28.11.2024 **Author: Petr Černý**

Name

Compression

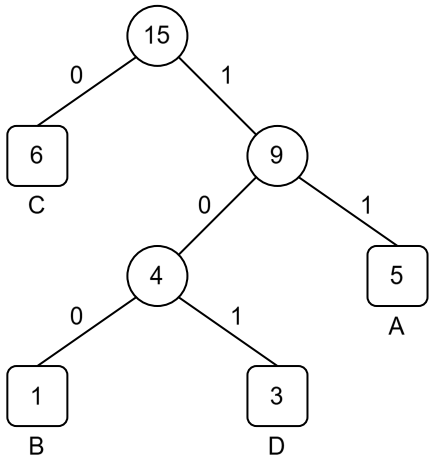
Assignment

The goal of this project was to study and use different methods of audio compression. Firstly a short description of theory used in project is described. After the project itself is described which was to compress an audio file of song.

Theory

There are two main types of compression a **Lossy** compression and **Lossless** compression. In both cases the goal is to reduce the size of a file. With the first type reducing the size by permanently deleting data that is deemed redundant. Therefore the compression cannot be reversed. The later type usually uses some kind of algorithm that is able to code the original data in a different, smaller format. Here no data is lost so the data can reconstructed perfectly.

Among the most used algorithms for lossless compression is **Huffmann coding**. The aim of this algorithm is to code the original data into a bit stream of varying lengths. A symbol that is most frequently used in the original data is coded into a bit stream with the shortest length. The algorithm firstly defines unique symbols and their frequency from which a binary tree is created.



Obrázek 1: Huffmann coding - Binary tree

A	B	C	D
11	100	0	101

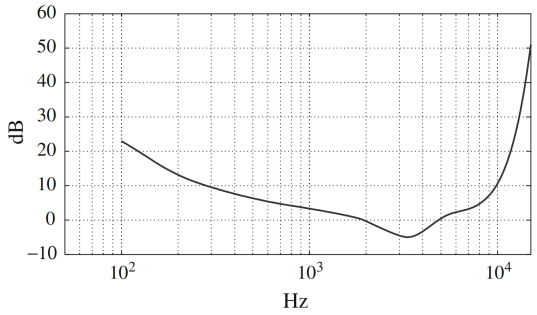
Obrázek 2: Huffmann coding - Dictionary

The figure 1 depicts an example of such a binary tree, that was created for symbols A B C D. The frequencies of these symbols are shown inside rectangles while circles represent the sum of previous frequencies. The dictionary (figure 2) is constructed by traversing the tree from the top.

The size of the original data can be reduced in this way, where the effectiveness becomes greater with larger data.

Lossy compression reduces the size be permanently removing part of the original data. In the simplest form a lossy compression can be as simple as **quantization**

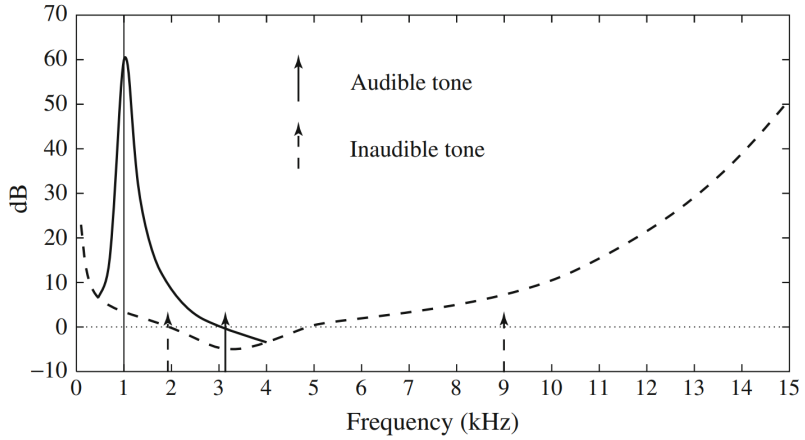
However, a more interesting approach to lossy compression is the usage of **psychoacoustic model**. The psychoacoustics is a science studying the perception of sound by human ear. A known fact is that the frequency range of human hearing is between 20 Hz and 20 kHz. But with the varying frequency a sensitivity of hearing is also affected. This is shown by a curve known as an absolute threshold of human hearing (figure 3). This is an experimentally determined curve that defines which sounds is the human ear capable of hearing. Sounds that are under this curve are not audible. From figure it is apparent that the human ear is most sensitive between 2-3 kHz.



Obrázek 3: Threshold of absolute hearing

This phenomenon could already be used for compression, that is remove the sounds (e.g. noise) outside of the curve. However, that would not be a very effective compression as most of sounds recorded tends to be in the audible range.

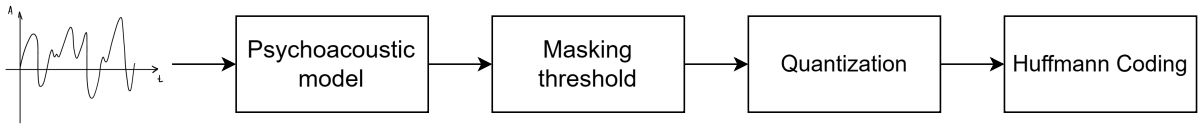
Another phenomenon used for compression that is of more interest is **tonal masking**. When a tone with given frequency and power is played it affects the audibility of another tones located in frequency proximity. As it turns out the affection is severe. Example is shown in figure 4, where a tone of frequency 1000 Hz and power of 60 dB was generated. As can be seen the tone affected the absolute threshold severely and the tone of frequency 2000 Hz and power of 5 dB normally audible is now completely masked. A tone like this is redundant for the final audio file and thus the information about it can be removed.



Obrázek 4: Tonal masking

Project

The plan of achieving compression in this project was to use lossless compression as well as lossy compression. The diagram of whole compression is depicted in figure 5. Firstly the input signal was divided into segments of 512 samples. Then a psychoacoustic model was used to determine the masking threshold. After that a quantization for multiple values was performed and based on the masking threshold the desired value of quantization was selected. The last step was to use the Huffmann coding.



Obrázek 5: Diagram of compression

Preparation of data

Analyzing the input audio data as whole in improper. Therefore the input data was divided into segments of 512 sample and each segment was analyzed separately. This ensures bigger precision as well as for example a faster calculation of FFT, because the FFT algorithm for length of power of two is faster than for any arbitrary length. The segments would exhibit discontinuities at the edges, therefore a Hanning window is applied at the segmented data, according to equation 1. This would smooth out the data on the edges. But because of that an overlap for the segments has to be implemented.

$$h(i) = \sqrt{\frac{8}{3}} \cdot \frac{1}{2}\left(1 - cos\left(2\pi \frac{i}{N-1}\right)\right), 0 < i < N-1 \tag{1}$$

The implementation of this in MATLAB is as follows with the function *hanning*.

```
h = (sqrt(8/3) * hanning(512, 'periodic'))';
```

Psychoacoustic model

The goal of the psychoacoustic model is to go through each segment and find tones. For these tones it has to find the function that affects the absolute threshold. The model is not specified unambiguously. In this project a model described in ISO 11172-3 was used.

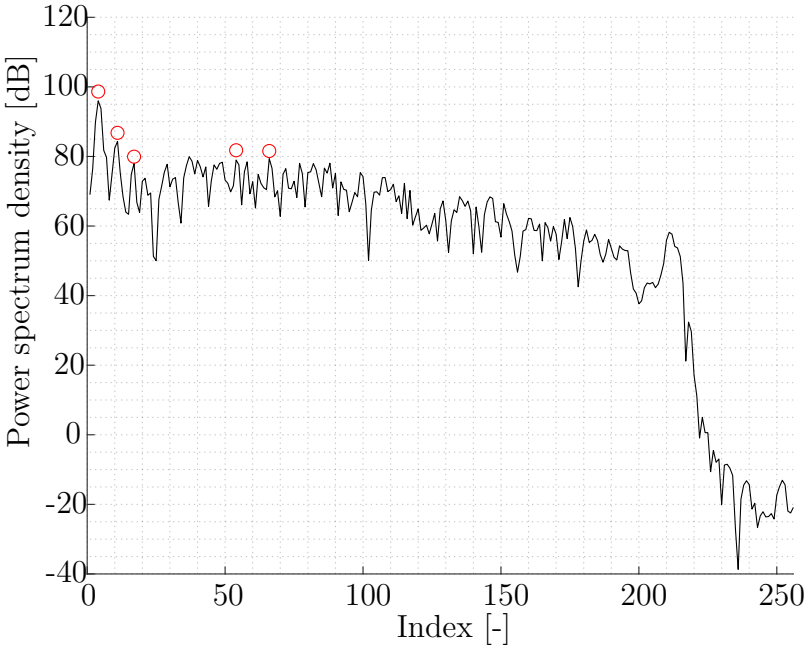
The segment is converted to frequency domain using FFT and after that to power spectral density (PSD) which is a measure of signal's power content versus frequency. To convert to PSD the equation 2 is used.

$$PSD(k) = 20 \cdot \log_{10}\left|\frac{1}{N}\sum_{l=0}^{N-1} h(l) \cdot s(l) \cdot e^{-j \cdot k \cdot l \cdot \frac{2\pi}{N}}\right| \quad \text{dB}, \quad k = 0, \ldots, \frac{N}{2} \tag{2}$$

The equation 2 is used to convert to PSD directly from the original data. The *h(l)* is a Hann window and *s(l)* is the original signal. In MATLAB this can be achieved in following way.

```
x_F = abs(fft(s.*h, FFTLength));
x_F = x_F(1:(FFTLength/2));
PSD_dB = 20 * log10(x_F / FFTLength);
```

Local extrema are then found on this data and and are compared as to in which way they affect the neighbouring frequency bands. For example if a local maxima, which is located in an index range of $2 < k < 63$, is more than 7 dB bigger than the values in the neighbouring indexes $k \pm 2$, it is considered to be a tone. The figure 6 shows the tones that were found for one particular segment.



Obrázek 6: Search of tones

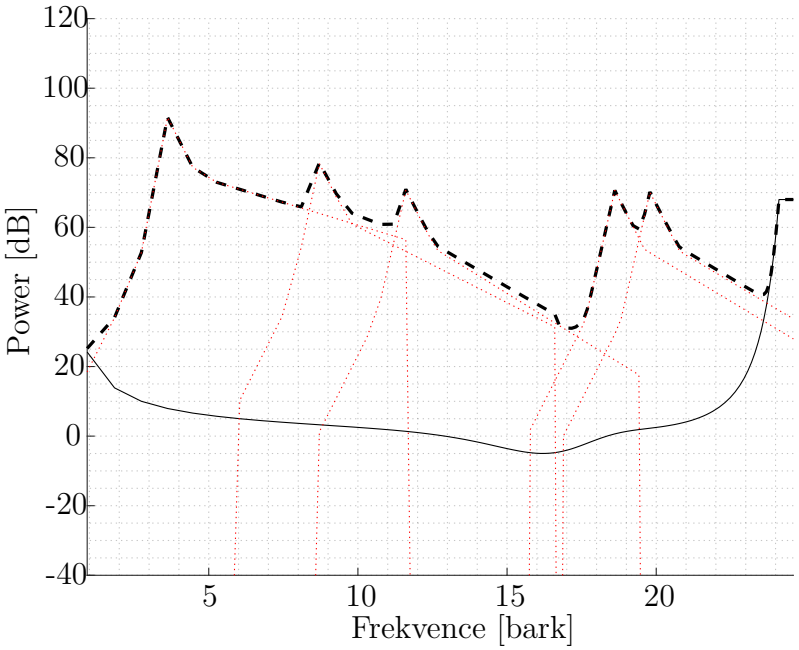
Then the way these notes affect the curve of the absolute threshold has to be determined. This is done based on tables and equation provided by ISO 11172-3. These curves are then summed with the absolute threshold and the global masking threshold is received. It is seen on figure 7.

Quantization

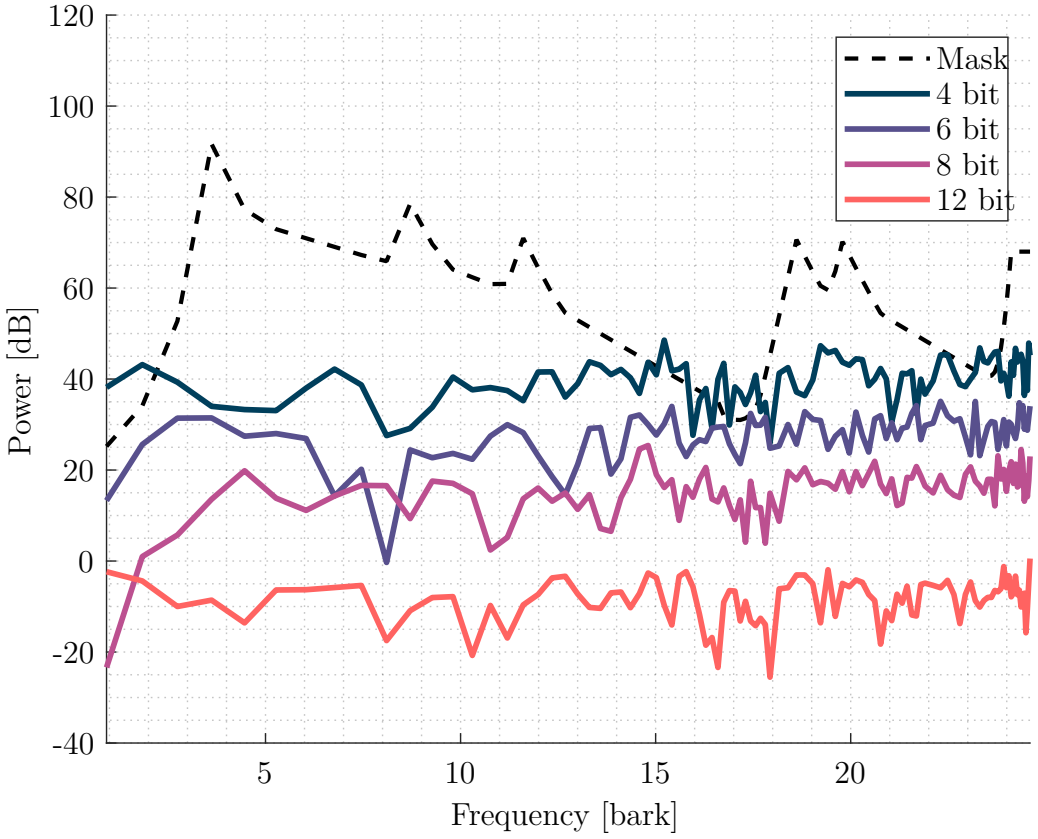
The next step was to quantize the segment. It is quantize to four different values, 4, 6, 8 and 12 bits and for quantized segments for these values a PSD of quantization error is calculated.

```
kvantovani_error = s - s_kvantovane;
error_F = abs(fft(kvantovani_error .* h, FFTLength));
error_F = error_F(1:(FFTLength/2));
error_PSD_dB = 20 * log10(error_F / FFTLength);
```

These errors are shown in figure 8. If an error curve is located under the masking threshold the effect of quantization should not be apparent in the final audio. So a quantized data is kept for the smallest number of bits where the quantization error curve is



Obrázek 7: Masking threshold



Obrázek 8: Effect of quantization

everywhere under the masking threshold. In this particular case the data that would be kept would most likely be a quantized signal for 6 bits as it seems the error curve for this value is under the masking threshold.

Huffmann coding

Two functions were written for this project that fulfill a task of encoding the data based on Huffmann algorithm described earlier and then decode the decoded data back to its origin. These functions are *Huffman_encode* and *Huffman_decode*. The input of the first function is an array of data and the output is a bitstream of encoded data in the form of character string as well as the structure of the binary tree used to encode this data. The later function takes this encoded data and a structure of the binary tree and returns the original data.

The last step of the compression was to use the function for encoding data and saving it.

Conclusion

The encoded data was then stored as a text file and the size of this file was compared to the original .wav file. The following table shows the sizes of each file. It is worth mentioning that the encoded data do not contain information about the binary tree which would be necessary for decoding the data back.

	Size [KB]
Encoded data	915
Original data	2239