Sem vložte zadání Vaší práce.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Digitální realizace elektronických výhybek pro reproduktorové soustavy

Bc. Pavel Dohnal

Katedra číslicového návrhu
Vedoucí práce: doc. Dr. Ing. Jan Kyncl

4. května 2022

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. května 2022 .....

Odkaz na tuto práci

Dohnal, Pavel. *Digitální realizace elektronických výhybek pro reproduktorové soustavy*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato práce popisuje, jakým způsobem lze navrhnout digitální reproduktorovou výhybku, která implementuje zadané přenosové funkce. Dále porovnává implementaci výhybky Eulerovou metodou, metodou Runge Kutta a implementaci pomocí rekurzivního filtru.

Klíčová slova reproduktorová výhybka, Euler, Runge Kutta, rekurzivní filtr, filtr, audio, reproduktor

Abstract

This thesis describes a design of digital speaker crossover circuit, that implements given transfer function. It compares implementation using Euler method, Runge Kutta method and implementation by a recursive filter.

Keywords speaker crossover, Euler, Runge Kutta, recursive filter, filter, audio, speaker

Obsah

Seznam obrázků

Úvod

Jednou ze součástí reproduktorové soustavy jsou tzv. reproduktorové výhybky. Jedná se o elektrické obvody, které vstupní signál rozdělí na několik frekvenčních pásem. Každé pásmo je poté vedeno do jiného reproduktoru.

Běžně se jedná o analogový obvod. Analogové obvody jsou ovšem přesné jen tak, jak přesné vlastnosti mají komponenty, ze kterých je stvořen. Mým úkolem bylo navrhnout digitální protějšek tohoto zařízení. V této práci popisuji návrh takového zařízení a srovnávám různé metody výpočtů.

Kapitola 1

Cíl práce

Tato práce si klade za cíl navrhnout digitální reproduktorovou výhybku, která implementuje zadanou přenosovou funkci. Prozkoumat různé metody výpočtů a porovnat je navzájem. Dále popsat, jak tyto metody fungují a jakým způsobem se dosáhne toho, aby implementovali zadanou přenosovou funkci.

Kapitola 2

Analýza

2.1 Zvuk a zvukové signály

Zvuk je mechanické vlnění, které jsme schopni vnímat našimi sluchovými orgány. Šířit se může nejenom vzduchem, ale i dalším libovolným materiálem (vodou, konstrukcí budovy atd.). Zdravý člověk je schopen slyšet zvuk o frekvencích 20Hz až 20kHz. Se zvyšujícím se věkem horní hranice klesá, takže starší lidé hůře slyší vysoké frekvence. Lidé jsou poměrně citliví na fázový posun zvuku mezi levým a pravým uchem. Hlasitost člověk nevnímá lineárně s amplitudou vlnění, ale logaritmicky. Proto se hlasitost měří v decibelech, což je logaritmická jednotka.

Chceme-li pracovat se zvukem digitálně, je třeba jej nějak digitálně reprezentovat. Digitální přístroje by měly problém pracovat se zvukem jako se spojitou veličinou, proto je zvuk v digitální podobě reprezentován jako posloupnost jednotlivých vzorků.

Každý vzorek udává amplitudu v daném čase. U takovéto posloupnosti vzorků sledujeme především dvě vlastnosti.

První vlastností je vzorkovací frekvence. Ta udává, kolik vzorků bylo za jednu vteřinu pořízeno. Pokud je vzorkovací frekvence příliš nízká, začnou se ze záznamu ztrácet vyšší frekvence. Navíc se začne objevovat aliasing, který způsobí, že se v záznamu naopak začnou objevovat nižší frekvence, které v původním nahrávaném zvuku nebyly. Více o aliasingu budu psát dále. Příliš vysoká vzorkovací frekvence zase zvyšuje velikost záznamu, zvyšuje výpočetní náročnost zpracovávání takového signálu a klade vyšší technické nároky na aparaturu zajišťující převod zvuku na digitální signál, která takovou vzorkovací frekvenci musí podporovat.

Typická frekvence, se kterou se setkáváme u většiny zařízení běžných uživatelů je 44.1kHz. Tato frekvence byla používána u audio CD. Mezi další užívané frekvence patří např. 48kHz, 96kHz, nebo 1192kHz.

Druhou důležitou vlastností je bitová hloubka. Ta udává, v kolika bitech je uložena hodnota jednoho vzorku. Nízké hodnoty vnášejí do nahrávky šum.

Jeví se „více nekonkrétní“ a ztrácejí se v ní detaily. Zbytečně vysoká bitová hloubka opět zvětšuje velikost záznamu a stejně jako vysoká vzorkovací frekvence musí být podporovaná nahrávací aparaturou. Typickými hodnotami jsou 16b, nebo 24b.

Vzorky lze ukládat jako celá čísla, nebo jako čísla s plavoucí desetinnou čárkou, tvz. „floating point“ čísla. Oba tyto přístupy mají svoje výhody a nevýhody. Celá čísla jsou v celém rozsahu, který lze do daného počtu bitů uložit, distribuována rovnoměrně. Naopak možné hodnoty čísel s desetinnou čárkou jsou nejvíce koncentrované kolem nuly a s vyššími hodnotami jejich hustota exponencielně klesá. Použití čísel s plavoucí desetinnou čárkou nemá příliš smysl u vstupních a výstupních signálů systému, ovšem jistý smysl může mít uvnitř, kde probíhají mezivýpočty. Při výpočtech s floating point hodnotami není tak vysoké riziko přetečení a hlavně se zeslabením signálu tolik neztrácí jeho přesnost. U celočíselných signálů je ten problém, že čím zeslabenější jsou, tím menší část bitového rozsahu se používá, a tím nižší bitovou hloubku reálně má. Mezivýpočty ve floating point hodnotách tedy nemusí být normalizované, a i přesto nutně neztrácejí v takové míře svou kvalitu. Na druhou stranu je aritmetika s floating point hodnotami výpočetně náročnější a zároveň bude mít signál v základu teoreticky za použití stejného počtu bitů horší kvalitu, protože se některé bity použijí pro uložení exponentu. V praxi to ale pravděpodobně nebude mít reálný efekt, protože se na běžných architekturách typicky používají 32bit floating pointy, které mají přesnost 23bitů, což je dostačující pro drtivou většinu aplikací. Pokud by se ovšem použila tzv. poloviční přesnost, tedy 16bit floating point, tak bude přesnost každé hodnoty jen 10bit. To nelze přímo srovnávat s 10bit signálem používajícím celočíselné vzorkování, ovšem lze předpokládat, že kvalita bude horší, než kdyby se jednalo o celočíselný 16bit signál, který je řádně normalizován.

Je tedy zřejmé, že bitová hloubka a vzorkovací frekvence jsou dvě klíčové vlastnosti, které velmi ovlivňují výslednou kvalitu zpracovávaného zvuku.

2.1.1 Latence

Navrhujeme-li audio systém, hraje důležitou roli i latence daného systému. Latence je časový údaj, který nám říká, jak dlouhá doba uplyne mezi tím, kdy signál do systému vstoupí a tím, kdy ho (resp. jeho zpracovaná podoba) opustí. Nároky na latenci se mohou značně lišit dle toho, k čemu dané zařízení slouží. Pokud se jedná o CD přehrávač, nehraje latence téměř žádnou roli. V případě videopřehrávače už je potřeba dbát na to, aby latence audia a videa byla shodná a byly přehrávány synchronně. V tomto případě lze ovšem očekávat, že latence dekódování videa bude patrně větší, než latence v „audio části“ zařízení, tudíž v tomto případě budeme latenci zvuku pravděpodobně naopak přidávat. Zcela jiná je ovšem situace u jiných zařízení. Jakmile má být zařízení více interaktivní, začíná být latence velice důležitým faktorem. U zvukového výstupu osobního počítače latence typicky není úplně kritická, ovšem např. u

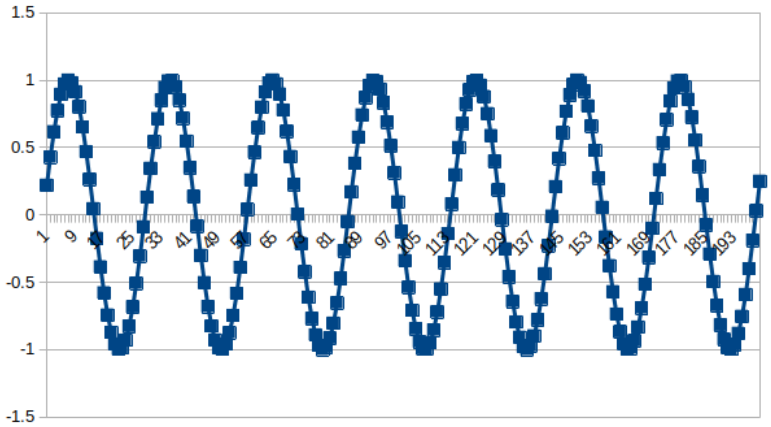
hraní střílečekmůže být nepříjemné, pokud má třeba zvuk výstřelu zpoždění za obrazem (obrazová latence je zejména u akčních her minimalizovaná, jak jen to prostředky dovolují), jedná se již o poměrně obtěžující jev, který kazí herní zážitek. Nejkritičtější je ovšem latence u hudebních nástrojů a aparatury pro živé hraní. Různí lidé jsou na latenci různě citliví, ale odhaduje se [1], že u elektronických kláves (u syntezátorů) lze tolerovat maximálně 23ms, u piána kolem 10ms, u kytary zhruba 6ms, u bicích 6ms a u vokálů pouze 3ms. Bavíme se zde o celkové latenci celého řetězce zařízení, tedy od zdroje (zvuku vstupujícího do mikrofonu, nebo stisknutí klávesy) až na konec (zvuk z reproduktorů přicházející k uchu hudebníka). Pokud by tedy např. reproduktorová výhybka přidávala jakoukoli signifikantní latenci, jednalo by se u reproduktorů pro obecné využití o velice špatnou reproduktorovou výhybku. Tím spíš, že klasické analogové reproduktorové výhybky prakticky žádnou latenci nemají.

Jak latence vzniká? V první řadě, pokud je na vstupu i na výstupu analogový signál, tak nějaký čas trvá převést ho do digitální podoby a poté zpět. Dále je potřeba ho předat z konvertoru do výpočetní jednotky, s každým vzorkem provést veškeré výpočty a poté jej přenést dále do výstupního konvertoru. Toto ovšem typicky nebývá největším zdrojem latence.

Největším zdrojem latence je typicky to, že zvuk obvykle není zpracováván po jednotlivých vzorcích, ale místo toho zpracovává vždy více vzorků najednou. To se dělá proto, aby se ušetřilo na režii a zvýšilo se tím množství vzorků, které je systém schopen za jednu vteřinu zpracovat (tedy umožnila se vyšší vzorkovací frekvence). Pokud se např. výpočet skládá ze tří částí A, B a C, kdy přechod z jedné části do druhé znamená jistou režii, je často výhodnější naakumulovat několik vzorků, a na všech hromadně nejdřív provést část A, poté část B, a poté teprve část C, než aplikovat A, B i C pro každý vzorek zvlášť. Dále pokud máme v zařízení procesor, které dělá střídavě i jiné věci, než jen zpracovávání zvuku, poté změna kontextu z jiné činnosti na zpracování zvuku představuje tak značnou režii, že je zcela určitě mnohem výhodnější zpracovávat vzorky po skupinách.

Při vzorkovací frekvenci 44.1kHz znamená 44 vzorků necelou 1ms zvuku. Při frekvenci 48kHz je 1ms záznamu reprezentovaná 48 vzorky. 128 vzorků při frekvenci 44.1kHz se rovná zhruba 2.9ms. 256 vzorků 5.8ms.

Představme si tedy, že máme systém, který má na vstupu převodník vstupního analogového signálu na digitální signál. Ten je následně zpracován a nakonec převodníkem převeden na analogový výstup. Zařízení pracuje se vzorkovací frekvencí 44.1kHz a využívá uvnitř buffery o velikosti 128 vzorků. Buffer tedy pojme zhruba 2.9ms zvuku. Pokud procesor provádí i jiné činnosti, musí vždy každých 2.9ms přepnout kontext, zpracovat přijaté vzorky, a poté se může vrátit k další činnosti. Jaký ale bude mít takový systém latenci? Nabízelo by se říci, že 2.9ms, ale takhle jednoduché to bohužel není. Celé to závisí na tom, jak je zařízení navrženo. Obecně lze ovšem předpokládat, že je nejdřív nutné buffer naplnit. To bude trvat 2.9ms. Poté je potřeba signál zpracovat. To musí nutně trvat kratší dobu, protože kdyby tomu tak nebylo, tak by



Obrázek 2.1: Vzorkování s dostatečnou frekvencí, bez aliasingu

po 2.9ms přišel další naplněný buffer a výpočetní jednotka by tedy nestíhala signál zpracovávat. Teoreticky je možné, že by měla výpočetní jednotka více jader a prováděla výpočet více bufferů paralelně, ovšem to je možné pouze tehdy, pokud nejsou výsledky výpočtů jednoho bufferu nutné pro počítání následujícího bufferu. Nakonec je signál odeslán na výstup, což může nějakou obtížněji specifikovanou dobu trvat. Celková latence tedy bude patrně vyšší, než délka zpracovávaných bufferů. Pro přesnější odhad je třeba podrobná znalost navrhovaného zařízení. Důležité je také následné empirické měření.

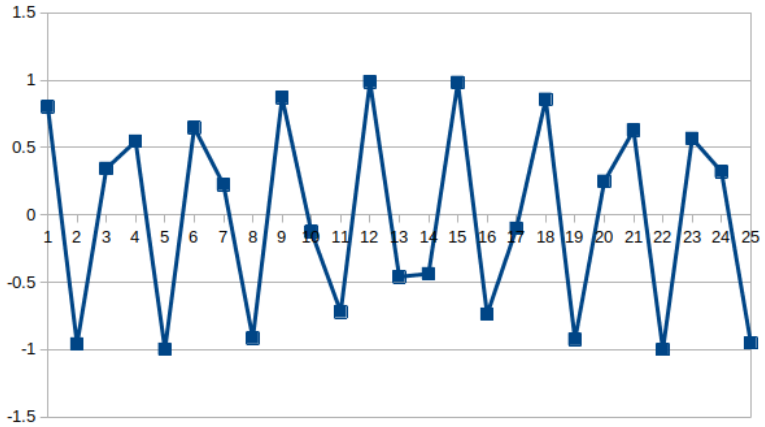
2.1.2 Aliasing

Při převodu analogového signálu na digitální může docházet k tvz. aliasingu. Aliasing je jev, kdy u zdiskretizovaného signálu přestane být možné odlišit od sebe jinak původně různé signály.

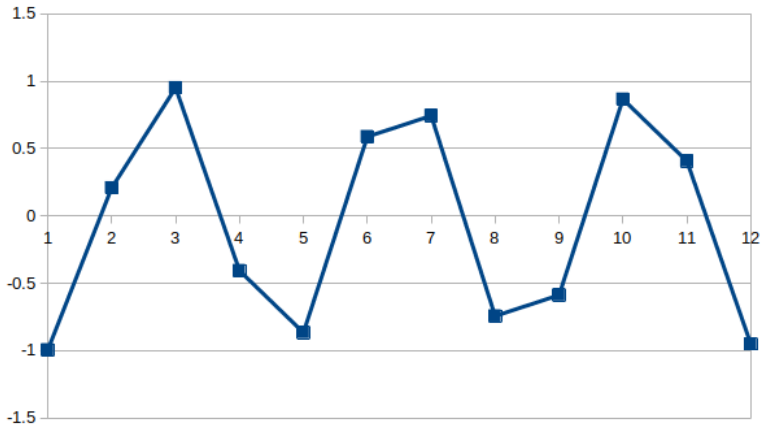
Je to vidět na tomto příkladu. Vidíme zde několik grafů stejné sinusoidy, která je pokaždé navzorkovaná s jinou frekvencí. První snímek dobře reprezentuje skutečný tvar původního signálu. Druhý již obsahuje sinusoidu nižší frekvence, která se ovšem v původním signálu nevyskytovala. Na třetím obrázku již signál vypadá úplně jinak, než signál původní.

Pokud je vzorkovací frekvence alespoň dvojnásobná oproti nejvyšší frekvenci vyskytující se ve vzorkovaném signálu, k aliasingu nedochází. [2, str. 82]

To, že se frekvence vyšší než je vzorkovací frekvence v zdiskretizovaném signálu ztratí, by se dalo snadno očekávat. Záludnost aliasingu je ovšem v tom, že do zdiskretizovaného signálu přidává frekvence, které v původním signálu vůbec nebyly.



Obrázek 2.2: Mírný aliasing



Obrázek 2.3: Významný aliasing

2.1.3 Reproduktorové výhybky

Protože je obtížné vytvořit reproduktor, který dostatečně reprodukuje celé slyšitelné spektrum, konstruují se reprobedny obvykle z vícero reproduktorů. Každý reproduktor poté pokrývá určitou část spektra. Je tedy nutné vstupní signál nějak rozdělit mezi jednotlivé reproduktory, aby např. reproduktor zaměřený na výšky nedostával na vstupu basovou část spektra. K tomuto rozdělení slouží tzv. reproduktorová výhybka. Reproduktorová výhybka je tedy elektrický obvod, který vstupní signál rozděluje na několik frekvenčních pásem.

2.2 Lineární systémy

Lineární systém je takový systém, který splňuje následující vlastnosti: aditivita a homogenita.

Aditivita znamená, že je výsledek stejný, pokud do systému pošleme součet dvou signálů, nebo pokud každý signál pošleme do systému zvlášť a sečteme výstupy.

$$f(x_1) + f(x_2) = f(x_1 + x_2)$$

Homogenita znamená, že vynásobíme-li vstup nějakou konstantou k, výstup se také změní k-krát.

$$k \cdot f(x) = f(k \cdot x)$$

Další vlastnost, která s linearitou přímo nesouvisí, ale u systémů, se kterými budeme pracovat, je velmi častá, je časová invariance. Ta znamená, že systém nemění své chování v čase. Jinými slovy, pokud vstupní signál posuneme v čase, výstup se posune stejně, ale jinak zůstane stejný.

2.2.1 Jednotkový skok a jednotkový impulz

Jednotkový skok je takový signál, který má zprvu hodnotu 0 a od nějaké chvíle dál hodnotu 1. Jednotkový impulz je takový signál, který má nultou hodnotu rovnu jedné, a všechny další hodnoty má nulové. Takový signál označujeme jako δ.

Na těchto signálech je zajímavé to, že lze velmi snadno libovolný signál rozložit na součet jednotkových skoků přenásobených konstantami, nebo jednotkových impulzů přenásobených konstantami.

Rozložení na jednotkové impulzy je velmi přímočaré. Signál x dlouhý n vzorků rozdělíme na n signálů stejné délky a_0 až a_{n-1}. Každý signál poté bude obsahovat samé nuly, pouze na n-té pozici bude mít hodnotu $x[n]$. Jedná se tedy vždy o posunutý jednotkový impulz přenásobený konstantou.

$$a_n[i] = x[n] \cdot \delta[i + n]$$

Převod na součet jednotkových skoků je jen o trochu složitější. Stejně jako u předchozího případu budeme signál dlouhý n vzorků rozkládat na n signálů, které budou v čase posunuté jednotkové skoky přenásobené konstantou. Tato konstanta ale nebude rovna odpovídajícímu vzorku původního signálu, ale rozdílu vůči předchozímu vzorku.

$$a_n[i] = (x[n] - x[n-1]) \cdot 1[i + n]$$

2.2.2 Impulzní odezva a konvoluce

Pokud přivedeme jednotkový impulz jako vstup do lineárního systému, odpovídající výstup označujeme jako impulzní odevzu daného systému. Označujeme ji h.

Impulzní odezva je velice užitečná, protože z ní lze odvodit, jak se daný systém bude chovat k libovolnému signálu.

Díky homogenitě lineárního systému víme, že přenásobíme-li jednotkový impulz konstantou, výsledkem bude impulzní odezva přenásobená tou samou konstantou. Stejně tak díky časové invarianci víme, že posuneme-li jednotkový impulz v čase, získáme impulzní odezvu stejně posunutou. Jsme tedy pouhým přenásobením konstantou a posunem v čase schopni získat odezvu na libovolný impulz.

Pokud jsme rozložili vstupní signál na jednotlivé impulzy, můžeme tedy snadno získat výstup ze systému pro každý jednotlivý impulz.

Díky aditivitě lineárních systému víme, že sečteme-li takto vzniklé odezvy, získáme stejný signál, jaký bychom získali, kdyby do systému vstoupil původní vstupní signál. Nemusíme tedy znát vnitřní mechanismy lineárního systému, abychom pro vstupní signál získali odpovídající výstup. Stačí nám na to jeho impulzní odezva, která plně popisuje chování lineárního systému.

Toho využívá tzv. konvoluce. Jedná se o způsob, jak za pomoci impulzní odezvy libovolného lineárního systému vytvořit ze vstupního signálu signál výstupní. Tato metoda je založená právě na rozkládání vstupního signálu na jednotlivé impulzy a následné sčítání odezvy každého impulzu do výsledného signálu.

Obvyklá implementace tohoto algoritmu se ovšem na problém dívá z druhé strany. Každý výsledný vzorek počítá jako součet vzorků ze vstupního signálu vážených podle jednotlivých vzorků impulzní odezvy.

$$y[i] = \sum_{j=0}^{n-1} h[j] \cdot x[i-j]$$

Tento způsob je velice přímočarý, ale pro dlouhé impulzní odezvy může být poměrně náročný na výpočetní výkon.

2.2.3 Frekvenční odezva

Jednou z vlastností lineárních systému je to, že pokud na vstup přivedeme signál ve tvaru sinusoidy, výstupem bude opět sinusoida. Ta může být fázově posunutá a může mít jinou amplitudu než vstup, ovšem bude mít vždy stejnou frekvenci. Sinusoida je jediný tvar signálu, který tuto vlastnost má.

Další způsob, jak popsat lineární systém je tedy tzv. frekvenční odezva. Ta nám právě říká, jakým způsobem daný lineární systém reaguje na sinusoidy. Každé frekvenci přiřazuje fázor, který vyjadřuje změnu amplitudy, a fázový posun, který systém dané frekvenci způsobí. Zajímavé je, že pokud provedeme Fourierovu transformaci nad impulzní odezvou systému, získáme tím jeho frekvenční odezvu. Stejně tak lze inverzní Fourierovou transformací převést frekvenční odezvu na impulzní odezvu. Z toho vyplývá, že stejně jako impulzní odezva, i frekvenční odezva zcela popisuje daný lineární systém.

2.3 Transformace signálů

2.3.1 Fourierova transformace

Nejpřirozenější způsob, jak se na signály obvykle díváme, je ten, že zachytíme, jak se jeho hodnota mění v čase. To ale není zdaleka jediný způsob, jak se na signál dívat. Jedním z dalších způsobů je převedení signálu do tzv. „frekvenční domény“. To znamená, že se nedíváme na to, jak se signál mění v čase, ale místo toho se díváme na to, jak lze tento signál vyjádřit jako součet sinusoid.

Sinusoidy mají tu vlastnost, že součet dvou sinusoid o stejné frekvenci je opět sinusoida o stejné frekvenci. Jediné, v čem se mohou lišit, jsou jejich amplitudy a fázový posun. Z toho plyne, že se na signál můžeme dívat tak, že je v něm každá frekvence obsažena právě jednou. U každé frekvence nás tedy zajímá pouze její amplituda a fázový posun.

S amplitudou budu pracovat jako s bezrozměrnou veličinou a fázový posun budu vyjadřovat jako úhel v radiánech. Tyto dvě informace si lze představit jako vektor, kdy amplituda A vyjadřuje délku tohoto vektoru a fázový posun P vyjadřuje úhel s osou X. Takovému vektoru říkáme fázor.

$$\vec{v} = (A \cdot \cos(P), A \cdot \sin(P))$$

Fázor můžeme obdobným způsobem vyjádřit i jako komplexní číslo.

$$v = A \cdot \cos(P) + j \cdot A \sin(P)$$

Toto vyjádření budu používat nejčastěji.

Fourierova transformace je transformace, která převádí signál mezi časovou a frekvenční doménou. Je definovaná následovně:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt$$

Jako f označujeme signál v časové doméně. Jako F označujeme signál ve frekvenční doméně. Podoba tohoto vzorce na první pohled nevypadá příliš intuitivně, proto se pokusím její podobu mírně rozklíčovat.

Parametr ω je frekvence přenásobená $2 \cdot \pi$. Tato substituce se používá pouze pro zkrácení matematických zápisů.

$$\omega = 2 \cdot \pi \cdot freq$$

Dále vidíme, že integrujeme podle proměnné t v rozsahu od $-\infty$ do $+\infty$. V tomto integrálu vidíme, že původní signál f násobíme $e^{-j\omega t}$. Ač se to může zdát zvláštní, tak tato exponenciela je ve skutečnosti sinusoidou. To nám říká Eulerův vzorec:

$$e^{jx} = \cos(x) + j \sin(x)$$

Fourierova transformace nám tedy zjišťuje korelaci původního signálu se sinusoidou o dané frekvenci. Výsledkem je navíc komplexní číslo, takže se nedozvíme pouze amplitudu, ale také fázor takové sinusoidy.

2.3.2 Laplaceova transformace

Laplaceova transformace je zobecnění Fourierovy transformace. Z Fourierovy transformace vznikne pouhým přidáním $e^{-j\omega t}e^{-\sigma t}$ a přidáním σ jako druhého argumentu.

$$F(\sigma,\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}e^{-\sigma t}dt$$

Vidíme, že pro $\sigma = 0$ je $e^{-\sigma t}$ rovno jedné a je pro tuto hodnotu Laplaceova transformace ekvivalentem Fourierovy transformace.

Pro kompaktnější zápis je dvojice argumentů σ a ω nahrazena jedním komplexním číslem s:

$$s = \sigma + j\omega$$

Tím dostáváme výslednou podobu Laplaceovy transformace.

$$F(s) = \int_{-\infty}^{\infty} f(t)e^{-st}dt$$

Jaká je ovšem motivace za tím, používat takovou transformaci? Vágně řečeno, Fourierova transformace se snaží vstupní signál korelovat se sinusoidami. Laplaceova transformace se jej snaží korelovat se sinusoidou přenásobenou exponencielou (tedy komplexní exponencielou, jak jsme se dozvěděli z Eulerova vzorce). Laplaceova transformace nám tedy umožňuje lépe pochopit signály, které jsou tvořeny součtem komplexních exponenciel. To může znít, jako zvláštně specifická skupina signálů, ovšem skutečnost je taková, že impulzní odezva systémů, které jsou popsány soustavou diferenciálních rovnic mají právě takovou odezvu. A diferenciálními rovnicemi lze popsat většinu systémů, které lze v přírodě a elektrotechnice potkat.[3, Str. 581]

Říkáme, že Laplaceova transformace převádí signál z časové domény, do tzv. s-domény.

2.3.3 Přenosová funkce

Další ze způsobů, jak popsat lineární systém je pomocí tzv. přenosové funkce. Přenosová funkce je definovaná jako podíl Laplaceovy transformace výstupu a Laplaceovy transformace vstupu systému.[3, Str. 594]

$$H(s) = \frac{L(Y)}{L(X)}$$

Přenosovou funkci lze navíc získat jako Laplaceovu transformaci impulzní odezvy systému. Obdobně jako lze Fourierovou transformací impulzní odezvy získat frekvenční odezvu.

Typická podoba přenosové funkce, se kterou se v tomto textu setkáme, je podíl dvou polynomů.

2.3.4 Kombinování přenosových funkcí

Je užitečné vědět, jak z více propojených systémů udělat jeden. Máme-li dva lineární systémy, u kterých známe jejich přenosové funkce, je velmi snadné získat přenosovou funkci systému, který vznikne sériovým, nebo paralelním zřetězením takovýchto systémů.[4]

Pokud chceme několik systémů zřetězit za sebe tak, že výstup jednoho je vstupem následujícího, stačí mezi sebou jejich přenosové funkce vynásobit.

$$H(s) = H_1(s) \cdot H_2(s) \cdot H_3(s) \dots H_n = \prod_{i=1}^{n} H_i(s)$$

Pro paralelní zapojení, kde je vstupní signál poslán jako vstup více systémům a jejich výstupy jsou následně sečteny (odečteny), stačí jednotlivé přenosové funkce sečíst (odečíst).

$$H(s) = H_1(s) + H_2(s) + H_3(s) + H_n = \sum_{i=1}^{n} H_i(s)$$

$$H(s) = H_1(s) - H_2(s)$$

Když se na tento postup podíváme z opačné strany, vidíme, že lze jeden lineární systém podle stejných pravidel rozdělit na více systémů. To může být velice užitečné, protože tyto „podsystémy“ mohou mít vhodnější vlastnosti pro jejich implementaci. Častým postupem je tedy to, že se jeden filtr rozdělí na kaskádu několika jednodušších filtrů.

2.3.5 Póly, nuly a stabilita systému

Představme si, že máme přenosovou funkci ve tvaru podílu dvou polynomů:

$$H(s) = \frac{a_0 + a_1 s + a_2 s^2 \dots}{b_0 + b_1 s + b_2 s^2 \dots}$$

Na takovéto přenosové funkci je zajímavé, že polynom v čitateli i jmenovateli bude mít kořeny. Tedy komplexní čísla, při kterých je hodnota tohoto polynomu rovna nule. Takovýmto hodnotám se říká „póly a nuly“. Protože se jedná o komplexní čísla, mluvíme o těchto hodnotách často tak, jako by to byly body v R^2. Kořenům polynomu čitatele říkáme nuly, protože v těchto bodech má přenosová funkce hodnotu 0. Kořenům ve jmenovateli říkáme póly, protože se hodnota přenosové funkce v jejich okolí limitně blíží nekonečnu.[3, Str. 596] Samozřejmě při těchto tvrzeních neuvažuji případy, kdy se nějaká nula s nějakým pólem rovnají.

Proto se často setkáváme také s přenosovou funkcí v následujícím tvaru:

$$H(s)\frac{(s-z_0)(s-z_1)(s-z_2)\dots}{(s-p_0)(s-p_0)(s-p_0)\dots}$$

V tomto tvaru jsou všechny póly a nuly na první pohled hezky viditelné.

Protože v takové přenosové funkci není nic jiného, než tyto součiny obsahující polohy nul a pólů, je zjevné, že pozice všech pólů a nul plně popisují přenosovou funkci jako celek. Z tohoto důvodu jsou často přenosové funkce znázorňovány diagramem ve formě komplexní roviny, ve které jsou pomocí kroužků vyznačeny nuly a pomocí křížků vyznačeny póly.

Poloha pólů a nul v s-doméně nám dává jistý kontext pro tvar přenosové funkce. Protože je přenosová funkce podél imaginární osy rovna frekvenční odezvě (viz to, jak Laplaceova transformace vychází z frekvenční odezvy), pomáhají nám polohy pólů a nul pochopit i tvar frekvenční odezvy.

Jak jsem již zmínil, Laplaceova transformace impulzní odezvy systému je rovna přenosové funkci systému. Když jsem popisoval, jakým způsobem vzniká Laplaceova transformace z Fourierovy transformace, mluvil jsem o tom, že se u signálu zkoumá exponenciální pokles (popř. vzestup) jednotlivých sinusoid obsažených v signálu. Lajcky řečeno, signál, který má exponenciálně zvyšující se amplitudu, bude mít v s-doméně zvýšené hodnoty v té polovině komplexní roviny, kde jsou reálné části kladné. Naopak signál, jehož amplituda exponencielně klesá bude mít zvýšené hodnoty v opačné, tedy záporné, polovině komplexní roviny. Pokud jsou v signálu obsaženy sinusoidy s konstantní amplitudou, „ukážou“ se v s-doméně na imaginární ose v místě, kde je reálná část rovna nule.

Z této představy lze intuitivně vytušit, že je-li pól přenosové funkce v pravé polovině komplexní roviny (tedy s kladým reálnými částmi), odpovídající impulzní odezva bude mít s časem exponenciálně se zvyšující amplitudu.[3, Str. 609] Systém s takovouto impulzní odezvou pro libovolný nenulový vstupní signál vygeneruje exponenciálně se zvětšující výstupní signál. O takových systémech říkáme, že jsou nestabilní. Při konstrukci filtrů je třeba dávat si na nestability pozor, protože v opačném případě může být výsledek značně ohlušující.

2.3.6 Z-transformace

Z-transformace je transformace vycházející z Laplaceovy transformace. Tak, jako se Laplaceova transformace používá k návrhu analogových filtrů, z-transformace se používá k návrhu digitálních filtrů. Zatímco Laplaceova transformace se používá při práci se spojitými signály, z-transformace se používá při práci se zdiskretizovanými signály.

Začněme s touto podobou Laplaceovy transformace, která místo komplexního parametru s používá dva reálné parametry σ a ω.

$$X(\sigma,\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}e^{-\sigma t}dt$$

Pro úpravu této rovnice na z-transformaci učiníme několik změn. Protože pracujeme se zdiskretizovanými signály, změníme integrál na sumu. Dále nahradíme $e^{-\sigma t}$ proměnnou r. Tím dostáváme následující rovnici:

$$X(r,\omega) = \sum_{n=-\infty}^{\infty} x[n] \cdot r^n \cdot e^{-j\omega n}$$

Nakonec oba dva parametry r a ω nahradíme jediným komplexním parametrem z:

$$z = re^{-j\omega}$$

Čímž dostáváme finální podobu z-transformace:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] \cdot z^{-n}$$

Stejně jako jsme u Laplaceovy transformace mluvili o s-doméně, budeme u z-transformace mluvit o z-doméně.

Efektivně jsme provedli tři věci. Přešli jsme ze spojitých signálů na diskrétní. Rychlost útlumu již neměníme lineárně, ale místo toho měníme základ této exponenciály. A do třetice jsme v jistém smyslu přešli z kartézských souřadnic do polárních souřadnic. Co tím myslím bude více jasné, když se podíváme na to, jakým způsobem se pozice pólů a nul v s-doméně mapují na pozice v z-doméně.

Jak je vidět na této stránce [3, Str. 609], namapuje se imaginární osa s-domény na jednotkový kruh v z-doméně.

2.3.7 Bilinerární transformace

Popsali jsme si Laplaceovu transformaci a Z-transformaci. Ukázali jsme si, že jsou si velmi podobné. Díky tomu je možné mezi nimi přecházet. To se hodí například tehdy, chceme-li převést analogový filtr na digitální filtr. Jedním ze způsobů, jak tento převod provést, je tzv. Bilineární transformace. Její použití je velmi snadné. Stačí na přenosovou funkci použít následující substituci:[5]

$$s = c \cdot \frac{1 - z^{-1}}{1 + z^{-1}}, c > 0, c = 2/T$$

$$z = \frac{1 + s/c}{1 - s/c}, c > 0, c = 2/T$$

Kde T je vzorkovací perioda.

2.4 Přenos a diferenciální rovnice

2.4.1 Převod přenosové funkce na diferenciální rovnici

Odvození diferenciální rovnice z přenosové funkce je v případě přenosových funkcí, se kterými budeme pracovat, naštěstí velmi přímočaré. Jak se můžeme dočíst v těchto výukových materiálech[6], přenosová funkce odpovídá

$$H(s) = \frac{b_0 + b_1 s + b_2 s^2 \dots}{a_0 + a_1 s + a_2 s^2 \dots}$$

diferenciální rovnici

$$a_0 y(t) + a_1 y'(t) + a_2 y''(t) \dots = b_0 x(t) + b_1 x'(t) + b_2 x''(t) \dots$$

2.4.2 Převod obyčejné diferenciální rovnice na soustavu obyčejných diferenciálních rovnic prvního řádu

Protože budeme potřebovat řešit diferenciální rovnice numericky, potřebujeme způsob, jak rovnice vyšších řádů převést na soustavu rovnic prvního řádu. Metodu, kterou zvolíme, je tzv. metoda postupné integrace.

Postup této metody je naznačen zde [7]. Vždy nejdříve osamostatníme nejvyšší řád rovnice, který poté nahradíme novou proměnnou. Celý postup je nejlepší vysvětlit na příkladu. Proto ho předvedu na příkladu z výše uvedeného zdroje.

Mějme následující rovnici:

$$y'' + 2y' + y = x'' + 3x' + 2x$$

Derivaci nahradíme za násobení proměnnou p:

$$p^2 y + 2py + y = p^2 x + 3px + 2x$$

Algebraickými úpravami dostaneme:

$$p^2 y = p^2 x + p(3x - 2y) + (2x - y)$$

Když tuto rovnici vydělíme p (což v kontextu původní diferenciální rovnice znamená integraci), dostaneme:

$$py = px + (3x - 2y) + \frac{1}{p}(2x - y)$$

Vidíme, že jedna závorka je násobená $\frac{1}{p}$. Z tohoto sčítance si uděláme novou proměnnou $w1$. Tím získáme soustavu rovnic:

$$w_1 = \frac{1}{p}(2x - y)$$

$$py = px + (3x - 2y) + w_1$$

První rovnici opět vydělíme p, čímž získáme soustavu rovnic:

$$w_1 = \frac{1}{p}(2x - y)$$

$$y = x + \frac{1}{p}(3x - 2y + w_1)$$

Ze sčítance, který obsahuje $\frac{1}{p}$ uděláme opět novou proměnnou, kterou pojmenujeme $w2$. Tím získáme soustavu:

$$w_1 = \frac{1}{p}(2x - y)$$

$$w_2 = \frac{1}{p}(3x - 2y + w_1)$$

$$y = x + w_2$$

Když nyní pro změnu první dvě rovnice vynásobíme p, získáme tuto soustavu rovnic ve tvaru, který je vhodný pro numerické výpočty pomocí metod Runde-Kutta, nebo pomocí Eulerovy metody popsané dále v této práci.

$$pw_1 = (2x - y)$$

$$pw_2 = (3x - 2y + w_1)$$

$$y = x + w_2$$

Máme vyjádřené derivace pomocných proměnných. Nikde se nevyskytuje derivace vstupu x. Máme vyjádřen výstup y. Nikde, krom levých stran prvních dvou rovnic se nevyskytují derivace (násobení proměnnou p).

2.5 Filtry

Základním nástrojem při zpracovávání signálů je filtr. V tomto textu budeme mluvit o lineárních filtrech. To jsou filtry, které jsou zároveň lineárními systémy. Filtry se používají pro dva základní úkony. Jedním úkonem je rozdělení signálu na více signálů. Druhým možným úkolem je zrekonstruování signálu.

Co si pod tím představit? Rekonstrukci signálu si můžeme představit tak, že máme nějaký původní signál, který prošel nějakou změnou. My se poté snažíme z takto pozměněného signálu získat signál původní. Pokud změnu způsobil nějaký lineární systém, stačí nám znát jeho impulzní odezvu. S její pomocí lze snadno odvodit inverzní systém, který tuto změnu vrátí zpět. Pokud impulzní odezvu neznáme, nebo se nejedná o lineární systém, můžeme stále navrhnout takový filtr, který má vlastnosti, že lze původní signál alespoň částečně obnovit.

Co se rozdělování signálů dle frekvence týče, snažíme se ze signálu filtrovat nějakou část frekvenčního spektra. Hezkým příkladem jsou právě reproduktorové výhybky, kdy chceme do jednoho reproduktoru poslat pouze basy, do druhého pouze středy a do třetího pouze výšky.

2.5.1 Typy filtrů

Jedno ze základních dělení filtrů je podle tvaru jejich frekvenční odezvy[3, Str. 268]. Ta může být samozřejmě teoreticky téměř libovolná, ovšem několik základních tvarů označujeme následovně:

- Low pass / High cut
- High pass / Low cut
- Band pass
- Band reject

Každý z těchto tvarů má celou řadu využití. Pro reproduktorovu výhybku potřebujeme low pass, hight pass a band pass filtry. Band reject se zase hodí například na potlačování různých rezonančních frekvencí, nebo třeba frekvencí způsobujících zpětnou vazbu při ozvučování živého koncertu.

Dále můžeme rozdělovat filtry na ty, které jsou optimalizované v časové doméně, frekvenční doméně, případně na nějakou kombinaci obojího. Co to znamená?

Tady záleží na tom, jaký typ informace je v signálu uložen. Jestli nás na něm zajímají hodnoty v jednotlivých místech, nebo to, jak se mění. Jinými slovy to, jestli je zajímavá informace obsažená v časové nebo frekvenční doméně. Pokud například budeme v pravidelných intervalech měřit teplotu v místnosti, je tou zajímavou informací spíše konkrétní hodnota teploty v konkrétní dobu, tedy informace v časové doméně. Naopak pokud bychom zaznamenali třeba vlnění struny na kytaře, tak nám její konkrétní pozice v danou chvíli nic moc zajímavého neřekne. Daleko zajímavější je zjistit, jakou frekvencí kmitá a jak moc jsou ve vlnění zastoupené různé harmonické frekvence. V tomto případě nás tedy více zajímají informace obsažené ve frekvenční doméně. Určitě se ovšem najdou příklady, kdy je zajímavá informace obsažená v obou doménách. Od toho, jaká informace nás zajímá více, se odvíjí to, jaké vlastnosti filtru se budeme snažit optimalizovat.

Pokud nás zajímají informace ve frekvenční doméně a snažíme se oddělit jeden pás frekvencí od druhého, očekáváme, že filtr neovlivní frekvence, které má nechat projít. Frekvenční odezva v tomto pásmu frekvencí by tedy měla být co nejvíce plochá. Dále chceme, aby frekvence, které mají být potlačené, byly potlačené co možná nejvíce. Do třetice nás zajímá, jak vypadá přechod mezi těmito dvěma pásmy. Typicky chceme, aby tento přechod byl co nejrychlejší, aby pásmo frekvencí, které je „utlumeno jen tak trochu“, mělo co nejmenší šířku.

Pokud nás naopak zajímají více informace v časové doméně, tak je naším cílem primárně to, aby jednotlivé hodnoty byly co nejblíže původnímu signálu, ze kterého se např. snažíme odfiltrovat rušení. To znamená, že nás u filtru nezajímá ani tak to, jak se chová ve frekvenční doméně, jako spíš to, aby způsoboval pokud možná co nejmenší překmity v časové doméně.

Problém je, že tyto dva požadavky jdou při návrhu filtru proti sobě, takže je nutné optimalizovat daný filtr na to, na co bude použit.

Pokud navrhujeme filtr pro filtrování audia, zajímají nás informace ve frekvenčním spektru a podle toho jej musíme optimalizovat.

Dalším faktorem, který může hrát v určitých aplikacích roli, je to, jak filtr mění fázi jednotlivých frekvencí.

2.5.2 Digitální filtry

Digitální filtry rozdělujeme na FIR filtry a IIR filtry. FIR filtry, tedy „finite impulse response“ mají konečně dlouhou impulzní odezvu. Oproti tomu IIR filtry, tedy „infinite impulse response“ mají impulzní odezvu nekonečně dlouhou. V kontextu digitálních filtrů vždy mluvíme o zdiskretizovaných signálech.

2.5.3 FIR filtry

Jak takový filtr implementovat? Nejpřímočařejší způsob, jak implementovat FIR filtr, je konvoluce, kterou jsem již popisoval dříve. Stačí mi impulzní odezva filtru, který chci uplatnit. Impulzní odezvu mohu odvodit z frekvenční odezvy pomocí fourierovy transformace. Tímto postupem lze implementovat libovolný FIR filtr, ovšem výpočetní náročnost konvoluce je závislá na délce impulzní odezvy, a takový filtr tedy může být dosti výpočetně náročný. Pro spočtení jednoho vzorku výstupního signálu je třeba vzít každý vzorek z impulzní odezvy, vynásobit jej vzorkem ze vstupního signálu a tyto součiny sečíst. Pro n vzorků výstupního signálu a m vzorků impulzní odezvy to je $n{\cdot}m$ operací násobení a sčítání, neboli $n \cdot m$ operací „multiply-accumulate“. Implementace pomocí konvoluce je tedy jednoduchá a flexibilní, ovšem výpočetně náročná.

Speciálním případem konvolučního FIR filtru je tzv. „moving averge“, tedy klouzavý průměr[3, Str. 277]. Ten každý vzorek vypočte jako průměr několika sousedních vstupních vzorků.

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j]$$

Alternativně lze vstupní vzorky pro průměrování brát symetricky kolem výstupního vzorku, což eliminuje posun, který by tento filtr jinak způsobil:

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j-\frac{M}{2}]$$

Výhodou tohoto filtru oproti obecným konvolučním FIR filtrům je jeho výpočetní nenáročnost. I když je samozřejmě možné implementovat jej konvolucí jako jakýkoli jiný FIR filtr, tento konkrétní filtr lze implementovat efektivněji.

Ve skutečnosti už implementace konvolucí může být značne optimalizována. Impulzní odezva tohoto filtru se skládá pouze z hodnot 0 a z M hodnot $\frac{1}{M}$. Když tedy počítáme výstupní vzorek, nemusíme každý vstupní vzorek násobit vzorkem z impulzní odezvy. Vybrané vzorky ze vstupního signálu se jednoduše sečtou a následně se součet vydělí M, což ušetří spoustu násobení.

Optimalizací lze ovšem odstranit i sčítání všech vstupních vzorků. Všimněme si, že když počítáme klouzavý průměr, tak většina vzorků aktuálně zahrnuta do průměrování je stejná, jako při výpočtu minulého výstupního vzorku. Ve skutečnosti se „okno průměrování“ vždy posune o jeden vzorek dále. Tedy všechny vzorky zůstanou stejné, jen jeden ubyde a jeden přibyde. Pokud si tedy budeme pamatovat sumu vstupních vzorků v průměrovaném okně z minulého výpočtu, stačí pouze jeden vzorek odečíst, nový vzorek přičíst a získáme sumu pro výpočet následujícího vzorku. Když pomineme výpočet prvního vzorku, tak výpočet každého dalšího vzorku znamená jedno sčítání, jedno odčítání a jedno dělení.

Tento filtr je ovšem optimalizovaný na výkon v časové oblasti, nikoli frekvenční. Proto pro využití v manipulaci se zvukovými signály typicky není příliš vhodný. Dalším problémem je, že výše popsaná optimalizace může mít problémy, pokud je implementována pomocí čísel s plavoucí desetinnou čárkou. Floating point výpočty totiž mají při výpočtech nepřesnosti. Jinými slovy nutně neplatí, že $x + y - x = y$. To znamená, že pokud je vzorek přidán do celkové sumy a následně je později ze sumy zase odečten, může se stát, že vlivem nepřesností v sumě, laicky řečeno „trocha z toho vzorku zůstane“. Pokud se tyto nepřesnosti začnou ve filtru kumulovat, projeví se to ve formě artefaktů na výstupním signálu. Například ve formě tzv. „DC offsetu“, neboli tím, že bude k výstupnímu signálu přičtena nějaká konstanta. Tento jev se navíc může postupně zhoršovat. Pokud jsou ovšem použity pro výpočty celá čísla, tak nejen, že jsou výpočty obecně rychlejší, ale k podobným problémem s nepřesnostmi nedochází, protože aritmetika s celými čísly na nepřesnosti netrpí.

Mezi další druhy filtrů patří např. tzv. Windowed-sync filtry, nebo filtry založené na algoritmu FFT, tedy rychlé Fourierově transformaci.

2.5.4 IIR filtry

V této části budu mluvit o filtrech, které mají nekonečně dlouhou impulzní odezvu, konkrétně tzv. rekurzivních filtrech.[3, Str. 219]

Rekurzivní filtry jsou v úplném základu založeny také na kovoluci. Každý nový vzorek se ovšem neskládá pouze z váženého součtu jiných vstupních

vzorků, ale i váženého součtu několika již spočtených výstupních vzorků. Tedy pro spočtení vzorku $y[n]$ použijeme třeba následující výpočet:

$$y[n] = a_0x[n] + a_1x[n-1] + a_2x[n-2] + \ldots + b_1y[n-1] + b_2y[n-2] + \ldots$$

Rekurzivní filtr je tedy určen několika parametry a_0, a_1, $a_2\ldots$ a několika parametry b_1, $b_2\ldots$ Všimněme si, že zde není žádný parametr b_0, protože to by znamenalo, že se k výpočtu hodnoty $y[n]$ používá samotná hodnota $y[n]$, což je pochopitelně nesmysl.

Jak jsem již naznačil, tak pokud jsou všechny parametry b_1, $b_2\ldots$ rovny 0, stanese z filtru obyčejný FIR filtr implementovaný konvolucí. Jinak má ovšem typicky impulzní odezvu v podobě součtu komplexních exponenciel. Jinými slovy je to součet sinusoid, které s časem exponenciálně snižují (nebo zvyšují, případně mají konstantní) amplitudu.

Všimněme si, že rekurzivní filtr je stejně výpočetně náročný, jako konvoluce s impulzní odezvou o délce pouhých nízkých jednotek vzorků. Přesto se chová jako filtr s mnohem delší impulzní odezvou.

Teoreticky by tedy šlo říci, že IIR filtry mohou implementovat impulzní odezvy, které pomocí FIR filtrů implementovat nelze. Ovšem pokud se impulzní odezva skládá ze součtu komplexních exponenciel, tak jsou tři možnosti. Buď amplituda takovéto odezvy exponenciálně roste, zůstává konstantní do nekonečna, nebo exponenciálně klesá. První případ znamená nestabilní filtr, druhý případ v praxi kvůli nepřesnostem výpočtů stěží nastane a těžko by se pro něj hledalo praktické využití. Ve třetím případě amplituda impulzní odezvy dříve nebo později klesne natolik, že se ztratí v šumu způsobeným výpočetními nepřesnostmi, takže ji lze od určité doby zaokrouhlit na nulu, a tím z ní efektivně udělat konečnou impulzní odezvu. Jinými slovy v praxi je i „nekonečná impulzní odezva“ v jistém smyslu buď konečná, nebo nemá praktické využití. Ovšem implementace konvolucí by byla nesrovnatelně více náročná.

Jak ovšem získat vhodné parametry a a b? Zde nám bude užitečná z-transformace. Jak se dočteme v [3, 610], tak přenosová funkce takového filtru má podobu

$$H[z] = \frac{a_0 + a_1z^{-1} + a_2z^{-2} + a_3z^{-3} + \ldots}{1 - b_1z^{-1} - b_2z^{-2} - b_3z^{-3} - \ldots}$$

Přenos v této formě tedy koeficienty rekurzivního filtru přímo obsahuje. Mějme ovšem na paměti, že zde není použita Laplaceova transformace a s-doména, ale Z-transformace a z-doména. Typickým způsobem, jak přenos v této podobě získáme, je ten, že nadesignujeme filtr v podobě pozic pólů na nul.

$$H[z] = \frac{(z - z_1)(z - z_2)(z - z_3)\ldots}{(z - p_1)(z - p_2)(z - p_3)\ldots}$$

Z této formy poté pomocí algebry lze převést přenos do formy výše, která v sobě obsahuje přímo parametry rekurzivního filtru. Opačný postup je obtížnější, protože vyžaduje faktorizaci polynomů, což je obecně obtížný problém.

2.6 Numerické metody řešení diferenciálních rovnic prvního řádu

Jedním ze způsobů, jak řešit diferenciální rovnice, je řešit je numericky. Tyto metody představují pouze přibližné řešení, nikoli matematicky exaktní řešení. Lze je ovšem relativně snadno a především rychle provádět na procesorech. Metod numerického řešení je několik. Mají různou přesnost, různou výpočetní náročnost, případně další vlastnosti. Je tedy vždy nutné najít nějaký vyhovující kompromis.

2.6.1 Eulerova metoda

Eulerova metoda je nejjednodušší metodou numerického řešení diferenciálních rovnic jedné proměnné. Snaží se z funkční hodnoty a derivace v daném bodě pomocí lineární extrapolace odhadnout funkční hodnotu následujícího bodu.

Konkrétně:

$$y(t+h) = y(t) + h \cdot f(t, y(t))$$

Kde h je délka časového kroku, t je aktuální čas, $f(t.y(t))$ je derivace funkce y v čase t při funkční hodnotě $y(t)$. Tato metoda má tu nevýhodu, že bere v potaz pouze první derivaci v každém bodě, což zvyšuje její nepřesnost. Na druhou stranu je velmi přímočará (doslova) a výpočetně nenáročná.

2.6.2 Metoda Runge-Kutta

Tato metoda vychází z Eulerovy metody, ovšem s tím rozdílem, že se snaží aplikovat určité korekce, které do jisté míry napravují to, že v každém bodě známe pouze první derivaci funkce. Korekce jsou založeny na tom, že nevychází pouze z derivace v jednom bodě, ale vychází z derivací ve více bodech.

Ve skutečnosti se nejedná o jednu konkrétní metodu, ale o celou skupinu metod, který mají stejný základ. Já budu používat obvyklou variantu pojmenovanou RK4, která funguje následujícím způsobem. [8]

Mějme funkci f, která nám vrací derivaci y v čase t a za nějaké hodnoty y.

$$\frac{dy}{dt} = f(t, y)$$

$$y(t_0) = y_0$$

Potom každou následující hodnotu y s krokem velikosti h vypočteme následovně:

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + \frac{h}{2}, y_n + k_1\frac{h}{2})$$

$$k_3 = f(t_n + \frac{h}{2}, y_n + k_2\frac{h}{2})$$

$$k_4 = f(t_n + h, y_n + hk_3)$$

Hodnoty k_1 až k_4 jsou derivacemi v různých bodech. Tyto body jsou časově vzdálené h nebo $\frac{h}{2}$, a hodnotu y odhadují pomocí Eulerovy metody, ovšem využívají různé hodnoty derivace.

k_1 je tedy derivace v bodě, který byl získán obyčenou eulerovou metodou. k_2 již používá poloviční krok a pro lineární extrapolaci nepoužívá derivaci v bodě y_n, ale hodnotu derivace v bodě získaným v předchozím kroku, tedy hodnotu k_1. k_3 používá opět poloviční krok a používá derivaci k_2. A konečně k_4 používá celý krok a lineárně extrapoluje pomocí k_3.

Výsledek se spočte vlastně stejně, jako u Eulerovy metody. Pro extrapolaci se ovšem nevyužije pouze derivace v bodě y, ovšem vážený průměr hodnot k_1,k_2, k_3 a k_4.

Tato metoda poskytuje se stejnou velikostí kroků h přesnější odhady, ovšem oproti Eulerově metodě, kde každý krok znamená jeden výpočet funkce f a jednu lineární extrapolaci, v případě RK4 je potřeba pro každý krok provést čtyři výpočty funkce f, čtyři lineární extrapolace a jeden vážený průměr.

2.7 Hardware

2.8 A/D převodníky

A/D převodník je zařízení, které převádí analogovou úroveň napětí na digitální hodnotu.[9] Mezi jejich základní parametry patří bitová hloubka, která definuje rozlišení každého jednotlivého naměřeného vzorku a maximální vzorkovací frekvenci. Často mají více vstupů, ze kterých vybírají analogovým multiplexorem. Díky tomu mohou převádět více vstupů, na úkor vzorkovací frekvenci.

Měření jednoho vzorku nějaký čas trvá. Po dobu měření vzorku se však typicky vstup nesmí měnit, proto je součástí převodníku tzv. „sample and hold“ obvod, který si analogově (např. nabitím kondenzátoru) „zapamatuje“ aktuální hodnotu vstupu, a tu následně drží konstantní po celou dobu měření. Celý cyklus převodu se tedy skládá ze dvou částí. Nejdříve si „sample and

hold“ obvod musí zapamatovat aktuální úroveň vstupu a následně proběhne samotné měření.

Převodník má typicky schopnost vyvolat na konci měření přerušení, takže si výpočetní jednotka může výsledek rovnou převzít.

Pro převod je nutné znát nejmenší a nejvyšší napětí, mezi kterými se bude napětí vstupu pohybovat. Tato dvě napětí určují, na jaká digitální čísla se budou vstupní napětí mapovat. Říkáme jim proto „referenční napětí“ a jsou dalšími dvěma vstupy převodníku. Máme-li např. převodník s 10bit rozlišením, bude nižší referenční napětí odpovídat hodnotě 0, zatímco vyšší referenční napětí bude odpovídat hodnotě $2^{10} - 1$.

Převodník může s výpočetní jednotkou komunikovat několika způsoby. Buď může být připojen pomocí sběrnice (UART, I2C atd.), nebo může být do výpočetní jednotky přímo integrovaný a být ovládaný pomocí speciálních registrů. Některé modely mohou pomocí DMA (direct memory access) přímo ukládat naměřené hodnoty do paměti, odkud si je výpočetní jednotka bude brát ve větších dávkách.

Převodníky mohou používat vnější nebo vnitřní hodiny.

2.9 D/A převodníky

D/A převodník je opak A/D převodníku. Převádí digitální hodnoty a analogový výstup. Stejně jako u A/D převodníku, mezi jeho základní vlastnosti patří bitová hloubka a maximální vzorkovací frekvence. Jeho obsluha je však typicky méně náročná než u A/D převodníků.

Některé převodníky obsahují FIFO frontu, do které lze vzorky vkládat. Převodník poté podle pravidelných hodin postupně hodnoty odebírá a promítá je na analogový výstup. Díky tomu lze digitální signál posílat na výstup hromadně v dávkách několika vzorků a o správné rozprostření hodnot v čase se už postará převodník sám. Jiné převodníky frontu mít nemusí a tak je potřeba nastavovat každou hodnotu zvlášť.

Kapitola 3

Návrh a Realizace

3.1 Implementace

Zařízení lze implementovat několika různými způsoby, ale základní blokové schéma bude vždy prakticky stejné.

Na obrázku je vyobrazen pouze jeden vstupní kanál. Protože je ale reproduktorová soustava stereofonní, je potřeba mít vstupy dva a výstupů šest. Toho můžeme teoreticky dosáhnout dvěma způsoby. Buď bude každá reprobedna samostatný systém postavený tak, jak je naznačeno na obrázku, a každá reprobedna tedy bude mít svůj vlastní rozbočovač. Nebo bude veškerá výpočetní technika jen v jedné reprobedně a tím pádem je potřeba, aby filtr měl skutečně dva vstupy a šest výstupů. Dva vstupy lze zajistit dvěma způsoby. Buď zde budou skutečně dva D/A převodníky, nebo pouze jeden s dvojnásobnou vzorkovací frekvencí, který bude multiplexorem přepínat mezi jedním a druhým kanálem.

Vzhledem k tomu, že pokud by veškeré filtrování probíhalo v jedné z reprobeden, musely by se do druhé bedny přivést všechny tři kanály každý zvlášť. To je samozřejmě možné, ale považuji to za poněkud nepraktické. Mnohem praktičtější se mi zdá možnost, kdy je každá bedna samostatnou jednotkou. Celý problém stereofonní třípásmové soustavy tedy zjednoduším na dvě in-

Obrázek 3.1: Blokové schéma zařízení

stance monofonní třípásmové soustavy.

Jádrem celého systému bude řídící a výpočetní jednotka. Může se jednat o CPU, DSP čip, případně třeba i FPGA, nebo dokonce ASIC.

A/D převodník a D/A převodníky mohou být součástí čipu řídící a výpočetní jednotky, nebo mohou být připojeny externí sběrnicí.

Jak jsem již zmínil dříve, častým způsobem, jak se s audio signálem pracuje, je po skupinách vzorků. Obvykle se nezpracovává každý vzorek zvlášť. Znamená to totiž typicky zvýšenou režii, což snižuje efektivitu celého systému. Je-li zpracováváno více vzorků zároveň, znamená to ovšem zvýšenou latenci. V případě reproduktorové výhybky je latence kriticky důležitá. Zpracovávání zvuku po delších úsecích je obzvlášť výhodné, pokud výpočetní jednotka provádí i jiné činnosti. Zde je určitá režie pramenící z toho, že zařízení musí přepínat kontext. To zde ovšem není náš případ. Dalším faktorem často bývá to, když je mezi fyzickými převodníky a kódem provádějícím výpočty více vrstev, jako např. v případě PC audio subsystém operačního systému. V našem případě ovšem na výpočetní jednotce žádný jiný kód neběží, tudíž je režie značně minimalizovaná. Díky tomu jsou hlavní důvody, proč se zvuk typicky zpracovává po delších úsecích v případě tohoto zařízení buď nevalidní, nebo minimalizované. Proto jsem se rozhodl, že bude zařízení zpracovávat každý vzorek zvlášť.

Komunikace s D/A převodníkem je poměrně přímočará. Z řídící jednotky do D/A převodníků jsou posílány spočtené výsledky a převodník je převádí na analogový výstup. Komunikace s A/D převodníkem je již složitější. Řídící jednotka do něj odešle instrukci k tomu, že má začít měření. Po dokončení měření dostane od převodníku přerušení. Řídící jednotka si v přerušení vyzvedne naměřenou hodnotu a poté přikáže převodníku začít následující měření. V mezičase provede potřebné výpočty nad právě obdrženým vzorkem a vyfiltrované vzorky pošle do výstupních D/A převodníků. Následně se výpočetní jednotka uspí do doby, než dostane další přerušení od A/D převodníku.

Je kriticky důležité, aby výpočet digitálního filtru a následné odeslání výsledků na výstupní převodníky proběhl rychleji, než je perioda vstupního A/D převodníku. V opačném případě by přerušení od A/D převodníku přišlo dřív, než došlo k dokončení výpočtu předchozího vzorku, a celé zařízení by tedy nestíhalo signál filtrovat. Takový případ by se musel řešit buď výkonnější výpočetní jednotkou, nebo snížením vzorkovací frekvence. První řešení zvyšuje cenu výsledného zařízení, druhé potenciálně snižuje kvalitu zvuku. Snížení vzorkovací frekvence lze dosáhnout snadno například tak, že měření dalšího vzorku nezačneme okamžitě po obdržení poslední hodnoty, ale nastartujeme A/D převod až v průběhu výpočtů. Protože počítáme tři různé výstupní hodnoty a každou z nich počítáme zvlášť, můžeme příkaz k měření dalšího vzorku umístit mezi první a druhý, nebo druhý a třetí výpočet. V horším případě můžeme příkaz vtěsnat až někam doprostřed posledního výpočtu. V nejhorším případě až úplně na konec.

Je nespornou výhodou, že nehledě na to, jakou hodnotu vzorek má, s ním

budou vždy prováděny stejné operace, které budou trvat vždy stejně dlouho. Nemůže se stát, že by najednou zpracovávání jednoho vzorku trvalo mnohem déle, než ostatní. To může u takto jednoduchého zařízení znít jako samozřejmost, ale u jiných zařízení tuto záruku často mít nemusíme, nebo ji budeme vynucovat za cenu značných komplikací v návrhu takového zařízení. Tato vlastnost je důležitá, protože nám zaručuje, že pokud zařízení navrhneme tak, aby výpočty stíhalo, máme jistotu, že je bude stíhat vždy. U systémů reálného času, jako je tento, bohužel amortizovaná složitost algoritmů nestačí. Mnohem důležitější, než amortizovaná složitost, je totiž složitost maximální. U všech tří výpočetních metod, které uvažujeme, je však amortizovaná i maximální složitost totožná.

3.1.1 Nepravidelné vzorkování

Jakým způsobem do výhybky implementovat nepravidelné vzorkování? Tuto otázku lze rozdělit na dvě podotázky. Jak nepravidelné vzorkování implementovat v kontextu řízení běhu celého zařízení, a jakým způsobem jej zohlednit v rámci samotného výpočtu.

Co se výpočtu týče, záleží na tom, kterou výpočetní metodu zvolíme. U rekurzivního filtru řešení není úplně přímočaré. Je popsáno v tomto [10] vědeckém článku.

U Eulerovy a Runge-Kutta metody je ovšem celá implementace mnohem přímočařejší. Jednou z konstant v těchto výpočtech je totiž tzv. delta, neboli čas, který uběhl od posledního vzorku. Všechny ostatní konstanty zůstávají stejné nehledě na to, jak dlouhý tento čas byl. Stačí nám zde tedy tuto hodnotu měnit pro každý výpočet podle toho, jaká prodleva zrovna nastala.

To nás přivádí k tomu, jak zajistit, aby vzorkování bylo nepravidelné, a jak pro každý vzorek zjišťovat časový údaj delta. Zde nám poněkud zjednodušuje návrh to, že každý vzorek zpracováváme zvlášť. Jediné, co je potřeba udělat, je odložit čas, ve kterém se započne měření následujícího vstupního vzorku. Toho dosáhneme pomocí časovače, který je součástí spousty mikroprocesorů. Časovač si lze představit jako takový „budík“, který lze nařídit na určitý čas, a který po uběhnutí tohoto času vyvolá přerušení.

Celý proces tedy bude vypadat tak, že jakmile přijde přerušení od vstupního A/D převodníku, výpočetní jednotka si přečte naměřenou hodnotu, nastaví časovač a začne výpočty. Někdy během výpočtů přijde přerušení od časovače. V tomto přerušení výpočetní jednotka zašle A/D převodníku instrukci, že má začít měřit následující vzorek, a poté bude pokračovat ve výpočtu. Nakonec výsledky odešle do výstupních D/A převodníků a uspí se do doby, než opět přijde přerušení od vstupního A/D převodníku.

Všimněme si, že nyní máme snadnou kontrolu nad vzorkovací periodou, kterou můžeme snadno měnit tím, že změníme čas, který budeme nastavovat v časovači. K nepravidelnému vzorkování nám už chybí jen nastavovat tuto dobu na hodnotu získanou z generátoru pseudonáhodných čísel. Protože nestojíme

o kryptografickou úroveň generování pseudonáhodných čísel, postačí pro tento účel bohatě lineární kongurentní generátor.

Známe tedy zpoždění, které jsme nastavili na časovači. Čas, který uběhne mezi spuštění měření v A/D převodníku a přerušením ohlašujícím naměřený výsledek známe z dokumentace tohoto převodníku. Dobu strávenou v obsluhách přerušení můžeme odvodit od délky tohoto kódu a taktovací frekvence výpočetní jednotky. Případně lze tyto hodnoty naměřit osciloskopem. První prodlevu si tedy program určuje sám a zbylé dvě jsou konstantní. Jejich součtem získáme časový údaj delta, který použijeme ve výpočtech.

Rozmezí, ve kterém se budou pohybovat hodnoty nastavené na časovači, je dalším z podstatných parametrů, který ovlivňuje zvukovou kvalitu. Příliš nízké hodnoty by nemusely mít požadovaný efekt potlačení aliasingu. Čím jsou ovšem hodnoty vyšší, tím více se degraduje vzorkovací frekvence.

3.1.2 Výpočet

Ať už bude mít výpočetní jednotka jakoukoli formu, bude provádět výpočty, kterými bude převádět jeden vstupní signál na tři různé výstupní signály. Každý výsledkem filtrování skrz jiný filtr. Výpočet těchto tří signálů bude prakticky totožný, pouze přenosové funkce filtrů se od sebe budou lišit.

Na úplném počátku tedy budou tři přenosové funkce. Jedna pro každý výstup. Její přesná podoba bude odvozena od specifikací konkrétních reproduků v soustavě. Pokud budeme výpočet provádět jako rekurzivní filtr, potřebujeme přenosovou funkci v z-doméně. Pokud budeme počítat diferenciální rovnice za pomocí Eulerovy nebo Runge-Kutta metody, potřebujeme přenosovou funkci v s-doméně. Pro případný převod mezi těmito dvěma doménami použijeme Bilineární transformaci.

Pro rekurzivní filtr odvodíme konstanty přímo z přenosové funkce v z-doméně, jak jsem již popsal dříve. Pro Eulerovu a Runge-Kutta metodu musíme nejdříve převést přenosovou funkci v s-doméně na diferenciální rovnici. Tu poté pomocí metody postupné integrace převedeme do požadované formy, kterou lze již přímo použít v těchto metodách. Oba tyto postupy jsem popsal dříve.

Převod přenosů filtrů druhého řádu na soustavu obyčejných diferenciálních rovnic prvního řádu jsem si pomocí metody postupné integrace předpočítal obecně. Přenos ve tvaru

$$\frac{A + Bs + Cs^2}{D + Es + Fs^2}$$

odpovídá soustavě

$$y = \frac{Cx + w_2}{F}$$

$$w_2' = Bx - Ey + w_1$$

$$w_1' = Ax - Dy$$

Nyní již známe všechno, co je nutné pro to, abychom mohli vložit požadovanou výpočetní metodu do řídícího software.

3.2 Srovnání metod

3.3 Srovnání výpočetních metod

3.3.1 Výpočetní náročnost

V této kapitole budu porovnávat výpočetní náročnost jednotlivých metod. Uvažuji zde implementaci filtru druhého řádu. Rekurzivní filtry vyšších řádů bývají často nestabilní a je lepší takový filtr složit jako kaskádu filtrů maximálně druhého řádu. [3, str. 339]

3.3.1.1 Rekurzivní filtr

Rekurzivní filtry jsou výpočetně zdaleka nejrychlejší. Rekurzivní filtr druhého řádu provádí následující výpočet:

$$y[n] = a_0 \cdot x[n] + a_1 \cdot x[n-1] + a_2 \cdot x[n-2] + b_1 \cdot y[n-1] + b_2 \cdot y[n-2]$$

Pro výpočet jednoho vzorku je tedy potřeba 5 násobení a 4 sčítání. K tomu lze ještě připočíst 5 přesunů historických záznamů $x[n-1], x[n-2], y[n-1], y[n-2]$ mezi registry.

Alternativně některé architektury (obzvlášť DSP procesory) podporují tzv. „Multiply accumulate“ instrukci, někdy také zvanou „Multiply and add“, která provede násobení dvou hodnot a výsledek přičte do dalšího registru. Rekurzivní filtr druhého řádu lze tedy implementovat pouhými 5 instrukcemi Multiply Accumulate. V případě některých přenosů mohou být některé konstanty rekurzivního filtru nulové, takže to může být dokonce méně.

3.3.1.2 Eulerova metoda

Eulerova metoda nejdřív spočte hodnoty derivací jednotlivých stavových proměnných a poté stavové proměnné lineárně extrapoluje. V každém kroku tedy potřebujeme spočíst hodnoty derivací stavových proměnných:

$$w_2' = Bx - Ey + w_1$$

$$w_1' = Ax - Dy$$

Zde je 5 násobení a 3 sčítání. Nebo 5 instrukcí Multiply accumulate. Následně je každá z takto spočtených derivací vynásobena časem, který uběhl od posledního vzorku, tedy hodnotou delta. To jsou dvě násobení. Dále je

potřeba přičíst k aktuálním hodnotám w_1 a w_2, což jsou dvě sčítání. Nakonec je potřeba spočíst výstupní hodnotu:

$$y = \frac{Cx + w_2}{F}$$

Což jsou (pokud dělení F změníme na násobení inverzní hodnotou) dvě násobení a jedno sčítání. Případně jedno násobení a jednu instrukci Multiply accumulate.

Dohromady je tedy na spočtení jednoho vzorku 9 násobení a 6 sčítání. To je téměř dvojnásobný počet násobení oproti rekurzivnímu filtru.

3.3.1.3 Runge Kutta

Když se podíváme na tuto metodu, zjistíme, že se v ní skrývají dohromady čtyři výpočty Eulerovou metodou. K tomu se zde počítají dva vážené průměry, každý se počítá ze čtyř hodnot, což znamená 6 násobení a 6 sčítání. Dohromady dostáváme 42 násobení a 30 sčítání. To je daleko více než u rekurzivního filtru - více než 8x větší počet násobení a více než 7x větší počet sčítání. Oproti Eulerově mětodě je tato metoda více než 4x pomalejší. Navíc tato metoda pracuje při výpočtu s více hodnotami, takže se při výpočtu bude muset více hodnot ukládat dočasně z registrů do operační paměti, což celý výpočet ještě zpomalí. Je tedy možné, že rychlostní rozdíl mezi touto metodou a rekurzivním filtrem bude ještě vyšší.

3.3.2 Přesnost výsledků

V této části se budu snažit porovnat výsledky, které jednotlivé výpočetní metody produkují. V jazyce C++ jsem všechny tři algoritmy naimplementoval. Za použití postupů popsaných dříve jsem převedl přenosovou funkci filtru na hodnoty konfiguračních parametrů. Poté jsem změřil frekvenční odezvu.

Frekvenční odezvu jsem měřil dvěma způsoby. První způsob do filtru posílal sinusoidy o různých frekvencích a měřil amplitudu výsledného signálu. Druhá metoda poslala do filtru jednotkový impulz a nad takto získanou impulzní odezvou provedla algoritmus Fast Fourier Transform. Obě metody dávají prakticky totožné výsledky. Jako směrodatné ale budu brát výsledky z Fourierovy transformace. Pro několik frekvencí jsem si z přenosové funkce spočetl, jaký útlum by daný filtr měl správně mít. Tyto hodnoty poté poměřuji s reálnou frekvenční odezvou, kterou jsem u filtrů naměřil.

Každý filtr zkouším na vzorkovacích frekvencích 44.1kHz, 96kHz a 192kHz. Dále tyto výpočty testuji s jednoduchou i dvojitou přesností čísel s pohyblivou čárkou. Zde v textu uvedu pouze grafy. Přesné hodnoty jsou ovšem v příloze této práce. Stejně jako C++ implementace, pomocí které byly pořízeny.

První filtr má následující přenosovou funkci:

$$\frac{1}{(0.00031831p + 1)^2}$$

Z této přenosové funkce jsem odvodil následující konstanty do soustavy diferenciálních rovnic: $A = 1.0, B = 0.0, C = 0.0, D = 1.0, E = 0.00063662, F = 1.01321 \cdot 10^{-}7$

Pro rekurzivní filtr jsem odvodil následující konstanty: Pro vzorkovací frekvenci 44.1kHz $a_0 = 1.0$, $a_1 = 0.03439399999999997$, $a_2 = 0.03439399999999997$, $b_1 = 0.931212$, $b_2 = 0.0$,

Pro 96kHz $a_0 = 0.016098999999999974$,$a_1 = 0.016098999999999974$, $a_2 = 0.0$,,$b_1 = 0.967802$,$b_2 = 0.0$,

Pro 192kHz $a_0 = 0.00811499999999998$, $a_1 = 0.00811499999999998$, $a_2 = 0.0$, $b_1 = 0.98377$, $b_2 = 0.0$,

Druhý rekurzivní filtr má následující přenosovou funkci:

$$\frac{0.0253303p^2}{(0.159155p + 4175.)^2}$$

Z této přenosové funkce jsem odvodil následující konstanty do soustavy diferenciálních rovnic: $A = 0.0, B = 0.0, C = 0.0253303, D = 1.74306 \cdot 10^7, E = 1328.94, F = 0.0253303$

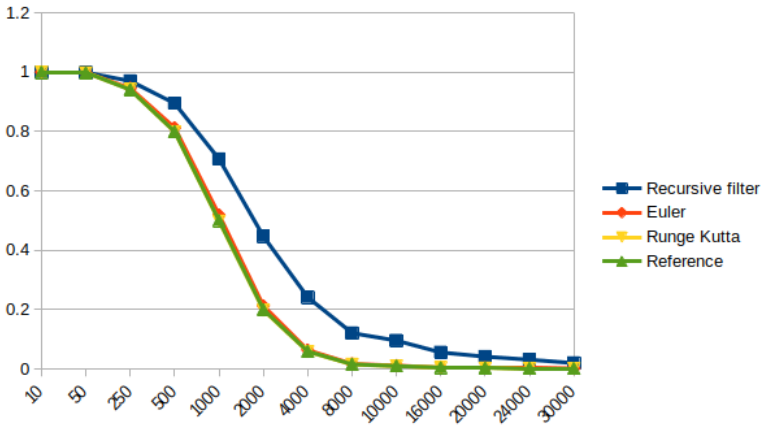
Pro rekurzivní filtr jsem odvodil následující konstanty: Pro vzorkovací frekvenci 44.1kHz $a_0 = 0.770761820868862$, $a_1 = -0.770761820868862$, $a_2 = 0.0$), $b_1 = 0.541523$, $b_2 = 0.0$,

Pro 96kHz $a_0 = 0.8797965772902191$,$a_1 = -0.8797965772902191$, $a_2 = 0.0$,,$b_1 = 0.759593$,$b_2 = 0.0$,

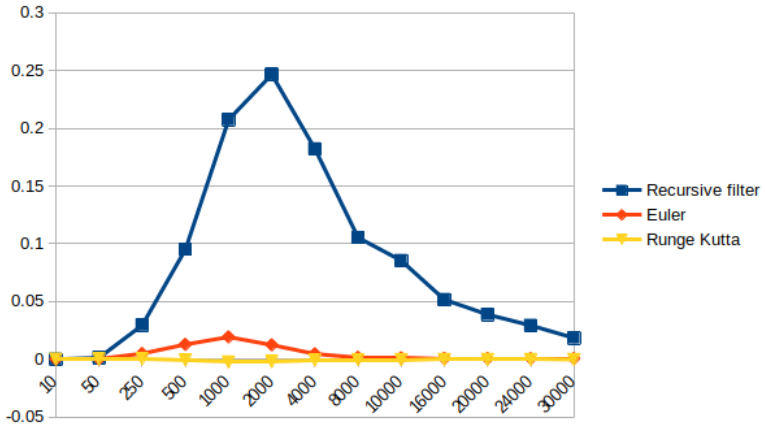
Pro 192kHz $a_0 = 0.9360550205581661$, $a_1 = -0.9360550205581661$, $a_2 = 0.0$, $b_1 = 0.87211$, $b_2 = 0.0$,

Hodnotu frekvenční odezvy jsem si v několika bodech nechal vypsat a porovnal tyto hodnoty mezi sebou. Dále je v tomto grafu vynesena referenční hodnota, kterou jsem spočítal z přenosové funkce. V druhém a třetím grafu jsou vidět odchylky od referenčních hodnot. V příloze lze nalézt celé průběhy frekvenční odezvy, které vytváří moje C++ implementace.

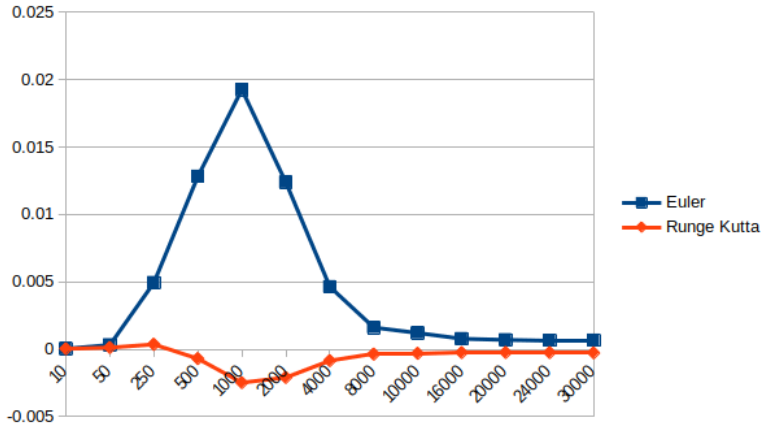
Z grafů je zřejmé, že se rekurzivní filtr odchyluje nejvíce. Tato odchylka se zdá být prakticky konstantní a nezávislá na vzorkovací frekvenci. Oproti tomu Eulerova a Runge Kutta metoda s vyššími vzorkovacími frekvencemi zlepšují. V přílohách jsou k dispozici i výsledky výpočtů s jednoduchou přesností. Výsledky se ovšem příliš neliší a tak je zde neuvádím.



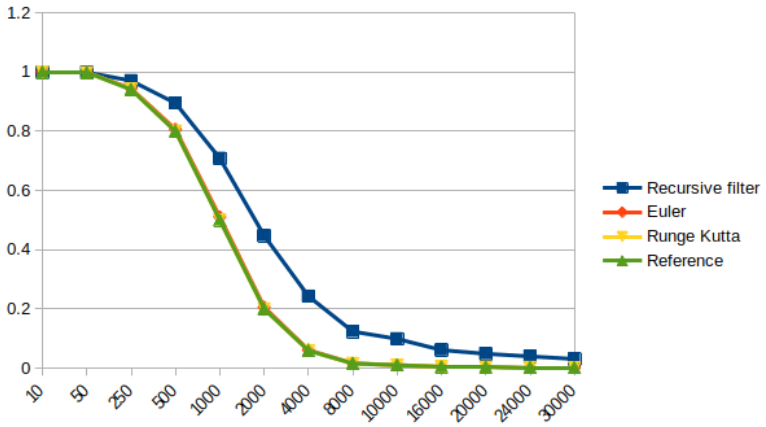
Obrázek 3.2: Srovnání prvního filtru, 44.1kHz



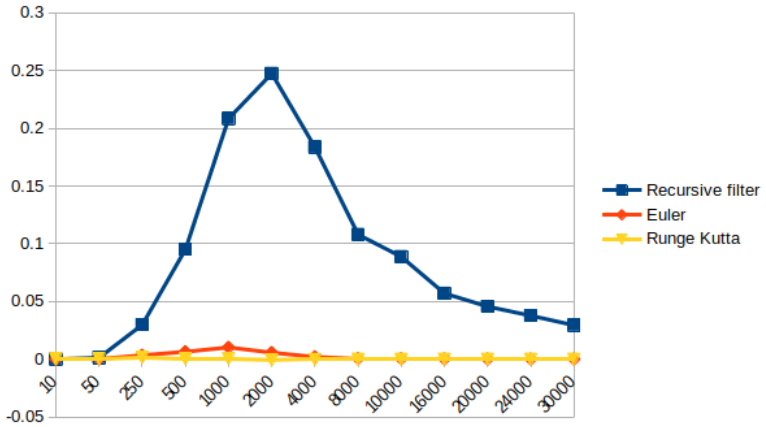
Obrázek 3.3: Odchylka od reference prvního filtru, 44.1kHz



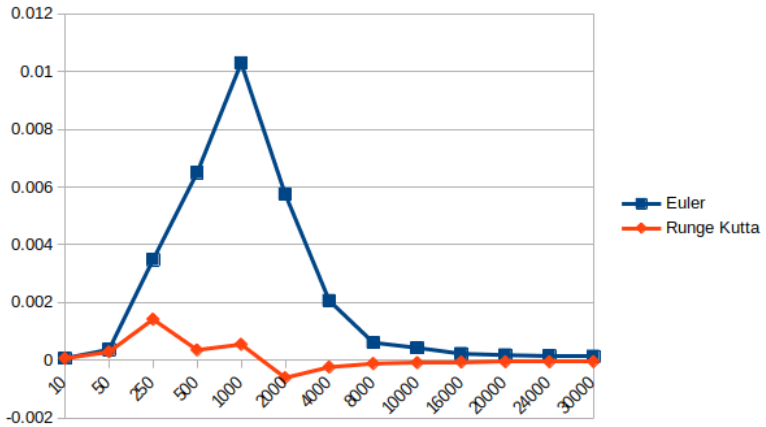
Obrázek 3.4: Odchylka od reference prvního filtru Eulera a Runge Kutta, 44.1kHz



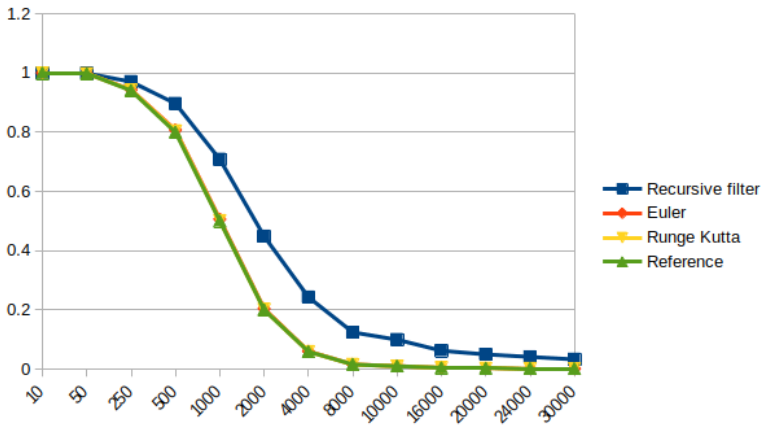
Obrázek 3.5: Srovnání prvního filtru, 96kHz



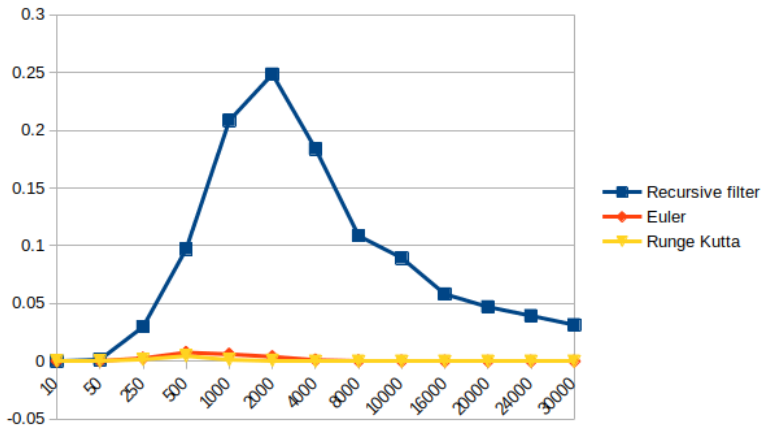
Obrázek 3.6: Odchylka od reference prvního filtru, 96kHz



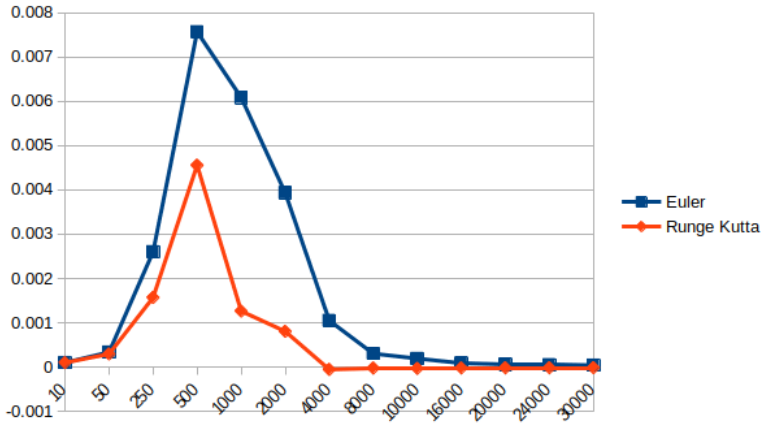
Obrázek 3.7: Odchylka od reference prvního filtru Eulera a Runge Kutta, 96kHz



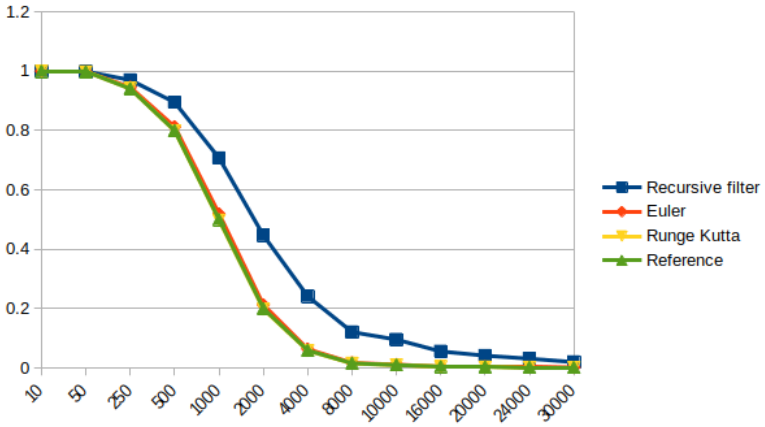
Obrázek 3.8: Srovnání prvního filtru, 192kHz



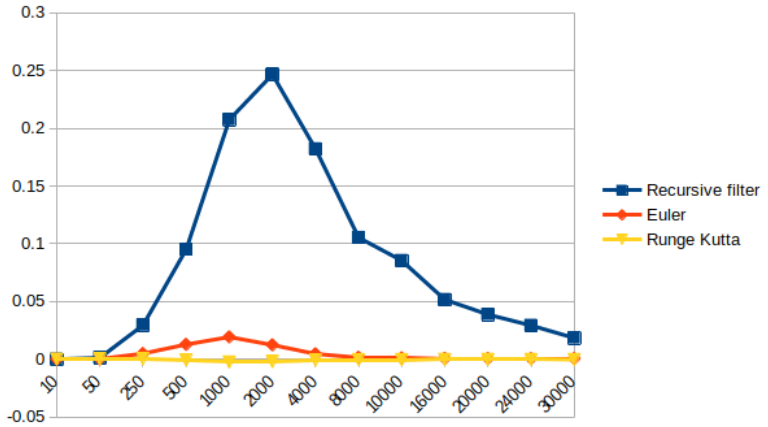
Obrázek 3.9: Odchylka od reference prvního filtru, 192kHz



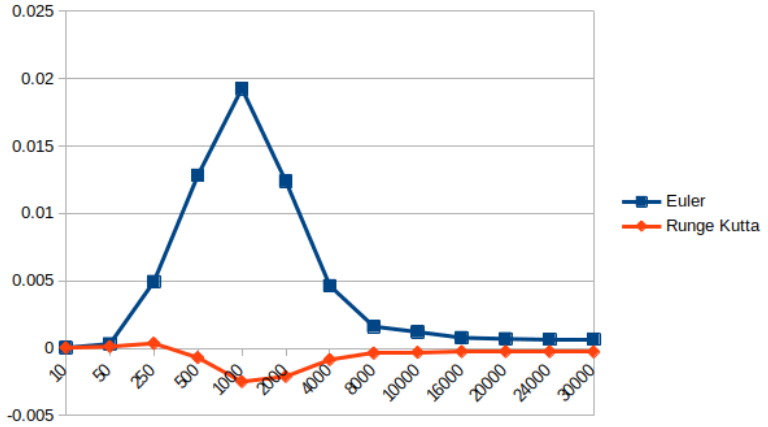
Obrázek 3.10: Odchylka od reference prvního filtru Eulera a Runge Kutta, 192kHz



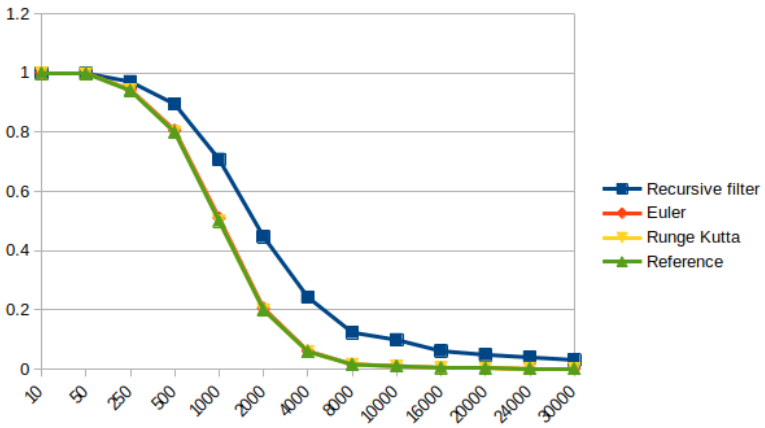
Obrázek 3.11: Srovnání druhého filtru, 44.1kHz



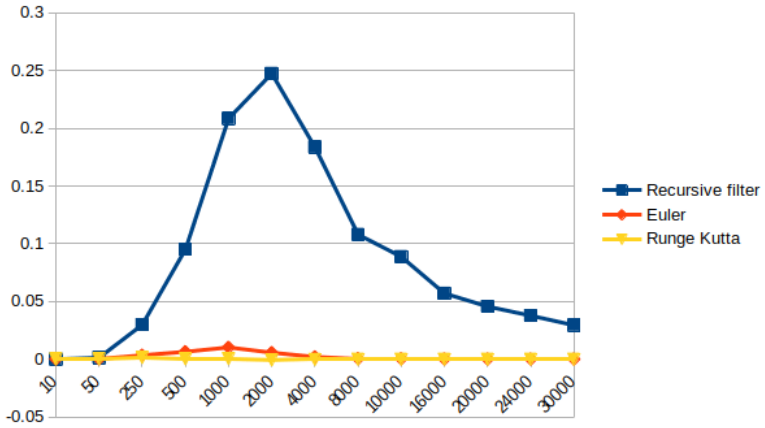
Obrázek 3.12: Odchylka od reference druhého filtru, 44.1kHz



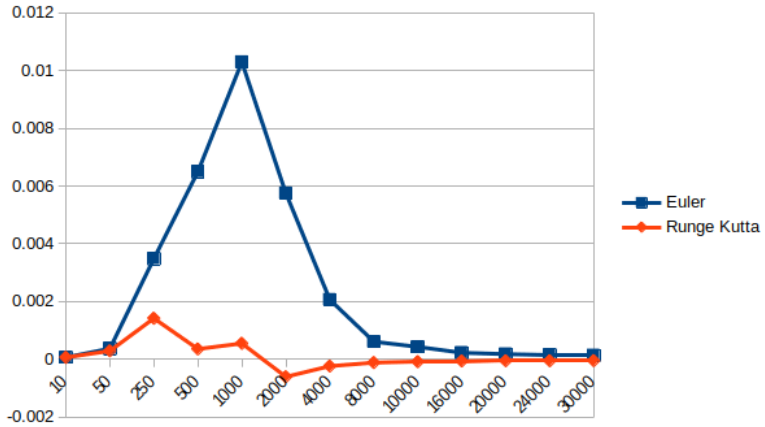
Obrázek 3.13: Odchylka od reference druhého filtru Eulera a Runge Kutta, 44.1kHz



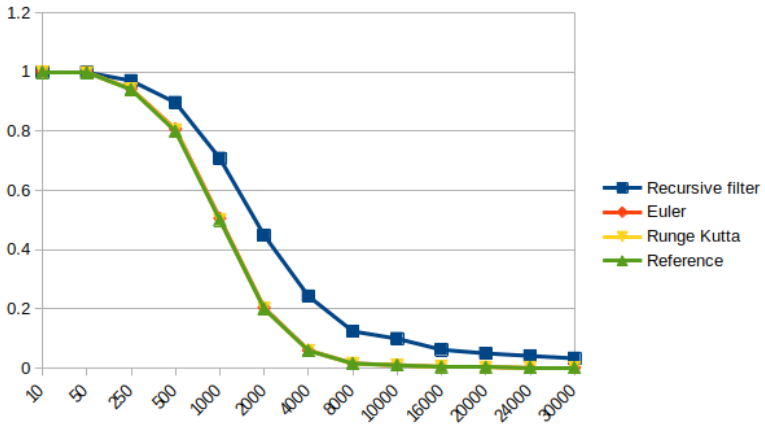
Obrázek 3.14: Srovnání druhého filtru, 96kHz



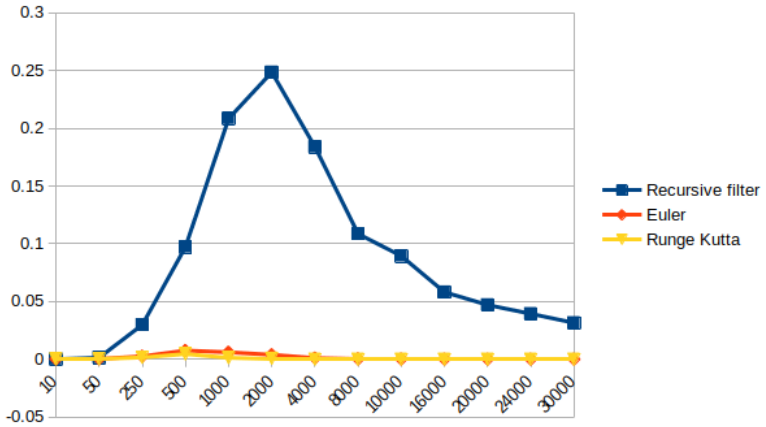
Obrázek 3.15: Odchylka od reference druhého filtru, 96kHz



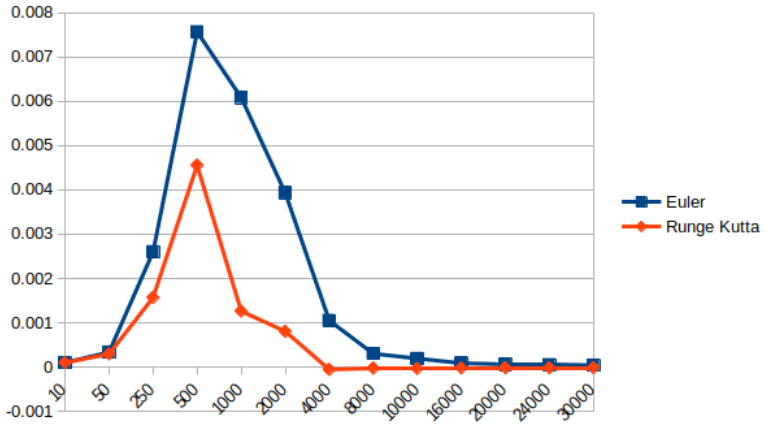
Obrázek 3.16: Odchylka od reference druhého filtru Eulera a Runge Kutta, 96kHz



Obrázek 3.17: Srovnání druhého filtru, 192kHz



Obrázek 3.18: Odchylka od reference druhého filtru, 192kHz



Obrázek 3.19: Odchylka od reference druhého filtru Eulera a Runge Kutta, 192kHz

Závěr

V této práci jsem prozkoumal a popsal základní teorii lineárních systémů, popsal jsem Laplaceovu transformaci a přenosovou funkci. Dále jsem popsal, jak z přenosové funkce vytvořit soustavu diferenciálních rovnic prvního řádu a popsal metody, jak tyto rovnice numericky řešit. Vysvětlil jsem fungování filtrů a popsal jejich základní charakteristiky a způsoby implementace. Vysvětlil jsem, jak souvisí Laplaceova transformace s Z-transformací a uvedl jsem, jak s její pomocí lze odvodit parametry pro rekurzivní filtr. Navrhl jsem podobu celého zařízení a popsal jeho komponenty. Nakonec jsem srovnal rekurzivní filtr s numerickým počítáním diferenciálních rovnic pomocí Eulerovy metody a podle metody Runge Kutta. Porovnal jsem jak výpočetní náročnost, tak přesnost výsledků těchto metod.

Zjistil jsem, že metoda Runge Kutta je nejpřesnější. Eulerova metoda je zhruba o řád méně přesná. Rekurzivní filtr je poté o další řád méně přesný. Runge Kutta a Eulerova metoda má navíc tendenci se zpřesňovat, zvyšuje-li se vzorkovací frekvence. Tento efekt se ovšem zdá se od určité frekvence může naopak otáčet zpět. Rekurzivní filtr se ovšem zdá mít nepřesnosti téměř konstantní nehledě na vzorkovací frekvenci.

Eulerova metoda je ovšem zhruba dvakrát pomalejší, než rekurzivní filtr. Dále Runge Kutta bude více, než osmkrát pomalejší, než rekurzivní filtr. Zdá se tedy, že by Eulerova metoda mohla být vhodným kompromisem pro takovéto zařízení.

Celé zařízení je navrženo tak, že může podporovat nepravidelné vzorkování jako prevenci proti aliasingu. Osobně si ovšem myslím, že je lepší zajistit dostatečně vysokou vzorkovací frekvenci na to, aby frekvence způsobující aliasing byly daleko za hranicí slyšitelnosti. O jejich odfiltrování už by se poté postaral jednoduchý analogový filtr zařazený ped vstup do zařízení. Protože by ovlivňoval pouze frekvence daleko za hranicí slyšitelnosti, byly by nároky na jeho přesnost jen velmi malé.

Literatura

[1] Sound of sound, Optimising The Latency Of Your PC Audio Interface. https://www.soundonsound.com/techniques/optimising-latency-pc-audio-interface#7, k datu 4.5.2022.

[2] Leis, J. W.: *Digital Signal Processing Using MATLAB for Students and Researchers*. John Wiley and Sons., ISBN 9781118033807.

[3] Smith, S. W.: *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997. Dostupné z: http://www.dspguide.com

[4] x-engineer.org Transfer function algebra. https://x-engineer.org/transfer-function-algebra/, k datu 3.5.2022.

[5] Physical Audio Signal Processing Bilinear transformation. https://www.dsprelated.com/freebooks/pasp/Bilinear_Transformation.html, k datu 3.5.2022.

[6] swarthmore Transformation: Single Diff Eq to Transfer Function. https://lpsa.swarthmore.edu/Representations/SysRepTransformations/TF2SDE.html, k datu 3.5.2022.

[7] Hrubý, M.: Modelování a simulace - spojité modelování Slajdy pro předmět ISM, Vysoké učení technické v Brně. https://www.soundonsound.com/techniques/optimising-latency-pc-audio-interface#7, slide 13.

[8] Iserles, A.: *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, ISBN 978-0-521-55655-2.

[9] Microchip 10-bit A/D converter. https://ww1.microchip.com/downloads/en/DeviceDoc/31023a.pdf, k datu 3.5.2022.

[10] Gastal, E. S. L.; Oliveira, M. M.: High-Order Recursive Filtering of Non-Uniformly Sampled Signals for Image and Video Processing. *Computer Graphics Forum*, ročník 34, č. 2, May 2015: s. 81–93, proceedings of Eurographics 2015.

Příloha **A**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `siglab` ..... adresář se spustitelnou formou implementace
  - `Makefile` ..... Makefile, pomocí kterého lze implementaci přeložit
- `prenosyDruhyRad` ..... soubor s přenosy použitými v praktické části
- `PrepocetPrenosoveFunkce.nb` Mathematica notebook použitý k výpočtu konstant z přenosové funkce
- `VysledkyFiltru.ods` ..... Komplentí výsledky měření přesnosti
- `text` ..... zdrojová forma práce ve formátu LaTeX
- `thesis.pdf` ..... text práce ve formátu PDF