ISS Projekt 2022/23

Roland Schulz xschul06

December 18, 2022

1 Import potřebných knihoven

[1]:
```
import numpy as np
import soundfile as sf
import matplotlib.pyplot as plt
from matplotlib import cm
import pandas as pd
from IPython.display import Audio
from pathlib import Path
import scipy.signal as ss
```

2 Načtení skladby a midi reference ze souborů

[2]:
```
# Načtení skladby
skladba = np.loadtxt("skladba.txt", delimiter=' ', dtype=np.dtype('f, f, i, i'))
pd.DataFrame(skladba)
```

[2]:
```
            f0        f1  f2  f3
0       7200.0    8100.0  64  44
1      12600.0   13500.0  64  44
2      23400.0   24300.0  64  44
3      28800.0   29700.0  64  44
4      50400.0   51300.0  64  49
...        ...       ...  ..  ..
1169   19350.0   19800.0  67  70
1170   46350.0   46800.0  67  70
1171   73350.0   73800.0  67  70
1172  100350.0  100800.0  67  70
1173  127350.0  127800.0  67  70

[1174 rows x 4 columns]
```

[3]:
```
# Načtení midi.text
midi = np.loadtxt("midi.txt", delimiter='\t')[::-1]
pd.DataFrame(midi)
```

[3]:
```
        0        1
0    24.0    32.70
1    25.0    34.65
2    26.0    36.71
3    27.0    38.89
4    28.0    41.20
..    ...      ...
80  104.0  3322.44
81  105.0  3520.00
82  106.0  3729.31
83  107.0  3951.07
84  108.0  4186.01

[85 rows x 2 columns]
```

3 Vytvoření adresářů pro zápis zvukových a obrazových výstupů

[4]:
```
plt_scale = 3
img_gen_folder = "./gen_img/"
Path(img_gen_folder).mkdir(exist_ok=True)

audio_folder = "../audio/"
Path(audio_folder).mkdir(exist_ok=True)
```

4 Získání konkrétních tónů z klavírní nahrávky

převzato ze zadání

[5]:
```
MIDIFROM = 24
MIDITO = 108
SKIP_SEC = 0.25
HOWMUCH_SEC = 0.5
WHOLETONE_SEC = 2
howmanytones = MIDITO - MIDIFROM + 1
tones = np.arange(MIDIFROM, MIDITO+1)
s, Fs = sf.read('klavir.wav')
N = int(Fs * HOWMUCH_SEC)
Nwholetone = int(Fs * WHOLETONE_SEC)
xall = np.zeros((MIDITO+1, N)) # matrix with all tones - first signals empty,
# but we have plenty of memory ...
samplefrom = int(SKIP_SEC * Fs)
sampleto = samplefrom + N
for tone in tones:
    x = s[samplefrom:sampleto]
    x = x - np.mean(x) # safer to center ...
    xall[tone,:] = x
```

```
        samplefrom += Nwholetone
        sampleto += Nwholetone

display(Audio(s, rate=Fs))
```

<IPython.lib.display.Audio object>

[6]:
```
my_tones = np.array([37, 77, 93]) # ze zadani
my_tones_fund = midi[my_tones - 24][:, 1]
a, b, c = (xall[my_tone] for my_tone in my_tones)

display(Audio(a, rate=Fs))
display(Audio(b, rate=Fs))
display(Audio(c, rate=Fs))
```

<IPython.lib.display.Audio object>

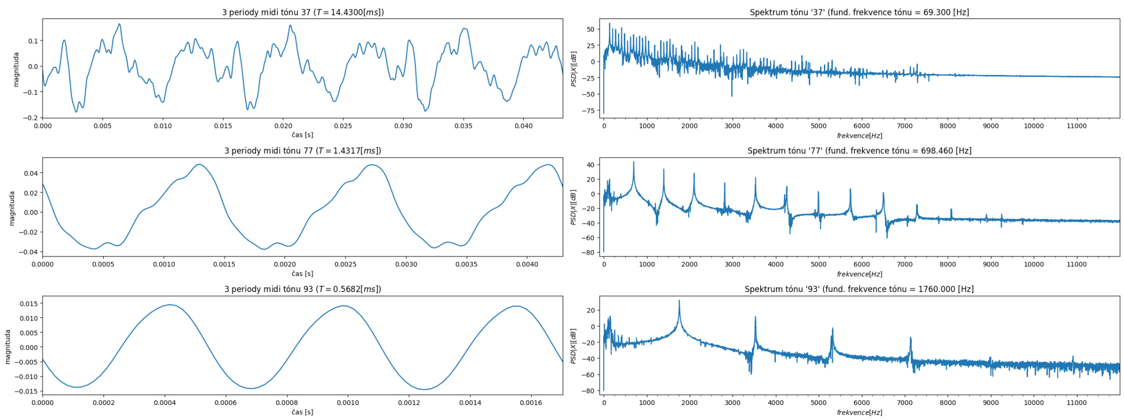
<IPython.lib.display.Audio object>

<IPython.lib.display.Audio object>

5 Základy

[7]:
```
fig, (a_plots, b_plots, c_plots) = plt.subplots(3, 2, figsize=(8*plt_scale,␣
 ↪3*plt_scale))
def plot_tone(tone, id, axis, zakladni_f, periods=3, freq_range_highlight=100):
    # compute n periods
    T = 1/zakladni_f # in [seconds]
    t = np.arange(tone.size) / Fs

    axis[0].set_ylabel(f"magnituda")
    axis[0].set_xlabel(f"čas [s]")
    axis[0].set_xlim(0, periods*T)
    axis[0].plot(t, tone)
    axis[0].set_title(f"{periods} periody midi tónu {id} ($T={T*1000:.4f}␣
 ↪[ms]$)")

    fft_of_tone = np.abs(np.fft.fft(tone)[:tone.size//2])
    t_dft = np.arange(fft_of_tone.size) * 2
    PSD = 10 * np.log10(np.abs(fft_of_tone)**2 + (0.00000001))

    axis[1].set_xticks(np.arange(0, len(fft_of_tone), 1000))
    axis[1].set_xticks(np.arange(0, len(fft_of_tone), 250), minor=True)
    axis[1].set_ylabel(f"$PSD |X| [dB]$")
    axis[1].set_xlabel(f"$frekvence [Hz]$")
    axis[1].set_title(f"Spektrum tónu \'{id}\' (fund. frekvence tónu =␣
 ↪{zakladni_f:0.3f} [Hz]")
```

```
    axis[1].set_xlim(-100, (t_dft.size))
    axis[1].plot(t_dft, PSD)
    return fft_of_tone

plot_tone(a, my_tones[0], a_plots, my_tones_fund[0])
plot_tone(b, my_tones[1], b_plots, my_tones_fund[1])
plot_tone(c, my_tones[2], c_plots, my_tones_fund[2])

fig.tight_layout()
fig.savefig(img_gen_folder + "my_tones_periods_dft.png")
```



Uložení tónů jako audio/a_orig.wav, audio/b_orig.wav, a audio/c_orig.wav.

[8]:
```
# uložení tónů jako audio/a_orig.wav, audio/b_orig.wav, a audio/c_orig.wav
for orig_tone, orig_name in zip([a, b, c], ["a", "b", "c"]):
    sf.write(audio_folder + f"{orig_name}_orig.wav", orig_tone, Fs)
```

6 Určení základní frekvence

[9]:
```
# funkce pro nalezení peaku ve výstupu DFT
# inspirováno Python notebooky z webu ISS
def get_dft_peak(dft, N):
    kall = np.arange(0,int(N/2) +1)
    f = kall / N * Fs
    Xmag = np.abs(dft[kall])
    #plt.plot(Xmag[:12000])
    #plt.xlim(0, 100)

    # finding the max and showing where we'll compute ...
    fmax = f[np.argmax(Xmag)]
    Xmax = np.max(Xmag)
    return fmax
```

```python
# funkce pro nalezení peaku ve výstupu autokorelace
# inspirováno Python notebooky z webu ISS
def get_autocorr_peak(xac):
    xac = xac[xac.size//2:]
    start = np.argmax(xac < 0)
    p = np.argmax(xac[start:]) + start
    fmax = Fs/p
    return fmax

# výpočet frekvencí metodou DFT
xall_dfts = []
for index, tone in enumerate(xall):
    fft_of_tone = np.fft.fft(tone)
    t_dft = np.arange(fft_of_tone.size)
    fmax = get_dft_peak(fft_of_tone, N)
    xall_dfts.append(fmax)

# výpočet frekvencí metodou autokorelace
xall_autocorr = []
raw_autocorr = []
for index, tone in enumerate(xall):
    xautocorr = ss.correlate(tone, tone, 'full')
    fmax = get_autocorr_peak(xautocorr)
    print(f"{index}: {fmax} [Hz]")
    xall_autocorr.append(fmax)
```

```
0: inf [Hz]
1: inf [Hz]
2: inf [Hz]
3: inf [Hz]
4: inf [Hz]
5: inf [Hz]
6: inf [Hz]
7: inf [Hz]
8: inf [Hz]
9: inf [Hz]
10: inf [Hz]
11: inf [Hz]
12: inf [Hz]
13: inf [Hz]
14: inf [Hz]
15: inf [Hz]
16: inf [Hz]
17: inf [Hz]
18: inf [Hz]
19: inf [Hz]
```

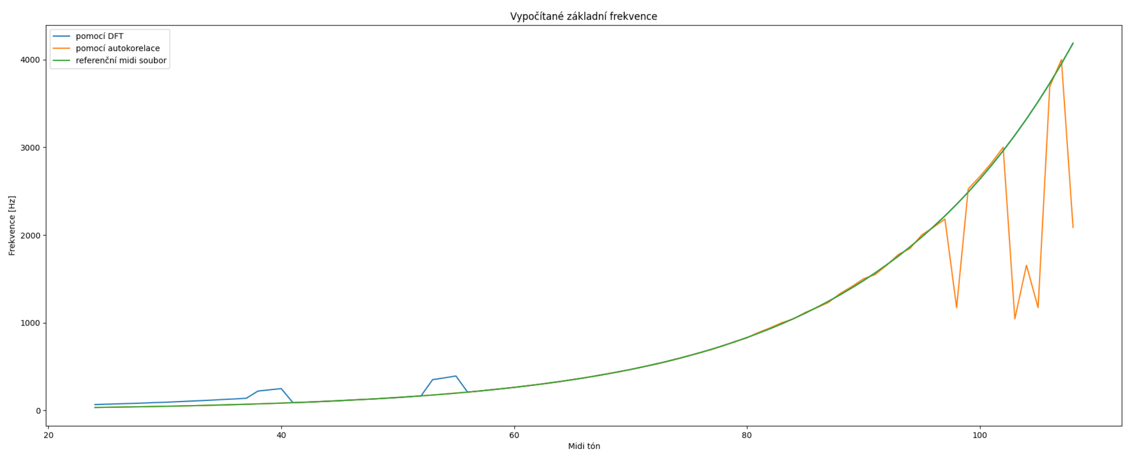
```
20: inf [Hz]
21: inf [Hz]
22: inf [Hz]
23: inf [Hz]
24: 32.80929596719071 [Hz]
25: 34.757422157856624 [Hz]
26: 36.83806600153492 [Hz]
27: 39.02439024390244 [Hz]
28: 41.343669250645995 [Hz]
29: 43.7956204379562 [Hz]
30: 46.42166344294004 [Hz]
31: 49.18032786885246 [Hz]
32: 52.11726384364821 [Hz]
33: 55.172413793103445 [Hz]
34: 58.465286236297196 [Hz]
35: 61.935483870967744 [Hz]
36: 65.57377049180327 [Hz]
37: 69.46454413892909 [Hz]
38: 73.61963190184049 [Hz]
39: 77.92207792207792 [Hz]
40: 82.61617900172116 [Hz]
41: 87.75137111517367 [Hz]
42: 92.84332688588007 [Hz]
43: 98.36065573770492 [Hz]
44: 104.34782608695652 [Hz]
45: 110.59907834101382 [Hz]
46: 117.07317073170732 [Hz]
47: 123.71134020618557 [Hz]
48: 131.14754098360655 [Hz]
49: 138.72832369942196 [Hz]
50: 147.23926380368098 [Hz]
51: 155.84415584415584 [Hz]
52: 164.94845360824743 [Hz]
53: 175.1824817518248 [Hz]
54: 185.32818532818533 [Hz]
55: 196.72131147540983 [Hz]
56: 207.7922077922078 [Hz]
57: 220.1834862385321 [Hz]
58: 234.14634146341464 [Hz]
59: 247.42268041237114 [Hz]
60: 262.2950819672131 [Hz]
61: 277.4566473988439 [Hz]
62: 294.47852760736197 [Hz]
63: 311.68831168831167 [Hz]
64: 328.7671232876712 [Hz]
65: 350.3649635036496 [Hz]
66: 369.2307692307692 [Hz]
67: 393.44262295081967 [Hz]
```

```
68: 417.39130434782606 [Hz]
69: 440.3669724770642 [Hz]
70: 466.0194174757282 [Hz]
71: 494.8453608247423 [Hz]
72: 521.7391304347826 [Hz]
73: 551.7241379310345 [Hz]
74: 585.3658536585366 [Hz]
75: 623.3766233766233 [Hz]
76: 657.5342465753424 [Hz]
77: 695.6521739130435 [Hz]
78: 738.4615384615385 [Hz]
79: 786.8852459016393 [Hz]
80: 827.5862068965517 [Hz]
81: 888.8888888888889 [Hz]
82: 941.1764705882352 [Hz]
83: 1000.0 [Hz]
84: 1043.4782608695652 [Hz]
85: 1116.2790697674418 [Hz]
86: 1170.7317073170732 [Hz]
87: 1230.7692307692307 [Hz]
88: 1333.3333333333333 [Hz]
89: 1411.764705882353 [Hz]
90: 1500.0 [Hz]
91: 1548.3870967741937 [Hz]
92: 1655.1724137931035 [Hz]
93: 1777.7777777777778 [Hz]
94: 1846.1538461538462 [Hz]
95: 2000.0 [Hz]
96: 2086.9565217391305 [Hz]
97: 2181.818181818182 [Hz]
98: 1170.7317073170732 [Hz]
99: 2526.315789473684 [Hz]
100: 2666.6666666666665 [Hz]
101: 2823.529411764706 [Hz]
102: 3000.0 [Hz]
103: 1043.4782608695652 [Hz]
104: 1655.1724137931035 [Hz]
105: 1170.7317073170732 [Hz]
106: 3692.3076923076924 [Hz]
107: 4000.0 [Hz]
108: 2086.9565217391305 [Hz]
```

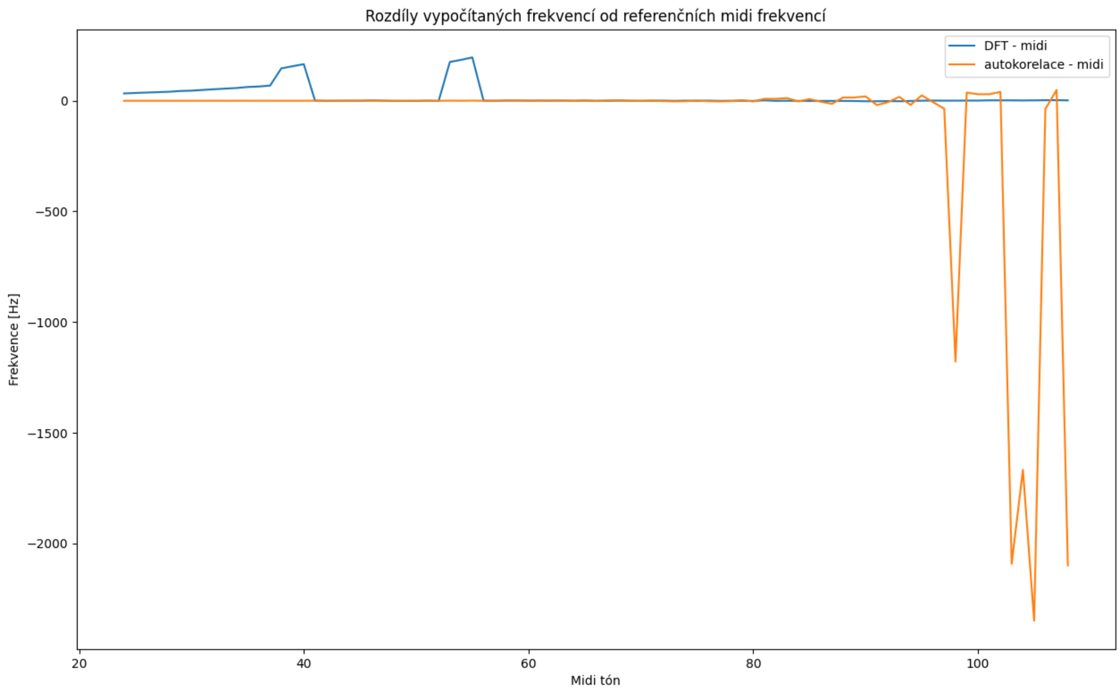
```
/tmp/ipykernel_64719/711234149.py:21: RuntimeWarning: divide by zero encountered
in long_scalars
  fmax = Fs/p
```

6.1 Srovnání vypočítaných základních frekvencí s referenčním midi

[10]:
```
xall_dfts_p = np.array(xall_dfts)[24:]
xall_autocorr_p = xall_autocorr[24:]

t = np.arange(0, xall_dfts_p.size) + 24
#plt.plot(t, abs(xall_autocorr_p), color="orange")
fig = plt.figure(figsize=(8*plt_scale, 3*plt_scale))
plt.plot(t, abs(xall_dfts_p), label="pomocí DFT")
plt.plot(t, xall_autocorr_p, label="pomocí autokorelace")
plt.plot(t, midi[:,1], label="referenční midi soubor")
plt.title("Vypočítané základní frekvence")
plt.xlabel("Midi tón")
plt.ylabel("Frekvence [Hz]")
plt.legend()
plt.savefig(img_gen_folder + "dft_autocorr_vs_midi.png")
```



[11]:
```
fig = plt.figure(figsize=(5*plt_scale, 3*plt_scale))
plt.plot(t, abs(xall_dfts_p) - midi[:,1], label="DFT - midi")
plt.plot(t, xall_autocorr_p - midi[:,1], label="autokorelace - midi")
plt.title("Rozdíly vypočítaných frekvencí od referenčních midi frekvencí")
plt.xlabel("Midi tón")
plt.ylabel("Frekvence [Hz]")
plt.legend()
plt.savefig(img_gen_folder + "computed_vs_midi.png")
```



Z grafů je patrné nevýhod jednotlivých metod.

Autokorelace dosahuje větších odchylek u vyšších tónů, kde naopak DFT dosahuje frekvencí bliších referenčním z midi souboru. U nižších tónů je situace opačná, DFT zde dosahuuje větších odchylek oproti autokorelaci.

Pro získání základních frekvencí s nejmenší odchylkou od referenčních frekvencí je tedy vhodné použít kombinaci těchto metod.

6.2 Výběr frekvencí podle metody s menší odchylkou od referenčního midi

[12]:
```
xall_computed_freqs_ac_dft = np.zeros(np.array(xall_dfts).size)
print("index | midi [Hz]| DFT (odchylka) [Hz]| AC (odchylka) [Hz]| Metoda s␣
↪menší odchylkou od midi | ")
print("-|" * 5)
i = 24
for ac, dft, midi_ref, odchylka_ac, odchylka_dft in zip(xall_autocorr_p,␣
↪xall_dfts_p, midi[:,1], abs(xall_autocorr_p - midi[:,1]), abs(np.
↪abs(xall_dfts_p) - midi[:,1])):
    print(f"{i} | {midi_ref} |  {dft:.3f} ({odchylka_dft:.3f}) | {ac:.3f}␣
↪({odchylka_ac:.3f}) | ", end='')
    if (odchylka_ac > odchylka_dft):
        print("DFT")
        xall_computed_freqs_ac_dft[i] = dft
    else:
        print("AC")
```

```
        xall_computed_freqs_ac_dft[i] = ac
    i+=1
```

index	midi [Hz]	DFT (odchylka) [Hz]	AC (odchylka) [Hz]	Metoda s menší odchylkou od midi
24	32.7	66.000 (33.300)	32.809 (0.109)	AC
25	34.65	70.000 (35.350)	34.757 (0.107)	AC
26	36.71	74.000 (37.290)	36.838 (0.128)	AC
27	38.89	78.000 (39.110)	39.024 (0.134)	AC
28	41.2	82.000 (40.800)	41.344 (0.144)	AC
29	43.65	88.000 (44.350)	43.796 (0.146)	AC
30	46.25	92.000 (45.750)	46.422 (0.172)	AC
31	49.0	98.000 (49.000)	49.180 (0.180)	AC
32	51.91	104.000 (52.090)	52.117 (0.207)	AC
33	55.0	110.000 (55.000)	55.172 (0.172)	AC
34	58.27	116.000 (57.730)	58.465 (0.195)	AC
35	61.74	124.000 (62.260)	61.935 (0.195)	AC
36	65.41	130.000 (64.590)	65.574 (0.164)	AC
37	69.3	138.000 (68.700)	69.465 (0.165)	AC
38	73.42	220.000 (146.580)	73.620 (0.200)	AC
39	77.78	234.000 (156.220)	77.922 (0.142)	AC
40	82.41	248.000 (165.590)	82.616 (0.206)	AC
41	87.31	88.000 (0.690)	87.751 (0.441)	AC
42	92.5	92.000 (0.500)	92.843 (0.343)	AC
43	98.0	98.000 (0.000)	98.361 (0.361)	DFT
44	103.83	104.000 (0.170)	104.348 (0.518)	DFT
45	110.0	110.000 (0.000)	110.599 (0.599)	DFT
46	116.54	118.000 (1.460)	117.073 (0.533)	AC
47	123.47	124.000 (0.530)	123.711 (0.241)	AC
48	130.81	130.000 (0.810)	131.148 (0.338)	AC
49	138.59	138.000 (0.590)	138.728 (0.138)	AC
50	146.83	146.000 (0.830)	147.239 (0.409)	AC
51	155.56	156.000 (0.440)	155.844 (0.284)	AC
52	164.81	164.000 (0.810)	164.948 (0.138)	AC
53	174.61	350.000 (175.390)	175.182 (0.572)	AC
54	185.0	370.000 (185.000)	185.328 (0.328)	AC
55	196.0	392.000 (196.000)	196.721 (0.721)	AC
56	207.65	208.000 (0.350)	207.792 (0.142)	AC
57	220.0	220.000 (0.000)	220.183 (0.183)	DFT
58	233.08	234.000 (0.920)	234.146 (1.066)	DFT
59	246.94	248.000 (1.060)	247.423 (0.483)	AC
60	261.63	262.000 (0.370)	262.295 (0.665)	DFT
61	277.18	278.000 (0.820)	277.457 (0.277)	AC
62	293.66	294.000 (0.340)	294.479 (0.819)	DFT
63	311.13	312.000 (0.870)	311.688 (0.558)	AC
64	329.63	330.000 (0.370)	328.767 (0.863)	DFT

```
65 | 349.23 |  350.000 (0.770) | 350.365 (1.135) | DFT
66 | 369.99 |  370.000 (0.010) | 369.231 (0.759) | DFT
67 | 392.0 |  392.000 (0.000) | 393.443 (1.443) | DFT
68 | 415.3 |  416.000 (0.700) | 417.391 (2.091) | DFT
69 | 440.0 |  440.000 (0.000) | 440.367 (0.367) | DFT
70 | 466.16 |  466.000 (0.160) | 466.019 (0.141) | AC
71 | 493.88 |  494.000 (0.120) | 494.845 (0.965) | DFT
72 | 523.25 |  524.000 (0.750) | 521.739 (1.511) | DFT
73 | 554.37 |  554.000 (0.370) | 551.724 (2.646) | DFT
74 | 587.33 |  588.000 (0.670) | 585.366 (1.964) | DFT
75 | 622.25 |  622.000 (0.250) | 623.377 (1.127) | DFT
76 | 659.26 |  660.000 (0.740) | 657.534 (1.726) | DFT
77 | 698.46 |  698.000 (0.460) | 695.652 (2.808) | DFT
78 | 739.99 |  740.000 (0.010) | 738.462 (1.528) | DFT
79 | 783.99 |  784.000 (0.010) | 786.885 (2.895) | DFT
80 | 830.61 |  830.000 (0.610) | 827.586 (3.024) | DFT
81 | 880.0 |  882.000 (2.000) | 888.889 (8.889) | DFT
82 | 932.33 |  932.000 (0.330) | 941.176 (8.846) | DFT
83 | 987.77 |  988.000 (0.230) | 1000.000 (12.230) | DFT
84 | 1046.5 |  1046.000 (0.500) | 1043.478 (3.022) | DFT
85 | 1108.73 |  1108.000 (0.730) | 1116.279 (7.549) | DFT
86 | 1174.66 |  1174.000 (0.660) | 1170.732 (3.928) | DFT
87 | 1244.51 |  1244.000 (0.510) | 1230.769 (13.741) | DFT
88 | 1318.51 |  1318.000 (0.510) | 1333.333 (14.823) | DFT
89 | 1396.91 |  1396.000 (0.910) | 1411.765 (14.855) | DFT
90 | 1479.98 |  1478.000 (1.980) | 1500.000 (20.020) | DFT
91 | 1567.98 |  1566.000 (1.980) | 1548.387 (19.593) | DFT
92 | 1661.22 |  1660.000 (1.220) | 1655.172 (6.048) | DFT
93 | 1760.0 |  1758.000 (2.000) | 1777.778 (17.778) | DFT
94 | 1864.66 |  1864.000 (0.660) | 1846.154 (18.506) | DFT
95 | 1975.53 |  1976.000 (0.470) | 2000.000 (24.470) | DFT
96 | 2093.0 |  2094.000 (1.000) | 2086.957 (6.043) | DFT
97 | 2217.46 |  2218.000 (0.540) | 2181.818 (35.642) | DFT
98 | 2349.32 |  2350.000 (0.680) | 1170.732 (1178.588) | DFT
99 | 2489.02 |  2490.000 (0.980) | 2526.316 (37.296) | DFT
100 | 2637.02 |  2638.000 (0.980) | 2666.667 (29.647) | DFT
101 | 2793.83 |  2796.000 (2.170) | 2823.529 (29.699) | DFT
102 | 2959.96 |  2962.000 (2.040) | 3000.000 (40.040) | DFT
103 | 3135.96 |  3138.000 (2.040) | 1043.478 (2092.482) | DFT
104 | 3322.44 |  3324.000 (1.560) | 1655.172 (1667.268) | DFT
105 | 3520.0 |  3522.000 (2.000) | 1170.732 (2349.268) | DFT
106 | 3729.31 |  3732.000 (2.690) | 3692.308 (37.002) | DFT
107 | 3951.07 |  3954.000 (2.930) | 4000.000 (48.930) | DFT
108 | 4186.01 |  4188.000 (1.990) | 2086.957 (2099.053) | DFT
```

7 Zpřesnění odhadu základní frekvence f0

Implementace DTFT převzata z dostupných zdrojů na webu ISS.

Hodnoty `range` a `points` byly vybrány manuálně opakovanými pokusy o zpřesnění výstupu.

[13]:
```
# převzato z dostupných python notebooků na stránkách ISS
# >>>>>>>>>>>>>>>>> #dtft <<<<<<<<<<<<<<<<<<<<<<<<<
x = xall[80]
# do the DTFT
def DTFT(tone, f, N=24000, Fs=48000, range=4, points=500):
    FREQRANGE = range
    FREQPOINTS = points

    n = np.arange(0,N)
    fsweep = np.linspace(f-FREQRANGE, f+FREQRANGE,FREQPOINTS)
    A = np.zeros([FREQPOINTS, N],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,tone.T)
    Xmax = np.argmax(np.abs(Xdtft))
    precisefmax = fsweep[Xmax]
    return Xdtft, precisefmax, fsweep, Xmax
```

7.1 Srovnání metody DTFT, metody s nejmenší odchylkou od midi a midi samotného

Jako vstupní odhad základní frekvence pro DTFT používáme frekvence získané kombinací autokorelace a DFT podle menší odchylky od referenčního midi.

[14]:
```
xall_dtft = np.zeros((xall_computed_freqs_ac_dft.size))
i = 0
for f, tone in zip(xall_computed_freqs_ac_dft[24:], xall[24:]):
    tone_sound = tone
    Xdtft, precisefmax, fsweep, xmax = DTFT(tone_sound, f)
    print(f"{i+24}: comp={f:.3f} dtft={precisefmax:.4f} midi={midi[:,1][i]}")
    xall_dtft[i+24] = precisefmax
    i+=1
```

```
24: comp=32.809 dtft=32.1600 midi=32.7
25: comp=34.757 dtft=32.5370 midi=34.65
26: comp=36.838 dtft=36.1888 midi=36.71
27: comp=39.024 dtft=38.3270 midi=38.89
28: comp=41.344 dtft=39.2355 midi=41.2
29: comp=43.796 dtft=44.3648 midi=43.65
30: comp=46.422 dtft=43.5279 midi=46.25
31: comp=49.180 dtft=46.7996 midi=49.0
32: comp=52.117 dtft=54.4660 midi=51.91
```

```
33: comp=55.172 dtft=55.5973 midi=55.0
34: comp=58.465 dtft=59.3871 midi=58.27
35: comp=61.935 dtft=61.9756 midi=61.74
36: comp=65.574 dtft=65.3413 midi=65.41
37: comp=69.465 dtft=69.0718 midi=69.3
38: comp=73.620 dtft=72.9543 midi=73.42
39: comp=77.922 dtft=77.3048 midi=77.78
40: comp=82.616 dtft=81.9669 midi=82.41
41: comp=87.751 dtft=87.6953 midi=87.31
42: comp=92.843 dtft=92.9315 midi=92.5
43: comp=98.000 dtft=98.4569 midi=98.0
44: comp=104.000 dtft=104.2966 midi=103.83
45: comp=110.000 dtft=110.5050 midi=110.0
46: comp=117.073 dtft=117.0652 midi=116.54
47: comp=123.711 dtft=123.1903 midi=123.47
48: comp=131.148 dtft=130.5143 midi=130.81
49: comp=138.728 dtft=138.2714 midi=138.59
50: comp=147.239 dtft=146.6862 midi=146.83
51: comp=155.844 dtft=155.3872 midi=155.56
52: comp=164.948 dtft=164.6038 midi=164.81
53: comp=175.182 dtft=174.5172 midi=174.61
54: comp=185.328 dtft=184.8873 midi=185.0
55: comp=196.721 dtft=195.8957 midi=196.0
56: comp=207.792 dtft=207.8002 midi=207.65
57: comp=220.000 dtft=220.1523 midi=220.0
58: comp=234.000 dtft=233.2385 midi=233.08
59: comp=247.423 dtft=247.0780 midi=246.94
60: comp=262.000 dtft=261.7675 midi=261.63
61: comp=277.457 dtft=277.3364 midi=277.18
62: comp=294.000 dtft=293.6232 midi=293.66
63: comp=311.688 dtft=311.0871 midi=311.13
64: comp=330.000 dtft=329.5912 midi=329.63
65: comp=350.000 dtft=349.1423 midi=349.23
66: comp=370.000 dtft=369.8958 midi=369.99
67: comp=392.000 dtft=391.8798 midi=392.0
68: comp=416.000 dtft=415.4469 midi=415.3
69: comp=440.000 dtft=440.1363 midi=440.0
70: comp=466.019 dtft=466.2679 midi=466.16
71: comp=494.000 dtft=493.8317 midi=493.88
72: comp=524.000 dtft=523.1904 midi=523.25
73: comp=554.000 dtft=554.2966 midi=554.37
74: comp=588.000 dtft=587.1904 midi=587.33
75: comp=622.000 dtft=622.1042 midi=622.25
76: comp=660.000 dtft=659.0942 midi=659.26
77: comp=698.000 dtft=697.5591 midi=698.46
78: comp=740.000 dtft=739.0782 midi=739.99
79: comp=784.000 dtft=783.0942 midi=783.99
80: comp=830.000 dtft=829.6072 midi=830.61
```

```
81: comp=882.000 dtft=878.9940 midi=880.0
82: comp=932.000 dtft=931.3507 midi=932.33
83: comp=988.000 dtft=987.6072 midi=987.77
84: comp=1046.000 dtft=1046.1523 midi=1046.5
85: comp=1108.000 dtft=1108.3126 midi=1108.73
86: comp=1174.000 dtft=1174.2164 midi=1174.66
87: comp=1244.000 dtft=1244.1042 midi=1244.51
88: comp=1318.000 dtft=1318.1683 midi=1318.51
89: comp=1396.000 dtft=1395.9279 midi=1396.91
90: comp=1478.000 dtft=1478.9379 midi=1479.98
91: comp=1566.000 dtft=1566.9058 midi=1567.98
92: comp=1660.000 dtft=1660.0561 midi=1661.22
93: comp=1758.000 dtft=1758.9539 midi=1760.0
94: comp=1864.000 dtft=1863.6072 midi=1864.66
95: comp=1976.000 dtft=1976.4729 midi=1975.53
96: comp=2094.000 dtft=2093.9920 midi=2093.0
97: comp=2218.000 dtft=2218.5050 midi=2217.46
98: comp=2350.000 dtft=2350.6493 midi=2349.32
99: comp=2490.000 dtft=2490.4248 midi=2489.02
100: comp=2638.000 dtft=2638.5210 midi=2637.02
101: comp=2796.000 dtft=2795.0621 midi=2793.83
102: comp=2962.000 dtft=2961.2545 midi=2959.96
103: comp=3138.000 dtft=3137.3828 midi=3135.96
104: comp=3324.000 dtft=3324.0240 midi=3322.44
105: comp=3522.000 dtft=3521.6874 midi=3520.0
106: comp=3732.000 dtft=3731.1423 midi=3729.31
107: comp=3954.000 dtft=3953.1904 midi=3951.07
108: comp=4188.000 dtft=4188.2485 midi=4186.01
```

Ve výpisu lze vidět, že DTFT dosahuje oproti kombinaci DFT a autokorelace vyšších odchylek od referenčního midi u nižších tónů.

Předpokádáme že nějaké odchylky od referenčního midi se ve vypočítaných frekvencích budou vyskytovat, kvůli drobným nepřesnostem výpočtu či možného “rozladění” klavíru ve vstupní nahrávce použité pro výpočet frekvencí může k takovýmto odchylkám dojít.

8 Reprezentace klavíru 10 FP čísly

[15]:
```python
xall_computed_frequencies = xall_dtft[24:] # pro koeficienty použijeme frekvence␣
↪zjištěné z DTFT
times_DTFT = 10 # získáme 10 modulů
def DTFT_n_times(f, sound, n=times_DTFT):
    modules = list()
    phases = list()
    freqs = list()
    dtfts = list()
    for xdtft, fmax, fsweep, xmax in [DTFT(sound, f*multiple) for multiple in␣
↪range(1, n+1)]:
```

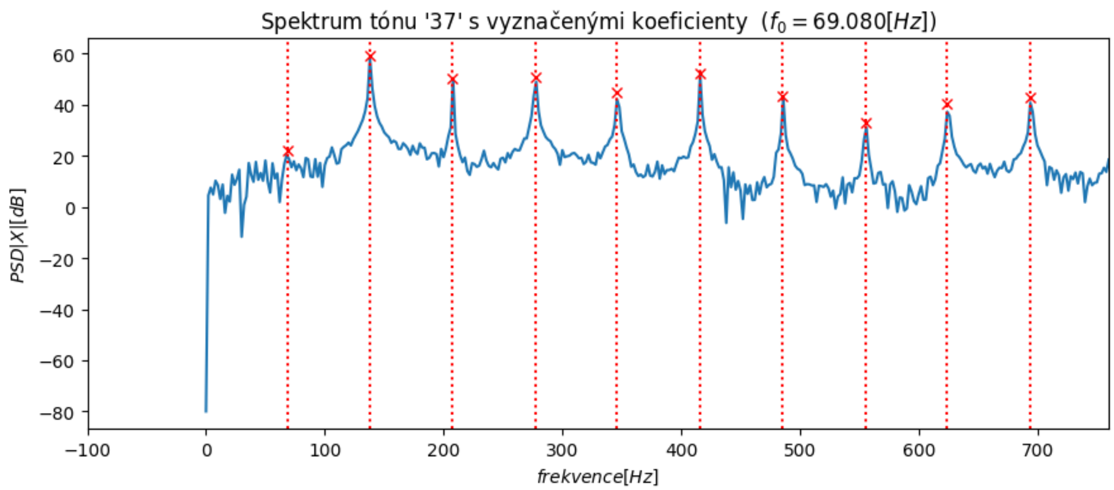
```
        dtfts.append(xdtft)
        modules.append(np.abs(xdtft[xmax]))
        phases.append(np.angle(xdtft[xmax]))
        freqs.append(fmax)
    return np.array(modules), np.array(phases), np.array(freqs), np.array(dtfts)

# spouštění DTFT paralelně pro zrychlení
import multiprocessing
from contextlib import closing
with closing(multiprocessing.Pool(maxtasksperchild=1)) as pool:
    keyboard_repr = pool.starmap(DTFT_n_times, zip(xall_computed_frequencies,␣
 ↪xall[24:]))
```

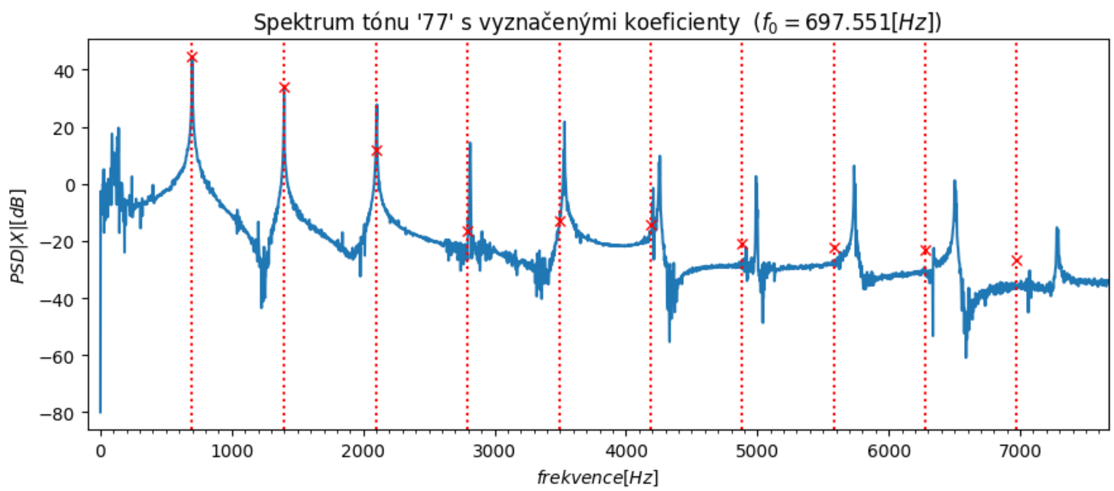
[16]:
```
# naplnění dále používaných struktur pro moduly, fáze, frekvence a dtft z␣
 ↪vypočítané reprezentace klavíru
i = 24
modules = np.zeros((xall.size, times_DTFT))
phases = np.zeros((xall.size, times_DTFT))
freqs = np.zeros((xall.size, times_DTFT))
dtfts = list()
for tone_repr in keyboard_repr:
    modules[i] = tone_repr[0]
    phases[i] = tone_repr[1]
    freqs[i] = tone_repr[2]
    dtfts.append(tone_repr[3])
    i+=1
```

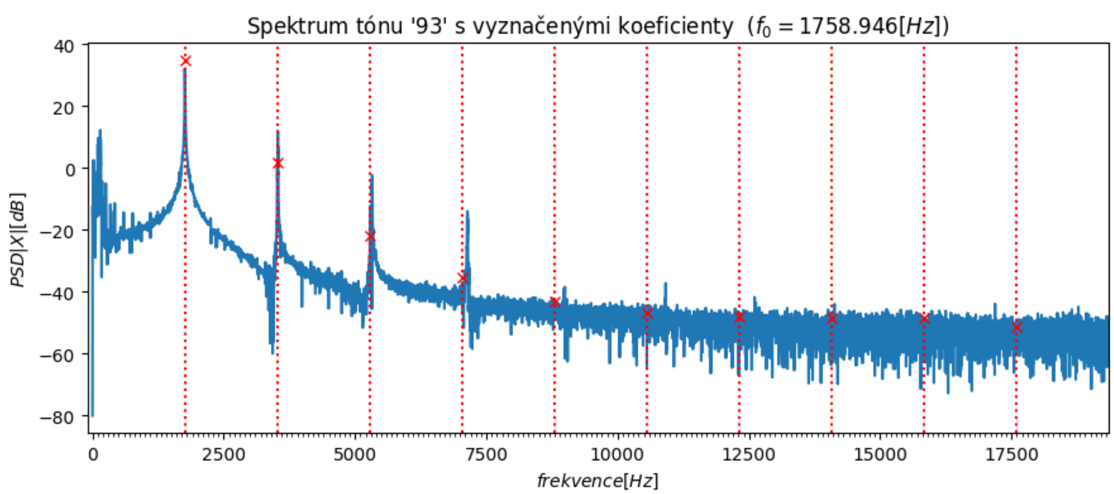
[17]:
```
# vykreslení spektra s koeficienty
fig, (a_plots, b_plots, c_plots) = plt.subplots(3, 1, figsize=(3*plt_scale,␣
 ↪4*plt_scale))
def plot_tone_coeff(tone, id, axis, zakladni_f):
    fft_of_tone = np.abs(np.fft.fft(tone)[:tone.size])
    t_dft = np.arange(fft_of_tone.size) * 2
    PSD = 10 * np.log10(np.abs(fft_of_tone)**2 + (0.00000001))

    axis.set_ylabel(f"$PSD |X| [dB]$")
    axis.set_xlabel(f"$frekvence [Hz]$")
    axis.set_title(f"Spektrum tónu \'{id}\' s vyznačenými koeficienty  ($f_0 =␣
 ↪{zakladni_f:0.3f} [Hz]$)")
    axis.set_xticks(np.arange(0, len(fft_of_tone), 100), minor=True)
    axis.set_xlim(-100, (freqs[id][0]*11))
    axis.plot(t_dft, PSD)

    # vyznačení koeficientů ve spektru
    for f, m in zip(freqs[id], modules[id]):
        point_x = int(f)
        point_y = 10 * np.log10(m**2 + (0.0000000000000001))
```

```
        axis.plot(point_x, point_y, "rx")
        axis.axvline(point_x, linestyle="dotted", color="r") # pro lepší␣
↪zvýraznění bodů
        print(f"({point_x},{point_y:.2f}), ", end="")
    print("")
    return fft_of_tone

plot_tone_coeff(a, my_tones[0], a_plots, freqs[my_tones[0]][0])
plot_tone_coeff(b, my_tones[1], b_plots, freqs[my_tones[1]][0])
plot_tone_coeff(c, my_tones[2], c_plots, freqs[my_tones[2]][0])

fig.tight_layout()
fig.savefig(img_gen_folder + "my_tones_spectrum_coeffs_paralell.png")
```

```
(69,22.08), (138,58.98), (207,50.15), (277,50.70), (346,44.63), (416,52.01),
(485,43.10), (555,33.06), (624,40.38), (694,43.07),
(697,44.47), (1398,33.86), (2096,11.87), (2788,-16.61), (3491,-12.98),
(4188,-14.06), (4880,-20.73), (5582,-22.20), (6278,-23.16), (6972,-26.57),
(1758,34.61), (3519,1.73), (5280,-21.90), (7038,-35.58), (8792,-43.15),
(10554,-46.71), (12314,-47.93), (14072,-48.36), (15830,-48.52), (17585,-51.25),
```


Spektrum tónu '37' s vyznačenými koeficienty ($f_0 = 69.080[Hz]$)
PSD|X|[dB]
frekvence[Hz]


Spektrum tónu '77' s vyznačenými koeficienty ($f_0 = 697.551[Hz]$)
PSD|X|[dB]
frekvence[Hz]


Spektrum tónu '93' s vyznačenými koeficienty ($f_0 = 1758.946[Hz]$)
PSD|X|[dB]
frekvence[Hz]

9 Syntéza tónů

[22]:
```
def get_synt(tone_id):
    X = np.zeros(N*2, dtype=complex)
    x = np.zeros(N)
    findexs = np.rint(freqs[tone_id]).astype(int) # indexy koeficientu, na ktere␣
 ↪se vlozi vypocitane moduly
    X[findexs] = X[-findexs]= modules[tone_id] #- 1j*phases[toneid]
    xs_nn = np.real(np.fft.ifft(X, n=Fs))
    norm_xs = xs_nn/max(xs_nn) # třeba ještě normalizovat...
    xs = norm_xs*max(xall[tone_id]) # a dostat na správnou výšku
    return xs, X

a_synt, Xa = get_synt(my_tones[0])
b_synt, Xb = get_synt(my_tones[1])
c_synt, Xc = get_synt(my_tones[2])
```

[23]:
```
print("Originální tón a")
display(Audio(xall[my_tones[0]], rate=Fs))
print("Syntetizovaný tón a")
display(Audio(a_synt, rate=Fs))
print("-"*30)
print("Originální tón b")
display(Audio(xall[my_tones[1]], rate=Fs))
print("Syntetizovaný tón b")
display(Audio(b_synt, rate=Fs))
print("-"*30)
print("Originální tón b")
display(Audio(xall[my_tones[2]], rate=Fs))
print("Syntetizovaný tón c")
display(Audio(c_synt, rate=Fs))
```

```
Originální tón a

<IPython.lib.display.Audio object>

Syntetizovaný tón a

<IPython.lib.display.Audio object>

------------------------------
Originální tón b

<IPython.lib.display.Audio object>

Syntetizovaný tón b

<IPython.lib.display.Audio object>

------------------------------
Originální tón b
```

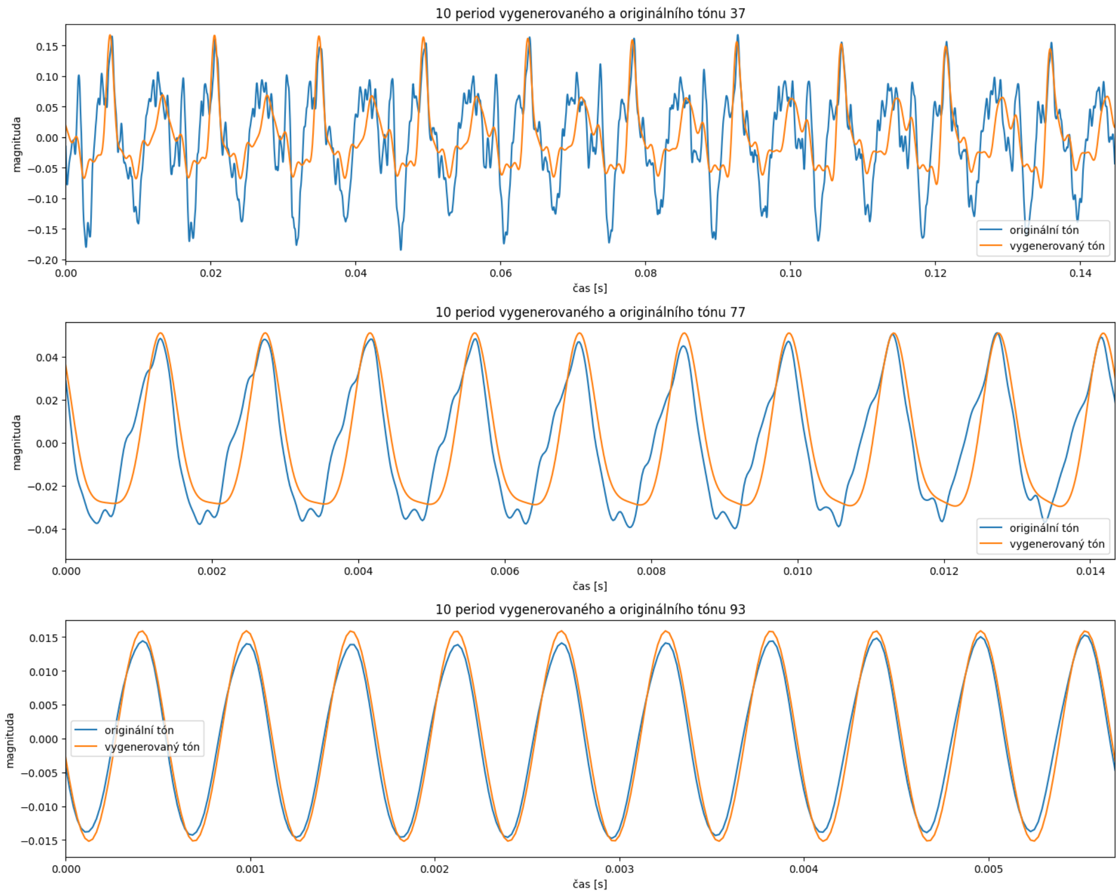
```
<IPython.lib.display.Audio object>
```

```
Syntetizovaný tón c
```

```
<IPython.lib.display.Audio object>
```

[24]:

```
fig, (a_plots, b_plots, c_plots) = plt.subplots(3, 1, figsize=(5*plt_scale,␣
↪4*plt_scale))
def plot_tone_synt(tone, synt, id, axis, zakladni_f, periods=10, posun=0):
    # compute n periods
    T = 1/zakladni_f # in [seconds]
    t = (np.arange(tone.size) / Fs)

    axis.set_ylabel(f"magnituda")
    axis.set_xlabel(f"čas [s]")
    axis.set_xlim(0, periods*T)
    axis.plot(t, tone, label="originální tón")
    axis.plot(t+posun, synt, label="vygenerovaný tón")
    axis.legend()
    axis.set_title(f"{periods} period vygenerovaného a originálního tónu {id}")

# plotneme naše vygenerovane tony s manuálním posunem pro každý z nich
synt_size = int(a_synt.size/2)
plot_tone_synt(a, a_synt[:synt_size], my_tones[0], a_plots,␣
↪freqs[my_tones[0]][0], posun=-0.0083)
plot_tone_synt(b, b_synt[:synt_size], my_tones[1], b_plots,␣
↪freqs[my_tones[1]][0], posun=-0.003)
plot_tone_synt(c, c_synt[:synt_size], my_tones[2], c_plots,␣
↪freqs[my_tones[2]][0], posun=-0.003)

fig.tight_layout()
fig.savefig(img_gen_folder + "original_tones_vs_generated_tones.png")
```



Vygenerovaný tón je mnohem hladší, postrádá vysokofrekvenční detaily, i přesto si však zachovává velmi podobný tvar původnímu tónu.

Vyhlazení je důsledkem redukovaného množství dat, které jsme pro syntézu použili.

[25]:
```
# uložíme naše 1s tóny
for orig_tone, orig_name in zip([a_synt, b_synt, c_synt], ["a", "b", "c"]):
    sf.write(audio_folder + f"{orig_name}.wav", orig_tone, Fs)
```

Zdroje: - https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.ifft.html - https://www.fit.vutbr.cz/study/courses/ISS/public/NEW_PRED/02_spectral_analysis_1/ - https://www.fit.vutbr.cz/study/courses/ISS/public/NEW_PRED/03_spectral_analysis_2/