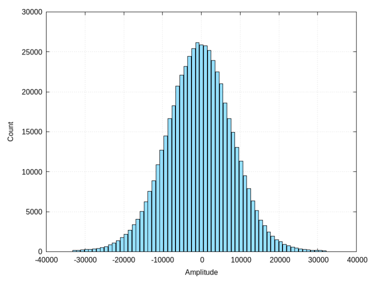
(b) Bin = 4



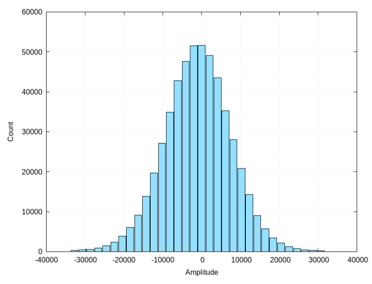
(c) Bin = 8



(d) Bin = 16



(e) Bin = 1024

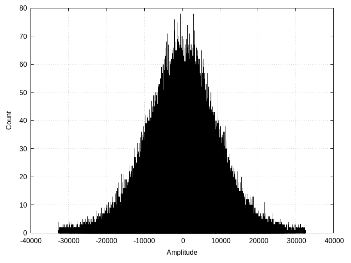


(f) Bin = 2048

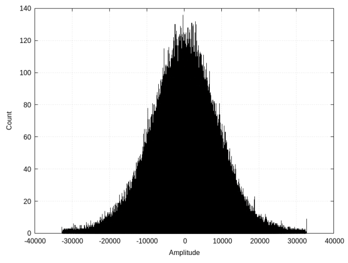
Figure 2.2: Histograms of the Left channel with different bin sizes (2, 4, 8, 16, 1024, 2048).

2.2 Exercise 2

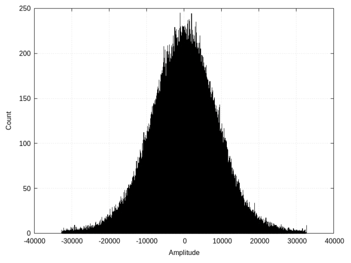
In this exercise, we implement a program named wav_quant that allows us to decrease the number of bits used to represent the samples of an audio signal, originally with 16-bit resolution. The program reduces the bit depth of each sample and generates a new WAV file with the quantized audio, making it possible to listen to the effect of lower-resolution audio and observe how reduced bit depth impacts sound quality. For example, with 16 bits we have a representation range of $[-32768; 32767]$, corresponding to 2^{16} quantization levels. Reducing the bit depth by n bits decreases the number of available levels by a factor of 2^n, giving 2^{16-n} levels. For instance, an audio file quantized to only 5 bits will have $2^5 = 32$ levels (ranging from $[-16; 15]$). This effect can be observed in Figure 2.3.



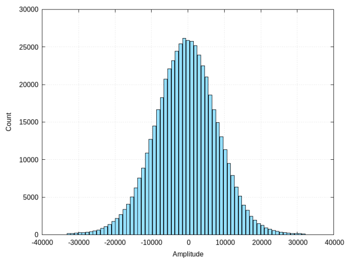
(a) 15-bit (32768 levels)



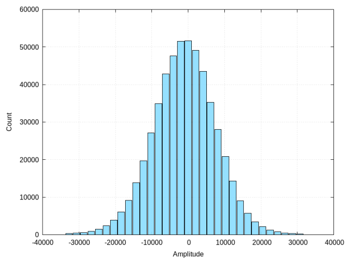
(b) 14-bit (16384 levels)



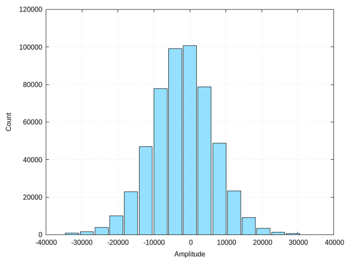
(c) 13-bit (8192 levels)



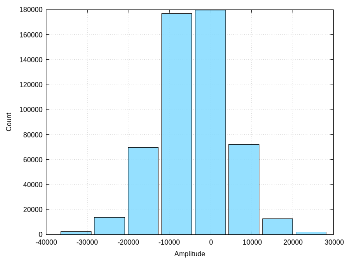
(d) 6-bit (64 levels)



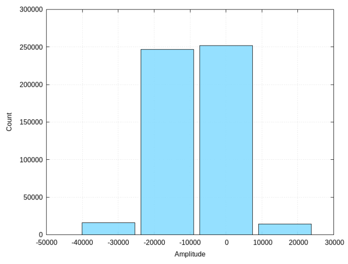
(e) 5-bit (32 levels)



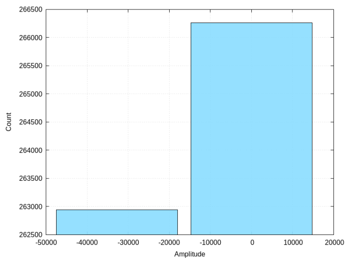
(f) 4-bit (16 levels)



(g) 3-bit (8 levels)



(h) 2-bit (4 levels)

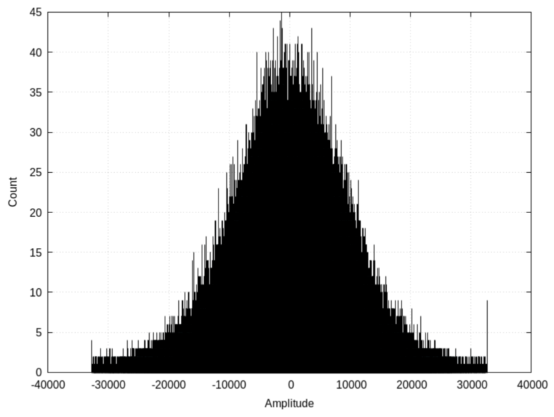


(i) 1-bit (2 levels)

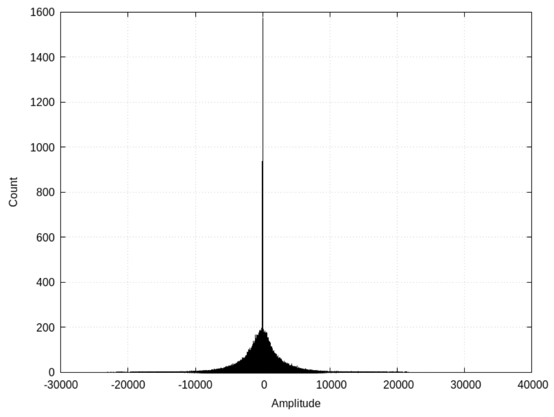
Figure 2.3: Histograms of the audio sample *Dream Theater – The Alien* at different quantization levels.

Quantization directly affects the quality of the sound, which becomes perceptible when listening to the audio sample. As the number of bits used to represent each sample decreases, we begin to notice a *hiss* or distortion in the signal. This occurs because reducing the number of bits also reduces the number of discrete levels available to represent the analog signal. With fewer levels, the difference between the actual analog value and its quantized digital representation increases, introducing more quantization noise into the signal.

To further explore the effects of quantization, we performed a small listening experiment using two different audio samples: *Dream Theater – The Alien* and *Carlos Paredes – Canção Verdes Anos*. These samples represent two opposite types of music in terms of amplitude: the first has higher overall amplitude, while the second is more subdued. As we progressively decreased the number of bits used for quantization, we observed that for the Dream Theater sample, the characteristic *hiss* became noticeable only at 6-bit representation. In contrast, for the Carlos Paredes sample, the noise was already perceptible at 8 bits. This difference in the audibility of quantization noise was likely caused by the amplitude range of the samples. As can be observed in Figures 2.4a and 2.4b, the amplitude range of the Carlos Paredes sample is almost three times smaller compared to the Dream Theater sample. This smaller range makes the signal more susceptible to quantization errors, since there are fewer discrete values available to represent the waveform, leading to a higher concentration of samples within fewer levels and making the noise more perceptible. This experiment was conducted using the computer's built-in speakers, using headphones or a higher-quality audio system could result in different thresholds for perceiving quantization noise.



(a) Dream Theater sample



(b) Carlos Paredes sample

Figure 2.4: Comparison of histograms in the left channel between two different types of audio samples.

2.3 Exercise 3

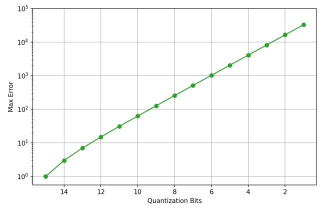
In this exercise, we implement a program named wav_cmp that evaluates the differences between an audio file and its original version. For each channel and for the average across all channels, it computes three error metrics: MSE, which measures the average error between samples, Max Error, which represents the largest deviation between any pair of samples, and SNR, which indicates how strong the original signal is compared to the error.

For these results, a sample of audio from *Dream Theater – The Alien* was used. From Table 2.1 and Figure 2.5, it is possible to verify the degradation of the signal as the number of quantization bits decreases. The RMSE increases, indicating that the average difference between the original and quantized samples becomes larger. The maximum error also increases approximately by a factor of 2^n as the number of bits decreases. Meanwhile, the SNR decreases, meaning that the noise introduced by quantization becomes more perceptible.

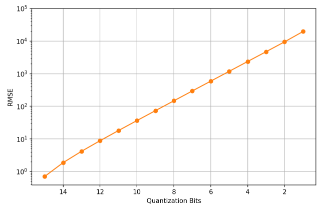
Additionally, as discussed in Section 2.2, the perceptibility of this quantization noise depends on both bit depth and the amplitude range of the audio. For instance, the characteristic *hiss* became noticeable at 8 bits for the Carlos Paredes sample, while for the Dream Theater sample it was only perceptible at 6 bits. In terms of SNR, the Carlos Paredes sample at 8 bits had an average SNR of 24.738 dB, whereas a similar SNR of approximately 24 dB (23.2637 dB) for the Dream Theater sample was reached only at 6 bits.

Table 2.1: Comparison of quantized audio to original sample

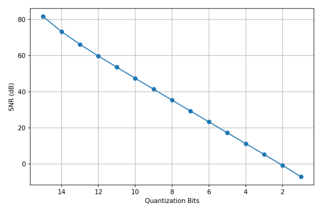
Bits	**Left Channel**			**Right Channel**			**Average**		
	RMSE	Max Error	SNR (dB)	RMSE	Max Error	SNR (dB)	RMSE	Max Error	SNR (dB)
15	0.7070	1	81.6961	0.7079	1	81.7054	0.7075	1	81.7007
14	1.8718	3	73.2391	1.8712	3	73.2627	1.8715	3	73.2509
13	4.1861	7	66.2482	4.1855	7	66.2700	4.1858	7	66.2591
12	8.8080	15	59.7868	8.8038	15	59.8116	8.8059	15	59.7992
11	18.0442	31	53.5577	18.0432	31	53.5788	18.0437	31	53.5682
10	36.5184	63	47.4342	36.5226	63	47.4538	36.5205	63	47.4440
9	73.4278	127	41.3672	73.4300	127	41.3876	73.4289	127	41.3774
8	147.341	255	35.3179	147.415	255	35.3342	147.378	255	35.3261
7	295.410	511	29.2759	295.301	511	29.2998	295.355	511	29.2878
6	590.778	1023	23.2559	591.126	1023	23.2715	590.952	1023	23.2637
5	1181.95	2047	17.2324	1181.32	2047	17.2577	1181.63	2047	17.2451
4	2365.10	4095	11.2074	2367.57	4095	11.2190	2366.33	4095	11.2132
3	4723.43	8191	5.1993	4737.15	8191	5.1947	4730.29	8191	5.1970
2	9468.02	16383	-0.8408	9447.60	16383	-0.8014	9457.81	16383	-0.8211
1	19643.2	32767	-7.1798	19563.2	32767	-7.1238	19603.2	32767	-7.1518



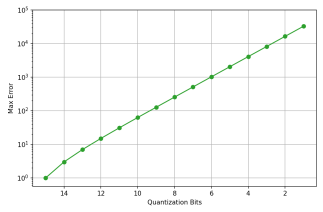
(a) Left channel Max Error



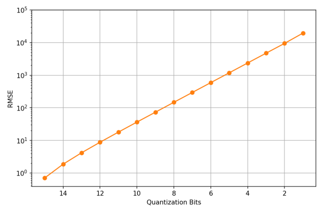
(b) Left channel RMSE



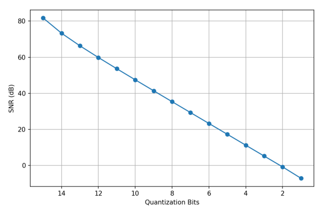
(c) Left channel SNR



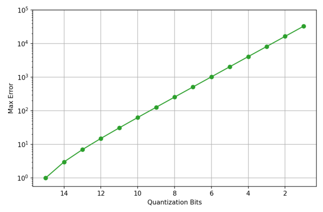
(d) Right channel Max Error



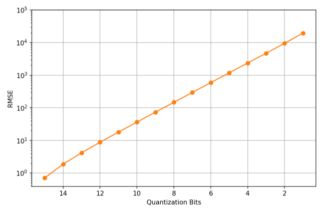
(e) Right channel RMSE



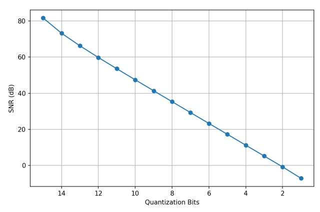
(f) Right channel SNR



(g) Average Max Error



(h) Average RMSE



(i) Average SNR

Figure 2.5: Comparison of Max Error, RMSE, and SNR for Left, Right, and Average channels at different quantization bit depths.

2.4 Exercise 4

2.4.1 Architecture

To improve code versatility, the code is divided into three main parts:

- The filter flow, which applies the chosen filters (filter_flow from Figure 2.7).
- The filters themselves: a group of functions, each applying a filter to a buffer and emptying the filter's buffer after finishing, if applicable (functions, structs from Figure 2.7).
- I/O: loads the signal from an external source (memory from Figure 2.7).

All the information needed for applying the requested filter should be loaded into the *Filter_Parameters* structure. Then, the filter uses that structure to apply the intended effects. This structure is also equipped with a circular buffer that stores a specified amount of samples that can be later used for delay, echo, and other effects that require signal memory.

Every effect that uses memory implements a function to empty its delay buffer under certain conditions: an empty input and the *Filter_Parameters*' *Finish* flag set to true.

A setup function is provided to initialize the parameters according to the given sample rate and number of channels. Each filter has its own setup function, which sets up the parameters according to the provided arguments, defining the buffer sizes and other important information.

The following formulas are implemented as written. Every time $S(x-d)$ is referenced, it implies the usage of a buffer of size d to access past signal values; each d value means a different buffer. Given the time constraints for the task, further memory optimization using a shared buffer among all effects was not possible. It was later considered closer to the deadline but not implemented.

Every single effect that could be vulnerable to accidental saturation, except for Gain (subsection 2.4.2), was mathematically designed to never saturate, provided the input is not already saturated and the limit values are respected.

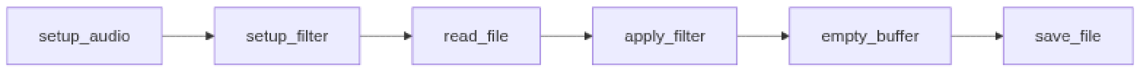


Figure 2.6: Workflow.

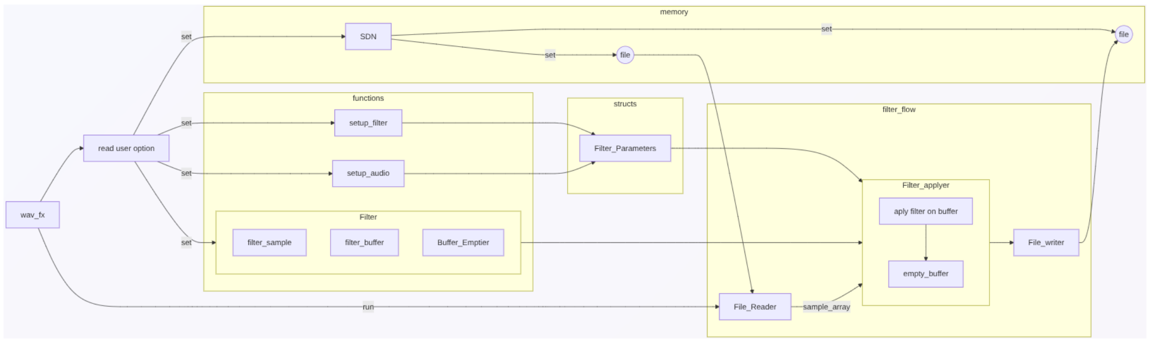


Figure 2.7: Program architecture.

2.4.2 Gain

Formula:

$$Y(x) = S(x) \cdot \text{Gain} \tag{2.1}$$

Deduction:

The Gain effect is the most fundamental audio operation, modifying the overall amplitude (volume) of the signal. The output signal $Y(x)$ is obtained by multiplying every sample of the input signal $S(x)$ by a constant Gain factor. If Gain > 1, the signal is amplified (made louder). If Gain $\in [0,1)$, the signal is attenuated (made quieter). A Gain of 0 results in silence.

2.4.3 Delay

Formula:

$$Y(x) = \begin{cases} 0, & x < n \\ S(x-n), & x \geq n \end{cases} \tag{2.2}$$

Deduction:

The Delay effect shifts the input signal $S(x)$ in time by a specified number of samples, n. The output signal $Y(x)$ is a time-shifted version of the input, where the original signal is "pushed" to the right. Consequently, the first n samples of the output (where $x < n$) are silent (0). At sample $x = n$, the output $Y(n)$ takes the value of the first input sample $S(0)$, and so on. This effect is the core building block for many other audio effects, such as echo, chorus, and flanger.

2.4.4 Fade in

Formula:

$$Y(x) = \begin{cases} S(x)\left(\frac{x}{n}\right)^b, & x < n \\ S(x), & x \geq n \end{cases} \tag{2.3}$$

where n is the fade duration in samples and x is the current sample index.

Deduction:

The objective was to manipulate the gain of the input signal between 0 and 1 ($\frac{x}{n}$) until the specified fade-in time (n). For the remaining signal, the output should match the input.

$$\begin{cases} S(x)\left(\frac{x}{n}\right), & x < n \\ S(x), & x \geq n \end{cases} \tag{2.4}$$

In this equation, the signal is 0 at time $x = 0$ and 1 at time $x = n$. The gain remains between 1 and 0 while x is between 0 and n, growing proportionally to the elapsed time. The problem with this formula is that the progressivity of the effect may be compromised, making it too smooth and difficult to notice. To improve the impact of the filter without further compromising its functionality, the gain was raised to the power of b, where $b \in \mathbb{Q}^+$. Since the gain $\in [0,1]$, the exponent b will only increase the rate of signal variation (i.e., the derivative), which increases the change per instant of the signal. This does not change the extremes or the function, but makes the change easier to notice, as the human ear catches signal variations more easily than static signals.

$$\begin{cases} S(x)\left(\frac{x}{n}\right)^b, & x < n \\ S(x), & x \geq n \end{cases} \tag{2.5}$$

2.4.5 Fade Out

Formula:

$$Y(x) = \begin{cases} S(x), & x \leq N_i \\ S(x)\left(1 - \frac{x-N_i}{N_f-N_i}\right)^b, & N_i < x < N_f \\ 0, & x \geq N_f \end{cases} \tag{2.6}$$

Deduction: The objective is to apply a diminishing gain to the signal $S(x)$, starting from time N_i until the end of the track at N_f, where the signal should be zero. The logic of this filter is the same as the subsection 2.4.4, but it is shifted on the x-axis to the end of the track (Gain $= \frac{x-N_i}{N_f-N_i}$) and inverted using the complement of 1 ($1-\text{Gain}$).

- N_i: The sample index where the fade out starts. For $x \leq N_i$, the output signal is the input signal, $Y(x) = S(x)$, meaning the gain is 1.
- N_f: The sample index where the track ends. For $x \geq N_f$, the output signal is $Y(x) = 0$.
- b: The aggressiveness (exponent) of the curve, $b \in \mathbb{Q}^+$.
- x: The current sample index (time).

The gain factor for the fade-out region ($N_i < x < N_f$) must decrease from 1 to 0. The expression $\frac{x-N_i}{N_f-N_i}$ represents the fraction of the fade-out duration elapsed (ranging from 0 to 1). Therefore, the linear gain factor is $1 - \frac{x-N_i}{N_f-N_i}$.

To introduce the non-linear progression for an improved perceived effect (controlled by b), this gain is raised to the power of b: $\left(1 - \frac{x-N_i}{N_f-N_i}\right)^b$.

This results in the complete piecewise function for the Fade Out:

$$Y(x) = \begin{cases} S(x), & x \leq N_i \\ S(x)\left(1 - \frac{x-N_i}{N_f-N_i}\right)^b, & N_i < x < N_f \\ 0, & x \geq N_f \end{cases}$$

2.4.6 Echo

Formula:

$$Y(x) = S(x) \cdot \left(\frac{1}{\text{Gain}+1}\right) + \frac{\text{Gain}}{\text{Gain}+1} \cdot S(x-d)$$

Deduction:

The default way to implement echo is to pull the signal from previous instants from a delay buffer and multiply it by a gain factor α.

$$S(x) + \alpha \cdot S(x-d)$$

The problem with this formula is signal saturation when the sound is too loud. The question is: is it possible to add an echo without signal saturation? For this purpose, a second α (α_1) was added to regulate the input signal.

$$\alpha_1 \cdot S(x) + \alpha_2 \cdot S(x-d)$$

The sum of the gains of both signals should not be bigger than 1; therefore, the sum of both α's was defined as 1.

$$\alpha_1 + \alpha_2 = 1 \iff \alpha_1 = 1 - \alpha_2$$

Then the final echo formula is:

$$S(x) \cdot (1-\alpha_2) + \alpha_2 \cdot S(x-d)$$

The gain is then calculated by:

$$\alpha \in [0;1], \quad \text{Gain} = \frac{\alpha}{1-\alpha} \iff \alpha = \frac{\text{Gain}}{\text{Gain}+1}$$

With this formula, it's possible to calculate α based on the desired gain. The final formula is:

$$S(x) \cdot \left(1 - \frac{\text{Gain}}{\text{Gain}+1}\right) + \frac{\text{Gain}}{\text{Gain}+1} \cdot S(x-d) \iff S(x) \cdot \left(\frac{\text{Gain}+1-\text{Gain}}{\text{Gain}+1}\right) + \frac{\text{Gain}}{\text{Gain}+1} \cdot S(x-d)$$

$$\therefore Y(x) = S(x) \cdot \left(\frac{1}{\text{Gain}+1}\right) + \frac{\text{Gain}}{\text{Gain}+1} \cdot S(x-d)$$

2.4.7 Multi Echo

Formula:

$$Y(x) = \frac{1}{2} S(x) \left(\frac{1}{\text{Gain}_{\text{total}}}\right) + \frac{1}{2N} \sum_{i=1}^{N} S(x-d_i)$$

$$\alpha, \text{Gain} \in [0;1] \in \mathbb{R} \wedge N, x \in \mathbb{N}_0$$

Deduction:

The main idea was to iterate several echoes with different delays and then calculate the average signal of these echoes. The main problem was signal saturation. An $\alpha \in [0,1]$ is used to keep the signal's gain normalized, as in the normal echo, but this time both the original signal and the echo receive the same α. Let β be the set of echoes to be applied, where β_i is the i-th echo, $\beta_i = S(x-d_i)$. Let α be the attenuator that prevents signal saturation, as in subsection 2.4.6.

The initial formulation is:

$$\frac{\sum_{i=1}^{N}(\alpha \cdot S(x) + \alpha \cdot \beta_i)}{N} = \alpha \cdot \frac{\sum_{i=1}^{N}(S(x)+\beta_i)}{N} = \alpha \left(\frac{N \cdot S(x) + \sum_{i=1}^{N}\beta_i}{N}\right) = \alpha \left(S(x) + \sum_{i=1}^{N}\frac{\beta_i}{N}\right) = \alpha S(x) + \frac{\alpha}{N}\sum_{i=1}^{N}\beta_i$$

The problem here is the loss of the echo's contribution with iteration (i.e., as N increases). The gain of the averaged echo part relative to the signal part is:

$$\text{Gain} = \frac{\alpha/N}{\alpha} = \frac{1}{N}$$

By subtracting a portion of the original signal, it is possible to improve the gain of the echoes. Since the original signal is being replicated N times (once for each of the N echoes), we subtract θ times the original signal's contribution, where $G \in [0,1]$.

$$\begin{aligned}\frac{\left[\sum_{i=1}^{N}(\alpha \cdot S(x) + \alpha \cdot \beta_i)\right] - \sum_{i=1}^{\theta} \alpha \cdot S(x)}{N} &= \alpha \cdot \frac{\left[\sum_{i=1}^{N} S(x) + \beta_i\right] - \theta \cdot S(x)}{N} \\ &= \left[\alpha \cdot \frac{\sum_{i=1}^{N} S(x) + \beta_i}{N}\right] - \frac{\theta \cdot \alpha \cdot S(x)}{N}\end{aligned}$$

$$\text{New Signal} = \left[\alpha\left(S(x) + \sum_{i=1}^{N} \frac{\beta_i}{N}\right)\right] - (\alpha S(x)) \cdot G$$

The factor G can be written as a portion of the iteration count N, where $G = \frac{\theta}{N}$. Here, θ is a parameter controlling the final signal amount, and $\theta \in [1,N]$. (Note: There appears to be an inconsistency in the deduction here. $G = \frac{N-\theta}{N}$ was used earlier for subtraction, but let's follow the user's final algebra.)

$$\begin{aligned}\text{New Signal} &= \left[\alpha\left(S(x) + \sum_{i=1}^{N} \frac{\beta_i}{N}\right)\right] - (\alpha S(x)) \cdot G \\ &= \alpha S(x) - \alpha S(x) \cdot G + \frac{\alpha}{N}\sum_{i=1}^{N} \beta_i \\ &= \alpha S(x)(1-G) + \frac{\alpha}{N}\sum_{i=1}^{N} \beta_i\end{aligned}$$

The remaining question is: How does this new method affect the gain?

The gain per echo, relative to the new, attenuated signal component, is:

$$\text{Gain}_i = \frac{\frac{\alpha}{N}}{\alpha(1-G)} = \frac{1}{N(1-G)}$$

If $1 - G = \frac{1}{\text{Gain}_{\text{total}}}$, then:

$$\text{Gain}_i = \frac{1}{N\left(\frac{1}{\text{Gain}_{\text{total}}}\right)} = \frac{\text{Gain}_{\text{total}}}{N}$$

The total gain of the combined echo signal (relative to the attenuated original signal) is:

$$\text{Gain}_{\text{total}} = N \cdot \text{Gain}_i = N \cdot \frac{\text{Gain}_{\text{total}}}{N} = \text{Gain}_{\text{total}}$$

Returning to the original formula for G, we can express it in terms of the desired total gain:

$$G = 1 - \frac{1}{\text{Gain}_{\text{total}}}$$

With all the variables, we can finally simplify the final formula:

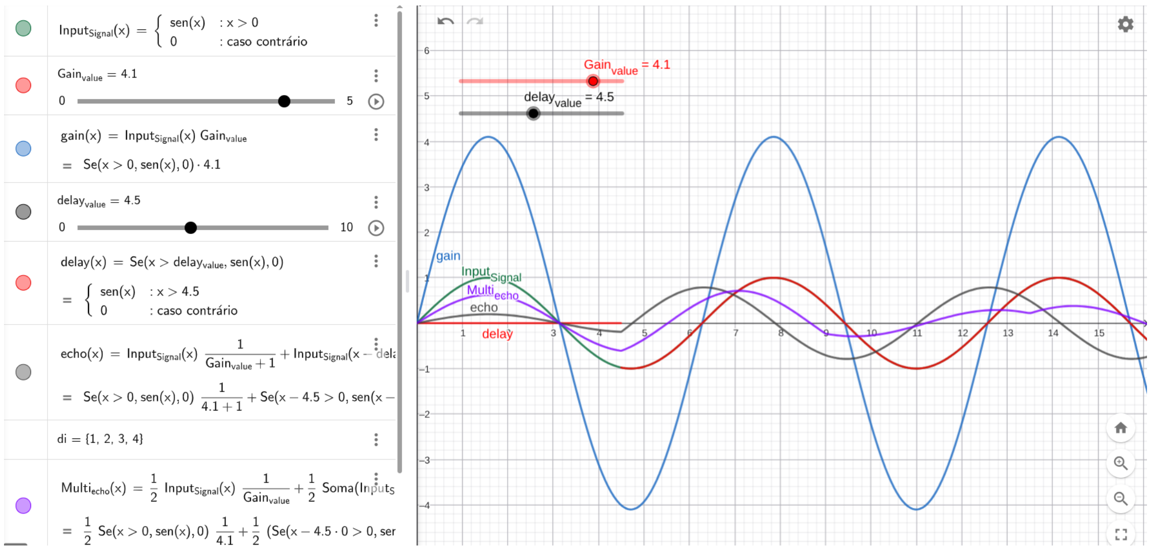
$$\alpha S(x)(1-G)+\frac{\alpha}{N}\sum_{i=1}^{N}\beta_i=\alpha S(x)\left(1-\left(1-\frac{1}{\text{Gain}_{\text{total}}}\right)\right)+\frac{\alpha}{N}\sum_{i=1}^{N}\beta_i=\alpha S(x)\left(\frac{1}{\text{Gain}_{\text{total}}}\right)+\frac{\alpha}{N}\sum_{i=1}^{N}\beta_i$$

To simplify the calculation, we assumed $\alpha=\frac{1}{2}$. This gives a reasonable value to both signals, keeping the sum of the gains at 1 by default. The final simplified formula will be:

$$Y(x)=\frac{1}{2}S(x)\left(\frac{1}{\text{Gain}_{\text{total}}}\right)+\frac{1}{2N}\sum_{i=1}^{N}\beta_i$$

$$\alpha,\text{Gain}\in[0;1]\in\mathbb{R}\wedge N,x\in\mathbb{N}_0$$

2.4.8 Diagrams and graphic previews


$\text{Input}_{\text{Signal}}(x) = \begin{cases} \text{sen}(x) & : x > 0 \\ 0 & : \text{caso contrário} \end{cases}$

$\text{Gain}_{\text{value}} = 4.1$

0 5

$\text{gain}(x) = \text{Input}_{\text{Signal}}(x)\ \text{Gain}_{\text{value}}$

$= \text{Se}(x > 0, \text{sen}(x), 0) \cdot 4.1$

$\text{delay}_{\text{value}} = 4.5$

0 10

$\text{delay}(x) = \text{Se}(x > \text{delay}_{\text{value}}, \text{sen}(x), 0)$

$= \begin{cases} \text{sen}(x) & : x > 4.5 \\ 0 & : \text{caso contrário} \end{cases}$

$\text{echo}(x) = \text{Input}_{\text{Signal}}(x) \frac{1}{\text{Gain}_{\text{value}} + 1} + \text{Input}_{\text{Signal}}(x - \text{dela}$

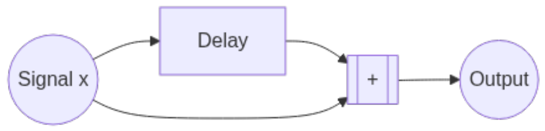
$= \text{Se}(x > 0, \text{sen}(x), 0) \frac{1}{4.1 + 1} + \text{Se}(x - 4.5 > 0, \text{sen}(x -$

$\text{di} = \{1, 2, 3, 4\}$

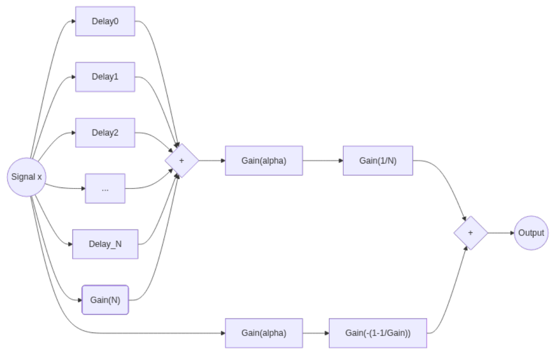
$\text{Multi}_{\text{echo}}(x) = \frac{1}{2} \text{Input}_{\text{Signal}}(x) \frac{1}{\text{Gain}_{\text{value}}} + \frac{1}{2} \text{Soma}(\text{Inputs}$

$= \frac{1}{2} \text{Se}(x > 0, \text{sen}(x), 0) \frac{1}{4.1} + \frac{1}{2} (\text{Se}(x - 4.5 \cdot 0 > 0, \text{ser}$

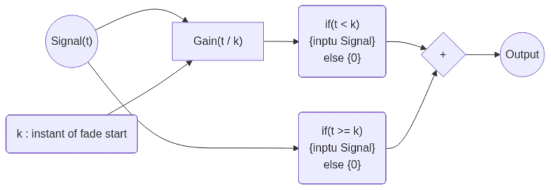
Figure 2.8: Effects preview: only the gain can be saturated; all the others never have a gain greater than the original signal.



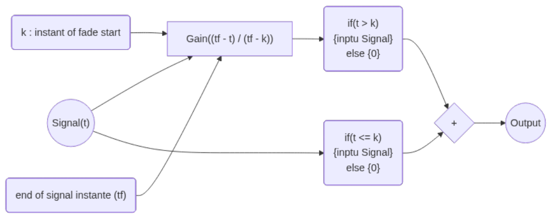
(a) Echo effect



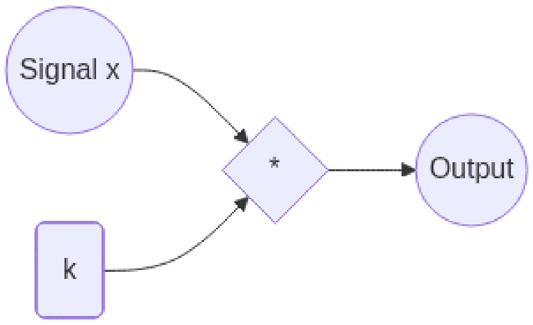
(b) Multi-echo effect


Signal(t)
Gain(t / k)
if(t < k) {inptu Signal} else {0}
+
Output
k : instant of fade start
if(t >= k) {inptu Signal} else {0}

(c) Fade in effect


k : instant of fade start
Gain((tf - t) / (tf - k))
if(t > k) {inptu Signal} else {0}
Signal(t)
+
Output
if(t <= k) {inptu Signal} else {0}
end of signal instante (tf)

(d) Fade out effect



(e) Gain effect

Figure 2.9: Diagrams of the signal transformation of each effect.

Part II

3.1 Exercise 6

In this exercise, we implement a simple codec using the `BitStream` class. The encoder, packs quantized audio samples into a compact bitstream, while the decoder, from the binary file reconstructs the quantized WAV file. Unlike in exercise 2.2, this approach produces a binary representation, allowing us to explore bit-level audio compression.

For this implementation, the encoding process begins by converting the signed audio samples into unsigned values, since the `BitStream` class cannot handle negative numbers. Next, the samples are right-shifted to perform quantization. Unlike in exercise 2.2, in this codec phase we do not left-shift the samples to restore the original range, which allows us to save more space in the binary representation. Finally, the `BitStream` class is used to write these quantized values into a packed binary format.

For the decoding, we follow the reverse procedure. First, we read the bits from the packed binary file using the `BitStream` class. Next, the samples are left-shifted to restore the original range, and finally converted back to signed values. This process allows us to reconstruct the quantized WAV file corresponding to the original audio sample.

Additionally, in this implementation, a small header was added to the binary file to store metadata, Figure 3.1. This allows the decoder to reconstruct the quantized WAV file without prior knowledge of the original sample's specifications.

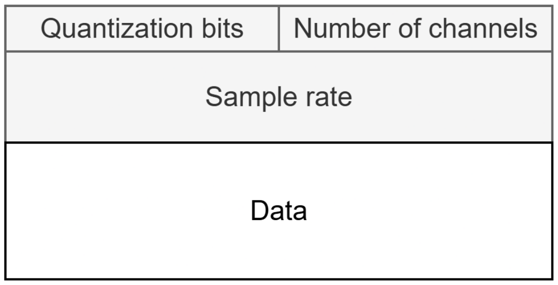


Figure 3.1: Metadata header of the quantized binary sample.

Comparing to exercise 2.2, there is no information loss during the encoding or decoding process itself. The only source of loss comes from the quantization of the bits, meaning that the output quantized samples obtained here are identical to those obtained in exercise 2.2. Additionally, when computing the samples quantized to 16 bits, the reconstructed samples are obviously equal to the original ones. Also, using the `wav_cmp` tool from exercise 2.3, all computed metrics, including RMSE, Max Error, and SNR are zero, confirming perfect reconstruction at full bit depth.

Additionally, using a binary file to store the quantized samples allows us to save space compared to a WAV file. The binary file is smaller because it stores the reduced sample values directly, whereas a quantized WAV file still uses the original fixed bit depth, resulting in larger file sizes. This difference can be seen in Table 3.1.

Table 3.1: File sizes of binary and WAV files for different quantization levels.

Quantization	**Binary file (bytes)**	**WAV file (bytes)**	**% of WAV size**
1	132 306	2 116 844	6.3%
2	264 606	2 116 844	12.5%
3	396 906	2 116 844	18.7%
4	529 206	2 116 844	25.0%
5	661 506	2 116 844	31.3%
6	793 806	2 116 844	37.5%
7	926 106	2 116 844	43.8%
8	1 058 406	2 116 844	50.0%
9	1 190 706	2 116 844	56.3%
10	1,323 006	2 116 844	62.5%
11	1 455 306	2 116 844	68.8%
12	1 587,606	2 116 844	75.0%
13	1 719 906	2 116 844	81.3%
14	1 852 206	2 116 844	87.5%
15	1 984 506	2 116 844	93.8%

Part III

4.1 Exercise 7

In this exercise, we implement a lossy audio codec that compresses audio data using the Discrete Cosine Transform (DCT). Unlike the previous exercise, which operated directly in the time domain, this codec works in the frequency domain, providing a more compact and efficient representation of the signal. The DCT concentrates most of the signal's energy in a few low-frequency components, while high-frequency components typically carry finer details that are less perceptible to the human ear. Since the human ear is less sensitive to distortions in higher-frequency sounds, these components can be quantized more coarsely or partially discarded without significantly affecting perceived audio quality.

To implement the encoder, we first divide the audio samples into blocks of fixed size, Each block is converted to mono by averaging across all channels. We then apply the Discrete Cosine Transform (DCT-II) to each block, converting the time-domain signal into frequency-domain coefficients, which concentrates most of the signal's energy in the low frequencies. The DCT coefficients are then normalized and scaled so that their values fit within the range representable by the chosen number of quantization bits (`qbits`). The scale factor for each block is stored in the binary file to allow the decoder to reconstruct the coefficients accurately. Finally, the coefficients are rounded to integers and written to the output using the `BitStream` class, producing a compact binary representation.

The decoder reverses the encoding process reading the compact binary data using `BitStream` for each block. Firstly, it reads the stored scale factor and the quantized coefficients from the binary file. The coefficients are reconstructed by reversing the scaling and normalization, and the inverse Discrete Cosine Transform (DCT-III) is applied to convert the block back to the time domain. Finally, the resulting samples are rounded and clipped to 16-bit PCM format before being written to the output WAV file.

Similarly to Exercise 3.1, a smaller header is stored at the beginning of the binary file, so that the original sample specifications do not need to be explicitly provided to the decoder, Figure 4.1.

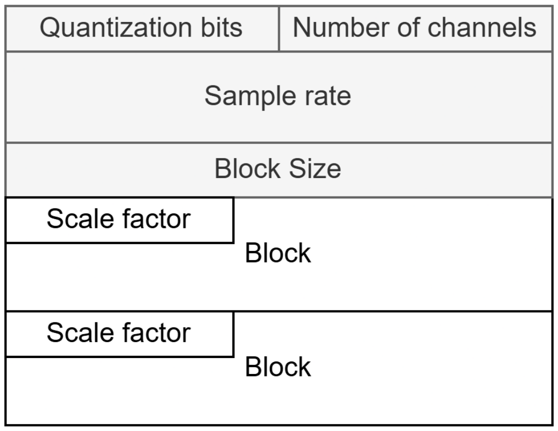


Figure 4.1: Metadata header of the quantized binary sample.

The binary file sizes produced by this lossy codec, Figure 4.1, are approximately half of those obtained in Exercise 3.1. This reduction is explained by the fact that the current codec encodes only a mono channel, whereas Exercise 3.1 processed stereo audio, having roughly double the data.

The most interesting perceptual effect observed during audio testing relates to how quantization noise appears in this lossy codec compared to the WAV files from Exercises 2.2 and 3.1. In those exercises, we manipulated the audio sample by sample in the time domain, which spreads the quantization noise evenly across the signal, producing a continuous *hiss* throughout the audio.

In contrast, this lossy codec works on blocks of samples in the frequency domain (using the DCT). Consequently, the quantization noise is concentrated in the higher-frequency components of each block. At higher numbers of quantization bits (small compression), this does not produce a continuous `hiss` but instead manifest as little sparks sound at certain moments in the audio.

Another notable effect in this lossy codec is that the little sparks in the audio are noticeable even at higher numbers of quantization bits, unlike the samples produced in previous exercises. This occurs because the codec is lossy, meaning that some information is discarded/lost during compression. Specifically, when applying the DCT, the floating-point frequency coefficients are rounded to integers during quantization, and values are sometimes clamped to fit within the allowed range.

Table 4.1: File sizes of binary and WAV files for different quantization levels.

Quantization	**Binary file (bytes)**	**WAV file (bytes)**	**% of WAV size**
1	68 252	1 058 860	6.44%
2	134 428	1 058 860	12.69%
3	200 604	1 058 860	18.96%
4	266 780	1 058 860	25.20%
5	332 956	1 058 860	31.46%
6	399 132	1 058 860	37.70%
7	465 308	1 058 860	43.95%
8	531 484	1 058 860	50.18%
9	597 660	1 058 860	56.45%
10	663 836	1 058 860	62.69%
11	730 012	1 058 860	68.96%
12	796 188	1 058 860	75.21%
13	862 364	1 058 860	81.43%
14	928 540	1 058 860	87.67%
15	994 716	1 058 860	93.93%