Tomáš Milostný (xmilos02)

Protokol k projektu ISS

4. ledna 2021

Tomáš Milostný (xmilos02)

Použité technologie

Zvoleným programovacím jazykem pro vytvoření tohoto projektu je C# 9. Byla vytvořena desktopová aplikace nad frameworkem Windows Forms a platformou .NET 5. Použitým editorem je Microsoft Visual Studio 2019 (pro přeložení a spuštění projektu přes soubor ProjectISS.sln). Použitými knihovnami jsou NAudio (načtení vzorků z .wav souborů) a OxyPlot (grafy).

Zdrojové soubory projektu obsahují jednotlivé formuláře, soubory se sdílenými funkcemi a strukturami. SamplesData je C# třída záznamu uchovávající informace o souboru a jeho vzorcích. Frame uchovává informace o jednom rámci. Některé funkce jsou naprogramovány asynchronním způsobem, kvůli urychlení aplikace při práci s rozsáhlejšími daty.

Implementované funkce jsou v souboru SharedFuncs.cs a jsou volány z hlavního formuláře Form1.cs.

Zdroje

https://github.com/naudio/NAudio

https://github.com/oxyplot/oxyplot

https://oxyplot.readthedocs.io/

https://docs.microsoft.com/

https://en.wikipedia.org/

https://automatizace.hw.cz/clanek/2006031701

https://towardsdatascience.com/understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520

https://stackoverflow.com/questions/4364823/how-do-i-obtain-the-frequencies-of-each-value-in-an-fft/4371627#4371627

bearcave.com/misl/misl_tech/signal/idft/idft.java

1., 2. Tabulka souborů

Nahráno programem Audacity, vzorkovací frekvence 16000 Hz, mono kanál.

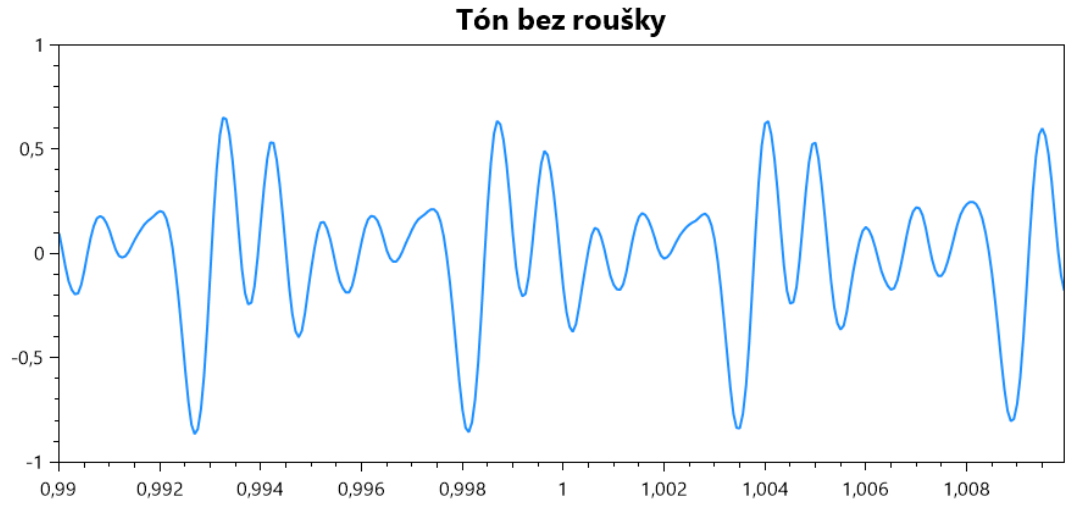
Soubor	**Délka [s]**	**Délka [vzorky]**
maskoff_tone.wav	1.039	16 624
maskon_tone.wav	1.088	17 408
maskoff_sentence.wav	1.010	16 160
maskon_sentence.wav	1.010	16 160

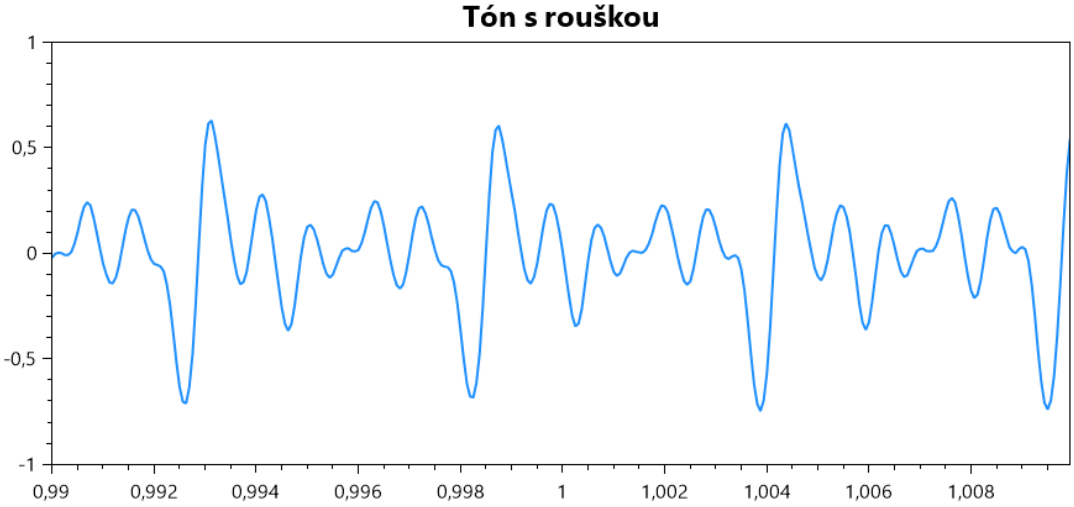
Tomáš Milostný (xmilos02)

3. Rozdělení na rámce

Extrahovaná 1s nahrávky je ustředněna, normalizována a rozdělena na rámce o délce 20ms, překrývající se po 10ms. Použita je nakonec délka 1.01s, kvůli zarovnání na celých 100 rámců.

Vzorec pro výpočet délky 1 rámce: $n = \frac{t_{ms}}{1000} \cdot Fs = \frac{20}{1000} \cdot 16000 = 320$ vzorků





4. Centrální klipování, autokorelace, lag, základní frekvence

Algoritmy byly aplikovány na 1. rámec tónu bez roušky

Tón bez roušky (f0):

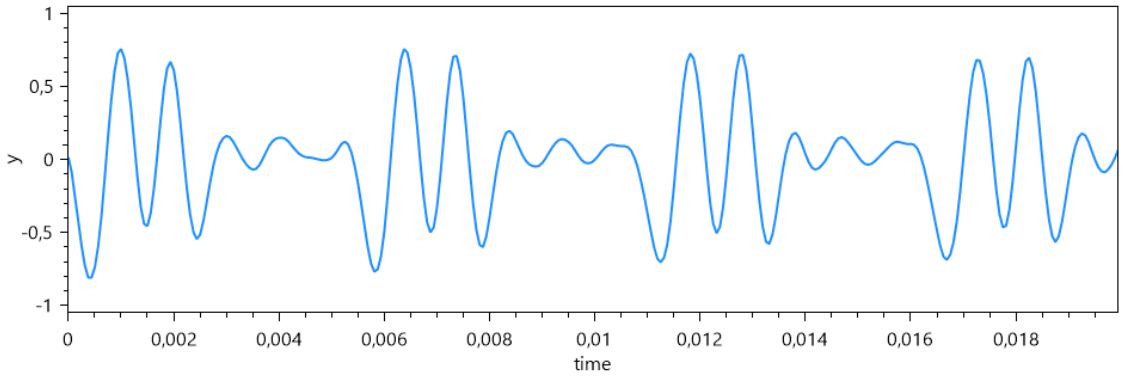
- Střední hodnota: 183,5148 Hz
- Rozptyl: 1,4005 Hz^2

Tón s rouškou (f0):

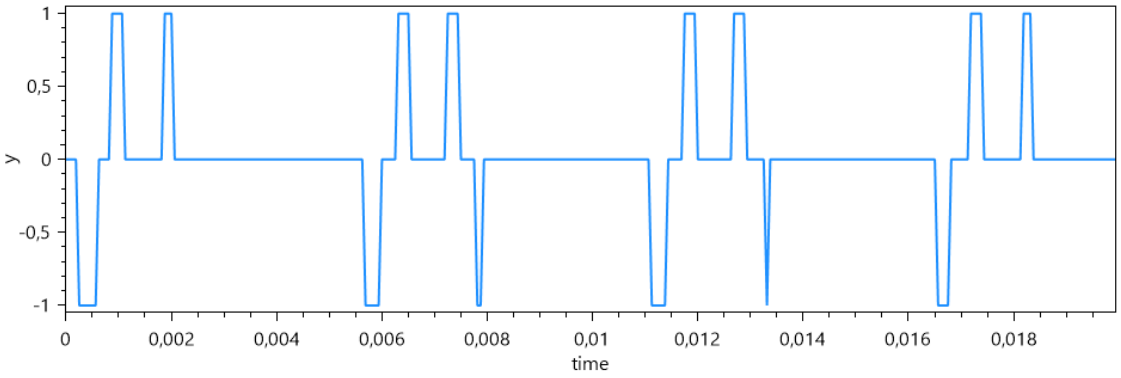
- Střední hodnota: 179,4194 Hz
- Rozptyl: 1,231 Hz^2

Tomáš Milostný (xmilos02)

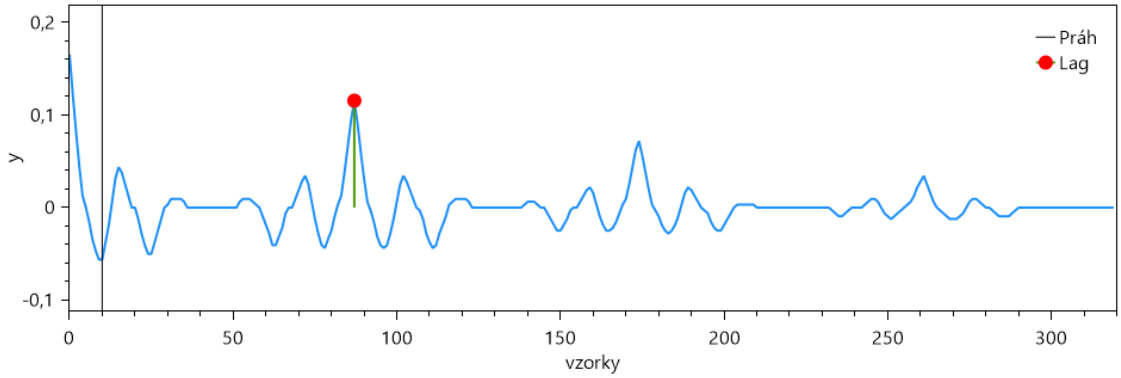
Rámec (tón bez roušky).



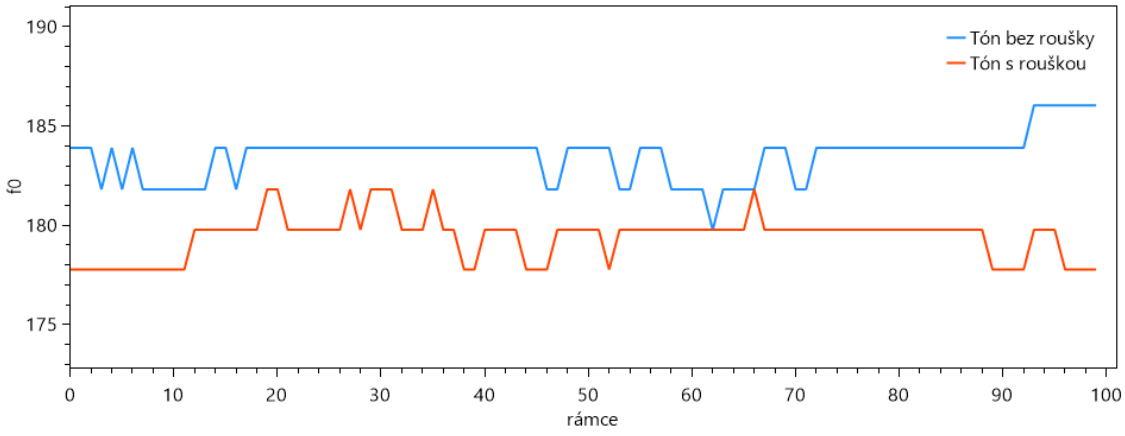
Centrální klipování s 70 %.



Autokorelace.



Základní frekvence rámců.



Tomáš Milostný (xmilos02)

Implementace autokorelace

Autokorelace je implementována jako asynchronní metoda volaná v cyklu pro všechny rámce.
Implementuje vzorec: $\hat{R}[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n] \cdot x[n+k]$

Práh (AutocorrelationThreshold) je 10 vzorků.

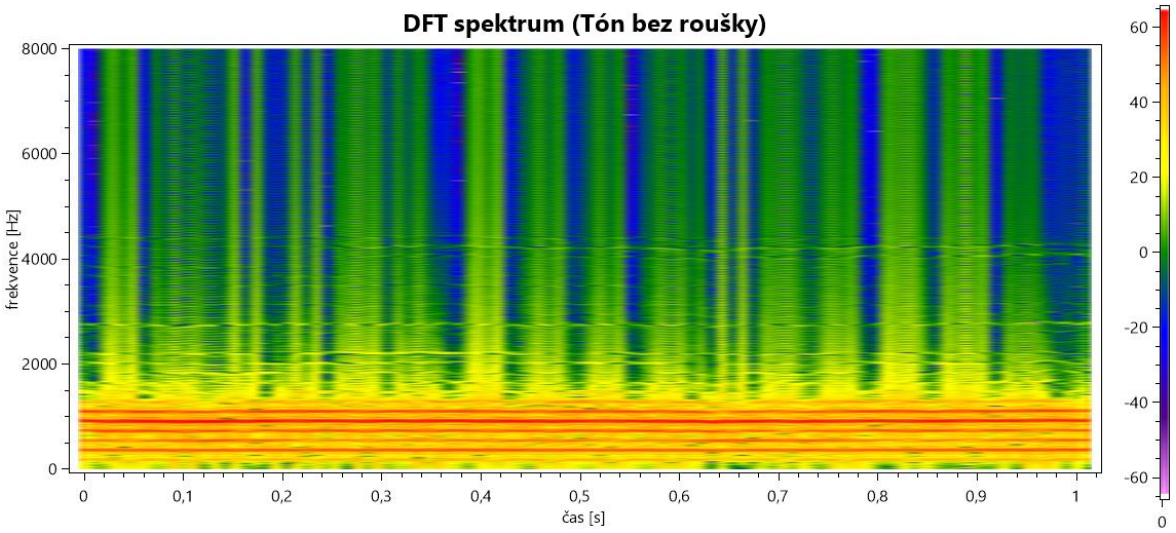
Postupně jsou ukládány základní frekvence f0 vypočítané z indexu lagu rámce.

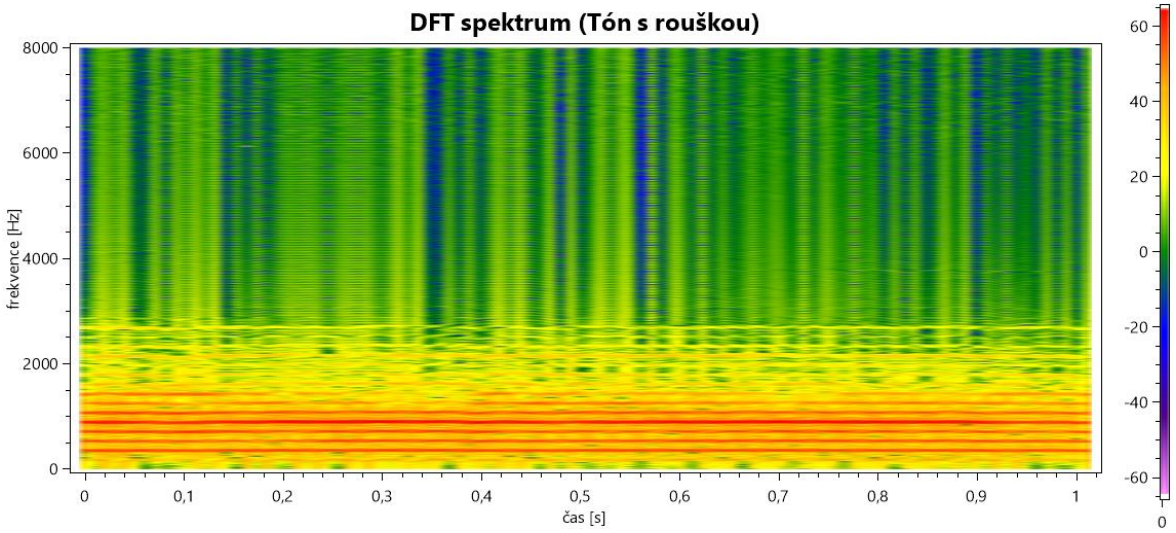
$$f_0 = \frac{Fs}{lagIndex}$$

```
private static async Task<double> Autocorrelation(Frame frame)
{
    return await Task.Run(() =>
    {
        int N = frame.DataPoints.Length;
        int lagIndex = AutocorrThreshold;
        for (int k = 0; k < N; k++)
        {
            var tmp = new List<double>();
            for (int n = 0; n < N - k; n++)
            {
                tmp.Add(frame.DataPoints[n].Y * frame.DataPoints[n + k].Y);
            }
            double sum = tmp.Sum() / N;
            frame.AutocorrelationCoeficients[k] = new(k, sum);

            if (k ≥ AutocorrThreshold && sum > frame.LagPoint.Y)
            {
                frame.LagPoint = frame.AutocorrelationCoeficients[k];
                lagIndex = k;
            }
        }
        return (double)Fs / lagIndex;
    });
}

public static async Task Autocorrelation(SamplesData x)
{
    var tasks = new List<Task<double>>();
    x.F0Points = new DataPoint[x.Frames.Count];

    //run autocorrelation for every frame
    foreach (var frame in x.Frames)
    {
        tasks.Add(Autocorrelation(frame));
    }

    //get lag indexes from each autocorrelation, transform into f0
    for (int i = 0; i < tasks.Count; i++)
    {
        x.F0Points[i] = new(i, await tasks[i]);
    }
}
```

Tomáš Milostný (xmilos02)

5. Spektra DFT





Tomáš Milostný (xmilos02)

Implementace DFT

DFT je implementována jako asynchronní metoda volaná v cyklu pro všechny rámce. Postupně jsou ukládány komplexní koeficienty.

```
private static async Task<Complex[]> DFT(Frame frame, int N)
{
    return await Task.Run(() =>
    {
        var result = new Complex[N];

        for (int k = 0; k < N; k++)
        {
            var reals = new List<double>();
            var imags = new List<double>();
            for (int n = 0; n < N; n++)
            {
                if (n < frame.DataPoints.Length)
                {
                    double arg = 2 * Math.PI * k * n / N;
                    reals.Add(frame.DataPoints[n].Y * Math.Cos(arg));
                    imags.Add(frame.DataPoints[n].Y * Math.Sin(arg));
                }
                else break; //zero padding (adding zeros to sums does nothing)
            }
            result[k] = new Complex(reals.Sum(), -imags.Sum());
        }
        return result;
    });
}

public static async Task DFT(SamplesData x, int N = 1024)
{
    var tasks = new List<Task<Complex[]>>();
    foreach (var frame in x.Frames)
    {
        tasks.Add(DFT(frame, N));
    }

    for (int i = 0; i < tasks.Count; i++)
    {
        x.Frames[i].DFTCoeficients = await tasks[i];
    }
}
```

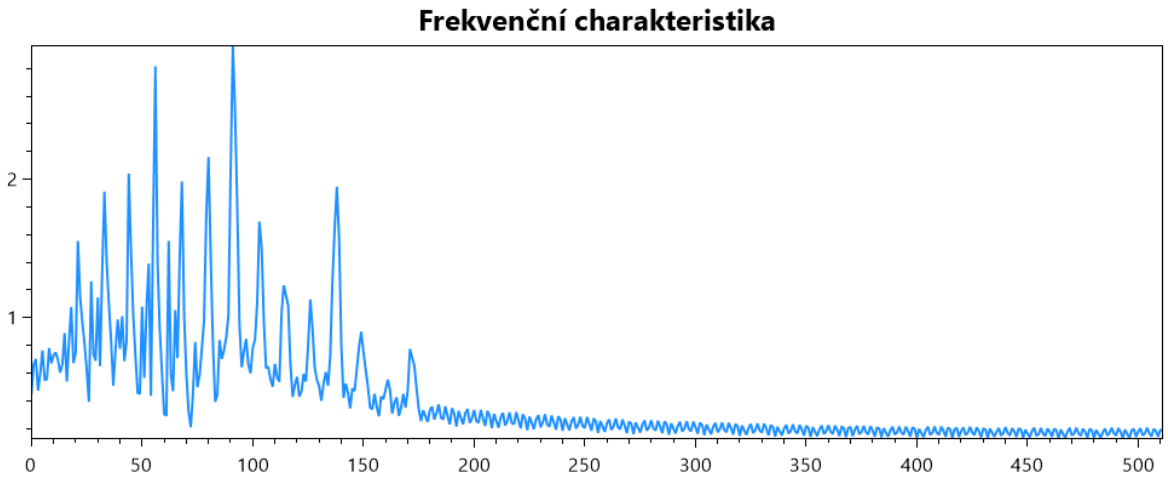
V souboru SpectrogramForm.cs se nachází implementace zobrazující výsledný spektrogram.

Výsledná frekvence na ose y je počítána pomocí vzorce: $f = k \cdot \frac{Fs}{N} \Rightarrow k = {f}/{\frac{Fs}{N}}$

```
int topX = data.Frames.Count;
int topY = SharedFuncs.Fs / 2;

var heatmapData = new double[topX, topY];
int N = data.Frames[0].DFTCoeficients.Length;

for (int x = 0; x < topX; x++)
{
    for (int freq = 0; freq < topY; freq++)
    {
        int k = (int)(freq / ((double)SharedFuncs.Fs / N));
        heatmapData[x, freq] = 10 * Math.Log10(Math.Pow(Complex.Abs(data.Frames[x].DFTCoeficients[k]), 2));
    }
}
```

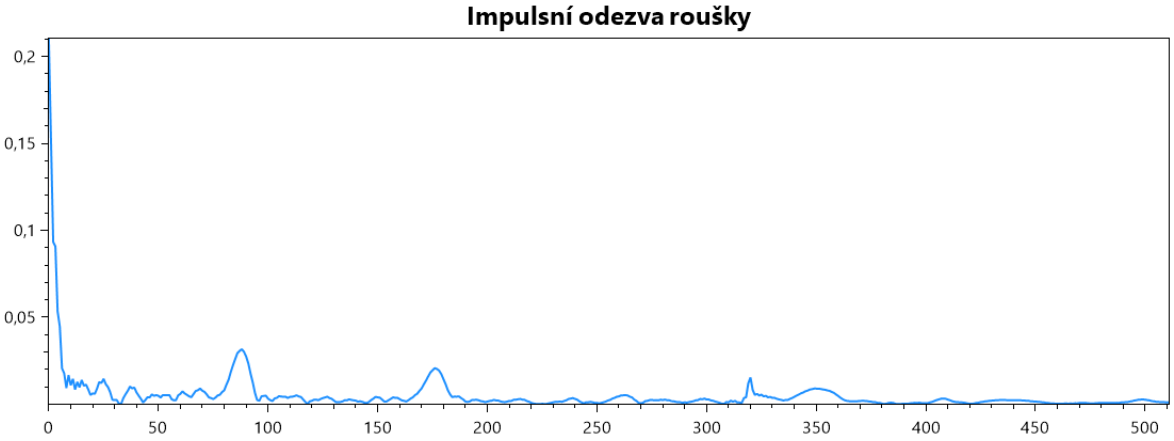
6. Frekvenční charakteristika

$$H(e^{j\omega}) = \frac{\sum_{i=0}^{q} b[i] \cdot e^{-j\omega}}{1 + \sum_{i=0}^{r} a[i] \cdot e^{-j\omega}} \Rightarrow \frac{\overline{|Y(e^{j\omega})|}}{1 + \overline{|X(e^{j\omega})|}}$$



Máme N/2 (symetrie) absolutních hodnot koeficientů DFT, které zprůměrujeme a postupně dosadíme do vzorce frekvenční charakteristiky.

7. Impulsní odezva, IDFT

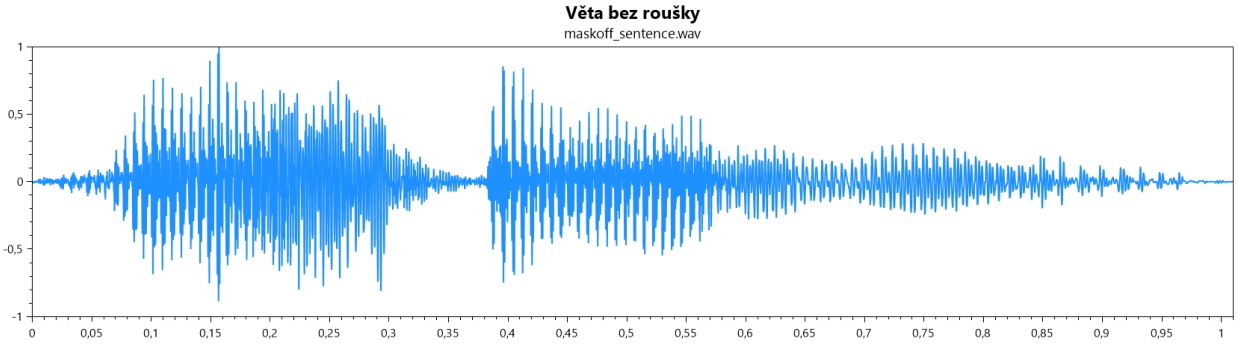


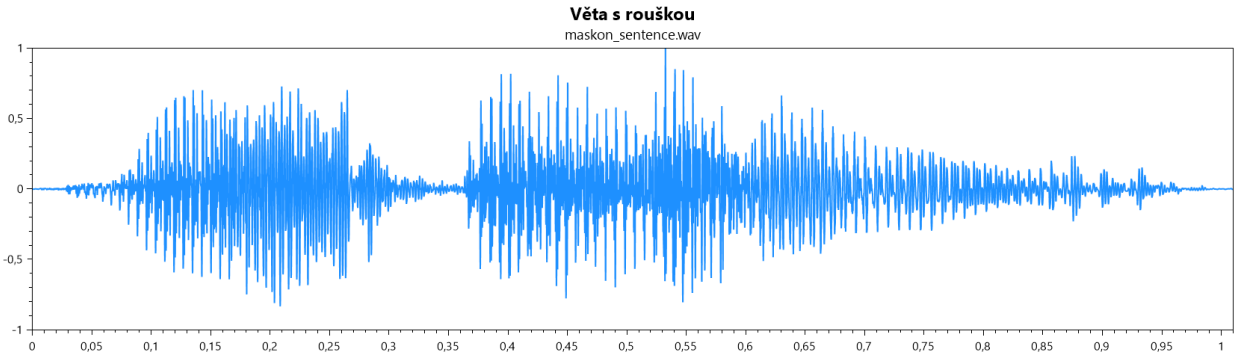
Implementace inverzní DFT

IDFT je implementována nad koeficienty frekvenční charakteristiky.

```
public static DataPoint[] IDFT (DataPoint[] freqCharPoints, int N = 1024)
{
    var result = new DataPoint[N / 2];
    for (int n = 0; n < N / 2; n++)
    {
        var reals = new List<double>();
        var imags = new List<double>();
        for (int k = 0; k < N / 2; k++)
        {
            double arg = 2 * Math.PI * n * k / N;
            reals.Add(freqCharPoints[k].Y * Math.Cos(arg));
            imags.Add(freqCharPoints[k].Y * Math.Sin(arg));
        }
        result[n] = new(n, Complex.Abs(new Complex(reals.Sum(), imags.Sum())) / N);
    }
    return result;
}
```

8. Grafy – věta s rouškou a bez roušky





9. Závěr

Řešení za použití technologií jako jazyk C#, Windows Forms, OxyPlot i .NET numerickou knihovnou bylo zajímavé, ale místy složité kvůli nestandardnímu postupu. Volný výběr jazyka jsem ocenil a jsem rád, že jsem si mohl rozšířit znalosti při práci s oblíbeným jazykem a na tomto projektu.

U nahrávek jsem měl trochu problém s držením intonace, ale toto snad nebude mít vliv na výsledné hodnocení. Celkově se jinak práce na projektu dařila až na pár problémů při řešení úkolů 6, 7 a 8 před zkouškovým obdobím a doufám, že jsem zde alespoň nakročil správným směrem.