MATLAB code for a distortion measure based on human auditory masking

Dimme de Groot
ver1.0

May 16, 2024

Abstract

This report provides a short overview on the distortion measure implemented in the MATLAB code provided on this GitHub project. Effectively, the code represents my interpretation of the work presented by Van de Par et al. in [1][1]. The distortion measure is used to estimate if a human observer can notice an acoustic distortion in the presence of some other acoustic signal. The major advantage of the implemented distortion measure is that it is computationally inexpensive and can be represented as a weighted l_2 norm on the distortion for a fixed masker.

1 Introduction

The distortion measure implemented on this GitHub was introduced by Van de Par et al. in [1] and will hereafter be referred to as the Par-measure. The measure predicts if a human can detect an acoustic disturbance ϵ in the presence of a different acoustic signal s. Signal s is called the masker, since it influences the audibility of the disturbance[2]. Let's illustrate this with a simple example.

Suppose I am playing a piece of music which ideally is given by signal s. However, my washing machine is making some noise in the background so I do not hear s but I hear the noisy audio $\hat{s}$. The difference $\epsilon = \hat{s} - s$ is the disturbance and would, in this example, be the noise of the washing machine. The Par-measure would take the disturbance ϵ and the received signal (the masker) $\hat{s}$ and compute the perceived distortion as a single number d. Mathematically, this is

$$D(\hat{s}, \epsilon) = d, \tag{1}$$

where $D(\cdot)$ represents the Par-measure. If $d > 1$, it is predicted that I can hear the difference between my original music s and what I actually hear $\hat{s}$. If $d \leq 1$, the Par-measure thinks that I can not notice the difference between the two. That is, I do not notice the washing machine! But how does this work?

In the following, I first describe the basic auditory model employed in the Par-measure and than give the equations resulting in the actual distortion measure D. After this, the functions in the code are briefly explained and an example in which the Par-measure is used to increase the loudness of acoustic signals is given. This example is based on [2]. I want to stress that none

[1] The article is an open access article, see `https://link.springer.com/content/pdf/10.1155/ASP.2005.1292.pdf`

[2] It should be mentioned that the Par-measure was developed for sinusoidal audio coding. So, "officially" it expects the distortions to be sinusoids. In my experience, it works quite well even for more complex distortions.

of the theoretical work given below is my own, so I tried to be complete in giving the references. Additionally, the content of this small report is mostly adapted from my master's thesis[3].

2 Auditory Model

The Par-measure is based on a very simple model of the human auditory system. The effect of the outer- and middle-ear is modelled by the inverse of the threshold in quiet[4] h_{om}. The effect of the inner-ear (cochlea) is modelled by a gammatone filterbank consisting of N_g filters $h_1, h_2, \ldots, h_{N_g}$. Lastly, the inability of humans to detect very soft tones is modelled by adding a constant c_1, which can be interpreted as the internal noise of the hearing system. The model is illustrated in Fig. 1.

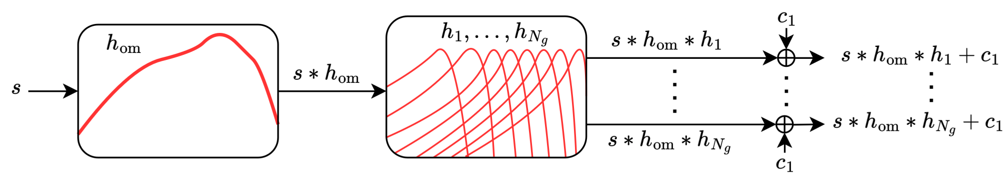

Figure 1: The model of the human hearing system used in the Par-measure. From left to right, we have an audio signal s which arrives at the ear of the listener. The outer-and middle-ear filter this signal by with a time-domain filter h_{om}. The inner-ear is modelled by a gammatone filterbank consisting of N_g time-domain filters $h_1, h_2, \ldots, h_{N_g}$. Lastly, the constant c_1 is added and models the inability of humans to hear very soft tones. The $*$ represents the convolution operator.

In sections 2.1 and 2.2 the expressions for the frequency-domain outer- and middle-ear filter $\hat{h}_{\mathrm{om}}$ and for the frequency domain gammatone filter $\hat{h}_g$ with $g \in \{1, 2, \ldots, N_g\}$ are given. The hat on top of the symbols simply indicates that we are in the frequency domain instead of in the time domain. In Section 2.3, the full equation defining the Par-measure is given and Fig. 2b of [1] is recreated.

2.1 The outer- and middle-ear filter

The first filter used in the Par-measure is referred to as the outer- and middle-ear filter. In my opinion, this is a bit of a weird name, since in practice the filter is taken to be the inverse of the threshold in quiet. The threshold in quiet, measured in dB SPL (sound pressure level), can be approximated as [4, Eq. (1)]

$$T_q(f) = 3.64 \left(\frac{f}{1000}\right)^{-0.8} - 6.5 \exp\left\{-0.6 \left(\frac{f}{1000} - 3.3\right)^2\right\} + 10^{-3} \left(\frac{f}{1000}\right)^4 \quad \text{[dB SPL]} \tag{2}$$

with f the frequency in hertz, which for human hearing ranges from about 20 Hz to 20 kHz[5]. There is a problem here! Namely, we need to work with digital presentations of the audio

[3] Accessible from `http://resolver.tudelft.nl/uuid:ee669571-b15a-4f0d-99cd-4c6c35acbd45`

[4] The threshold in quiet is a curve showing what is the minimum sound pressure that is needed for a human to be able to detect a given tone when there is no sound otherwise. See Chapter 2 of [3].

[5] I'm not entirely sure what is the frequency range of the measurements on which this curve was fitted. Anyway, it is likely correct at least approximately over the mentioned range and simply gives very large values outside that range, which is also fine!

signals, but human hearing is based on a physical representation: the sound pressure level. To circumvent this, I assume to know that a certain value in the digital domain (say $s = 1$) corresponds to a sound pressure level of a certain amount, say $s_{\mathrm{SPL}} = 70$ dB SPL. We can than compute a constant $\frac{\alpha}{p_0}$ which allows to map the digital number to a physical sound pressure level. This is explained in more detail in Appendix A. For now, it suffices to note that

$$s_{\mathrm{SPL}} = 20 \log_{10} \left(\frac{\alpha}{p_0} |s| \right) \quad \text{[dB SPL]}, \tag{3}$$

which allows to compute the frequency-domain outer- and middle-ear filter as

$$\hat{h}_{\mathrm{om}}(f) = \left(\frac{\alpha}{p_0} \right)^{-1} 10^{-T_q(f)/20}. \tag{4}$$

Note that a hat was added on top of h_{om}. This hat represents frequency domain variables and will be used to distinguish between variables when there is both a time- and frequency domain version of them.

2.2 The Gammatone filterbank

The inner-ear filter used in the Par-measure is a gammatone filterbank[6] consisting of N_g filters. The magnitude spectrum of a gammatone filter centered at frequency f_g (in Hz) is approximated as [1, Eq. 2],

$$\hat{h}_g(f) = \left(1 + \left(\frac{f - f_g}{\kappa \mathrm{ERB}(f_g)} \right)^2 \right)^{-\eta/2}, \tag{5}$$

with $g \in \{1, 2, \ldots, N_g\}$. In the equation, κ is a normalising constant, η is the order of the filter and $\mathrm{ERB}(f_g)$ is the equivalent rectangular bandwidth[7] of the gammatone filter centered at f_g (see (7)). The filter order is assumed to be $\eta = 4$ [1]. The corresponding normalising constant is given by

$$\kappa = \frac{2^{\eta-1}(\eta-1)!}{\pi(2\eta-3)!!} \tag{6}$$

with ! the factorial and !! the double factorial. The double factorial $n!!$ equals $2 \cdot 4 \cdot \ldots \cdot n$ for even positive numbers n, and $1 \cdot 3 \cdot 5 \cdot \ldots \cdot n$ for odd positive numbers n. Lastly, the value $\mathrm{ERB}(f_g)$ is approximated using [5]

$$\mathrm{ERB}(f_g) = 24.7 \left(\frac{4.37 f_g}{1000} + 1 \right). \tag{7}$$

Par et al. choose the N_g center frequencies f_g such that they linearly divide the ERB-rate scale[8] on the interval $[E(0), E(f_s/2)]$ in N_g steps [1]. Here, f_s is the sampling frequency. The ERB-rate scale is approximated as [5]

$$E(f_g) = 21.4 \log_{10} \left(\frac{4.37 f_g}{1000} + 1 \right). \tag{8}$$

[6] A filterbank is just a bunch of filters placed in parallel. The gammatone filterbank is often used as a simple model for the cochlea (the “snailhouse” in your ear).

[7] The Equivalent Rectangular Bandwidth (ERB) of some reference filter is the bandwidth of a rectangular filter which has the same maximum magnitude as the reference filter and transmits the same power when white noise is given as input [3]. Even though they are not rectangular, auditory filters are often characterised using their ERB. Intuitively, you can interpret $\mathrm{ERB}(f_g)$ as the sensitivity of our hearing to distortions at frequencies around f_g: a higher ERB means that our ears are less sensitive.

[8] The ERB-rate scale measures how much ERBs are below a given frequency.

2.3 The distortion measure

We now have expressions for the gammatone filterbank h_g with $g \in \{1, \ldots, N_g\}$ (Eq. (4)) and for the outer- and middle-ear filter h_{om} (Eq. (5)). Together with Fig. 1, it can be found that the masking power corresponding to filter g and masker s is given by [1, Eq. 3]

$$P_{\text{mask}}^{(g)} = \frac{1}{N} \sum_f |\hat{h}_{\text{om}}(f)|^2 |\hat{h}_i(f)|^2 ||\hat{s}(f)|^2 + \frac{1}{N} c_1 \tag{9}$$

and, similarly, the distortion power corresponding to filter g and disturbance ϵ is given by

$$P_{\text{dist}}^{(g)} = \frac{1}{N} \sum_f |\hat{h}_{\text{om}}(f)|^2 |\hat{h}_i(f)|^2 ||\hat{\epsilon}(f)|^2. \tag{10}$$

There are three things to note in these equations. Firstly, the value N is the length of a single audio-frame[9]: the Par-measure operates on frames of about 20 to 40 ms, so that for a sample frequency of 16 kHz the frame length ranges from $N = 320$ to $N = 640$. A little bit more information on the framing of audio signals is given in Section B. Secondly, the distortion power does not $P_{\text{dist}}^{(g)}$ does not have the constant c_1 added. This is because c_1 can be considered as part of the masking signal. Thirdly, the summation is over all N frequency bins[10] f.

The distortion $D_g(s, \epsilon)$ per filter g is found as the ratio between (10) and (9). That is

$$D_g(s, \epsilon) = \frac{P_{\text{dist}}^{(g)}}{P_{\text{mask}}^{(g)}} = \frac{\sum_f |\hat{h}_{\text{om}}(f)|^2 |\hat{h}_i(f)|^2 ||\hat{\epsilon}(f)|^2}{\sum_{f_m} |\hat{h}_{\text{om}}(f_m)|^2 |\hat{h}_i(f_m)|^2 ||\hat{s}(f_m)|^2 + c_1}, \tag{11}$$

so that the total distortion $D(s, \epsilon)$ is given as

$$D(s, \epsilon) = c_2 \sum_{g=1}^{N_g} D_g(s, \epsilon) = c_2 \sum_{g=1}^{N_g} \frac{\sum_f |\hat{h}_{\text{om}}(f)|^2 |\hat{h}_i(f)|^2 ||\hat{\epsilon}(f)|^2}{\sum_{f_m} |\hat{h}_{\text{om}}(f_m)|^2 |\hat{h}_i(f_m)|^2 ||\hat{s}(f_m)|^2 + c_1}, \tag{12}$$

where c_2 is a weighting factor which is set to ensure that $D(s, \epsilon) = 1$ when the distortion is just noticeable. The values c_1 and c_2 are set through a calibration procedure, of which I won't go into detail. The calibration is based on the reference value in digital domain s and the corresponding physical sound pressure level s_{SPL}. For the calibration, [1] references [6], who proofs that the calibration always converges. However, the proof contains an error, which luckily in practice is not that much of an issue.

With (12) we have a nice but pretty complicated expression for $D(s, \epsilon)$. Let's see if it can be made a bit more simple. Eq. (12) can be rewritten as

$$\begin{aligned} D(s, \epsilon) &= c_2 \sum_{g=1}^{N_g} \sum_f |\hat{\epsilon}(f)|^2 \frac{|\hat{h}_{\text{om}}(f)|^2 |\hat{h}_i(f)|^2|}{\sum_{f_m} |\hat{h}_{\text{om}}(f_m)|^2 |\hat{h}_i(f_m)|^2 ||\hat{s}(f_m)|^2 + c_1} \\ &= \sum_f |\hat{\epsilon}(f)|^2 c_2 \sum_{g=1}^{N_g} \frac{|\hat{h}_{\text{om}}(f)|^2 |\hat{h}_i(f)|^2|}{\sum_{f_m} |\hat{h}_{\text{om}}(f_m)|^2 |\hat{h}_i(f_m)|^2 ||\hat{s}(f_m)|^2 + c_1}, \end{aligned} \tag{13}$$

[9] The reason that this N shows up in the equations is due to a property of the discrete Fourier transform, which is typically defined such that $||\hat{s}||^2 = N||s||^2$. I.e. if we want the discrete time-domain and discrete frequency-domain signals to have equal power, there needs to be a correction by dividing by the signal length N. See `https://en.wikipedia.org/wiki/Parseval%27s_theorem`

[10] A frequency bin is similar to a 'normal' frequency but for discrete signals. Since the signal is discrete, there is only a finite number of frequencies which are not connected. A single frequency bin thus represents a small range of frequencies, which is why it is called a frequency bin.

which still looks pretty complicated! However, note that the right summation (including c_2) is only dependent on the masker s and not on the disturbance ϵ. It follows that we may write

$$D(s, \epsilon) = \sum_f \frac{|\hat{\epsilon}(f)|^2}{(v(f))^2}, \tag{14}$$

where $\frac{1}{(v(f))^2}$ is given by

$$\frac{1}{(v(f))^2} = c_2 \sum_{g=1}^{N_g} \frac{|\hat{h}_{\mathrm{om}}(f)|^2 |\hat{h}_i(f)|^2|}{\sum_{f_m} |\hat{h}_{\mathrm{om}}(f_m)|^2 |\hat{h}_i(f_m)|^2 ||\hat{s}(f_m)|^2 + c_1}. \tag{15}$$

which can be computed based on the masker s only! Thus, if the masker remains constant, we can evaluate a lot of distortions without having to recompute (15) all the time. In fact, [1] shows that $(v(f))^2$ estimates the masking curve for sinusoidal distortions. A sinusoidal acoustic distortion can be considered inaudible if it lies below the masking curve. In Fig. 2, an example masking curve is shown where the masker is a sinusoid at a frequency of 1 kHz and with a sound pressure level of 50 dB SPL. This figure reproduces Fig. 2b of [1].

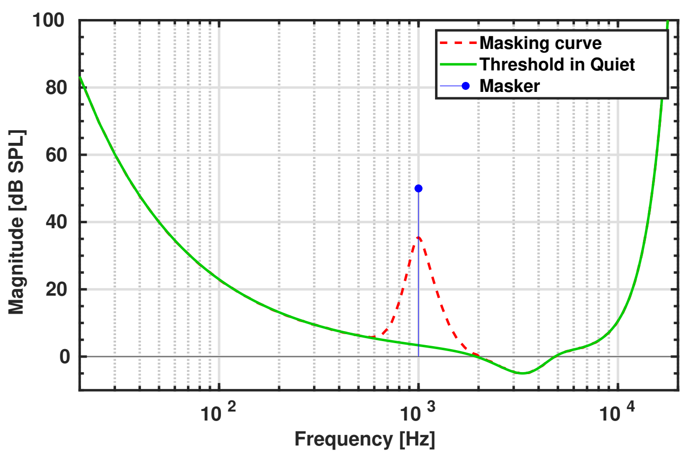

Figure 2: The estimated masking curve $(v(f))^2$ for a 1 kHz 50 dB SPL. Note that the masking curve is only affected in the frequency range around the masker and reduces to the threshold in quiet at all other frequencies. All sinusoidal distortions which lie below the dashed red line can be considered inaudible. This figure reproduces Fig. 2b of [1].

This finalises background of the Par-measure. In the next section, the MATLAB code is discussed briefly.

3 The MATLAB Par-measure object

The Par-measure is implemented as a MATLAB object and constructed by calling `par_measure(Fs, Tframe, x_ref, x_dB_ref, F_cal, Ng)`. Here, `F_s` is the sampling frequency, `Tframe` the length of a single frame in seconds, `x_ref` the reference value in the digital domain, `x_dB_ref` the corresponding value in dB SPL, `F_cal` the frequency at which the Par-measure is calibrated (i.e. for which the constants c_1 and c_2 are calculated) and `N_g` is the number of gammatone filters in the gammatone filterbank. A typical input is

```
Fs = 48000;                 %[Hz],  Sampling frequency
Tframe = 0.04;              %[s],   the length of the input frame in seconds.
x_ref = 1; x_dB_ref = 65;   %[-],[dB SPL]; the reference values in digital and physical domain
F_cal = 1000;               %[Hz],  The calibration frequency.
Ng = 64;                    %[-],   The number of gammatone filters used
Par_measure = par_measure(Fs, Tframe, x_ref, x_dB_ref, F_cal, Ng);
```

The value `x_dB_ref = 65` dB SPL was taken because this is a typical value in conversational speech [3]. For `N_g`, 32 or 64 are typical values. For `Fs`, 48000, 44100, 16000 and 8000 are typical values. For `Tframe`, anything from 20 to 40 ms is typical. For `F_cal`, 1000 Hz is the typical value. Please not that, for most parameters, I did not test a wide range of values. In case you notice any issues, feel free to leave a comment (or whatever it is called) in the GitHub. The object `Par_measure` comes with a few functions, which are listed below.

Firstly, to plot the masking curve (i.e. a figure which is similar to Fig. 2), you can use `Par_measure.plot_maskcurve(masker)`, where `masker` is the discrete time-domain masker of the proper length (i.e. of length `Tframe`. When you create the object, the expected length of the masker is printed in the MATLAB command window! In case you also want to plot the disturbance, you can use `Par_measure.plot_maskcurve(masker, disturbance)`.

Secondly, to convert from a physical sound pressure level in dB SPL to the digital amplitude, you can use `A_digital = Par_measure.physical_to_digital(A_physical)`. There is also the inverse function `A_physical = Par_measure.digital_to_physical(A_digital)`.

Lastly, to obtain the masking curve $(v(f))^2$ (see (15)) you can use `[maskcurve, maskcurve_spl, p_par] = Par_measure.comp_maskcurve(masker)`. Here, `masker` is again the discrete time-domain masker with a length `Tframe`. Up to some normalisations, `maskcurve` represents $v(f)$ and `p_par` represents $1/v(f)$. Here, f are frequency bins ranging from $f = -f_s/2$ to $f = f_s/2$. Variable `maskcurve_spl` is simply the masking curve $v(f)$ expressed in dB SPL and has a frequency ranging from $f = 0$ to $f = f_s/2$. Note that f_s is the sample frequency.

4 Examples

On the GitHub, I added two example files. The first file, `example1_basics.m`, shows most of the functionality mentioned above. In my opinion, this is not a very interesting example, as you don't really see how the Par-measure can be used. In the second example, the loudness of acoustic signals is increased by making use of the Par-measure. This example is based entirely on the work of Jeannerot et al. [2] and clearly shows why measuring distortion based on human perception is useful. To run this example you need CVX[11], which is a free MATLAB package with which you can do a lot of cool stuff (solving convex optimisation problems, you might find it boring...). The example is described in detail in the following subsecion.

4.1 Example 2: loudness increase of acoustic signals

The work presented below is entirely based on [2].

When you are using a loudspeaker the highest volume which can be reproduced is typically dependent on constraints such as the maximum displacement of the voice coil or the maximum amplifier voltage. Now consider that we are listening to some music which is represented by s_0. The maximum value which our loudspeaker can represent is λ, i.e. we should have that

[11] At the time of writing CVX is available from `https://cvxr.com`

$\max(|s_0|) \leq \lambda$. However, since we are listening to a nice song[12] we want to turn up the volume by a factor $\beta > 1$. Our new (louder) signal is called $s^\star = \beta s_0$, where the * represents that it is the desired signal. After turning up the volume a lot, the playback signal suddenly starts to become distorted: we expected to get an undistorted louder version of our original signal ($s^\star$), but instead we get a distorted version $\tilde{s}$: the loudspeaker is used outside its operating range! This is illustrated in Fig. 3.

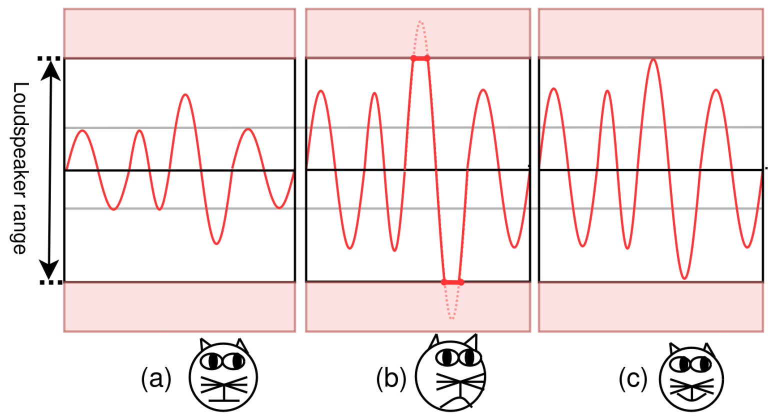

Figure 3: An illustration of what happens when the volume is increased to much. In (a), the undistorted signal s is shown. In (b), the playback volume is increased beyond the operating range. We expected to get the louder signal $s^\star$ (dotted red line) but instead clipping occurs and we get a distorted version $\tilde{s}$. In (c), the clipping artefacts are reduced by adding less amplification near the region where clipping occurs.

In the example code, I have implemented two methods to perform the loudness increase: hard clipping loudness increase based on the Par-measure.

4.1.1 Hard clipping

Hard clipping effectively results in the effect as observed in Fig. 3b. If the maximum allowable signal value is given by λ, the result after clipping is

$$s_{\text{clip}}[n] = \begin{cases} s^\star[n] & \text{if } |s^\star[n]| < \lambda, \\ \text{sign}\left(s^\star[n]\right)\lambda & \text{otherwise.} \end{cases} \tag{16}$$

So that $\max(|s_{\text{clip}}|) \leq \lambda$. Note that n is the sample-index.

4.1.2 Loudness increase using the Par-measure

The second method to increase loudness is the method which was proposed by Jeannerot et al. in [2] and is slightly more involved. First of all, let me briefly illustrate my understanding of the concept.

Instead of simply amplifying the signal, [2] proposes to try to *reduce* the maximum value of the original signal $\max(|s_0|)$. Mathematically, this looks something like this (though strictly

[12] In this example that song is *Twin Shadow* from Canine, see `https://www.youtube.com/watch?v=XHIAbhjSZFI`

speaking the equation is incorrect!):

$$\tilde{s}_{\text{Jeannerot}} = \min \max\left(|s_0|\right). \tag{17}$$

Note that this *minimize* the *maximum* value. After this step has been done, the actual playback signal $s_{\text{Jeannerot}}$ is obtained as

$$s_{\text{Jeannerot}} = \lambda \frac{\tilde{s}_{\text{Jeannerot}}}{\max(|\tilde{s}_{\text{Jeannerot}}|)}. \tag{18}$$

Consequently, $\max\left(|s_{\text{Jeannerot}}|\right) = \lambda$, which is inside the operating rang of the loudspeaker[13]. The concept of (17) and (18) is illustrated in Fig. 4.

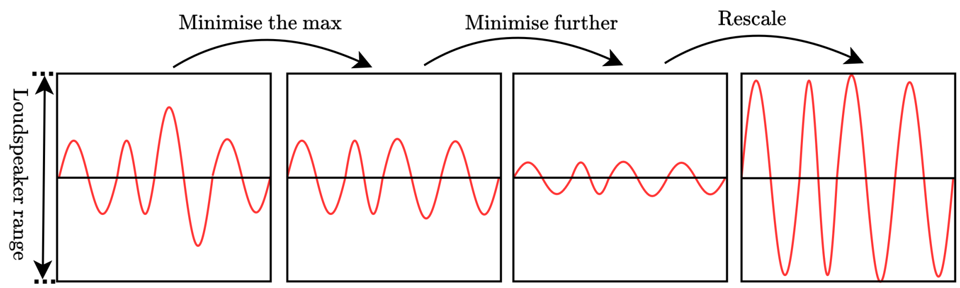

Figure 4: The conceptual steps of the method to increase loudness as proposed in [2]. The maximum value is minimised, after which the total result is rescaled to get a signal with an increased loudness. The minimisation can continue as long as the new signal resembles the target signal sufficiently close.

The approach as outlined above can, in principle, continue the minimisation forever until there is no playback signal left. To avoid $s_{\text{Jeannerot}}$ deviating to much from the target signal, we constrain the minimisation to keep the distortion below a certain value d. This is done using the Par-measure. Mathematically, this can be written as

$$\begin{array}{ll} \text{minimize} & \max(|s|) \\ \text{subject to} & D(s, s - s_0) \leq d. \end{array} \tag{19}$$

The value s satisfying this equation is $\tilde{s}_{\text{Jeannerot}}$ and can be rescaled according to (18), yielding $s^{\star}_{\text{Jeannerot}}$. There are two caveats: a) the Par-measure operates on short-time frames, and b) when s is used as masker in D, equation (19) is not convex[14]. Luckily, we can easily change (19) to alleviate the previously mentioned problems. Firstly, we can assume that the output signal s resembles s_0 sufficiently closely so that it is reasonable to use s_0 instead of s as the masker[15]. Eq. (19) becomes

$$\begin{array}{ll} \text{minimize} & \max(|s|) \\ \text{subject to} & D(s_0, s_0 - s) \leq d. \end{array} \tag{20}$$

[13] In case this is too loud, you can just use some different $\lambda' < \lambda$.

[14] The reason that we are searching for convex optimisation problems is that these are relatively easy to solve since their local minimum is also their global minimum. The standard textbook on convex optimisaiton is by S. Boyd and L. Vandenberghe and can be accessed from `https://web.stanford.edu/~boyd/cvxbook/`

[15] The signal s_0 accordingly so that it also has the correct sound pressure level, but I will ignore this here. If you do so, you would use $s_0' = \gamma s_0$ as masker instead of s_0 directly, where γ is the required scaling. Anyhow, choosing γ is a bit of a guess.

Now we only need to figure out how to work in frames! Framing is explained in Appendix B, where you can read that the l^{th} frame $s_0^{(l)}$ is given by

$$s_0^{(l)}[n] = w_1[n - lR]s_0[n], \tag{21}$$

here $w_1[n]$ is a window which is zero for $n < 0$ and $n > L-1$. Parameter L is called the window length. For each frame l, the window is shifted by $R \leq L$ samples, which allows subsequent frames to have some overlap. In the code, I used the square root Hanning window for w_1, see (49) and (50). The frames $s^{(l)}$ of s can be defined similarly. Putting the frames in (20) yields

$$\begin{aligned} &\text{minimize} && \max(|s^{(l)}|) \\ &\text{subject to} && D\left(s_0^{(l)}, s_0^{(l)} - s^{(l)}\right) \leq d. \end{aligned} \tag{22}$$

The value $s^{(l)}$ for which this equation is satisfied is called $\tilde{s}^{(l)}_{\text{Jeannerot}}$. The total signal $\tilde{s}_{\text{Jeannerot}}$ is found as

$$\tilde{s}_{\text{Jeannerot}}[n] = \sum_l w_2[n - lR]\tilde{s}^{(l)}_{\text{Jeannerot}}[n], \tag{23}$$

so that, from (18), the loudness increased signal $s_{\text{Jeannerot}}$ is given as

$$s_{\text{Jeannerot}} = \lambda \frac{\tilde{s}_{\text{Jeannerot}}}{\max(|\tilde{s}_{\text{Jeannerot}}|)}. \tag{24}$$

In the MATLAB implementation, the window w_2 is a square root Hanning window.

MATLAB Code While equations (21), (22), (23) and (24) give the full method, they do not really resemble the MATLAB code yet. Since the purpose of this report is to understand how to use the Par-measure object, I will rewrite (22) to resemble the MATLAB implementation slightly more. We can define a vector

$$\mathbf{s}_0^{(l)} = \begin{bmatrix} s_0^{(l)}[lR] \\ s_0^{(l)}[lR+1] \\ s_0^{(l)}[lR+2] \\ \vdots \\ s_0^{(l)}[lR+L-2] \\ s_0^{(l)}[lR+L-1] \end{bmatrix}, \tag{25}$$

which looks pretty complicated but does nothing else than stacking the non-zero values of s_l into a vector. We may similarly define the vector $\mathbf{s}^{(l)}$

$$\mathbf{s}^{(l)} = \begin{bmatrix} s^{(l)}[lR] \\ s^{(l)}[lR+1] \\ s^{(l)}[lR+2] \\ \vdots \\ s^{(l)}[lR+L-2] \\ s^{(l)}[lR+L-1] \end{bmatrix}, \tag{26}$$

and the disturbance $\boldsymbol{\epsilon}$

$$\boldsymbol{\epsilon}^{(l)} = \mathbf{s}_0^{(l)} - \mathbf{s}^{(l)}. \tag{27}$$

Note that all these vectors are in $\mathbb{R}^L$ (i.e. they consist of L real scalars). We can convert them to the frequency domain by taking the discrete Fourier transform (DFT). This transform can be implemented using the $L \times L$ DFT-matrix[16] $\mathbf{W}$. So that

$$\hat{\mathbf{s}}_0^{(l)} = \mathbf{W}\mathbf{s}_0^{(l)}, \tag{28a}$$

$$\hat{\boldsymbol{\epsilon}}^{(l)} = \mathbf{W}\boldsymbol{\epsilon}^{(l)} = \mathbf{W}\left(\mathbf{s}_0^{(l)} - \mathbf{s}^{(l)}\right). \tag{28b}$$

Note that each elements of these vectors corresponds to a frequency f, so that we have L frequency bins given by f_0, f_1, $\ldots$, f_{L-1}. We will now use (14) and (15) to write the Par-measure in a vector form. First, define vector $\mathbf{p}$, which is obtained by stacking the elements $p(f) = \sqrt{\frac{1}{(v(f))^2}}$ and can be computed from $\hat{\mathbf{s}}_0^{(l)}$ using (15). It is given by

$$\mathbf{p} = \begin{bmatrix} p(f_0) \\ p(f_1) \\ p(f_2) \\ \vdots \\ p(f_{L-2}) \\ p(f_{L-1}) \end{bmatrix}. \tag{29}$$

Using (14), the Par-measure for vectors can now be written as

$$D(\mathbf{s}_0^{(l)}, \mathbf{s}_0^{(l)} - \mathbf{s}^{(l)}) = \left|\left|\mathrm{diag}(\mathbf{p})\mathbf{W}\left(\mathbf{s}_0^{(l)} - \mathbf{s}^{(l)}\right)\right|\right|_2^2, \tag{30}$$

where $||\cdot||_2$ represents the l_2 norm and diag constructs a matrix with $\mathbf{p}$ on the diagonal and zeros otherwise. Lastly, the maximum absolute value of a vector $\mathbf{x}$ is given by the so-called infinity norm, $||\mathbf{x}||_\infty$. Using this and (30), we may write (22) in a vector-form as

$$\begin{aligned} &\text{minimize} && \left|\left|\mathbf{s}^{(l)}\right|\right|_\infty \\ &\text{subject to} && \left|\left|\mathrm{diag}(\mathbf{p})\mathbf{W}\left(\mathbf{s}_0^{(l)} - \mathbf{s}^{(l)}\right)\right|\right|_2^2 \leq d, \end{aligned} \tag{31}$$

which almost directly resembles the loudness increase as implemented in MATLAB and finalises this example!

[16] See, for example, `https://ccrma.stanford.edu/~jos/st/Matrix_Formulation_DFT.html`.

A The sound pressure level

The Sound Pressure Level (SPL) gives the intensity of an acoustic stimuli with respect to a reference value. It is given by

$$L_{\text{SPL}} = 20 \log_{10}\left(\frac{p}{p_0}\right) \quad [\text{dB SPL}], \tag{32}$$

with p the absolute sound pressure in Pascals and p_0 a reference value equal to 20 µPa [7].

To determine L_{SPL}, one needs to have access to the sound pressure p. In most applications, this value is not straightforwardly known since only a digital representation x is available. However, a false but, in my experience, satisfactory assumptions is to assume the sound pressure level to be a linear function of the input signal, i.e. $p = \alpha|x|$. Thus, we can write (32) as

$$L_{\text{SPL}} = 20 \log_{10}\left(\frac{\alpha|x|}{p_0}\right) = 20\log_{10}(|x|) + 20\log_{10}\left(\frac{\alpha}{p_0}\right) \quad [\text{dB SPL}] \tag{33}$$

where α depends on environmental factors such as the loudspeaker, cables, amplifier, and room.

To find α, prior information on the sound level needs to be available or an estimate needs to be made. To be specific, let us have access to a value $|x| = x_{\text{ref}}$ for which the received sound pressure level is $x_{\text{dB, ref}}$ (in dB SPL). Substituting this into (33) yields

$$20\log_{10}\left(\frac{\alpha}{p_0}\right) = x_{\text{dB,ref}} - 20\log_{10}\left(x_{\text{ref}}\right), \tag{34}$$

which can straightforwardly be solved for α (or $\frac{\alpha}{p_0}$).

As an example, consider a normalised digital representation, so $\max(|x|) = 1$. Suppose that this corresponds to 70 dB SPL. Solving (34) results in $\alpha = 0.0632$, or, equivalently, $\alpha/p_0 \approx 3162$. Once α is available, it is straightforward to find the amplitude corresponding to a different sound pressure level and vice versa using

$$L_{\text{SPL}} = 20\log_{10}(|x|) + 20\log_{10}\left(\frac{\alpha}{p_0}\right) \Leftrightarrow |x| = \left(\frac{\alpha}{p_0}\right)^{-1} 10^{\frac{L_{\text{SPL}}}{20}} \tag{35}$$

For this example, the value $A_{70} = 1$ and $A_{52} \approx 0.1259$. The value $A_{\text{Tq}}(f_m)$ is slightly more involved. For $f_m = 1000$ Hz, $T_q(f_m) \approx 3.37$ dB SPL. Thus, $A_{\text{Tq}}(f_m) \approx 0.466 \times 10^{-3}$.

B The framing of audio signals

First of all, there are probably a lot of sources who explain this a lot better than me (for example [8]), but for the sake of completeness, below is a small section which explains the framing of audio signals. Framing is often done in signal processing applications since audio signals can be arbitrary long and since some statistical properties are only present in short frames. Similarly, the Par-measure operates on frames of about 20 to 40 ms.

Consider an audio signal x, this signal could, for example, be a music or speech signal. Let's take Europapa by Joost Klein as an example[17]. If it is sampled at 48 kHz (i.e. 48000 samples per second), we have a total of $221 \times 48000 = 10608000$ samples, which is a lot! The individual samples are referred to as $x[n]$, with $n = 0, 1, \ldots, 10608000 - 1$. The -1 is because I started counting at zero. Because x consists of so many samples, we want to do the processing in frames.

[17] `https://www.youtube.com/watch?v=gT2wY0DjYGo`

Let's take a frame of 20 ms. This corresponds to frames of length $L = 960$ samples. Let's call the first frame s_1, the second frame s_2, etc. If we take frames without overlap, the frames are given by

$$
\begin{aligned}
s_1[n] &= \begin{cases} x[n] & \text{for } n = 0, 1, \dots 959 \\ 0 & \text{otherwise} \end{cases} \\
s_2[n] &= \begin{cases} x[n] & \text{for } n = 960, 961, \dots 1919 \\ 0 & \text{otherwise} \end{cases} \\
s_3[n] &= \begin{cases} x[n] & \text{for } n = 1920, 1921, \dots 2879 \\ 0 & \text{otherwise} \end{cases} \\
&\text{and so on and so forth.}
\end{aligned}
\tag{36}
$$

Using frame index l, we may write this a bit more compact as

$$
s_l[n] = \begin{cases} x[n] & \text{for } n = lL,\, lL+1,\, \dots,\, lL+L-1 \\ 0 & \text{otherwise.} \end{cases}
\tag{37}
$$

Note that we can recover x from the individual frames by simply summing over them

$$
x[n] = \sum_l s_l[n].
\tag{38}
$$

In practice, the frames are chosen to have some overlap. The amount of overlap is defined by the "hop length" $R \leq L$. Let us consider a hop length $R = 480$. Eq. (36) becomes

$$
\begin{aligned}
s_1[n] &= \begin{cases} x[n] & \text{for } n = 0, 1, \dots 959 \\ 0 & \text{otherwise} \end{cases} \\
s_2[n] &= \begin{cases} x[n] & \text{for } n = 480, 481, \dots 1439 \\ 0 & \text{otherwise} \end{cases} \\
s_3[n] &= \begin{cases} x[n] & \text{for } n = 960, 961, \dots 1919 \\ 0 & \text{otherwise} \end{cases} \\
&\text{and so on and so forth.}
\end{aligned}
\tag{39}
$$

We may again write this more compact as

$$
s_l[n] = \begin{cases} x[n] & \text{for } n = lR,\, lR+1,\, \dots,\, lR+L-1 \\ 0 & \text{otherwise.} \end{cases}
\tag{40}
$$

Note that we lost the property that the frames sum to $x[n]$, i.e.

$$
x[n] \neq \sum_l s_l[n],
\tag{41}
$$

for this reason, the signals are often multiplied with a window $w_1[n]$ which has a nonzero value for $n \in \{0, 1, \dots, L-1\}$ and is zero outside this range. Using this window in (40) gives

$$
s_l[n] = \begin{cases} w_1[n - lR]x[n] & \text{for } n = lR,\, lR+1,\, \dots,\, lR+L-1 \\ 0 & \text{otherwise,} \end{cases}
\tag{42}
$$

where the shift by $-lR$ is to shift the support of the window[18] to the correct set of samples. Note that, since the window is zero outside its support, (42) simplifies to

$$s_l[n] = w_1[n - lR]x[n]. \tag{43}$$

We now have that

$$x[n] = \sum_l s_l[n] \tag{44}$$

if

$$\sum_l w_1[n - lR] = 1 \quad \text{for all } n. \tag{45}$$

Eq. (45) is known as the constant overlap add condition. Recall that n are integers. To see that (45) is important, let us rewrite (44) using (43). This gives

$$x[n] = \sum_l s_l[n] = \sum_l w_1[n - lR]x[n] = x[n] \sum_l w_1[n - lR], \tag{46}$$

so if $\sum_l w_1[n - lR] = 1$, we indeed get that the frames sum to $x[n]$ again! This might all look kind of complicated, so let's have a look at what the frames look like in a simple example shown in Fig. 5. In this example, a so called "Hanning" window is used with a length $L = 20$ and a hop length of $R = 10$. The windows $w_1[n - lR]$ are plotted for $l = 0$ up to and including $l = 9$. The sum of the windows is plotted as well. Note that the constant overlap add condition is

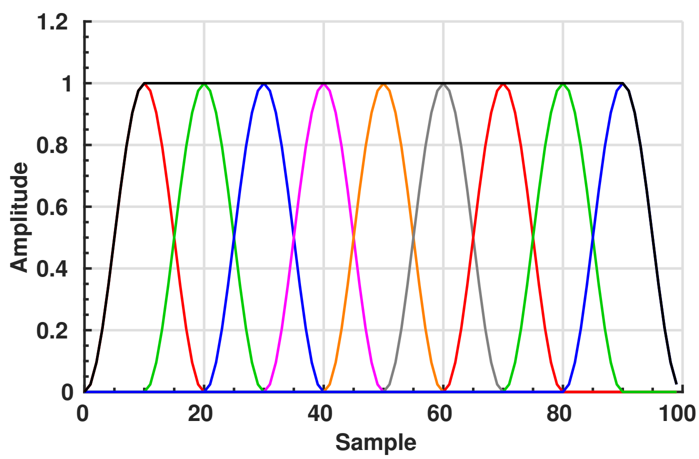

Figure 5: An illustration of a window satisfying the constant overlap add condition with a length of $L = 20$ and a hop length $R = 10$. The window is a Hanning window. The colored lines are the individual frames and the black line is the sum of the frames.

satisfied everywhere apart from at the boundaries. This is typically not a problem, since at the boundaries the audio signal is typically very soft. The Hanning window of length L is given by [8]

$$\text{hann}_L(n) = \begin{cases} \frac{1}{2} + \frac{1}{2}\cos\left(\frac{2\pi}{L}\left(n - \frac{L}{2}\right)\right) & \text{for } n = 0, 1, \ldots, L-1 \\ 0 & \text{otherwise.} \end{cases} \tag{47}$$

[18] The support of the window is the range of n where the window its value is nonzero.

As a final note, when doing nonlinear processing on the frames, it is often better to also use a window when gluing the frames back together. I.e.

$$x[n] = \sum_l w_2[n - lR] s_l[n]. \tag{48}$$

In this case, the constant overlap add condition (Eq. (45)) becomes

$$\sum_l w_2[n - lR] w_1[n - lR] = 1 \quad \text{for all } n. \tag{49}$$

A simple example of a set of windows for which this holds is the square root Hanning window, i.e.

$$w_1[n] = w_2[n] = \sqrt{\text{hann}_L[n]} \tag{50}$$

with an overlap $R = L/2$. Note that this assumes the windows have even length L.

References

[1] S. van de Par, A. Kohlrausch, R. Heusdens, J. Jensen, and S. H. Jensen, "A Perceptual Model for Sinusoidal Audio Coding Based on Spectral Integration," *EURASIP Journal on Advances in Signal Processing*, vol. 2005, p. 317529, June 2005.

[2] A. Jeannerot, N. de Koeijer, P. Martínez-Nuevo, M. B. Møller, J. Dyreby, and P. Prandoni, "Increasing loudness in audio signals: A perceptually motivated approach to preserve audio quality," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1001–1005, 2022.

[3] B. Moore, *An Introduction to the Psychology of Hearing: Sixth Edition*. Leiden, The Netherlands: Brill, 2013.

[4] E. Terhardt, "Calculating virtual pitch," *Hear Res*, vol. 1, pp. 155–182, Mar. 1979.

[5] B. R. Glasberg and B. C. Moore, "Derivation of auditory filter shapes from notched-noise data," *Hearing Research*, vol. 47, no. 1, pp. 103–138, 1990.

[6] G. Charestan, R. Heusdens, and S. van de Par, "A gammatone-based psychoacoustical modeling approach for speech and audio coding," in *SAFE - ProRISC - SeSens 2001: proceedings. Semiconductor Advances for Future Electronics - Program for Research on Integrated Systems and Circuits - Semiconductor Sensor and Actuator Technology*, pp. 321–326, STW Technology Foundation, 2001.

[7] T. Painter and A. Spanias, "Perceptual coding of digital audio," *Proceedings of the IEEE*, vol. 88, no. 4, pp. 451–515, 2000.

[8] J. O. Smith, *Spectral Audio Signal Processing*. `http://ccrma.stanford.edu/~jos/sasp/`, accessed 03-03-2023. online book, 2011 edition.