

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ISS projekt 2022/23

Obsah

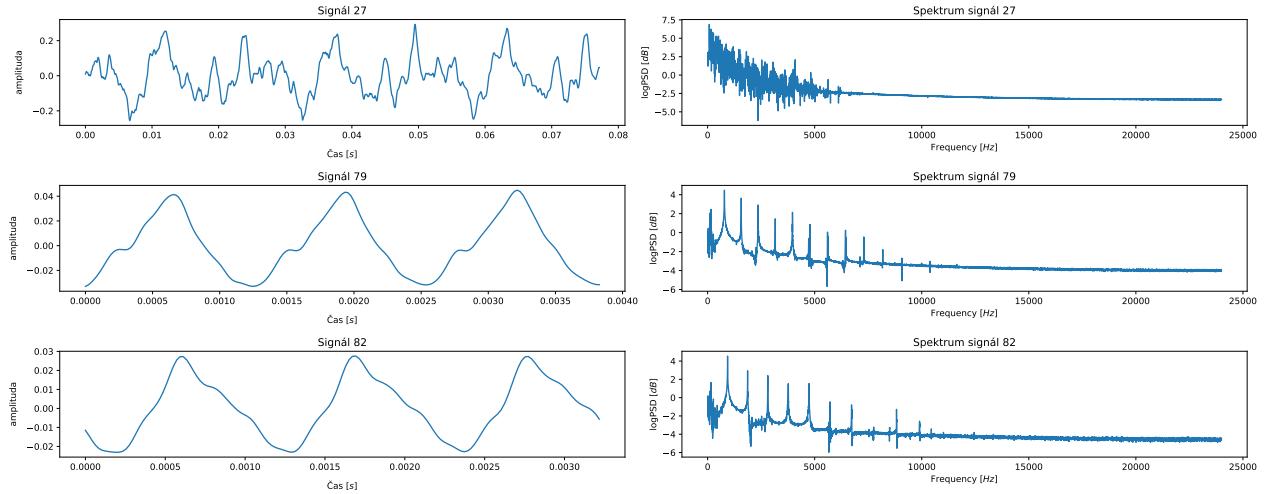
1	Úvod	2
2	Základy	3
3	Určení základní frekvence	4
4	Zpřesnění odhadu základní frekvence f_0	6
5	Reprezentace klavíru	7
6	Syntéza tónů	8
7	Generování hudby	9
8	Spektrogram	10
9	Závěr	11

1 Úvod

Cílem projektu je vytvořit co nejvěrnější syntetické piano jehož zvuková charakteristika bude komprimovaná pouze do několika parametrů.

2 Základy

Načtení půl sekundy ustálené části tónů 27, 79, 82, zobrazení jejich 3 period a vypočítání spektra. Spektrum počítám pomocí funkce `np.log(seg_spec + 10**-5)`, hodnota `10**-5` se připočítává z důvodu, že logaritmus není v 0 definovaný.



Obrázek 1: Nalevo 3 periody, napravo spektra tónů

3 Určení základní frekvence

Na vypočítání frekvencí tónů jsem použil autokorelaci do tónu 49 a poté jsem použil DFT. Na nižší tóny jsem použil autokorelaci z důvodu, že je na menší signály s frekvencemi přesnější než DFT, které mi vracelo hodnoty 2x větší než referenční hodnota. Všechny mnou vypočítané hodnoty frekvencí jsem vypsal do souboru freqs.txt.

Pro výpočet základní frekvence tónu 27 jsem použil autokorelaci:

```
● ● ●

def autocorr(x):
    result = np.correlate(x, x, mode='full')
    return result[result.size // 2:]

def find_base_frequency_using_autocorr(signal, sample_rate):
    autocorr_signal = autocorr(signal)
    peaks, _ = scipy.signal.find_peaks(autocorr_signal)
    best_peak = peaks[0]
    for peak in peaks:
        if autocorr_signal[peak] > autocorr_signal[best_peak]:
            best_peak = peak

    base_frequency = sample_rate / best_peak
    return base_frequency
```

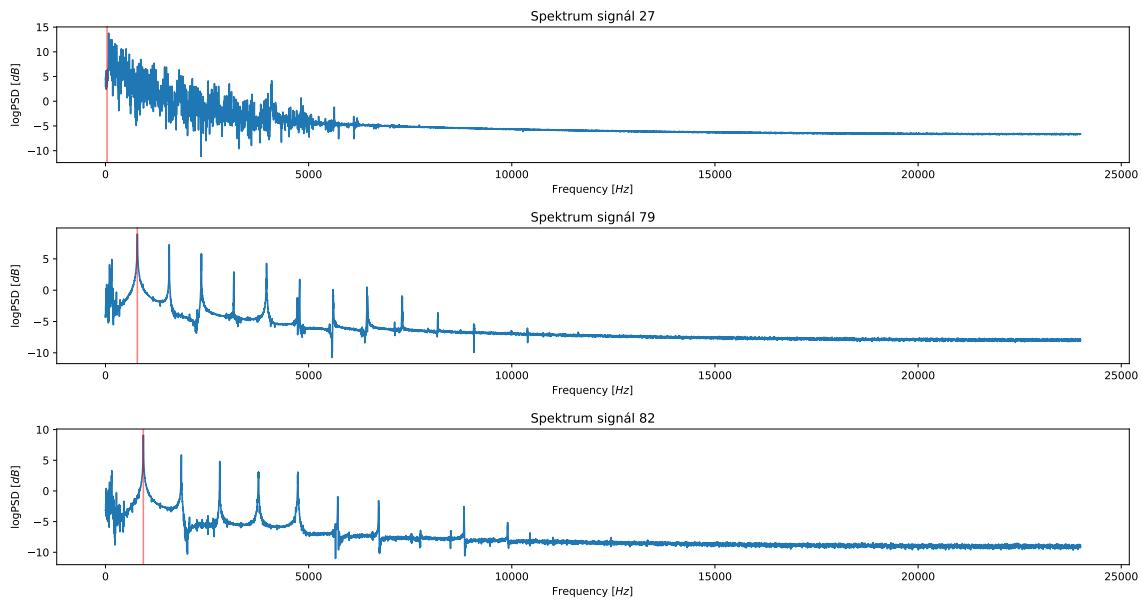
Obrázek 2: Výpočet základní frekvence pomocí autokorelace

Pro tóny 79, 82 jsem použil DFT.

```
● ● ●

def find_base_frequency_using_fft(signal, sample_rate):
    fft = np.fft.fft(signal)
    fft = fft[:len(fft)//2]
    max_freq = np.argmax(np.abs(fft))
    base_frequency = max_freq * sample_rate / len(signal)
    return base_frequency
```

Obrázek 3: Výpočet základní frekvence pomocí DFT



Obrázek 4: Znázornění základní frekvence v grafu

Mezi nalezenými frekvencemi a referenčními jsem našel odchylky oproti referenčním frekvencím. Tato chybovost je z důvodu přesnosti metody DFT a autokorelace.

4 Zpřesnění odhadu základní frekvence f_0

Na zpřesnění odhadu skutečné frekvence jsem použil výpočet koeficientů rozložených 100 centů okolo nejbližší midi. Z výsledků jsem zjistil, že některé hodnoty jsou velice blízké referenčním, ale jsou tam určitě rozdíly. Všechny vypočtené tóny jsem uložil do souboru freqs_dtft.txt.

```
def precise_dtft(x,fmax,FREQRANGE,FREQPOINTS):
    n = np.arange(0, x.size)
    fsweep = np.linspace(fmax-FREQRANGE,fmax+FREQRANGE,FREQPOINTS)

    A = np.zeros([FREQPOINTS, x.size],dtype=complex)
    for k in np.arange(0,FREQPOINTS):
        A[k,:] = np.exp(-1j * 2 * np.pi * fsweep[k] / Fs * n)      # norm. omega = 2 * pi * f / Fs ...
    Xdtft = np.matmul(A,x.T)
    precisefmax = fsweep[np.argmax(np.abs(Xdtft))]
    return precisefmax

def find_base_frequency(base_frequency, signal):
    FREQRANGE = 100 / 1200 * base_frequency
    FREQPOINTS = 200
    val = precise_dtft(signal, base_frequency, FREQRANGE, FREQPOINTS)
    return np.abs(val)
```

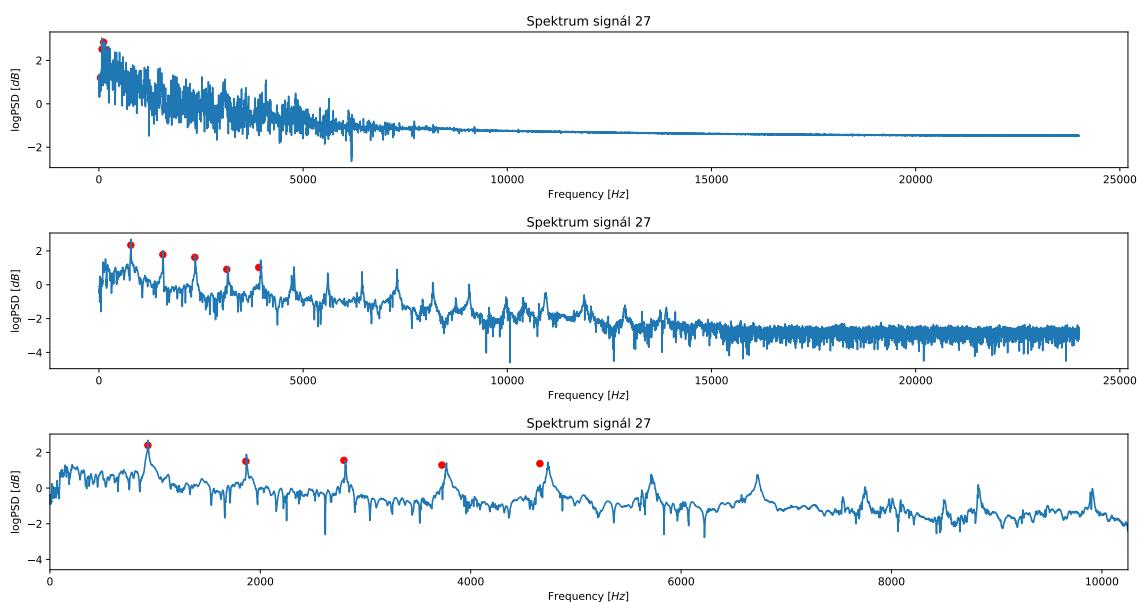
Obrázek 5: Výpočet DTFT

5 Reprezentace klavíru

Vypočítané hodnoty jsou poblíž vrcholů, bohužel si ale myslím, že je zde až příliš velká odchylka u některých bodů a je tedy možné, že můj výpočet u některých bodů nesedí.

```
def represent_tone(tone, freq):
    N = tone.size
    coeffs = []
    FREQRANGE = 100 / 1200 * freq
    FREQPOINTS = 200
    for k in range(5):
        precise_freq = precise_dtft(tone, (k+1)*freq, FREQRANGE, FREQPOINTS)
        X = np.fft.fft(tone)
        coeffs.append(X[int(precise_freq * N / Fs)])
    return coeffs
```

Obrázek 6: Reprezentace klavíru tónů pomocí 10 floating point čísel



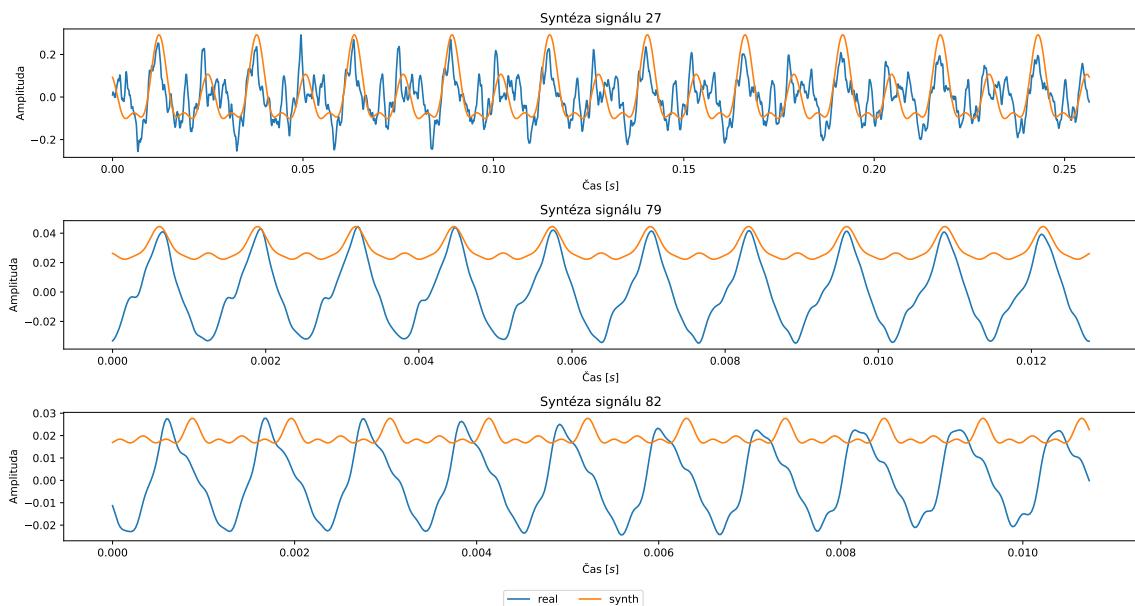
Obrázek 7: Graf reprezentace tónů klavíru pomocí 10 floating point čísel

6 Syntéza tónů

Z grafu vidíme, že syntetizované tóny obkresují získané tóny z nahrávky. U vyšších frekvencí dochází k posunu v amplitudě. U nižších frekvencí mi syntetizování funguje lépe než u vyšších zřejmě z odchylky výpočtu.

```
● ● ●  
  
def synthesis_tone(coeffs, freq, amplitude, Length=1):  
    """synthesis of tone using FR and given coefficients"""  
    N = Fs * Length  
    tone = np.zeros(N)  
    for k in range(5):  
        tone += np.real(coeffs[k] * np.exp(2j * np.pi * k * freq * np.arange(N) / Fs))  
  
    tone = tone / np.max(np.abs(tone)) * amplitude  
    return tone
```

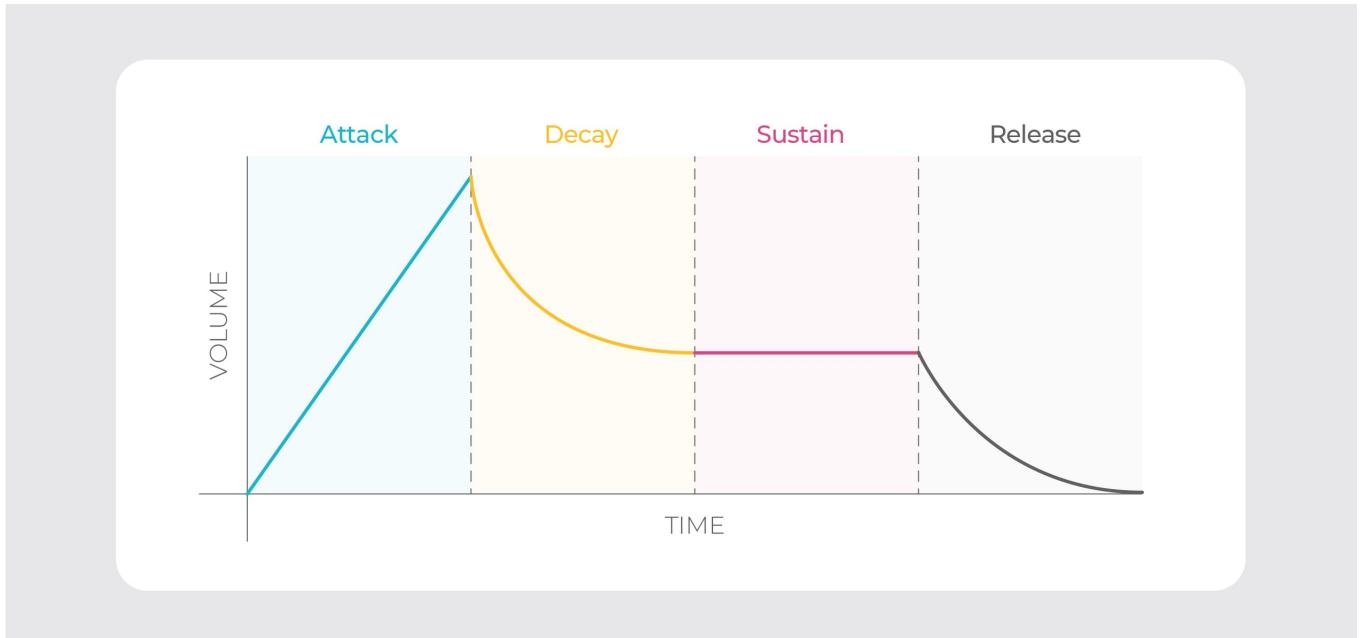
Obrázek 8: Syntetizování tónů



Obrázek 9: Syntetizování tónů

7 Generování hudby

Při generování hudby jsem všechny tóny postupně reprezentoval, poté syntetizoval a nakonec uložil do souboru `audio/out_48k.wav` a `audio/out_8k.wav`. Aby neprobíhal výpočet tak jsem implementoval i uložení koeficientů pro rychlejší načtení při opětovném spouštění. Také jsem se pokusil simulovat ADSR na skladbě podle obrázku.



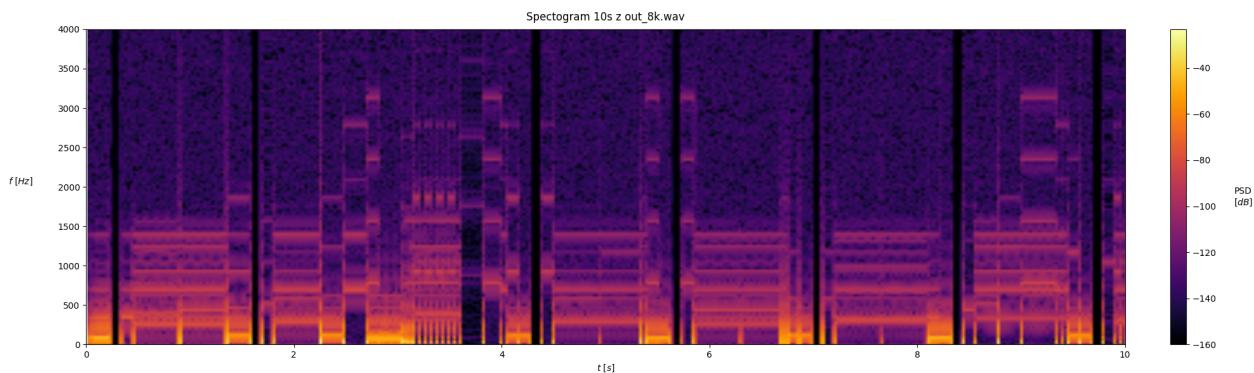
Obrázek 10: ADSR

```
● ● ●  
# make attack and decay  
attack = np.linspace(0, 1, int(0.1 * Fs))  
decay = np.linspace(1, 0, int(0.1 * Fs))  
song[:int(0.1 * Fs)] *= attack  
song[-int(0.1 * Fs):] *= decay
```

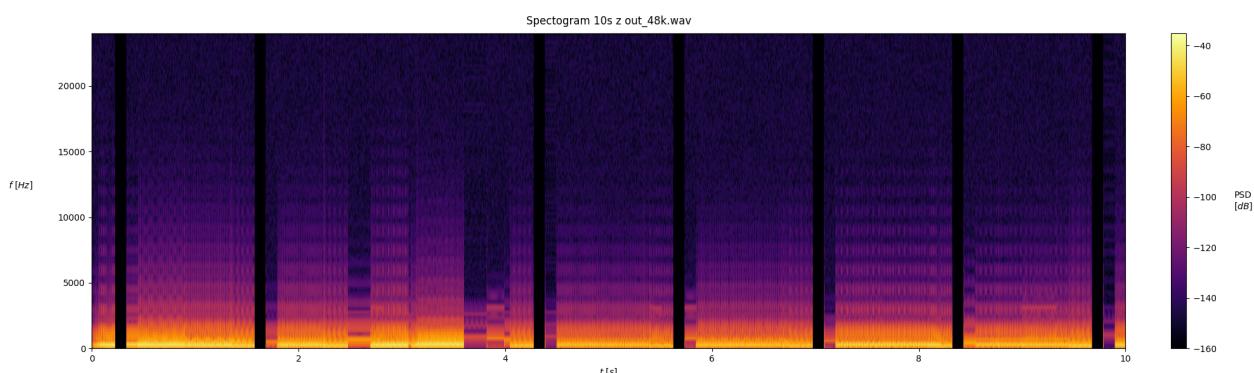
Obrázek 11: ADSR

8 Spektrogram

K vykreslení spektrogramu jsem využil dostupných příkladů z python notebooků. Jednotlivé hodnoty počítám přes logaritmus $10 * \text{np.log10}(sgr + 1e-20)$.



Obrázek 12: Spektrogram 8k nahrávka



Obrázek 13: Spektrogram 48k nahrávka

9 Závěr

Syntetizované tóny zní velice podobně původní nahrávce i finalní píseň zní docela dobře, ale je zřejmé, že tam dochází k nějakému problému, kdy občas zní vypadá jako by tam tón nebyl, ale je jen velice potichu. Projekt byl vypracován v pythonu, dokumentace v \LaTeX u.