A novel Make-Up Gain stage for the software-based Moog 4-pole audio filter

Jaypat Piamjariyakul

May 2021

Final year project thesis submitted in support of the degree of Master of Engineering in Electrical and Electronic Engineering

Department of Electrical & Electronic Engineering

University of Bristol

Declaration and Disclaimer

Signed: Jaypat Piamjariyakul

Dated: 4th May 2021

Abstract

The objective of this project is to develop a novel method of applying the 'make-up' gain to a filtered signal such that its loudness would be restored to that of the original unfiltered signal, with particular focus on the output of the Moog voltage-controlled low-pass filter. Many previous works have attempted to capture, with varying successes, the analogue nonlinearities of the filter which gives it the distinctive 'warm' distorted output. This makes developing the gain stage difficult, as the nonlinear properties of the filter made analytical derivations of the required amplification impossible. As such, a numerical approach in the time domain, as opposed to the commonly-discussed frequency domain, was used instead.

Loudness, as a psychoacoustic concept, is a subjective term describing how a set of audio signals is perceived by the human auditory system. Using the ITU-R BS.1770 loudness standard, an algorithm based on the moving average filter was developed to evaluate the loudness of a signal in real-time with low memory requirements and fast speed, with a preceding 'K-weighting' scheme that simulates how the human auditory system perceives different frequencies in the input and filtered signals.

The make-up gain algorithm was then implemented in MATLAB to evaluate the loudness of the input and filtered signals to obtain the set of gain values necessary to increase the loudness of the filtered signal to that of the input's. The developed method demonstrated that the signal loudness was successfully restored, however the algorithm had issues with determining loudness of brief signal pulses instead of long-running signals, where the algorithm has little issues with evaluating loudnesses of the latter. The finalised system was implemented as an audio plugin that demonstrates its real-time capabilities.

Contents

Acronyms

VST	Virtual Studio Technology
DAW	Digital Audio Workstation
SDK	Software Development Kit
SPL	Sound Pressure Level
ATH	Absolute Threshold of Hearing
RMS	Root Mean Square
ISO	International Organization for Standardization
ITU	International Telecommunications Union
ITU-R	ITU, Radiocommunications Sector
EBU	European Broadcast Union
RLB	Revised Low-Frequency B-Curve
LKFS	Loudness, K-weighted, Relative to Nominal Full Scale
LUFS	Loudness Unit Relative to Full Scale
LU	Loudness Unit (non-relative)
AGC	Automatic Gain Control
VCA	Voltage-Controlled Amplifier
ADC	Analogue-Digital Converter
VCF	Voltage-Controlled Filter
DSP	Digital Signal Processing
LPF	Low-Pass Filter
FIR	Finite Impulse Response
SMA	Simple Moving Average
IIR	Infinite Impulse Response
EMA	Exponential Moving Average

1 | Introduction

1.1 Background

Dr. Robert Moog patented his design of the voltage-controlled low-pass filter (VCF, LPF) in 1965 [1] that would become a staple module in many of Moog's eponymous brand of analogue musical synthesiser products, and have been used in a variety of musical genres from electronic music by Kraftwerk to Hans Zimmer's film scores. Subsequently, a discrete-time implementation of the filter was soon realised to allow for digital implementation, either on hardware chips or software plugins for a digital audio workstation (DAW) program [2].

However, despite the many digital implementations of the Moog VCF being presented, all of these models suffer the same issue of attenuating the band-pass frequencies when only those outside the passband were to be attenuated. As such, the loudness of filtered signals (albeit not exclusive to the Moog VCF) are almost always perceived as softly in comparison to the unfiltered signal.

Given an increase in remote music production and virtual orchestration, and even more so due to the Covid-19 pandemic preventing musicians from gathering or performing live music, it is important that these digital tools are refined and available for musicians to use. One of such tools include a novel method of restoring the signal amplitude post-filtering with no intermediary human interaction required.

1.2 Motivation

Loudness has been a topic of contention between audio engineers and musicians, mainly due to how abstract and difficult to quantify auditory perception is. In 1933 Fletcher and Munson presented a series of curves that showed how the auditory system perceived different frequencies [3], while in 1953 Stevens's research showed empirically that loudness can be estimated in an exponential form [4]. It was only recently that in 2004 Soulodre investigated that loudness of an audio signal (be it analogue or digital) can be quantified using integrals and logarithms [5], and in 2006 the first sets of loudness standards were presented by the International Telecommunications Union (ITU) [6].

Meanwhile, in 1996 Stilson and Smith's paper [7] introduced multiple methods of analysing the VCF and transforming it into a discrete system, and in 2004 Huovilainen's paper [8] further improved upon that by presenting a nonlinear digital model that captures the nonlinearities intrinsic to the circuitry of the VCF. Neither parties solved the issue of loudness reduction in the output time domain waveform. Since VCFs attenuate frequencies outside the passband, this inevitably attenuates the time-domain magnitude of the signal and therefore reduces the overall volume of the output relative to the original input. The issue became more complex with Huovilainen's works, where analytically obtaining the restoration gain of the nonlinear variant was more difficult.

Daly's works in 2012 [9] suggested that, considering focus on developing software-based digital musical instruments and plugins, a scalable gain stage in the filter's passband would account for attenuation dependant on feedback. Additionally, current plugins of gain stages seldom provide automatically-adjusting gains and require users to manually adjust the applied gain.

1.3 Objectives and specifications

This project aims to go a step further from Daly's suggestion to implement a make-up gain stage that automatically adjusts to the filtered signal that would, in brief, restore the perceived output volume to match that of the original input audio, ideally as a DAW plugin.

In further details, the specifications are as follows:

- Implement an automatic make-up gain stage to the filter output, such that its perceived loudness is restored to that of the original input signal.
- Ensure that the gain stage maintains the original signal envelope and frequency contents in the amplified output signal, while minimising the output audio overshoot.
- Develop methods of allowing real-time audio input buffering and producing output with the nonlinear VCF model, and implement this with the make-up gain stage as a DAW-compatible audio plugin.

- Implement the entire system as a DAW-compatible audio plugin.

1.4 Software overview

All software packages used are listed in Table A.2 in the Appendix. MATLAB (R2020a) by MathWorks was used to implement the necessary algorithms and the full system of the project [10], while its Audio Toolbox provided the means of implementing the audio plugin as a Virtual Studio Technology (VST) file [11]. Diagrams were drawn using diagrams.net package. All relevant scripts and test audio files are uploaded to the project repository on GitHub (link to the GitHub repository) and are listed in Table A.1.

1.5 Limitations

Despite the Covid-19 pandemic, the project was entirely completed in software and did not require any hardware components. However, this inferred that only a software-based digital implementation was made and no hardware-based solutions were available due to the circumstances. Access to a replica of the Moog VCF or building the circuit in the lab was also not possible, and therefore data obtained in this project were completely derived from the software processing platform.

1.6 Contributions

This thesis contributes the following:

- A novel method of restoring loudness of a signal, relative to a reference signal, in real-time.
- Investigated several loudness-acquisitioning techniques and evaluating their accuracies and performance.
- A modular method of applying loudness-restoring amplification to the Moog VCF.
- Discussed the responses of a real-time automatic signal amplification module and its downsides in terms of performance and accuracy.
- Devised a method of using the seldom-looked feed-forward automatic gain control (AGC) system as the basis of the gain stage.
- Implemented the system as a DAW-compatible audio plugin.

2 | Literature review

2.1 Quantifying loudness

Loudness is a subjective assessment of the perceived audio magnitude, and there exists many definitions in audio literature that attempt to quantify its values [12]. Musicians assign the variations in loudness between notes and phrases as dynamics and are used extensively in western classical music [13], however these terms do not identify an absolute scale of loudness and therefore are subject to interpretation by composers and musicians. Meanwhile, audio enthusiasts often use the signal's root-mean-squared (RMS) values as a measure of loudness, however the International Telecommunications Union (ITU) had since defined a standard of loudness via its ITU-R BS.1770 standard [6], which will be discussed later.

To offer a more objective criterion into loudness, the peak value and signal energy (thus the signal power) of the audio signal can be considered instead. Considering discrete-time, the peak value of a signal at time instance n gives the instantaneous magnitude of such signal.

2.1.1 Signal energy/power and root mean square

Signal energy is defined as the summation of the square of all samples' magnitudes in a signal.

$$E_x = \sum_{n=-\infty}^{\infty} |x[n]^2| \tag{2.1}$$

where x is the signal concerned, E_x is the signal energy, and n is the index of the signal sample.

Given a finite signal of length N (where $0 < N < \infty$), Eqn. 2.1 can be altered to account for the finite energy such that $0 < E_x < \infty$:

$$E_x = \sum_{n=0}^{N-1} |x[n]^2| \tag{2.2}$$

The signal power P_x can be defined as the average of $x[n]$ signal samples squared, thus the average signal energy:

$$P_x = \frac{1}{N}\sum_{n=0}^{N-1} |x[n]^2| \equiv \frac{1}{N}E_x \tag{2.3}$$

The root mean square (RMS) value x_{rms} of a signal $x[n]$ refers to the square-root of the average power of a signal, and has the same dimensions as $x[n]$:

$$x_{rms} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} |x[n]|^2} \tag{2.4}$$

RMS values allow engineers to calculate the average power dissipation in analogue electrical components [14]. Due to RMS values in audio signals being correlated to the acoustic power dissipated in the audio equipment[1], thus the sound projected by the speaker, musicians can use these RMS values as a more reliable measure of audio loudness as compared to the signal amplitude (hence referred by musicians as the *true loudness*) and therefore is a common metric available in modern DAWs [15, 16]. As described by Kosbar, a brief signal jump may result in a high peak amplitude but does not result in a loud sound, whereas a large RMS value infers a loud signal and a small value infers a quiet signal [16].

2.1.2 Stevens's power law

In 1953 Stevens empirically derived the "psychophysical power law" defined as

$$\psi(I) = kI^{\alpha} \tag{2.5}$$

where $\psi(I)$ is the sensation magnitude of a continuum, I is the stimulus magnitude and therefore the intensity of the stimulus, k is an arbitrary constant dependent on the units of measurement, and α is the sensory factor. It was shown that the sensation magnitude exponentially grows as a power function of the stimulus magnitude. For a loudness continuum, α was derived to be 0.67. Given signal energy is directly correlated to sound intensity, I can be substituted with Eqn. 2.1. Eqn. 2.5 can then be reduced to

$$\psi(E_x) = kE_x^{0.67} = k\left(\sum_{n=0}^{N-1} |x[n]^2|\right)^{0.67} \tag{2.6}$$

[1]Audio literature often erroneously equate RMS power of a signal to its *average power* which is incorrect: average power is proportional to the square of the RMS voltage or current in a resistive load.

where N is the finite length of the signal. The power law therefore provides an estimate to loudness in terms of signal energy [4], however is not as a widely used metric in modern DAWs as the RMS meter, and is ultimately not a reliable metric due to its static nature.

2.2 Frequency weighting techniques

While RMS and peak values allow engineers and musicians to determine the objective magnitudes of a signal, neither accounted for the psychoacoustic aspects in loudness perception. The human auditory system behaves as a band-pass filter such that some frequency bands (within the auditory range of 20 to $20,000$ Hz) are more audibly perceived than others. As such, frequency normalisation techniques are devised to compensate for the psychoacoustic factors.

2.2.1 Fletcher-Munson Equal Loudness Curves

In 1933 Fletcher and Munson conducted their experiment into how the human ear perceives different frequencies at different levels, and presented their Fletcher-Munson curves, one of the first *equal-loudness contours* as shown in blue per Fig. 2.1 [3]. A curve was generated by varying the sound intensity, referred as the sound pressure level (SPL), of a reference pure tone across a range of loudness levels, referred to as *phons*, and recorded the SPL where such loudness was perceived as identical to that of a test tone. A phon refers to a tone being as perceptually loud as a 1 kHz reference tone of the same SPL; for example, a sound of 40 phons has the same perceived loudness as a 1 kHz tone at 40 dB SPL [12].

The curves show that equal-amplitude tones of different frequencies exhibit different levels of loudness. For example, observe the 0 phons Fletcher-Munson contour, where a 1 kHz reference tone with SPL of 0 dB exhibits the same perceived loudness as a 100 Hz test tone with 20 dB SPL. As the loudness level increases, the curve shifts towards higher SPLs.

It was deduced that the perception of loudness in audio behaved akin to a band-pass filter, where frequencies beyond certain frequency bands are more attenuated than those within the passband. Since then, there have been many attempts at standardising equal loudness contours, eventually resulting in the ISO 226 standard as shown in red per Fig. 2.1 [17].

The absolute threshold of hearing (ATH) was defined such that, for a given frequency, the sound cannot be perceived (considering an average person) below such threshold [12], and is shown in Fig. 2.1 where the ATH was found to closely resemble the lowest-phon contour of the Fletcher-Munson curves.

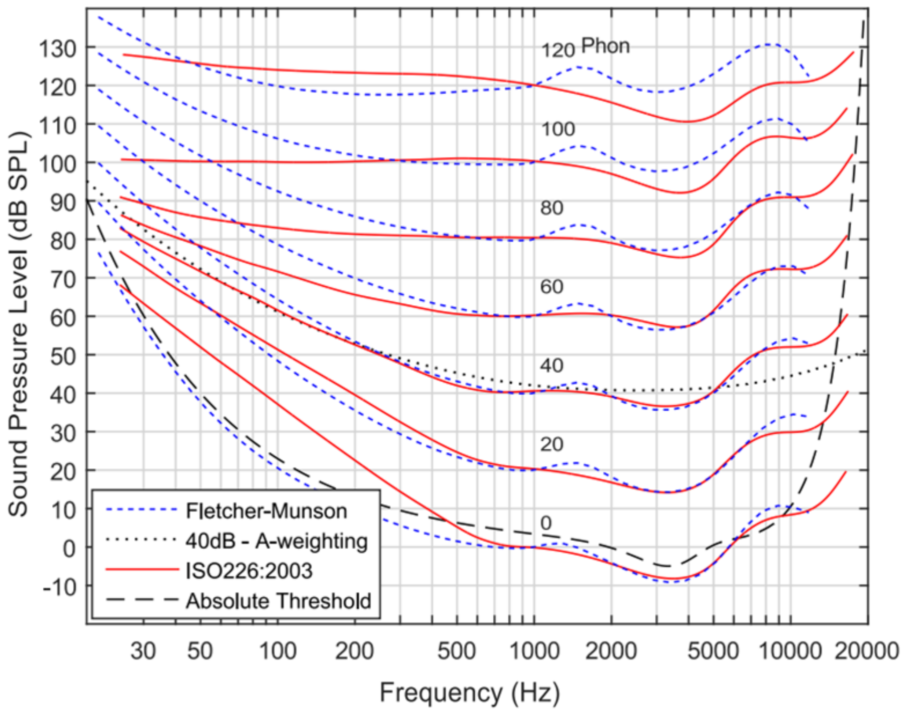


FIGURE 2.1: A comparison between the Fletcher-Munson and ISO 226 equal loudness contours exhibit similar loudness-frequency response shapes, however the ISO 226 variant show steeper responses at low frequencies and flatter responses at higher phons. The A-weighting scheme (shown in a dashed black line crossing the 40-phon threshold at 1 kHz) provides a universal SPL compensation format that can be applied regardless of frequencies or sound intensities, and is based on the 40-phon equal loudness contour. Obtained from [12], original plots and specifications from [3, 17, 18].

2.2.2 A-weighting

Since the human auditory system exhibits behaviors resembling a band-pass filter where auditory sensitivity is reduced for lower and (albeit to a lesser extent) very high frequencies, frequency compensation schemes for SPL measurements were devised. A-weighting, as shown in Fig. 2.1, was developed based on the 40-phon curve to be a simplified weighting scheme for all frequencies and sound intensities; this made A-weighting a very easy-to-implement and flexible frequency compensation scheme.

However, since analysing the shapes of equal loudness curves are dependent on the loudness (i.e. intensity) of the reference tone as discussed in Fig. 2.1, the scheme was criticised for ineffectively compensating for low-frequencies at higher phons, and its failure to represent perception of noise instead of pure tones (which were used in defining A-weighting). Other weighting techniques were soon devised to address these shortcomings.

2.2.3 Measuring the equivalent continuous sound level

In 2004 Soulodre conducted an ITU-R (ITU, Radiocommunication Sector) investigation in determining suitable objective metering techniques for obtaining the perceived loudness of a multi-channel signal. He developed the $L_{eq}(W)$ loudness metering algorithm as a performance baseline for the other presented metering techniques [5, 6].

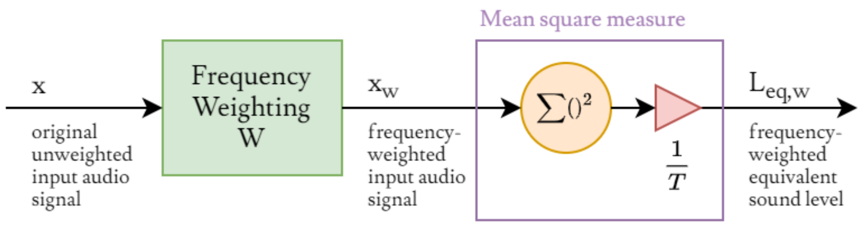

Mean square measure
x
original unweighted input audio signal
Frequency Weighting W
x_w
frequency-weighted input audio signal
∑()²
1/T
L_eq,w
frequency-weighted equivalent sound level

FIGURE 2.2: The $L_{eq,W}$ algorithm is a frequency-weightable equivalent sound level measure. Processing can be done with simple time-domain blocks with very low computational requirements. Modified from [6].

The algorithm to obtain the equivalent sound level (L_{eq}) as described by Fig. 2.2 calculates the mean-square measure of a sequence, and evaluate the frequency-compensated energy (per Eqn. 2.1) in dB [5]:

$$L_{eq,w}(t) = 10\log_{10}\left[\frac{1}{T}\int_0^T \left(\frac{x_w^2}{x_{ref}^2}\right) dt\right] \equiv 20\log_{10}\sqrt{\frac{1}{T}\int_0^T \left(\frac{x_w}{x_{ref}}\right)^2 dt}, \quad dB_w \tag{2.7}$$

where the subscript w denotes the frequency weighting scheme used, T is the duration of the discrete-time audio sequence, x is the weighted signal at the output of the weighting filter and thus input of the meter, and x_{ref} is some reference level, typically the *absolute threshold of hearing* of 20μ Pa. An unweighted signal would result in L_{eq} of units dB, whereas an A-weighted signal results in $L_{eq,A}$ with units dB, A.

It was determined that the L_{eq} algorithm utilises simple time-domain blocks with very low computational requirements [6]. The unweighted L_{eq} corresponded to the RMS of the signal which, according to Soulodre, obtained a good performance [5].

2.2.4 Soulodre's Revised low-frequency B-curve weighting

The best performance in Soulodre's study was obtained by applying the revised low-frequency B-curve (RLB) weighting scheme to the input signal (thereby obtaining $L_{eq,RLB}$). As shown in Fig. 2.3 (in blue) the RLB weighting scheme is specified as a second-order high-pass filter per

Fig. 2.4, with the coefficients considering sampling rate of 48 kHz shown in Table 2.1:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{2.8}$$

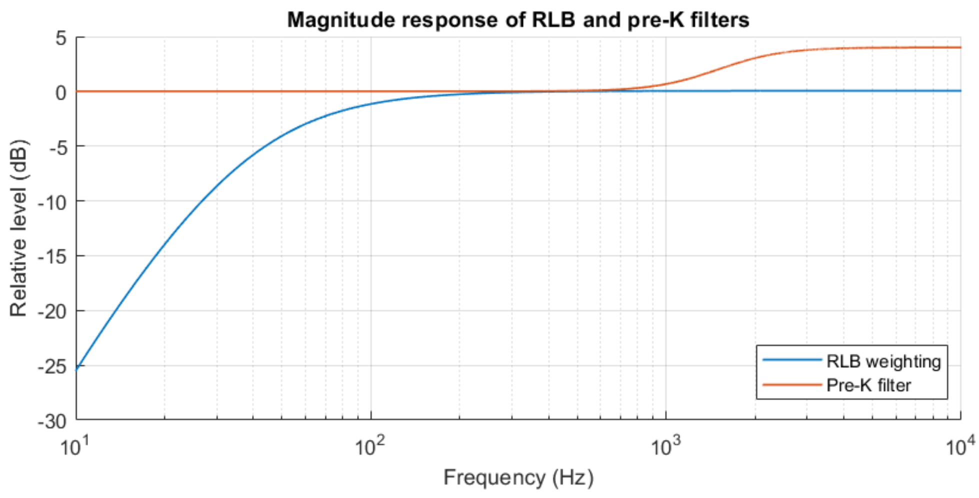


FIGURE 2.3: The RLB weighting scheme (shown in blue) consists of a simple high-pass filter, whereas the pre-filter (in orange) is a high-shelf filter. Both take the form of a second-order filter presented in Fig. 2.4 and combine to form the K-weighting filter. Generated with MATLAB using coefficients from [19] where $f_s = 48$ kHz.

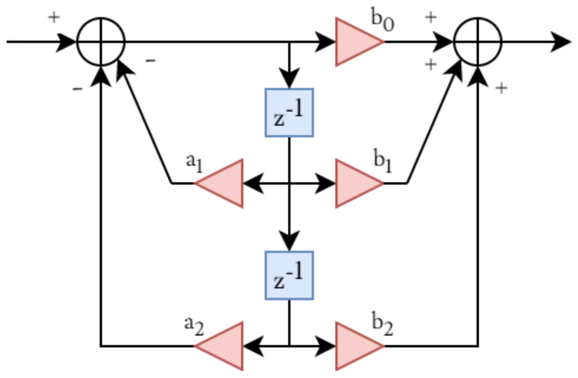


FIGURE 2.4: The RLB and K-weighting schemes can be realised as a second-order filter of the form per Eqn. 2.8. Adapted from [5].

The study showed that applying the RLB weighting scheme to $L_{eq,RLB}$ obtained outputs that showed closer correlations to the subjective loudness ratings, with the unweighted RMS equivalent showing similar correlative results.

2.2.5 K-weighting (ITU-R BS.1770)

In 2006 the ITU-R BS.1770 technical report presented a novel algorithm for objectively measuring the perceived loudness of multi-channel audio signals, designed to be implemented with low-cost components. This algorithm, instead of the RLB weighting filter, uses a new frequency-weighting scheme called "K-weighting", and is formed from a cascade of the RLB filter and a

Coefficients	Pre-filter (high-shelf)	RLB (high-pass)
a_1	-1.99004745483398	-1.69065929318241
a_2	0.99007225036621	0.73248077421585
b_0	1.0	1.53512485958697
b_1	-2.0	-2.69169618940638
b_2	1.0	1.19839281085285

TABLE 2.1: The coefficients to the feedforward and feedback paths of the RLB weighting and pre-filter curves per the second-order filter shown in Fig. 2.4, considering a sampling frequency of 48 kHz. Values obtained from [5].

high-shelf "pre-filter" per Fig. 2.5. The block diagram of a stereo-channel loudness measure is shown in Fig. 2.6.

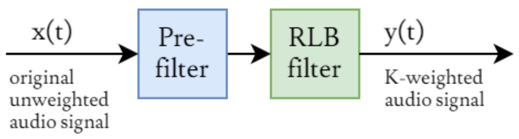


FIGURE 2.5: The K-weighting filter can be realised as a cascade of RLB filter and a pre-filtering block. Adapted from descriptions of the K-filter per [6].

Similar to the RLB filter, the pre-filter is constructed in the form of a second-order high-shelf filter per Fig. 2.4 with the coefficients shown in Table 2.1. The frequency response of such filter is shown in Fig. 2.3 (in orange), and its parameters (considering $f_s = 48$ kHz) are shown in Table 2.1. As such, the pre-filter's coefficients can be obtained with Eqn. 2.13 for a given sampling frequency.

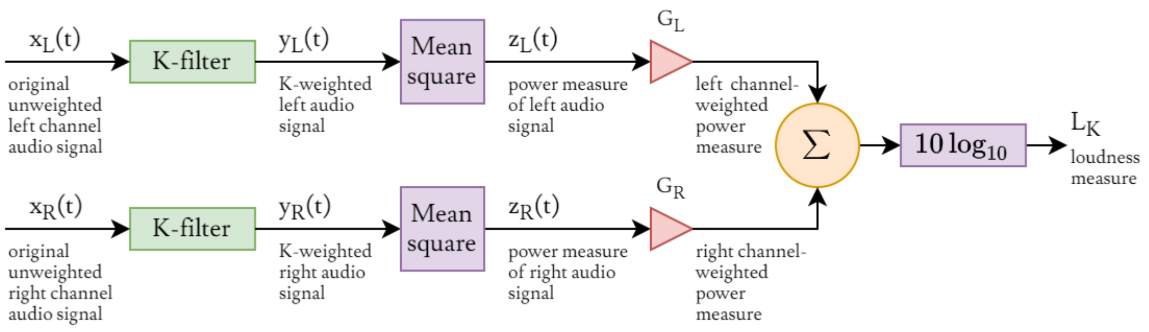

$x_L(t)$
original unweighted left channel audio signal
K-filter
$y_L(t)$
K-weighted left audio signal
Mean square
$z_L(t)$
power measure of left audio signal
G_L
left channel-weighted power measure
$x_R(t)$
original unweighted right channel audio signal
K-filter
$y_R(t)$
K-weighted right audio signal
Mean square
$z_R(t)$
power measure of right audio signal
G_R
right channel-weighted power measure
Σ
$10 \log_{10}$
L_K
loudness measure

FIGURE 2.6: The ITU-R BS.1770 multi-channel K-weighted loudness measure modifies the RLB-weighted L_{eq} measure from Fig. 2.2. The pre-filter prepending the RLB filter superimposes to form the K-weighting filter. Additional channel and gate weighting were applied. Modified from [6] to only consider left and right stereo audio channels.

As shown in Fig. 2.5, both the RLB and high-shelf filters combine to form the K-weighting filter. Its magnitude response is shown per Fig. 2.7 The mean-square of the filtered signal over interval

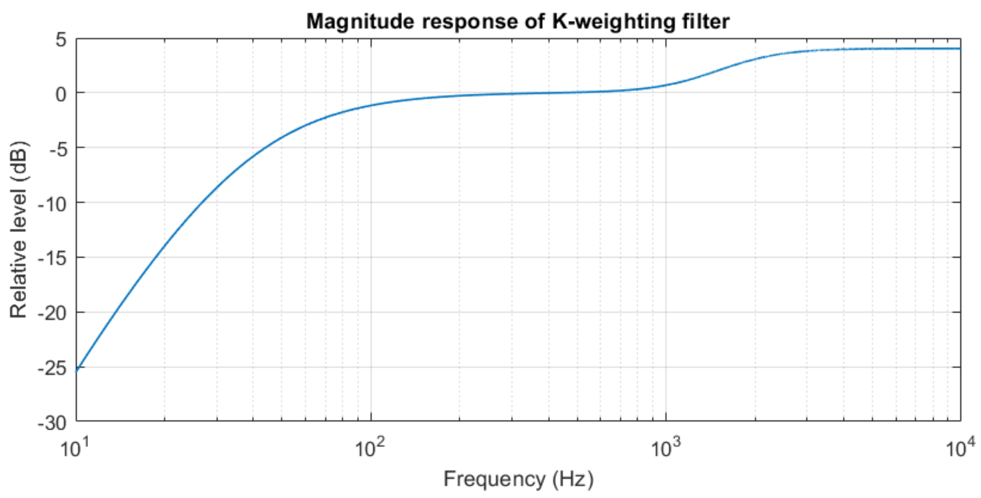


FIGURE 2.7: The K-weighting filter's magnitude response is a combination of the RLB weighting and the high-shelf pre-filter per Fig. 2.3. Generated with MATLAB using coefficients from [19] where $f_s = 48$ kHz.

T, thus the signal power, can be measured as:

$$z_i = \frac{1}{T}\int_0^T y_i^2 dt \tag{2.9}$$

where y_i is the K-weighted signal (as defined in Fig. 2.6), and suffix i denotes the type of audio channel i.e. left L, right R, central C, left-surround Ls, or right-surround Rs channels. The aggregate loudness L_k is therefore defined as:

$$L_k = 10\log_{10}\left[\sum_i G_i z_i\right] - 0.691 \qquad LKFS \tag{2.10}$$

where -0.691 compensates for the response of the K-filter at 1 kHz. For each i^{th} audio channel, an additional channel weight G_i is applied: the weight of left, right, and central channels are unity (i.e. $0\ dB$), whereas left-surround and right-surround channels have a weight of 1.41 (approximately $+1.5\ dB$) [6, 19]. The designation LKFS, the "Loudness, K-weighted, relative to nominal full scale", equates to a decibel, such that an increase in signal level by $1\ dB$ increases the meter reading by $1\ LKFS$.

2.2.6 Other techniques and developments

Since 2006 there have been considerable improvements upon the loudness model proposed by the ITU. In 2015 the fourth revision of the ITU-R BS.1770 had appended an audio-gating stage to the system presented in Fig. 2.6 where a set of LKFS thresholds were presented such that the quiet periods in the audio signal do not influence the calculated average, and therefore only signal powers of time-domain blocks above such threshold contribute to the final LKFS measurements [20, 19].

The European Broadcast Union (EBU) proposed the 'EBU-mode' meter in 2011 that measures real-time loudness in accordance to [20], using three measurement methods of momentary loudness, short-term loudness, and integrated loudness; the latter utilises the gating procedure described by ITU-R BS.1770. The EBU also introduced the "Loudness unit relative to full scale", the LUFS, which is a synonym to the LKFS but ensures that the naming convention is compliant with other existing standards. Furthermore, the EBU had suggested that loudness measurement be categorised into either of the following:

- **Relative** measurement, e.g. to a reference level or a range (i.e. $L_k = xx.x\ LU$)
- **Absolute** measurement (i.e. $L_k = xx.x\ LUFS$)

where, as discussed in [20], L_k is compliant to the naming convention L_w such that w indicates the frequency weighting scheme used. These changes would also ensure that, given LUFS indicating the value of L_k with reference to digital full-scale, the values of L_k and LUFS are equivalent [21].

Additionally, the previously explored methods only considered single-band spectra, which do not account for more complex spectral effects on perception of loudness, and outputs based on a time-series [19]. Skovenborg and Nielsen proposed a novel long-term loudness metering method in 2004 called LARM as part of his TC Electronic study [22] which, instead of outputting a time series, estimates the global loudness, found by calculating the power mean of the amplitude envelope.

2.3 Calculating second-order filter coefficients

As Soulodre's technical report only presented the filter coefficients for $f_s = 48$ kHz, in 2017 Ward presented a method of determining the coefficients of the second-order filter for any sampling frequency by considering its continuous biquadratic transfer function [19]:

$$H(s) = \frac{V_H s^2 + V_B \frac{\omega}{Q} s + V_L \omega^2}{s^2 + \frac{\omega}{Q} s + \omega^2} \tag{2.11}$$

where V_H, V_B, and V_L respectively represent the high-pass, band-pass, and low-pass filter gains. Q is the quality factor and ω is the angular cut-off frequency f_c in radians per second ($rads^{-1}$).

Since the RLB and pre-K weighting filters presented in [5] uses the form presented in Eqn. 2.8, mapping the transfer function to digital domain where $s = \frac{1-z^{-1}}{1+z^{-1}}$ and $\omega \rightarrow \Omega = tan\left(\pi \frac{f_c}{f_s}\right)$ yielded:

$$H(z) = \frac{\left(V_L\Omega^2 + V_B\frac{\Omega}{Q} + V_H\right) + 2\left(V_L\Omega^2 - V_H\right)z^{-1} + \left(V_L\Omega^2 - V_B\frac{\Omega}{Q} + V_H\right)z^{-2}}{\left(Q^2 + \frac{\Omega}{Q} + 1\right) + 2\left(\Omega^2 - 1\right)z^{-1} + \left(Q^2 - \frac{\Omega}{Q} + 1\right)z^{-2}} \tag{2.12}$$

Given the known filter coefficients for $f_s = 48$ kHz, equating the coefficients of Eqn. 2.12 to Eqn. 2.8 yielded the filter coefficients in terms of Ω and the other parameters, which can then be normalised to a_0:

$$\begin{aligned} b_0 &= V_L\Omega^2 + V_B\frac{\Omega}{Q} + V_H & a_0 &= Q^2 + \frac{\Omega}{Q} + 1 \\ b_1 &= 2\left(V_L\Omega^2 - V_H\right) & a_1 &= 2\left(\Omega^2 - 1\right) \\ b_2 &= V_L\Omega^2 - V_B\frac{\Omega}{Q} + V_H & a_2 &= \Omega^2 - \frac{\Omega}{Q} + 1 \end{aligned} \tag{2.13}$$

The filter parameters for the high-pass RLB and high-shelf pre-K weighting filters, as described in [5, 6] where $f_s = 48$ kHz, is shown in Table 2.2. As shown in Eqn. 2.13, the coefficients are Ω-dependent, therefore only Ω needs to be updated for a new sampling frequency f_s.

Parameters	Pre-filter (high-shelf)	RLB (high-pass)
Q	0.7071752	0.5003270
V_L	1	0
V_B	1.2587209	0
V_H	1.5848647	1.0049949
f_c (Hz)	1681.9744510	38.1354709
Ω (rad)	0.1105318	0.0024960

TABLE 2.2: Parameters of the high-shelf and high-pass filters define the RLB and K-weighting filters for $f_s = 48$ kHz. Applying these to Eqn. 2.13 yielded the filter coefficients as described in [5, 6]. Adapted from [19].

2.4 Automatic gain control

Automatic gain control (AGC) is a type of control system that, given dynamic input signal strengths, automatically adjusts the output of a system to a rated level by controlling a variable gain amplifier, essentially maintaining an almost-constant output level independent of the input signal level. These systems often ensure that the subsequent blocks utilising the AGC output, such as an analogue-digital converter (ADC), would not have a saturated input dynamic range, and prevent the system from falling below the tolerable noise level [23].

AGC is a nonlinear, time-dependent, and signal-varying feed system, and as such is very difficult to analyse using time-domain or z-domain analysis techniques. This results in AGC analysis to be empirical in nature, and hence why there hasn't been much discussion of AGCs in digital signal processing (DSP) literature [14]. Despite the lack of analytical studies, many successful applications have been made that utilise these AGC techniques including a number of audio DSP tools [24].

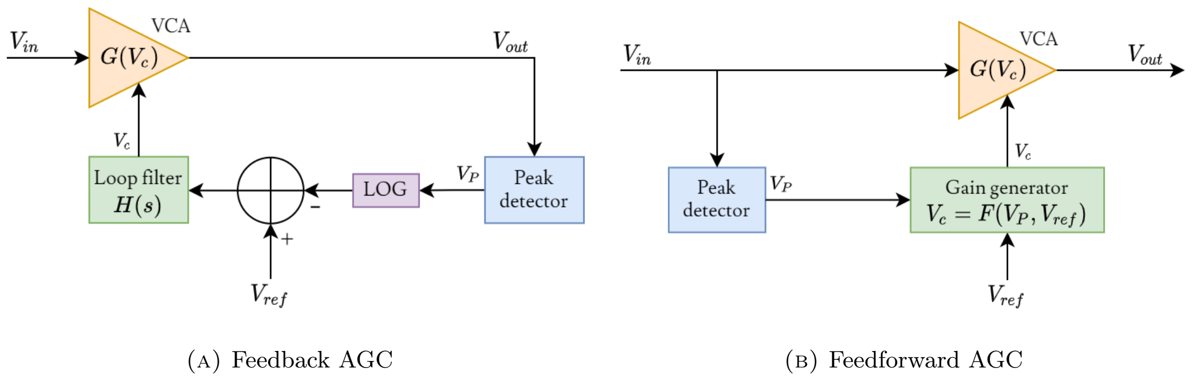


Figure 2.8: The AGC loops can be either feedback or feedforward. Feedback systems generate the desired control signal by observing the output of the VCA V_{out}, whereas feedforward systems observe the input V_{in}. Both systems compare the observed levels to the reference signal V_{ref}. Redrawn from [23].

The variable-controlled amplifier (VCA) is a signal-conditioning amplifier with a tunable gain function $G(V_c)$, such that it amplifies or attenuates an incoming signal to the desired level depending on the value of V_c, the control voltage into the VCA, as set by the control loop.

A feedback AGC control loop observes the VCA output V_{out} and compares it to the reference voltage V_{ref} such that we want V_{out} set to this value. Meanwhile, a feedforward system observes the current input into the VCA and compares its value to the reference voltage. Both types generate the control voltage V_c that sets the VCA gain $G(V_c)$. The two types of AGC loops are shown in Fig. 2.8.

The system presented utilises logarithmic scales in calculations; Lyons suggested that this is to increase the dynamic range of the AGC. A low-pass filter can also be implemented at the output of the VCA to mitigate rapid gain fluctuations.

2.5 Analysis of the analogue Moog transistor ladder VCF

In brief, the Moog VCF is a transistor ladder circuit of four identical buffered stages [1] as shown in Fig. 2.9, with each stage comprising of two identical NPN transistors and a capacitor. Each ladder stage is driven by the output current of the previous stage, while the first stage is driven

Type	Advantages	Disadvantages
Feedback	Lower input dynamic range required by peak detector Inherently higher linearity	Instabilities with high compression or expansion Higher settling-time required
Feedforward	No instability problems Ideally, zero settling-time	AGC input dynamic range required by peak detector High linearity required in loop

TABLE 2.3: Summary of comparisons between the main AGC control loop characteristics. Copied from [23].

by a differential transistor amplifier drawing the control current I_c. The differential output is drawn from the transistors' emitters at the final stage, and a portion of the output current is fed back to the first stage via a feedback path.

With each stage, the base-emitter resistances of the two transistors form a one-pole voltage-controlled RC filter, resulting in 3 dB attenuation per each stage at the cut-off frequency specified, assuming no feedback. Given the four stages in the VCF core, four one-pole filters are cascaded onto each other resulting in 12 dB total attenuation with 24 dB/octave roll-off. The magnitude response of the VCF varies with the feedback level, which can be tuned by separate knobs for the cut-off frequency and its Q-value [25].

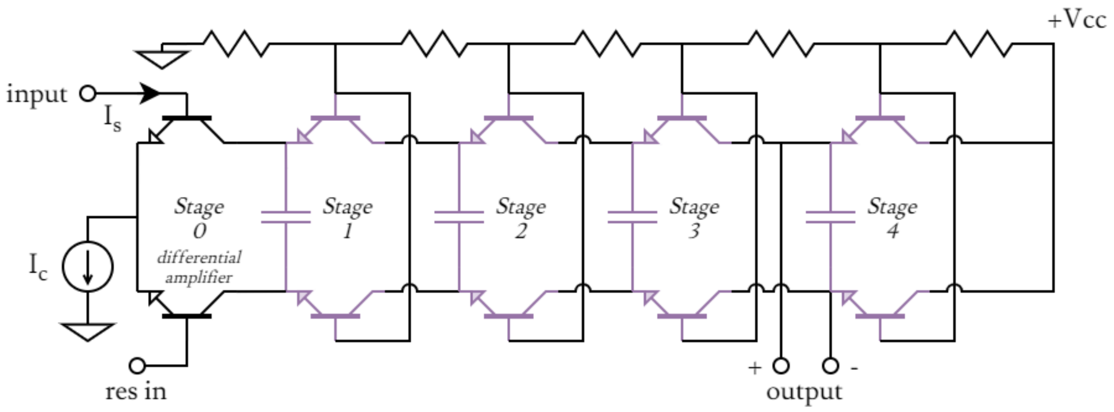


FIGURE 2.9: The core of the Moog VCF is the ladder circuit comprising of four transistor pair stages. The input signal is fed at Stage 0 (i.e. the differential-pair amplifier stage) of the ladder, while the differential output is obtained across Stage 4; part of the output is fed back as an inverted feedback signal to Stage 0 of the ladder. Adapted from [26], original schematic from [1].

Stinchcombe's paper [26] analysed the analogue Moog VCF in terms of the differential current in each stage. Defining subscripts "1" and "2" as the upper and lower transistors with respect to Fig. 2.9, for each stage V_1 and V_2 represent the transistors' base-emitter voltages, while I_1 and I_2 denote the respective collector currents.

2.5.1 Huovilainen's analysis of the differential pair

The differential output of the ladder was found by identifying the relationship between V_1, V_2, and $(I_1 - I_2)$ with the following assumptions [8, 27]:

- DC current gain β is sufficiently large such that it is considered infinite, and therefore the collector current and emitter current are equal.
- Both transistors in each stage are perfectly matched.
- Early effect is negligible.
- Resistor values at the transistors' bases are small such that the base voltages are constant for all stages, thereby ensuring each filter stage depended only on its state and the current from the previous stage, and thus analysing only one stage was needed to extrapolate the effects of the complete core.

The current different in a differential transistor pair per stage is defined as

$$I_1 - I_2 = (I_1 + I_2) tanh \left(\frac{V_1 - V_2}{2V_T} \right) \tag{2.14}$$

where $V_T \approx 25mV$ (at room temperature) is the transistor thermal voltage.

2.5.2 Differential equation for a single stage

The currents I_1 and I_2 combine such that

$$\begin{aligned} I_1 + I_2 &= 2I_{ctrl} \\ I_1 - I_2 &= 2I_{in} - 2I_c \end{aligned} \tag{2.15}$$

where I_{ctrl} is the control current of the ladder stage, I_c is the current through the capacitor, and I_{in} is the input audio signal current. Applying Eqn. 2.15 to Eqn. 2.14 shows a new relation:

$$I_c = I_{in} - I_{ctrl} tanh \left(\frac{V_c}{2V_T} \right) \tag{2.16}$$

where V_c is the voltage across the capacitor. The equation for current I_c through a capacitor of capacitance C is

$$I_c = C \frac{dV_c}{dt} \tag{2.17}$$

Replacing I_c from Eqn. 2.17 into Eqn. 2.16 gives

$$C\frac{dV_c}{dt} = I_{in} - I_{ctrl} tanh\left(\frac{V_c}{2V_T}\right) \tag{2.18}$$

Since each stage is driven by the previous one (or the differential input amplifier given the first stage), Eqn. 2.16 can be written as

$$\frac{dV_c}{dt} = \frac{I_{ctrl}}{C}\left(tanh\left(\frac{V_{in}}{2V_T}\right) - tanh\left(\frac{V_c}{2V_T}\right)\right) \tag{2.19}$$

where V_{in} is the potential difference across the capacitor from the preceding stage.

2.6 Implementation of the discrete-time VCF

2.6.1 Stilson and Smith's linear VCF analysis

Stilson and Smith's analysis [7] approximated a linear response in each stage (assuming a small-signal differential audio input signal [26]), and determined that the VCF employs the filter structure per Fig. 2.10, where each stage $G_1(s)$ of the VCF resembles a one-pole LPF:

$$G_1(s) = \frac{1}{1 + \frac{s}{\omega_c}} \tag{2.20}$$

where a real-valued pole exists if and only if $s = -\omega_c$. The cut-off frequency ω_c determines the location of $-12dB$ attenuation.

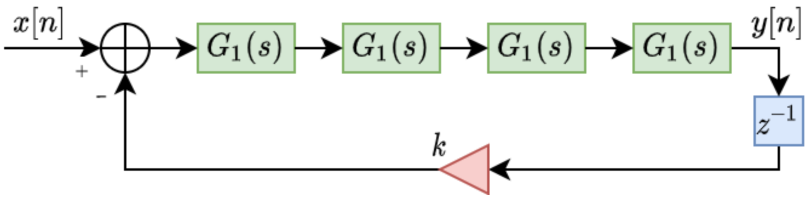


FIGURE 2.10: The Moog ladder filter can be represented as four cascading stages in open-loop, with a portion of the output fed back to the first stage. Adapted from [7].

Given a complete 4-stage core with feedback, the transfer function is

$$H(s) = \frac{Y(s)}{X(s)} = \frac{G_1(s)^4}{1 + kG_1(s)^4} \equiv \frac{1}{k + \left(1 + \frac{s}{\omega_c}\right)^4} \tag{2.21}$$

where $k = [0, 4]$ is the feedback coefficient per Eqn. 2.32. The frequency response is therefore

$$H(j\omega) = \frac{1}{k + \left(1 + \frac{j\omega}{\omega_c}\right)^4} \tag{2.22}$$

where $|H(j\omega)| \approx \frac{1}{1+k}$ when $\omega \ll \omega_c$, and $|H(j\omega)| \approx \frac{1}{\omega^4}$ as $\omega \gg \omega_c$, its behaviours reminiscent to that of a low-pass filter [27]. The variations between feedback coefficient k and cut-off frequency ω_c is seen in Fig. 2.11, where it is observed that increasing k results in a higher and narrower peak at ω_c and an increasingly attenuated pass-band.

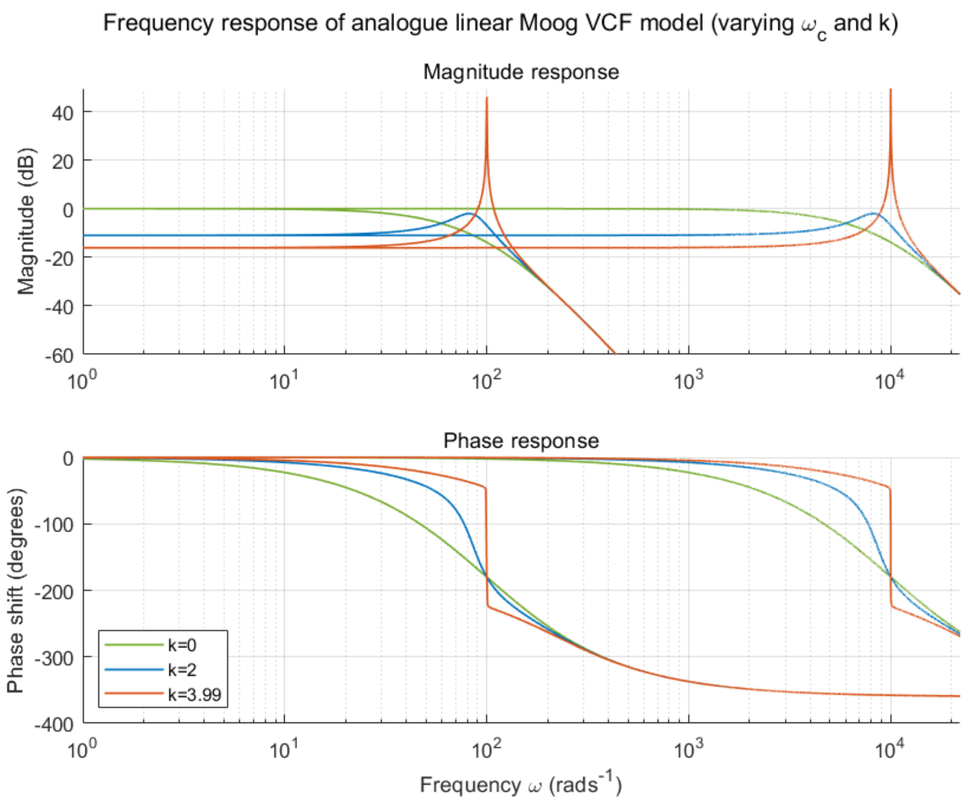


FIGURE 2.11: The magnitude response of varying $\omega_c \in (100, 10000)$ $rads^{-1}$ and $k \in (0, 2, 3.99)$ shows an increase in magnitude for frequencies around ω_c as $k \to 4$, where the most emphasised frequency is the resonant frequency. At $\omega = 0$ the phase shift is $0°$, whereas at $\omega \to \infty$ the phase shift approaches $-360°$. Meanwhile, at ω_c the feedback signal is inverted (respective to the input signal) with phase shift of $-180°$; this results in superposition and thus emphasises frequencies around ω_c. Modified from [9] and generated with my own code, original plot from [7].

Stilson and Smith determined the complex gain of each stage at ω_c [7] to be

$$G_1(j\omega_c) = \frac{1}{1+j} \equiv \frac{1}{\sqrt{2}} e^{j\frac{\pi}{4}} \tag{2.23}$$

This allowed them to obtain the complex gain of the entire 4-stage core:

$$G_1^4(j\omega_c) = \frac{1}{4} e^{j\pi} \equiv \frac{1}{4}(-1) \tag{2.24}$$

Thus, the total gain is 1/4 with an inverting phase (of $-180°$), meaning that a $\omega_c\ rads^{-1}$ input sinusoid passing through the VCF core will have an output of 25% amplitude compared to the input, and $-180°$ phase-shifted (therefore inverted). Contrasting this to $\omega = 0$ or $\omega \to \infty$ where the phases are non-inverting at $0°$ and $-360°$ respectively, as shown in Fig. 2.11.

The Moog VCF core therefore comprise of four one-pole LPF stages as shown in Fig. 2.10, each with cut-off frequency ω_c that results in an inverted signal when $\omega = \omega_c$. As k is incremented, the feedback signal provides increasing constructive interference to the input signal (due to the complex gain increasing) at frequencies around ω_c. This was called 'corner peaking' in analogue VCF design [28], and refers to the *resonance* around the cut-off frequency (or *resonant* frequency, when properly tuned) [25].

2.6.2 Stinchcombe's pole-zero analysis of the linear VCF

The analysis of the linear model by Stilson and Smith coincided with Stinchcombe's analysis using the pole-zero technique [26]. He determined that the analogue frequency response is determined by the positions of poles in the complex plane; given the linear Moog VCF model [7] (thus a small-signal input assumption), it is possible to analytically determine such pole positions by considering the denominator of the transfer function from Eqn. 2.21:

$$k + (1 + S)^4 = 0 \tag{2.25}$$

where $S = \frac{s}{\omega_c}$ is the normalised frequency [9]. S can therefore be obtained and plotted on a pole-zero plane per Fig. 2.12.

$$S = -1 + (-k)^{\frac{1}{4}} \equiv -1 \pm k^{\frac{1}{4}} e^{\pm j\frac{\pi}{4}} \tag{2.26}$$

As $k = [0, 4]$ increases, the poles diverge from $S = -1 + j0$ at the same rate in a cross-like pattern. At $k = 4$ the rightmost poles converge to $\sigma = 0$ thus the system self-oscillates at $\omega_c\ rads^{-1}$.

2.6.3 Huovilainen's implementation of the nonlinear VCF

As Stilson and Smith's analysis (and the eventual digital form of the linear VCF) provided useful information regarding the operations and the output of the Moog VCF, their paper assumed that the components are linear and the input is of a small-signal value, and thus had to ignore the nonlinearities in the analogue circuitry. These digital derivations were criticised

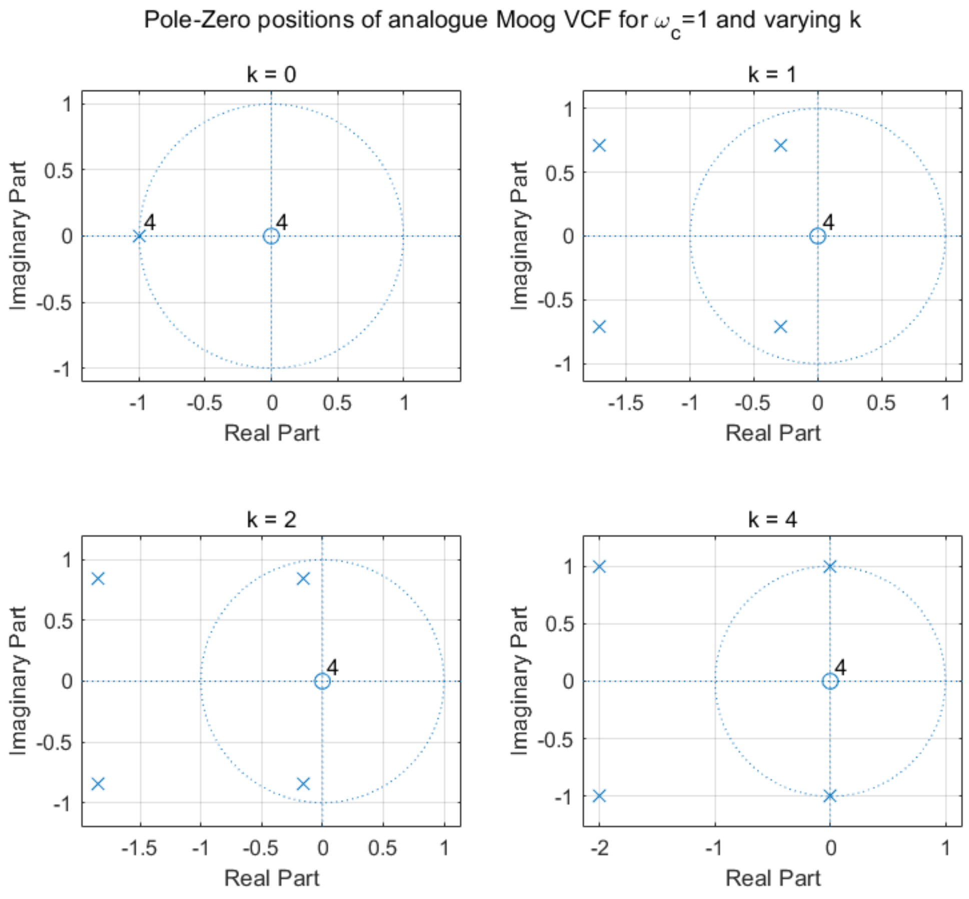


FIGURE 2.12: The 4-stage analogue VCF poles diverge in a cross-like pattern from $S = -1 + j0$ with increasing $k = [0, 4]$. At $k = 4$, the rightmost poles converge to $\sigma = 0$ resulting in the filter self-oscillating at the frequency ω_c. Adapted from [9], original 3-dimensional model from [26].

by musicians, notably in Rossum's paper [29], with the audio output lacking in "warmth"[2] or analogue distortion characteristics.

Huovilainen's paper instead analysed the analogue circuit and obtained Eqn. 2.19, where he would use Euler's method, at time step n, to numerically solve the equation:

$$V_c[n] = V_c[n-1] + \frac{I_{ctrl}}{CF_s}\left(tanh\left(\frac{V_{in}[n]}{2V_t}\right) - tanh\left(\frac{V_c[n-1]}{2V_t}\right)\right) \tag{2.27}$$

where $F_s = \frac{1}{T_s}$ is the sample rate, such that T_s is the time interval between samples. This allowed him to derive the difference equations for the full ladder filter [8].

He noted that, given small input levels ($|x| < 0.5$), the $tanh$ function is approximately linear, agreeing to Stilson and Smith's analysis, and therefore found that Eqn. 2.27 has the same form

[2]The term "warmth" is subjective both in audio engineering and musical academia, however the general and more quantifiable consensus is that "warmth" refers to an emphasis on lower to mid-range frequencies by audio equipment [30]. In this case, the lack of "warmth" would refer to the audio output lacking in emphasis of low-mid frequencies. However, tread this definition very lightly as it is ultimately anecdotal and difficult to quantify.

as a digital one-pole LPF:

$$y[n+1] = y[n] + g(x - y[n]) \tag{2.28}$$

where $g = I_{ctrl}/(2V_tCF_s)$ is an exponential weighting coefficient [31].

Applying scaled impulse invariant transform, such that the DC gain is unity, to Eqn. 2.27 [32] showed that

$$g = 1 - e^{-2\pi(F_c/F_s)} \equiv 1 - e^{-\omega_c/F_s} \tag{2.29}$$

where F_c is the cut-off frequency such that $\omega_c = 2\pi F_c$. Applying this to Eqn. 2.27 gives

$$y[n] = y[n-1] + 2V_tg\left(tanh\left(\frac{x[n]}{2V_t}\right) - tanh\left(\frac{y[n-1]}{2V_t}\right)\right) \tag{2.30}$$

Given an analogue VCF and a ω_c $rads^{-1}$ sinusoidal signal, each filter stage incurs a $-45°$ phase shift, thus resulting in a total $-180°$ overall phase shift. This phase shift and the signal inversion, as discussed by Stilson and Smith, resulted in positive feedback on frequencies around ω_c. However, the unit delay in the feedback path results in an additional phase shift, causing the total phase shift to be

$$p_{total} = 4p_{stage}(f, F_c) + 180°\frac{f}{F_s} \tag{2.31}$$

where p_{total} is the total phase shift in degrees, and p_{stage} is the phase shift of one VCF stage. The additional phase shift causes the resonance frequency to offset from the cut-off frequency, and additionally makes the feedback amount required to produce the desired resonance frequency-dependent.

Huovilainen compensated by altering the filter structure (instead of Stilson and Smith's method of using a tuning and resonance compensation tables). Applying a half-unit delay, via averaging two output samples (at n and $n-1$), allows the model's frequency response to be improved, and ensures the resonance frequency is close to F_c as possible.

2.6.4 Daly's dimensionless difference equation variants

Daly's thesis eventually aggregated the previous analyses and digital derivations [7, 8, 26], and recasted Eqn. 2.19 into a form such that their values utilise dimensionless variables [9]:

$$\begin{aligned} \frac{dv_i}{dt} &= \omega_c\left(tanh(v_{i-1}) - tanh(v_i)\right) \\ v_0 &= v_{in} - kv_4 \end{aligned} \tag{2.32}$$

where v_{in} is the input audio signal into the VCF ladder, $i = \{1, 2, 3, 4\}$ labels the four ladder stages, $k = [0, 4]$ is the feedback gain, and ω_c is the filter cut-off frequency in $rads^{-1}$.

2.7 Virtual Studio Technology (VST) plugins

Virtual Studio Technology (VST) is a type of audio plugin software interface that allows users to add functionalities to DAWs, be it synthesisers, effects, or sound libraries [33]. In 1996 Steinberg GmbH released its first VST interface specifications and the software development kit (SDK), and has since been updated to version 3.6.7 in 2017 [34]. The MATLAB Audio Toolbox, first released in 2016, allows MATLAB users to build their own VST plugins from programmed specifications [11].

3 | Design and implementation

3.1 Design parameters and assumptions

The goal of the project was to design and implement a novel make-up gain stage that would amplify or attenuate a filtered signal, such that its volume, i.e. loudness, would match to that of the original signal. At its simplest, the make-up gain is realised as an open-loop amplifier [27]:

$$x_o[n] = G_n\left(L_{i,n}, L_{f,n}\right) x_f[n] \tag{3.1}$$

where n is the current sample index in the signal, $x_o[n]$ is the desired amplified output signal, $x_f[n]$ is the filtered audio signal at sample n, and $G_n\left(L_{i,n}, L_{f,n}\right)$ is the make-up gain to apply to $x_f[n]$, such that $L_{i,n}$ and $L_{f,n}$ are the values to consider for the input and filtered signals, respectively, at sample $[n]$.

This is shown in Fig. 3.1 where the make-up gain is dependent on the values of $L_{f,n} = F(x_f[n])$ and $L_{i,n} = F(x_i[n])$ such that $F(x)$ is an arbitrary function that computes the loudness of $x[n]$, be it a single value or over a vector. As these calculations use processing time, the delay block z^{-M} was added to represent these functionalities in real-time, where M is the total delay required to calculate $L_{f,n}$, $L_{i,n}$, and G_n such that their values correspond to the same n index. Meanwhile, the delay block z^{-N} delays the original signal before entering the gain stage, where N is the total delay generated from applying the Moog 4-pole filter (or any other processing prior to the gain stage).

As such, the system has the following parameters and assumptions:

- The system is causal; calculating the current make-up gain only depends on the current and/or past values of the signals. This is realistic as real-time audio streaming does not have future values.

- The audio rate, i.e. sampling frequency, is 44.1 kHz. This sample rate was conceived to accommodate the audiovisual media stored in compact discs [35], and thus resulted in many audio equipments to run at 44.1 kHz; 48 kHz was rarely used reserved for professional usage. While the higher sampling rate became used more commonly over time, many audio equipments still operate at 44.1 kHz in spite of no available discussions for the smaller sampling rate in the reviewed literatures [5, 6, 19], and as such is a subject of study. Note that other sampling frequencies are applicable with further calculations, however the subject of this thesis is the sampling frequency 44.1 kHz.

- There are no previous audio signals stored in the system before initialisation (i.e. $n = 0$) (therefore no prior loudness was stored), thus initial conditions are $x_i[n < 0] = 0$, $x_f[n < 0] = 0$, and $x_o[n < 0] = 0$.

- Calculations presented in Fig. 3.1 infer delay blocks. As computations require time to complete, software DSP operations (i.e. in MATLAB) abstract these delays in the form of concurrent lines of code, whereas hardware designs have explicit delay blocks introduced. Thus, delays must be considered to conform to real-time implementations for all platforms.

- The frequencies and resonances retained by the preceding LPF must remain intact. Likewise, frequencies beyond the LPF passband should remain attenuated as much as possible.

- For all incoming values, the algorithm should return acceptable results as fast as possible. That is, streaming audio into the algorithm should return results that do not noticeably lag behind the source. Very small delays are acceptable as musicians and audiences would not likely notice such small lag times.

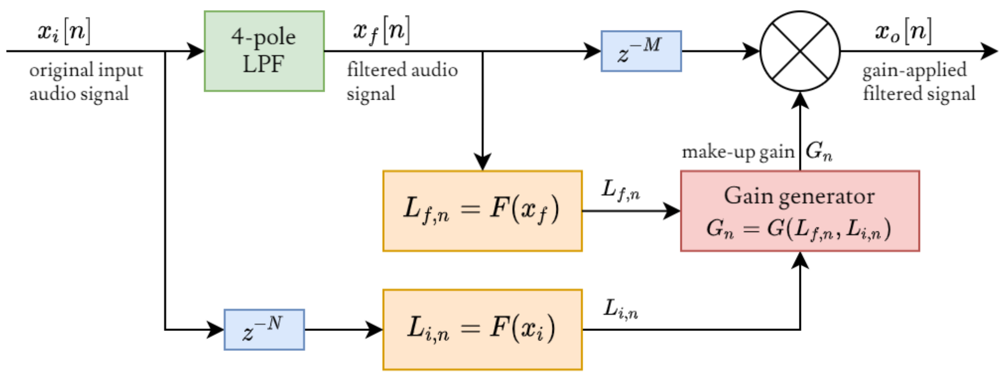


FIGURE 3.1: The baseline automatic gain stage can be visually represented as a feed-forward AGC system not unlike Fig. 2.8b. The make-up gain G_n is generated by considering values from x_f and x_i, such that the loudness of the gain-applied $G_n x_f[n] = x_o[n]$ is equal to that of $x_i[n]$. The delay blocks z^{-M} and z^{-N} represent the total delays in the system and represents how this algorithm would operate in real-time.

3.1.1 Choosing the programming language

MATLAB (version R2020a) [10] is utilised as the main processing platform since it provides many optimised built-in functions that allow for rapid mathematical calculations, which would be required for a real-time processing program. Other programming languages were considered, such as Python 3 due to familiarity with the language, however these languages do not allow for VST audio plugin generation and benchmarking techniques that MATLAB's Audio Toolbox provided [11]. While C++ (via the Steinberg SDK [33]) also allowed for plugin developments, MATLAB provides a platform that effectively abstracts issues unrelated to the algorithms themselves (i.e. memory allocation, architecture deployment, code verbosity) and allows for rapid prototyping of algorithms and techniques used in the project, while C++ would flag these issues during compilation, thereby slowing down the development process.

However, MATLAB is slower than C++ due to being an interpreted language instead of a compiled language, where the latter would be noticeably faster due to optimisations done in compile-time. Despite this, MATLAB's built-in libraries were pre-optimised by MathWorks such that the speed differences in simulations and plugin runtimes are effectively negligible, and its abstracting nature made MATLAB a reasonable choice for this project.

3.2 Signal energy as a loudness metric

As discussed in Section 3.1, the goal of the gain generator is to process the loudness values $L_{f,n}$ and $L_{i,n}$ of the filtered and input sequences respectively, and return the make-up gain G_n that would amplify the loudness of the filtered signal $x_f[n]$ such that $L_{f,n} = L_{i,n}$. An intuitive method would be to divide $L_{i,n}$ by $L_{f,n}$ and derive G_n:

$$G_n = \frac{L_{i,n}}{L_{f,n}} \equiv \frac{F(x_i)}{F(x_f)} \tag{3.2}$$

where $F\left(x_{[i,f]}\right)$ is the mathematical function that generates the loudness values $L_{[i,f]}$. As loudness is a temporal property over a range of time, or window of size N samples considering discrete signals, a rudimentary loudness criterion derived from signal energy can be used. For reiteration from Eqn. 2.2, the signal energy of a signal is defined as:

$$E_x = \sum_{n=0}^{N-1} |x[n]^2| \tag{3.3}$$

where N is the number of samples to consider within a window such that its time duration equivalent is sufficient (i.e. a 400 ms window was suggested by the EBU [21]; given $f_s = 44.1$ kHz that makes $N = 17640$), n is the sample index, and x is the signal to extract the loudness from. By defining the loudness as the energy level of a windowed signal, the make-up gain can be defined as:

$$G_n = \frac{E(x_i)}{E(x_f)} = \frac{\sum_{n=0}^{N-1} x_i[n]^2}{\sum_{n=0}^{N-1} x_f[n]^2} \tag{3.4}$$

3.2.1 Psychoacoustic compensation with Stevens's power law

Signal energy alone is not a sufficient metric to compute loudness due to psychoacoustics being unaccounted for. Stevens's power law uses a variant of signal energy providing a more accurate perception of loudness, as reiterated:

$$\psi(E_x) = k \left(\sum_{n=0}^{N-1} x[n]^2 \right)^{0.67} \tag{3.5}$$

where the exponent of 0.67 was chosen as the continuum concerned is loudness [4]. Thus, a more appropriate loudness-derived gain factor can be obtained in the same form as Eqn. 3.4, and the block diagram of the algorithm is shown in Fig. 3.2:

$$G_n = \frac{\psi(E(x_i))}{\psi(E(x_f))} = \left(\frac{\sum_{n=0}^{N-1} x_i[n]^2}{\sum_{n=0}^{N-1} x_f[n]^2} \right)^{0.67} \tag{3.6}$$

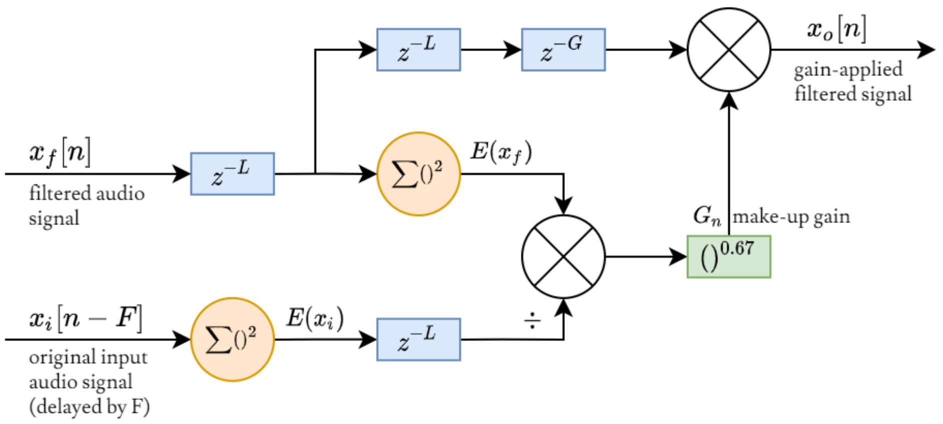
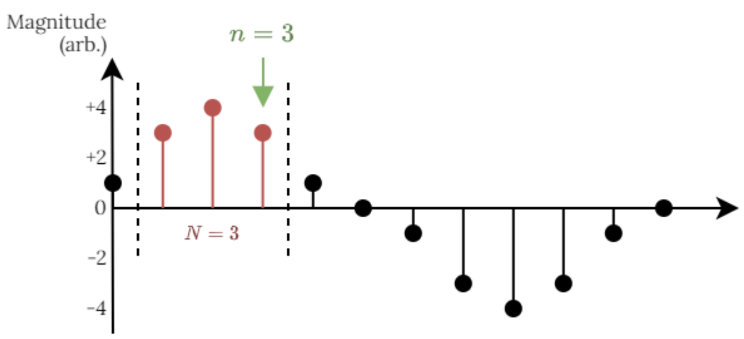


FIGURE 3.2: The make-up gain algorithm implemented with Stevens's power law compares the two signal energies of the filtered and input signals. L delays are inferred at each square summation block, and G total delays resulted from the division and exponentiation functions to obtain G_n. F delays precede the input signal x_i due to such delays being introduced by obtaining x_f as shown in Fig. 3.1.

Stevens's power law is not the only available criterion for loudness, as there are many other measures that provide a more precise set of loudnesses for the same audio signal that will be explored in detail later. It does, however, provide a proof-of-concept for the feedforward AGC structure that can accommodate for more advanced loudness evaluation techniques.

3.2.2 Designing a moving window buffer

Calculations involving summation such as signal energy and loudness via Stevens's power law (among other methods explored later) require a signal window of size N, where N is the discrete duration of the signal to consider. Considering a moving window into the signal, the window would be initialised with a preallocated queue array of zeroes under the assumption that are are no pre-existing signals in the system. This window would move through the signal capturing all the points individually, as older values are pushed further into the array until discarded. The process is demonstrated in Fig. 3.3.



n	window
<0	[0, 0, 0]
0	[0, 0, +1]
1	[0, +1, +3]
2	[+1, +3, +4]
3	[+3, +4, +3]
11	[-3, -1, 0]

FIGURE 3.3: The windowing process captures the signal values one by one and moves them through the memory array, with older values getting gradually discarded. In this example $N = 3$ is the window size, and n is the individual values in the full signal (and thus iterations required to capture the whole signal). The size of the window array is static and pre-defined, and is assumed to only contain zeroes before initialisation (i.e. $n < 0$). Due to the causal assumption, the window only includes current and previous values.

Windowing allows the make-up gain algorithm to observe loudness-varying signals in small timeframes, referred to as *buffers*, with consideration of previous values, allowing for smoother gain transitions in contrast to considering one value without a window. This is also a realistic approach to compute for real-time audio streaming, as new values would be constantly fed to the algorithm without precognition of future values, thus the results would have to be obtained quickly.

The maximum size of the window determines how much physical memory will be required. A larger window size will consume more memory and take longer to process, but returns more smoothly-transitioning values, whereas a smaller window consumes less memory and takes a short time to process, but returns values that may not transition smoothly between each other.

As such, a reasonable window size will be required that transitions values smoothly while not consuming excessive memory. The ITU and EBU's reports recommended that a sliding rectangular window of 400 ms should be used [20, 21]. The pseudocode implementation of the windowing process is shown in Fig. 3.4.

```
# Initialises memory of zeros
memoryBuffer = zeros(length = windowSize)
foreach valueNow in inputSequence do
      # Truncates current sequence
      # Then pushes current value at the end of buffer
      append valueNow to memoryBuffer[2nd to end]

      ... # Processes the sequence
end foreach
```

FIGURE 3.4: Pseudocode implementation of windowing audio sequences.

3.3 Generating gain with RMS values

The mean-square average presented in Eqn. 2.7 provides a more objectively precise loudness compared to other techniques, where its discrete form can be derived [5]:

$$\begin{aligned} l_{eq}[n] &= \frac{1}{N}\sum_{n=0}^{N-1} x_w[n]^2 \\ L_{eq}[n] &= 20\log_{10}\sqrt{l_{eq}[n]} \equiv 10\log_{10}(l_{eq}[n]), \quad dB_w \end{aligned} \tag{3.7}$$

where w denotes the frequency-weighting scheme used, l_{eq} is the intermediary (linear scale) loudness, x_w is the frequency-weighted signal to measure the loudness of, and N is the length of such signal. It can be observed that L_{eq} is a frequency-weighted RMS square-summation in the form presented in Fig. 2.2, and by observing Eqn. 3.7 it was deduced that the algorithm resembles a finite impulse response (FIR) filter as shown in Fig. 3.5.

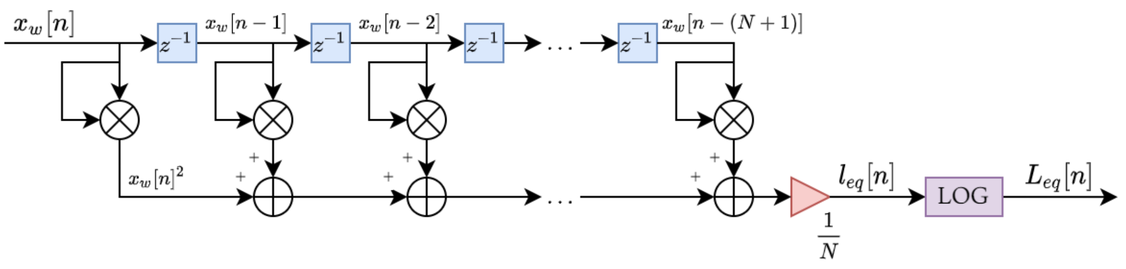


FIGURE 3.5: The RMS algorithm described in Eqn. 3.7 takes the form of an FIR filter with a window size of N.

Since a window of 400 ms was used to capture the signal's values, x_w can be considered as a memory sequence storing the current sample and past 400 ms. A sampling rate of $f_s = 44.1$ kHz results in $N = 0.4(44100) = 17640$ samples. This simple algorithm utilised a considerable amount of multiplication and summation blocks in the mean-square calculation process, and is computationally slow and intensive which is not ideal in real-time applications such as audio processing. Thus, an alternative method was derived to be more suitable for real-time processing.

3.3.1 Implementing the RMS algorithm as a simple moving average filter

Realising the RMS summation algorithm from Eqn. 3.7 as an equivalent recursive simple moving average (SMA) digital filter, as shown in Fig. 3.6, simplified the equation to:

$$\begin{aligned} l_{eq}[n] &= l_{eq}[n-1] + \frac{1}{N}\left(x_w[n]^2 - x_w[n-N]^2\right) \\ L_{eq}[n] &= 10\log_{10}(l_{eq}[n]), \quad dB_w \end{aligned} \tag{3.8}$$

where L_{eq} is the logarithmic loudness (in dB), l_{eq} is the non-logarithmic loudness value, and N is the 'window' size. The pseudocode implementation is shown in Fig. 3.7.

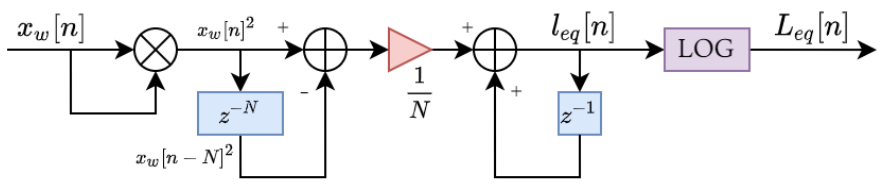


FIGURE 3.6: The discrete SMA L_{eq} filter calculates the RMS loudness of a signal with only two memory elements required (i.e. one for the squared-input signal, and at the linear loudness output), and requires only two multiplier blocks and two addition blocks (with an additional logarithm converter at the output). These changes greatly reduced the complexity and memory requirements of the method in Fig. 3.5 while retaining the same functionalities.

This method operates the same as Eqn. 3.7 and yet reduces the amount of computation required to obtain the same output, only requiring two addition and three multiplier blocks (although an additional multiplier and logarithmic blocks are required to obtain the output in logarithm form), and stores only the previous output $l_{eq}[n-1]$ and the previous input of $x_w[n-(N+1)]$ without the full N-size window in memory. Therefore unlike Eqn. 3.7, utilising a large N would not affect the amount of memory utilised.

However, the implemented filter cannot capture information outside such 'window' due to its rectangular FIR property (i.e. the filter 'cuts off' signals outside the 'window') as shown in Fig. 3.8, and so a sufficiently large N would still be required to capture all the information.

```
delaySize = [value of chosen delay size]
# Obtains loudness with the input memory at [n - delaySize] and the
  previous loudness memory
function getLoudness_SMA(sample, loudPrev, valueDelay):
      # Obtains sum of current and delayed squared sample
      sum = ((valueNow)² - (valueDelay)²)
      loudNow = loudPrev + (sum/delaySize) # Adds previous loudness
      # Sets memory value to delayed sequence at [n - windowSize]
      valueDelay = audioSequence[n - delaySize]
      loudPrev = loudNow # Updates the stored loudness
      # Returns loudness of current and memory values
      return loudNow, loudPrev, valueDelay # Loudness in linear form
```

FIGURE 3.7: Pseudocode implementation of the SMA algorithm.

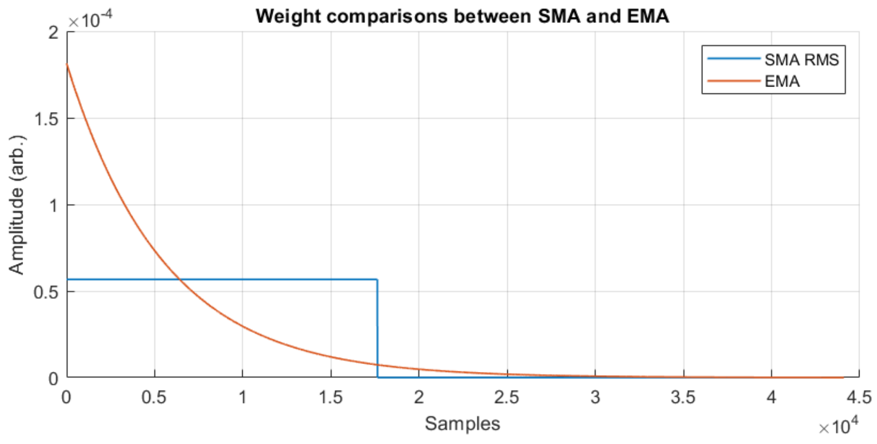


FIGURE 3.8: The impulse response of the SMA and EMA's weighting schemes affect the amount of information observed by the time-weighting algorithms. The SMA algorithm utilises a rectangular window of finite length (i.e. in this plot, of $N = 17640$ samples) that drops all values outside its range, while the EMA algorithm uses a decaying exponential 'window' of infinite length and applies exponentially decreasing weight to older values without dropping the previous values.

3.4 Obtaining loudness with the exponential moving average

A solution to the 'cut-off' windowing issue presented in Eqn. 3.8 was to use a window with an infinite impulse response (IIR), such that the IIR filter does not 'cut off' data points outside window (due to having an infinite window). The exponential moving average (EMA) filter [19] shown in Fig. 3.10 bore similarities to the SMA filter with two key difference: the 'window' exponentially decays and has infinite length as shown in Fig. 3.8, and only one memory element is required to store the previous loudness $l_\tau[n-1]$:

$$\begin{aligned} l_\tau[n] &= \alpha x_w[n]^2 + (1-\alpha) l_\tau[n-1] \\ L_\tau[n] &= 10\log_{10}(l_\tau[n]), \quad dB_w \end{aligned} \tag{3.9}$$

where τ denotes the time constant of the exponential (in seconds). The pseudocode implementation of the algorithm is shown per Fig. 3.9.

```
# Obtain exponential weight coefficient
tConstant = [value of chosen time constant]
coefWeight = 1 - exp(-1 / (fs * tConstant))
# Obtains loudness with only the previously stored loudness
function getLoudness_EMA(sample, loudPrev):
      # Stores current loudness before summation
      loudNow = coefWeight * (valueNow)²
      # Sums current loudness with previous loudness
      loudNow += (1 - coefWeight) * loudPrev
      loudPrev = loudNow # Updates the stored loudness
      # Returns loudness of current and updated previous 'window'
      return loudNow, loudPrev # In linear form
```

FIGURE 3.9: Pseudocode implementation of the EMA algorithm.

The weighting coefficient α is calculated as:

$$\alpha = 1 - e^{-1/(f_s\tau)} \tag{3.10}$$

where f_s is the sampling frequency. A greater τ increases the exponential decay rate and reduces the contribution of new input values to the overall loudness. The filter is stable if $0 < \alpha < 1$, thus $0 < (1 - \alpha) < 1$. Three time constants were often used in these measures [36]:

- Slow: 1 second
- Fast: 0.125 seconds
- Impulse (obsolete): 0.035 seconds (attack[1] phase), 1.5 seconds (decay[2] phase)

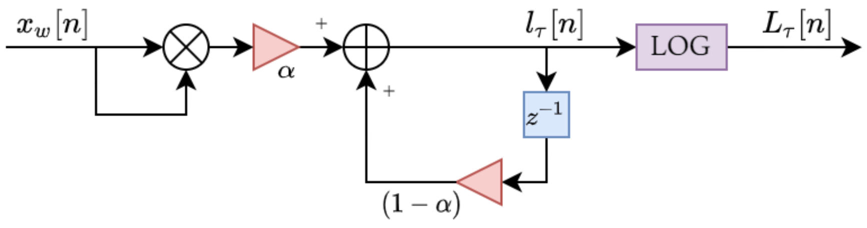


FIGURE 3.10: The EMA L_τ filter calculates the average loudness of a signal with an infinite-length window, where past values have less weight than current values according to Eqn. 3.10. The sole memory element stores the previous loudness calculation, and four arithmetic blocks are required (excluding the logarithm converter at the output).

These recursive moving average filters are often called ’time-weighting’ filters due to weighting previous values over time instead of ’frequency-weighting’ filters that weight the frequency contents of the incoming signal.

[1]Attack and decay refers to how the signal changes with respect to its current value. If a signal is rising, then it is in its attack phase, otherwise it is in its decay phase. This is NOT the same attack or decay as in ADSR envelopes, which is irrelevant to and outside of the project scope.

[2]See footnote 1.

3.5 Generating gain with SMA/EMA filters

The SMA and EMA algorithms can be adapted to Fig. 3.2, and thus the make-up gain can be obtained per Fig. 3.11 and Eqn. 3.11:

$$\begin{aligned} err[n] &= L_i[n] - L_f[n] = 20\log_{10}\sqrt{\frac{l_i[n]}{l_f[n]}} \equiv 10\log_{10}\left(\frac{l_i[n]}{l_f[n]}\right) \\ G[n] &= 10^{\frac{err[n]}{20}} = \sqrt{\frac{l_i[n]}{l_f[n]}} \end{aligned} \tag{3.11}$$

where $err[n]$ is the loudness difference (in logarithmic scale) between the reference signal and the signal to apply the gain to, and $G[n]$ is the correcting gain required (in linear scale) to 'correct' the loudness of the signal at sample n. The $10\log_{10}()$ calculations shown in Eqn. 3.7 provides the gain-generating block $G[n]$ with a large dynamic range (via $err[n]$) to accurately calculate the gains required for each sample. The effects of not using a logarithmic scale to calculate G_n will be explored in a future section.

The same time-weighting filters were used for both the original input x_i and the filtered signal x_f (i.e. applying EMA to x_i requires applying EMA to x_f and not SMA) and not in conjunction.

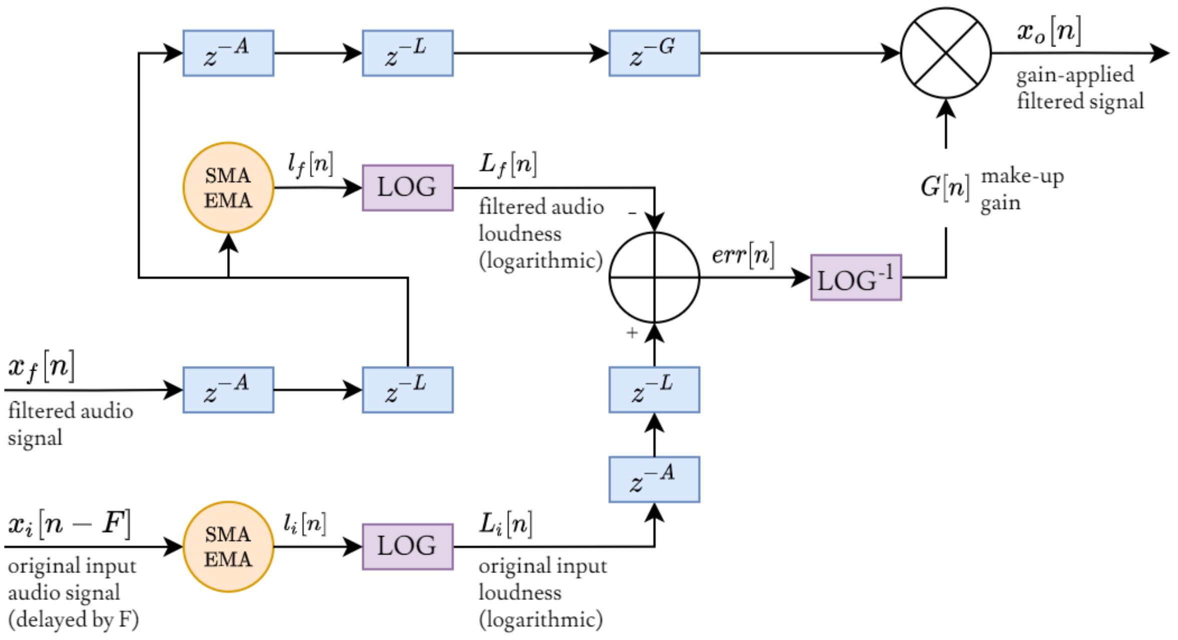


FIGURE 3.11: The moving average filters (i.e. SMA and EMA) can be used to obtain the make-up gain G_n, where l_f and l_i are the linear outputs of the time-weighting filters, and L_f and L_i are their logarithmic equivalents per Eqn. 3.8 and Eqn. 3.9 used for obtaining the error $err[n]$, and thus the make-up gain $G[n]$. Delay blocks z^{-A}, z^{-L}, and z^{-G} refer to the inferred delays due to the time-weighting algorithms, logarithmic conversions, and calculating $err[n]$ and $G[n]$.

3.6 Calculating K-filter coefficients at 44.1 kHz

As discussed in Section 2.2, frequencies in audio waveforms are not heard at equal loudness perceptions. The ITU-R BS.1770 was therefore used to weight frequency contents of incoming audio signals, ensuring that the algorithm would normalise the filtered and reference signals according to human perception of loudness.

The K-filter was defined as a cascade of the RLB filter and the pre-K weighting filter, both which are implemented in the form of a second-order filter per Fig. 2.4 [6]. Ward derived a method to calculate the filter coefficients for any sampling frequencies by utilising the set of equations [19], as reproduced here:

$$\begin{aligned} b_0 &= V_L\Omega^2 + V_B\frac{\Omega}{Q} + V_H & a_0 &= Q^2 + \frac{\Omega}{Q} + 1 \\ b_1 &= 2\left(V_L\Omega^2 - V_H\right) & a_1 &= 2\left(\Omega^2 - 1\right) \\ b_2 &= V_L\Omega^2 - V_B\frac{\Omega}{Q} + V_H & a_2 &= \Omega^2 - \frac{\Omega}{Q} + 1 \end{aligned} \tag{3.12}$$

where $\Omega = tan\left(\pi\frac{f_c}{f_s}\right)$ is the parameter dependent on the specified sampling frequency f_s, and therefore the independent variable in Eqn. 3.12. The calculated coefficients for $f_s = 44.1$ kHz are shown in Table 3.2 using parameters from Table 3.1.

Parameters	Pre-filter (high-shelf)	RLB (high-pass)
Q	0.7071752	0.5003270
V_L	1	0
V_B	1.2587209	0
V_H	1.5848647	1.0049949
f_c (Hz)	1681.9744510	38.1354709
Ω (rad)	0.1105318	0.0027166

TABLE 3.1: For an arbitrary sampling frequency, only the values of Ω needed updating to obtain the parameters defining the cascaded K-weighting filters. Values except Ω copied from [19], $\Omega = tan\left(\pi\frac{f_c}{f_s}\right)$ calculated manually for $f_s = 44.1$ kHz.

Coefficients	Pre-filter (high-shelf)	RLB (high-pass)
a_1	−1.66365510075392	−1.98916967282163
a_2	0.712595415205997	0.989199034978896
b_0	1.53084122270645	0.999560065414779
b_1	−2.65097997332479	−1.99912013082956
b_2	1.16907906507042	0.999560065414779

TABLE 3.2: The coefficients to the feedforward and feedback paths of the RLB weighting and pre-filter curves per the second-order filter shown in Fig. 2.4, considering a sampling frequency of 44.1kHz. Values calculated with Eqn. 3.12.

3.7 Constructing the K-filter

Previous discussions in Section 2.2.5 showed that the K-filter is a cascade of the RLB and pre-K weighting filters, as shown in Fig. 3.12. Essentially, the signal concerned (i.e. the filtered or reference signals) is passed through the pre-K weighting filter, then the output is passed to the RLB filter. The resulting output of the RLB filter is considered the output of the K-filter, where loudnesses measured are K-weighted (and given dB, units are $LKFS$).

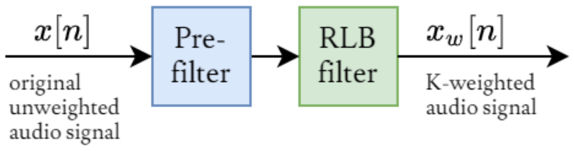


FIGURE 3.12: The K-weighting filter is a cascade of RLB filter and a pre-filtering block. Redrawn from Fig. 2.5 to consider discrete signals.

3.7.1 Implementing the RLB filter

The RLB filter can be implemented using the derived coefficients shown in Table 3.2. The difference equations for the filter shown in Fig. 2.4 is:

$$\begin{aligned} w[n] &= x[n] - a_1 w[n-1] - a_2 w[n-2] \\ y[n] &= b_0 w[n] + b_1 w[n-1] + b_2 w[n-1] \end{aligned} \tag{3.13}$$

where y is the filter output, x is the filter input, and w are the internal delay values of the filter. The system diagram is shown in Fig. 3.14 where Eqn. 3.13 is represented as two cascaded filters. A pseudocode implementation of an arbitrary second-order filter is shown in Fig. 3.13, where the entire signal can be processed altogether or in small buffers (including individual values). The pseudocode processes the feedback path first in order to obtain the intermediary delay values, before computing the output with the forward path.

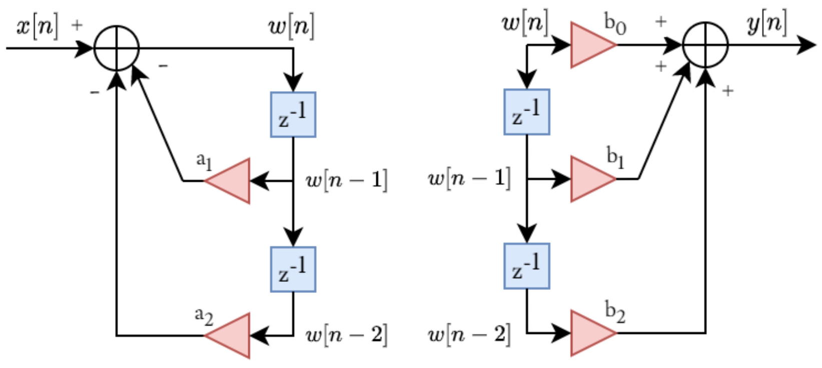
Passing the filtered and reference signals into separate filter function calls, with the RLB filter's coefficients at 44.1 kHz, returned the respective RLB-weighted signals.

3.7.2 Implementing the pre-K weighting filter

The pre-K weighting filter can be implemented in the same way as the RLB. Since the pre-K weighting scheme is in the form of a second-order digital filter, the filter can be realised in the form of Eqn. 3.13. As such, the pre-K high-shelf filter's coefficients shown in Fig. 3.2 can be passed as parameters into the filtering function per Fig. 3.13 and applied to the desired signals.

```
delay = zeros(length = 2) # Initialises two delay elements to store
# Inputs coefficients, signal to filter, and initial delay conditions
function runFilter(coef_b[b0, b1, b2], coef_a[a0, a1, a2], in, delay[-1, -2]):
      out = zeros(length = in) # Stores output of filtering input signal
      # Filters the signal by going through each sample
      foreach sample in in do
            # Calculates the current 'delay' value first
            delay_now = sample - (a1 * delay[-1]) - (a2 * delay[-2])
            # THEN calculates the output value
            out = (b0 * delay_now) + (b1 * delay[-1]) + (b2 * delay[-2])
            # Updates stored delays AFTER calculations
            delay[-2] = delay[-1]
            delay[-1] = delay_now
      end foreach
      return out, delay # Returns the output and final filter delays
```

FIGURE 3.13: Pseudocode implementation of a second-order digital filter. The function should accept and filter either a singular value or an array of samples. The returned delays can be used for the next continuous set of signals.



(A) Feedback path (B) Forward path

FIGURE 3.14: The second-order filtering algorithm can be considered as two cascaded filters, where the feedback path obtains the values required by the forward path.

3.7.3 Applying K-weighting to gain generation

The K-filter blocks were set such that they prepended the loudness calculations for both the filtered and reference signals, as shown in Fig. 3.15 adhering to the ITU-R BS.1770 presented in Fig. 2.6 [6].

3.8 Implementing the nonlinear Moog VCF

Considerable work had been done by Stilson and Smith, Huovilainen, and Daly [7, 8, 9] in deriving the digital variants of the nonlinear Moog VCF. Huovilainen started his implementation by modelling the linear Moog VCF as a recursive digital filter as shown in Fig. 3.16:

$$\begin{aligned} y_i[n] &= y_i[n-1] + g(y_{i-1}[n] - y_i[n-1]) \\ y_0[n] &= x[n] - ky_4[n-1] \end{aligned} \tag{3.14}$$

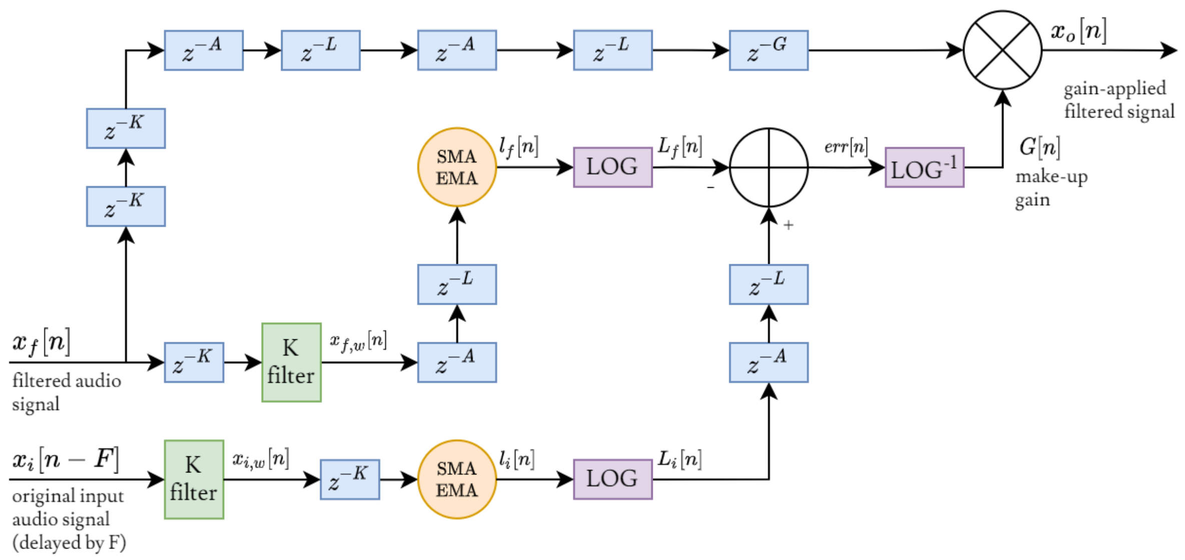


Figure 3.15: The K-weighting filters were placed before loudness calculation blocks from Fig. 3.11. This ensured that the specification of calculating loudness would conform to the ITU-R BS.1770 recommendation [6], where the SMA and EMA algorithms provided the same functionalities as a mean-square measure. Delay blocks z^{-K} were introduced to both filtered and reference inputs as a result of the filter's functionalities.

where x is the input signal, $i = \{1, 2, 3, 4\}$ labels the four ladder stages, y_i are a set of outputs from each stage of the VCF ladder, y_0 is the input into the ladder itself, $k = [0, 4]$ is the feedback gain, and g is the exponential weighting coefficient per Eqn. 2.29. Huovilainen later modified the model such that the fed-back value was the average of two delayed outputs:

$$y_0[n] = x[n] - k\left(\frac{y_4[n-1] + y_4[n-2]}{2}\right) \tag{3.15}$$

while $y_i[n]$ was retained, claiming that this modified feedback delay improves the phase response of the VCF. The block diagram of this modification is shown in Fig. 3.17.

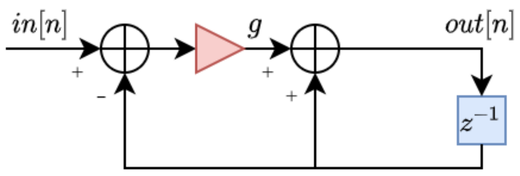


Figure 3.16: A linear digital Moog VCF stage can be modelled as a recursive filter. Four of such stages are cascaded and fed back to the input in the same form as Fig. 2.10.

Huovilainen transformed his linear, delay-modified model into the nonlinear variant by applying *tanh* functions at the inputs of each ladder stage:

$$y_i[n] = y_i[n-1] + g\left(tanh\left(y_{i-1}[n]\right) - tanh\left(y_i[n-1]\right)\right) \tag{3.16}$$

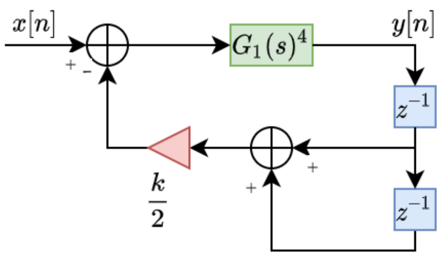


FIGURE 3.17: The modified variant of the feedback model in Fig. 2.10 claimed to improve the phase response of the filter.

while using the delay-modified feedback model per Eqn. 3.15. As discussed in [8], *tanh* blocks were introduced to model the nonlinearities in the analogue components that claim to give the Moog VCF its revered 'warmth' [29].

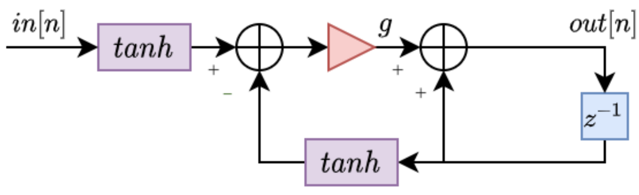


FIGURE 3.18: The nonlinear digital Moog VCF stage is a modified variant of the stage shown in Fig. 3.16, where *tanh* blocks were place at the input and the fed-back output.

The pseudocode to this nonlinear VCF implementation is shown per Fig. 3.19.

```
y[0, 1, 2, 3, 4] # Stores the outputs of each ladder stage
yDelay2 = 0 # Stores previous ladder output
feedback = [amount of feedback from 0 to 4]
coefG = [weighting coefficient]
function moogNonlinear(inSequence):
      out_moogFiltered = [] # Initialises output sequence
      foreach in in inSequence do # Filters the entire sequence
            # Obtains value to feed to ladder
            y(0) = in - (feedback * (y(4) * yDelay2)/2)
            # Goes through each filter stage
            y(1) = y(1) + (coefG * (tanh(y(0)) - tanh(y(1))))
            y(2) = y(2) + (coefG * (tanh(y(1)) - tanh(y(2))))
            y(3) = y(3) + (coefG * (tanh(y(2)) - tanh(y(3))))
            # To store the 2nd-delay feedback, intercept the current y(4)
            yDelay2 = y(4) # Stores 2nd delay output before overriding
            y(4) = y(4) + (coefG * (tanh(y(3)) - tanh(y(4))))
            append y(4) to out_moogFiltered
      end foreach
      return out_moogFiltered
```

FIGURE 3.19: Pseudocode implementation of Huovilainen's nonlinear digital Moog VCF model, with Daly's dimensionless adaptations as discussed in Eqn. 2.32.

3.9 Combining the VCF and make-up gain stages together

With the filter implemented, the full system can be realised as shown in Fig. 3.20. The VCF prepended the make-up gain stage and accepted the original input signal to be filtered. Controlling the cut-off frequency and feedback gain, the filter would output the filtered version of the original input signal. The filtering process would infer a certain amount of F delays, and as such the input signal was buffered that specific amount (this was abstracted on MATLAB and many high-level programming languages).

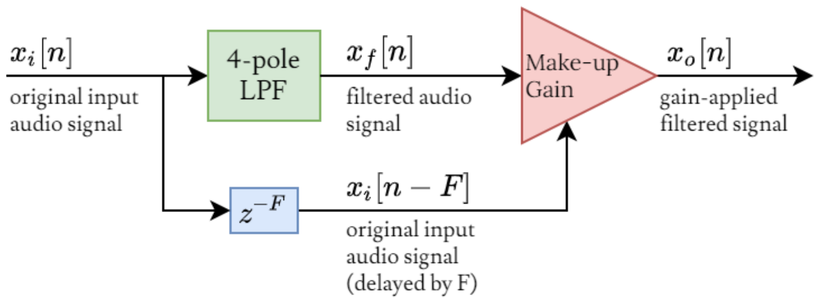

$x_i[n]$
original input audio signal
4-pole LPF
$x_f[n]$
filtered audio signal
Make-up Gain
$x_o[n]$
gain-applied filtered signal
z^{-F}
$x_i[n-F]$
original input audio signal (delayed by F)

FIGURE 3.20: The full system combining the feedforward AGC with the Moog VCF model. The delay block z^{-F} buffers the original input signal so that the gain calculated by the make-up gain stage is synchronised with the VCF output.

The make-up gain follows the structure from Fig. 3.15 and took the inputs of the original (delayed) and the filtered signals, outputting the gain-applied filtered signal, thus restoring the loudness of the filtered signal to closely match that of the original input. As shown in the diagram, the K-weighting, logarithm-antilog operations, and SMA/EMA filters all inferred delays upon the filtered output as the samples and gains must be synchronised (once again, this is abstracted in MATLAB).

3.10 Quantitative metrics to evaluate make-up gain

A number of metric could be devised to help quantitatively evaluate the performance of the make-up gain stage.

- Make-up gain calculations of a signal must not take longer than the time duration of the signal itself, ensuring that calculations are fast and reliable.
- The waveform envelope of the output signal must have similar shape to the original signal.
- Loudness envelopes between the original input and gain-restored output signals are similar.

Comparing the impulse responses between the output of the VCF and the make-up gain output shows the amount of distortion that the gain stage would impose on a signal, and allows us to observe how much of the pass-band frequencies are preserved per the requirements stated earlier.

As the make-up gain algorithm intends to restore signal loudness in a waveform over time, Waveform envelopes between the input and output signals should also be compared to compute the amount of loudness restored by the algorithm. Applying an EMA with a long-duration time constant (i.e. 1 second [19]) extracts the envelope of the signal concerned. Subtracting the envelope of the final output with that of the initial input obtains the amount of loudness that the make-up gain stage was able to restore.

Additionally, many high-level programming languages and development platforms offer methods of estimating time taken to run codes. Essentially, a timer is started before running the make-up gain calculations, then stopped once the calculations are completed. The difference between stop and start times can be considered the *elapsed time* of the algorithm. The value of this elapsed will depend on the duration of the signals concerned.

3.10.1 Test data for benchmarking the system

A number of audio data sets can be used to test the functionalities of the make-up gain and Moog VCF, including the following:

- Impulse (full signal of 1 second) to obtain the impulse response and observe its frequency response.
- Square wave of 10 seconds at 220 Hz to test the Moog VCF's low-pass characteristics and restoration of the filtered signal's equivalent loudness.
- Eric Whitacre's musical composition "Water Night" [37] to test the system's practical applications in low-pass noise removal and media streaming.

These test data are to be generated/sampled at $f_s = 44.1$ kHz.

3.11 VST plugin implementation

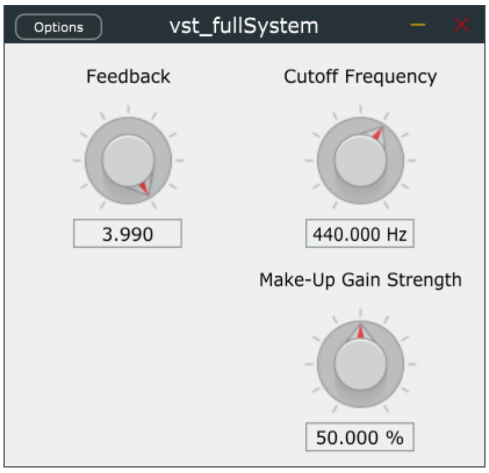
The developed make-up gain and VCF systems can be used in a DAW program by converting the system into a VST audio plugin using MATLAB's Audio Toolbox [11].

To minimise memory consumption, the EMA method was used for real-time processing instead of the SMA, as the latter still required storing previous N values within the window as shown in Fig. 3.6, while the EMA only requires storing the previously obtained loudness per Eqn. 3.9.

Since the make-up gain stage has no tuning parameters involved, the only controls were the cut-off frequency and feedback gain of the VCF stage. To control the strength of the make-up gain applied on the filtered signal, a new 'gain strength' factor can be applied. Ranging from 0 to 1 (equivalently $0 - 100\%$), this factor scales how much of the calculated make-up gain will be applied to the filtered signal:

$$x_o[n] = (1 - \lambda)x_f[n] + \lambda(G[n] \times x_f[n]) \tag{3.17}$$

where λ is the 'gain strength' factor, G is the make-up gain, x_f is the filtered signal, and x_o is the final system output. At $\lambda = 0$ (i.e. at 0%) the system outputs does not apply the make-up gain and functions only as the VCF, while $\lambda = 1$ (i.e. at 1000%) results in the full gain being applied. Any values of λ where $0 < \lambda < 1$ scales the make-up gain accordingly. The proposed layout of the plugin is shown in Fig. 3.21.



Feedback: 3.990
Cutoff Frequency: 440.000 Hz
Make-Up Gain Strength: 50.000 %

FIGURE 3.21: Layout of the audio plugin implementation of the entire system, including the VCF and make-up gain stages. The tunable parameters are the VCF cut-off frequency and feedback gain, and a factor scaling the make-up gain.

4 Results and discussions

4.1 VCF impulse response

Analysis of the impulse response of Huovilainen's Moog VCF implementation [8] had been discussed extensively in Daly's thesis, with the nonlinear implementation referred to as the 'unit-and-a-half delay' model by Daly [9]. The frequency responses obtained in this iteration of the nonlinear implementation will be briefly explored as such is the same model implemented by Daly and Huovilainen before. An example impulse response of the VCF, given $f_c = 440$ Hz and the original impulse of magnitude unity, is shown in Fig. 4.1.

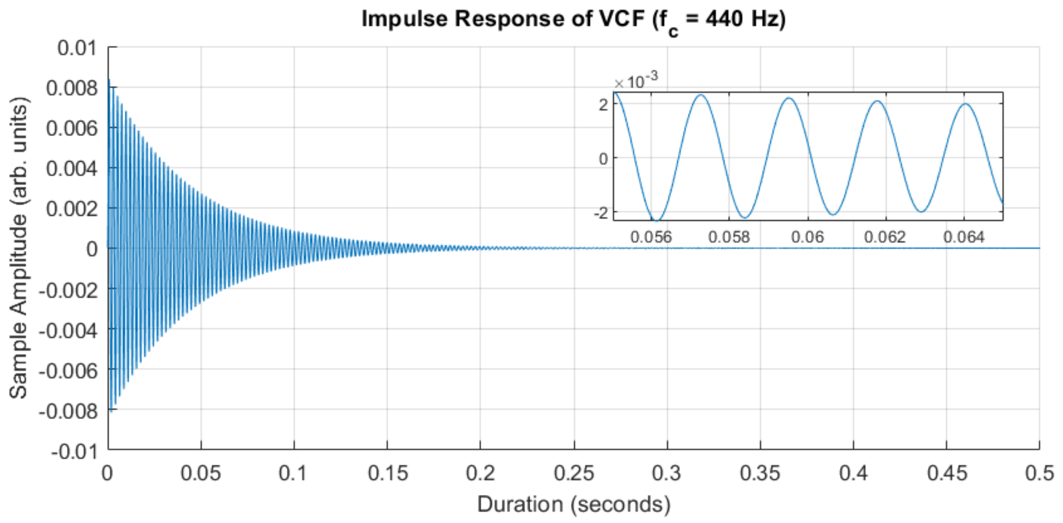


FIGURE 4.1: The impulse response of the Moog VCF.

4.1.1 VCF frequency response

The interesting characteristics of the analogue Moog VCF come from its nonlinear phase distortions, and therefore analysis of the VCF's frequency response is crucial to comparing how the implemented digital model holds up to its analogue counterpart.

The frequency response of the VCF, given varying f_c and constant feedback gain $k = 3.99$, is shown in Fig. 4.2. The magnitude response profile closely resemble the analogue equivalent model until $f_c \geq 10$ kHz where the resonant peaks become mismatched from f_c. In conjunction, the phase responses for $f_c < 10$ kHz bore similarities to their analogue counterparts (albeit with increasing divergents for increasing f_c), after which the phase response became wholly inaccurate.

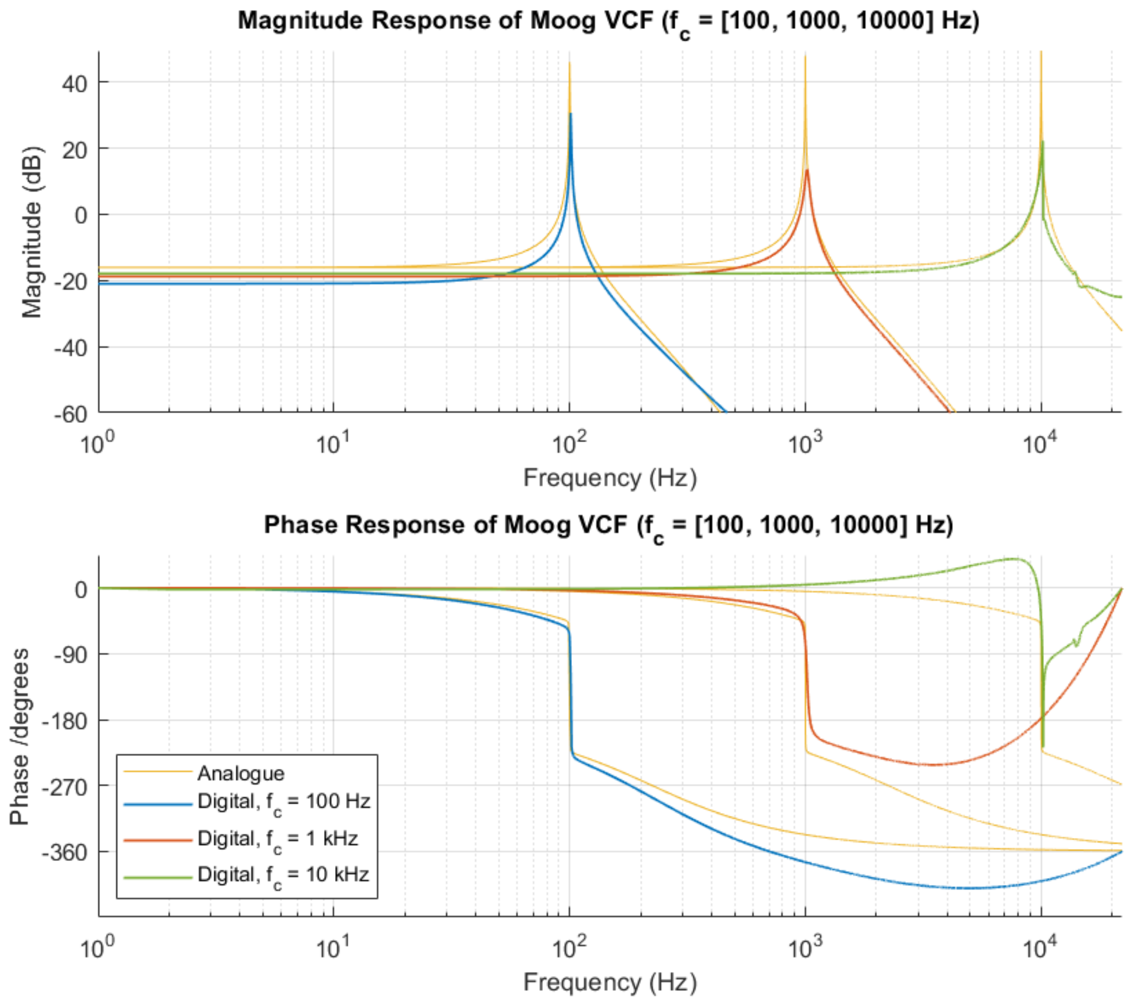


FIGURE 4.2: A comparison between the implemented nonlinear model of the Moog VCF and the analogue frequency responses as discussed in [7]. The magnitude and phase responses of the nonlinear model are an improvement over the single-delay model as described in Eqn. 3.14, however magnitude and phase responses become increasingly distorted at higher frequencies. Unlike the linear model, the phase response at 180° is not firmly flat but instead curved, signifying the nonlinearities in the system.

4.2 Impulse response of the make-up gain stage

The impulse response of the make-up gain stage was obtained to evaluate whether the gain stage would, per the stated design requirements, preserve the frequency contents of the passband, while mitigating any side-effects of frequencies outside the passband.

To do this, the impulse response of the prepending VCF at $f_c = 440$ Hz, as shown in Fig. 4.1, was passed as the input to the make-up gain stage, while the original impulse was delayed and passed as the reference signal. The resulting impulse responses for the gain stage, using both the SMA (of 400 ms window) and EMA (with $\tau = 125$ ms) algorithms, are shown in Fig. 4.3.

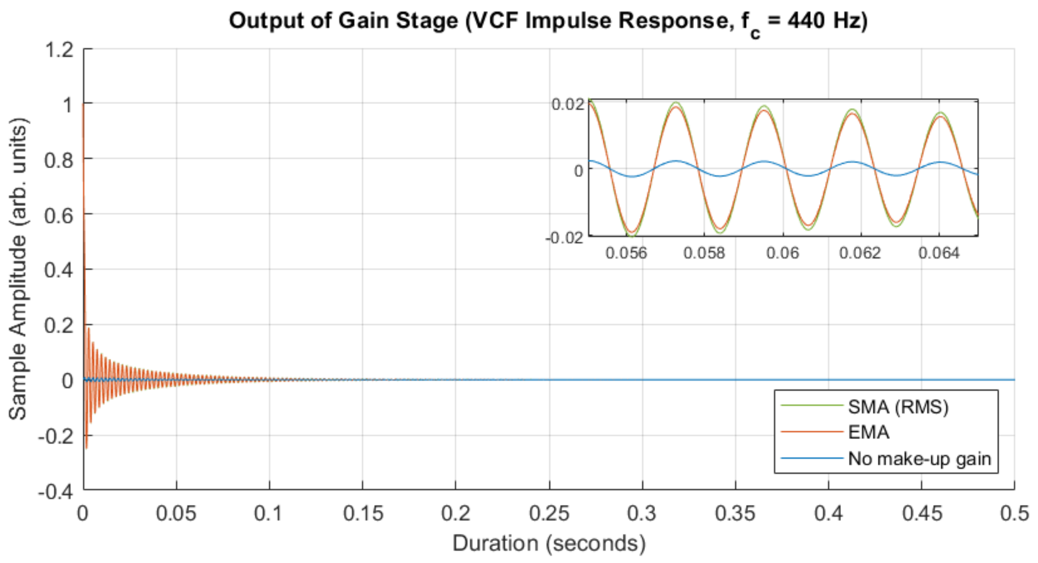


FIGURE 4.3: Outputs of the gain stage using both SMA (equivalent to RMS) and EMA algorithms were similar to each other. Both algorithms attempted to amplify and restore the volume of the filtered signal such that their loudnesses are as close to the original impulse (of magnitude unity) as possible.

It was determined that the gain stage had not only restored the waveform envelope to resemble that of the original impulse, due to restoring the loudness, but also ensured that no rapid phase jumps occured in the output waveform.

4.2.1 Frequency response of make-up gain output

To verify whether the make-up gain successfully retained the passband characteristics while minimising the stopband's contributions, the frequency response of the gain-applied output can be examined as shown in Fig. 4.4.

Magnitude response

Frequencies of the gain-applied output within the VCF passband (i.e. $f \leq f_c$), including the resonance, remained intact, and the resonant frequency of the gain-applied output was the same as that of the filtered signal input. The only distinct differences observed between the filtered signal and the gain-applied output are:

- The magnitude of the passband frequencies where $f \leq f_c$ were amplified to levels greater than the original gain-applied signal.

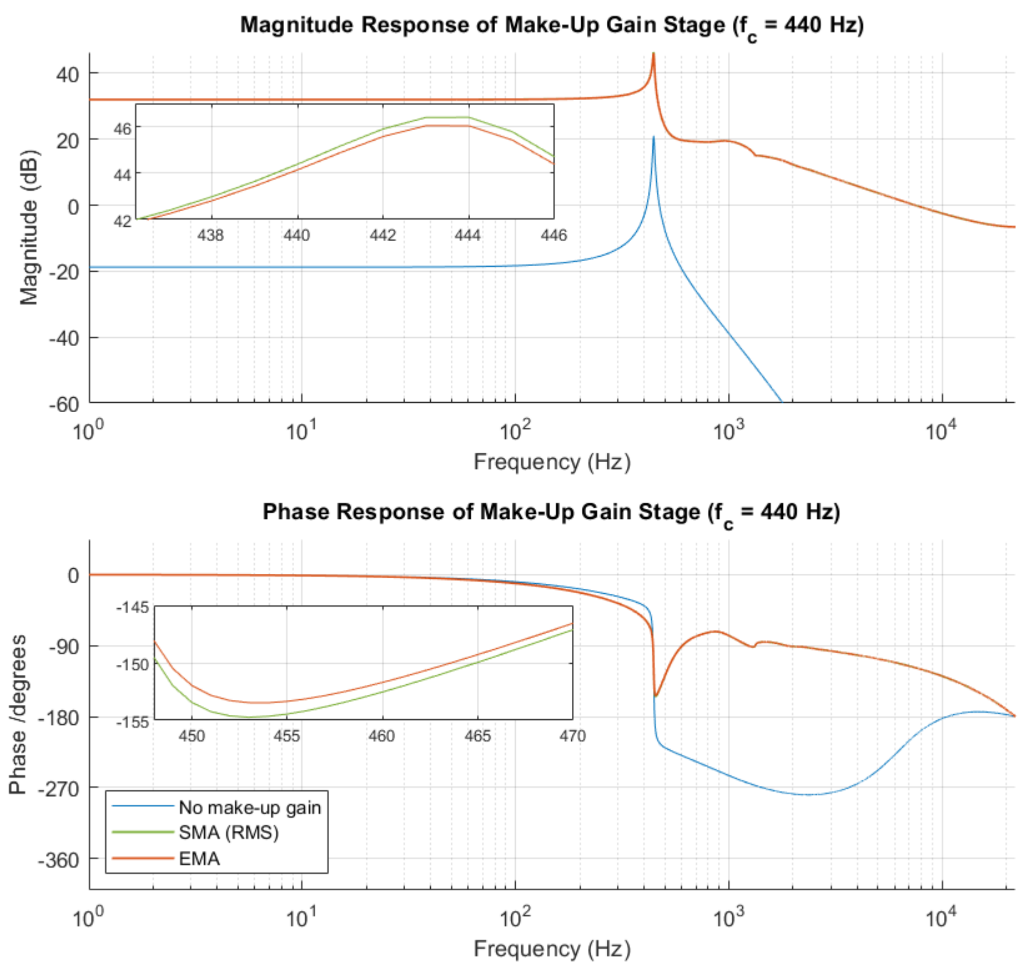


FIGURE 4.4: Outputs of the gain stage using both SMA (equivalent to RMS) and EMA algorithms show similar frequency contents both in magnitude and phase. Both the filtered input signal and the gain-applied output signal displayed resonance at 443 Hz (note that this discrepancy between resonance and f_c was due to the digital VCF model, and thus the resonance was carried over to the make-up gain stage). The upper subplot shows the resonance of the SMA and EMA schemes, whereas the lower subplot shows the inflection points of the SMA and EMA phase responses.

- The ratio between magnitudes of bandpass frequencies and the resonant peak were decreased, i.e. the slope of the passband magnitude rising to the peak resonance in the was less steep than that of the filtered signal.

However, frequencies outside the passband (i.e. $f > f_c$) showed greater distinctions. The frequency slope beyond the output passband were not only amplified but also slightly distorted, and the dB/decade slope was also reduced.

Both the SMA (equivalent to RMS) and EMA methods exhibited near-identical responses with these distortive features, as shown in the example frequency response from Fig. 4.4.

Phase response

The phase response of the gain-applied output exhibited arguably more distortions than its magnitude response as shown in Fig. 4.4. For $f \leq f_c$ the phase response remained roughly the same, although at higher frequencies the rate of phase response showed higher degrees/decade than that of the filtered input signal.

At $f > f_c$ the phase response became wholly deviant as the phase no longer remained decreasing. Instead the phase response buckled up and incremented with higher frequencies, all without falling below $-180°$. The phase would eventually converge back to $0°$ in the same way as the filtered input signal at $f = \frac{f_c}{2}$.

4.2.2 Equivalent continuous sound levels of gain-stage outputs

Signal loudness can be quantified into the equivalent continous sound level L_{eq}, or L_τ given time weighting scheme used. The input impulse, VCF impulse response, and make-up gain-applied signals were passed through the EMA filter to obtain their L_τ envelopes as described in Eqn. 3.9.

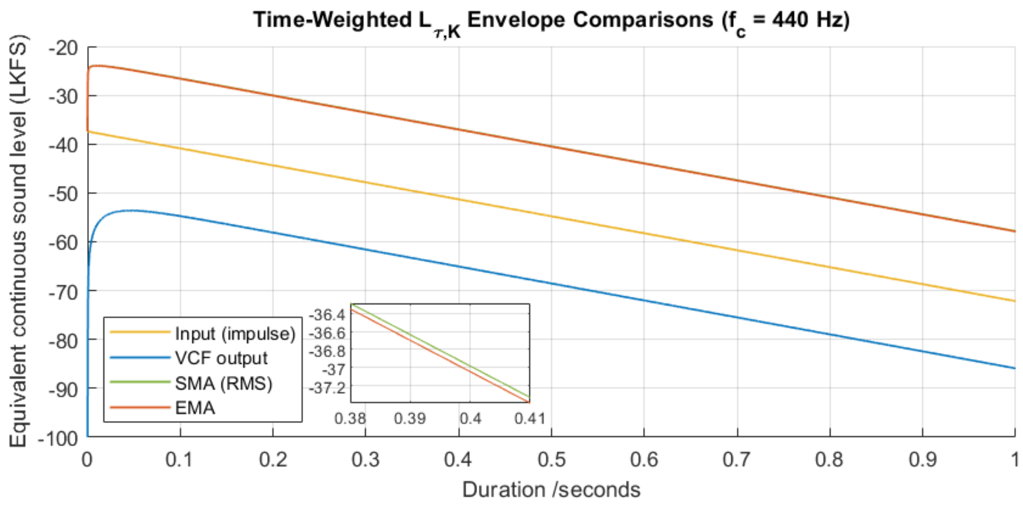


FIGURE 4.5: The SMA (equivalent to RMS) and EMA algorithms provided similar make-up gains to the VCF output with very small discrepancies (as shown in the smaller plot). Given an impulse input and the VCF response, the algorithms over-applied the necessary make-up gains and resulted in over-amplification over time.

The resulting gain-applied loudnesses shown in Fig. 4.5 showed that both the SMA and EMA schemes applied similar make-up gains to the VCF output (the differences are shown in the smaller plot). However, the applied gain resulted in over-amplification in the output loudness and resulted in a a percentage error (i.e. the loudness of the output relative to input, in linear scale) of 405.495%. It was inferred that the make-up gain stage had difficulties predicting the long-term loudness of a brief non-zero signal (i.e. an impulse trailed by silence), although the instantaneous loudness at the impulse (i.e. when the impulse was applied) was restored as shown in Fig. 4.6, and eventually decayed at the same logarithmic rate.

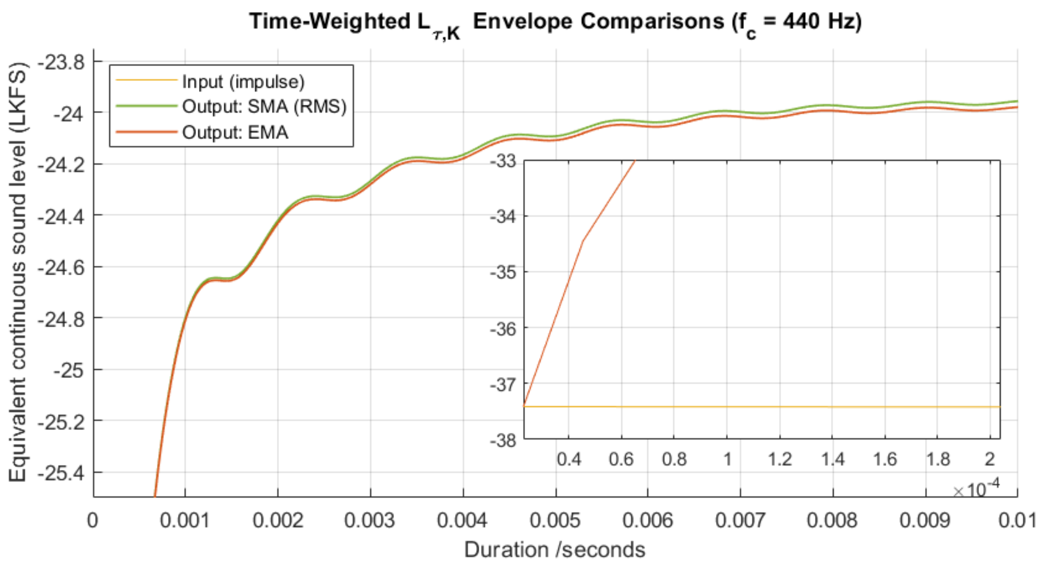


FIGURE 4.6: The make-up gain outputs of the SMA and EMA schemes directly inherited the loudness of the VCF output, shown in Fig. 4.7, and slowly diverged from each other over time. The sub-plot shows that, at the time of impulse, the make-up gain did successfully restore the signal loudness to the instantaneous loudness of the reference (original input) impulse signal.

Fig. 4.7 showed that the SMA and EMA gains' overshoots were due to the VCF output's loudness envelope not being instantaneously loud, peaking at 46.3 ms; this was due to the output of the VCF where its impulse response waveform did not instantaneously peak but at 46.3 ms while the make-up gain estimations were not able to account for future times (due to the real-time causal requirements set in Section 3.1). As such, the gain stage depended on the output signal of any preceding blocks to reliably restore the signal loudness to that of the reference signal.

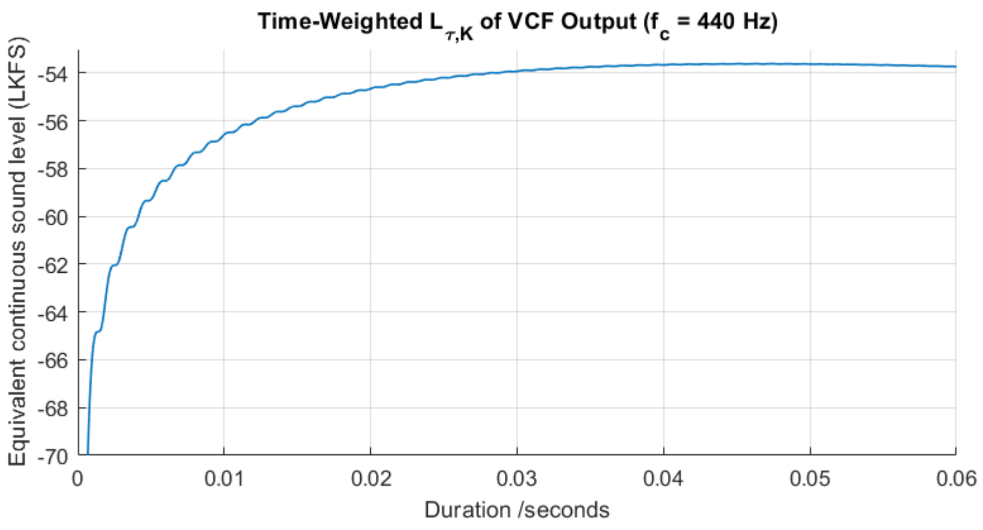


FIGURE 4.7: The VCF output resulted in a loudness envelope ripple causing ripples in make-up gain calculations as shown in Fig. 4.5 and Fig. 4.6.

4.3 Elapsed time

One of the discussed metrics for evaluating the make-up gain stage was to ensure that the algorithm does not take a longer time to process the signal than the signal duration itself. To calculate the time required for the SMA and EMA algorithms to compute the loudness of a 1-second signal (i.e. the previously discussed impulse and VCF output signals), a Monte Carlo simulation of 1000 runs was used.

It was found that the SMA algorithm took 7.535 seconds on average to process an impulse signal of 1 s, whereas the EMA algorithm took only 951.8 ms. It was therefore evaluated that the EMA is faster in processing the make-up gain compared to the SMA.

4.4 Testing system with other audio data

While brief non-zero signals resulted in some gain over-predictions as shown with impulse's results, the make-up gain stage had less problems, both in terms of accuracy and speed, in obtaining the make-up gain to match continuously-loud signals.

For brevity, only the EMA output to the make-up gain stage will be shown/discussed (unless stated otherwise). As the SMA output has very similar values to the EMA (assuming sufficient window length), per previous discussions, and its values can therefore be the assumed to be the same as that of the EMA.

4.4.1 Square wave of unit amplitude

Inputting a square wave of frequency $f = 220$ Hz and amplitude of unity into the VCF-gain system returned the outputs shown in Fig. 4.8. The peak amplitudes of the gain-stage output were greater than that of the square wave (at 1.68 arb. units), with an overshoot of 1.99 arb. units at the start.

However, as shown in the subplot of Fig. 4.9, the make-up gain successfully restored the loudness of the VCF-filtered square wave with a discrepancy of 0.4 LKFS due to over-estimation of loudness by the make-up gain module. While the same issue persisted from the impulse respose loudness discussed in Fig. 4.5, the loudness discrepancy of such a lengthy signal (and not a brief impulse) allowed the make-up gain to consider the entirety of both the reference input and VCF output signals.

On average (using a Monte Carlo simulation of 1000 runs), the EMA make-up gain algorithm

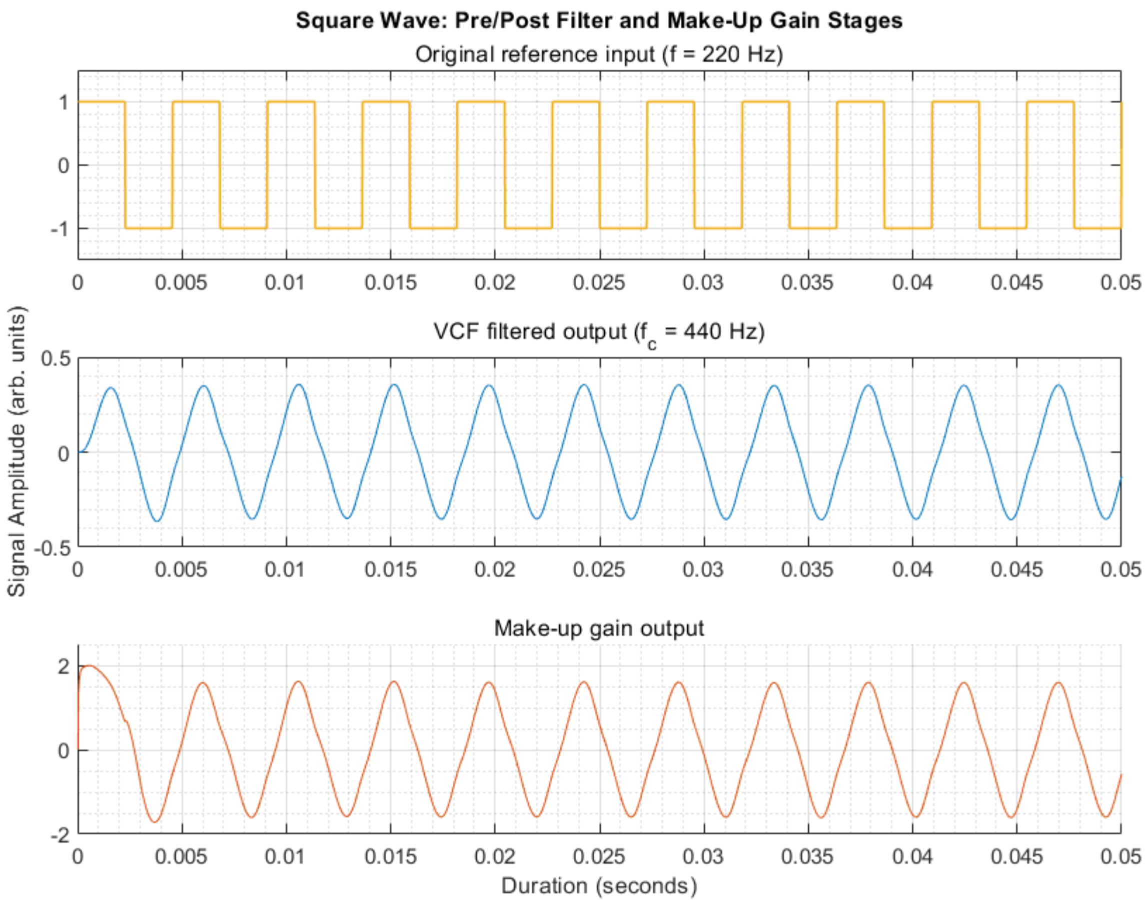


FIGURE 4.8: A reference square wave of 220 Hz (yellow) was passed to the VCF at $f_c = 440$ Hz. Its results (blue) were then passed to the gain stage to produce the amplified output (orange).

took 0.9295 seconds, while the SMA algorithm took 11.543 seconds to process 1 second of the VCF output (with a square wave input). It was therefore shown again that the EMA has superior speed compared to the SMA gain scheme.

4.4.2 Whitacre's "Water Night" composition

To simulate a realistic application of the system, the choral composition "Water Night" by composer Eric Whitacre [37] was used to test real-time functionalities of the VCF and the make-up gain stages due to the following:

- The piece was composed to employ rich (i.e. many) overtones and harmonics. This allowed the VCF to filter as many overtones as required by the user (in this test, at $f_c = 440$ Hz, or at A4), and provided the opportunity to observe the gain stage's performance in handling wide ranges of dynamics[1] (i.e. musical loudnesses).

[1]Composers refer to the term 'dynamics' as the variations of loudness that the musician was to play the musical notes or passages at. It is not immediately quantifiable, and is ultimately dependent on the interpretation of the performer and the musical context.

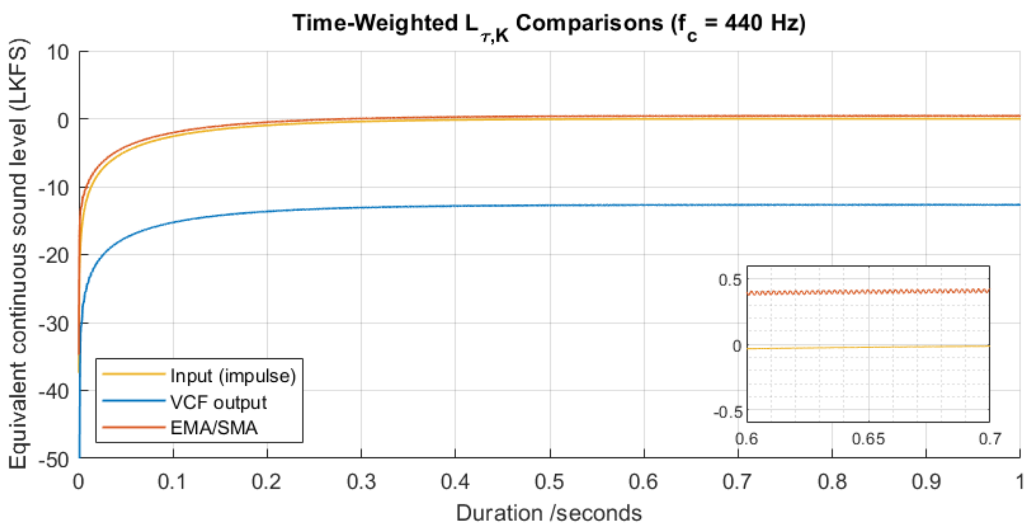


FIGURE 4.9: The EMA loudness $L_{\tau,K}$ calculations show very minor discrepancies between reference input and gain-applied output, in contrast to Fig. 4.5. The make-up gain over-restored 0.4 LKFS of loudness to the VCF output.

- The duration of the piece is 5:47 minutes, and is a standard-length piece of music in streaming services. This allowed us to measure the time taken to process the full audio, thereby evaluating the make-up gain's algorithms in real-time and simulate a streaming platform.

The output of this real-time processing can be observed in Fig. 4.10, while its loudness plots can be found in Fig. 4.9. As shown, the make-up gain has little issues calculating the necessary make-up gain to match the VCF-filtered signal loudness to the original recording's. This confirmed the postulate, discussed regarding the square wave, that the algorithm has issues with calculating gain of very brief pulses, but has little difficulty processing long-running loudnesses.

It was also found that the algorithm took 302.7748 seconds to process the musical piece of 327.4533 seconds, showing that the algorithm runtime is 7.54% faster than the duration of the piece. This infers that the algorithm, provided the streaming samples were buffered, could operate in real-time processing.

4.5 Optimising the make-up gain system

4.5.1 Obtaining make-up gain in linear scale

The error/gain calculation of the algorithm from Fig. 3.11 can be mathematically simplified, shown in Fig. 4.12, by directly dividing the reference loudness $l_i[n]$ with the VCF-filtered $l_f[n]$

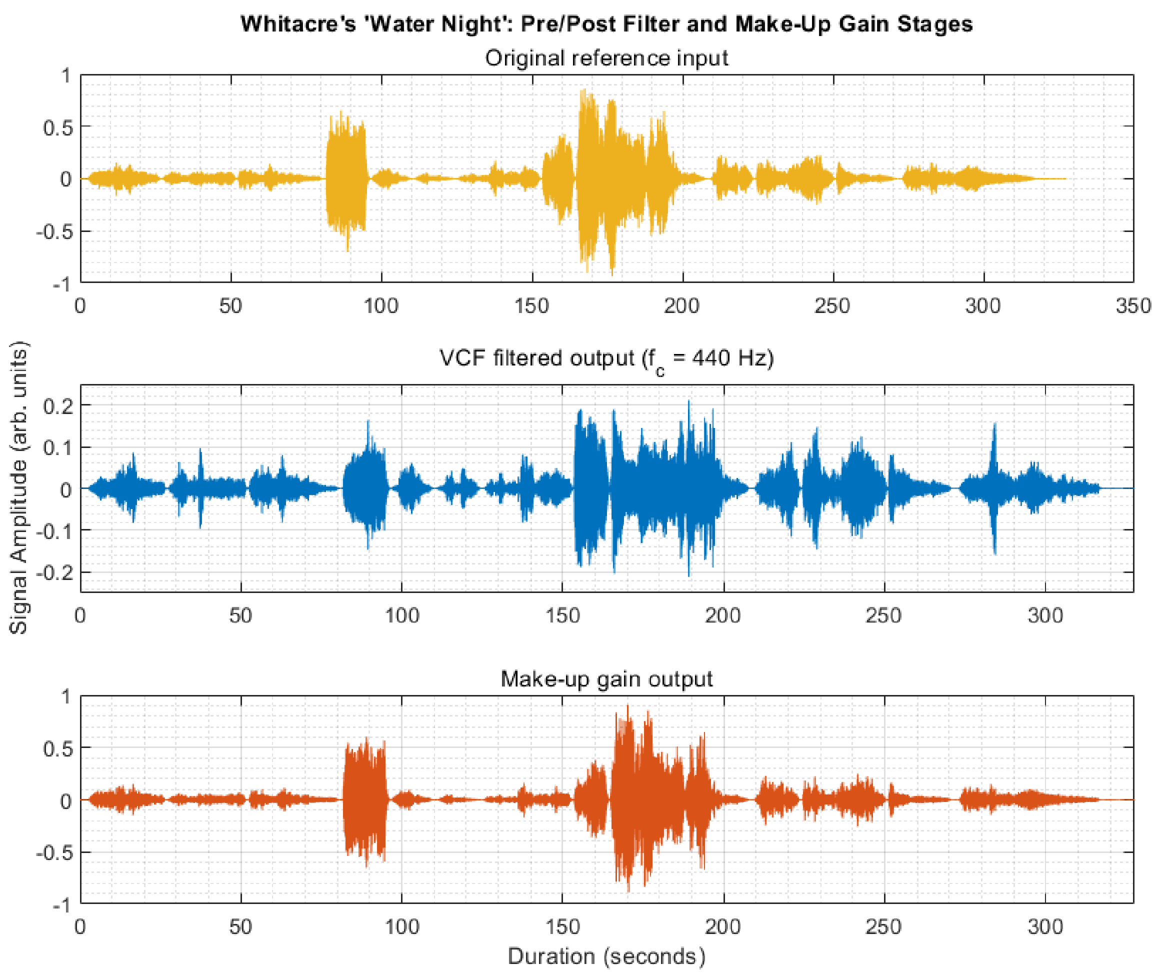


FIGURE 4.10: The original recording (yellow) was streamed (i.e. passed sample by sample) to the VCF at $f_c = 440$ Hz, where its immediate result (blue) was passed to the gain stage to produce the amplified output (orange), simulating the real-time filtering and make-up gain pipeline.

to obtain the make-up gain $G[n]$:

$$G[n] = \frac{l_i[n]}{l_f[n]} \tag{4.1}$$

The requirements of using logarithm-antilog converter blocks (and therefore their associated delays) became obsolete as a result, however this traded for a lower dynamic range of floating point values due to limited floating point storage in software implementation [19]. In some circumstances, the lack of precision resulted in catastrophic gain calculations, be it over-predicting or under-predicting, as shown when passing the same square wave from Fig. 4.8 resulted in erroneous gain calculations shown in Fig. 4.13.

4.5.2 Optimising the algorithm within logarithmic scale

An alternative optimisation technique involved exploiting the fact that the loudnesses and gains were to be converted between linear and logarithmic scales. Considering RMS calculations,

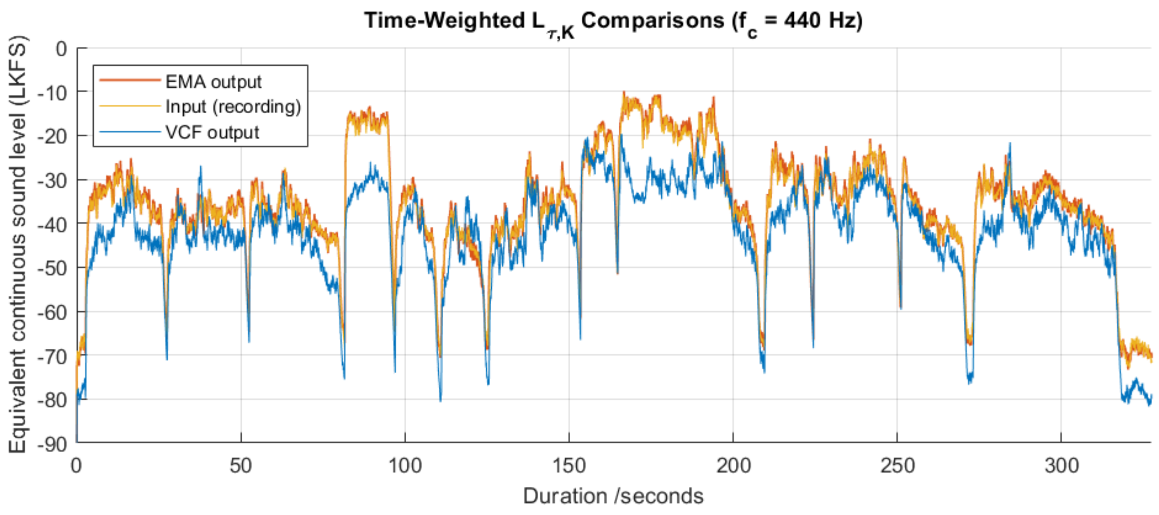


FIGURE 4.11: The real-time EMA loudness $L_{\tau,K}$ calculations show closely matching dynamics with very minor discrepancies between reference input (i.e. the original recording) and the gain-applied output.

conversions from Eqn. 3.11 was simplified to:

$$\begin{aligned} err[n] &= L_i[n] - L_f[n] = 2\log_{10}\sqrt{\frac{l_i[n]}{l_f[n]}} \equiv \log_{10}\left(\frac{l_i[n]}{l_f[n]}\right) \\ G[n] &= 10^{\frac{err[n]}{2}} \end{aligned} \tag{4.2}$$

where the factor 10 was omitted from the logarithmic conversions. This removed two multiplication processes from the gain-obtaining algorithm while adhering to the structure presented in Fig. 3.11. Processing an impulse (of the same variant shown in Fig. 4.3) showed that the improved model sped up the make-up gain stage by 27.76%, taking only 0.7215 seconds (on average with a Monte Carlo simulation of 1000 runs) to process a 1-second impulse. Similarly, a 1-second square wave (from Fig. 4.8) took 0.7991 seconds, yielding a 20.09% speed-up.

4.6 Implementing the system as an audio plugin

Using MATLAB's Audio Toolbox [11] the VST plugin was generated. The layout of the plugin is shown in Fig. 3.21. Such audio plugin was tested on multiple DAWs including REAPER and Audacity, and successfully filtered incoming signals while restoring their loudnesses, confirming the real-time functionalities of the system.

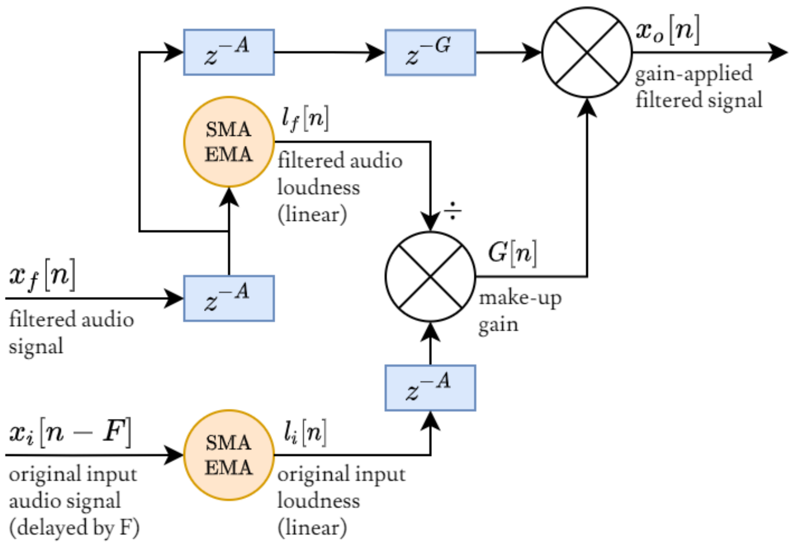

z^{-A}
z^{-G}
$x_o[n]$
gain-applied filtered signal
SMA EMA
$l_f[n]$
filtered audio loudness (linear)
$\div$
$G[n]$
make-up gain
$x_f[n]$
filtered audio signal
z^{-A}
z^{-A}
$x_i[n-F]$
original input audio signal (delayed by F)
SMA EMA
$l_i[n]$
original input loudness (linear)

FIGURE 4.12: As subtraction in logarithmic scale equates to division in linear scale, obtaining $G[n]$ was simplified by directly dividing $l_i[n]$ by $l_f[n]$, similar to obtaining the gain with Stevens's power law from Fig. 3.2. The amount of delays inferred would be reduced, however the dynamic range of the loudness obtained would be reduced.

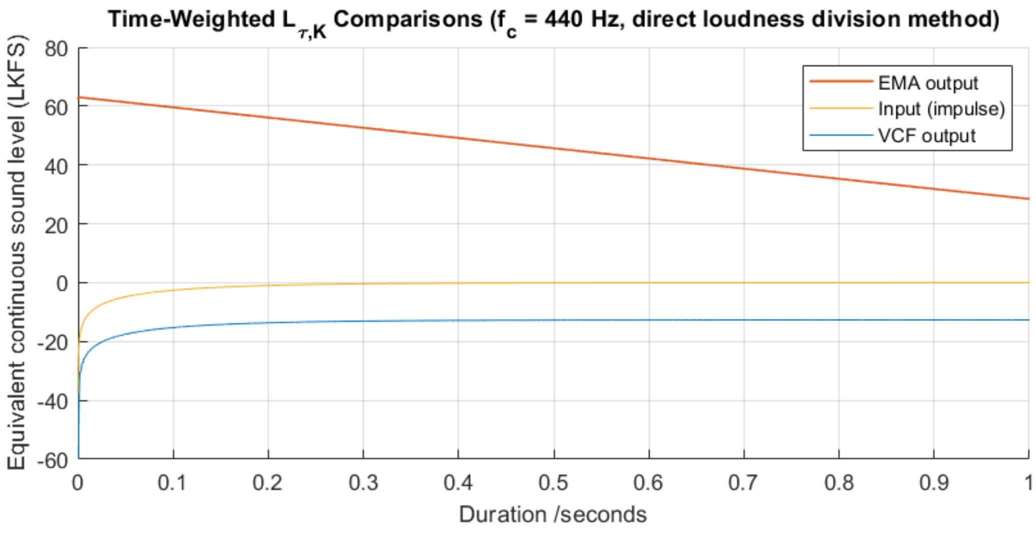


FIGURE 4.13: Obtaining the make-up gain using the method provided in Eqn. 4.1 resulted in numerical issues that, while the equations are mathematically correct, do not account for precision loss in numbers, and therefore led to massive loudness over-amplifications.

5 | Conclusions

5.1 Conclusions drawn from results

A novel method of amplifying the loudness of a filter's output, in this case the Moog nonlinear digital low-pass VCF as derived by Huovilainen [8], has been successfully implemented and shown to operate in real-time processing. By using the SMA and EMA filters to obtain the loudness at the current sample in the streaming audio, a reliable make-up gain was successfuly obtained.

Implementing the RMS scheme as a recursive SMA filter greatly reduced the memory required to process a signal. Same benefits were found with the EMA scheme which required even less memory. Both schemes were superior to the original RMS schemes in terms of speed and memory usage, although ultimately the EMA scheme triumpthed due to its lower processing time. The entirety of the system was successfully implemented as a VST audio plugin.

The optimisation scheme presented in Eqn. 4.2 improved the processing speed by about 20% in comparison to the original scheme in Eqn. 3.11, and was proven to successfully process incoming audio in an audio-streaming simulation.

By implementing the K-weighting filters to process incoming audio in a streaming-esque manner (i.e. sample by sample), the filtering speed was greatly improved as no longer was it required to process individual signal buffers (which would reduce the accuracy of the K-weighting over the entire signal), but instead only requiring the current signal sample and previous delays within the filter.

However, the make-up gain algorithm has issues with obtaining loudnesses of brief pulse signals (i.e. an impulse), where its loudness perceptions by humans would differ greatly from that obtained by the program. The algorithm itself is also not fully optimised as a number of delay blocks persisted. The proposed loudness-division optimising scheme presented in Eqn. 4.1 has issues with numerical precision that resulted in an overly-large gain being obtained.

Additionally, the make-up gain stage was implemented for the Moog VCF and no other filters. Its nonlinear model as presented by Huovilainen [8] still has stability issues in increasingly higher frequencies, and accuracies of resonance frequencies remained an issue in the digital models observed.

A hardware-based implementation was also not realised due to hardware restrictions. Since this project was undertaken during the Covid-19 pandemic, access to hardware equipments were not made possible, and instead a software-based implementation was instead conceived on MATLAB. Hardware-based implementation (i.e. on FPGAs or DSP chips) could allow many processes used in this project to be run in parallel or pipelined, in particular the K-weighting and SMA/EMA algorithms for the reference input and VCF filter output. This would considerably improve the currently-fast speed and guarantee real-time implementations as, with reference to Fig. 3.15, this would nullify $K + A + L$ delays from the make-up gain pipeline.

The ITU-R BS.1770 and EBU's recommendations also stated other modules along the K-weighting pipeline that were not implemented in this project, including noise gating and the impulse time constant. The current algorithm only considers left-right stereo audio channels, however other channels such as the centre, left-surround, and right-surround channels exist.

5.2 Recommended further works

Potential further studies are listed as follows:

- Implement other types of filters that could be implemented in real-time in place of the Moog VCF stage.
- Investigate Huovilainen's and Daly's comments regarding their digital VCF models and research their recommended further works.
- Improve on the current make-up gain system with other loudness calculation methods as discussed in Ward's thesis [19].
- Implement the VCF and make-up gain system on hardware-based platforms, and convert some processes such that they are run in parallel or pipelined.
- Investigate whether feedback AGCs (or combined feedback-feedforward) can be implemented, such that it could help with gain stability (i.e. in floating-point errors or issues

with the impulse loudness).

- Given so, convert the hardware-based implementation into a DIY synthesiser module (i.e. compatible with the Eurorack modular synthesisers).
- If implemented on a modular synthesiser block, investigate whether the make-up gain can be its own module, and whether the input and reference signals can be input to the module externally.
- Devise new means of obtaining the make-up gain such that precision loss is minimised.
- Investigate the ITU and EBU's noise gating techniques, and discuss how its additions improved loudness measures, or otherwise.
- Implement the algorithm and make necessary modifications to accommodate for all channels as discussed by the ITU-R BS.1770 [20].
- Adjust the audio plugin such that it allows all sampling frequencies and any number of channels.
- Implement the entirety of this project in C++, and port the algorithms as a C++-based plugin (i.e. with the JUCE framework).

A | Appendix

Written scripts/codes and software used

All listed scripts and test audio files are uploaded to the project repository on GitHub:

github.com/jpiamjariyakul/makeUpGainStage

Algorithms & Scripts	Source	Usage & Modifications
`main.m`	MATLAB	Main simulation file to obtain results of the VCF from `f_runVcf.m`, and make-up gain stages from `f_makeup_ema.m` and `f_makeup_sma.m`, and plot the time and frequency responses (the latter using `f_getFrqcResp.m`).
`f_vcf_nonlinear.m`	MATLAB	Student-written implementation of nonlinear Moog VCF. Based on works from [8], difference equations from [9].
`f_runVcf.m`	MATLAB	Student-written entry point and input buffer generator for `f_vcf_nonlinear.m`.
`f_getCoef_rlb.m`	MATLAB	Student-written implementation of method from [19] to generate RLB filter [6] coefficients for arbitrary frequencies.
`f_getCoef_preK.m`	MATLAB	Student-written implementation of method from [19] to generate the pre-K filter [6] coefficients for arbitrary frequencies.
`f_makeup_ema.m`	MATLAB	Student's own algorithm to generate make-up gain for a signal given another reference signal. Uses EMA loudness-calculating algorithms described in [19].
`f_makeup_sma.m`	MATLAB	Student's own algorithm to generate make-up gain for a signal given another reference signal. Adapts the RMS formulas described in [5] into the SMA algorithm.
`f_1dFilter.m`	MATLAB	Student's implementation of the digital biquad filter described in [19], uses coefficients obtained from `f_getCoef_preK.m` and `f_getCoef_rlb.m`.
`vst_fullSystem.m`	MATLAB	Student's own audio plugin code. Combines the implemented EMA make-up gain and VCF stages from `f_makeup_ema.m` and `f_vcf_nonlinear.m`.

Table A.1: Table of scripts written for the project

Software	Source	Usage
MATLAB	MathWorks	Main simulation and computation platform used in project. Used for generating results from simulations and experiments. Platform where Audio Toolbox is used.
Audio Toolbox	MATLAB	Used for generating the project's VST plugin.
diagrams.net	diagrams.net	Platform for drawing block diagrams for the thesis.

Table A.2: Table of software packages used in the project

Bibliography

[1] R. A. Moog, "A voltage-controlled low-pass high-pass filter for audio signal processing," in *Audio Engineering Society Convention 17*, Audio Engineering Society, 1965.

[2] A. P. Kefauver and D. Patschke, *Fundamentals of digital audio*. AR Editions, Inc., 2007, vol. 22.

[3] H. Fletcher and W. A. Munson, "Loudness, its definition, measurement and calculation," *Bell System Technical Journal*, vol. 12, no. 4, pp. 377–430, 1933.

[4] S. S. Stevens, *Psychophysics: Introduction to its perceptual, neural and social prospects*. Routledge, 1975.

[5] G. A. Soulodre, "Evaluation of objective loudness meters," in *Audio Engineering Society Convention 116*, Audio Engineering Society, 2004.

[6] International Telecommunications Union, "ITU-R BS.1770 Algorithms to measure audio programme loudness and true-peak audio level," 2006. [Online]. Available: https://www.itu.int/dms_pubrec/itu-r/rec/bs/R-REC-BS.1770-1-200709-S!!PDF-E.pdf.

[7] T. Stilson and J. Smith, "Analyzing the Moog VCF with considerations for digital implementation," in *Proceedings of the 1996 International Computer Music Conference, Hong Kong, Computer Music Association*, 1996.

[8] A. Huovilainen, "Non-linear digital implementation of the Moog ladder filter," in *Proceedings of the International Conference on Digital Audio Effects (DAFx-04)*, 2004.

[9] P. Daly, "A comparison of virtual analogue Moog VCF models," *Master's thesis, Univ. of Edinburgh, Edinburgh, UK, Aug*, 2012.

[10] MathWorks, Inc., *MATLAB Release 2021a*. [Online]. Available: https://mathworks.com/products/matlab.html.

[11] ——, *Audio Toolbox Release 2021a*. [Online]. Available: https://mathworks.com/products/audio.html.

[12] P. Hill, *Audio and speech processing with MATLAB.* CRC Press, 2018.

[13] D. M. Randel, *The Harvard Dictionary of Music.* Harvard University Press, 2003.

[14] R. G. Lyons, *Understanding digital signal processing, 3rd Edition.* Pearson Education, 2004.

[15] "Home recording studio solutions for everyday musicians," *E-Home Recording Studio*, Accessed: 03-04-2021. [Online]. Available: https://ehomerecordingstudio.com/.

[16] K. Kosbar, "Loudness," *EE 3430 - Digital Communications*, Accessed: 03-04-2021. [Online]. Available: https://web.mst.edu/~kosbar/ee3430/ff/fourier/notes_loudness.html.

[17] "Acoustics - normal equal-loudness-level contours," International Organization for Standardization, Geneva, Switzerland, Standard, 2003.

[18] American Engineering and Industrial Standards, *American tentative standards for sound level meters for measurement of noise and other sounds.* American Standards Association, 1936.

[19] D. Ward, "Applications of loudness models in audio engineering," Ph.D. dissertation, Birmingham City University, 2017.

[20] International Telecommunications Union, "ITU-R BS.1770 Algorithms to measure audio programme loudness and true-peak audio level, 4th revision," 2015. [Online]. Available: https://www.itu.int/dms_pubrec/itu-r/rec/bs/R-REC-BS.1770-4-201510-I!!PDF-E.pdf.

[21] European Broadcast Union, "EBU Tech 3341 Loudness metering: 'EBU mode' metering to supplement loudness normal-isation in accordance with EBU R 128," 2011. [Online]. Available: https://tech.ebu.ch/docs/tech/tech3341.pdf.

[22] E. Skovenborg and S. H. Nielsen, "Evaluation of different loudness models with music and speech material," in *Audio Engineering Society Convention 117*, Audio Engineering Society, 2004.

[23] J. P. A. Pérez, S. C. Pueyo, and B. C. López, *Automatic gain control.* Springer, 2011.

[24] THAT Corporation, "The Mathematics of Log-Based Dynamic Processors," *Application Note 101A*, 2009.

[25] J. Palance, *The Minimoog synthesiser operation manual*, Accessed: 24-03-2021. [Online]. Available: www.fantasyjackpalance.com/fjp/sound/synth/synthdata/16-moog-minimoog.html.

[26] T. E. Stinchcombe, "Analysis of the Moog transistor ladder and derivative filters," 2008.

[27] A. S. Sedra and K. C. Smith, *Microelectronic circuits*. New York: Oxford University Press, 1998.

[28] B. A. Hutchins, *Musical Engineer's Handbook: Musical Engineering for Electronic Music*. Electronotes, 1975.

[29] D. Rossum, "Making digital filters sound "analog"," in *Proceedings of 1992 ICMC*, 1992.

[30] C. v. Wensem, "How to make a track sound 'warm' and 5 other confusing audio terms, explained," *Sonicbids Blog*, Accessed: 29-03-2021. [Online]. Available: https://blog.sonicbids.com/how-to-make-a-track-sound-warm-and-5-other-confusing-audio-terms.

[31] G. James, *Modern Engineering Mathematics 5th Edition*. Pearson Prentice Hall, 2008.

[32] J. O. Smith, *Impulse Invariant Method*, Accessed: 30-03-2021. [Online]. Available: https://ccrma.stanford.edu/~jos/pasp/Impulse_Invariant_Method.html.

[33] Steinberg Media Technologies GmbH, *Technologies*, Accessed: 16-04-2021. [Online]. Available: https://www.steinberg.net/en/company/technologies.html.

[34] *VST SDK 3.6.7 released*, Accessed: 22-04-2021. [Online]. Available: https://www.steinberg.net/en/newsandevents/news/newsdetail/article/vst-sdk-367-released-4165.html.

[35] J. Watkinson, *The Art of Digital Audio*. Taylor & Francis, 2001.

[36] I. Mateljan, "Arta," *User Manual, Program for Impulse Response Measurement and Real Time Analysis of Spectrum and Frequency Response, Version 1.9.1*, vol. 1, no. 1, 2019.

[37] E. Whitacre, *Water Night*, Audio media, Accessed: 26-04-2021. [Online]. Available: https://youtu.be/1DQQmtNuXUU.