
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS


# ZVUKOVÁ SYNTÉZA V REÁLNÉM ČASE V PROSTŘEDÍ MATLAB

REAL-TIME SOUND SYNTHESIS IN MATLAB

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS


AUTOR PRÁCE — Vojtěch Kovanda
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Jiří Schimmel, Ph.D.
SUPERVISOR



BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Bakalářská práce

bakalářský studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

***Student:*** Vojtěch Kovanda ***ID:*** 209409
***Ročník:*** 3 ***Akademický rok:*** 2021/22

**NÁZEV TÉMATU:**

## Zvuková syntéza v reálném čase v prostředí Matlab

**POKYNY PRO VYPRACOVÁNÍ:**

Prostudujte možnosti Audio toolboxu pro prostředí Matlab a s jeho pomocí, případně s pomocí dalších dostupných nástrojů, vytvořte prostředí pro jednoduchou implementaci algoritmů číslicové syntézy zvukových signálů v reálném čase. Prostředí bude umožňovat řízení parametrů syntézy pomocí protokolu MIDI, polyfonii, tvorbu jednoduchého grafického rozhraní, spektrální analýzu výstupního signálu v reálném čase a výstup na zvukovou kartu a do zvukového souboru. Vytvořené prostředí otestujte na základních metodách syntézy zvuku, vytvořte k němu dokumentaci a tutoriál demonstrující tvorbu jednoduchého syntezátoru.

**DOPORUČENÁ LITERATURA:**

[1] RUSS, M., Sound Synthesis and Sampling. Focal Press, 1996. ISBN 0-240-51429-7
[2] GIANNKOPOULOS, T., PIKRAKIS, A., Introduction to Audio Analysis: A MATLAB Approach, 1st ed. Academic Press, 2014. ISBN 978-0080993881

***Termín zadání:*** 7.2.2022 ***Termín odevzdání:*** 31.5.2022

***Vedoucí práce:*** doc. Ing. Jiří Schimmel, Ph.D.

**doc. Ing. Jiří Schimmel, Ph.D.**
předseda rady studijního programu

# Abstrakt

Cíle této práce jsou vytvořit program používající digitální zvukovou syntézu, kterou lze přehrát v reálném čase, dále vytvořit prostředí vhodné k implementaci digitální zvukové syntézy v reálném čase řízenou protokolem MIDI a tutoriál k vytvoření jednoduchého syntezátoru v tomto prostředí. Práce vysvětluje teorii k základním metodám zvukové syntézy. Popisuje protokol MIDI a práci se zvukem v prostředí Matlab. Ukazuje základní metody digitální zvukové syntézy v tomto prostředí: vytvoření základních průběhů jako je například sinusovka nebo obdélník, aditivní syntézu, filtraci nebo ořezání signálu. Představuje princip pro řízení zvukové syntézy v reálném čase skrz protokol MIDI. Dále představuje prostředí vhodné k vytvoření následné syntézy naprogramované jako audioplugin, které umožňuje přehrání v reálném čase a nabízí řízení parametrů syntézy protokolem MIDI. Také umožňuje zápis na zvukovou kartu a do souboru a zobrazení průběhu a frekvenčního spektra v reálném čase. Jako součást práce byl vytvořen audioplugin, který funguje jako jednoduchý syntézátor využívající více základních druhů digitální zvukové syntézy a audioplugin, který je prostředím pro implementaci digitální zvukové syntézy. K tomuto prostředí byl vytvořen i tutoriá jako návod pro výrobu jednoduchého syntezátoru.

# Abstract

Goals of this paper are to create a programm which is able to use digital sound synthesis and play created signals in real time. Another goal is to create an environment for implementation real-time digital sound synthesis in Matlab controlled by MIDI protocol and to create tutorial how to programme a simple synthesizer in that environment. The paper shows basic theory for sound syntehsis methods. It describes MIDI protocol and how to process sounds in Matlab. It shows basic methods of sound synthesis in Matlab such as waveshaping, additive synthesis or subtractive synthesis. It presents ways how to achieve real-time synthesis controlled by external MIDI instrument. It presents programmable enviroment for real-time digital sound synthesis which works as Matlab audioplugin. The environment offers to play created sound signals in real time and controll parametres of synthesis by MIDI protocol. It also offers write signals down to file and show signal and its frequency spectrum in real time. As a part of this thesis an audioplugin was made which works as synthesizer using basic methods of digital sound synthesis. Another audioplugin was made as the environment for implementation digital sound synthesis. This audioplugin includes tutorial how to create simple synthesizer in this environment.

# Klíčová slova

Audioplugin, digitální zvuková syntéza, Matlab, metody zvukové syntézy, protokol MIDI, zpracování zvuku

# Keywords

Audioplugin, digital sound synthesis, Matlab, sound synthesis methods, MIDI protocol, sound processing

# Bibliografická citace

KOVANDA, Vojtěch. *Zvuková syntéza v reálném čase v prostředí Matlab.* Brno, 2022. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/141288. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Jiří Schimmel.

# Prohlášení autora o původnosti díla

| | |
|---|---|
| **Jméno a příjmení studenta:** | *Vojtěch Kovanda* |
| **VUT ID studenta:** | *209409* |
| **Typ práce:** | *Bakalářská práce* |
| **Akademický rok:** | *2021/22* |
| **Téma závěrečné práce:** | *Zvuková syntéza v reálném čase v prostředí Matlab* |

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.
V Brně dne: 27. května 2022

# Obsah

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# SEZNAM VÝPISŮ POČÍTAČOVÝCH KÓDŮ

# Úvod

Tato práce se zabývá problematikou číslicové zvukové syntézy, tvorbou umělého zvuku a jeho přehráním v reálném čase s využitím prostředí Matlab. Cílem práce je vytvořit prostředí vhodné pro implementaci algoritmů číslicové zvukové syntézy v reálném čase s proměnnými parametry řízenými na základě protokolu MIDI. Prostředí má dále graficky analyzovat vytvořené zvukové signály a jejich spektra v reálném čase, následné má umožnovat výstup na zvukovou kartu a do zvukového souboru.

V rámci semestrální práce byla v Matlabu vytvořena funkce generující zvukové signály v reálném čase využívající jednoduché číslicové syntézy, která je řízena externími klávesy, které využívají protokol MIDI. V rámci bakalářské práce se v Matlabu využívá audioplugin, jako šablona pro realizaci různé zvukové syntézy v reálném čase.

Práce je členěna do tří částí. První kapitola představuje teorii potřebnou k dosažení stanovených cílů práce, jako jsou témata základní metody zvukové syntézy, protokol MIDI a práce se zvukem v Audiotoolboxu v prostředí Matlab. Ve druhé kapitole je popsáno použití zmíněných prostředků pro zvukovou syntézu v reálném čase, dále jsou v ní popsány vytvořené funkce a dokumenty řešící tuto problematiku. Poslední kapitola shrnuje dosažné výsledky práce.

# 1. TEORETICKÝ ÚVOD

## 1.1 Zvuková syntéza

Následující text popisuje, co je to zvuková syntéza, představuje její základní metody a jejich principy podle zdroje [1].

Zvuková syntéza je proces, který vytváří zvuk. Může generovat zcela nové zvuky elektronicky nebo mechanicky anebo zpracovávat již existující zvuk. Zařízení, které slouží ke zvukové syntéze, se potom nazývá syntezátor. V širším pojetí je možné považovat za syntezátor každý klasický hudební nástroj, protože každý klasický hudební nástroj je zařízení mechanicky generující zvuk a dále využívající určitých metod syntézy pro vytvoření jedinečné barvy zvuku. Nicméně syntezátor se spíše chápe jako zařízení pracující se zvukem elektronicky, nebo jako zařízení vytvářející syntetický zvuk, ve smyslu uměle vytvořený. U syntezátorů se pro řízení výsledku syntézy nejčastěji používá klaviatura.

Zvuky vytvořené zvukovou syntézou se dají rozdělit do několika skupin podle zdroje [1]:

- **Imitace a emulace** – jsou imitací nebo zastoupením reálných nástrojů tak, aby zněly co nejpodobněji. Slouží k nahrazení nástroje například v polohách, kde není možné nástroj použít, nebo v ladění, které nahrazenému nástroji nevyhovuje.
- **Návrhy a náznaky** – mají společné rysy s určitými hudebními nástroji, ale je zde výrazná jejich odlišnost. Například syntetické žestě, jako zvuky často používané v popové hudbě 70. a 80. let minulého století. Svojí barvou připomínají nástroje spadající do kategorie žestě, ale zároveň se svým zvukem výrazně liší od reálných nástrojů.
- **„Alien“ a „off-the-wall“** – synonyma pro toto pojmenování by mohlo být zvuky mimozemské, divné nebo velmi odlišné. Bývají úplně uměle vytvořené. Svojí barvou nepřipomínají známé hudební nástroje, ale připomínají je charakteristikou vytvořených zvuků: stálost barvy zvuku při změně výšky tónu; změny vyšších harmonických frekvencí v čase.
- **„Noise-like“** – jsou zvuky pracující s variacemi šumu. Šumu se přiřazují určité frekvence a stává se tak hratelným. Využívá se například pro kombinaci s jinak znějícími zvuky, které dohromady znějí zajímavěji.
- **„Factory presets“** – jsou zvuky z komerčních zvukových syntezátorů, které seřizuje sám uživatel. Má ukazovat široké možnosti nástroje a využívá například různé efekty, může spojovat zvuky do cyklů v různých variacích nebo pracuje se samotnými samply.

### 1.1.1 Metody zvukové syntézy

Metody zvukové syntézy je možné dělit na analogové, digitální nebo hybridní, kde se kombinují metody jak analogové tak i digitální. Většina základních metod se může provádět digitálně i analogově.

- **Aditivní syntéza** – nebo také součtová, sčítá dílčí signály o různých frekvencích pro vytvoření výsledné barvy tónu. Speciální aditivní metoda je například harmonická syntéza, kde se sčítají pouze harmonické signály, jejichž frekvence jsou násobky základního kmitočtu. Nástroj využívající aditivní syntézy jsou například Hammondovy varhany, které obsahují 8 generátorů harmonických složek v určitém frekvenčním poměru k základnímu tónu.
- **Subtraktivní syntéza** – vezme určitý zvukový signál, obvykle bohatý na spektrální složky, jako je pila, obdélník nebo trojúhelník a filtruje ho tak, aby pozměnil spektrum a tím i barvu výsledného zvuku. Je prováděna buď analogově realizací samotných filtrů, nebo digitálně spočtením přenosové funkce.
- **Modulační syntéza** – výsledný signál je výstupem nosného signálu, který je ovlivňovaná modulačním signálem. Základní modulační metody jsou PWM modulace, kde střída obdélníku je ovlivňována okamžitou hodnotou modulačního signálu, kruhová modulace, kde amplituda nosného signálu je ovlivňována okamžitou hodnotou modulačního signálu, amplitudová modulace, což je kruhová modulace s přičteným nosným signálem a kmitočtová modulace, kde je kmitočet nosného signálu ovlivňován okamžitou hodnotou modulačního signálu.
- **Tvarování vlny** – při této metodě prochází signál oscilátoru systémem s nelineární převodní charakteristikou. Analogově se využívají tvarovače signálu nebo funkční měniče. Dochází k omezení signálu a obohacení výsledného spektra (Soft-clip, Hard-clip), nebo ke změně tvaru vlny.
- **Tabulkové metody** – jsou digitální metody. Přehrávají vzorky nebo průběhy z paměti, se kterými dále pracují. Patří sem Wavetable syntéza, Wavecykle syntéza, Sample & Synthesis.
- **Fyzikální modelování** – analyzuje skutečné nástroje budící zvuk mechanicky a nahrazuje je matematickým modelem. Využívá se zde princip elektromechanické a elektroakustické analogie.

## 1.2 Protokol MIDI

V této kapitole je představen princip a způsob komunikace hudebního komunikačního rozhraní MIDI podle zdroje [1]. Zkratka znamená musical instrument digital interface. MIDI slouží ke komunikaci mezi digitálními hudebními nástroji nebo pro jejich připojení k počítači. Přenášená data nejsou zvukovým signálem, ale pouze řídící data odpovídající událostem během hry na digitální hudební nástroj.

### 1.2.1 Struktura protokolu MIDI

Jednotlivé události, které MIDI přenáší, se nazývají MIDI zprávy. Jedna MIDI zpráva obsahuje jeden stavový byte a několik datových bytů. U každého bytu nejvýznamnější bit určuje, zda jde o stavový nebo datový byte. U stavového bytu jsou další 3 bity vyhrazeny pro identifikátor typu zprávy a poslední 4 bity jsou vyhrazeny pro identifikátor MIDI kanálu. U datových bytů je všech zbývajících 7 bitů vyhrazeno pro samotnou hodnotu související s typem zprávy.

Tabulka 1.1.1 1 Typy MIDI zpráv

| typ MIDI zprávy | význam | ID typu zprávy | počet datových bytů |
|---|---|---|---|
| Note Off | nota vypnuta | 0 | 2 |
| Note On | nota zapnuta | 1 | 2 |
| Polyphonic Key Pressure | individuální tlaková citlivost | 2 | 2 |
| Control Change | změna kontroléru | 3 | 2 |
| Program Change | změna programu | 4 | 1 |
| Channel Pressure | společná tlaková citlivost | 5 | 1 |
| Pitch Bend Change | ohýbání tónu | 6 | 2 |

Tato tabulka je převzatá z přednášky předmětu Studiová technika. Nejzákladnější MIDI zprávy jsou „nota zapnuta“ a „nota vypnuta“ a jejich zápis vypadá takto:

| typ MIDI zprávy | stavový byte | 1. datový byte | 2. datový byte |
|---|---|---|---|
| nota vypnuta | 1001nnnn | 0kkkkkkk | 0vvvvvvv |
| nota zapnuta | 1000nnnn | 0kkkkkkk | 0vvvvvvv |

Písmeno n ve stavovém bytu značí bity, které určují hodnotu kanálu. První datový byte v těchto případech určuje číslo noty sedmi bity označenými písmenem k. Lze tak rozlišit 0-127 půltónů jdoucích od sub-kontra C až po šestičárkované G. Druhý datový byte obsahuje rychlostní data, s jakou dynamikou byla klávesa zahrána. Opět lze rozlišit 0-127 hodnot dynamiky, kdy 0 znamená nota vypnuta, 1 znamená *pianissimo piano*, 64 znamená *mezzo-piano* nebo *mezzo-forte* a 127 *fortissimo forte*.

Důležitou součástí MIDI komunikace jsou potom také kontroléry nesoucí informace například o použitých efektech, modulaci, nebo o použití pedálu. Typů kontroléru je 127 určených prvním datovým bytem.

Kontroléry se dělí:

- 0 – 31: Průběžné kontroléry, jejichž hodnota se zapisuje do 14 bitů. Jejich 2. datový byte nese 7 významnějších bitů určujících hodnotu konkrétního kontroléru.
- 32 – 63: 7 méně významných bitů předchozích 32 různých kontrolérů.
- 64 – 69: Spínače.
- 70 – 95: Kontroléry s hodnotou pouze se sedmi bity.
- 96 – 101: Speciální vyhrazená čísla kontrolérů.
- 102 – 119: Nedefinované jednobytové kontroléry,
- 119 – 127: Povely.

## 1.3 Práce se zvukem v prostředí Matlab

V následující kapitole je popsána práce se zvukem v prostředí Matlab podle zdroje [2]. V Matlabu se pracuje se zvukem jako se zvukovým signálem s diskrétním časem. Audio signál je zde reprezentovaný vektorem reálných čísel jako posloupnost hodnot zvukového signálu, kde jednotlivé hodnoty jsou odděleny stejnými časovými úseky. Zavádí se tak pojem jako vzorkovací perioda nebo vzorkovací frekvence, které jsou ve vztahu $T_S = \frac{1}{fs}$. Významově jde o to, s jakou častostí vzorků je popsán signál. Vzorkovací kmitočet by měl být zvolený minimálně dvojnásobný oproti nejvyšší frekvenci audiosignálu, aby nedošlo k aliasingovému zkreslení. Vyrobení jednoduchého zvukového signálu lze v Matlabu docílit vytvořením určitého počtu vzorků oddělených vzorkovací periodou a přiřazením funkční hodnoty každému vzorku například funkcí sinus.

```
fs = 48000; Ts = 1/fs;

t = 0:Ts:1;

f = 440;
x = sin(2*pi*f*t);
```

Výpis počítačových kódů 1 Generování základního signálu v Matlabu.

V tomto případě je *t* posloupnost vzorků reprezentující časové hodnoty od 0 do 1 sekundy oddělené hodnotou vzorkovací periody $T_S$. Frekvence výsledného signálu je 440Hz. Takto vytvořený signál je možné v Matlabu přehrát pomocí funkce sound(x, fs), která

vezme vytvořený vektor vzorků a se zadanou vzorkovací frekvencí jej pošle na zvukovou kartu zařízení.

Pro práci s wav soubory slouží funkce wavread(). Tato funkce vrací zvukový signál *x*, vzorkovací frekvenci $f_S$, se kterou byl zvukový soubor nahraný nebo vygenerovaný a číslo počtu bitů na vzorek *nBits*. Vstup funkce je jméno zvukového souboru *'test'*.

```
[x, fs, nBits] = wavread('test');
```

Výpis počítačových kódů 2 Volání funkce wavread.

Další funkce pro práci s wav soubory je wavwrite(), která naopak ukládá zvukové signály do wav souborů. Všechny proměnné mají stejný význam jako v předchozím případě, jenom *'test'* je jméno souboru, do kterého se zapíše zvukový signál. Číslo 16 zde reprezentuje počet bitů na vzorek, předchozím případě nBits.

```
wavwrite(x, fs, 16, 'test');
```

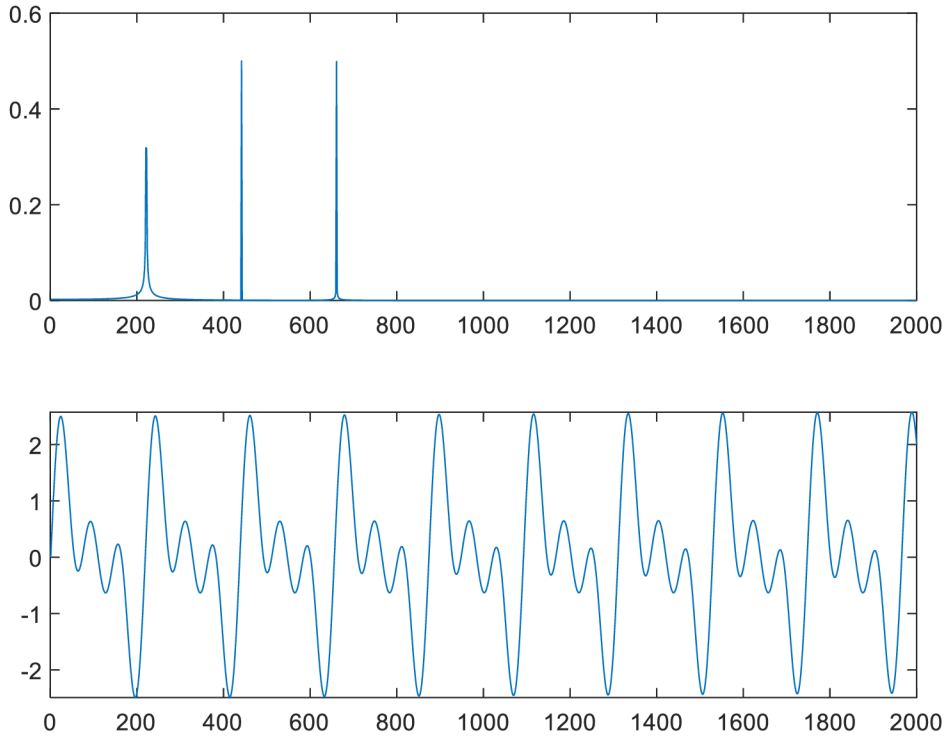
Výpis počítačových kódů 3 Volání funkce wavwrite.

### 1.3.1 Spektrální analýza zvukových signálů

Důležitou součástí práce se zvukem v Matlabu je především grafické zobrazení zvukového signálu a jeho kmitočtového spektra. Pro zobrazení takového spektra je zapotřebí výpočtu Diskrétní Fourierovy Transformace (DFT). Takový výpočet se v Matlabu realizuje funkcí fft(), která původnímu signálu vrací výpočet jeho Furierovy transformace. Jednotlivých úrovní DFT lze docílit výpočtem její absolutní hodnoty. Pro příklad bude zadán signál jako součet tří sinusovek. Nejjednodušší výpočet jednostranného spektra takového signálu lze realizovat například takto, kdy při dostatečném počtu vzorků dostaneme odpovídající hodnoty frekvencí zadaných signálů.

```
fs = 48000; Ts = 1/fs; N = 30000;

t = 0:Ts:1;

x1 = sin(2*pi*440*t);
x2 = sin(2*pi*220*t);
x3 = sin(2*pi*440*(2^(7/12))*t);

x = x1 + x2 + x3;

X = fft(x);
X = abs(X);
X = X(1:ceil(N/2));

F = (fs/2) * (1:ceil(N/2)) / ceil(N/2);

figure;

subplot(2,1,1);
plot(F,X);
xlim([0 2000]);

subplot(2,1,2);
plot(x);
xlim([0 2000]);
```

Výpis počítačových kódů 4 Zobrazení frekvenčního spektra a průběhu signálu.




Obrázek 1.1 Zobrazení frekvenčního spektra a průběhu zadaného signálu.

Na prvním grafu představuje vodorovná osa frekvenci v Hz a svislá osa úroveň jednotlivých frekvencí. V druhém grafu je vodorovná osa pořadí vzorků při vzorkovacím kmitočtu 48 000 Hz a svislá potom hodnota jednotlivých vzorků.

### 1.3.2 Příklady digitální syntézy

Jedním z příkladů digitální syntézy spadající do metody tvarování vlny je tvrdé ořezání signálu. V prostředí Matlab jej lze realizovat jednoduchou funkcí, nazvanou hardlimit(), tato funkce je převzatá funkce hel_hardlimit() vytvořená v předmětu Hudební elektronika, kde vstupní parametry jsou vstupní signál *in*, nejmenší hodnota, při které dochází k ořezání *minval* a nejvyšší hodnota, kdy dochází k ořezání signálu *maxval*. Výstup funkce je potom proměnná *out*.

```
function [out] = hardlimit(in, maxval, minval)

    out = in;
    out(in > maxval)= maxval;
    out(in < minval)= minval;

end
```

Výpis počítačových kódů 5 Funkce na ořezání signálu.

Pro příklad byla tato funkce použita na ořezání harmonického signálu vytvořeného další funkcí prubeh1():

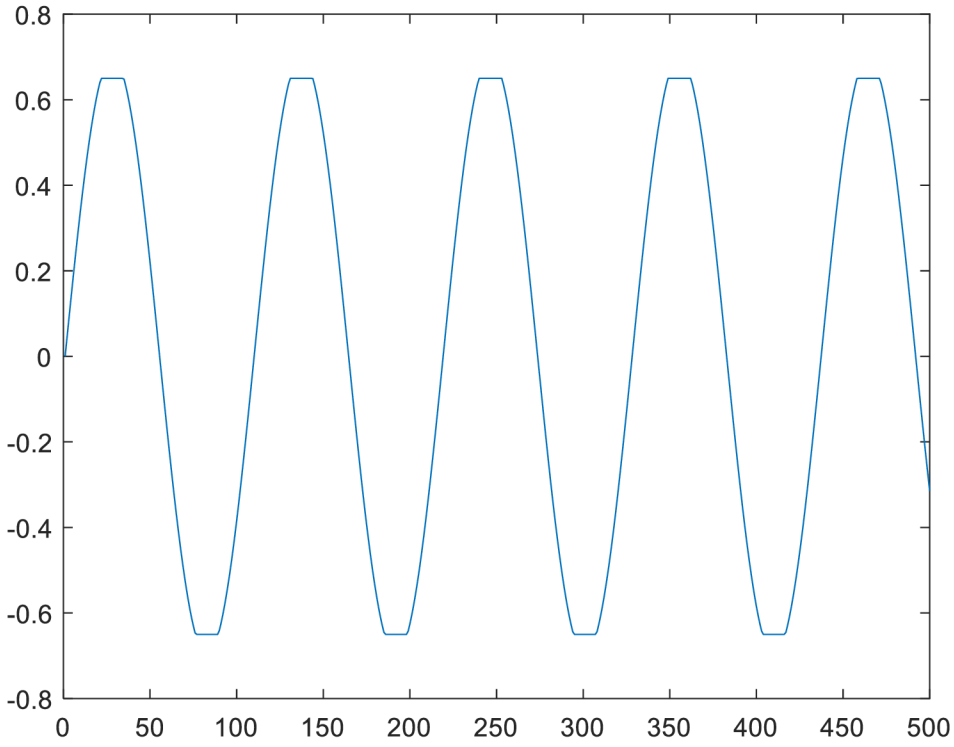
```
function [out, samples_out] = prubeh1(f, fs, sampleframe, samples_in)

    samples_out = samples_in + sampleframe;
    t = (samples_in :(samples_out-1))/fs;

    out = 0.7*sin(2*pi*f*t);
    out = hardlimit(output, 0.65, -0.65);

end
```

Výpis počítačových kódů 6 Generování harmonickéh průběhu a následného ořezání.

Tato funkce generuje signál o kmitočtu *f* při vzorkovací frekvenci *fs*, generuje určitý počet vzorků daný hodnotou *sampleframe* a hodnota *samples_in* určuje, kolikátý vzorek od počátku bude první. Výstup funkce jsou proměnné *output*, nesoucí výstupní signál a proměnná *samples_out*, která nabývá hodnoty pořadí prvního vzorku, který se již negeneruje. Pomocí parametru *t* se vytvoří časová osa rozdělená do počtu vzorků, který byl určen vstupními parametry. Výstupu *output* se potom přiřadí harmonický průběh, na který je následně aplikována funkce *hardlimit*. Zavoláním funkce s určitými parametry vznikne následující průběh. Osa X znázorňuje pořadí vzorků a osa Y potom jejich hodnotu.

```
out = prubeh1(440, 48000, 500, 0);
figure
plot(out);
```

Výpis počítačových kódů 7 Zobrazení průběhu po ořezání.




Obrázek 1.2 Průběh výsledného signálu vytvořený ořezáním harmonického průběhu.

Dalším příkladem digitální syntézy může být třeba složková syntéza realizována součtem dílčích sinusovek funkcí prubeh2():

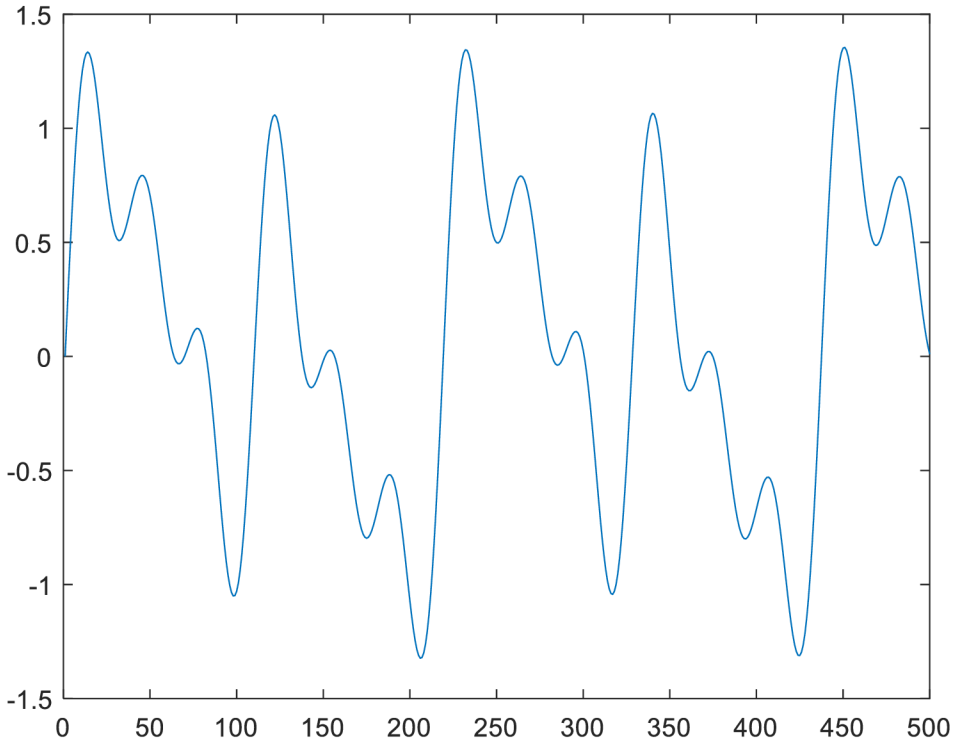
```
function [out, samples_out] = prubeh2(f, fs, sampleframe, samples_in)

    samples_out = samples_in+sampleframe;
    t = (samples_in:(samples_out-1))/fs;

    out     =     0.7*sin(2*pi*f*t)     +     0.4*sin(2*pi*1/2*f*t)     +
0.4*sin(2*pi*2*f*t) + 0.4*sin(2*pi*2*f*(2^(7/12))*t);

end
```

Výpis počítačových kódů 8 Součet generovaných sinusovek.

Parametry funkce zůstávají stejné jako v předchozím příkladu. Proměnné *output* je zde přiřazen součet 4 sinusovek s kmitočtem základního tónu, o oktávu níž, o oktávu výš a poslední o oktávu a kvintu výš. Zavoláním funkce vznikne následující průběh.

```
out = prubeh2(440, 48000, 500, 0);
figure
plot(out);
```

Výpis počítačových kódů 9 Zobrazení průběhu součtu harmonických signálů.




Obrázek 1.3 Průběh výsledného signálu vytvořený složkovou syntézou.

Poslední příklad digitální syntézy je filtrace. Pomocí funkce sign se v Matlabu vytvoří posloupnost obdélníkového signálu se střídou 50%. Funkce sign vrací hodnotu plus nebo mínus jedna podle, toho jestli je zkoumaná neznámá kladná nebo záporná. Dále se pomocí funkce modulo se vytvoří perioda tohoto střídání kladných a záporných hodnot. V kódu to vypadá takto:

```
function [out] = prubeh3(f, fs, sampleframe, Q, Fc)
t = (0:sampleframe)/fs;

    out = sign(2*f*mod(t, 1/f)-1);

    K = tan(pi*Fc/fs);
    den = (K^2*Q+K+Q);

                b0 = Q / den;
                b1 = -2*Q / den;
                b2 = Q / den;
                a1 = (2*Q*(K^2 - 1)) / den;
                a2 = (K^2*Q-K+Q) / den;

     b = [b0 b1 b2];
     a = [1 a1 a2];

      out = filter(b, a, out);

end
```

Výpis počítačových kódů 10 Generování obdélníku a následná filtrace.

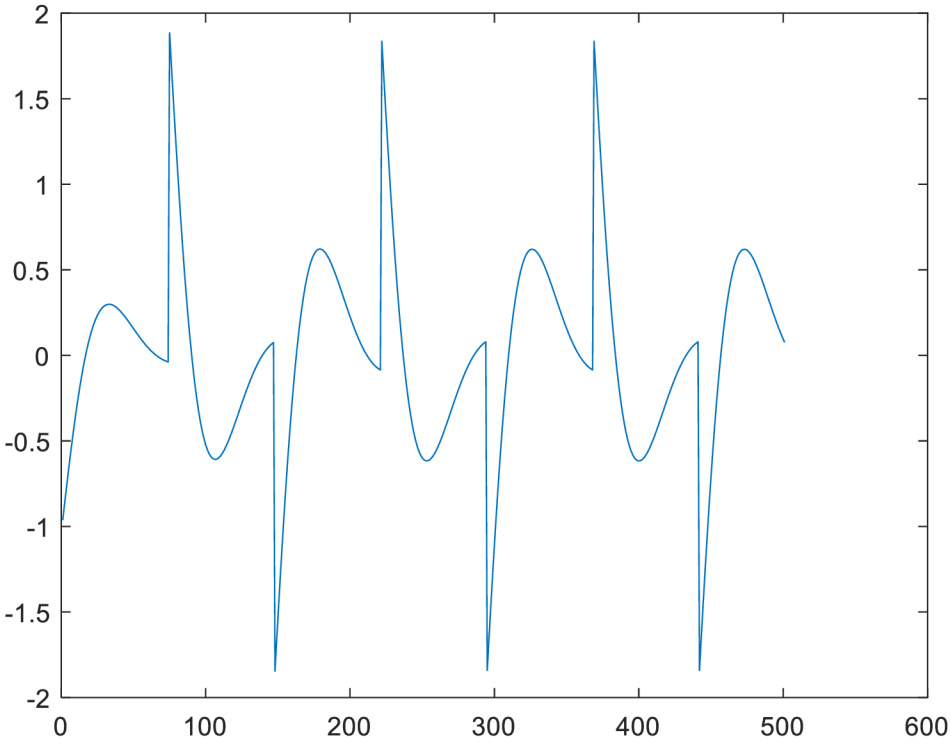
Vstupní parametry Q a Fc jsou činitel jakosti a mezní kmitočet filtrace, před zavoláním funkce filter se vypočítají koeficienty pro filtraci. Funkce filter vezme posloupnost hodnot, převede ji transformací Z a přepočítá hodnoty posloupnosti podle přenosové rovnice, kde figurují koeficienty *a* a *b*, v tomto případě tak, aby filtrace odpovídala typu pásmová propust. Obecná přenosová rovnice v oblasti Z vypadá takto:

$$Y(z) = \frac{b(1) + b(2)z^{-1} + \cdots + b(n_b + 1)z^{-n_b}}{1 + a(2)z^{-1} + \cdots + a(n_a + 1)z^{-n_a}} X(z)$$

Zavoláním takové funkce o kmitočtu 300 Hz, mezním kmitočtu filtrace 520 Hz, činiteli jakosti 1 vzorkovací frekvencí 44 100 Hz a 500 vzorky se vytvoří následující průběh:

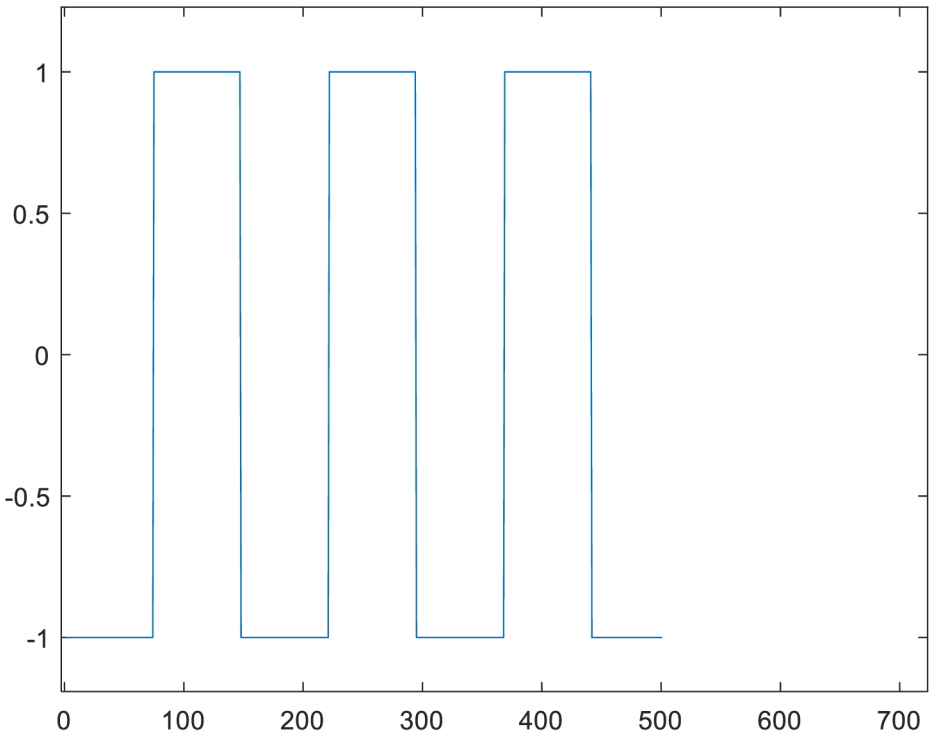
```
out = prubeh3(300, 44100, 500, 1, 520);

figure;
plot(out);
```

Výpis počítačových kódů 11 Zobrazení průběhu filtrovaného obdélníku.



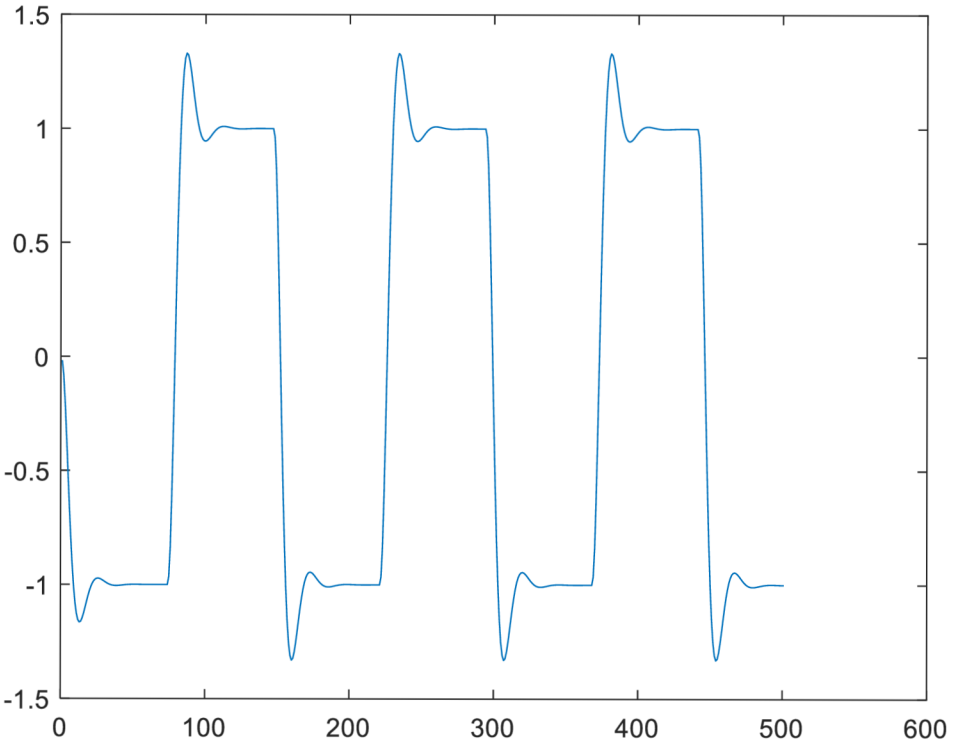

Obrázek 1.4 Obdélníkový signál po filtraci typu pásmová propust.

Stejný signál před filtrací přitom vypadá takto:




Obrázek 1.5 Obdélníkový signál před filtrací.

Další obrázek ukazuje filtraci obdélníkového signálu s pozměněnými parametry filtrace, mezní kmitočet je tentokrát 2000 Hz, a počítání koeficientů *a* a *b* bylo pozměněno tak, aby filtrace odpovídala typu dolní propusti. Zde jde vidět nepropuštění vyšších frekvencí u obdélníku tím, že jeho hrany nejsou rovné.




Obrázek 1.6 Obdélníkový signál po filtraci typu dolní propust.

# 2. ZVUKOVÁ SYNTÉZA V REÁLNÉM ČASE

## 2.1 Princip čtení MIDI zpráv

K syntéze v reálném čase je zapotřebí signály vytvářet a přehrávat po bufferech, kvůli zachování nízké latence. Výsledné zvukové signály se tedy dostávají na zvukovou kartu po částech v prvním případě vytvořeného programu pomocí systémového objektu audioPlayerRecorder() [4]. Syntezátor je řízený protokolem MIDI, a proto je nutné připojit MIDI zařízení k počítači a následně zařízení připojit k Matlabu. Pro čtení MIDI zpráv se vychází z funkce simplesynth() popsané ve zdroji [5]. Ve funkci jsou dále použité funkce pro zjištění MIDI zprávy nota zapnuta nebo nota vypnuta a funkce pro převedení čísla noty na hodnotu frekvence.

### 2.1.1 Funkce realtimesynth()

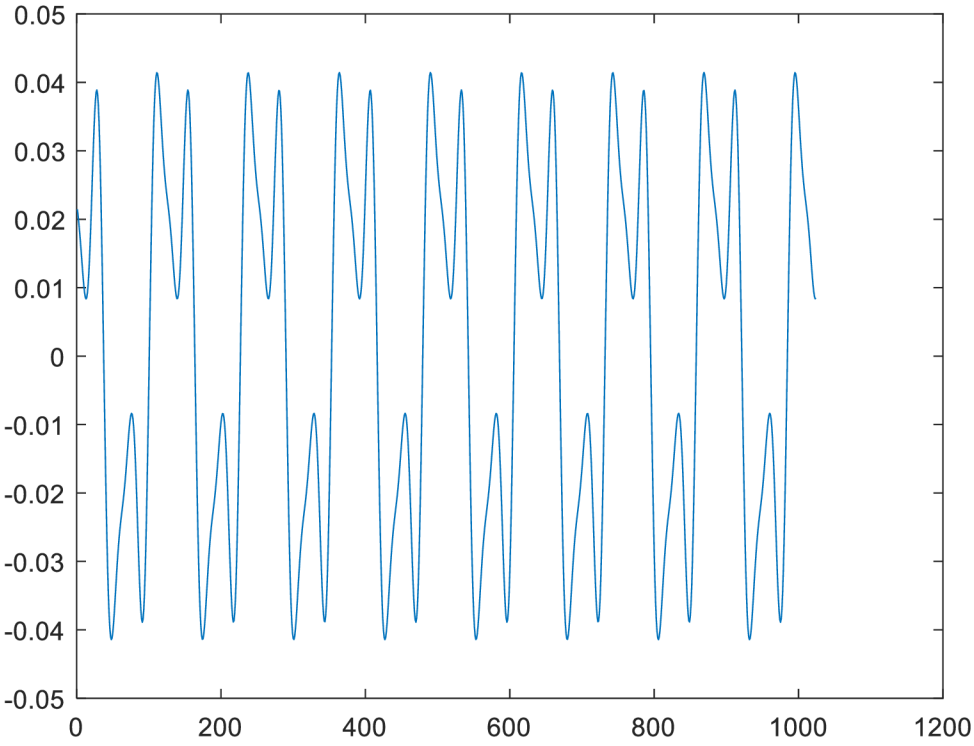
Podle funkce simplesynth() byla vytvořena nová funkce realtimesynth(), jako nástřel použitého principu, pro čtení zpráv a následnou syntézu, obě funkce jsou v přílohách. Vstupní parametr funkce je pouze jméno MIDI zařízení, řídící syntézu. Zařízení je následně připojeno k Matlabu, potom je vytvořený systémový objekt audioPlayerRecorder a nastaveny jeho properties, jako je použité zvukové zařízení a velikost bufferu. Dále jsou zde určeny proměnné určující počáteční podmínky pro generování průběhů nebo konkrétní parametry pro jednotlivé druhy syntézy, v tomto případě jsou to amplitudy jednotlivých tónů, které mohou znít současně, jejich frekvence, počáteční pořadí vzorků a parametry pro určení následní filtrace. Proměnná *p* určuje, kolik tónů dohromady hraje. Následuje hlavní smyčka while, ve které se čtou MIDI zprávy. Všechny přijaté MIDI zprávy se přiřadí do proměnné *msgs* a v následujícím cyklu for se vyšetřuje každá přijatá MIDI zpráva zvlášť. V tomto případě je polyfonie omezená na čtyři tóny. Je-li MIDI zpráva nota zapnuta, parametr *p* nabývá hodnoty o jednu vyšší než doposud jako další nota, která se má přehrávat až do hodnoty, kdy je *p* 4. Parametr *f* nabývá hodnoty frekvence zapnuté noty a parametr *A* nabývá amplitudy, nyní hodnoty 1. Pokud je MIDI zpráva nota vypnuta, rozpoznává se, která nota byla vypnuta, a její parametry jsou vynulovány. Hodnota *p* se o 1 sníží. Po té se přechází k druhému cyklu for, kde se po částech vytváří na sebe navazující průběhy vytvořené syntézy. Průběhy se násobí hodnotou amplitudy podle toho, jestli je nota zapnutá nebo vypnutá. Další blok je funkce provádějící možnou filtraci. Cyklus se zopakuje čtyřikrát pro čtyři různé tóny, které můžou znít současně. Průběhy se následně sečtou a jsou posílány na zvukovou kartu systémovým objektem audioPlayerRecorder. V programu se navíc počítají parametry pro následnou filtraci.

Funkce má dvě hlavní části. První část je neměnná a zajišťuje pouze čtení MIDI zpráv a polyfonii přehrávaných tónů. Druhá část je vlastní syntéza přehrávaných zvuků, kterou

může přetvářet uživatel podle požadavků na to, jak má výsledný tón znít. Nedostatek tohoto programu tkví v tom, že naprogramovanou syntézu nelze měnit v průběhu hraní.

### 2.1.2 Použitá syntéza

Ve funkci realtimesynth() je použitá aditivní syntéza voláním vytvořené funkce sin_gen() pro generování a součet různých sinusovek. Vstupy této funkce jsou frekvence základního tónu, vzorkovací frekvence, pořadí prvního vzorku, který se má generovat, délka generovaného signálu a poměry 9 sinusovek s celočíselnými násobky frekvencí. Vygenerovaný signál se dále upravuje filtrací typu dolní propust pomocí funkce filter1(). Před zavoláním této funkce se musí vypočítat koeficienty *a* a *b*, na základě zadané mezní frekvence a činitele jakosti filtrace. Vstupní parametry funkce jsou vstupní signál, koeficienty *a* a *b* a proměnné *z*, které zajišťují návaznost výstupního signálu, i když se filtrace provádí po částech. Výstup takové syntézy, při mezním kmitočtu filtrace 880 Hz a činiteli jakosti 1 s generováním sinusovek pouze prvního, třetího a pátého násobku základní frekvence, vypadá například takto:




Obrázek 2.1 Průběh výsledného signálu vytvořený funkcí realtimesynth.

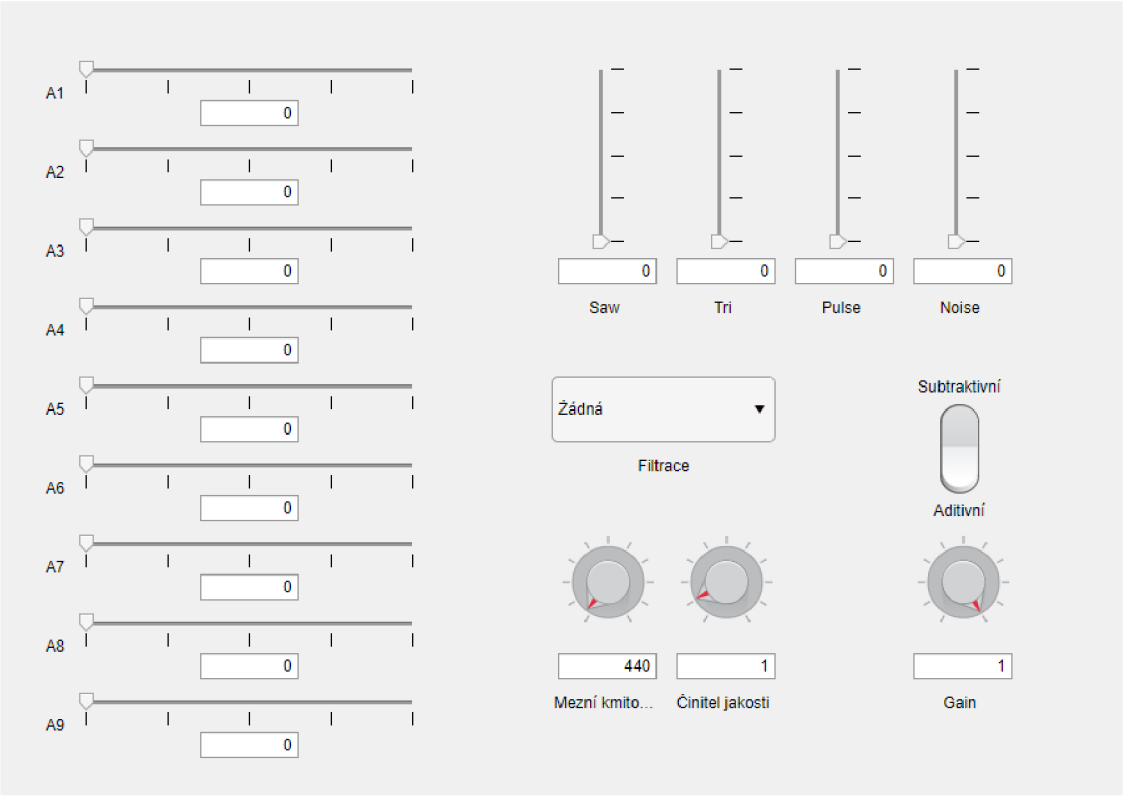
## 2.2 Použití audiopluginu

Pro lepší ovladatelnost a výběr syntézy je vhodné použít předchozí algoritmus v rámci audiopluginu. Audioplugin je v audiotoolboxu matlabu navrhovaný model k vytváření algoritmů pro zpracování zvukových signálů. Jeho výhoda je, že může být používán v rámci DAW (digital audio Workstation). Nabízí nastavitelné prvky v rámci zpracování zvuku a jejich ovládání v reálném čase v uživatelském rozhraní Audio Test Bench. Může být ovládaný připojeným MIDI zařízením a umožňuje grafické zobrazení výstupního zvukového signálu nebo jeho frekvenčního spektra. Dále je schopen zvuk poslat jak na zvukovou kartu tak uložit do souboru. Vlastní audioplugin byl vytvořený podle zdroje [6].

### 2.2.1 Vytvořený syntezátor

V audiopluginu byl vytvořený ovladatelný syntezátor využívající různé druhy syntézy. Syntezátor má dvě možnosti pro generování základního signálu. Buď sčítá maximálně 9 sinusovek s celočíselným násobkem frekvence, nebo sčítá maximálně 4 různé základní průběhy jako je pila, obdélník, trojúhelník nebo šum. Poměry sečtených signálů jsou nastavitelné. Součty těchto různých průběhu je poté možno filtrovat a nastavovat parametry filtrace, které jsou typy filtrace na dolní propust, pásmová propust, horní propust a žádná (filtrace), mezní kmitočet – Fc a činitel jakosti Q.

Ve vlastním kódu syntezátoru, pojmenovaný syntezator(), se nejprve definují různá properties. Nejprve ty ovladatelné, což jsou podíly výsledného signálu, podíly jednotlivých dílčích signálů, typ použité syntézy, typ filtrace a parametry filtrace. V další části jsou definované neovladatelné properties, které se používají ve funkci pro syntézu nebo čtení MIDI zpráv. Jsou to jméno MIDI zařízení, vzorkovací frekvence, parametr pro počet hrajících tónů současně, pořadí prvního vzorku, který se má u jednotlivých tónů generovat kvůli návaznosti, počet vzorků které se generují v jednom cyklu, parametry frekvencí a amplitud hrajících tónů, parametr nabývající informaci o vstupním MIDI zařízení a parametry pro návaznost filtrace. V poslední části před definováním metod audiopluginu se definují neměnné properties. V tomto případě definování Audioplugin Interface, které vypadá takto:




Obrázek 2.2 Uživatelské rozhraní audiopluginu syntezator.

Metody audiopluginu jsou nejprve funkce syntezator(), která nastavuje parametr pro vstupní MIDI zařízení jako nastavené MIDI zařízení v properties a parametr používaný pro vzorkovací frekvenci nastavuje jako vzorkovací frekvenci audiopluginu.

```
function plugin = syntezator
            plugin.midiInput = mididevice(plugin.device);
            plugin.sr = getSampleRate(plugin);
        end
```

Výpis počítačových kódů 12 Funkce audiopluginu syntezator.

Následuje hlavní funkce process s výstupním parametrem *out* a vstupními parametry *in* a *plugin*, ihned se definuje délka výstupního signálu.

```
function out = process(plugin, in)

            out = zeros(size(in));
```

Výpis počítačových kódů 133 První část funkce process audiopluginu syntezator.

Dále se definují pomocné parametry volané v jednotlivých funkcích kódu pro syntézu a čtení MIDI zpráv podle nastavených properties.

```
            % nastavení parametrů podle parametrů pluginu
             start = plugin.Pstart;
             A = plugin.amp;
             f = plugin.Pf;
             p = plugin.Pp;
             fs = plugin.sr;
             frame = plugin.framesize;
             z = plugin.zet;
```

Výpis počítačových kódů 14 Druhá část funkce process audiopluginu syntezator.

*A* a *f* jsou vektory amplitud a frekvencí hrajících tónů. Parametr *p* je počet tónů znějících současně. Parametr *fs* je vzorkovací frekvence, *frame* je počet vzorků, které se generují v jednom cyklu a *z* jsou proměnné pro návaznost filtrace.

Dále se počítají koeficienty *a* a *b* kvůli filtraci pro tři různé případy podle zvolené filtrace: dolní propust, pásmová propust nebo horní propust.

Následuje čtení MIDI zpráv stejným principem jako v případě funkce realtimesynth(), tentokrát může parametr *p* nabývat hodnoty až 8. V této části kódu se notám přiřadí jejich frekvence a, amplituda se nastaví na hodnotu 1 v případě MIDI zprávy nota zapnuta, v případě nota vypnuta se amplituda nastaví zpátky na hodnotu 0.

Po přečtení MIDI zpráv následuje cyklus for, ve kterém probíhá syntéza a všechny noty hrající současně se sečtou. Funkce, které se volají v rámci syntézy, jsou sin_gen(), kde se generují a sčítají již zmíněné sinusovky a nebo wavesum(), kde se generují a sčítají ostatní průběhy.

```
            for m = 1:8
                if plugin.Type == 0
                [outpart, start(m)] = sin_gen(f(m), fs, frame, start(m),
plugin.A1,  plugin.A2,  plugin.A3,  plugin.A4,  plugin.A5,  plugin.A6,
plugin.A7, plugin.A8, plugin.A9);
                elseif plugin.Type == 1
                [outpart, start(m)] = wavesum(f(m), fs, frame, start(m),
plugin.Saw, plugin.Tri, plugin.Pulse, plugin.Noise);
                end
                if plugin.Filtrace < 3
                    [outpart, z(:,m)] = filter1(outpart, z(:,m), b, a);
                end
                outpart = A(m) * 1/8 * outpart;
                outpart = outpart';
                out = out + outpart;
            end
```

Výpis počítačových kódů 15 Třetí část funkce process audiopluginu syntezator.

Pro maximálně 8 současně znějících not se podle zvolené syntézy volají funkce pro generování součtu různých průběhů a tyto průběhy se následně pro každou notu zvlášť podrobí filtraci voláním funkce filter1() s už vypočítanými koeficienty pro zvolenou

filtraci. Následně se vytvořené průběhy not sečtou. Funkce pro generování signálů a filtraci vypadají takto:

```
function [output, samples_out] = sin_gen(f, fs, sampleframe, samples_in,
g1, g2, g3, g4, g5, g6, g7, g8, g9)
    samples_out = samples_in+sampleframe;
    t = (samples_in:(samples_out-1))/fs;
   output    =    1/9    *(g1*sin(2*pi*f*t)    +    g2*sin(2*pi*2*f*t)    +
g3*sin(2*pi*3*f*t) + g4*sin(2*pi*4*f*t)
   +  g5*sin(2*pi*5*f*t)  +  g6*sin(2*pi*6*f*t)  +  g7*sin(2*pi*7*f*t)  +
g8*sin(2*pi*8*f*t) + g9*sin(2*pi*9*f*t));
end
```

Výpis počítačových kódů 16 Funkce sin_gen použitá v audiopluginu.

Generování těchto průběhů funguje stejně jako průběhy v kapitole 1.3.21.3.2, g1- g9 jsou podíly pro sčítané sinusovky. V následující funkci jsou potom podíly jednotlivých průběhů parametry Saw, Tri, Rect a Noise.

```
function [out, samples_out] = wavesum(f, fs, sampleframe, samples_in,
Saw, Tri, Rect, Noise)
    samples_out = samples_in+sampleframe;
    t = (samples_in:(samples_out-1))/fs;

    out  =  0.25*Tri*(2*abs(2*f*mod(t,  1/f))  -1)  +  0.25*Saw*2*f*mod(t,
1/f)   +   0.25*Rect*sign(2*f*mod(t,   1/f)-1)   +   0.25*Noise*randn(1,
sampleframe);

end
```

Výpis počítačových kódů 17 Funkce wavesum použitá v audiopluginu.

Funkce filter1() používá funkci filter(), která filtruje daný signál podle zvolených koeficientů a přepisuje parametry *z* pro zajištění návaznosti. Je-li vstupní parametr pro filtraci *zi* nulový, filtruje se počáteční část signálu a není třeba zadávat do funkce filter() žádné parametry *z*.

```
function [out, z] = filter1(in, zi, b, a)

% in - vstupni signal
% fs - vzorkovaci kmitocet vstupniho signalu v Hz


%% filtrace signalu
if zi(1,:) == 0

[out, z] = filter(b, a,in);


else

[out, z] = filter(b, a, in, zi);


end

end
```

Výpis počítačových kódů 18 Funkce filter1 použitá v audiopluginu.

Po vygenerování signálu, který se zapíše do výstupní proměnné *out*, je třeba přenastavit parametry pluginu podle přepočítaných parametrů použitých v rámci syntézy a čtení MIDI zpráv kvůli zachování návaznosti dalšího cyklu.

```
            % Nastavení parametrů pluginu podle přepočítaných parametrů
            plugin.Pstart = start;
            plugin.Pf = f;
            plugin.Pp = p;
            plugin.amp = A;
            plugin.zet = z;
```

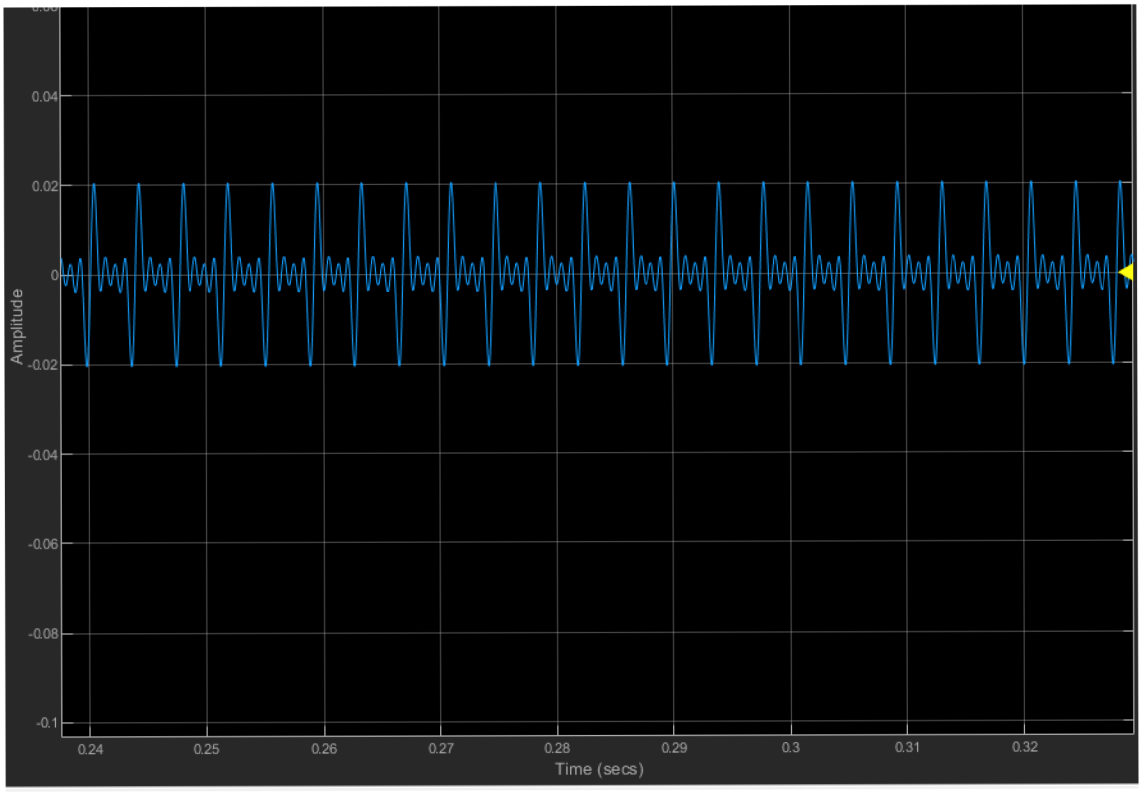
Výpis počítačových kódů 19 Čtvrtá část funkce process audiopluginu syntezator.

Poslední funkce v metodách audiopluginu je funkce reset(), která parametry pluginu resetuje, jak byly definované na začátku.

```
        function reset(plugin)
            plugin.Pp = 0;
            plugin.Pstart = [0 0 0 0 0 0 0 0];
            plugin.framesize = 256;
            plugin.Pf = [0 0 0 0 0 0 0 0];
            plugin.amp = [0 0 0 0 0 0 0 0];
            plugin.zet = zeros(2, 9);

        end
```

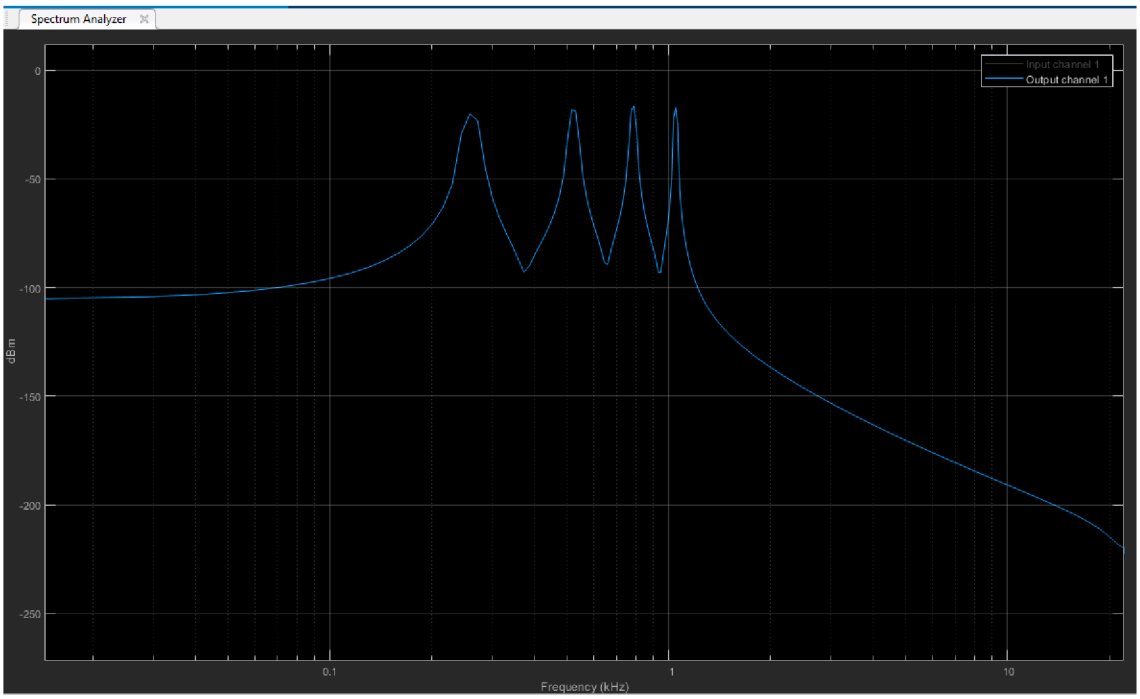
Výpis počítačových kódů 20 Funkce reset audiopluginu syntezator.

Pro příklad byla zaznamenána na osciloscopu a spektrálním analyzéru zahraná nota C1 vytvořená součtem čtyř sinusovek.




Obrázek 2.3 Průběh výsledného signálu zobrazený na osciloscopu pluginu.

Spektrum takového signálu potom bylo zobrazeno takto:

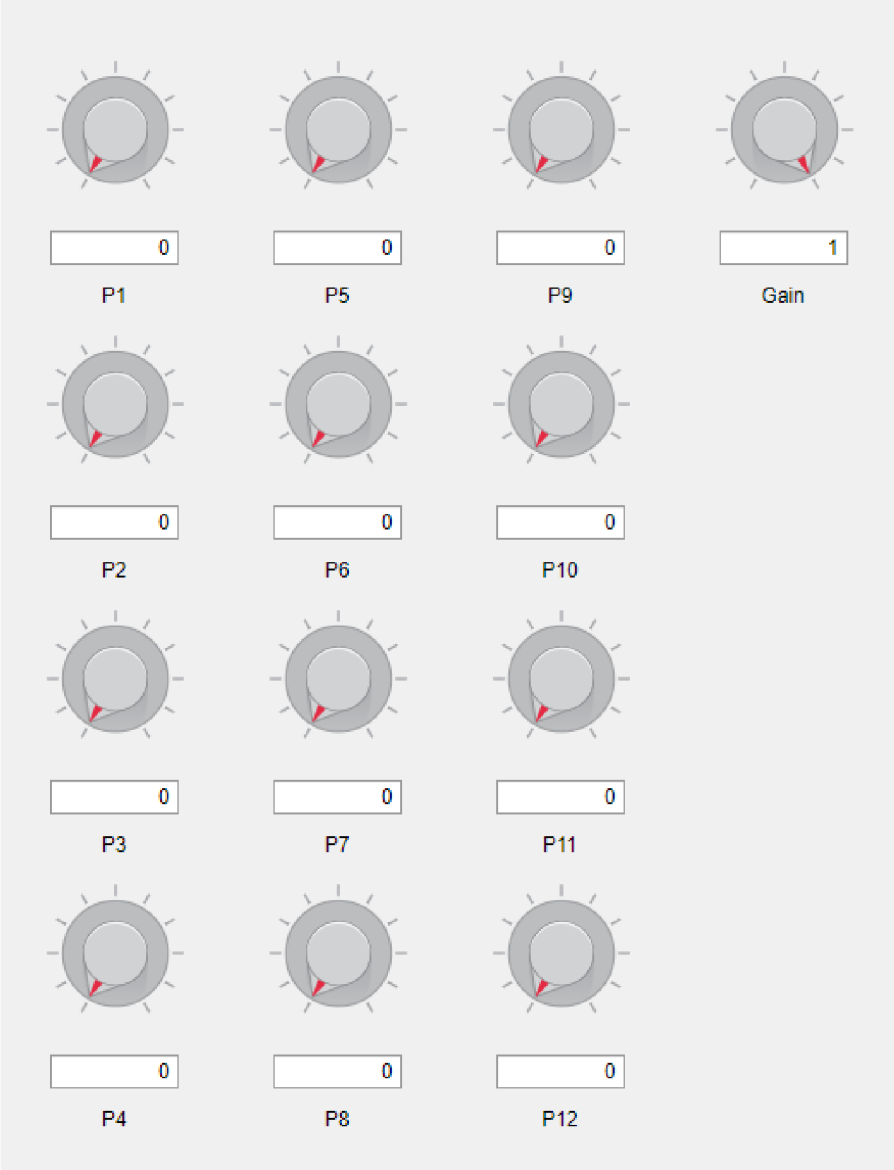



Obrázek 2.4 Spektrum signálu zobrazené na spektrálním analyzéru pluginu.

Zvuk vytvořený v tomto syntezátoru byl zaznamenaný do souboru z nabízeného výstupu Audio File Writer s vzorkovací frekvencí 44 100 Hz a bitovou hloubkou 16 bitů. Ve zvukovém souboru se při hraní střídá nabízená syntéza a používá se filtrace typu dolní propust. Soubor má název output.wav.

## 2.3 Framework

Jeden z hlavních úkolů této práce je vytvořit v Matlabu prostředí pro uživatele jako šablonu, do které by se doplnila určitá zvuková syntéza, kterou by bylo možné hned přehrát v reálném čase. V audiopluginu byl tedy vytvořen framework na stejném principu jako syntezátor v předchozí kapitole. Prostředí nabízí uživateli 12 různých ovladatelných potenciometrů, u kterých si může definovat jejich význam. Celou syntézu lze vytvářet v jedné funkci, která se volá z hlavní funkce audiopluginu, ve které dochází ke čtení MIDI zpráv a umožnění polyfonie not jako v minulém případě syntezátoru. Uživatelské rozhraní frameworku vypadá takto:




Obrázek 2.5 Uživatelské rozhraní audiopluginu framework.

V hlavní funkci audiopluginu je vyhrazené místo odkud uživatel volá funkci, ve které probíhá zvuková syntéza, uživatel tak programuje pouze tuto funkci. Vstupy této funkce jsou vzorkovací frekvence, základní frekvence tónu, počáteční pořadí generovaných vzorků, počet generovaných vzorků, případně koeficienty pro filtraci a 12 parametrů představující ovladatelné potenciometry.

### 2.3.1 Tutoriál

Tato kapitola slouží jako tutoriál, pro použití zvukové syntézy ve vytvořené šabloně Framework v Matlabu využívající audiotoolbox.

**Úvod**

Framework je v Matlabu naprogramovaný audioplugin. Slouží jako prostředí pro vytváření zvukové syntézy a následné přehrání v reálném čase hudebním nástrojem přes protokol MIDI. V následujících krocích je ukázáno jakým způsobem používat toto prostředí.

**Krok 1**

Prvním krokem je připojit k počítači MIDI zařízení a následně otevřít Matlab vyžívající Audiotoolbox a v něm otevřít kódy „framework.m“ a „funkce.m“.

**Krok 2**

Nejprve je potřeba přepsat kód audiopluginu Framework tak, aby používal připojené MIDI zařízení. MIDI zařízení se definuje na 20. řádku v sekci properties, do uvozovek k parametru device je potřeba napsat jméno používaného MIDI zařízení.

```
        properties (SetAccess=private)

device = ('Steinberg UR242-1')      % MIDI zařízení
sr = 44100                          % samplerate pro vytvářené průběhy
Pp = 0                              % parametr pro počet hrajících not
Pstart = [0 0 0 0 0 0 0 0]          % počáteční pořadí vzorků pro
framesize = 256                     % počet vzorku, co se generuje
Pf = [0 0 0 0 0 0 0 0]              % frekvence dílčích tónů
amp = [0 0 0 0 0 0 0 0]             % amplitudy dílčích tónů
midiInput                           % vtup pro MIDI zprávy
zet = zeros(2, 9)                   % parametr z pro filtraci
```

Výpis počítačových kódů 21 Nastavení properties v audiopluginu framework.

V případě, že neznáme jméno zařízení, Matlab umí vypsat dostupné zařízení zavoláním příkazu mididevinfo do Command Window.

```
Command Window
>> mididevinfo
  MIDI devices available:
  ID  Direction  Interface   Name
   0   output    MMSystem   'Microsoft MIDI Mapper'
   1    input    MMSystem   'Steinberg UR242-1'
   2   output    MMSystem   'Microsoft GS Wavetable Synth'
   3   output    MMSystem   'Steinberg UR242-1'
fx >>
```

Obrázek 2.6 Příklad vypsání dostupných MIDI zařízení.

**Krok 3**

Dalším krokem je programování vlastní syntézy v kódu funkce. Tam už je přichystaný kód pro zachování návaznosti vzorků a generování časové osy vzorků oddělených časovými úseky podle vzorkovací periody. Přichystané vstupy pro funkci jsou:

```
funkce(f, fs, sampleframe, samples_in, P1, P2, P3, P4, P5, P6, P7, P8,
P9, P10, P11, P12);
```

Výpis počítačových kódů 22 Volání funkce, kde probíhá syntéza.

Vstupy představují f - frekvenci hrané noty, fs – vzorkovací frekvenci, sampleframe – počet generovaných vzorků, samples_in – pořadí prvního vzorku, který se má generovat a P1 – P12 jsou ovladatelné parametry, které může uživatel libovolně využívat, můžou nabývat hodnoty 0 - 127. Pro příklad se bude používat aditivní syntéza o součtu tří sinusovek. Vytvořte každou ze sinusovek o různých frekvencích a následně je sečtěte. Výsledek vypadá například takto:

```
out1 = sin(2*pi*f*t);
out2 = sin(2*pi*f*2^(5/12)*t);
out3 = sin(2*pi*f*2*t);

out = 1/3 * (out1 + out2 + out3);
```

Výpis počítačových kódů 23 Příklad generování harmonických průběhů.

**Krok 4**

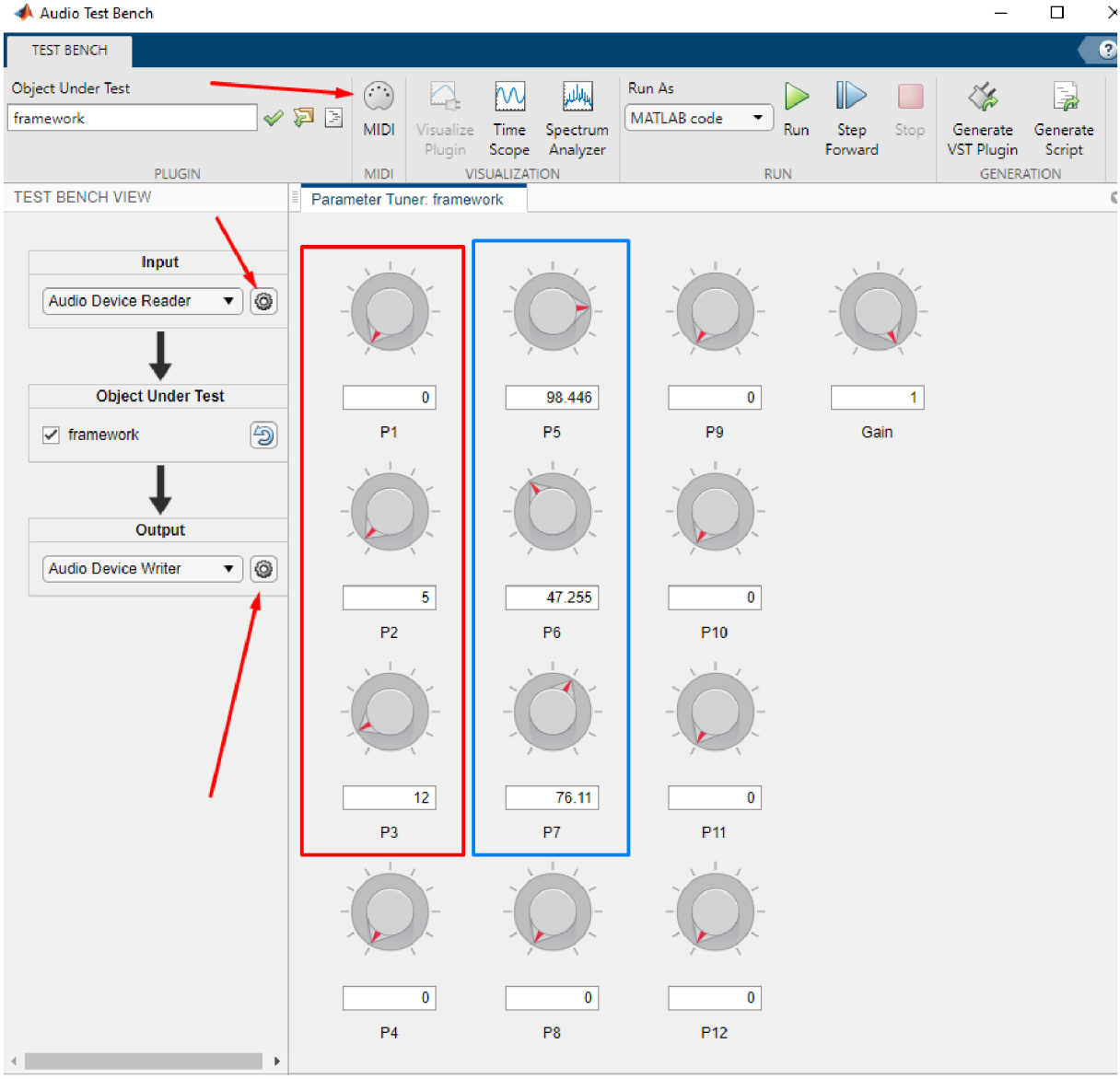
Nyní se do kódu doplní ovladatelné prvky, abychom mohli syntézu jednoduše ovládat. Doplňte do kódu 6 dalších parametrů, kterými se budou ovládat amplitudy a frekvence jednotlivých sinusovek.

```
out1= P5/127 * sin(2*pi*f*2^(P1/12)*t);
out2= P6/127 * sin(2*pi*f*2^(P2/12)*t);
out3= P7/127 * sin(2*pi*f*2^(P3/12)*t);

out = 1/3 * (out1 + out2 + out3);
```

Výpis počítačových kódů 24 Přidání ovladatelných parametrů.

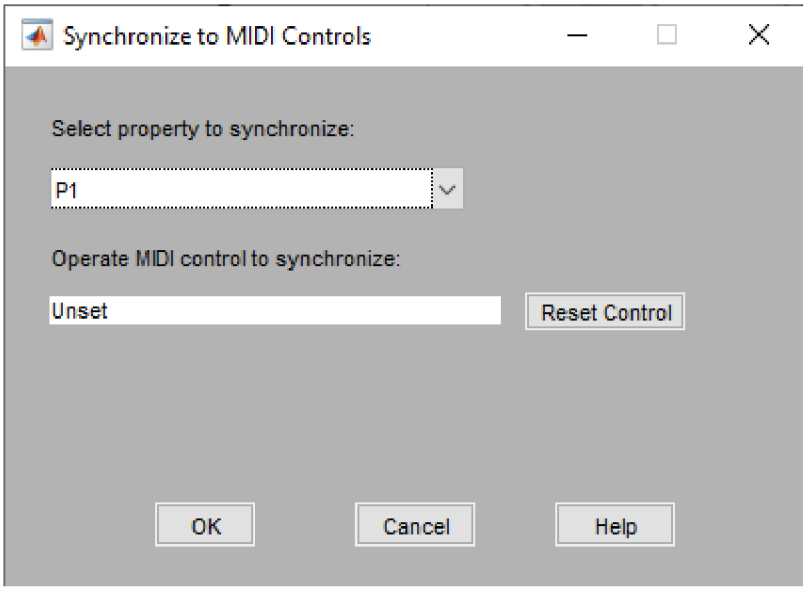
**Krok 5**

V dalším kroku spusťte Audio Test Bench příkazem do konzole „audioTestBench(framework)“, nastavte hodnoty používaným prvkům a ověřte správnost předpokládané syntézy. Případně nastavte ovladatelné prvky k MIDI kontrolérům na vašem MIDI zařízení. Dále zkontrolujte nastavení použitého driveru pro vstup a výstup.



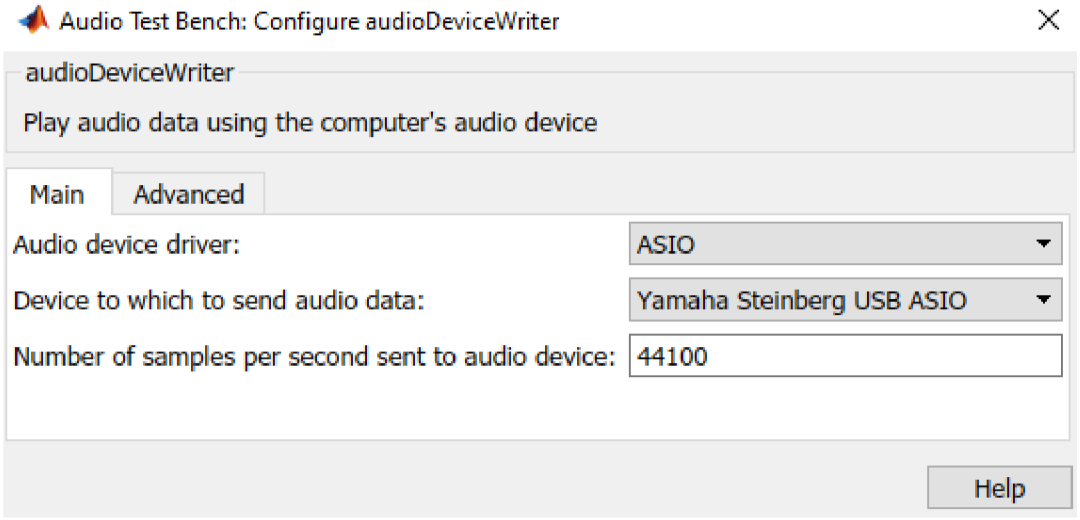

Obrázek 2.7 Okno Audio Test Bench.

V tomto obrázku jsou používané potenciometry vyznačené červeným a modrým obdélníkem. Šipky v obrázku ukazují, kde se nastavuje propojení k MIDI kontrolérům, nebo kde se nastavuje vstupní a výstupní zařízení.




Obrázek 2.8 Nastavení synchronizace s MIDI kontroléry.

Synchronizace k MIDI kontrolérům probíhá ručně. Nejprve se zvolí parametr, který se má s kontrolérem synchronizovat a následně se propojí manipulací s konkrétním kontrolérem.




Obrázek 2.9 Nastavení výstupu.

V tomto případě chceme nastavit hodnoty stejné, s jakými se počítá v kódu framework, což jsou vzorkovací frekvence 44 100 Hz a velikost bufferu 256, která se nastavuje v záložce advanced.

# 3. ZÁVĚR

Práce splňuje zadané očekávání. V rámci řešení bylo naprogramováno prostředí pro implementaci zvukové syntézy, jenž je možné řídit protokolem MIDI. Dále byl vytvořen tutoriál pro tvorbu jednoduchého syntezátoru v tomto prostředí. Prostředí je naprogramováno jako audioplugin, umožňuje zápis na zvukovou kartu i do souboru a zobrazuje frekvenční spektrum i průběh výsledného zvukového signálu. Součást řešení jsou všechny vlastní kódy zmíněné v práci a dokumentace s tutoriálem k prostředí Framework, dále také zvukový soubor zaznamenané syntézy audiopluginu syntezator. Zařízení a software, který byl použit v rámci této práce, je počítač Lenovo ThinkPad 13, zvuková karta Steinberg UR-242 a klávesy Yamaha PSR-275 a používaný software MATLAB R2020b.

# LITERATURA


[1] RUSS, M., Sound Synthesis and Sampling. Focal Press, 1996. ISBN 0-240-51429-7

[2] GIANNKOPOULOS, T., PIKRAKIS, A., Introduction to Audio Analysis: A MATLAB Approach, 1st ed. Academic Press, 2014. ISBN 978-0080993881

[3] 1-D digital filter - MATLAB filter [online].Natick [cit. 2021-12-11]. Dostupné z: https://www.mathworks.com/help/matlab/ref/filter.html

[4] Simultaniosly play and record using an audio device - MATLAB [online].Natick [cit. 2021-12-11]. Dostupné z: https://www.mathworks.com/help/audio/ref/audioplayerrecorder-system-object.html

[5] Design and Play a MIDI Synthesizer - MATLAB & Simulink [online].Natick [cit. 2021-12-11]. Dostupné z: https://www.mathworks.com/help/audio/ug/midi-synthesizer.html

[6] Debug, test and play audioplugin – MATLAB [online].Natick [cit. 2022-5-24]. Dostupné z: https://www.mathworks.com/help/audio/ref/audiotestbench-app.html

# SEZNAM SYMBOLŮ A ZKRATEK

Zkratky:

| | |
|---|---|
| MIDI | Musical instrument digital interface |
| PWM | Pulse width modulation, modulace šířky pulsu |
| AM | Amplitudová modulace |
| RM | Ring modulation, kruhová modulace |
| FM | kmitočtová modulace |
| DFT | Diskrétní Fourierova transformace |

Symboly:

| | | |
|---|---|---|
| $f$ | frekvence | [Hz] |
| $f_S$ | vzorkovací frekvence | [Hz] |
| $T$ | perioda | [s] |
| $T_S$ | vzorkovací perioda | [s] |
| $t$ | diskrétní čas, pořadí vzorků | |
| $x$ | diskrétní zvukový signál | |
| $N$ | počet generovaných vzorků | |
| $in$ | vstupní signál | |
| $out$ | výstupní signál | |
| $Q$ | činitel jakosti | |
| $Fc$ | mezní kmitočet | [Hz] |

# SEZNAM PŘÍLOH

# Příloha A - Funkce simplesynth

```
function simplesynth(midiDeviceName)

    midiInput = mididevice(midiDeviceName);
    osc = audioOscillator('square', 'Amplitude', 0);
    deviceWriter = audioDeviceWriter;
    deviceWriter.SupportVariableSizeInput = true;
    deviceWriter.BufferSize = 64; % small buffer keeps MIDI latency low

    while true
        msgs = midireceive(midiInput);
        for i = 1:numel(msgs)
            msg = msgs(i);
            if isNoteOn(msg)
                osc.Frequency = note2freq(msg.Note);
                osc.Amplitude = msg.Velocity/127;
            elseif isNoteOff(msg)
                if msg.Note == msg.Note
                    osc.Amplitude = 0;
                end
            end
        end
        deviceWriter(osc());
    end
end

function yes = isNoteOn(msg)
    yes = msg.Type == midimsgtype.NoteOn ...
        && msg.Velocity > 0;
end

function yes = isNoteOff(msg)
    yes = msg.Type == midimsgtype.NoteOff ...
        || (msg.Type == midimsgtype.NoteOn && msg.Velocity == 0);
end

function freq = note2freq(note)
    freqA = 440;
    noteA = 69;
    freq = freqA * 2.^((note-noteA)/12);
end
```

# Příloha B - Funkce realtimesynth

```
funkce simplesynth function realtimesynth(midiDeviceName)

    midiInput = mididevice(midiDeviceName);

    aPR = audioPlayerRecorder;
    aPR.Device = ("Yamaha Steinberg USB ASIO");
    aPR.SupportVariableSize = true;
    sampleframe = 1024;
    aPR.BufferSize = sampleframe;

    global quit_main_loop;
    quit_main_loop = false;
    figure('CloseRequestFcn',@my_closereq)

    framestart = [0, 0, 0, 0];
    A = [0, 0, 0, 0];
    f = [0, 0, 0, 0];
    p = 0;
    Q = 1;
    Fc = 880;
    z = zeros(2,5);
    K = tan(pi*Fc/44100);
            den = (K^2*Q+K+Q);

                b0 = (K^2*Q) / den;
                b1 = (2*K^2*Q) / den;
                b2 = (K^2*Q) / den;
                a1 = (2*Q*(K^2 - 1)) / den;
                a2 = (K^2*Q-K+Q) / den;
                b = [b0 b1 b2];
                a = [1 a1 a2];

    while ~quit_main_loop
          msgs = midireceive(midiInput, 10);

        for i = 1:numel(msgs)
                msg = msgs(i);
            if isNoteOn(msg)
                if p < 5
                  p = p + 1;
                  f(p) = note2freq(msg.Note);
                  A(p) = 1;
                end

            elseif isNoteOff(msg)
                    freq = note2freq(msg.Note);
                if freq == f(1)
                  A(1) = 0;
                  framestart(1) = 0;
                  z(:,1) = zeros;
                elseif freq == f(2)
                    A(2) = 0;
                    framestart(2) = 0;
                     z(:,2) = zeros;
```

```
                elseif freq == f(3)
                    A(3) = 0;
                    framestart(3) = 0;
                     z(:,3) = zeros;
                elseif freq == f(4)
                    A(4) = 0;
                    framestart(4) = 0;
                     z(:,4) = zeros;
                end

                p = p - 1;

            end
        end

        h = 0;

        for m=1:4
            [out, framestart(m)] = sin_gen(f(m), 44100, sampleframe,
framestart(m),1,0,1,0,1,0,0,0,0);
             out = A(m) * out;
            [out,z(:,m)] = filter1(out, z(:,m), b, a);
             out = 0.25*out;
             h = h + out;

        end
              plot(h);
              drawnow;
              [audiofromdevice, underruns] = aPR(h');

                    if underruns > 0
                    disp(strcat(num2str(underruns),' ','underruns'));
                    end

    end
end

function my_closereq(src,event)
    global quit_main_loop;
    quit_main_loop = true;
    delete(gcf);
end
function yes = isNoteOn(msg)
    yes = msg.Type == midimsgtype.NoteOn ...
        && msg.Velocity > 0;
end

function yes = isNoteOff(msg)
    yes = msg.Type == midimsgtype.NoteOff ...
        || (msg.Type == midimsgtype.NoteOn && msg.Velocity == 0);
end

function freq = note2freq(note)
    freqA = 440;
    noteA = 69;
    freq = freqA * 2.^((note-noteA)/12);
end
```