Informatique
iut Nord Franche-Comté

SAE 1.01-02 - BUT INFORMATIQUE - GROUPE 7

Comparative Study of Two Low-Pass Filters for Sound Data Playback: Analysis and Performance

Auteur:
Yahia KHERZA(*A1*)
Olivier MARAVAL(*A1*)

Client:
Gilles PERROT
Référent:
Olivier MARAVAL

January 10, 2024

Contents

Chapter 1

Introduction

The objective is to examine and compare two low-pass filter designs with the end goal of enhancing the clarity of text data transmitted via audio signals. This comparison is a component of a larger project, the DosOok challenge, which involves the development of the two programs DosSend and DosRead for sending and receiving text data through sound.

In this context, a low-pass filter is crucial as it allows the desired signal to pass while attenuating frequencies that are not needed. The necessity for this comparison comes from the fact that we need to find the best speed to noise reduction ratio.

For this study, two distinct low-pass filtering techniques have been selected. The criteria for their evaluation is their speed of processing while making the data readable by the DosRead program. The filter will get a pass if the initial text isn't altered after the signal has been processed. The filters must output a similar signal after processing to be comparable.

This report will detail the theoretical concepts behind each filter, the process of their implementation in Java, and the outcomes of their performance tests. The aim is to determine which filter provides a superior solution for the problem at hand.

Chapter 2

Methodology

The sound containing the data in a text file is created with the DosSend program. It will be read back with DosRead to see how efficiently it can process the signal without altering the data it contains.

2.1 DosSend and DosRead usage

2.1.1 Using DosSend to create a wav file

The program will convert the input text into an audio signal and save it as `DosOok_message.wav`. It will also print characteristics of the signal to the console and display a graphical representation of the signal waveform. It is a sinusoïdal wave modulated using a mix of OOK modulation, quick fade-in transition for 0 to 1 transitions and slow fade-out transition from 1 to 0. This was done to better understand the difference between the two filters' algorithms.

2.1.2 Using DosRead

The program will read, process (through a low pass filter), and analyze audio data from the `WAV` file. It will then output information about the file and the text corresponding to the binary sequence to a terminal. It will show a graphical representations of the original signal and it's filtered counterpart.

2.2 Explanation of the Low-Pass Filters

2.2.1 LPFilter1 : Moving Average Filter

2.2.1.1 Theory behind the Simple Moving Average Filter

The moving average filter is a basic type of Low Pass FIR (finite-impulse response) filter that is widely employed to smooth out a sequence of data points or a signal. It operates by averaging a set number, n, of input samples at once, and then outputs a single averaged data point.

2.2.1.2 Java implementation of the SMA Filter

- The method initializes an array called `filteredAudio` to store the filtered signal. This array is the same length as the input signal.
- The actual filtering is done by averaging a number of samples around each sample in the input signal.

- For each sample in the input signal, the method calculates the sum of the samples in a window centered around the current sample. The size of this window is determined by `n` and represents the cut off frequency of the filter.
- The average is calculated by dividing the sum by the count of samples included in the sum.
- This average becomes the new filtered value for the current sample position in the `filteredAudio` array.

2.2.2 LPFilter2 : Exponential Moving Average Filter

2.2.2.1 Theory behind the Exponential Moving Average Filter

The exponential moving average (EMA) filter is a Low Pass, infinite-impulse response (IIR) filter. It prioritizes recent data by giving it more weight and discounting older data in an exponential manner. In contrast to a simple moving average (SMA) this ensures that the trend is maintained by still considering a significant portion of the reactive nature of recent data points.

2.2.2.2 Java implementation of the EMA Filter

- The calculates the time constant `rc` using the formula `1 / (cutoffFreq * 2 * Math.PI)`. This time constant is used in RC (resistor-capacitor) circuits to determine the filter's response time to changes in the input signal.
- The time step `dt` is calculated as the inverse of the sampling frequency `1 / sampleFreq`.
- The `alpha` value, which determines the weight given to new samples in the EMA, is calculated using `dt / (rc + dt)`.
- The output signal array, `outputSignal`, is initialized to have the same length as the input signal.
- The first element of the output signal is set to be equal to the first element of the input signal, serving as the initial condition for the EMA.
- The method then iterates over each element in the input signal array starting from the second element.
- For each element, it applies the EMA filter using the formula : `outputSignal[i] = outputSignal[i - 1] + alpha * (inputSignal[i] - outputSignal[i - 1])`. This equation takes the previous output value and moves it towards the current input value by a fraction `alpha`. `alpha` determines how quickly the filter responds to changes in the input signal. A smaller `alpha` would make the filter respond more slowly, emphasizing lower frequencies and attenuating higher frequencies more strongly.
- After processing all samples, the method returns the outputSignal array, which contains the low-pass filtered signal.

2.3 Method to monitor the speed of the filters

Each filter will process 3 files : A short one (two words), A medium one (one paragraph, 97 words), A long one (10 paragraphs, 939 words). Each of the wav files will have been created with DosSend.java and processed within DosRead.java. This will create ideals conditions for the filters as the noise will be minimal. For each files 3 cut off frequencies will be used against a 1kHz signal and each of them must not alter DosRead decoding capability.

Chapter 3

Results

3.1 Speed

3.1.1 LPFilter's Speed to Filtering Strength

Cut Off Frequency Hz	SMA processing time (ms)	EMA processing time (ms)
20	2788	947
500	11980	972
1000	26930	972

Table 3.1: Processing Time Comparison

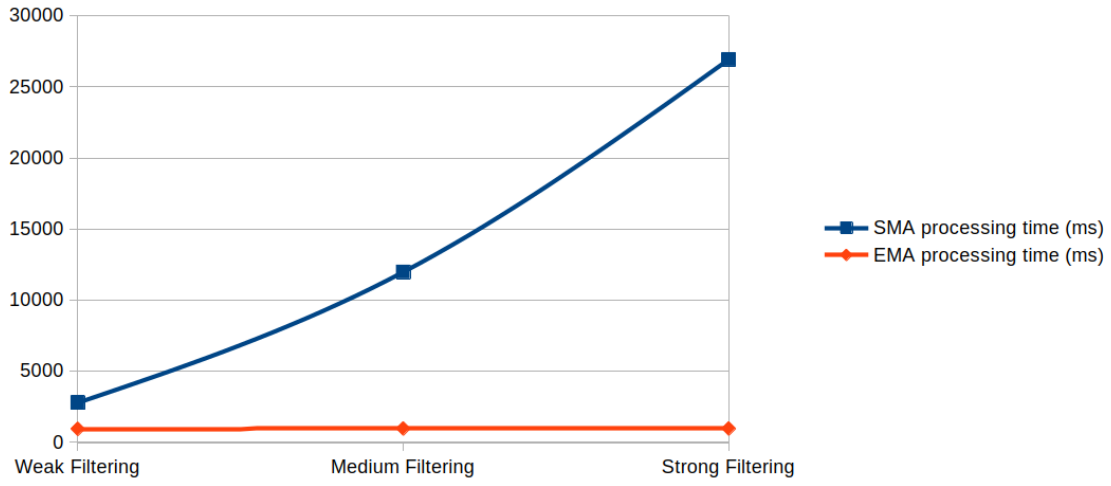

Figure 3.1: Processing speed of each filter as their cut off frequency decreases

3.1.2 LPFilter's Speed to Data Length

3.1.3 LPFilter's Efficiency

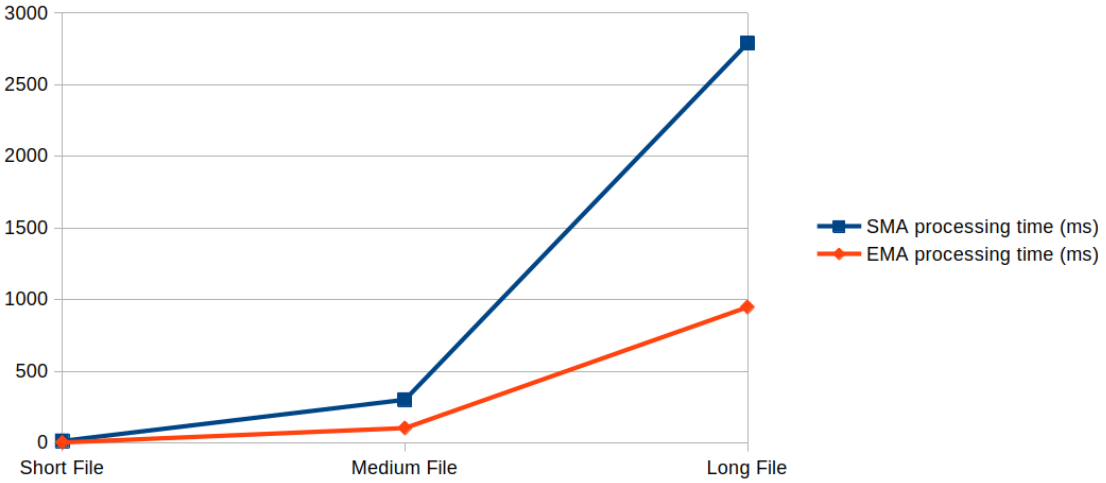

Figure 3.2: Processing speed of each filters as the length of the data increases

File	Characters	SMA processing time (ms)	EMA processing time (ms)
Short File	**13**	14	3
Medium File	**641**	301	104
Long File	**6307**	2788	947

Table 3.2: Processing Time Comparison

File	Characters	SMA processing time (ms)	EMA processing time (ms)
Short File	**13**	14	3
Medium File	**641**	301	104
Long File	**6307**	2788	947

Table 3.3: Processing Time Comparison

3.1.4 LPFilter's Output

3.1.4.1 Filters Frequency Response and efficiency

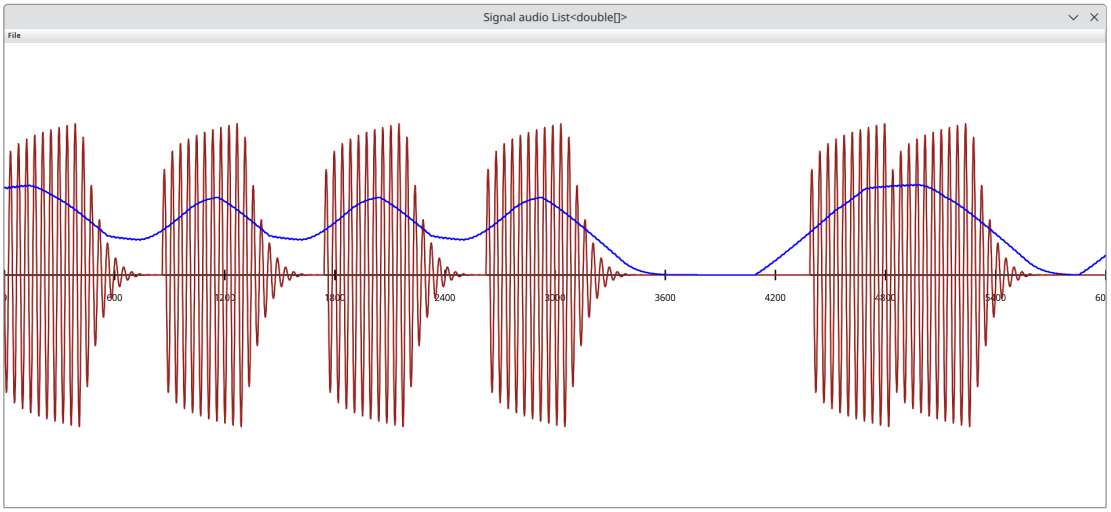

Figure 3.3: **SMA** Filter output in blue for a cut off frequency of **20Hz**

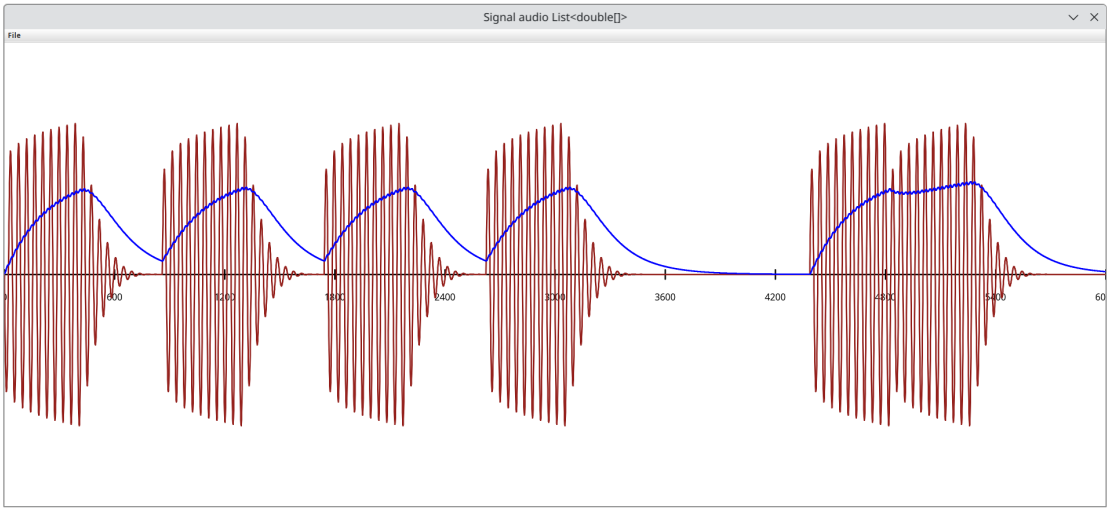

Figure 3.4: **EMA** Filter output in blue for a cut off frequency of **20Hz**

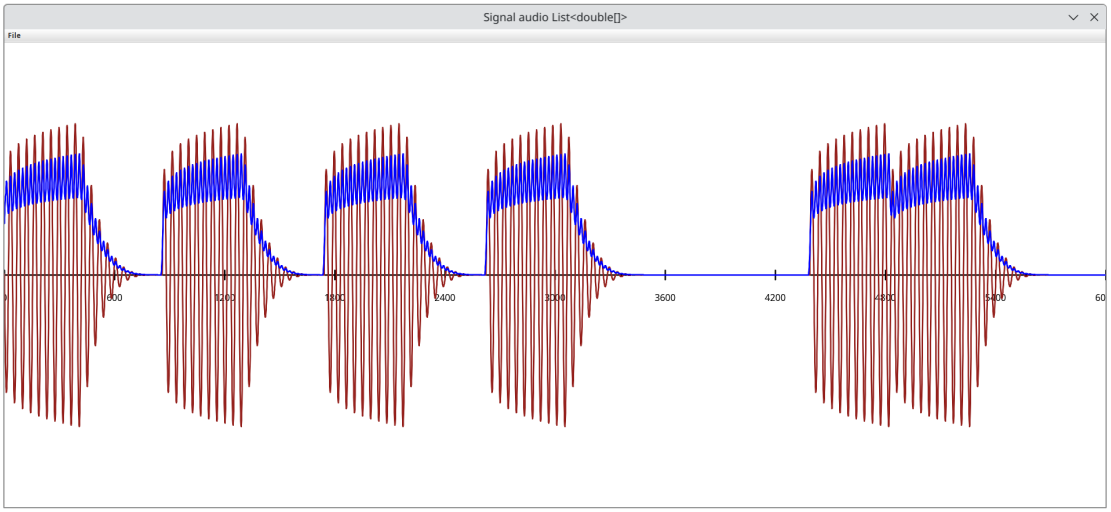

Figure 3.5: **SMA** Filter output in blue for a cut off frequency of **1kHz**

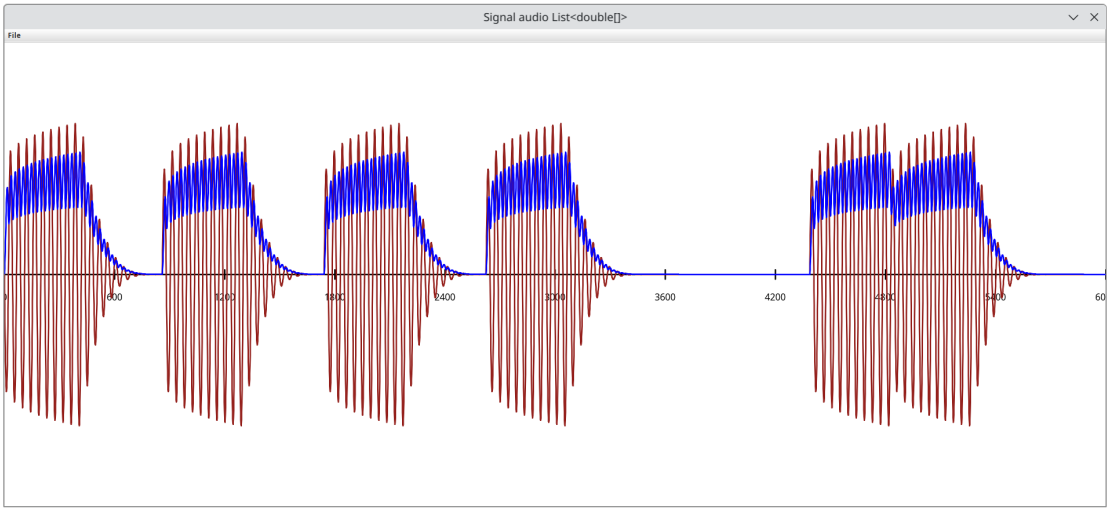

Figure 3.6: **EMA** Filter output in blue for a cut off frequency of **1kHz**

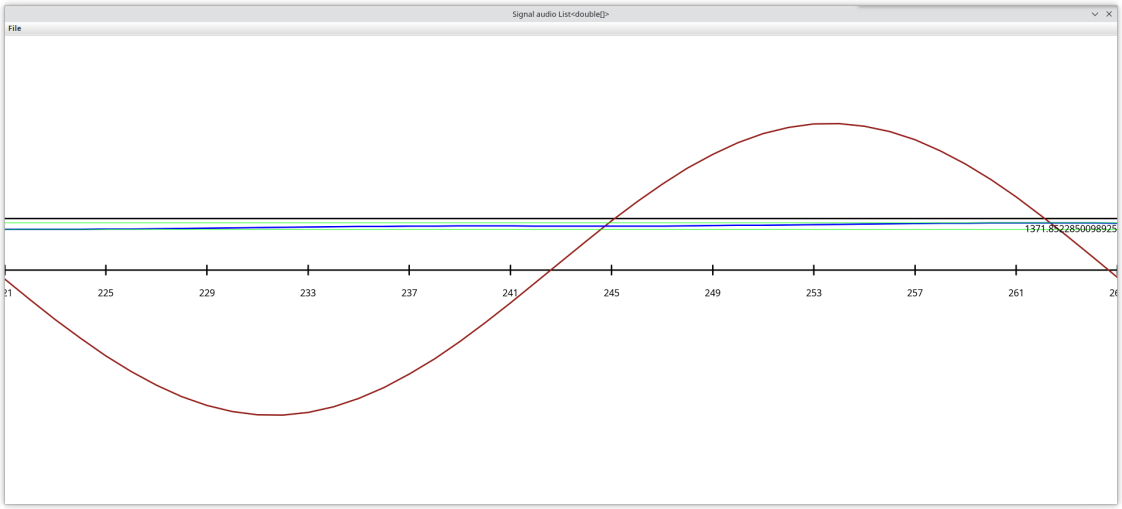

Figure 3.7: **EMA** Delta of the signal amplitude for a cut off frequency value of 20Hz : 1371

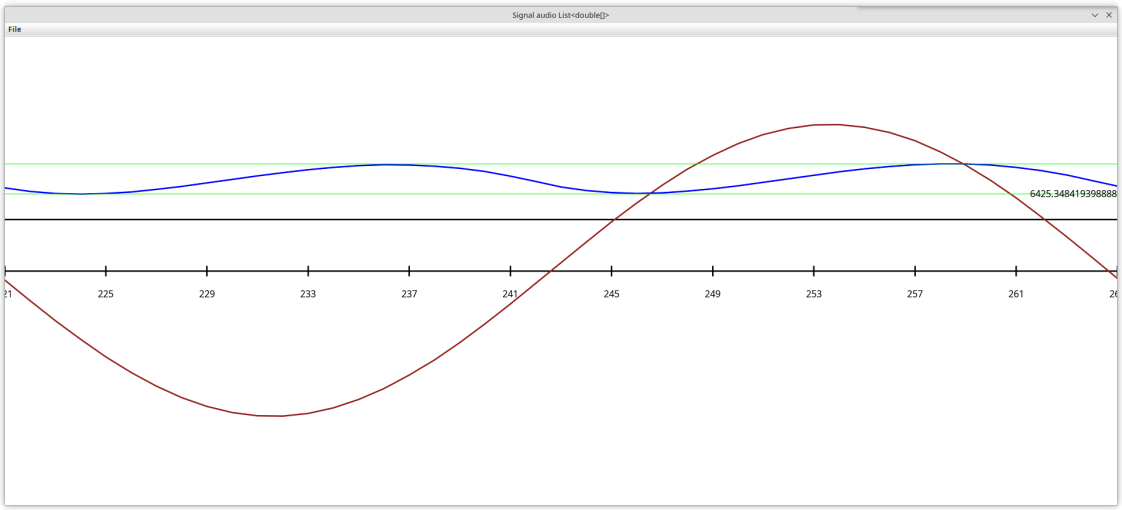

Figure 3.8: **EMA** Delta of the signal amplitude for a cut off frequency value of 500Hz : 6425

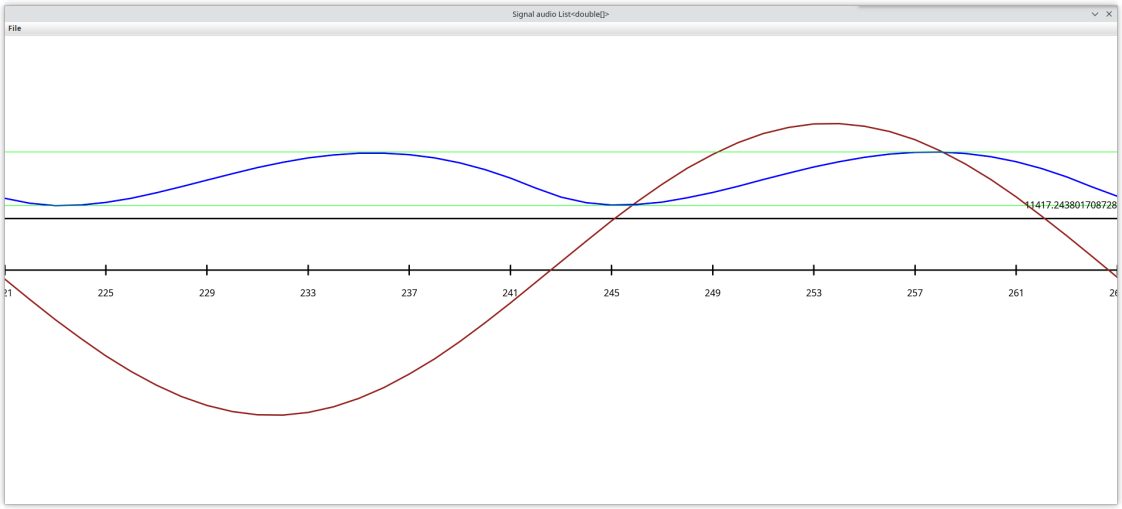

Figure 3.9: **EMA** Delta of the signal amplitude for a cut off frequency value of 1000Hz : 11417

Chapter 4

Discussion

Both filters worked correctly for cut off values around 20 Hz making the delta between maximum and minimum amplitude for a period unoticable, going higher however increased that delta as seen by the signal oscillation being visible again, going above 1000Hz made the signal unreadable as it's amplitude was too fluctuent and needed a lower threshold that made noise be read as 1 instead of 0.

As both filters return the same kind of signal after processing a sinusoïd, it can be said that they are both usable for the same kind of work. It is important to note that the EMA filter offers a sharper frequency response rate as it drops and rise quicker than the SMA filter for a low cut off frequency. For a higher cut off frequency value, both filter seems to behave similarly keeping the shape of the original signal but with significant noise.

Performance wise the SMA filter is way behind the EMA Filter with processing getting longer as the number of samples it needs to filter the signal rises. On the other hand, the EMA filter is not impacted by the value for it's cut off frequency making it better than the SMA filter as the frequency of signal to process increases. This is reflected on the time it takes for a filter to process an increasingly bigger file. There is already an 11 ms difference for a file containing 13 characters but it goes up to a 2 seconds difference for a 6307 characters long file. An other way to interpret this data is to consider that it takes the SMA filter 4.27 ms of processing per character while the EMA filter hovers around 0.15 ms per character.

Chapter 5

Conclusion

Looking at the data, there seems to be no benefits to using the SMA filter instead of an EMA filter to filter out the high frequencies of a sinousoïdal signal. The EMA filter is quicker and sharper than the SMA filter.