The Matching Compressor Methods Description

1. Introduction

This document describes the methods used in the Matching Compressor plugin.

Before discussing the process of matching dynamic characteristics, it is necessary to briefly describe the fundamentals of compression. This is covered in the next (second) section of the article. Since there is a lot of information available online about this topic, we will not go into detail, and those who are already familiar with compression methods can skip much of this section. At the end, we will introduce the concept of extended dynamic processing, specifically a multi-knee compressor-expander. Considering multiple knees is important for more accurate matching of dynamic characteristics.

The third section is dedicated to methods for determining optimal processing parameters for matching dynamic characteristics. It begins by discussing the simplest algorithm for a compressor that does not use an envelope. Next, it covers an algorithm that takes attack and release into account, and finally, it presents a modification for multi-knee compression.

2. Fundamentals of Compression

2.1. Simplest Case

A compressor is designed to reduce the dynamic range of an audio signal. The simplest compressor can be described with the following formula:

$$d(x) = \begin{cases} G + x, x \leq T \\ G + T + \dfrac{x - T}{R}, x > T, \end{cases}$$

where:
d is the dynamic processing function of the signal (the loudness of the processed signal),
x is the loudness of the input signal,
G is the gain level,
T is the compression threshold,
R is the compression ratio.

All parameters in this section, unless otherwise specified, are measured in dB, except for the dimensionless *R*. Thus, the function *d(x)* is a piecewise linear function made up of two segments.

R can also be less than 1. In this case, the processor works similar to an expander, i.e. expands the dynamic range. However, a classic expander only processes samples with a loudness below the threshold, leaving others unchanged (only adding gain). For convenience, we will refer to dynamic signal processing as compression, regardless of the value of *R*.

To smooth the function *d(x)*, many compressors use Hermite quadratic interpolation (the so-called soft knee): in the vicinity of the break point, the original function is replaced with a quadratic function that touches both segments at points *T*-*W*/2 and *T*+*W*/2, where *W* is the knee width. In this case, the compression function *d(x)* is:

$$d(x) = \begin{cases} G + x, & x \le T - \frac{W}{2} \\ G + x + \left(\frac{1}{R} - 1\right)\frac{\left(x - T + \frac{W}{2}\right)^2}{2W}, & T - \frac{W}{2} < x < T + \frac{W}{2} \\ G + T + \frac{x - T}{R}, & x \ge T + \frac{W}{2} \end{cases}$$

This function has no breaks.

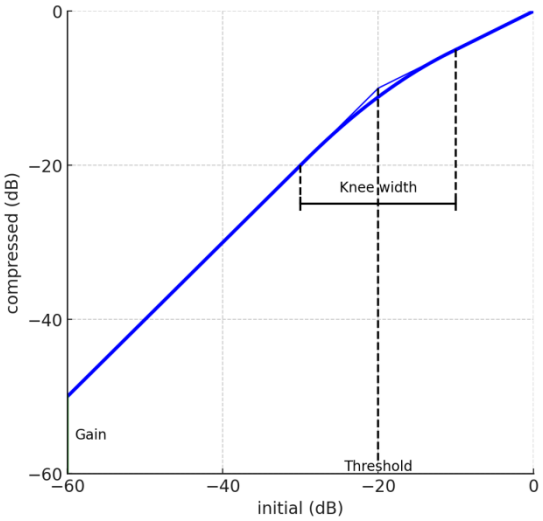

Fig.1. Soft-knee compressor

2.2. Envelope

The compression function discussed above depends on one argument: with given compression parameters *G*, *T*, *R*, *W*, the loudness of the output sample depends only on the loudness of the input sample at the same moment. However, in practice, this type of compression does not yield satisfactory results: overload effects and other artifacts can occur, making the sound muddy.

To restore natural sound, the envelope is used to calculate the compression level. The envelope is obtained by pre-filtering that adds some "inertia" to the signal, smoothing out rapid changes in level. The general form of the compression function can be expressed as:

$$d(x, x_f) = G + x + d_e(x_e).$$

As seen from this formula, the level of the input signal *x* increases by the value d_e, which depends only on the envelope x_e, after which the resulting loudness is increased or decreased by a constant value *G*. The formula discussed above can be rewritten as:

$$d(x, x_f) = \begin{cases} G + x, & x_e \le T - \frac{W}{2} \\ G + x + \left(\frac{1}{R} - 1\right)\frac{\left(x_e - T + \frac{W}{2}\right)^2}{2W}, & T - \frac{W}{2} < x_e < T + \frac{W}{2} \\ G + x + \left(\frac{1}{R} - 1\right)(x_e - T), & x_e \ge T + \frac{W}{2} \end{cases}$$

The envelope is usually obtained by applying EMA filtering to the input signal. A classic EMA filter can be represented by the following formula:

$$x_{f\,n} = \alpha x_n + (1 - \alpha)x_{f\,n-1}, \qquad 0 < \alpha \le 1.$$

The application of filtering in compression has the following features:

1. The specified filter is applied to the signal expressed not in dB, as represented in the above formulas for compression, but in relative power. The formula for converting it to decibels is:

$$x_{(dB)} = 20\log_{10}|x_{(P)}|$$

2. The envelope represents the smoothed loudness - therefore, it is always non-negative regardless of the sign of the input sample. Depending on the type of envelope - peak or RMS - it is represented by one of the following formulas:

- for peak: $x_{e\,n} = \alpha|x_n| + (1 - \alpha)x_{e\,n-1}$;
- for RMS: $x_{e\,n} = \sqrt{\alpha x_n^2 + (1 - \alpha)x_{e\,n-1}^2}$.

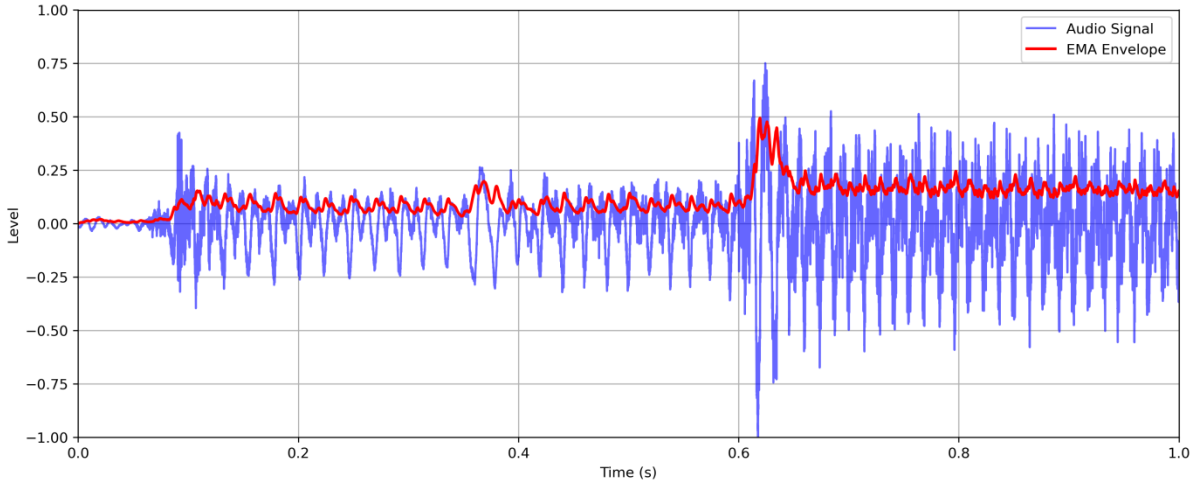

Fig.2. Audio signal and its envelope

3. Often, a switchable smoothing coefficient is used during filtering:

$$\alpha = \begin{cases} \alpha_a(a), x_n \ge x_{e\,n-1} \\ \alpha_r(r), x_n < x_{e\,n-1}. \end{cases}$$

The coefficients α_a, α_r are calculated based on the parameters a (attack) and r (release), which are usually specified in milliseconds.

Typically, the attack a is described as the time it takes for the compressor to fully engage (apply suppression) after the input signal level exceeds the threshold T, while the release r is the time it takes for the compressor to stop compressing the signal and return to the normal level after the signal level drops below the threshold. However, the term "fully" in these definitions is an oversimplification, as after applying the EMA filter, the compressor approaches the target level only asymptotically. A more precise definition depends on the nature of the dependencies $\alpha_a(a)$, $\alpha_r(r)$. In this implementation, the following formulas are used:

$$\alpha_a = e^{-\frac{2\pi}{af_s}},$$

$$\alpha_r = e^{-\frac{2\pi}{rf_s}},$$

where f_s - is the sampling frequency of the audio data (samples per second). In this case, the attack time is the duration to reach about 99.8% of the target level (the release is defined similarly).

Thus, with unchanged compression parameters G, T, R, W, the average loudness of the processed signal increases with a longer attack time and decreases with a longer release time.

2.3. Processing Multi-Channel Audio

When processing multi-channel signals, there are three possible approaches:

- each channel is processed separately (each channel uses its own mono-compressor with the same parameters);
- the compression level is determined by the maximum smoothed loudness among all channels;
- the compression level is determined by the average smoothed loudness.

2.4. Multi-knee Compressor

To more accurately match two audio signals, which will be discussed later, it is desirable to have a more complex dynamic curve. This can be achieved by considering a multi-knee compressor. The parameters of such a compressor include overall gain, attack, release, the type of multi-channel signal processing (separate, maximum or average), and the type of envelope (peak or RMS). Additionally, each knee has its own threshold, compression ratio, and knee width.

Let the number of knees be N_L.

If $x_e \leq T_0 - 0.5W_0$, then no compression is applied, and $y = x + G$.

Otherwise, to determine the output loudness, we first need to find which knee the current envelope value belongs to: the knee index i is defined as the highest index such that $x_e > T_i - 0.5W_i$.

Next, the auxiliary value G_i is calculated:

$$G_i = \begin{cases} G, & i = 0 \\ G_{i-1} + \left(\frac{1}{R_{i-1}} - 1\right)(T_i - T_{i-1}), & i > 0 \end{cases}$$

The output loudness of the signal is calculated as:

$$d(x, x_e) = \begin{cases} G_i + x + \left(\frac{1}{R_i} - \frac{1}{R_{i-1}}\right)\frac{\left(x_e - T_i + \frac{W_i}{2}\right)^2}{2W_i}, & x_e < T_i + \frac{W_i}{2} \\ G_i + x + \left(\frac{1}{R_i} - 1\right)(x_e - T_i), & x_e \geq T_i + \frac{W_i}{2} \end{cases}$$

This structure allows for more precise control over the dynamic characteristics of the audio signals being processed.

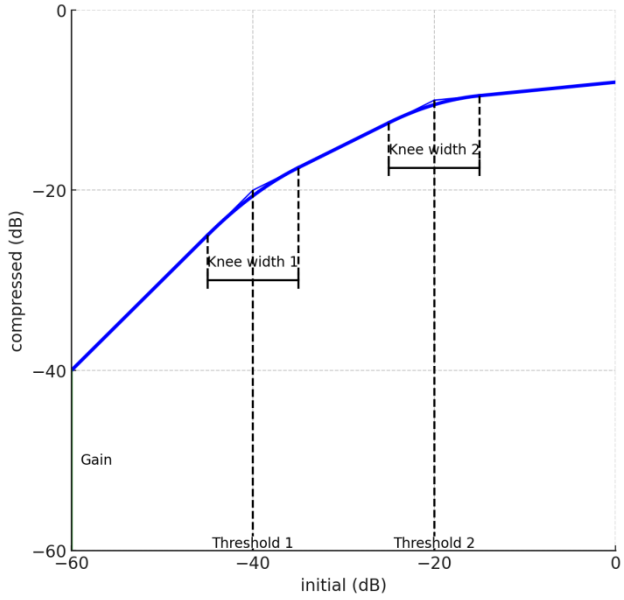

Fig.3. 2-knees compressor

3. Matching Dynamic Characteristics

Having reviewed the formulas used for signal compression, we now turn to the main task of this article: calculating the compression parameters that provide the best matching of dynamic characteristics between two audio fragments.

3.1. The Simplest Case

First, let's consider the following scenario:

- the compressor consists of a single knee;
- the attack and release times are set to zero;
- each channel (in the case of multi-channel audio) is processed separately.

In this case, at any moment, the output of the compressor, given its known parameters, depends only on its input at that same moment: $d = d(x)$.

For each of the two audio fragments—the input and the reference—we can define a probability distribution function $F(x)$, where x represents the loudness level. For each level x, the value $F(x)$ reflects the proportion of samples with a loudness not exceeding x. In other words, it is the probability that a random sample from the given fragment will have a loudness no greater than x. The function $F(x)$ is non-decreasing, with values ranging from 0 to 1. The task is to find the compressor parameters such that the resulting function of the processed audio, F_{comp}, closely matches the reference function F_{ref}.

The first question to address is: in what units should the signal level be measured?

Comparing values in decibels is not appropriate, as illustrated by the following example: suppose we need to compare two pairs of samples presented in the table below.

	I	II
x_{ref}	-100 dB	-1 dB
x_{comp}	-200 dB	-13 dB
Δx	100 dB	12 dB

From the table, we can conclude that the difference between -100 and -200 dB is much more significant. In reality, however, it is negligible because the loudness values themselves are extremely low. On the other hand, the difference between -1 and -13 dB (i.e., a factor of 4) is quite significant, as the loudness values are relatively high.

Therefore, levels should be compared in relative power—meaning we should use the units in which the information is stored in the audio file. In this case, the table would look like this:

	I	II
x_{ref}	10^{-5}	0.89
x_{comp}	10^{-10}	0.22
Δx	$\sim 10^{-5}$	0.67

Additionally, using decibels would require dealing with an infinite range of values from $-\infty$ to 0, while relative power can take values from 0 to 1.

The second question is: how do we determine the degree of similarity between two distribution functions?

The experiment shows that it is better to compare not the values of $F(x)$ for the average levels x of different intervals, but rather the values of quantiles x_q. These are the values at which the proportion of samples in the audio fragment that do not exceed x_q equals q: $F(x_q) = q$. In other words, this approach considers not the function F itself, but its inverse, F^{-1}.

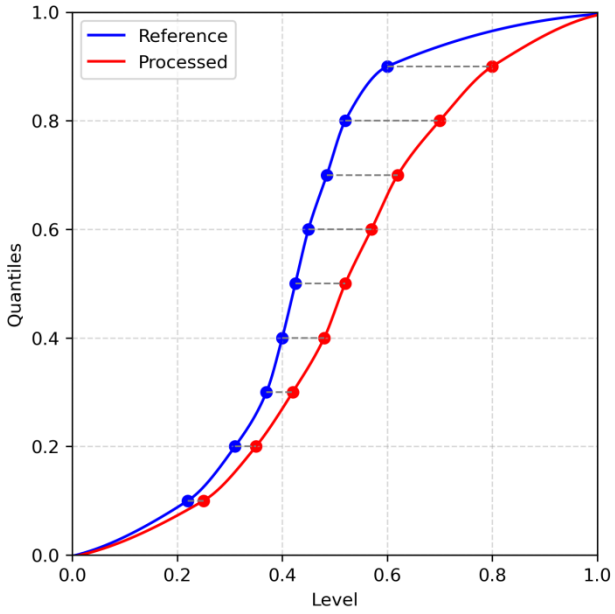

Fig.4. Distribution matching

Let the number of points needed to discretize the function be N_q. Then the values q_i are calculated as:

$$q_i = \frac{i+1}{N_q+1}, 0 \le i \le N_q - 1.$$

For example, if we need to discretize the function into four points, these will be the loudness values that do not exceed 20%, 40%, 60%, and 80% of the samples.

To determine the quantiles for a given audio fragment, we need to:

- combine all audio channels into one, creating an array $\boldsymbol{a}_s$ from $N_s \cdot N_c$ values (N_s is the number of samples, N_c is the number of channels), storing not the actual values but their magnitudes (the loudness of a sample does not depend on its sign);
- sort the resulting array in ascending order;
- determine the indices of the desired elements in the array as follows:

$$k_i = \frac{N_s N_c (i+1)}{N_q+1}, 0 \le i \le N_q - 1.$$

If k_i is an integer, the desired value will be the k_i-th element of the array; otherwise, linear interpolation can be applied between the two neighboring elements.

From the description of this algorithm, it is clear that its time complexity is determined by the time complexity of the sorting operation, which is $N \cdot \log(N)$, where $N = N_s \cdot N_c$. It is possible to speed up the calculations by replacing the exact F^{-1} computation (if we disregard the error from linear interpolation between neighboring elements) with an approximate one. In this case, we need to:

- combine all audio channels into one array $\boldsymbol{a}_s$, just like in the previous algorithm;
- divide the range of relative power from 0 to 1 into N_G equal sub-ranges (discretization by loudness) and, for each of the sub-ranges, count the number of elements in the array that fall within that sub-range; fill the array $\boldsymbol{f}$ with these values ($\boldsymbol{f}$ represents the probability density function values multiplied by N);
- knowing the probability density function, calculate the probability distribution function multiplied by N: $F_0=f_0$, $F_i = F_{i-1}+f_i$;
- for each j from 0 to N_G - 1, the quantile value $\frac{F_i}{N_s N_c}$ will be approximately equal to $\frac{j+1}{N_G+1}$ (the midpoint of the corresponding loudness range);
- to discretize the quantile function into uniform values $\frac{j+1}{N_q+1}$, apply linear interpolation.

The time complexity of this algorithm is $O(N)$. At the same time, it is possible to choose a sufficiently large N_G such that the accuracy does not significantly decrease.

By applying the described algorithm to the processed and reference audio fragments, we obtain two non-decreasing data arrays $\boldsymbol{x}_q$ and $\boldsymbol{y}_q$ of length N_q with values ranging from 0 to 1. These two arrays serve as the input data for the task of determining the compression parameters.

Next, we need to solve the problem of finding the minimum sum of squared differences between the values of the reference array and the input array after being processed by the compressor with given parameters:

$$\sum_{i=0}^{N_q-1} \left(F_{comp}^{-1}(i) - F_{ref}^{-1}(i)\right)^2 \rightarrow \min$$

where F^{-1} - is the inverse function of the probability distribution function, with the subscript "*ref*" indicating reference values and the subscript "*comp*" indicating values after the compression of the input audio.

But $F_{ref}^{-1}(i) = y_{q\,i}$. Also, in the problem at hand, the output loudness at given compression parameters depends only on the input level at the same moment in time for the same channel, we have $F_{comp}^{-1}(i) \approx d\left(\boldsymbol{c}, x_{q\,i}\right)$. Taking this into account, the problem can be reformulated as:

$$\sum_{i=0}^{N_q-1} \left(d\left(\boldsymbol{c}, x_{q\,i}\right) - y_{q\,i}\right)^2 \rightarrow \min.$$

The compressor parameters $\boldsymbol{c} = \{G, T, R, W\}$ that achieve the minimum are the solution. The elements of the array $\boldsymbol{c}$ are constrained by the minimum and maximum values of the corresponding parameter:

$$c_{\,j\,min} \leq c_{\,j} \leq c_{\,j\,max}.$$

Thus, the task is to approximate a known set of points ($x_{q\,i}$, $y_{q\,i}$) with a function of a known type. This is a standard nonlinear regression problem that can be solved using various libraries. For example, the AlgLib library provides the capability to solve it using the Levenberg-Marquardt method.

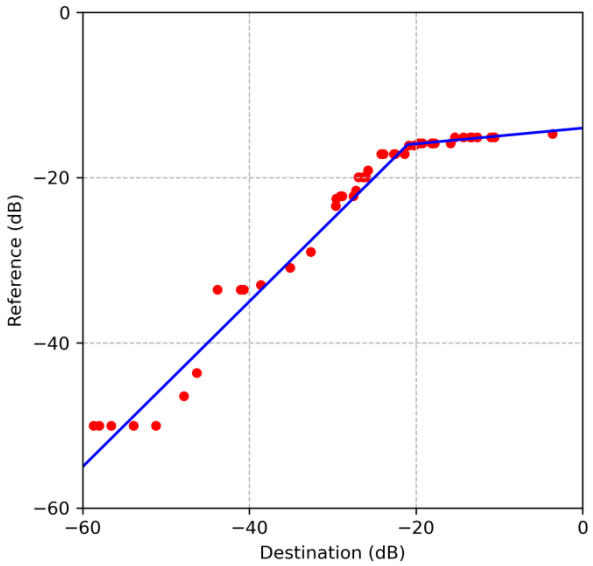

Fig.5. Regression problem for 1-knee mono compressor without envelope

It should be noted that the relationship between output loudness and input loudness at given parameter values is analytical. Therefore, for each set of parameters considered, it is not necessary to fully process the entire audio fragment, which significantly speeds up the solution. Instead, at each step of the algorithm, the output loudness values for N_q points are calculated using known analytical formulas for compression.

Additionally, at each step of the algorithm, it is necessary to compute the gradient—the vector of partial derivatives of the loudness function with respect to each of the parameters:

$$\mathbf{grad} = \left\{\frac{\partial F}{\partial g}, \frac{\partial F}{\partial t}, \frac{\partial F}{\partial r}, \frac{\partial F}{\partial w}\right\}.$$

Since the compression formula $d(\boldsymbol{c},\ x_{q\ i})$ is known, the gradient can be easily computed analytically. Experiments show that for audio fragments lasting about a minute, the described algorithm takes less than a second. If further acceleration is needed, it is also possible to compute the second derivatives (Hessian) analytically.

3.2. Matching with Consideration of the Envelope

Since the result of compression without using an envelope will be unsatisfactory, in real audio processing tasks, the attack and release times will be non-zero. If we calculate the compressor parameters that best match the processed audio with the reference audio based on their zero values and then increase them, the resulting loudness distribution will deviate from the reference. To avoid this, we need to reformulate the task: to determine the compressor parameters G, T, R, W such that after processing, the loudness distribution in the processed

audio is as close as possible to the reference, given the specified values for attack, release, envelope type, and multi-channel audio processing type.

For the reference audio, the calculation of the inverse probability distribution function is done in the same way as described above. However, for the processed audio, the resulting loudness function is now a function of two arguments: the original loudness and the envelope value. In this case, it is no longer possible to take N_q original loudness values of the processed signal and compute the output values. A similar situation arises when processing multi-channel audio, even with zero attack and release, if the compression level is determined not for each channel separately but based on the maximum or average loudness among all channels.

In this case, the task of finding the optimal compressor parameters is formulated as follows:

$$\sum_{i=0}^{N_q-1} \left(F_{comp}^{-1}(i, \boldsymbol{c}, \boldsymbol{x}, \boldsymbol{x_e}) - y_{q\,i}\right)^2 \rightarrow \min$$

The entire audio fragment must be processed at each step to compute F_{comp}^{-1}. Moreover, the gradient can only be computed using numerical methods by evaluating the function values in the vicinity of the point being considered. This significantly reduces the speed of the algorithm.

To speed up the process while allowing for a permissible decrease in the accuracy of computing F_{comp}^{-1}, we can collect statistics on the input audio into a matrix **M** of size $N_G \times N_G$ before starting the iterations. Each value m_{ij} in this matrix should reflect the proportion of samples whose levels fall within the range $\frac{i}{N_G} \le x < \frac{i+1}{N_G}$, while the corresponding envelope level at the same moment in time falls within the range $\frac{j}{N_G} \le x_e < \frac{j+1}{N_G}$.

Then, at each step for computing F_{comp}^{-1}, we process this matrix instead of the audio fragment itself. An array $\boldsymbol{a_G}$ is created in the same way as when processing the reference audio. For each non-zero m_{ij}, the value after compression $d\left(\boldsymbol{c}, \frac{i+0.5}{N_G}, \frac{j+0.5}{N_G}\right)$. is calculated. Based on this value, the index of the element in the array $\boldsymbol{a_G}$ that needs to be increased is determined. This element is increased by the value m_{ij}. After filling $\boldsymbol{a_G}$, the computation of F_{comp}^{-1} is performed in the same way as for the reference audio.

Thus, each step of minimization requires $O(N_G^2)$ time, assuming that $N_q < N_G$. This is justified if $N_G^2 \ll N_s N_c$. At the same time, N_G should be sufficiently large to maintain acceptable accuracy. However, when N_G^2 is comparable to $N_s N_c$, it is more practical to process the entire audio fragment sample by sample.

Also, to optimize the process, it is reasonable to memorize of already computed arrays F_{comp}^{-1}.

3.3. Matching in the Case of a Multi-Knee Compressor

In the case where the compressor consists of N_L knees, the vector of computed compression parameters is represented as:

$$\boldsymbol{c} = \{G, T_1, R_1, W_1, \dots, T_{N_L}, R_{N_L}, W_{N_L}\}$$

The calculation time increases due to the higher dimensionality of the gradient (its dimensionality equals that of $\boldsymbol{c}$) - especially if it is calculated numerically. Additionally, there is an added complexity related to the constraints on the compression parameter. The minimization problem formulation assumes constraints on each parameter in the form of its minimum and maximum values, which are constants. However, in the case of multiple knees, it is necessary to ensure additional constraints such as:

$$T_{i-1} + 0.5W_{i-1} \leq T_i - 0.5W_i$$

where the threshold and knee width are expressed in dB.

To address this issue, a penalty technique can be utilized. The minimization problem can then be formulated as:

$$\sum_{i=0}^{N_q-1} \left(F_{comp}^{-1}(i, \boldsymbol{c}, \boldsymbol{x}, \boldsymbol{x_e}) - y_{q\,i} + s(\boldsymbol{c})\right)^2 \rightarrow \min$$

The penalty function *s* should be chosen so that it takes a value of zero if all knees of the compressor satisfy the required condition, and a sufficiently large value if at least one condition is violated. This way, the minimized functional will take on a value that will definitely not be selected as optimal.

When selecting the function *s*, it is important to ensure the continuity and smoothness of the minimized functional. For example, an instantaneous increase in the penalty from zero to a certain value is unacceptable. However, if a gradual increase in *s* is ensured, there is a risk during minimization of stopping at parameters that violate the specified constraint, even if not significantly enough to greatly affect the minimized functional. This possibility must be eliminated. To do this, the original constraint can be replaced with the following:

$$T_{i-1} - T_i + 0.5(W_{i-1} + W_i) + \delta \leq 0$$

A function that meets the above requirements can be defined as:

$$s(\boldsymbol{c}) = \sum_{i=2}^{N_L} k_s\left(\max\left(0, T_{i-1} - T_i + 0.5(W_{i-1} + W_i) + \delta\right)\right)^2$$

The coefficient k_s and the margin δ should be chosen based on a compromise such that:

- any violation of the original constraint results in a sufficiently large penalty to ensure rejection of that parameter set during minimization;
- δ is not too large to maintain the possibility of selecting acceptable parameters;
- the increase in penalty when constraints are violated is not too abrupt.

Based on the types of the penalty function and the minimized functional, a condition for selecting k_s and δ can be derived: $k_s \geq \delta^{-2}$. As both k_s and δ should be minimized, the optimal choice would be $k_s = \delta^{-2}$.

The following figure shows the penalty function's dependence on the threshold of the second knee T_2 for a 2-knee compressor, as depicted in Fig.3. Here, T_1=-40 dB, W_1=W_2=10 dB, δ=0.5 dB, and k_s=4.

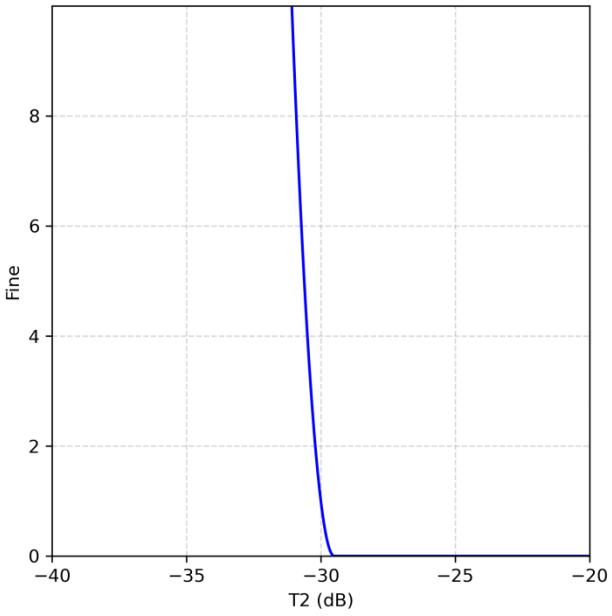

Fig.6. Penalty function

The described penalty technique can be used to solve problems both with and without envelope.

3.4. Parameter Selection

It should be noted that the discussed regression problem is nonlinear. Therefore, it is impossible to guarantee convergence to a global minimum. As a result, the following factors become very important:

- the parameters specified by the user;
- the initial values for the compression parameters (initialization);
- the tuning parameters for the minimization algorithm.

Regarding the user-specified parameters, it can be said that as the number of unknowns increases, both the risk of missing the global minimum and the computation time also increase. Therefore:

- solving the problem with a large number of knees does not always yield a more accurate result;

- in most cases, it is advisable to solve the problem with a hard knee (with zero knee width) and then manually set the desired width: this will result in only minor changes to the loudness distribution while speeding up the solution.

For initial values of the compression parameters, one can suggest parameters where compression does not alter the data: $G = 0$, $R_i = 1$, $W_i = 0.5W_{\max}$ (in the case of a hard knee, $W_{\max} = 0$), $T_i=T_{\min}\cdot i/(N_L+1)$ ($i = 1...N_L$). However, another choice is also possible, depending on the loudness distribution in the input and reference audio.

The parameters of the minimization algorithm (which should either be set programmatically or provided for user selection) depend on the mathematical method used and its implementation. Below is a sample list:

- penalty coefficient k_s;
- minimum non-penalized range between two knees δ;
- maximum number of iterations;
- maximum step size at which the solution is considered found;
- step size for numerical derivative calculation;
- number of ranges for loudness discretization;
- number of values (quantiles) used to discretize the minimized function.

4. Conclusion

Finally, a few concluding remarks. The processed audio will sound more similar to the reference if their amplitude-frequency characteristics are also similar. If this is not the case, one might consider using an equalizer before this dynamic processor (if, of course, it aligns with the processing goals). Plugins that allow for adjusting the amplitude-frequency response of the processed audio to match the reference can be helpful in this regard. And, of course, when processing musical material, it is advisable to choose a fragment in the same musical style as the reference.

The source code of the Matching Compressor can be found at https://github.com/justonem0reuser/MatchingCompressor.