SGN-14006 – Audio & Speech Processing

Project Work Instructions

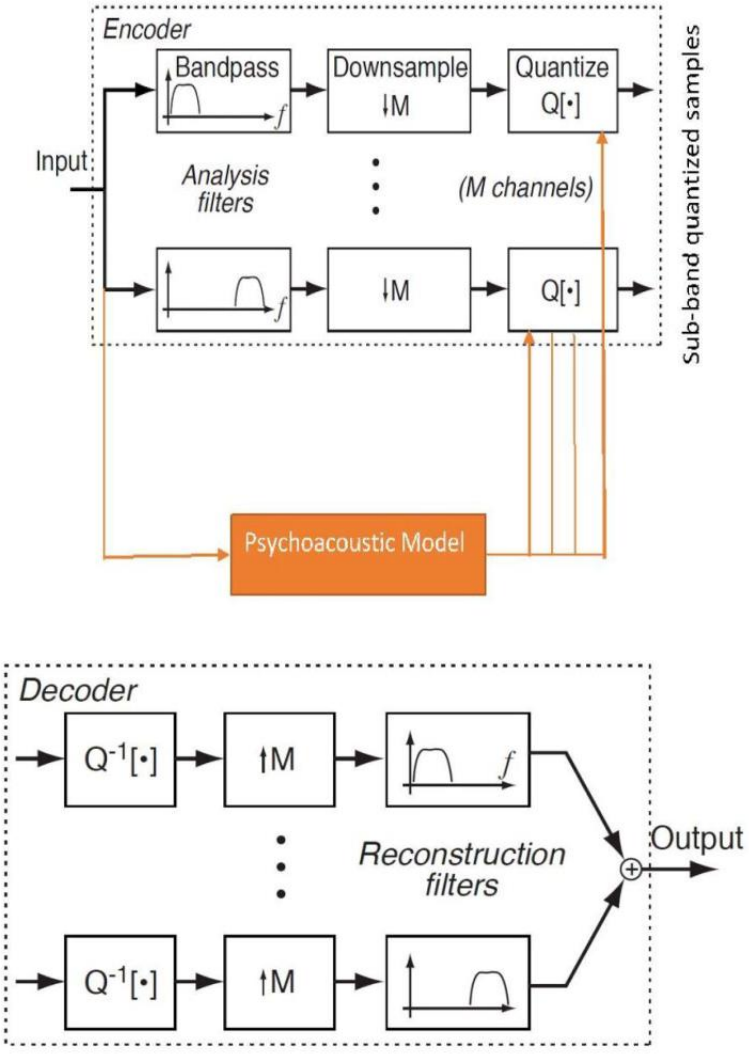

Fig 1. The block diagram highlights the psychoacoustic model that controls the quantization, which needs to be implemented along with the analysis and synthesis parts.

1 Description of project

1.1 Theory

A lossy audio coder is implemented using python, by exploiting a uniform filterbank with critical sampling and perfect reconstruction along with a simplified psychoacoustic model to produce the masking threshold for each time-frequency point.

1.2 Model specifications

1.2.1 Analysis-Synthesis filterbank

Use an M-band MDCT-based uniform filter-bank to obtain a critically sampled, perfect reconstruction analysis-synthesis filterbank. Perform the sub-band filtering in time-domain to obtain the subband signals. (see exercise 2 for instructions on how to generate the analysis-synthesis filterbank). Test the model if it can perform perfect reconstruction with M number of bands (without quantization).

1.2.2 Psychoacoustic model

The psychoacoustic model is implemented separately from the analysis-synthesis filterbank. It uses discrete Fourier transform (DFT) to evaluate the magnitude spectrum of the input signal in short time frames. The length of the time frame should be a power of two, so that the temporal duration is approximately 20 ms at the given sampling frequency. This depends on your sampling frequency, typical values are 512, 1024, ..., 4096. (convert 20ms to samples, take log2, round the result upward)

It is most convenient to first create the critically sampled sub-band signals, then loop the data once over by windowing using short time frames and taking the DFT of the windowed frames to evaluate the masking threshold for each frame. Use the corresponding time-indices of the sub-bands for the current time-window of the psychoacoustic model.

```
# We now have the sub-band signals for M bands, given by the analysis

NDFT = None
si = 0
ei = NDFT #start and end indices of input signal for each frame
win = 0

#Main Loop
while ei < len(x_input_signal):
    time_index_fullband = np.arange(si, ei)

    # Subband time indices at 1/M sampling rate (for reach band).
    # these sample indices need to be then quantized in this window

    time_index_subband = np.arange(si//M, ei//M )

    # The most important part: SPL, Masking threshold, SMR and
    # quantization here for samples, and to count the number of bits
    # used, e.g. sum up how many sub-band samples were quantized with
    # nbits over all sub-bands and all processing frames

    si = si + NDFT # advance 1 full frame in input data (no overlap)
    ei = si + NDFT
    win += 1 # keep track of number of processed windows / frames
             # indices of datapoints of full-band original signal
             # for the psychoacoustic model:


# Proceed to reconstruct the signal from the quantized sub-band signals
```

Obtaining normalized Sound Pressure Level (SPL) from amplitude spectrum:

If `X_dft[:, win]` is the DFT of the Hanning windowed frame (indexed as `win`) of the original signal, SPL can be obtained as:

```
SPL = 96 + 20*np.log10(np.abs(X_dft[:NDFT//2, win] / maxlevel1k))
```

Tip: Normalized SPL is found from the `DFT` of the original data, not from the `DFT` of the band-filtered and downsampled data! To obtain `maxlevel1k`, create a 1kHz sinusoid of length `NDFT` samples (use the same sampling rate with the audio signal) and compute the maximum absolute value of its DFT (which will now be `maxlevel1k`).

Do not forget to Hanning window the 1 kHz sinusoid too.

Detour: *For the eager minds, what is happening above? We need to compute the absolute (normalized) SPL of a signal that depends on gain settings used on the playback (for example, higher volume settings lead to higher SPLs), which are not known a priori. So, we need to make assumptions about the target playback gain for the signal. Here we assume that the input PCM data has been recorded and quantized so that quantization error falls near the bottom of the hearing threshold at normal playback levels. In particular, we define a sinusoid with an amplitude equal to ½ the PCM quantizer spacing as having an SPL of 0 dB.*

*For 16-bit PCM data, the standard assumption implies that a sinusoid with amplitude equal to overload level of the quantizer would have an SPL of about 96 dB (6 dB/bit * 16 bits).*

Self-quiz: *If we define our quantizer overload level x_{max} to be equal to 1, what is the SPL of a sinusoidal input with an amplitude A?*

Note: The explanation above has been reproduced from [1]

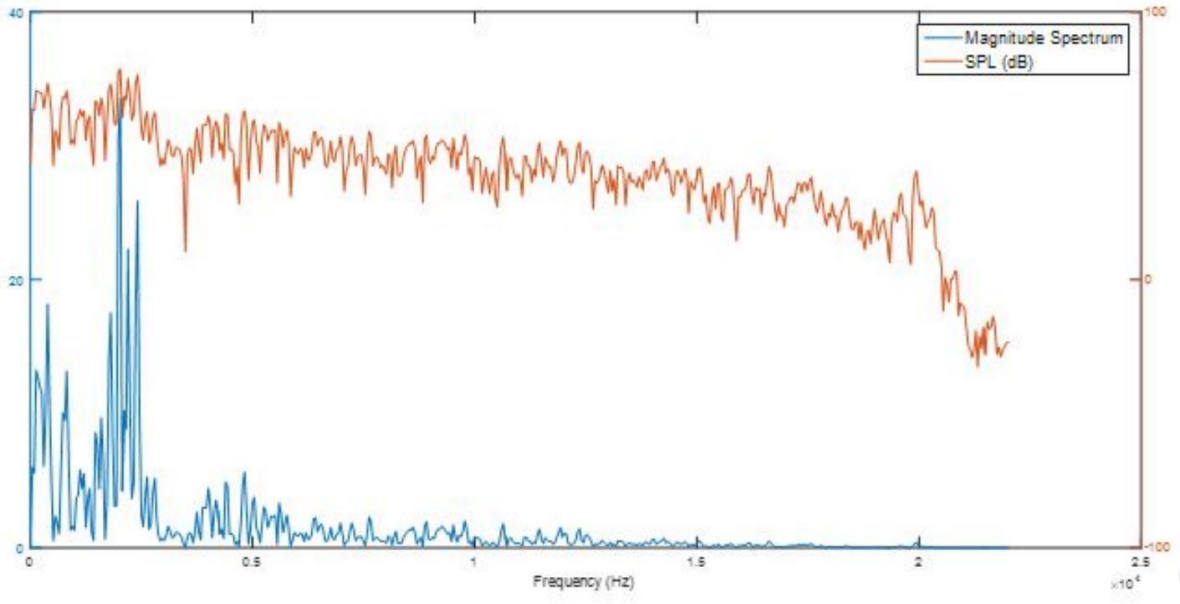

Fig 2. Spectrum and SPL of a windowed frame

Obtain maximum SPL for each sub-band:

Create vectors to index M different sub-bands from DFT output. For every band ("of every frame" is already implied) search for the maximum SPL value within each band as:

```
for bandIndex in range(M):
    SPL_band[bandIndex, win] = \
                    np.max(SPL[dftIndices[:,bandIndex]])
```

`dftIndices[:,bandIndex]` is a vector containing the DFT indices of the band indexed by integer value `bandIndex`. For eg., if you have M=64 bands, and use NDFT=1024 samples for the psychoacoustic model, you need to index DFT output values as 1, ..., 512 (corresponding to real part of the spectrum) for each of the 64 bands. For example: `dftIndices[:,0]= np.arange(0,7)` and `dftIndices[:,1] = np.arange(8,15)`, etc… (`numpy arange, numpy reshape`).

Look at the threshold in quiet line from lecture slides: L03-hearing.pdf, page 14 (the lowest curve).

Now, generate the threshold in quiet for each of the M sub-bands. For band center frequency f [kHz] the threshold in quiet is (in dB): [2]

```
tiq = 3.64 * f**(-0.8) - 6.5*np.exp(-0.6*(f-3.3)**2)+(10**-3)*f**4
```

Self-quiz: *Why is the threshold of quiet high at low/high frequencies? Are these frequencies more sensitive or less sensitive to sound pressure? How should your perceptual coder assign bits, high or low, to these frequencies in general for e.g. music signals?*

Detour: *For the eager-minds, the above strange looking expression is obtained by experimentation. Refer to [1] (pg 152-156) for in-depth details.*

Computing masking threshold from the SPL levels:

- Convolve bandwise SPL between adjacent bands to simulate the effect of spread
    - Note that we omit the effect of amplitude and frequency to the spread of the mask. In general, the masking is more widespread towards higher frequencies, and more widespread if the signal is louder.
- Subtract MASK_dB dB from the convolved SPL. Merge the obtained mask level with the threshold in quiet.
- Merge the obtained mask level with the threshold in quiet.

```
TT = convolve(SPL_band[:,win], [.05, .5, .4, .05 ], 'valid')
TT = np.append(TT, np.full((3,1),TT[-1]))
maskThr_current = np.maximum(???, ???)
```

Here `MASK_dB` = 16 dB.

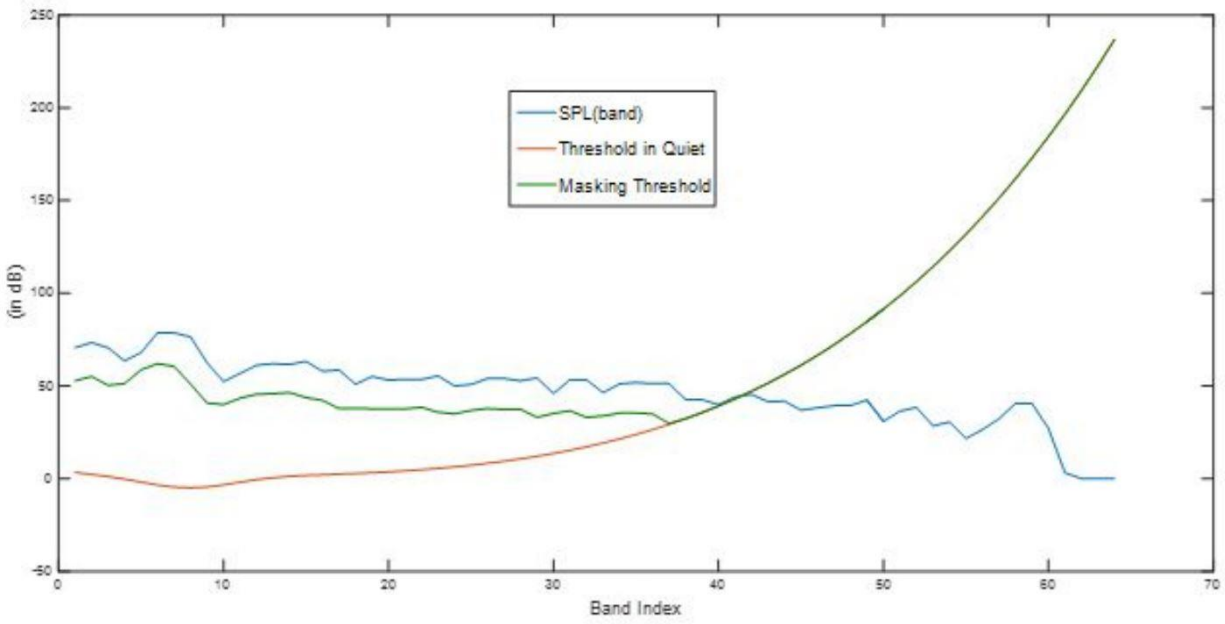

Fig 3. Masking Threshold and Threshold in Quiet for the same windowed frame as in fig 2.

Threshold with Temporal Masking consideration

To simulate temporal masking, e-g- the spread of auditory masking from previous time-frames, use the maximum among the threshold of the current band or a decay constant (0.9) times the threshold of the previous window.

```
th_mask = np.maximum( th_mask*0.9 ,maskThr_current )
```

(Note difference between np.max and np.maximum)

Figure 4 illustrates these two masks and their combination

Self-quiz: *Let us say that the signal contains drums and then silence. Which of the masking thresholds will be stronger after the drum hit: the current frame's masking threshold or the masking threshold from the previous frame with the drum?*

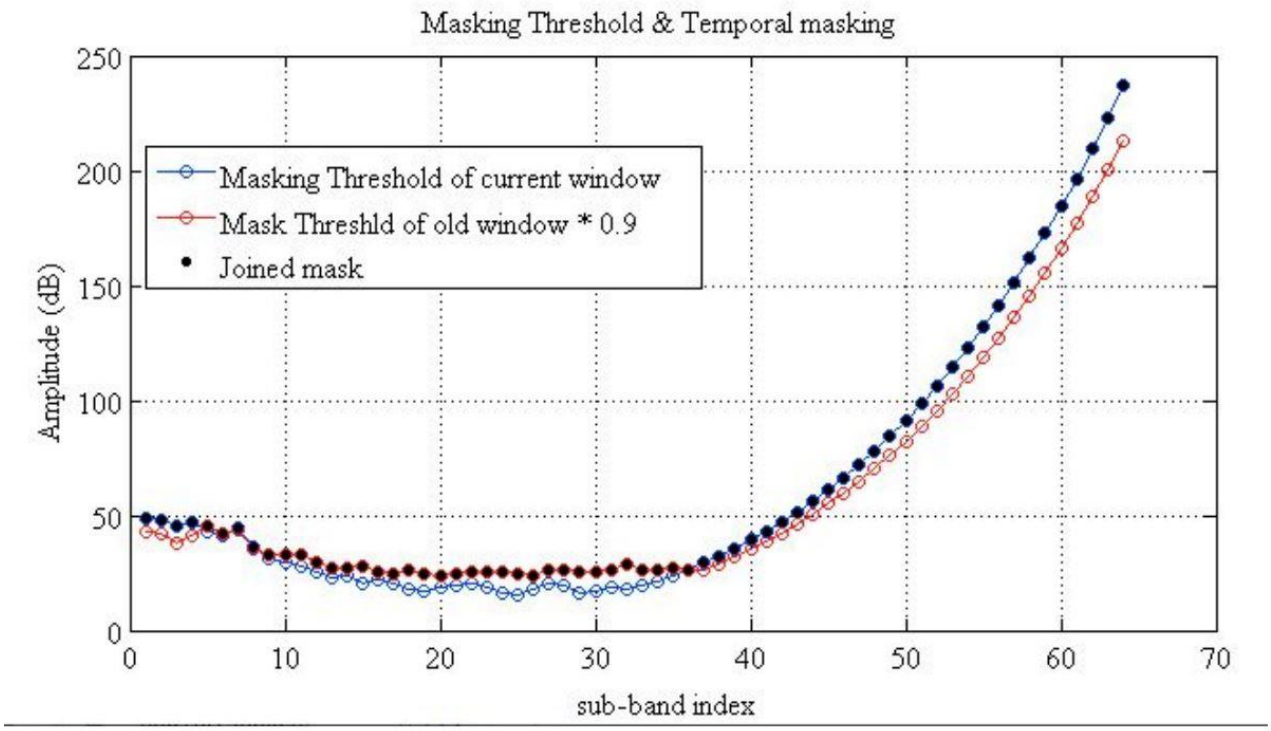

Fig 4. Masking threshold of current frame only (blue), masking threshold of previous frame multiplied with a decay constant (red line), and joined mask (black dots, the maximum of these two). Note that there is slight temporal masking happening in this frame at bands (8-35).

Computing Signal to Mask Ratio (SMR)

Now we have the masking threshold obtained, and now compute the SMR for every windowed frame by a very simple method of subtracting the masked threshold (`th_mask`) from the SPL value of the band (`SPL_band`).

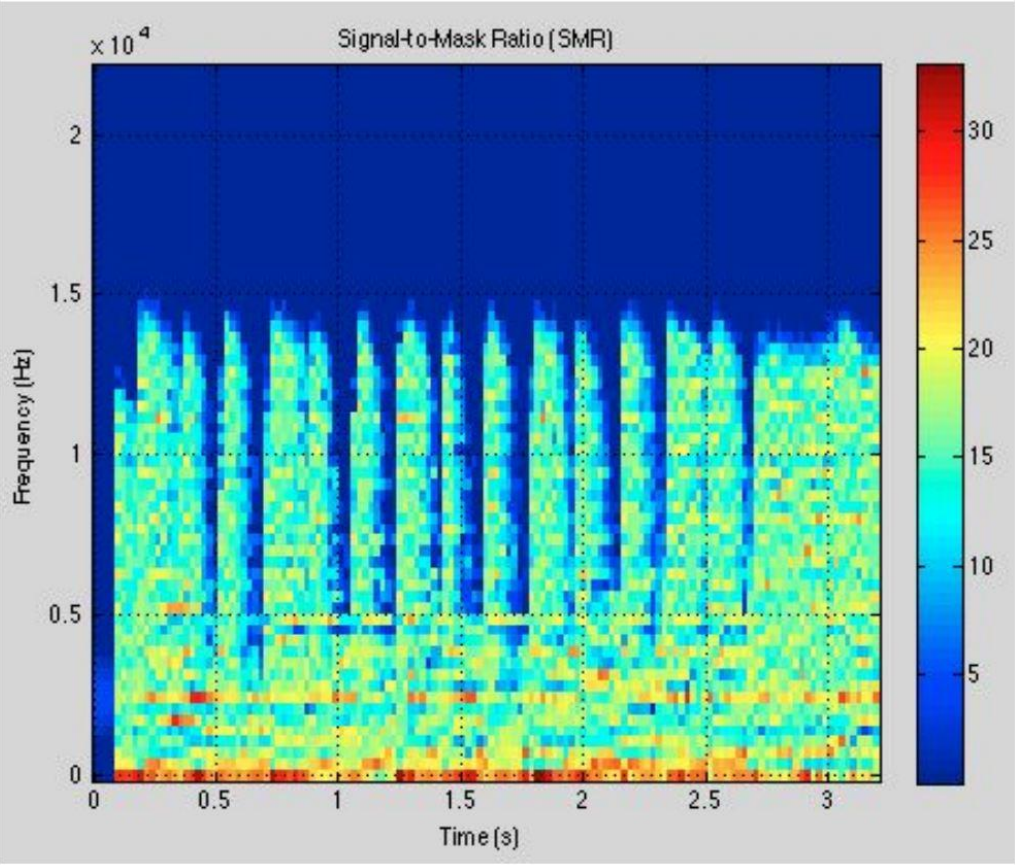

Fig 5. Sample Signal-to-Mask ratio plot from a 3.5s input signal

Bit allocation

Having obtained the SMR in dB, we now allocate bits to each sub-band. Remember that SMR tells us how much masking capacity the band has. Divide the SMR of each sub-band by 6.02dB/bits to convert the SMR values into required number of bits. Make sure you properly round the results and that the number of bits allocated is not negative (negative SMR values correspond to zero bits allocated). Figure 6 illustrates this. Observe that no bits are assigned for bands over 39. Keep a count of the number of bits you have assigned for each sub-band in each frame.

```
matrix_of_bits[ind_subband,win] = bits_for_subband*samples_in_subband;
```

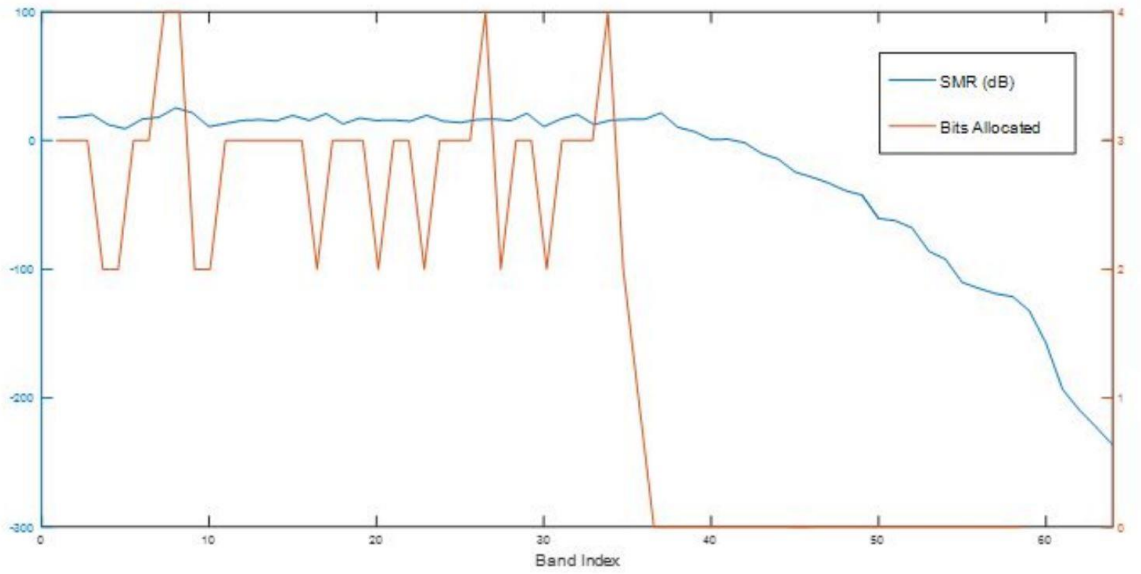

Fig 6. Bits allocated to each sub-band.

1.2.3. Quantization

Quantize the sub-band time-domain signals with a quantization function that

1. Normalizes the samples to be quantized with the absolute maximum value
2. Quantizes the signal using uniform quantization with Y bits (i.e. 2**Y levels, with Y=4, you should have possible values of -1.0000 -0.8667 -0.7333 -0.6000 -0.4667 -0.3333 -0.2000 -0.0667 0.0667 0.2000 0.3333 0.4667 0.6000 0.7333 0.8667 1.0000

You can use the function myquantize:

```
def myquantize( x, bits ):
    scale = np.max(np.abs(x));
    x = x/scale;
    levels = np.arange(0,2**bits);
    levels = levels - np.mean(levels);
    levels = levels / np.max(levels);
    y = np.zeros([len(x)])
    for i in range(len(x)):
        ind = np.argmin(np.abs(x[i] - levels));
        y[i] = levels[ind]*scale;
    return y
```

Self-quiz: Note that the quantization process includes scaling the input values to the maximum of the quantization scale before quantization. This scaling coefficient should also be transmitted to the de-coder, but here it is directly used to scale the quantized values back without transmission. Here, we omit the transmission of scaling values. How would this transmission affect the bitrate of an actual encoder?

1.2.4. Decoder

Interpolate and (inverse) band-pass filter the sub-band signals (as done in exercise 3). Do not forget to delay the summed output by the filter group delay. Analyze the average number of bits/sample used in

your psychoacoustic encoder. (sum of the bits used in each sub-band for each frame divided by the number of samples in the signal).

2. Deliverables

Return a .py file including a function

dct_filterbank_psycho(x_input, fs, M, MASK_dB)
return values: *[y, bitrate]*

where ***x_input*** is the input signal, ***fs*** is sampling frequency of input signal, ***M*** is the number of sub-bands, ***MASK_dB*** is the fixed masking threshold value in decibels. The output *y* consists of the reconstructed signal, ***bitrate*** is the average bitrate used [bits/sample] (sum of bits used for each window and for each sub-band divided by the length of the input signal in samples).

The .py file can include a main function using the above described model and the function in practice.

Prepare a report where you present the following:

- What problem is being solved in the project work?
- How is this problem solved?
- What assumptions were made?
- Short description of implementation. What stages does the algorithm consist of?
- What is evaluated in your experiments, and how?

In addition, analyze and compare results from different values of MASK_dB. Include visualization of the original signals spectrogram, as well as the SMR spectrogram of your model.

References:

[1] Marina Bosi & Richard.E.Goldberg, Introduction to Digital Audio Coding and Standards, Kluwer Academic Publishers.

[2] DiracDelta.co.uk, Science and Engineering Encyclopedia, http://www.diracdelta.co.uk/science/source/t/h/threshold%20of%20hearing/source.html#.Vgo2y2Qzf3w