

Дигитално процесирање звук

Бранислав Геразов

Факултет за електротехника и информациски технологии
Универзитет „Св. Кирил и Методиј“ во Скопје, Македонија

Дигитално процесирање звук v1.00

4. март 2024, Скопје, Македонија

Институт за електроника

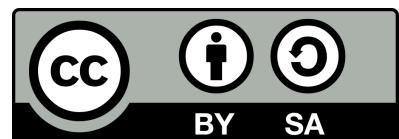
Факултет за електротехника и информациски технологии

Универзитет „Св. Кирил и Методиј“ во Скопје, Македонија

© Copyright by Branislav Gerazov 2016 -- 2024

This work is licensed under a Creative Commons Attribution-
ShareAlike 4.0 International.

<https://creativecommons.org/licenses/by-sa/4.0/>



Содржина

0 Вовед во дигитален звук	5
0.1 Предности на дигиталниот звук	5
1 Дигитализација на звукот	8
1.1 Земање примероци	8
1.2 Квантизација	14
1.3 Формати на дигитален звук	23
2 Вовед во процесирање звучни сигнали	25
2.1 Вчитување на звук во Python	25
2.2 Скратување на звукот и префрлање во моно	26
2.3 Прикажување звук	26
2.4 Преслушување на аудиосигналот	27
2.5 Менување на амплитудата на аудиосигналот	28
2.6 Нормализација на аудиосигналот	30
2.7 Генерирање на звук во Python	31
3 Фреквенциски спектар на звучните сигнали	33
3.1 Основи на Фуриеовата анализа	33
3.2 Анализа на спектарот на звучните сигнали	40
3.3 Фуриеова трансформација со прозорци STFT	44
4 Филтри	52
4.1 Основи на дигиталните филтри	52
4.2 Дизајн на FIR филтер	56
4.3 Филтрирање на аудиосигнал со FIR филтер	57
4.4 Дизајн на IIR филтри	59
4.5 Употреба на IIR филтри за еквализација	61
4.6 IIR филтри непропусни на фреквенција	63
5 Дигитални аудиоэффекти	64
5.1 Аудиоэффекти во временски домен	64
5.2 Аудиоэффекти во фреквенциски домен	68
6 Компресија на звукот	74
6.1 MPEG-1 ниво III	75
6.2 Компресија на говор	80
7 Синтеза на звук и говор	82
7.1 Линеарна предиктивна анализа	82
7.2 Анализа и синтеза на глас со линеарна предикција	83
7.3 Синтеза на говор	88

8 Препознавање на звук и говор	97
8.1 Основи на машинското учење	97
8.2 Препознавање на звучен извор	107
8.3 Автоматско препознавање на говор	113
Додаток А Слободен и отворен софтвер за инженерска и научна работа	125
A.1 Слободен софтвер	125
A.2 Четири слободи	125
A.3 Предности на слободниот софтвер	126
A.4 Одржливост	127
A.5 Слободен софтвер за инженерска и научна работа	128
Додаток Б Основи на користење на GNU/Linux	131
B.1 Инсталирање на GNU/Linux	131
B.2 Основи поставки во GNU/Linux	134
B.3 Основни на работа во командна линија	135
Додаток В Python за процесирање на звучни сигнали	142
B.1 Зошто Python?	142
B.2 Python интерпретер	144
B.3 Виртуелни средини за Python	144
B.4 Развојни средини за Python	147
B.5 Основи на Python	149
B.6 Контрола на текот на програмата	153
B.7 Генерирање на листи во еден ред	154
B.8 Функции	155
B.9 NumPy, SciPy и Matplotlib	156
B.10 Платформи за создавање на GUI во Python	158

Глава 0

Вовед во дигитален звук

Како што преодот кон обработка и зачувување на звукот во електричен домен донел своевидна револуција во квалитетот на снимениот звук, така преодот на звукот од електричен во [дигитален домен](#) уште повеќе го издигнал нивото на поимањето на звукот во техниката и секојдневието. Во извесна смисла станува збор за втора [револуција](#) и тоа: во начинот на зачувување на звукот, во неговата обработка и анализа, како и во неговата масовна дистрибуција.

Дигиталните алатки се речиси семоќни кога станува збор за обработка на звучните записи. Од засилување и слабеење на сигналот, што се сведува на просто множење на нумеричкиот запис на сигналот со одреден коефициент, до комплексни аудиоэффекти, синтеза на нови звучни облици, издвојување на мелодиска линија, отстранување на шумот итн. -- обработката на звукот е ограничена само од човековата визија за она што сака од звукот да го добие. Уште повеќе, ако се има предвид дека компјутерите и микроконтролерите се главните платформи за обработка на дигиталниот звук, може да кажеме дека опремата потребна за тоа е лесно достапна и масовно раширена, а не привилегија на професионалците.

0.1 Предности на дигиталниот звук

Дигиталниот звук во однос на аналогниот запис на звукот носи низа на придобивки. Во продолжение ќе се осврнеме на главните.

Поширок динамички и фреквенциски опсег

Дигиталниот звук кодиран со 16-битна резолуција теориски нуди однос сигнал шум од 96 dB, споредено со динамичкиот опсег од околу 80 dB кај најдобрите аналогни системи. Ова ниво е уште поголемо ако го зголемиме бројот на битови со кои го кодираме звукот. Ваков динамички опсег во реалноста не е навистина неопходен. Тој доаѓа во игра само во исклучителни случаи кога во рамките на едно музичко дело еден симфониски оркестар ја искористува целосната динамика која може да ја даде. На пример, тоа е случај кога имаме релативно тивки мелодии на флејта па потоа громогласни изливи на целиот оркестар. Тогаш, зголемениот динамички опсег станува потреба, бидејќи менувањето на силината на влезниот сигнал за време на снимањето не е дозволено.

Уште повеќе, иако во професионалните носачи на аналоген звук можел да биде запишан целиот слушен опсег на фреквенции, во оние за општа употреба, како на пр. аудиокасетите, фреквенцискиот опсег бил дефиниран со Hi-Fi¹ стандардот и изнесувал 16 kHz. Од друга страна, дури и кај носачите на дигитален звук за масовна потрошувачка како аудио-CD стандардот, фреквенцискиот опсег од 22,05 kHz целосно го покривал слушното подрачје на човекот.

¹анг. *high fidelity*

Зголемена отпорност на шум

Дигиталниот аудиозапис со својата еднозначност овозможува отпорност на загадувањето на звучниот сигнал со шум. Додека кај аналогните системи шумот предизвикан од електромагнетните пречки, како и термичкиот шум се акумулираат во звучниот сигнал долж неговото движење низ електронските кола; во дигиталните системи таква акумулација на шум нема. Професионалната дигитална аудиопрема работи со 24-битно кодирање на звукот, што соодветствува со однос сигнал-шум од 144 dB, а единствениот присутен шум е оној кој ќе влезе во аудиосистемот пред степенот за дигитализација.

Усовршено и олеснето умножување

Дигиталниот аудиозапис, повторно поради својата еднозначност, може да се пресними од еден дигитален носач на звук на друг, без било каква загуба во квалитетот. За споредба, кај аналогните записи секогаш имаме загуби во квалитетот на записот, па и додавање на нов шум при преснимувањето. Така, дури и кај најдобрите аналогни системи постои загуба од околу 3 dB во односот сигнал-шум при правењето на копија на некој запис. Проблемот станува сериозен кога имаме синцир на преснимувања копија од копија од копија, при што квалитетот на конечната снимка ќе биде намален за сумата на штетни влијанија внесени од секој од уредите искористени во синцирот. Со други зборови квалитетот на аналогната снимка е условен од должината на синцирот, како и од квалитетот на уредите кои учествуваат во него. Дигиталното преснимување е отпорно на обата овие параметри.

Уште повеќе, умножувањето на дигиталните аудиозаписи е олеснето поради зголемената брзина со која можат да се направат копиите. Кај аналогното преснимување, поради самата карактеристика на аналогните носачи на звук, речиси секогаш мора да се прави во реално време. Така, за преснимување на грамофонска плоча на аудиокасета и во најдобар случај мора да пројде околку време колку што трае самиот звучен запис. Истото важи и кај преснимувањето од аудиокасета на касета.² Што значи дека за преснимување 60-минутен запис скоро секогаш се потребни 60 минути за преснимување. Од друга страна, за да преснимиме 80-минутно аудио-CD на хард дискот на компјутерот ни требаат 5 минути, а за понатамошно умножување на записот во рамките на хард дискот по копија ни се потребни само 15 секунди!³

Зголемена трајност на дигиталниот запис

Оптичките носачи на звук како CD-ата се многу поотпорни на стареење од магнетните носачи на звук. Кај нив не постои саморазмагнетизација како кај магнетните ленти а непостоењето на физички контакт меѓу механизмот за читање и самиот диск ги оневозможува оштетувањата при преслушувањето на записот. Најголеми штетни влијанија врз долготрајноста на оптичките медиуми на запис се радијацијата, влажноста, како и температурните влијанија. Сепак, производителите на CD-R дискови рекламираат трајност од 75 години при соодветно складирање, наспроти максималните 30 години за магнетната лента. Кај златните и платинестите CD-а оваа граница оди и до 100, односно 200 години соодветно.

Уште повеќе, погодно е преку примената на ефикасни методи за кодирање и внесување на редунданса да се осигури интегритетот на дигиталниот аудиозапис. Повеќето дигитални носачи на звук како на пример CD-то и DAT касетите имаат вградени механизми за корекција на грешка. Со нивна помош, дури и ако површината на дискот физички е оштетена, читачот автоматски ќе ја употреби сета останата информација да ги реконструира изгубените податоци.

²Некои системи нудат двократно скратување на ова време со зголемена брзина на движење на магнетните ленти при преснимувањето, но ова може да биде по цена на намалување на квалитетот на направената копија.

³Точните вредности зависат од параметрите на компјутерскиот хардвер. Тука се дадени заокружени вредности за да се даде перспектива на нивниот сооднос.

Неограничени можности за обработка

Дигиталниот звук за првпат може визуелно да се претстави и директно да се работи со таа негова визуелизација. Во компјутерските системи звукот е зачуван како временска низа од амплитудни вредности која лесно може да се прикаже на екраните при обработката. Тоа овозможува бескрајна прецизност при сечењето и монтирањето на аудиоматеријалот. Уште повеќе како низа од податоци звукот е достапен за математичка анализа и обработка со помош на софтверски алатки, без потреба од софистициран хардвер.

Врз дигиталната форма на звукот може директно да се применат техниките за процесирање на сигнали, со чија помош тој може да се анализира и видоизмени. Тука пред сè спаѓаат филтрите кои можат полесно да се реализираат во дигитален домен, но и да се искористат за градба на комплексни структури составени од филтерски единици, наречени и банки на филтри. Понатаму, можеме да ги споменеме аудиоэффектите кои можат да се базираат на процесирање на аудиосигналите во временски или спектрален, односно трансформациски домен. Конечно, со помош на техниките на дигитално процесирање, аудиото може да се параметризира преку негово моделирање. Со ова може на пример да се одвои мелодијата од звучната боја, тие да се обработат и изменат, па повторно да се изврши ресинтеза на звучниот сигнал. Така истата мелодија може да биде отсвирена со боја на друг инструмент, или пак мелодијата може да се повиши или снижи и да се отсвири со оригиналната боја. Ова е нешто што се користи во дигиталните синтетизатори на звук, популарно наречени синтисајзери.

Напредни технологии

Освен техниките за обработка на аудиосигналите, нивниот дигитален запис овозможува примена на напредни технологии од пошироката област на вештачката интелигенција, поточно техниките за машинско учење. Овие технологии направија своевидна трета револуција во аудиосистемите, а и пошироко во многу други области. Со нивната примена се решени проблеми за кои долго време немало решение, или пак решенијата биле нездадоволителни. Уште повеќе, со нивна примена, некои проблеми се решени на ниво подобро од тоа што го може човекот, т.е. имаат надчовечки перформанси. Таква е на пример областа на препознавање на говорот, во која овие методи имаат подобра точност во препознавање на говорот од таа на човекот. Конечно, овие технологии направија исчекор напред кон нови можности во обработката и синтезата на звучните сигналите, кои претходно беа незамисливи, на пр. автоматско создавање на музика.⁴

Уште еден пример за ова е синтезата на нови звучни форми. Имено, иако во аналоген домен постојат синтетизатори на звучни форми, нивните можности во поглед на бојата на добиениот звук се ограничени. Во дигитален домен пак, освен можноста да се употреби произволна боја при синтезата, постојат напредни алгоритми за спектро-темпорално обликување базирани на длабоки невронски мрежи кои можат да произведат звуци кои се половина виолина, половина удар на гром или мјаукање на мачка.⁵

Освен во синтезата на нови звучни форми, техниките на машинско учење се употребуваат за: препознавање на звучен извор, издвојување на звучен извор, препознавање на жанр, препознавање на говор, говорник и јазик, синтеза на говор, симултан превод од јазик во јазик итн. Во синтезата на говор на пример, напредните алгоритми можат да синтетизираат говор кој не може да се разликува од природниот.⁶ Во спој со техниките за препознавање на говор и процесирање на природни јазици ова им овозможува на виртуелните интелигентни асистенти да комуницираат непречено со лубето во ограничени контексти.⁷

⁴OpenAI MuseNet <https://openai.com/blog/musenet/>

⁵Повеќе за ова може да прочитате на сајтот на Магента <https://magenta.tensorflow.org/nsynth-instrument>, а самите можете да направите ваш звук на <https://experiments.withgoogle.com/ai/sound-maker/view/>

⁶Обидете се да препознаете кои снимки се од природен говор а кои се синтетизирани од Такотрон 2 со Вејвнет невронскиот вокодер во последниот дел на следната страна <https://github.com/tacotron/publications/tacotron2/index.html>

⁷Во следното видео асистентот на Гугл се јавува и закажува фризер и резервира ресторан за корисникот, притоа без корисникот да забележи дека не се работи за вистински човек https://www.youtube.com/watch?v=JvbHu_bVa_g

Глава 1

Дигитализација на звукот

Под **дигитализација** се подразбира процесот на префлување на еден аналоген процес, односно сигнал, во **дигитален домен**. Кај звукот, работиме со неговата претстава во аналоген електричен домен добиена со електроакустички претворувач -- микрофон. Така добиениот електричен сигнал е од аналоген карактер, односно тој е континуиран во време и е со континуирана распределба на амплитудите. Амплитудата на сигналот може да биде еднаква на било кое напонско помеѓу двете максимални нивоа на сигналот. Исто така, при премин од една конкретна вредност на неговата амплитуда во друга, тој поминува низ бесконечен број на состојби (во време и амплитуда) на патот меѓу нив.

Првиот чекор во дигитализацијата е **дискретизирање во време** на овој аналоген електричен сигнал, уште наречено и **земање примероци**¹. Дискретизацијата во време се прави со периодично земање примероци од дадениот аналоген сигнал во одредени временски интервали. Добиениот временски дискретен сигнал е во вид на Диракови импулси, но тој суштите има континуирано распределени амплитуди. За да се претстави амплитудата на секој од примероците со конечен број на битови, потребно е заокружување на нивните амплитуди до известни конкретни вредности од множеството вредности кои можат да се запишат. Ова е вториот чекор во дигитализацијата -- **дискретизација по амплитуда** или **квантизација**.

Во скlop на квантизацијата се прави и последниот чекор од процесот -- **кодирањето** на амплитудните вредности со низи од единици нули, со што конечно го добиваме звукот претставен во дигитален домен. Овие три чекори се спроведуваат во соодветни електронски кола наречени **аналогно--дигитални (A/D) конвертори**. Обратниот процес, односно реконструирањето на аналогниот сигнал од неговата дигитална претстава се врши соодветно со уредите наречени **дигитално--анalogни (D/A) конвертори**.

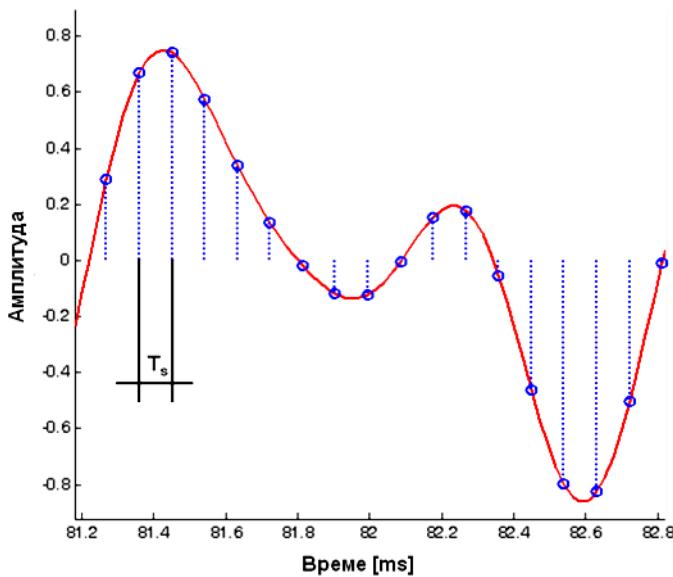
1.1 Земање примероци

Земањето примероци претставува запомнување на вредностите на аналогниот влезен сигнал согласно со ритамот одреден од фреквенцијата на земање примероци f_s , односно во временски интервали T_s дефинирани како:

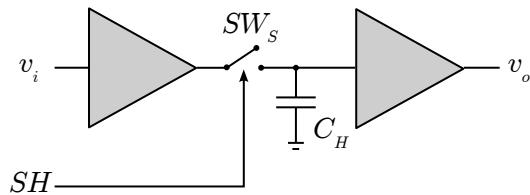
$$T_s = \frac{1}{f_s}. \quad (1.1)$$

Процесот на земање примероци е прикажан на Сл. 1.1. Ова се прави со помош на соодветни **Sample/Hold (SH)** електронски кола, а може математички да се претстави како множење на аналогниот сигнал со низа на Диракови импулси. Идеализирано SH коло е прикажано на Сл. 1.2. Тоа врши земање примероци на влезниот аналоген сигнал v_i со употреба на аналоген преклопник

¹Често се употребува и терминот семплирање кој доаѓа директно од англискиот термин *sampling*. Но, важно е да се напомене дека во музиката под поимот семплирање се подразбира земањето на отсекоци од готови музички траки или звуци од музички инструменти и нивната употреба за создавање на ново музичко дело.



Сл. 1.1: Земање примероци со фреквенција f_s , односно период T_s .



Сл. 1.2: Идеализирано Sample/Hold коло за земање примероци на аналоген влезен сигнал.²

SW_S , вообичаено реализиран со пар на MOSFET транзистори, односно во CMOS технологија, контролиран со поворка на правоаголни SH импулси. Кога прекинувачот е вклучен, C_H се полни на, односно „ја лови“, амплитудата на аналогниот сигнал, а кога е исклучен, C_H има задача да го држи „уловеното“ ниво за понатамошна квантизација. Двата операцијски засилувачи се во режим на напонски следила, т.е. кусо врзана негативна повратна врска и заземјен неинвертирачки влез. Првиот има задача *i*) да не го оптовари изворот на аналогниот сигнал и *ii*) да даде/прими голема струја за брзо попнење/празнење на C_H . Вториот има задача да оневозможи празнење на C_H во рамките на Н режимот, кога SW_S е исклучен, со својата бесконечна влезна импеданса.

Главната идеа е временската информација за сигналот содржана во земените примероци да биде доволна, за да може од нив потоа тој верно да се реконструира. На прв поглед ова изгледа невозможно -- како може нешто што трае континуирано во време да се претстави со конечна низа на вредности? Што станува со информацијата за сигналот помеѓу примероците? Сепак, математички е докажано дека ова е возможно. Станува збор за познатата **Теорема за земање примероци**³, која за првпат егзактно открива Владимир Котельников⁴ и ја применува во телекомуникациите во 1933^{ta} година. До нејзе независно дошле и други научници, и тоа: Хери Најквист⁵, Клод Шенон⁶, и Едмунд Витакер⁷. Теоремата за земање примероци гарантира дека за веродостојно зачувување на целосната информација содржана во еден аналоген сигнал

²Модифицирано од Ring0 - Own work, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=6990038>

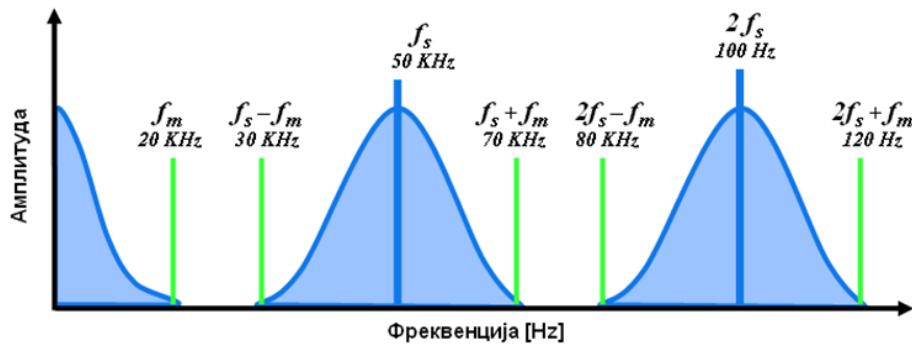
³Wikipedia: Nyquist-Shannon sampling theorem https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

⁴Wikipedia: Vladimir Kotelnikov https://en.wikipedia.org/wiki/Vladimir_Kotelnikov

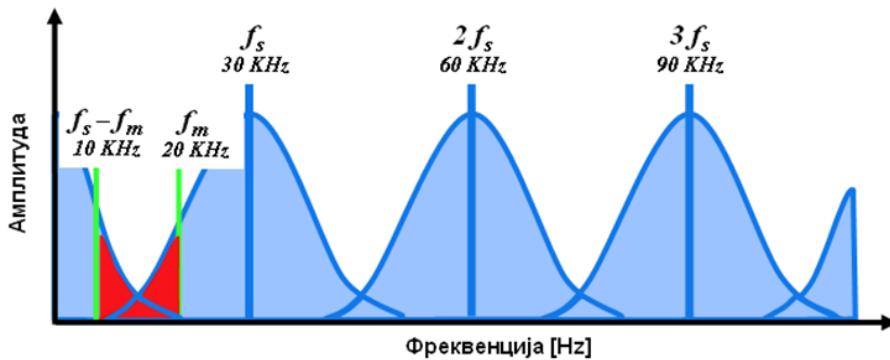
⁵Wikipedia: Harry Nyquist https://en.wikipedia.org/wiki/Harry_Nyquist

⁶Wikipedia: Claude Shannon https://en.wikipedia.org/wiki/Claude_Shannon

⁷Wikipedia: Edmund Taylor Whittaker https://en.wikipedia.org/wiki/E._T._Whittaker



Сл. 1.3: Спектар на дискретизираниот сигнал.



Сл. 1.4: Преклопување на спектрите при ниска фреквенција на земање на примероци.

со максимална фреквенција на неговиот спектар f_m , доволно е од него да земаме примероци со фреквенција $f_s \geq 2f_m$. Минималната фреквенција на земање примероци $f_s = 2f_m$ се нарекува **Најквистова фреквенција**.

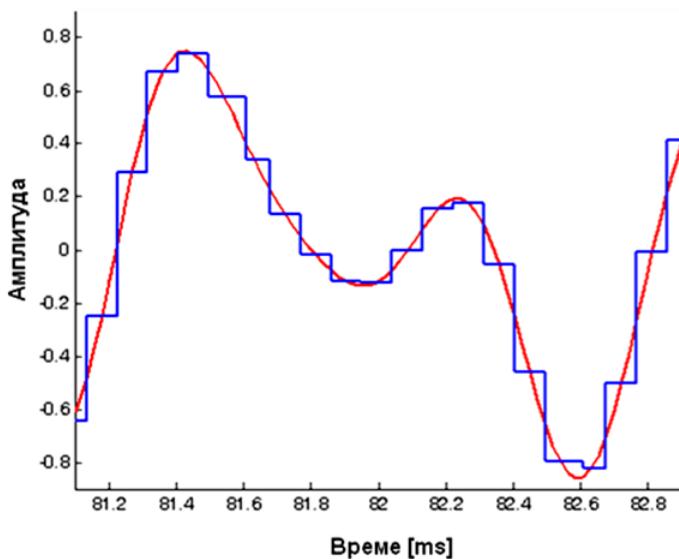
Спектарот на дискретизираниот сигнал е ист со овој на оригиналниот, со тоа што сега постојат копии на овој спектар центрирани на целобройни мултипили од фреквенцијата на земање на примероци f_s како на Сл. 1.3. На пример, ако звукот кој е ограничен на 20 kHz го дискретизираме со f_s од 50 kHz, тогаш слики од оригиналниот спектар ќе се наоѓаат во интервалите од 30 -- 70 kHz, 80 -- 120 kHz итн.

Бидејќи овие копии на спектарот на сигналот не смеат да се преклопуваат потребна е фреквенција на земање примероци поголема од $2f_m$, односно од Најквистовата фреквенција. Од друга страна бидејќи сигналите скоро никогаш немаат строго ограничен спектар, мора сигналот да се филтрира и да се сведен на фреквенциски опсег со максимална фреквенција f_m . Кога сигналот би имал спектрални компоненти над f_m , или пак кога би го дискретизидале со фреквенција $f_s < f_m$, сликите на оригиналниот спектар би се преклопувале. **Преклопувањето на спектрите**, прикажано е на Сл. 1.4.⁸ Тоа внесува изобличувања во оригиналниот спектар на сигналот -- овој во опсегот од 0 до f_m . Поради тоа, реконструкцијата на аналогниот сигнал нема да соодветствува на овој пред дискретизацијата.

Реконструирањето на аналогниот сигнал од неговите примероци се врши со употреба на нископропусен филтер (LP)⁹ со гранична фреквенција f_m . Тој во фреквенциски домен има задача да ги отстрани сликите од спектарот на сигналот лоцирани околу мултиплите на фреквенцијата на земање примероци f_s , по што ќе остане само оригиналниот спектар на сигналот. Во временски домен тоа се сведува на измазнување на дискретниот сигнал, прикажано на Сл. 1.5.

⁸Во англиската литература преклопувањето се нарекува **aliasing**, од зборот alias што значи лажно име, односно лажно претставување.

⁹анг. *low-pass filter*.



Сл. 1.5: Реконструкција на аналогниот од дискретизираниот сигнал со нископропусно филтрирање.

Фреквенции на земање примероци кај дигиталните звучни сигнали

Брзини со кои се врши земањето примероци кај дигиталните аудиосигнали се:

- 8.000 Hz -- се употребува во телефонската комуникација и е доволна за пренесување на говор со зачувана разбираливост,
- 11.025 Hz и 22.050 Hz -- четвртина и половина од фреквенцијата на земање примероци во аудио-CD стандардот 44.100 Hz, се употребува за зачувување на дигитален звук со понизок квалитет,
- 16.000 Hz -- доволна за да го опфати поголем дел од спектарот на говорниот сигнал којшто е од значење за разбираливоста на говорот, па се употребува кај системите за препознавање на говор (ASR)¹⁰, наместо поголеми фреквенции на земање примероци кои би ја зголемиле големината на податоците кои треба да се процесираат,
- 24.000 Hz -- го опфаќа целиот спектар на говорниот сигнал и се употребува кај системите за синтеза на говор од текст (TTS)¹¹ за да се добие квалитетна синтеза,
- 32.000 Hz -- се употребува во miniDV стандардот за дигитално видео кој го користат дигиталните видео камери, во Digital Audio Tape (DAT) стандардот во модовите за зголемено траење на касетата (long play mode), како и Германското дигитално сателитско радио Digitales Satelitenradio,
- 44.100 Hz -- фреквенција на земање примероци кај аудио-CDто, наследена од PCM адаптерите, исто така најчесто се употребува кај MPEG кодираниот звук (како кај VCD, SVCD, MP3),
- 48.000 Hz -- во употреба кај miniDV, дигиталната Телевизија, DVD, D/AT, филмови и професионална аудиоопрема,
- 96.000 Hz или 192.000 Hz -- во употреба кај DVD стандард, во новата генерација на DVD--а со син ласер (Blu--ray Disc) и во HD--DVD (High--Definition DVD),
- 2,8224 MHz -- се употребува во SACD (Super Audio CD) стандардот развиен од Сони и Филипс, кој употребува 1--битна сигма--делта квантација. Овој начин на дигитализација

¹⁰анг. *Automatic Speech Recognition*.

¹¹анг. *Text-to-Speech*.

не нуди предности над стандардниот кој е во општа примена.

§ Дополнително. Аудио--CD стандардот употребува фреквенција на земање примероци од 44,1 kHz што соодветствува на максимална фреквенција на спектарот на сигналот од 22 kHz. Ова е поголем опсег од слушниот опсег на човекот. Тој е избран поради тоа што кога се воведувал аудио--CD стандардот во 1980^{ta}, единствениот систем кој овозможувал фреквенциски опсег на записот доволно голем за да се зачувува звукот во дигитален формат бил VHS системот за видео касети. Тогаш постоеле PCM адаптери кои аналогните звучни сигнали го дигитализирале а потоа повторно го претворале во аналоген видео сигнал кој се носел во видеото за снимање. Тие можеле да запишат по 3 примероци од секој звучен канал во една хоризонтална видео линија. Бидејќи PAL системот има 294 линии и фреквенција на работа од 50 Hz, следува дека брзината на земање примероци може да изнесува $294 \times 50 \times 3 = 44.100$ примероци/секунда.

Надземање примероци

Директниот начин на кој може да се изведе земањето примероци со фреквенција од на пример 44,1 kHz, како кај аудио--CDто, е да се направи токму тоа -- со еден A/D конвертор да се одмери сигналот во кратки временски интервали со фреквенција од 44,1 kHz. Меѓутоа, ова бара влезниот аналоген ниско пропусен (LP) филтер, види Глава 4, во A/D конверторот низ кој најпрвин минува сигналот, да е со многу стрма амплитудна карактеристика веднаш над 20^{ot} kHz. Овој филтер треба во многу краток фреквенциски интервал да обезбеди слабеење од повеќе од 80 dB, колку што изнесува потребниот однос сигнал-шум за Hi-Fi репродукција на звук. Но, бидејќи динамичкиот опсег кој го нуди дигиталниот аудио--CD запис е 96 dB, всушност слабеењето на филтерот треба да е уште поголемо. Во практиката, ова подразбира употреба на аналоген филтер од 8^{mi} или 10^{ti} ред што е скапо и решение кое не се препорачува, поради потребата за прецизна усогласеност на составните аналогни компоненти.

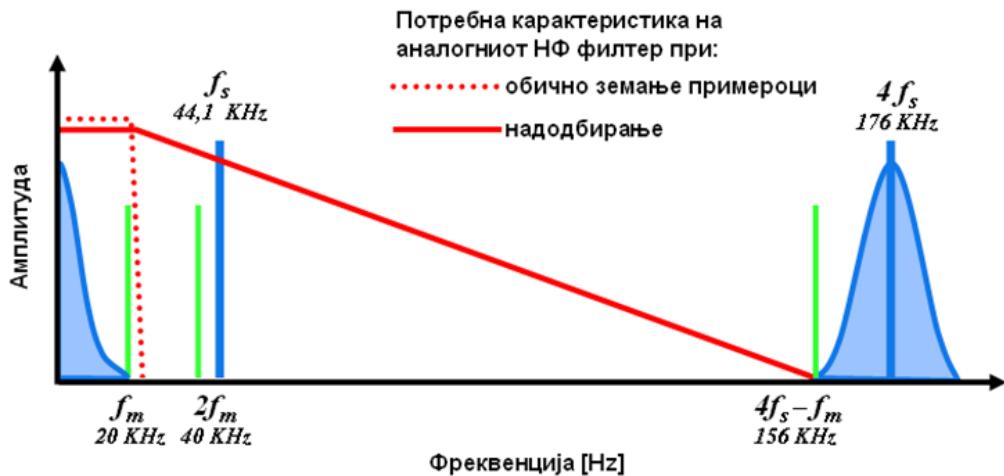
Постои уште еден поекономичен начин да се реализира земањето примероци, а тоа е со употребата на **надземање примероци**¹². Надземањето примероци претставува земање на примероци со фреквенција неколку пати поголема од Најквистовата. Со него се постигнува олеснување на условите кои влезниот LP филтер треба да ги задоволи при обликувањето на аналогниот сигнал. Потребното ограничување на спектарот на сигналот, сега може да се изврши во дискретен (дигитален) домен каде нископропусното филтрирање на сигналот е многу поедноставно, а и поевтино. Уште една голема предност на дигиталното филтрирање над аналогното е строгата линеарност на фазната карактеристика која е гарантирана кај **FIR**¹³ **филтрите** со конечен импулсен одсив.¹⁴ Потоа може да се отфрлат вишокот примероци, со што доаѓаме до целната фреквенција на земање примероци.

Фреквенцијата со која се прави надземање примероци најчесто се движи од $4\times$, $8\times$, па сè до $32\times$ од Најквистовата фреквенција. Најчесто тоа се прави со 4 или 8-кратно поголема фреквенција. Во SACD стандардот се употребува фреквенција на надземање примероци од 2,8224 MHz што соодветствува на $64\times$ надземање примероци.

¹²Во англиската терминологија тоа е познато како **oversampling**

¹³анг. Finite Impulse Response.

¹⁴Повеќе за филтри во Главата 4.



Сл. 1.6: Благиот наклон на влезниот LP филтер овозможен со надземањето примероци во споредба со потребната карактеристика на филтерот приично земање на примероци.

◀ Пример За да ги илустрираме придобивките од надземањето примероци ќе претпоставиме $4 \times$ надземање примероци во однос на аудио-CD стандардот, прикажано на Сл. 1.6. Тогаш фреквенцијата на надземање примероци f'_s ќе биде 4 пати поголема од номиналната (целната) f_s од 44,1 kHz и ќе изнесува:

$$f'_s = 4f_s = 4 \cdot 44,1 = 176,4 \text{ kHz.} \quad (1.2)$$

Во овој случај за да нема преклопување на спектрите кај дискретниот сигнал, потребно е спектраторот на сигналот да не содржи значителни компоненти над:

$$f'_m = \frac{f'_s}{2} = \frac{176,4}{2} = 88,2 \text{ kHz.} \quad (1.3)$$

Така, аналогниот филтер може да биде со поблаг пад во карактеристиката и сепак да ги задоволи условите за висок квалитет на записот. Падот може да започне нешто над 20 kHz и да трае сè до 88st kHz, каде треба да го достигне слабеењето од 80 dB. Уште повеќе, нас не нè интересира верна репродукција на целиот опсег од 88,2 kHz кога корисниот аудиосигнал ни се наоѓа до 20 kHz. Значи, фреквенцијата по која филтерот треба да го постигне бараното слабеење се поместува надесно до $176,4 - 20 = 156,4$ kHz. Следува дека сега преодниот фреквенциски опсег на филтерот изнесува 120 kHz споредено со 4,1 kHz во случајот кога не се користи надземање примероци. Ова во голема мера ја поедноставува изградбата на аналогниот филтер.

По надземањето примероци, сигналот ги следи останатите два чекора на дигитализацијата (квантизација и кодирање) со што од него се добива дигитален сигнал. При тоа, поради надземањето примероци, овој дигитален сигнал е дигитална претстава на аналоген сигнал со спектрални компоненти од 0 до 88,2 kHz. Овој опсег нас не ни е од интерес, па треба да го сведеме дигиталниот сигнал на целната фреквенција на земање примероци $f_s = 44,1$ kHz. Најпрвин за да го издвоиме корисниот спектар, дигиталниот сигнал треба да го исфилтрираме со дигитален LP филтер со стрмно отсекување на фреквенциите над 22 kHz, кое може лесно да се реализира во софтвер, односно во дигиталната интегрирана техника. Исто така филтерот може да има идеално линеарна фазна фреквенциска карактеристика, што е тешко да се направи во аналоген домен, но и не е критично кај звучните сигнали поради спомнатата фазна неосетливост на човековото уво.

Потоа ја намалуваме фреквенцијата на земените примероци преку процесот наречен

скастрување¹⁵. Ова наједноставно се прави со отфрлање на 3 од секои 4 примероци со што добиваме верна претстава на аналогниот звучен сигнал со спектар до 22 kHz во дигитален домен. Во стварноста не се отфрлаат по 3 примероци од секои 4 туку самиот дигитален филтер ја пресметува вредноста на еден излезен примерок на секои 4 влезни со нивно усреднување.

§ Дополнително. Процесот на надземање примероци наоѓа примена и во реконструкцијата на аналогниот сигнал од неговата дигитална претстава. Кај D/A конверторите надземањето примероци се изведува со додавање на екстра нулти примероци на секој реален примерок.^a Притоа нивната амплитуда најчесто се интерполира помеѓу вредностите на реалните примероци или пак математички се моделира, во зависност од изведбата на D/A конверторот. Овој процес е фундаментално различен од надземањето примероци при дискретизацијата на влезниот сигнал. Имено тој не внесува додатна информација во дигиталниот сигнал. Сепак, како и претходно, надземањето примероци ја олеснува изградбата на излезниот аналоген LP филтер во D/A конверторот, преку зголемувањето на минималната фреквенција на првата слика на звучниот спектар.

^aНа англиски ова се нарекува up-sampling.

Цитер

Цитерот претставува грешка во временските моменти во кои се прави земањето примероци поради варијациите во временската база на уредот за дигитализација. Односно, A/D конверторот со фреквенција на земање примероци од 44,1 kHz, не секогаш ги зема примероците точно секој 44.100^{ти} дел од секундата. Тие може да бидат земени малку порано или покасно со што снимената вредност на влезниот сигнал не сосема соодветствува на вредноста која тој би ја имал во предвидениот временски момент. Ефектите на цитерот на временска база стануваат понагласени при присуство на високи фреквенции, како и на големи амплитуди кај влезниот сигнал.

1.2 Квантизација

За теоремата за земање примероци да важи целосно, амплитудата на секој примерок мора да е еднаква со онаа на влезниот сигнал во тој временски момент. Меѓутоа, поради тоа што секој примерок во дигиталниот запис мора да се запише (кодира) со ограничен број на битови, наречен **дигитален збор**, следува дека и бројот на амплитуди, односно множеството амплитуди кои можат да се запишат е конечно. Така, ако бројот на битови во дигиталниот збор е N , тогаш, вкупниот број на амплитуди кои со него можат да се запишат е 2^N . Поради тоа континуираните вредности на влезните амплитуди мора да се заокружат или да се мапираат во ним најблиските им вредности во рамките на ова множество. Тој процес се нарекува **квантизација**, а електронскиот уред со кој таа се изведува **квантизер**¹⁶. Бројот на битови кој квантизерот може да го генерира се нарекува уште и негова **резолуција**.

Квантизацијата се одвива на тој начин што квантизерот врши споредба на амплитудата на влезниот сигнал со дискретните нивоа кои тој може внатрешно да ги генерира, наречени **квантизациски нивоа**. Разликата помеѓу две соседни квантизациски нивоа се нарекува **чекор на квантизација** и се означува со q .

Постојат два главни пристапи за квантизација:

- **паралелна** -- кога влезниот сигнал се споредува паралелно со низа на интерни напонски нивоа кои одговараат на квантизациските нивоа на квантизерот. Овие нивоа вообично се генериирани со повеќенировски отпорнички делител од референтниот напон на квантизерот

¹⁵Соодветниот англиски термин е downsampling.

¹⁶Влезниот аналоген филтер, земачот на примероци, квантизерот и на крај кодерот се составни делови на еден А/D конвертор.

V_{REF} . Поради својата брзина на работа овие квантизери се нарекуваат и флеш¹⁷ квантизери. Бидејќи во својата конструкција изискуваат ист број на компаратори (операцијски засилувачи) колку што има квантизациски нивоа, овој тип на квантизери се употребуваат во мал број на апликации кај кои е потребна голема брзина на земање примероци. Вообичаено за добивање поголеми резолуции се реализираат како каскада од неколку флеш секции.

- **секвенцијална** -- споредбата тече во неколку чекора низ кои квантизерот ја „лови“ амплитудата на влезниот сигнал, од најгрубата негова проценка, која соодветствува на половина од неговиот референтен напон V_{REF} , а се кодира со битот со најголема тежина -- MSB¹⁸, до најголемата прецизност која тој може да ја постигне, која е $V_{REF}/2^N$ и се кодира со битот со најмала тежина LSB¹⁹. Процесот е воден од внатрешната логика на квантизерот. Важно е брзината со која се доаѓа до конечниот коден збор да е поголема од брзината со која доаѓаат примероците од влезниот сигнал. Едно коло за секвенцијална квантизација е прикажано на Сл. 1.7.

§ Дополнително. Прикажаното коло за секвенцијална квантизација содржи еден операцијски засилувач во режим на работа на компаратор кој го споредува влезниот аналоген напон V_a со внатрешно генерираните напонски нивоа на C_4 . Во првата периода на тактот на ова коло му се носи правоаголен импулс за ресетирање φ_R , којшто го полни C_1 на V_{REF} и го празни C_2 на 0. По ова тактот на φ го врши секвенцијалното „ловење“ на V_a . Во првата периода по ресетирањето C_3 се полни на V_{REF} , па во негативниот дел од оваа периода, кога е активен $\bar{\varphi}$, преку паралелната врска на C_3 и C_4 кои имаат иста капацитивност, доаѓа до поделба на напонот на половина. Така, првото ниво со кое се споредува V_a е всушност $V_{REF}/2$, а резултатот на оваа споредба ќе го даде првиот бит од конверзијата MSB на излезот b_i . Ако $V_a > V_{REF}/2$, MSB ќе биде 1, а во следниот такт C_3 повторно ќе се наполни од горе на V_{REF} , што во негативниот дел од овој такт ќе го качи C_4 на $3/4V_{REF}$. Во спротивно, MSB ќе биде 0, а C_3 ќе се испразни на 0, па следното ниво на C_4 ќе биде $1/4V_{REF}$. Постапката продолжува сè до генерирање на последниот бит LSB. Ако компонентите се идеални, оваа постапка може да трае бесконечно, но во реалноста поради нивните толеранции, резолуцијата на квантизерот е конечна.

✓ **Задача за дома.** Испртајте ја работата на ова коло за 10 такта на φ и $V_a = 0.7V$, $V_{REF} = 1V$.

Постојат два главни типа на квантизација кои се разликуваат според начинот на кој го ловат влезниот сигнал. Тие се **квантизација со заокружување** и **квантизација со отсекување**. Кај првата излезот на квантизерот го дава квантизациското ниво најблисоко до амплитудата на влезниот сигнал, додека кај втората тоа е квантизациското ниво веднаш под амплитудата на влезниот сигнал. Типот на квантизација зависи од електронската реализација и механизмот на работа на квантизерот. Разликите помеѓу двата типа се илустрирани на Сл. 1.8.

Квантизација на дигиталниот звук

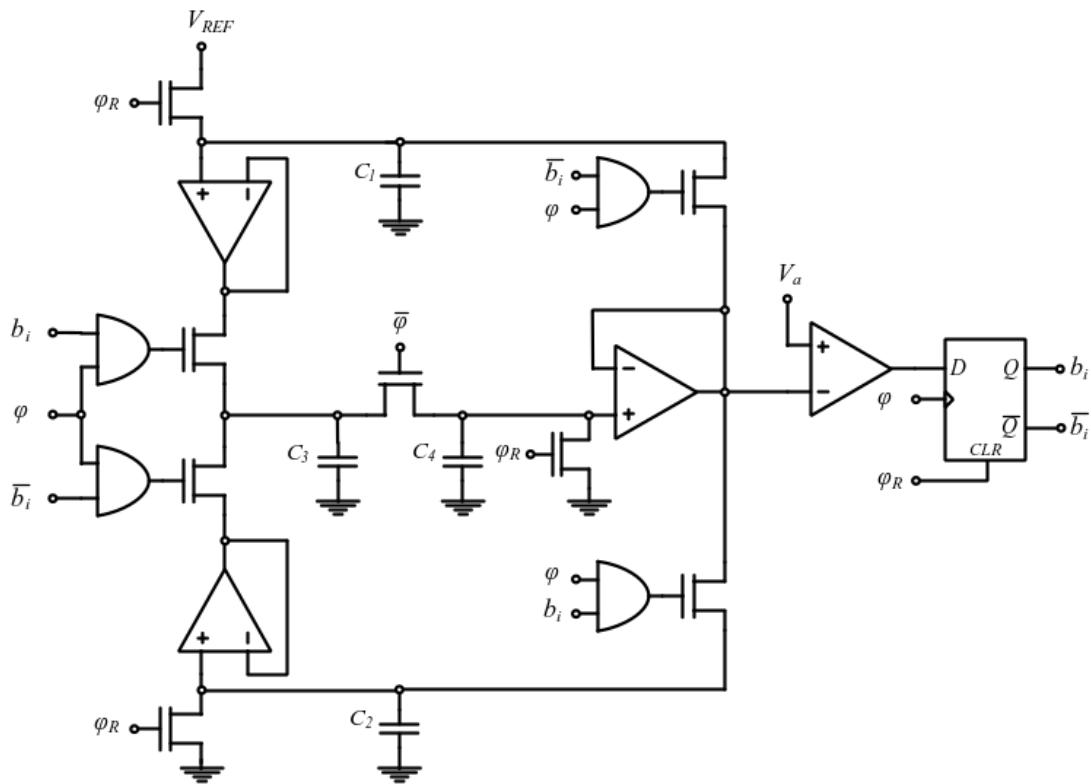
При дигитализација на аудиосигналите се употребуваат следните резолуции на квантизација:

- **1 бит** -- кај супер аудио-CD стандардот (SACD) кој користи надземање примероци и техники за обликување на шумот на квантизација за постигнување на голем однос сигнал-шум и при 1-битна резолуција.
- **4 бита** -- ретко се употребува при линеарна импулсно кодна модулација (PCM), а во употреба е кај адаптивната квантизација.

¹⁷анг. *flash*

¹⁸Most Significant Bit.

¹⁹Least Significant Bit.



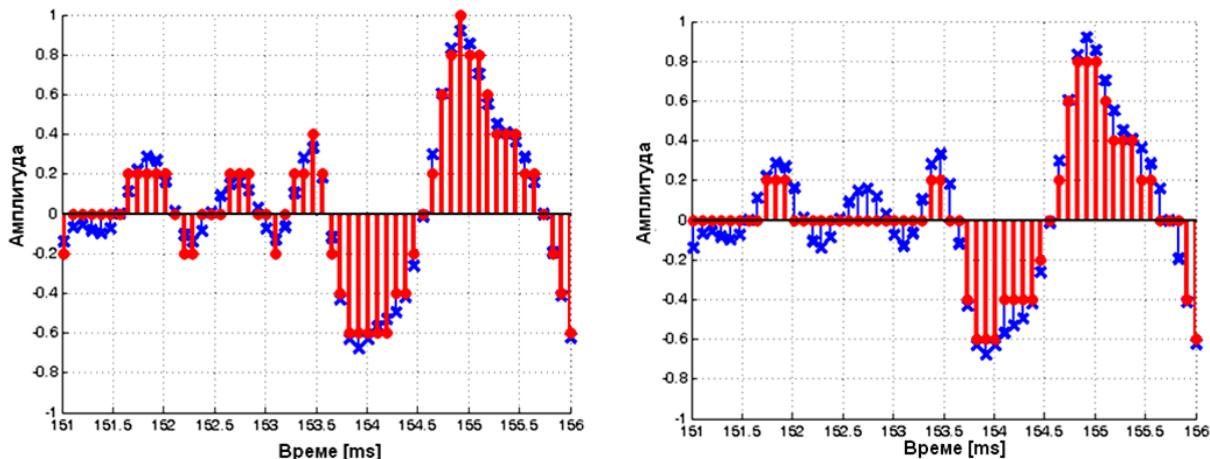
Сл. 1.7: Коло за секвенцијална квантација на аналоген напон.

- 8 бита -- во употреба во телефонските системи за кодирање на говорот при нелинеарна квантација, стандардна резолуција на старите звучни карти во ерата на флопи-дискетите,²⁰
- 12 бита -- во употреба во 1980^{te} во дигиталните уреди за ефекти употребувани во музицирањето, како за електричните гитари.
- 16 бита -- дел од аудио-CD стандардот, денес стандардна резолуција за повеќето уреди за дигитален звук (на пример звучни картички).
- 20 бита -- вградена во многу A/D конвертори, во новата серија 20-битни A/DAT повеќеканални машини и кај некои звучни картички.
- 24 бита -- новиот стандард за висока дефиниција, имплементиран во нови професионални уреди за дигитален звук, новите DAT машини и во аудио-DVD стандардот.
- 32 и 64 бита -- не се употребува за снимање на звук, туку за неговата дигитална обработка и спектрална анализа со софтверските алатки; служи за зголемување на точноста во пресметките, особено значајно кога тие би вовеле дисторзија во 16-битно запишан звук.

Кодирање

Под кодирање се подразбира начинот на кој дискретната амплитуда на квантизерот ќе биде запишана во битови и тоа е вградено во самата негова изведба. Наједноставниот начин на кодирање на амплитудите на влезниот сигнал е со нивно директно претворање во бинарни (кодни, дигитални) зборови. Притоа паровите амплитуда-коден збор се [линеарно распоредени](#), со најниската амплитуда запишана со најмалиот дигитален збор (сите 0), а највисоката амплитуда со најголемиот дигитален збор (сите 1). Малку посложена е [нелинеарната квантација](#), односно кодирање, кај која амплитудите со најголема веројатност на појавување пофино се квантизираат

²⁰Денес се јавува во уметничкото движење кое употребува 8 битен звучен израз.



Сл. 1.8: Квантизација (првено) на дискретизираниот сигнал (сино) со заокружување (лево) и со отсекување (десно).

од оние кои поретко се појавуваат. На пример, за крајно високите амплитуди се отстапени помалку кодни зборови отколку за амплитудите поблиску до нулата.

Така, при квантизирање на говорот, посебно во класичните 8-битни телефонски системи, но и денес во врвните синтетизатори на говор како WaveNet, вообичаено се употребува нелинеарна квантизација со крива за компресија, наречена и -крива.²¹ -кривата е дефинирана со G.711 стандардот на Меѓународната унија за телекомуникации и е описана со равенството:

$$f(x) = \text{sgn}(x) \begin{cases} \frac{A|x|}{1+\ln(A)}, & |x| < \frac{1}{A} \\ \frac{1+\ln(A|x|)}{1+\ln(A)}, & \frac{1}{A} \leq |x| \leq 1, \end{cases} \quad (1.4)$$

каде A во Европа е дефинирано да биде 87,6. На Сл. 1.9 е дадена преносната функција на оваа крива. Како што може да се забележи, по компресијата поголем број на квантни нивоа се на располагање за квантизација на помалите амплитуди на влезниот сигнал, а помал број за поголемите. Ова одговара на статистичката распределба на амплитудите на говорниот сигнал, во кој позастапени се малите амплитуди, па и од таму нужноста за нивна поточна квантизација.

Исто така постојат и методи на квантизација кои ја кодираат разликата помеѓу две последователни амплитуди на влезниот сигнал, така наречена адаптивна квантизација. Тие се употребуваат во апликации каде големината на дигиталниот проток е критичен параметар.

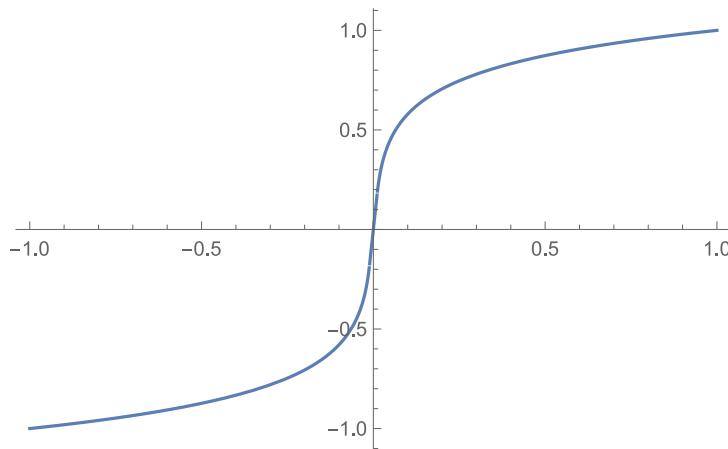
Шум на квантизација

Грешката при квантизација може да ја сметаме и како шум додаден на некоја инаку идеална амплитудна вредност. Гледано од тој аспект, таа се нарекува и **шум на квантизација**. Под услови:

1. влезниот сигнал да може да се разгледува како случаен процес и
2. тој да има **многу поголема амплитудна динамика** од чекорот на квантизација q , што е еквивалентно на тоа чекорот на квантизација да е доволно мал,

шумот на квантизација може да го моделираме како **бел Гаусов шум**, односно тој ќе биде случаен и **некорелиран** со влезниот сигнал, со средна вредност 0 и максимална вредност $\pm q/2$ во случај на квантизација со заокружување, односно средна вредност $q/2$ и максимална амплитуда q , кога се врши квантизација со отсекување. Во овие услови, моќноста на шумот на квантизација ќе биде рамномерно распределена долж целиот спектар и со константа амплитуда наречена **ниво**

²¹A-law algorithm https://en.wikipedia.org/wiki/A-law_algorithm



Сл. 1.9: Графичка претстава на преносната функција добиена со А кривата за нелинеарна квантизација на говор.²²

на шумот на квантизација. Во праксата дава услови се исполнети речиси секогаш, особено за сигнали кои имаат мал коефициент на автокорелација, како што се говорот и музиката, како и кога квантизерот е со доволно голема резолуција. Условите не се исполнети само во многу специјални случаи како што се константни сигнали, простопериодични сигнали синхронизирани со фреквенцијата на дискретизација и многу слаби сигнали.

Односот меѓу просечната моќност на аналогниот сигнал и нивото на шумот на квантизација се нарекува **однос сигнал-шум (SNR)**²³ на дигиталниот сигнал. Неговата вредност може да се добие со помош на равенството:

$$SNR = 10 \log \frac{P_x}{P_N} = 10 \log \frac{\sigma_x^2}{\sigma_q^2}, \quad (1.5)$$

каде P_x и P_N се моќноста на аудиосигналот и шумот на квантизација, кои ако се случајни процеси можат да се пресметаат преку нивните стандардни девијации σ_x и σ_q .

Ако влезниот сигнал $x[n]$ ги задоволува гореспоменатите два услови тогаш густината на веројатност на грешка е рамномерно распределена во интервалот од $-q/2$ до $q/2$ па за σ_q добиваме:

$$\sigma_q^2 = \frac{q^2}{12}. \quad (1.6)$$

Ако земеме дека $V_{REF} = 1$, тогаш $q = \frac{1}{2^N}$, па добиваме:

$$\sigma_q^2 = \frac{1}{12 \cdot 2^{2N}}, \quad (1.7)$$

па (1.5) станува:

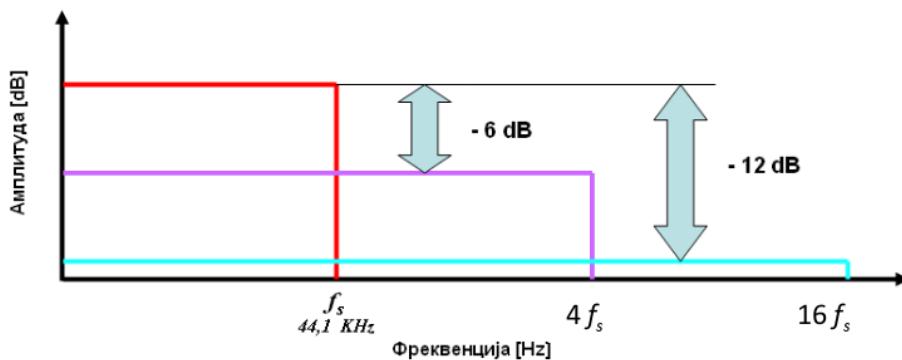
$$SNR = 10 \log \left(\frac{\sigma_x^2}{\frac{1}{12 \cdot 2^{2N}}} \right) = 10 \log(12 \cdot 2^{2N} \sigma_x^2). \quad (1.8)$$

При пресметување на равенството (1.8) можеме влезниот сигнал да го моделираме како Гаусов, при што е важно тој да не ја надмине границата на квантизаторот, којашто земавме дека е 1. Ако варијансата на Гаусовиот процес $\sigma_x < 0,25$, тогаш веројатноста да се случи пречекорување е занемарлива, т.е. $< 0,0001$, што речиси секогаш е исполнето при моделирањето на звукот како Гаусов процес. Неколку вредности на варијансата на Гаусовиот процес добиен со моделирање на различни типови звук се дадени во Табела 1.1. Конечниот израз за односот сигнал-шум во однос на должината на кодниот збор N кој се добива со земање на $\sigma_x = 0,25$ е:

$$SNR = 6,02n - 1,25 \simeq 6n [\text{dB}]. \quad (1.9)$$

Табела 1.1: Варијанса на Гаусовиот процес употребен за моделирање на неколку звучни сигнали.

Вид на звучен сигнал	σ
Чалгиска музика	0,150
Изворна музика	0,236
Пеење	0,326
Говор	0,223



Сл. 1.10: Намалувањето на нивото на шумот со употреба на надземање примероци.

Во праксата речиси секогаш се користи поедноставената форма, поради поврзаноста на константниот член со варијансата на звучниот сигнал, како и поради неговата едноставност. Како што гледаме, односот сигнал-шум расте за 6 dB со секој додатен бит во должината на кодниот збор. Според тоа за односот сигнал-шум во аудио-CD стандардот, со 16-битна резолуција, добиваме вредност од 96 dB (94,75 dB). За резолуција од 24 бита по примерок пак, добиваме однос сигнал-шум од 144 dB, што е еквивалентен на слушниот опсег на човекот.

Квантизација при надземање примероци

Ако влезниот аналоген сигнал е случаен процес, па шумот на квантизација е бел и некорелиран, тој ќе биде рамномерно распределен долж целото фреквенциско подрачје кое го зафаќа сигналот. Што се случува ако направиме $4 \times$ надземање примероци -- односно ако A/D конверторот зема примероци со фреквенција од $4 \times 44,1$ kHz? Во овој случај шумот од квантизација ќе биде распределен на 4 пати поширок фреквенциски опсег, па соодветно и амплитудата на неговата спектрална густина на моќност ќе биде 4 пати помала. Тоа соодветствува на ниво на шум помало за $10 \log^{1/4} = -6$ dB. Со други зборови $4 \times$ надземање примероци има ист ефект врз односот сигнал-шум како и зголемувањето на должината на дигиталниот збор за 1 бит. Слично, $16 \times$ надземање примероци соодветствува на зголемена резолуција на A/D конверторот од 2 бита, како што е прикажано на Сл. 1.10.²⁴

Ова може да се каже и на следниов начин: 16-битен A/D конвертор со брзина на земање примероци од 44,1 kHz нуди ист однос сигнал-шум како 15-битен A/D конвертор со $2 \times$ надземање примероци. Ако одиме понатаму по оваа логика стигнуваме до поедноставени A/D конвертори кои со помалку битови по примерок но со многу поголема фреквенција на земање примероци нудат перформанси исти со оние на посложените 16 или 24 битни A/D конвертори. Крајната точка во оваа дискусија е реализација на квалитетни A/D конвертори кои работат со 1 бит по

²²Преработено од [https://commons.wikimedia.org/wiki/File:Plot_of_F\(x\)_for_A-Law_for_A_%3D_87.6.svg](https://commons.wikimedia.org/wiki/File:Plot_of_F(x)_for_A-Law_for_A_%3D_87.6.svg) од Masterxilo1992 CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>

²³Signal-to-Noise Ratio.

²⁴Ова може да се протолкува како поништување на шумот при усреднување на екстра земените примероци. Сличен процес се користи во астрономската фотографија каде поради слабата осветленост, шумот од сензорот е многу голем, па се земаат низа од фотографии од објектот и се усреднуваат во една во која шумот е видливо намален.

примерок, како кај супер аудио-CD (SACD) стандардот²⁵.

§ Дополнително. Надземањето примероци нуди подробности и во спротивната насока, односно кај D/A конверторите. Тука повторно шумот се прераспределува на поширок фреквенциски опсег па можно е отфрлање на дел од битовите по примерок, а при тоа задржување на истиот однос сигнал-шум.

Дитер

Претходната дискусија за нивото на шум на квантизација важи во случај кога шумот на квантизација е потполно некорелиран со влезниот сигнал. Во праксата, колку повеќе амплитудата на влезниот сигнал се приближува кон амплитудата означена со најмалку значајниот бит, толку повеќе се зголемува корелираноста на шумот на квантизација со самиот сигнал. Како последица шумот на квантизација не може веќе да се моделира како некорелиран бел шум. Неговата корелираност значи дека шумот на квантизација ќе има периодична природа па соодветно ќе внесе сопствени хармоници во спектарот на сигналот. Увото ги толкува овие изобличувања како непријатна дисторзија.

Дитерот²⁶ е процес со кој може да се елиминира појавата на дисторзија во сигналот под дејство на шумот од квантизација.²⁷ Тој е воведен од Лоренс Робертс²⁸, еден од основачите на интернетот, во својата магистерска теза на МИТ во 1961. Во него, на влезниот сигнал пред дигитализацијата му се додава мало ниво на аналоген шум. Случајниот карактер на шумот внесува случајност и во однесувањето на квантizerот при ниски влезни нивоа на сигналот -- ефективно внесувајќи несигурност во неговата работа. На тој начин, грешката од квантизација се декорелира од сигналот. Ова доаѓа од таму што сега, излезните нивоа од квантizerот веќе не зависат само од влезниот сигнал туку и од некорелираниот случаен процес внесен како шум.

Концептот зад треперењето може интуитивно да изгледа нелогичен, но принципот на кој функционира е многу едноставен. Треперењето се базира врз одредено специфично однесување на човековото уво. Увото е помалку чувствително на широкопојасен шум со рамномерно ниво, отколку на хармониски изобличувања во спектарот на сигналот кои се јавуваат при корелираност на шумот со сигналот. Бидејќи увото може да открие звук во опсегот на средните фреквенции 10 до 12 dB под нивото на шумот, со треперење може да се постигне 18-битна резолуција кај еден 16-битен запис. Имено, при правилен избор на сигналот со кој се изведува треперењето се овозможува да се запишат сигнали и 110 dB под максималното ниво, иако 16-битната резолуција теориски нуди динамички опсег од 96 dB. Треперењето е во редовна употреба при намалувањето на резолуцијата на еден дигитален звучен запис, како што е редовно случај при мастерирањето. Во тој случај, материјалот снимен во студиото со професионална опрема со 24-битна резолуција, треба да се прилагоди за дистрибуција на носач на звук -- CD со 16 бита. Со внесувањето на одредено ниво на шум, дитер, пред конверзијата се овозможува зачувување на дел од вишокот информација која ја содржи оригиналниот запис. Ако ова не се направи, тогаш простото отфрлање на информацијата зачувана во 24-битна резолуција со исфрлање на долните 8 битови се нарекува **скастрување**²⁹. Ова неминовно ќе доведе до појава на корелиран шум на квантизација како дисторзија во конечниот аудиозапис.

Пример за употреба на дитер³⁰ е прикажан на Сл. 1.11. На левиот графикон е прикажан простопериодичен тон со f_0 од 750 Hz со f_s од 48 kHz и квантизиран со 4 битна резолуција без употреба на дитер. Бидејќи фреквенцијата на земање примероци на синусот е мултипл од

²⁵SACD користи и техники за обликување на шумот дискутиирани во Главата за уште поголеми придобивки во SNR.

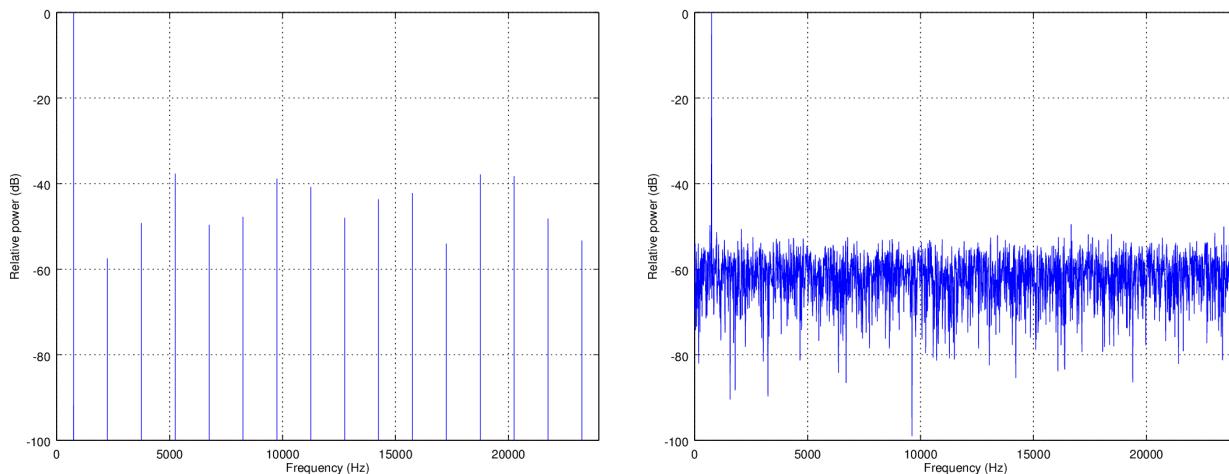
²⁶Македонскиот термин за дитер би бил „треперење“, а следи од адаптација на англиското *to dither*, чие потекло е од старо-англискиот збор *didderen* кој значи тресење како од студ.

²⁷Wikipedia: Dither <https://en.wikipedia.org/wiki/Dither>

²⁸Wikipedia: Lawrence Roberts https://en.wikipedia.org/wiki/Lawrence_Roberts_%28scientist%29

²⁹англ. *truncation*.

³⁰Wikipedia: Noise shaping https://en.wikipedia.org/wiki/Noise_shaping



Сл. 1.11: Синусен тон на фреквенција од 750 Hz од кои се земени примероци со мултипл од таа фреквенција од 48 kHz и квантизиран на 4 бита без (лево) и со употреба на дитер (десно).³¹

неговата основна фреквенција ($64\times$), шумот на квантација е корелиран со сигналот и самиот е периодичен. Поради тоа спектрарот на шумот на квантација е составен од низа на хармоници на f_0 , односно станува збор за хармониско изобличување на влезниот сигнал. На графиконот десно е прикажан истиот тон, но овојпат на него е додаден дитер со триаголна дистрибуција пред квантацијата на 4 битови. Може да видиме дека и покрај тоа што нивото на шум е поголемо, шумот на квантација е сега бел шум и немаме појава на хармониски изобличувања. Односот сигнал-шум постигнат со примена на дитерот е 60 dB.

§ Дополнително. Дитерот бил употребен прв пат во II Светска војна, кога забележале дека механичките компјутери за пресметување на траекторијата на бомбите при бомбардирањето работеле подобро во авионите отколку надвор од нив. Се покажало дека причината за ова е постојаното тресење на авионот кое овозможило полесно движење на деловите од овие компјутери кои биле замастени и слепени. Подоцна минијатурни мотори -- вибратори наречени дитери се вградувале во сите механички компјутери.

§ Дополнително. Ефикасноста на дитерот може добро да се илюстрира во неговата примена при намалување на битската длабочина кај сликите. Така, на Сл. 1.12 е прикажана реквантација на една 8-битна сива слика на 1-битна црно-бела слика без и со примена на дитер.

Обликување на шумот

Дитерот додава шум со рамна спектрална распределба на моќноста во аналогниот сигнал, но увото не е еднакво осетливо во целиот слушен опсег како што може да се види од изофонските криви прикажани на Сл. 1.13. Изофонските криви ја прикажуваат субјективната јачината на звукот и физичкото ниво на звучниот притисок. Први ги измериле Флечер и Мунсон (1933), па потоа и Робинсон и Дадсон (1956), за сега да бидат стандардизирани со ISO стандардот 226:2003. Тие ја отсликуваат фреквенциски нелинеарната осетливост на сетилото за слух.³³

³¹By Andrew D'Addesio - Own work, CC0, <https://commons.wikimedia.org/w/index.php?curid=44975277> и <https://commons.wikimedia.org/w/index.php?curid=4497526>

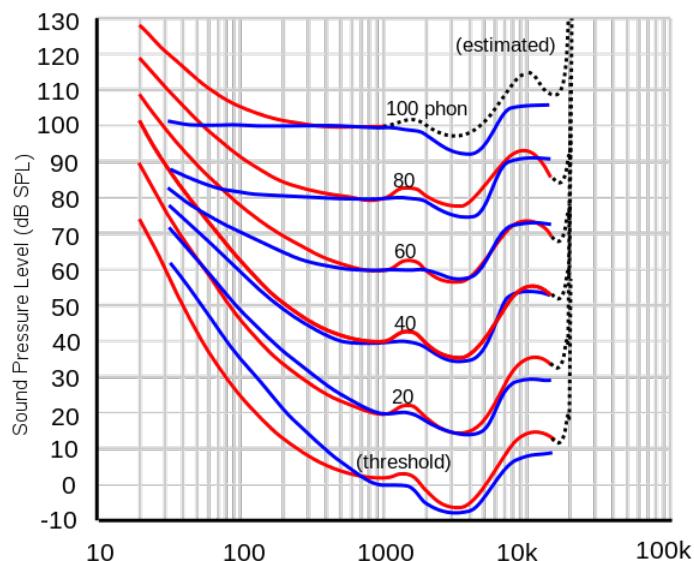
³²By David by Michelangelo - cropped from Image:Dithering algorithms.png, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=2694287>, <https://commons.wikimedia.org/w/index.php?curid=2694298> и <https://commons.wikimedia.org/w/index.php?curid=2694299>

³³Повеќе за нив може да видите во делот Психоакустика во предметот Електроакустика чии материјали се поставени на Гитлаб <https://gitlab.com/feeit-freecourseware/electroacoustics>.

³⁴"Lindos4" by Lindosland <https://commons.wikimedia.org/wiki/File:Lindos4.svg#/media/File:Lindos4.svg>



Сл. 1.12: Сива слика со 256 нивоа по пиксел (лево), реквантанизирана на две нивоа (црно и бело) без (средина) и со употреба на дитер (десно).³²

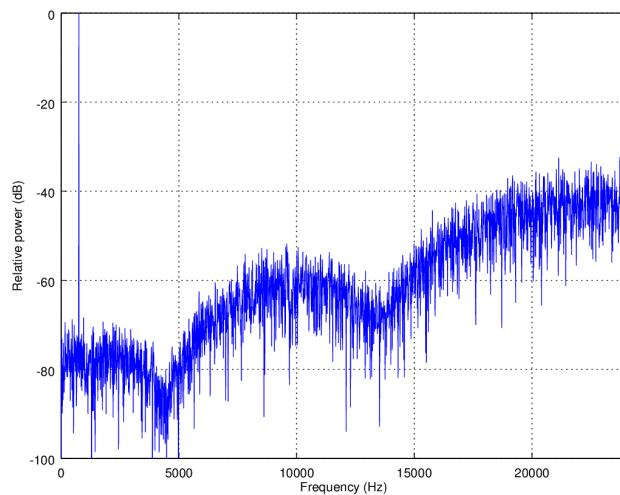


Сл. 1.13: Криви на еднаква гласност, или изофонски криви, измерени според Флечер и Мунсон (сино) и стандардизирани според ISO стандардот (црвено).³⁴

Со техниката на **обликување на шумот** може да ја менуваме фреквенцијска распределба на дитерот, што ни овозможува поголем негов дел да го распределиме надвор од фреквенцијскиот опсег во кој увото е најосетливо. Притоа севкупната моќност на шумот останува иста. Со помош на овој пристап може во еден 16-битен сигнал да се постигне однос сигнал-шум од 120 dB, а некои алгоритми овозможуваат тој на средни фреквенции да достигне дури 150 dB! Обликувањето на шумот е особено згодно при употреба на надземање примероци, каде шумот може да се потисне надвор од слушното подрачје -- над 20th kHz. Како и дитерот, и обликувањето на шумот се употребува најмногу при конверзијата на дигиталниот сигнал од поголема во помала резолуција.

Пример за употреба на обликување на шумот со дитер е прикажана на Сл. 1.14, каде истиот синусен тон е квантизиран на 4 битови со употреба на обликување на шумот. Споредено со претходниот случај, кога дитерот беше употребен без обликување, сега SNR достигнува 80 dB за фреквенции околу 4 kHz каде увото е најосетливо. Да потсетиме дека без примената на овие техники, според (1.9) SNR за 4 битна квантација изнесува само 24 dB!

³⁵By Andrew D'Addesio - Own work, CC0, <https://commons.wikimedia.org/w/index.php?curid=44975275>



Сл. 1.14: Синусен тон на фреквенција од 750 Hz од кој се земени примероци со мултипл од таа фреквенција од 48 kHz и квантизиран на 4 бита со употреба на дитер со примена на техниките за обликување на шумот.³⁵

◀ Пример Каде 64× надземање примероци во SACD стандардот, фреквенцијата на земање примероци изнесува 2,8 MHz, па шумот може да се распредели до 1,4 MHz што овозможува однос сигнал-шум од 100 dB при кодирање со 1 бит, што би требало да овозможи динамички опсег од само 6 dB.

1.3 Формати на дигитален звук

По земањето на примероци, квантацијата и кодирањето ја добиваме трансформацијата на звукот во дигитален домен. Методот со кој е извршена дигитализацијата го одредува и начинот на кој звукот ќе биде запишан со низа од 1 и 0, односно го одредува **форматот на дигиталниот звук**. Основен формат на дигитален звук е аудиото кодирано со **импулсно кодна модулација** (PCM)³⁶. PCM претставува најкавалитетниот, најсигурниот и наједноставниот начин на дигитално запишување на звучните аналогни сигнали и се користи кога не ни е пречка големината на битскиот проток генериран при дигитализација. Поради релативната едноставност и сигурност во однос на другите типови на квантација и кодирање, какви што се на пр. адаптивната PCM и линеарно–предиктивното кодирање, PCM е најраспространетиот формат за кодирање на високо квалитетен дигитален звук.

PCM кодираниот дигитален звучен сигнал на компјутерот се зачувува во вид на датотека, додека пак кај аудио-CD-то тој е запишан во вид на постојан проток, дефиниран според Црвената Книга на аудио-CD стандардот. Разликата е во тоа што на датотеките содржат воведен дел, заглавје³⁷ во кој се зачувани информации за должината на датотеката и параметрите на дигиталниот звук, како фреквенцијата на земање примероци и резолуцијата на записот, додека кај аудио-CD-то овие информации се излишни. Форматот во кој е зачувано PCM аудиото во компјутерските системи зависи од оперативниот систем кој го користи компјутерот. Windows оперативните системи го користат **WAV**³⁸ форматот, развиен од страна на Microsoft за Intel базираните компјутерски системи во 1991, врз база на општиот RIFF (Resource Interchange File Format) дефиниран за чување на разни видови на податоци. Компјутерите со Macintosh платформи го употребуваат **AIFF**³⁹ форматот на датотеки, развиен од Apple во 1988. Двата паралелни стандарда се базираат на IFF форматот за зачувување на податоци воведен од Electronic Arts во 1985, со клучна разлика

³⁶англ. *Pulse Code Modulation*.

³⁷англ. *header*.

³⁸Waveform audio format.

³⁹Audio Interchange File Format.

во редоследот на запишување на дигиталните податоци. WAV форматот најпрвин ги запишува 8^{te} бита со најмала тежина, па потоа тие со поголема, што во информатиката се нарекува „little-endian“ тип на запис. Од друга страна, AIFF форматот го прави тоа обратно, со запишување на байтите од најзначајниот кон најмалку значајниот, познато како „big-endian“.

§ Дополнително. Во WAV форматот се користат два начини на запишување на негативните примероци во аудиосигналот, во зависност од бројот на битови по примерок. Ако станува збор за звук квантизиран со 8 бита по примерок тогаш амплитудните вредностите се зачувуваат без назнака за знакот `uint8`. Најмалата вредност што може да биде запишана `0x00`, соодветствува на најнегативната вредност на сигналот, додека најголемата вредност `0xFF`, соодветствува на неговата најпозитивна вредност. Кај 16-битната квантизација, негативните вредности на сигналот се зачуваат со употреба на двоен комплемент. Со тоа првиот бит од 16^{te} го дефинира знакот.

Дигиталниот звук често се користи во негова компримирана форма, со што се постигнува намалување на побарувањата за меморија и дигитален проток при неговото зачувување и пренос. За таа цел се развиени низа на техники за компримирање кои ќе бидат разгледани во Главата 6.

Глава 2

Вовед во процесирање звучни сигнали

Со преминот на звукот во [дигитален домен](#), се отвораат вратите кон примена на различните техники на дигиталната обработка на сигналите за негова обработка.

Поради философијата на отвореност и соработка која е основа зад движењето за слободен софтвер, денес сè повеќе инженери и научници ја засноваат својата работа на платформи базирани на слободен софтвер. Повеќе за историјата, предностите и продорот на слободниот софтвер во инженерската и научно истражувачката практика може да прочитате во [Додатокот А](#).

Во духот на философијата зад слободниот софтвер, а следејќи ги светските инженерски и научни трендови, вежбите во овој предмет во целост ќе бидат изработени со слободен и отворен софтвер, поточно во слободниот програмскиот јазик [Python](#). За вовед во екосистемот на Python направен за инженерска и научна работа и за основните системски поставувања погледнете во [Додатокот В](#).

Во оваа вежба ќе се запознаеме со основните принципи на работата со звукот во дигитален домен со употреба на програмскиот јазик Python.

2.1 Вчитување на звук во Python

Вчитувањето на звук во Python може да го направиме со употреба на модулот SciPy. Да го вчитаме нашиот прв звучен запис! Стартувајте го IPython во терминалот и извршете го следниот код:

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.io import wavfile

audio_path = 'audio/'
wav_name = 'Skopsko_stereo.wav'
fs, wav = wavfile.read(audio_path + wav_name)
```

Со овој код во матрицата `wav` сме ги вчитале примероците на дигитализираниот звук сместени во датотеката `Skopsko_stereo.wav`. Променливата во која ги вчитуваме звуците во Python, во зависност од бројот на канали, може да има една или две вектор колони. Во нашиот случај станува збор за стерео датотека па соодветно променливата `wav` ќе има две вектор колони. Во `fs` ќе биде запишана фреквенцијата на земање примероци со која е запишан звукот, а бројот на битови по примерок, односно должината на дигиталниот збор, може да се види од типот на променливите во матрицата `wav`.

За да го видиме ова променливи во работниот простор може да напишеме:

```
whos
```

Variable	Type	Data/Info
fs	int	44100
np	module	<module 'numpy' from '/usr/local/lib/python3.6/site-packages/numpy/__init__.pyc'>
plt	module	<module 'matplotlib.pyplot' from '/usr/local/lib/python3.6/site-packages/matplotlib/pyplot.pyc'>
wav	ndarray	811782x2: 1623564 elems, type int16, 3247128 bytes (3 Mb)
wavfile	module	<module 'scipy.io.wavfile' from '/usr/local/lib/python3.6/site-packages/scipy/io/wavfile.pyc'>

Гледаме дека `wav` има точно две вектор колони со по 811.782 примероци, што при `fs` од 44,1 kHz е звучен сигнал со должина од 18,4 s. Ова може да го пресметаме на следниот начин:

```
wav.shape[0] / fs
18.407755102040817
```

Освен димензиите на `wav` со `whos` може да видиме и колкав мемориски простор таа зафаќа -- околу 3 MB, колку што е и голема `.wav` датотеката на дискот. Резолуцијата на квантизација е 16 битови, па затоа елементите на NumPy матрицата се од типот `int16`.

2.2 Скратување на звукот и префрање во моно

Звучниот сигнал сместен вака во матрицата `wav` може да се обработува како и сите други вектори и матрици во Python. За да го скратиме аудиосигналот на 4 s треба да внесеме:

```
wav = wav[: 4 * fs, :]
```

Постојат неколку начини да го направиме стерео аудиосигналот во моно. Наједноставно тоа може да се направи со задржување на само еден од каналите:

```
wav = wav[:, 0]
```

или:

```
wav = wav[:, 1]
```

Најправилниот начин тоа да се направи е преку усреднување на двета канали:

```
wav_mono = wav[:, 0] / 2 + wav[:, 1] / 2
```

што може да се пресмета и како:

```
wav_mono = np.sum(wav / 2, axis=1)
```

односно наједноставно со:

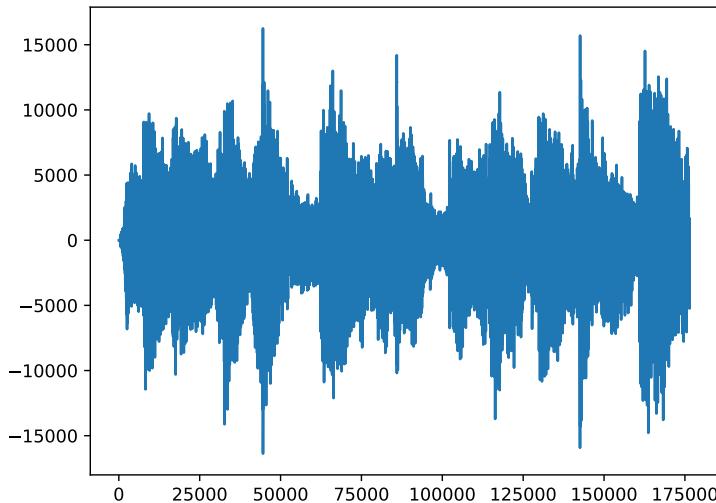
```
wav_mono = np.mean(wav, axis=1)
```

Во функциите `np.sum` и `np.mean`, вториот параметар `axis` кажува по која оска да се изврши пресметката, со 1 е избрана првата димензија -- односно по колони, т.е. по хоризонтала. Со 0 може да се избере нултата димензија - односно по редици, т.е. по вертикалa, која е и зададена автоматски.

2.3 Прикажување звук

За да го прикажеме аудиосигналот ќе го искористиме Matplotlib:

```
plt.plot(wav_mono)
plt.show()
```



Сл. 2.1: Приказ на звукот `wav_mono` како вектор од елементи од типот `int16`.

Со што ќе биде исцртан графиконот прикажан на Сл. 2.1. На овој графикон на x -оската е даден бројот на примерокот, а на y -оската неговата амплитуда. Бидејќи аудиосигналот е со 16-битна резолуција амплитудите на примероците се движат соодветно од -2^{15} до $2^{15} - 1$.

За да ги доведеме примероците во опсег од -1 до 1 , сè што треба да направиме е да го поделиме векторот `wav_mono` со 2^{15} :

```
wav_mono = wav_mono / 2**15
whos

Variable      Type      Data/Info
-----
...
wav_mono      ndarray   176400: 176400 elems, type float64, 1411200 bytes (1 Mb)
...
```

Може да забележиме дека сега елементите од векторот не се веќе од типот `int16`, ами од типот `float64`. Сега да создадеме временски вектор и да го прикажеме сигналот во време. Овој пат ќе ја искористиме опцијата `%matplotlib` за интерактивно плотирање.

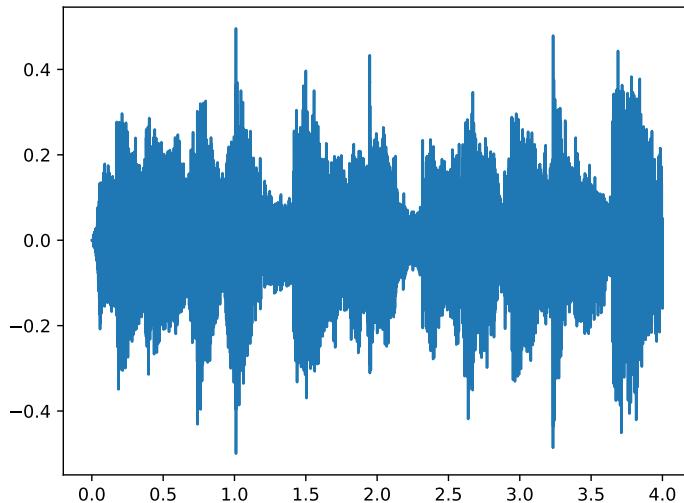
```
t = np.arange(0, wav_mono.size) / fs
%matplotlib
plt.figure()
plt.plot(t, wav_mono)
```

Со што ќе биде исцртан графиконот прикажан на Сл. 2.2.

2.4 Преслушување на аудиосигналот

Наједноставниот начин да го преслушаме аудиосигналот е првин да го запишеме како аудиозапис, а потоа да ја искористиме системската `play` наредба од софтверскиот пакет Сокс, види Додаток B.

```
wavfile.write(audio_path + 'Skopsko_mono.wav', fs, wav_mono)
!play Skopsko_mono.wav
```



Сл. 2.2: Приказ на звукот `wav_mono` како вектор од елементи од типот `float64` со време на земање на примероците на x -оската.

`Skopsko_mono.wav:`

```
File Size: 1.41M      Bit Rate: 2.82M
Encoding: F.P. PCM
Channels: 1 @ 53-bit
Samplerate: 44100Hz
Replaygain: off
Duration: 00:00:04.00

In:37.2% 00:00:01.49 [00:00:02.51] Out:65.5k [ ==|= == ]           Clip:0
```

Може да забележиме дека Python го запишал аудиозаписот со 64-битна прецизност наместо во оригиналната 16-битна. За ова да го поправиме треба да напишеме:

```
wav_16bit = np.int16(wav_mono * 2**15)
wavfile.write(audio_path + 'Skopsko_mono.wav', fs, wav_16bit)
!play Skopsko_mono.wav
```

`Skopsko_mono.wav:`

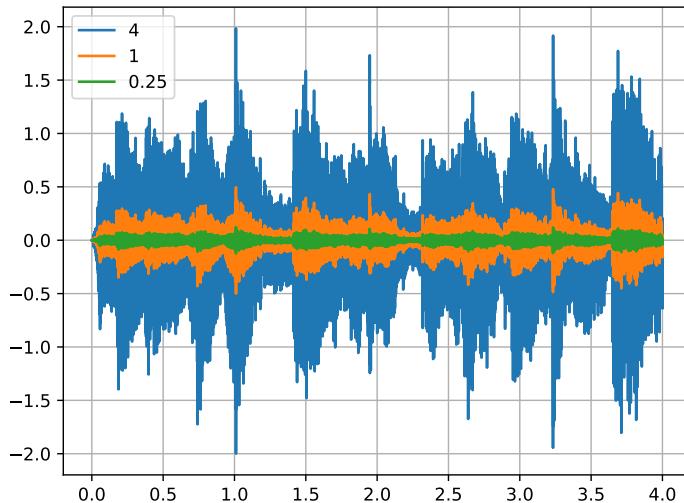
```
File Size: 353k      Bit Rate: 706k
Encoding: Signed PCM
Channels: 1 @ 16-bit
Samplerate: 44100Hz
Replaygain: off
Duration: 00:00:04.00

In:100% 00:00:04.00 [00:00:00.00] Out:176k [ ==|= == ]           Clip:0
Done.
```

2.5 Менување на амплитудата на аудиосигналот

За да го засилиме или втишаме аудиосигналот сè што треба да направиме е да го помножиме со некој коефициент за скалирање:

```
wav_loud = wav_mono * 4
wav_quiet = wav_mono * .25
```



Сл. 2.3: Приказ на аудиосигналот `wav_mono` скалиран со различни коефициенти.

```
plt.figure()
plt.plot(t, wav_loud)
plt.plot(t, wav_mono)
plt.plot(t, wav_quiet)
plt.legend([4, 1, 0.25])
plt.grid()
```

Добиениот графикон е даден на Сл. 2.3. Може да видиме дека засилениот сигнал `wav_loud` има амплитуда над дозволениот опсег од -1 до 1 . Ова значи дека при снимањето на сигналот во wav формат со 16-битна резолуција, вредностите надвор од опсегот нема да бидат запишани правилно, туку ќе бидат преклопени поради употребата на двојниот комплемент.¹ Ова може да го чуеме со `play` наредбата.

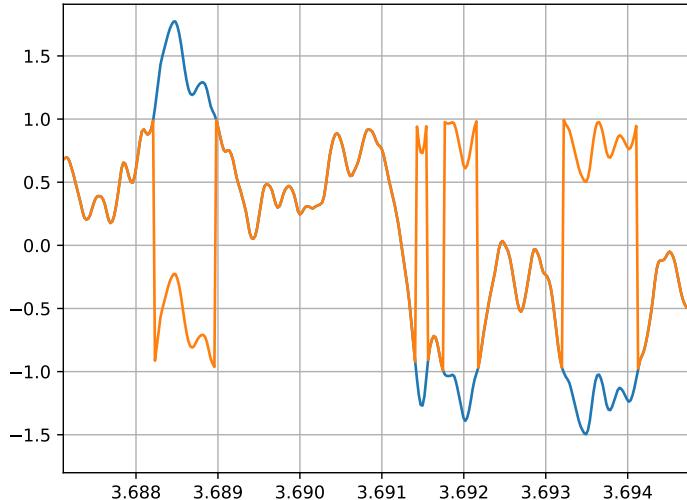
```
wav_16bit = np.int16(wav_loud * 2**15)
wavfile.write('wav_loud.wav', fs, wav_16bit)
!play wav_loud.wav
```

За да ја потврдиме оваа појава може да го вчитаме запишаниот аудиосигнал и да го исцртаме заедно со оригиналниот како на Сл. 2.4.

```
fs, wav_loud_on_disk = wavfile.read('wav_loud.wav')
t = np.arange(0, wav_loud_on_disk.size) / fs
plt.figure()
plt.plot(t, wav_loud)
plt.plot(t, wav_loud_on_disk / 2**15)
plt.grid()
```

Поврзана појава со ова е појавата на пресекување на амплитудите на звучниот сигнал ако тие го надминуваат дозволениот опсег, која се нарекува [дисторзија](#) и во некои типови на музика се користи како пожелен аудиоэффект, види Глава 5. За да ја прикажеме дисторзијата графички може да напишеме:

¹На пример, ако користиме 16-битна резолуција, најголемата позитивна вредност е $2^{15} - 1 = 32767$, која се запишува како 0111 1111 1111 1111, со тоа што MSB битот е 0 што е ознака за позитивни броеви запишани во двоен комплемент. Ако сигналот има вредности поголеми од 1 (всушност поголеми од $1 - 2^{-15}$) за едно квантизацијско ниво (2^{-15}) тие ќе бидат запишани со следната поголема вредност која е 1000 0000 0000 0000, што во двоен комплемент претставува -32768.



Сл. 2.4: Приказ на засилениот аудиосигнал и начинот на којшто Python го запиша на дискот.

```
wav_dist = wav_loud.copy()
wav_dist[wav_dist > 1] = 1
wav_dist[wav_dist < -1] = -1
plt.figure()
plt.plot(t, wav_loud)
plt.plot(t, wav_dist)
plt.grid()
```

Копирањето во првата линија создава нова низа, односно објект, со иста содржина како `wav_loud`, кон кој ќе покажува новата променлива `wav_dist`. Во спротивно, новата променлива ќе покажува кон истиот објект кон кој покажува `wav_loud`, па промените кои следат ќе бидат направени врз оригиналната низа. Со приближување може да го добиеме приказот на Сл. 2.5.

2.6 Нормализација на аудиосигналот

Во дигиталното процесирање на звукот, поради ограничениот опсег во којшто може да биде звучниот сигнал, од посебно значење е процесот на **нормализација**. Под нормализација се подразбира доведување на максималната амплитуда на аудиосигналот A_{max} на ниво зададено со:

$$L = 20 \log \frac{A_{max}}{1} [\text{dBFS}], \quad (2.1)$$

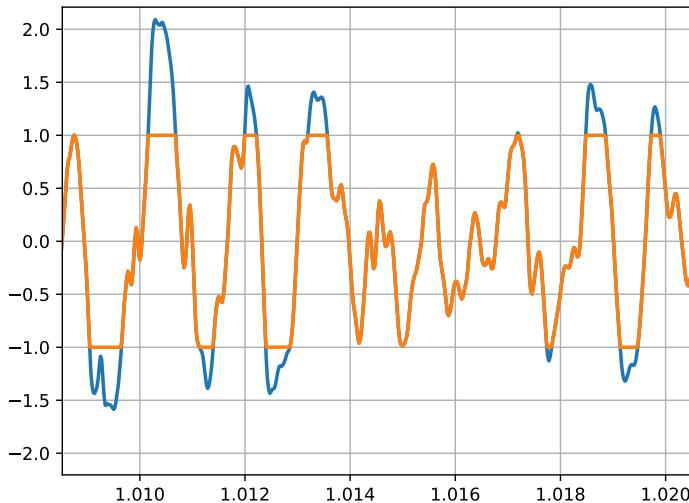
каде како референтно е земено максималното ниво во дигитален домен 1. Ова дигитално ниво на звукот се изразува во dBFS, што е кратенка од dB од полна скала². Од (2.1) следува дека за максималната дозволена амплитуда аудиосигналот има ниво од 0 dBFS. Во праксата вообичаено звучните сигнали се нормализираат на амплитуда помала од 1, за при нивното понатамошно процесирање или пренос да не дојде до дисторзија.³

Да напишеме функција за нормализација:

```
def normalize(wav_in, level):
    amp_new = 10**(level / 20)
    amp_max = np.max(np.abs(wav_in))
```

²На англиски *Full Scale*.

³Оваа амплитудна маргина која што се остава до дозволениот максимум на англиски се нарекува *headroom*.



Сл. 2.5: Приказ на дисторзија во аудиосигналот `wav_loud`.

```
return amp_new * wav_in / amp_max
```

која може да ја употребиме за нормализација на звучните сигнали:

```
wav_mono_0dB = normalize(wav_mono, 0)
wav_mono_3dB = normalize(wav_mono, -3)
wav_mono.max()
wav_mono_0dB.max()
wav_mono_3dB.max()
```

2.7 Генерирање на звук во Python

За да генерираме еден простопериодичен синусен тон на фреквенција од 200 Hz ќе напишеме:

```
sound = np.sin(2 * np.pi * 200 * t)
plt.plot(sound)
wavfile.write('sin.wav', fs, np.int16(sound * 2**15))
play sin.wav
```

Овој код ќе го поместиме во скрипта `make_sound.py` која ќе може да ја извршуваме и директно од терминал. Отворете ново јазиче во терминалот со притискање на `ctrl-shift-t` и стартувајте го стандардниот едитор за вашата дистрибуција. За Ubuntu десктоп средината тоа е `Gedit`⁴:

```
$ pluma make_sound.py &
```

Во скриптата внесете го следниот код:

```
#!/usr/bin/env python3
import numpy as np
from scipy.io import wavfile
import sys
import os

# користејќи го модулот sys можеме на нашата скрипта да и пренесеме
# влезни параметри преку командната линија
```

⁴За Plasma тоа е `kate`, за Xfce тоа е `mousepad` ...

```

print('sys.argv : ', sys.argv)
f = int(sys.argv[1])

fs = 44100
t = np.arange(0, 4 * fs) / fs
sound = np.sin(2 * np.pi * f * t)
wavfile.write('audio/sin.wav', fs, np.int16(sound * 2**15))
os.system('play audio/sin.wav')

```

и повикајте ја од IPython⁵:

```
%run make_sound.py 500
```

Новосоздадената Python скрипта може да ја извршиме и директно од командната линија со:

```
$ python ./make_sound.py 500
```

§ Дополнително. Бидејќи во првата линија од скриптата со синтаксата `#!`, на англиски позната и како *shebang*, кажуваме со кој интерпретер треба да биде извршен кодот, што е независно од екstenзијата на датотеката која може да биде и целосно изоставена.

Причината зашто во нашата скрипта имаме напишано `!/bin/env python3`` наместо `!/bin/python`` затоа што сакаме да ја извршиме скриптата со Python интерпретерот во виртуелната средина, а не системската Python средина во која ги немаме инсталерирано пакетите NumPy и SciPy. Всушност, при извршување на секоја датотека во Linux, шелот го извршува процесот во нов шел кој го создава само за таа причина. Тоа што `env` го прави е ги проследува системските променливи, меѓу која и `PATH` на новиот шел, со што првиот Python што ќе биде најден ќе биде оној од виртуелната средина.

Конечно, за да ја извршиме скриптата во терминал сега може да напишеме:

```
$ ./make_sound.py 500
```

Но, за ова да работи треба најпрвин да ги смениме пермисиите на датотеката, односно да го сетираме знаменцето за извршување `x` за нашиот корисник `u` со наредбата:

```
$ chmod u+x make_sound.py
```

✓ Задача за дома. Користејќи ја скриптата `make_sound.py` тестирајте ги границите на вашиот слух!

⁵За да го смените јазичето искористете ја комбинацијата `text Ctrl + PgUp/PgDwn`.

Глава 3

Фреквенциски спектар на звучните сигнали

Фреквенцискиот спектар на звучните сигнали е клучен во нивното поимање од страна на човекот. Спектарот на звучните сигнали е клучен и при нивната анализа, синтеза и обработка во дигиталните аудиосистеми. Со негова помош можеме да ги видиме карактеристиките на звучниот сигнал кои ги слушаме, а кои не се јасно видливи во неговиот временски облик. За пресметка на спектарот на аудиосигналите ќе се послужиме со **Фурьеовата трансформација** и нејзината временски определена форма **Фурьеова трансформација со прозорци (STFT)**¹. Ќе видиме како STFT на звучниот сигнал води до една прегледна форма за претставување на неговите временско-спектрални, односно спектро-темпорални особини – спектрограмот.

3.1 Основи на Фурьеовата анализа

За потребите на овој курс, во ова поглавје ќе направиме краток преглед на основите на Фурьеовата анализа. Притоа ќе се задржиме само до особините и деталите кои се релевантни за дигиталното процесирање на аудиосигналите. За подетален преглед погледнете ја книгата **Богданова (1997)**, по која се работи на предметот **Основи на дигитално процесирање на сигнали** на додипломските на **ФЕИТ**.



§ Дополнително. Францускиот математичар и физичар **Жан-Батист Жозеф Фурье**² (1768–1830) анализирајќи го ширењето на топлината, во 1822 за првпат постулирал дека која било функција може да биде претставена преку серија на синусоиди. Тој бил син на кројач и останал без својот татко на 9 годишна возраст. За време на француската револуција бил член на локалниот револуционерен комитет, за што и бил во затвор. Го придружувал Наполеон Бонапарта во неговиот поход во Египет во 1798, а по враќањето во Франција во 1801 станува претседател на општината Изер во Гренобл, каде започнува да прави експерименти со топлината.

¹анг. *Short-Time Fourier Transform*.

²Wikipedia -- Joseph Fourier https://en.wikipedia.org/wiki/Joseph_Fourier. Слика од Louis-Léopold Boilly - <https://www.gettyimages.com.au/license/169251384>, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=3308441>

Табела 3.1: Позначајни својства на Фурьеовата трансформација.

Својство	Временски домен	Фурьеов домен
Линеарност	$ax(t) + bg(t)$	$aX(\omega) + bG(\omega)$
Поместување	$x(t - t_0)$	$e^{-j\omega_0 t} X(\omega)$
Модулација	$e^{-j\omega_0 t} x(t)$	$X(\omega - \omega_0)$
Конволуција	$x(t) * g(t) = \int_{-\infty}^{\infty} x(\tau)g(t - \tau) d\tau$	$X(\omega)G(\omega)$
Мултипликација	$x(t)g(t)$	$\frac{1}{2\pi} X(\omega) * G(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\Omega)G(\omega - \Omega) d\Omega$

Фурьеов интеграл

Фурьеовата трансформација (FT)³ на сигналот $x(t)$ е дефинирана со интегралот:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t} dt, \quad \omega \in (-\infty, \infty), \quad (3.1)$$

каде со $\omega = 2\pi f$ е означена кружната фреквенција.

Инверзната Фурьеова трансформација (IFT)⁴ е дефинирана со интегралот:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega)e^{j\omega t} d\omega, \quad t \in (-\infty, \infty). \quad (3.2)$$

$X(\omega)$ се нарекува **фреквенциски спектар** на сигналот $x(t)$ и претставува комплексна функција од ω која може да се запише како:

$$X(\omega) = |X(\omega)|e^{j\angle X(\omega)} = A(\omega)e^{j\varphi(\omega)}, \quad (3.3)$$

каде $A(\omega)$ претставува **амплитуден спектар**, а $\varphi(\omega)$ **фазен спектар** на сигналот $x(t)$. Притоа, ако сигналот е реална функција од t тогаш амплитудниот спектар е парна функција од ω , а фазниот е непарна функција од ω . Бидејќи интеграл од непарна функција во интервал симетричен околу координатниот почеток е 0, тогаш од (3.2), користејќи ја **Ојлеровата формула**, добиваме:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} A(\omega)e^{j\varphi(\omega)}e^{j\omega t} d\omega, \quad (3.4)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} A(\omega) [\cos(\omega t + \varphi(\omega)) + j \sin(\omega t + \varphi(\omega))] d\omega, \quad (3.5)$$

$$= \frac{1}{\pi} \int_0^{\infty} A(\omega) \cos(\omega t + \varphi(\omega)) d\omega. \quad (3.6)$$

Основните својства на Фурьеовата трансформација се дадени во **Табелата 3.1**.

Сигналот $x(t)$ и неговата Фурьеова трансформација $X(\omega)$ чинат **Фурьеов пар** што се означува како $x(t) \leftrightarrow X(\omega)$. За Фурьеовата трансформација се користи и симболот \mathcal{F} , односно пишуваме $\mathcal{F}\{x(t)\} = X(\omega)$, или за IFT $\mathcal{F}^{-1}\{X(\omega)\} = x(t)$.

Во **Табелата 3.2** се дадени некои позначајни Фурьеови парови.

³англ. *Fourier Transform*.

⁴англ. *Inverse Fourier Transform*.

Табела 3.2: Позначајни Фуриеови парови.

$x(t)$	$X(\omega)$
$\delta(t)$	1
1	$2\pi\delta(\omega)$
$\Pi(t) = \begin{cases} 1 & t < T \\ 0 & t > T \end{cases}$	$2T \frac{\sin(T\omega)}{T\omega}$
$\frac{\sin(\Omega t)}{\pi t}$	$\Pi(\omega) = \begin{cases} 1 & \omega < \Omega \\ 0 & \omega > \Omega \end{cases}$
$\cos(\omega_0 t)$	$\pi [\delta(\omega + \omega_0) + \delta(\omega - \omega_0)]$
$\sin(\omega_0 t)$	$j\pi [\delta(\omega + \omega_0) - \delta(\omega - \omega_0)]$

Енергијата на сигналот може да се пресмета преку неговиот спектар со употреба на теоремата на Парсевал⁵:

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |X(\omega)|^2 d\omega = \int_{-\infty}^{\infty} |X(2\pi ft)|^2 df \quad (3.7)$$

* Важно! Во Табелата 3.2 забележете дека Фуриеовата трансформација на 1 е $2\pi\delta(\omega)$, додека на $\cos(\omega_0 t)$ е $\pi [\delta(\omega + \omega_0) + \delta(\omega - \omega_0)]$. Ова е во склад со теоремата на Парсевал дека енергијата во спектарот на косинусот помножена со коефициент $1/2\pi$ треба да е еднаква со онаа во временскиот сигнал. Поради тоа двата огледални Диракови импулси имаат двојно помала амплитуда, што не е случај кај константниот сигнал за кој имаме единствен Дираков импулс на фреквенција 0.

Фуриев ред

Ако функцијата $x(t)$ е периодична со периода T , односно има основна фреквенција $f_0 = 1/T$, тогаш важи:

$$x(t) \equiv x(t + T), \quad \forall t. \quad (3.8)$$

Ако со $\hat{x}(t)$ го означиме сегментот на $x(t)$ во основниот интервал $t \in [-T/2, T/2]$:

$$\hat{x}(t) = x(t) \cdot \Pi(t), \quad (3.9)$$

каде:

$$\Pi(t) = \begin{cases} 1, & |t| \leq \frac{T}{2}, \\ 0, & |t| > \frac{T}{2}. \end{cases}, \quad (3.10)$$

тогаш можеме да ја изразиме $x(t)$ како сума од вакви функции $\hat{x}(t)$ поместени за мултикли од T :

$$x(t) = \sum_{r=-\infty}^{\infty} \hat{x}(t - rT). \quad (3.11)$$

Во тој случај во Фуриев домен, функцијата нема да има ненулти вредности за сите вредности на кружната фреквенција ω , ами ќе биде еднаква на сумата од експоненцијални функции, односно комплексни синусоиди со дискретни фреквенции, која се нарекува и **Фуриев ред**:

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{jk\frac{2\pi}{T}t} = \sum_{k=-\infty}^{\infty} c_k e^{jk\omega_0 t}, \quad (3.12)$$

⁵Wikipedia -- Parseval's Theorem https://en.wikipedia.org/wiki/Parseval%27s_theorem

каде ω_0 е основната кружна фреквенција на сигналот, дадена со:

$$\omega_0 = 2\pi f_0 = \frac{2\pi}{T}. \quad (3.13)$$

Ова значи дека фреквенциите на комплексните синусоиди претставуваат мултикли од основната кружна фреквенција на сигналот ω_0 . Овие мултикли уште се нарекуваат и **хармоници** или **виши хармоници**, а f_0 и ω_0 се нарекува **основен хармоник**. Притоа кофициентите c_k кои претставуваат амплитуди на комплексните синусоиди можат да се добијат преку:

$$c_k = \frac{1}{T} \int_{-T/2}^{T/2} x(t) e^{-jk\frac{2\pi}{T}t} dt. \quad (3.14)$$

Фреквенцискиот спектар на ваков периодичен сигнал $x(t)$ претставува сума од поместени евидистантни Диракови импулси $\delta(\omega)$:

$$X(\omega) = 2\pi \sum_{k=-\infty}^{\infty} c_k \delta(\omega - k\omega_0). \quad (3.15)$$

\mathcal{Z} трансформација

За да видиме како Фуреовата анализа може да се примени врз дискретни сигнали ќе ја воведеме \mathcal{Z} трансформацијата (Rabiner and Schafer, 1978) дефинирана со:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n}, \quad (3.16)$$

каде $x[n]$ е дискретна верзија од континуираната функција $x(t)$:

$$x[n] = x[nT_s], \quad (3.17)$$

$$T_s = \frac{1}{f_s}. \quad (3.18)$$

Тука, f_s е фреквенцијата на земање примероци, а T_s е периодата на земање примероци.

Инверзната \mathcal{Z} трансформација е дефинирана со:

$$x[n] = \frac{1}{2\pi j} \oint_C X(z) z^{n-1} dz, \quad (3.19)$$

каде C е затворена контура во \mathcal{Z} рамнината која го вклучува центарот, а се наоѓа во пределот на конвергенција на трансформацијата $z \in (R_1, R_2)$. Позначајни особини на \mathcal{Z} трансформацијата се дадени во Табелата 3.3, а позначајни \mathcal{Z} трансформации во Табелата 3.4.

Дискретна Фуреова трансформација

Фуреовата трансформација на дискретен сигнал може да се добие од равенката за \mathcal{Z} трансформацијата (3.16) со избор на јадрото $z = e^{-j\omega T_s}$, што се поедноставува на $z = e^{-j\omega}$ за $T_s = 1$. Со тоа \mathcal{Z} трансформацијата се ограничува на единечната кружница:

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n}, \quad (3.20)$$

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega) e^{j\omega n} d\omega. \quad (3.21)$$

Табела 3.3: Позначајни својства на \mathcal{Z} трансформација.

Својство	Временски домен	\mathcal{Z} домен
Линеарност	$ax_1(t) + bx_2(t)$	$aX_1(z) + bX_2(z)$
Поместување	$x[n - n_0]$	$z^{n_0} X(z)$
Конволуција	$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$	$X(z)H(z)$
Мултипликација	$x[n]\Pi[n]$	$\frac{1}{2\pi j} \oint_C X(\nu)W(z/\nu)\nu^{-1}d\nu$

Табела 3.4: Позначајни \mathcal{Z} парови.

$x[n]$	$X(z)$
$\delta[n - n_0]$	z^{-n_0}
$\Pi[n] = \begin{cases} 1, & 0 \geq n < N, \\ 0, & \text{поинаку.} \end{cases}$	$\sum_{n=0}^{N-1} z^{-n} = \frac{1 - z^{-N}}{1 - z^{-1}}$
$a^n u[n]$	$\frac{1}{1 - z^{-1}}, \quad a < z $

Во овој случај, ја ограничуваме \mathcal{Z} трансформацијата на единечната кружница во z -рамнината опишана со $e^{-j\omega} = \cos(\omega) + j \sin(\omega)$, а дигиталната кружна фреквенција ω има интерпретација на агол во оваа рамнина. Со други зборови Фурьеовата трансформација на дискретниот сигнал претставува специјален случај на \mathcal{Z} трансформацијата.

Бидејќи новото јадро $e^{-j\omega}$ е периодична функција со периода 2π , следува дека фреквенцискиот спектар на дискретните сигнали е исто така периодичен со 2π во однос на кружната фреквенција ω . За произвилно T_s , спектарот има периода $2\pi/T_s$ во однос на кружната фреквенција ω , односно периодата е $1/T_s = f_s$ во однос на фреквенцијата f , како што беше дискутирано во Главата 1.1.

* Важно! Фурьеовата трансформација $X(\omega)$ на дискретниот сигнал $x[n]$ е сеуште континуирана функција од ω !

Како што беше случај во аналоген домен, ако еден дискретен сигнал е периодичен со периода N , односно:

$$\tilde{x}[n] = \tilde{x}[n + N], \quad -\infty < n < \infty, \quad (3.22)$$

тогаш $\tilde{x}[n]$ во Фурьеов домен ќе биде претставена како дискретна сума од синусоиди:

$$\tilde{X}[k] = \sum_{n=0}^{N-1} \tilde{x}[n] e^{-jk\frac{2\pi}{N}n}, \quad (3.23)$$

$$\tilde{x}[n] = \frac{1}{N} \sum_{k=0}^{N-1} \tilde{X}[k] e^{jk\frac{2\pi}{N}n}. \quad (3.24)$$

Ако сега ја земеме Фурьеовата трансформација на дискретниот сигнал $x[n]$ дадена во (3.20), која претставува \mathcal{Z} трансформација на сигналот $x[n]$ долж единечната кружница, и од истата земаме примероци на еквидистантни фреквенции ω_k дадени со:

$$\omega_k = k \frac{2\pi}{N}, \quad (3.25)$$

тогаш ќе го добијеме изразот за дискретната Фурьеова трансформација (DFT)⁶:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-jk\frac{2\pi}{N}n}, \quad (3.26)$$

кој е еквивалентен со (3.23)!. Ова значи дека семплирирајм спектар што го добиваме со (3.26) соодветствува на сигнал $\tilde{x}[n]$ која претставува периодична верзија од сигналот $x[n]$ со периода N , односно важи:

$$x[n] = \tilde{x}[n] \cdot \Pi[n], \quad (3.27)$$

каде

$$\Pi[n] = \begin{cases} 1, & 0 \leq n < N, \\ 0, & \text{поинаку.} \end{cases} \quad (3.28)$$

Со други зборови, дискретниот спектар на еден дискретен сигнал $x[n]$ добиен со DFT претставува всушност спектар на периодичната верзија на сигналот во која тој се повторува бесконечно со периода еднаква на неговата должина N .

* Пример. Да земеме дека должината на дискретниот сигнал е $N = 4$, тогаш спектарот на сигналот добиен со DFT ќе биде пресметан за следните вредности на кружната фреквенција ω : $\omega_0 = 0$, $\omega_1 = \pi/2$, $\omega_2 = \pi$ и $\omega_3 = 3\pi/2$. Поради симетричноста на амплитудниот спектар за реални сигнали, што се сведува на симетричност на единечната кружница во однос на реалната оска во z рамнината, примероците ω_1 и ω_3 се огледални слики и се идентични. Поради тоа што тие не носат дополнителна информација, вообичаена пракса е огледалните слики да се исфрлат, при што мора да се удвојат соодветните примероци кои остануваат за да се зачува валидноста на теоремата на Парсевал. По исфрлањето, последниот примерок од спектарот ќе биде за кружна фреквенција $\omega_3 = \pi$, односно, за произвилно T_s , за фреквенција $f = f_s/2$. Така, конечната скратена верзија на DFT спектарот би се состоела од: $X(0)$, $2X(\omega_1)$ и $X(\pi)$.

За произвилно T_s примероците ги земаме во кружните фреквенции:

$$\omega_k = k \frac{2\pi}{NT_s}, \quad (3.29)$$

односно:

$$f_k = k \frac{1}{NT_s} = k \frac{f_s}{N}. \quad (3.30)$$

Резолуцијата на спектарот добиен со DFT:

$$\Delta\omega = \frac{2\pi}{N}, \quad (3.31)$$

е одредена од должината на сигналот N . За произвилно T_s таа ќе биде:

$$\Delta\omega = \frac{2\pi}{NT_s}, \quad (3.32)$$

односно:

$$\Delta f = \frac{f_s}{N}. \quad (3.33)$$

За зголемување на резолуцијата на спектарот добиен со DFT постојат две методи: *i*) да се земат повеќе примероци на сигналот N , и *ii*) да се додадат N_0 нули на крајот од сигналот. Последново

⁶анг. *Discrete Fourier Transform*

кеје ја зголеми резолуцијата на земање примероци во јадрото во (3.34), без да ги смени границите на сумата:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-jk \frac{2\pi}{N+N_0} n}. \quad (3.34)$$

Сепак, добиената резолуција нема да донесе нова информација во DFT спектарот, туку само ќе генерира интерполирана верзија на оригиналниот спектар со N точки.

Соодветно **инверзната дискретна Фуриеова трансформација (IDFT)**⁷ е дадена со изразот:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{jk \frac{2\pi}{N} n}. \quad (3.35)$$

За пресметка на DFT во праксата се користат алгоритми за нејзино брзо пресметување чија пресметковна сложеност е пропорционална со $N \log N$, кои се нарекуваат **брза Фуриеова трансформација (FFT)**(Богданова, 1997).⁸ Бидејќи FFT работи брзо само за должини на сигналот од $N = 2^M$, вообичаена пракса е додавањето нули на сигналот за да се задоволи овој услов.

Проценка на амплитудата на сигналот од неговиот спектар

За да видиме колкава е амплитудата на FT спектарот на еден дискретен сигнал ќе го пресметаме спектарот на една простопериодична низа $x[n]$ со кружна фреквенција ω_0 :

$$x[n] = A \cos[\omega_0 n] = A \cos[2\pi f_0 n]. \quad (3.36)$$

Спектарот на овој сигнал е даден со равенството:

$$X(\omega) = \frac{A}{2} 2\pi (\delta(\omega + \omega_0) + \delta(\omega - \omega_0)). \quad (3.37)$$

Ако земеме отсечок на овој сигнал со должина од N примероци и ја пресметаме континуираната FT, тоа е еквивалентно на пресметка на FT врз производот на $x[n]$ со правоаголен прозорец $w[n] = \Pi[n]$ со траење $-N/2 \leq n < N/2$. Според својството на Фуриевата трансформација според кое множењето на два сигнали во временски домен претставува нивна конволуција во Фуриев домен, имаме:

$$\hat{X}(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\Omega) W(\omega - \Omega) d\Omega \quad (3.38)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{A}{2} 2\pi (\delta(\Omega + \omega_0) + \delta(\Omega - \omega_0)) W(\omega - \Omega) d\Omega \quad (3.39)$$

$$= \frac{A}{2} (W(\omega + \omega_0) + W(\omega - \omega_0)), \quad (3.40)$$

каде $W(\omega)$ е FT на прозорецот дадена со:

$$W(\omega) = \frac{\sin(N \frac{\omega}{2})}{\sin(\frac{\omega}{2})}. \quad (3.41)$$

Според Лопиталовото правило⁹, кое вели дека во случаи кога за гранична вредност на функција се добива неопределен израз како $0/0$ или ∞/∞ , тогаш гранична вредност постои и таа се наоѓа преку изводите на двете функции, односно:

$$W(0) = \lim_{\omega \rightarrow 0} \frac{\sin(N \frac{\omega}{2})}{\sin \frac{\omega}{2}} = \lim_{\omega \rightarrow 0} \frac{\sin'(N \frac{\omega}{2})}{\sin' \frac{\omega}{2}} = \lim_{\omega \rightarrow 0} \frac{N \cos(N \frac{\omega}{2})}{\cos \frac{\omega}{2}} = N \quad (3.42)$$

⁷анг. *Inverse Discrete Fourier Transform*

⁸Wikipedia -- Fast Fourier transform (FFT) https://en.wikipedia.org/wiki/Fast_Fourier_transform

⁹Википедија -- Лопиталово правило https://mk.wikipedia.org/wiki/Лопиталово_правило

Бидејќи амплитудата на спектарот на правоаголниот прозорец за $\omega \neq 0$ е мала во споредба со N за $\omega = 0$, спектарот на $\hat{x}[n]$ за фреквенција ω_0 ќе биде:

$$\hat{X}(\omega_0) = \frac{A}{2} (W(2\omega_0) + W(0)) \approx \frac{A}{2} N \quad (3.43)$$

3.2 Анализа на спектарот на звучните сигнали

Спектар на простопериодичен звук

Да го пресметаме и прикажеме спектарот на еден простопериодичен, односно синусоиден, тон ќе го искористиме звукот генериран од звучната вилушка, којшто е еден од ретките природни звуци со простопериодична природа. Снимка од овој сигнал имаме во датотеката `viluska.wav` кој е дел од аудиозаписите од предметот [Електроакустика](#)¹⁰, но го имаме и во проектната папка на предметов.

За нашата понатамошна работа ќе ги искористиме можностите на развојната средина Spyder, види [Додаток В](#). За почеток, датотеката која автоматски се отвора со стартирање на Spyder да ја снимиме под името `vezba02_spekar.py` во папката `da`. Во него да ги увеземе основните модули и да го вчитаме, преслушаме и прикажеме аудиосигналот.

```
"""
Дигитално процесирање звук
Вежба 2 - спектар
Created on Wed Mar 23 23:21:08 2016
@author: da
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
import os

audio_path = 'audio/'
wav_name = 'viluska.wav'
fs, wav = wavfile.read(audio_path + wav_name)
os.system('play ' + audio_path + wav_name)

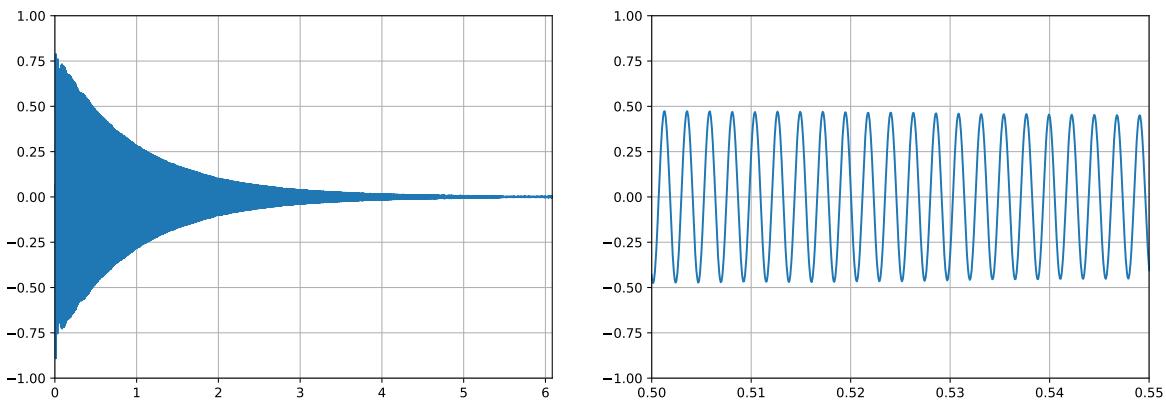
wav = wav / 2**15
n = wav.size
t = np.arange(0, n) / fs

plt.figure(figsize=(15, 5))
plt.subplot(121) # 1X2 графици, прв графикон
plt.plot(t, wav)
plt.grid()
plt.axis([0, t[-1], -1, 1]) # [xmin, xmax, ymin, ymax]
plt.subplot(122) # 1X2 графици, втор графикон
plt.plot(t, wav)
plt.grid()
plt.axis([0.5, 0.55, -1, 1]) # [xmin, xmax, ymin, ymax]
```

Графиците кои ќе ги генерира дадениот код се прикажани на Сл. 3.1

§ Дополнително. При првото извршување на Python скрипта во едиторот на Spyder треба да одберете каде да биде извршен кодот. Треба да ја одберете првата опција: `Execute in current Python or IPython console`. На овој начин сите променливи генериирани во сегашниот работен простор можат да бидат искористени од кодот и ќе бидат на располагање и по завршувањето на скриптата.

¹⁰Материјалите за предметот Електроакустика на ФЕИТ може да ги најдете на <https://gitlab.com/feeit-freecourseware/electroacoustics>



Сл. 3.1: Временски облик на аудиосигналот генериран од звучна вилушка.

§ Дополнително. Графиците во Spyder по дифолт ќе бидат исцртани во рамките на панелот за плотирање. Ова е згодно за работа со голем број на графици. Втората опција е тие да бидат исцртани во посебен прозорец, што овозможува интерактивност од типот на понатамошно исцртување на нови содржини на веќе генериран графикон, како и можност за зумирање и анализа на деталите од прикажаното. За да ја активираме втората опција треба да напишеме `%matplotlib` во IPython конзолата. Истата можеме да ја направиме дифолтна со потребните поставувања во Tools > Preferences > IPython console > Graphics > Backend : Automatic .

За да го пресметаме спектарот на овој звучен сигнал ќе ја искористиме имплементацијата на FFT алгоритамот во SciPy модулот `fftpack`, кој содржи и низа други функции за Фурьеова анализа. Во нашиот код ќе ја додадеме синтаксата `# %%` што во Spyder претставува нова ќелија, а со тоа овој дел од скриптата ќе може едноставно да биде извршен без да се извршува целиот код со притискање `Ctrl + Enter`.

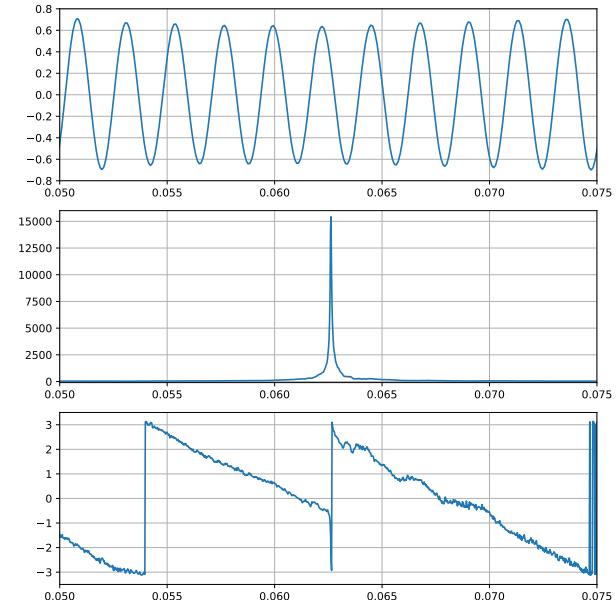
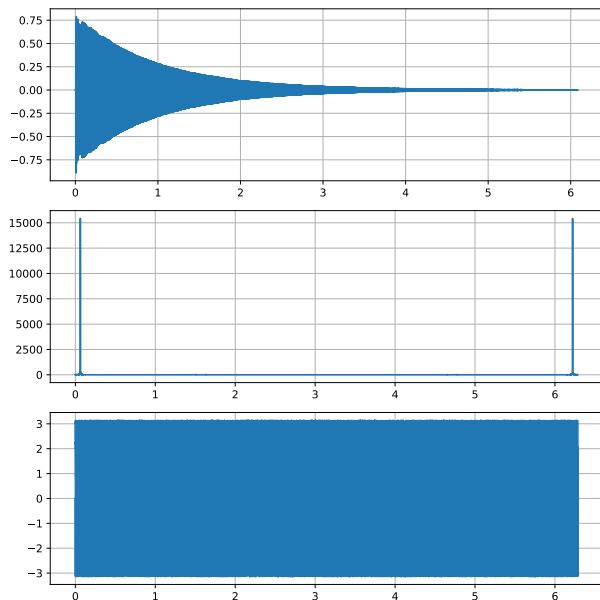
```
# %% пресметај го спектарот на аудиосигналот
from scipy import fftpack as fp

wav_fft = fp.fft(wav)
wav_amp = np.abs(wav_fft)
wav_ph = np.angle(wav_fft)
w = np.arange(0, 2*np.pi, 2*np.pi / n)

plt.figure()
plt.subplot(311)
plt.plot(t, wav)
plt.grid()
plt.subplot(312)
plt.plot(w, wav_amp)
plt.grid()
plt.subplot(313)
plt.plot(w, wav_ph)
plt.grid()
```

Графиците добиени со овој код се прикажани на Сл. 3.2. Може да забележиме дека амплитудниот и фазниот спектар се движат од 0 до 2π , а амплитудата на вториот оди од $-\pi$ до π . Вообично овие два спектри:

- се прикажуваат во однос на фреквенција во Hz,
- амплитудниот спектар вообичаено се изразува во dB, што е поблиску до начинот на кој човекот го поима звукот, и
- фазниот спектар нужно се одмотува од опсегот $-\pi$ до π и покрај неговата периодичност за да се воочи подобро неговата динамика.



Сл. 3.2: Амплитуден (средина) и фазен (долу) спектар на аудиосигналот генериран од звучна вилушка (горе).

Уште повеќе, поради тоа што за реални сигнали, какви што се аудиосignalите, половина од спектарот е огледална слика на првата половина, таа може да ја занемариме. Конечно, за побрза работа на FFT должината на влезниот сигнал треба да биде степен од 2. За таа цел, сигналот вообичаено се дополнува со нули до следната поголема должина која е степен од 2, што може да се пресмета на следниот начин:

$$N_{FFT} = 2^x, \quad (3.44)$$

каде:

$$x = \lceil \log_2 N \rceil \quad (3.45)$$

а N е должината на сигналот. Сите овие работи може да ги направиме со следниот код:

```
# %% пресметај го спектарот на аудиосигналот
from scipy import fftpack as fp

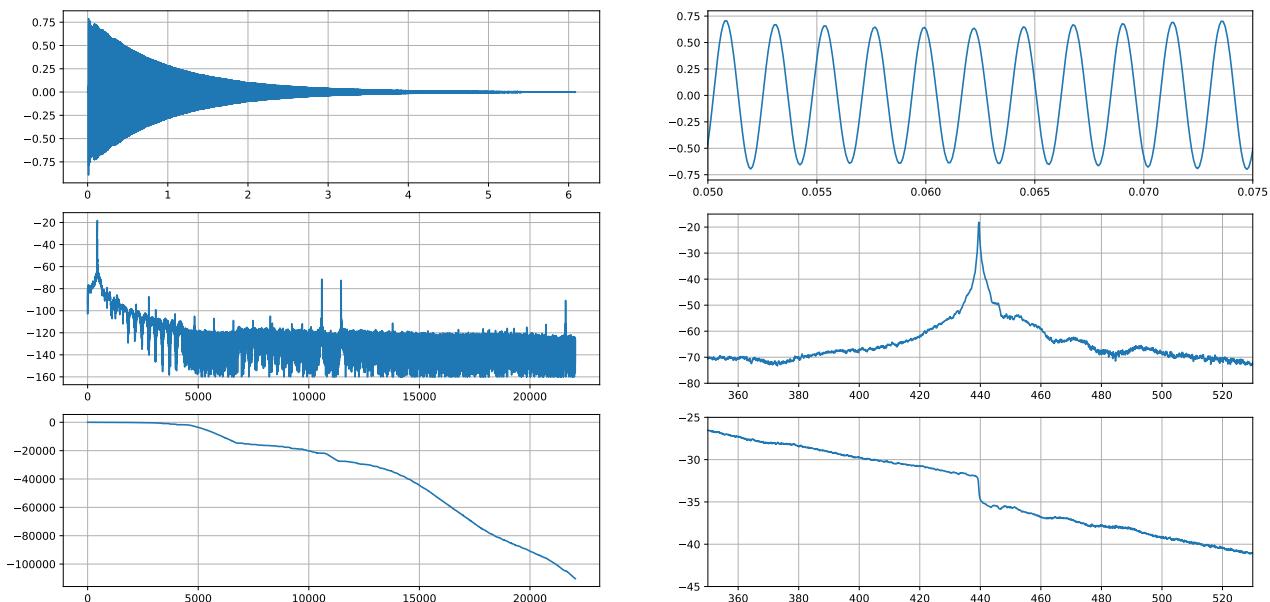
x = np.ceil(np.log2(n))
n_fft = int(2**x)
n_keep = n_fft // 2 + 1
wav_fft = fp.fft(wav, n_fft)
wav_fft = wav_fft[0: n_keep]
wav_fft = wav_fft / n
wav_fft[1: -1] = wav_fft[1: -1] * 2

wav_amp = np.abs(wav_fft)
eps = 1e-8
wav_amp[wav_amp < eps] = eps
wav_amp = 20 * np.log10(wav_amp)

wav_ph = np.angle(wav_fft)
wav_ph = np.unwrap(wav_ph)

f = np.linspace(0, fs/2, n_keep)
```

Притоа сме направиле скалирање на FFT коефициентите со должината на сигналот N , а енергијата на коефициентите кои сме ги отфрлиле сме ја додале на нивните огледални слики, со што сме направиле амплитудата на спектарот да соодветствува на амплитудата на сигналот според (3.43). Дополнително, пред конверзија на амплитудниот спектар во dB, ја пресекуваме



Сл. 3.3: Вообичаен приказ на амплитудниот (средина) и фазниот (долу) спектар на аудиосигналот генериран од звучна вилушка (горе).

неговата најмала вредност на произволно избрана мала вредност `eps` за да избегнеме пресметка на $\log(0)$. Добиениот приказ на двета спектри на аудиосигналот е даден на Сл. 3.3.

✓ **Задача за час.** Проверете што работи функцијата `fftshift` од модулот `scipy.fftpack`.

Бидејќи пресметката на спектарот на аудиосигналите е од суштествено значење во дигитално процесирање на аудио, можеме дадениот код да го искористиме за пишување на функција за таа намена. На функцијата ќе додадеме краток опис, или т.н. докстринг¹¹ и заедно со функцијата `normalize` која ја направивме во Главата 2.6 ќе ги поместиме во наш модул кој ќе го наречеме `da.py` со следната содржина:

```
#!/usr/bin/env python3
import numpy as np
from scipy import fftpack as fp

def normalize(wav_in, level=0):
    """Normalize input audio signal at level given in dBFS.

    Parameters
    -----
    wav_in : ndarray
        Input audio signal.
    level : float, optional
        Output level in dBFS. The default is 0.

    Returns
    -----
    wav_norm : ndarray
        Normalized audio signal.
    """
    # level = 20 * log(amp)
    amp_out = 10**(level / 20)
    amp_in = np.max(np.abs(wav_in))
```

¹¹анг. *docstring*.

```
wav_norm = wav_in / amp_in * amp_out
return wav_norm

def get_spectrum(wav, fs=44100, n_fft=None):
    """Get spectrum from audio signal.

    Parameters
    -----
    wav : ndarray
        Audio signal.
    fs : int, optional
        Sampling frequency. The default is 44 100 Hz.
    n_fft : int, optional
        Number of samples for FFT. The default is None.

    Returns
    -----
    f : ndarray
        Frequency vector.
    wav_amp : ndarray
        Amplitude spectrum.
    """

    n = wav.size
    n_fft = 2**np.ceil(np.log2(n))
    n_keep = int(n_fft / 2) + 1
    wav_spec = fp.fft(wav)
    wav_amp = np.abs(wav_spec)
    wav_amp = wav_amp / n
    wav_amp = wav_amp[0: n_keep]
    wav_amp[1: -1] = wav_amp[1: -1] * 2
    eps = 1e-8
    wav_amp[wav_amp < eps] = eps
    wav_amp = 20 * np.log10(wav_amp)
    f = np.linspace(0, fs / 2, n_keep)
    return f, wav_amp
```

Вака создадениот модул ќе ни послужи да ги искористиме направените функции во нашата понатамошна работа со дигиталните аудиосигнали. Така, за користење на новата функција `get_spectrum()` сè што треба да направиме е да го увеземе модулот исто како и другите. Така `wav_amp` може сега да го добијеме на следниот начин:

```
import da
f, wav_amp = da.get_spectrum(wav, fs)
```

§ Дополнително. Ако го поставите курсорот на името на функцијата и притиснете `Ctrl + i` во панелот за помош Spyder ќе го изрендерира напишаниот докстринг. Во докстрингот може да вклучиме многу повеќе информации вклучувајќи равенства и примери во код за употреба на нашата функција. Како пример за богат и добро форматиран докстринг погледнете ја функцијата `fft`.

3.3 Фуриеова трансформација со прозорци STFT

Главниот недостаток на Фуриевата трансформација е тоа што таа не ни дава никаква временска информација за сигналот кој што го анализираме. Поради нестационарната природа на аудиосигналите, Фуриевата анализа на целиот сигнал може да ни каже некои општи карактеристики за сигналот, но не може да ни ги покаже деталите. На пример, вкупниот спектар на еден музички сигнал може да ни каже дали во него има изразен бас преку анализата на енергијата на сигналот во ниските фреквенции, но ваквиот вкупен спектар не може да ни каже колку е брз ритамот на музичкиот сигнал. Оттаму потребата за временска локализација на

спектралната информација што ја нуди Фуриевата трансформација со прозорци (STFT)¹².



§ Дополнително. Унгарскиот електроинженер Денеш Габор¹³ (1900–1979) е овој кој во 1946 ја адаптира Фуриевата трансформација за временска анализа и ја добива STFT. Тој е и изумителот на холографијата за што добива Нобелова награда по Физика во 1971. Една од неговите најпознати изјави е: „Најдобриот начин да се предвиди иднината е таа да се создаде.“

STFT го анализира спектарот во низа од кратки временски отсекоци наречени **рамки**, земени од аудиосигналот со помош на функција за селекција $w[n]$ која се нарекува **прозорец**, според равенството (3.46). Тука, со i е означен редниот број на рамката, N е големината на прозорецот, а H е големината на скокот кој го прави прозорецот долж сигналот од една рамка до друга. Оваа метода на анализа на нестационарните сигнали преку земање на отсекоци со лизгање на прозорец по нивната должина се нарекува и **метода на прозорци**.

$$X[i, k] = \sum_{n=-N/2}^{N/2-1} x[n]w[n+i \cdot H]e^{-jk\frac{2\pi}{N}n}, \quad i = 0, 1, 2, \dots \quad (3.46)$$

На овој начин наместо еден вкупен спектар, се добива низа од спектри од сигналот пресметани за различни временски моменти во избрана нивна околина.

Видови на прозорци

Претходната анализа исто така го илустрира фактот дека спектарот на прозорецот има критично влијание за точната претстава на спектарот на аудиосигналите. Поради оваа причина дизајнирани се низа од различни видови на прозорци, секој со различни спектрални карактеристики, а со тоа и со различна примена.¹⁴ Сите тие во временски домен можат да се претстават како сума од косинуси, а во спектрален домен како сума од $\text{sinc}[n]$ функции.

Во STFT анализата на аудиосигналите најупотребувани прозорци (Serra and O Smith III, 2014) се:

- Правоаголен прозорец

$$w[n] = 1, \quad -N/2 \leq n \leq N/2 \quad (3.47)$$

$$W[k] = \text{sinc}[k] \quad (3.48)$$

- Ханов прозорец

$$w[n] = 0,5 + 0,5 \cos[2\pi \frac{n}{N}], \quad -N/2 \leq n \leq N/2 \quad (3.49)$$

$$W[k] = 0,5 \text{sinc}[k] + 0,25 (\text{sinc}[k-1] + \text{sinc}[k+1]) \quad (3.50)$$

- Хамингов прозорец

$$w[n] = 0,54 + 0,46 \cos[2\pi \frac{n}{N}], \quad -N/2 \leq n \leq N/2 \quad (3.51)$$

- Блекманов прозорец

$$w[n] = 0,42 - 0,5 \cos[2\pi \frac{n}{N}] + 0,080,5 \cos[4\pi \frac{n}{N}], \quad -N/2 \leq n \leq N/2 \quad (3.52)$$

¹²англ. *Short-Time Fourier Transform*.

¹³Wikipedia: Dennis Gabor. https://en.wikipedia.org/wiki/Dennis_Gabor

¹⁴Wikipedia: Window function https://en.wikipedia.org/wiki/Window_function

- Блекман-Харисов прозорец

$$w[n] = \frac{1}{N} \sum_{i=0}^3 a_i \cos\left[2i\pi \frac{n}{N}\right], \quad -N/2 \leq n \leq N/2 \quad (3.53)$$

каде

$$a_0 = 0,35876, a_1 = 0,48829, a_2 = 0,14128, a_3 = 0,01168 \quad (3.54)$$

Карактеристики на прозорците

Најважните спектрални карактеристики на прозорците се ширината на главното крило и амплитудата на најголемото споредно крило. Идеалниот прозорец има бескрајно тесно главно крило и нема споредни крила. Во реалноста станува збор за компромис помеѓу овие два параметри. Различните видови на прозорци можеме да ги генерираме со функцијата `get_window` од модулот `scipy.signal`. Во следниот код тоа е направено за правоаголниот прозорец `boxcar`. Притоа, амплитудните спектри се нормализираат до 0 dB.

```
import numpy as np
from matplotlib import pyplot as plt
from scipy import fftpack as fp
from scipy import signal as sig

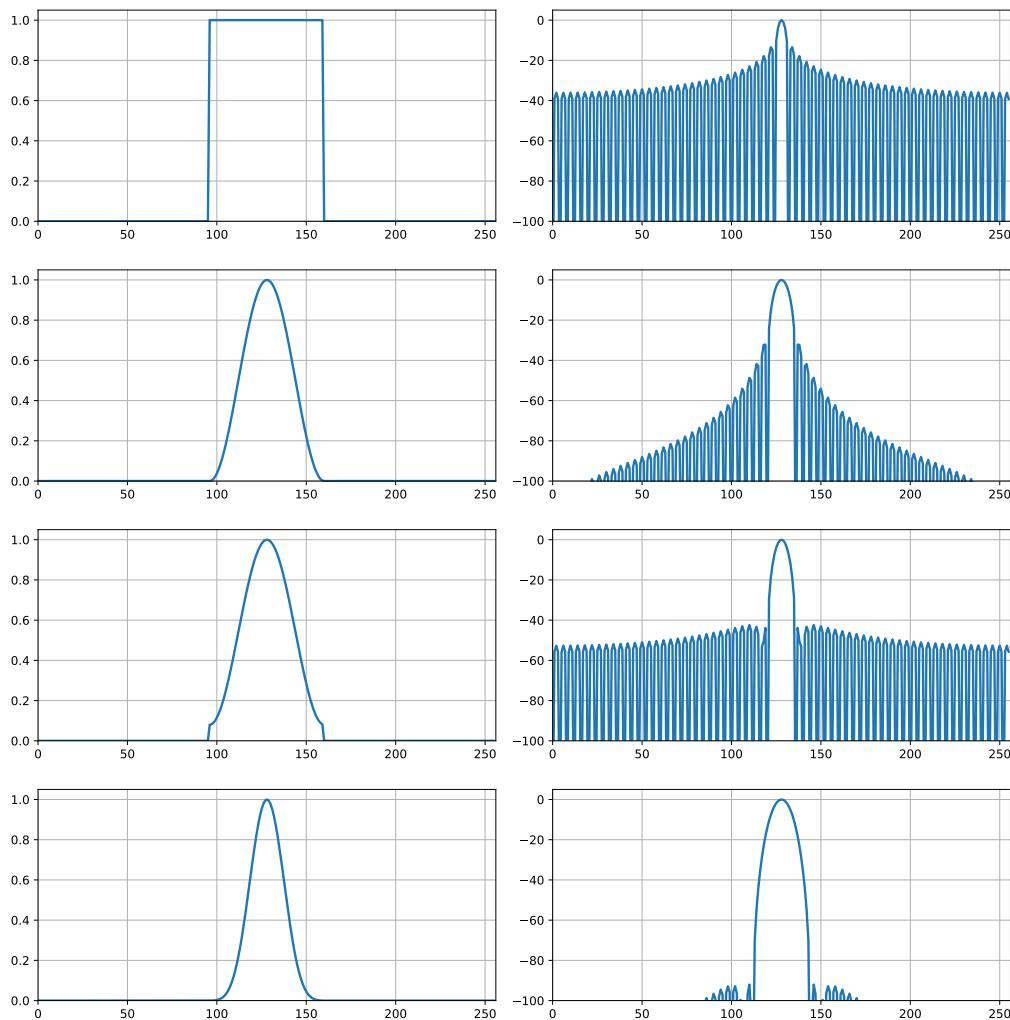
m = 512 # околина за анализа
n_win = 64 # должина на прозорец
m_half = m // 2
n_half = n_win // 2
w = np.zeros(m)
w[m_half - n_half: m_half + n_half] = sig.get_window('boxcar', n_win)
w_spec = fp.fft(w, m)
w_amp = np.abs(w_spec) / n_win
eps = 1e-8
w_amp[w_amp < eps] = eps
w_log = 20 * np.log10(w_amp)
w_log = w_log - np.max(w_log)
w_shift = fp.fftshift(w_log)

plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.plot(w)
plt.axis([0, m, 0, 1.1]) # [xmin, xmax, ymin, ymax]
plt.grid()
plt.subplot(122)
plt.plot(w_shift)
plt.grid()
plt.axis([0, m, -100, 10])
```

Притоа повторно ја воведовме произволно малата вредност `eps` за избегнување на пресметка на логаритамот за вредности на спектарот близки до нулата. Вистинската нумеричка прецизност за `float64` може да се добие со NumPy наредбата `np.finfo(float).eps` и таа изнесува $2,220446049250313 \cdot 10^{-16}$.

Различните прозорци и нивните амплитудни спектри генериирани со овој код се прикажани на Сл. 3.4. Може да се види дека правоаголниот прозорец иако е добар за процесирање во временски домен има најлоши спектрални карактеристики во поглед на амплитудата на страничните крила. Од друга страна, најмало влијание на страничните крила имаме кај Блекман-Харисовиот прозорецот, но тој за сметка на тоа има најшироко главно крило. Хановиот и Хаминговиот прозорец имаат иста широчина на главното крило на различно распоредена енергија во страничните крила.

Спектрограм



Сл. 3.4: Споредба на временскиот и спектрален облик на четири најчесто користени прозорци (од горе надолу): правоаголен, Ханов, Хамингов и Блекман-Харисов.

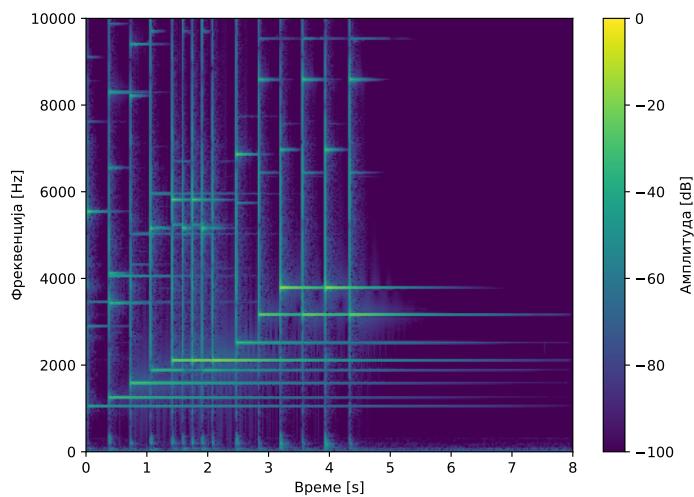
Спектрограмот е еден од најзначајните прикази на аудиосигналите. На него во исто време може да се набљудуваат карактеристиките на сигналот и во временски и во спектрален домен. За да го добиеме ќе се послужиме со STFT. Приказот е даден на Сл. 3.5.

```
import da

audio_path = 'audio/'
wav_name = 'zvona2.wav'
fs, wav = wavfile.read(audio_path + wav_name)
os.system('play ' + audio_path + wav_name)

wav = wav / 2**15
t = np.arange(0, wav.size/fs, 1/fs)

# %% пресметка на спектрограмот
n_win = 2048 # должина на прозорецот N (2**11)
n_hop = n_win // 2 # големина на скокот H
win = sig.get_window('hamming', n_win)
pad = np.zeros(n_win // 2)
wav_pad = np.concatenate((pad, wav, pad))
pos = 0 # позиција на почетокот на прозорецот
while poz < m - n_half:
    frame = wav_pad[pos-n_half : poz+n_half] * win
    f_frame, frame_spec = da.get_spectrum(frame, fs)
```



Сл. 3.5: Спектрограма на аудиозаписот `zvona2.wav`¹⁵ добиен со употреба на Хамингов прозорец со должина 2048 примероци (46 ms)

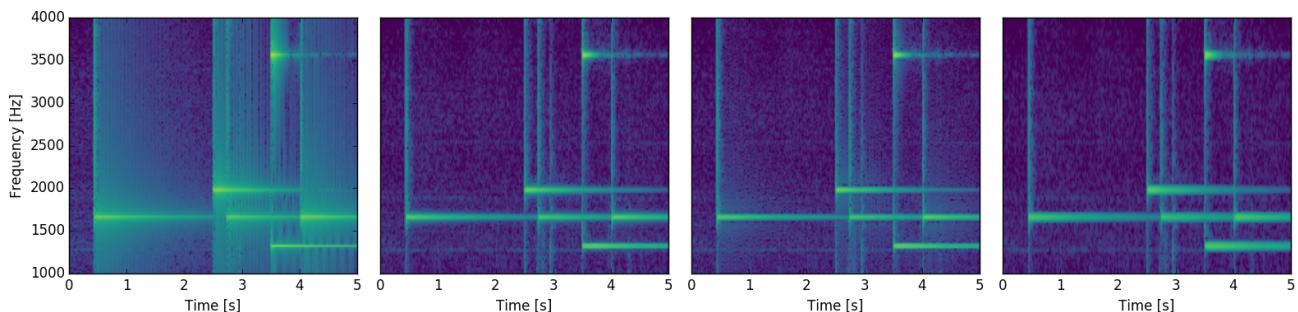
```

frame_spec_2d = frame_spec[:, np.newaxis]
if frames_spec is None:
    frames_spec = frame_spec_2d
else:
    frames_spec = np.hstack((frames_spec, frame_spec_2d))
poz += h

t_frames = np.arange(frames_spec.shape[1]) * n_hop / fs

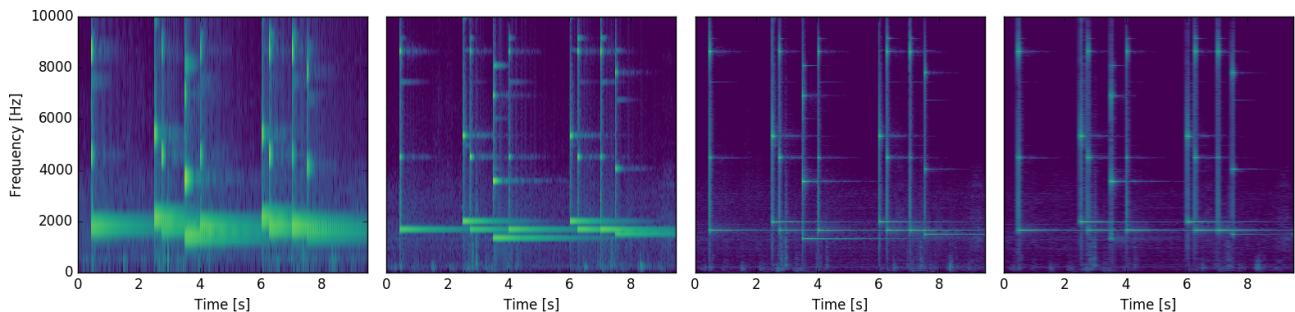
# %% приказ на спектрограмот
plt.figure()
plt.imshow(frames_spec, aspect='auto',
           origin='lower',
           extent=[0, t[-1], 0, f_frame[-1]],
           vmin=-100, vmax=0,
           cmap='viridis'
)
plt.colorbar()
plt.xlabel('Време [s]')
plt.ylabel('Фреквенција [Hz]')
cbar.ax.set_ylabel('Амплитуда [dB]')
plt.axis([0, t[-1], 0, 10000])

```



Сл. 3.6: Детали од спектрограмот на аудиозаписот `zvona2.wav` добиени за различни типови на прозорци: правоаголен, Ханов, Хамингов и Блекман-Харисов, сите со должина 2048 примероци.

¹⁵Звукот Gentle Glockenspiel од bbatv е земен од Freesound.org. <http://freesound.org/people/bbatv/sounds/332932/>



Сл. 3.7: Спектрограми на аудиозаписот `zvona2.wav` добиени со употреба на Хамингов прозорец со различни должини: 128, 512, 4096 и 8192 примероци.

★ Важно! Најважните два параметри во пресметување на спектрограмот се:

1. Типот на прозорец. Разликите во спектрограмот кои се добиваат со различните прозорци може да се воочат во деталите прикажани на Сл. 3.6.
2. Големината на прозорецот. Колку е поголем прозорецот толку повеќе точки, односно поголема резолуција, во спектарот на рамката земена од сигналот. Но исто така, колку поголем прозорец толку погруба временска резолуција на STFT анализата, Сл. 3.7. Имено, временските моменти на промена во спектарот биваат размачкани. Така, за добра временска резолуција ни требаат пократки прозорци (рамки од сигналот), кои пак повлекуваат лоша фреквенциска резолуција. Потребата од компромис меѓу време и фреквенција е главниот недостаток кај спектрограмите, односно STFT анализата.

§ Дополнително. За прикажување на амплитудата на спектрограмот постојат најразлични низи на бои, наречени мали на боја. Долго време во стандардна употреба е мапата `jet`, но таа има низа на недостатоци, поради кои денес се исфрла од употреба. Мапата искористена за приказ на спектрограмите во ова поглавје е новата `viridis` која е претставена на конференцијата SciPy 2015¹⁶.

Останува да ги внесеме функциите за пресметка на спектрограмот и неговото испртување надополнети со описи на влезните и излезни параметри во модулот `da.py`:

```
from matplotlib import pyplot as plt
import scipy.signal as sig

def get_spectrogram(wav, fs=44100, n_win=2048, n_hop=None, win_type='hann', plot=True):
    """Calculate spectrogram of signal.

    Parameters
    -----
    wav : ndarray
        Audio signal.
    fs : int, optional
        Sampling frequency. The default is 44 100 Hz.
    n_win : int, optional
        Window length, should be of the form 2**X. The default is 2048.
    n_hop : int, optional
        Hop size. The default is n_win / 2.
    win_type : str, optional
        Window type to use. Default is Hann.
    plot : bool, optional
        Plot the spectrogram. The default is True.
```

¹⁶ Особено интересното претставување на новата мапа може да го погледнете на следниот линк: Nathaniel Smith and Stéfan van der Walt -- A Better Default Colormap for Matplotlib <https://www.youtube.com/watch?v=xAoIjeRJ3IU>

```

>Returns
-----
t_frames : ndarray
    Time locations of frame centers.
f_frame : ndarray
    Frequency of spectrum bins in Hz.
spectrogram : ndarray, shape [n_freq_bins, n_frames]
    Calculated spectrogram.
"""

win = sig.get_window(win_type, n_win)
pad = np.zeros(n_win // 2)
wav_pad = np.concatenate([pad, wav, pad])
if n_hop is None:
    n_hop = n_win // 2
pos = 0
while pos <= wav_pad.size - n_win:
    frame = wav_pad[pos: pos + n_win] * win
    f_frame, frame_spec = get_spectrum(frame, n_fft=n_win, fs=fs)
    frame_spec = frame_spec[:, np.newaxis]
    if pos == 0:
        spectrogram = frame_spec
    else:
        spectrogram = np.concatenate([spectrogram, frame_spec], axis=1)
    pos += n_hop
t_frames = np.arange(spectrogram.shape[1]) * n_hop / fs
if plot:
    plot_spectrogram(t_frames, f_frame, spectrogram)
return t_frames, f_frame, spectrogram

def plot_spectrogram(t_frames, f_frame, spectrogram, f_max=10000):
    """Plot spectrogram.

Parameters
-----
t_frames : ndarray
    Time positions of window center for each frame.
f_frame : ndarray
    Frequency bins of spectrum.
spectrogram : ndarray, shape [n_freq_bins, n_frames]
    Calculated spectrogram.
f_max : float, optional
    Maximum frequency on the y-axis. Default is 16 kHz.
"""

    if f_max is None:
        f_max = f_frame[-1]
    plt.figure(figsize=(8, 6))
    plt.imshow(
        spectrogram,
        aspect='auto',
        origin='lower',
        extent=[0, t_frames[-1], 0, f_max],
    )
    plt.axis([0, t_frames[-1], 0, f_max])
    plt.colorbar()
    plt.tight_layout()

```

✓ Задача за час. Прикажете го спектарот на сигналот без методата на прозорци и протолкувајте ја неговата содржина повикувајќи се на изгледот на спектрограмот.

✓ Задача за дома. Со помош на спектрограмот да се илустрира ефектот на преклопување на спектарот преку намалување на фреквенцијата на земање примероци без нископропусно филтрирање на аудиосигналот.

Глава 4

Филтри

Филтрите претставуваат системи чија примарна намена е обликувањето на спектарот на аудиосигналите. Тие се нераздвоен дел од дигиталниот звук. Аналогни и дигитални филтри се потребни за ограничување на спектарот во А/Д конверзијата и за обликување на излезниот аналоген сигнал во D/A конверзијата; дигитални филтри имаме долж каналот за пренос на дигиталниот звук (трансмисија, дигитално снимање), а се основни градбени единки на еквилизаторите. Аналогните филтри се реализираат со помош на збир на пасивни и активни електронски елементи, чија нагоденост е неопходна, но скапа. Од друга страна, реализацијата сложени филтерски структури во дигитален домен е многу поекономична, а нумеричките операции на кои се базираат дигиталните филтри не се подложни на временски и температурни влијанија.

4.1 Основи на дигиталните филтри

Дигиталните филтри претставуваат линеарни и временски инваријантни (LTI), односно линеарни и инваријантни на поместување дискретни системи¹. Овие две особини изискуваат ако за даден влезен сигнал $x[n]$ системот го дава излезниот сигнал $y[n] = H\{x[n]\}$, тогаш мора да важи (Богданова, 1997):

$$H\left\{\sum_{m=1}^M a_m x_m[n]\right\} = \sum_{m=1}^M a_m y_m[n], \quad \forall a_m \text{ -- линеарност и} \quad (4.1)$$

$$H\{x[n-k]\} = y[n-k], \quad \forall k \text{ -- инваријантност на поместување.} \quad (4.2)$$

Секој LTI систем е во потполност описан од неговиот импулсен одсив $h[n]$ односно неговата преносна функција во z -домен $H(z)$, или пак конечно од неговата фреквенциска карактеристика во Фуриев домен $H(\omega)$. Како и за останатите LTI системи, излезниот сигнал на дигиталните филтри $y[n]$ за даден влезен сигнал $x[n]$ може да се добие во трите домени со следните релации (Rabiner and Schafer, 1978):

$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m], \quad (4.3)$$

$$Y(z) = H(z)X(z), \quad (4.4)$$

$$Y(\omega) = H(\omega)X(\omega). \quad (4.5)$$

За системот да биде остварлив, нужно е тој да биде каузален, односно да важи:

$$h[n] \equiv 0, \quad n < 0. \quad (4.6)$$

¹анг. *Linear time-invariant (LTI)*, односно *linear shift-invariant (LSI)* системи.

За системот пак да биде стабилен потребен и доволен услов е да важи:

$$\sum_{n=-\infty}^{\infty} |h[n]| < \infty. \quad (4.7)$$

Типови филтри според должината на импулсниот одсив

Според импулсниот одсив филтрите ги делиме на две основни групи и тоа филтри со:

- конечен импулсен одсив (FIR)² и
- бесконечен импулсен одсив (IIR)³.

FIR филтрите имаат одредени предности над IIR филтрите, а тоа се пред сè нивната стабилност и нивната линеарна фазна карактеристика. IIR филтрите пак можат да постигнат подобра амплитудна карактеристика за помал ред на филтерот.

Сите LTI системи кои се практично применливи како дигитални филтри можат да се описват со диференцната равенка:

$$y[n] = \sum_{i=0}^p b_i x[n-i] - \sum_{i=1}^q a_i y[n-i]. \quad (4.8)$$

Според ова равенство секој примерок на излезниот сигнал $y[n]$ зависи од p претходни примероци на влезниот сигнал и q минати примероци од излезниот, односно во системот има повратна врска, па велиме дека тој е **рекурзивен**. IIR филтрите секогаш се реализираат со повратна врска, од каде произлегуваат и проблемите со нивната стабилност. FIR филтрите можат да бидат реализирани и со повратна врска, но најчесто се без неа. Преносната функција на системот описан од диференцната равенка можеме да ја добиеме од (4.8):

$$y[n] + \sum_{i=1}^q a_i y[n-i] = \sum_{i=0}^p b_i x[n-i], \quad (4.9)$$

$$\left(1 + \sum_{i=1}^q a_i z^{-i} \right) Y(z) = \sum_{i=0}^p b_i z^{-i} X(z), \quad (4.10)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^p b_i z^{-i}}{1 + \sum_{i=1}^q a_i z^{-i}}. \quad (4.11)$$

Тука имплицитно претпоставуваме дека $a_0 = 1$. Во праксата овој услов секогаш се обезбедува преку нормализирање на останатите a коефициенти. Бројот на коефициенти во повратната врска на IIR филтрите вообичаено се зема да е еднаков со бројот на коефициенти во директната врска. Па, важи:

$$q = p = N, \quad (4.12)$$

каде со N се означува **редот на филтерот**. Преносната функција $H(z)$ исто така може да биде представена преку нејзините **полови и нули** во z -рамнината:

$$H(z) = \frac{A \prod_{i=1}^p (1 - c_i z^{-1})}{\prod_{i=1}^q (1 - d_i z^{-1})}. \quad (4.13)$$

Типови филтри според амплитудната карактеристика

Фреквенциската карактеристика на филтрите ја добиваме директно од преносната функција со замената $z = e^{j\omega}$:

$$H(\omega) = \frac{\sum_{i=0}^p b_i e^{-ji\omega}}{1 + \sum_{i=1}^q a_i e^{-ji\omega}}. \quad (4.14)$$

²анг. *Finite Impulse Response*.

³анг. *Infinite Impulse Response*.

Таа може дополнително да се изрази преку нејзината амплитуда и фаза како:

$$H(\omega) = |H(\omega)|e^{j\angle H(\omega)} = A(\omega)e^{j\varphi(\omega)}, \quad (4.15)$$

каде $A(\omega)$ е амплитудната фреквенциска карактеристика на филтерот, додека $\varphi(\omega)$ е неговата фазна карактеристика. Според амплитудната карактеристика разликуваме пет типови на филтри и тоа:

- нископропусни (LP)⁴,
- високопропусни (HP)⁵,
- пропусни на опсег (BP)⁶,
- непропусни на опсег (BS)⁷, и
- сепропусни (AP).⁸

Иако сепропусните филтри се контраинтуитивни, поради амплитудната карактеристика која изнесува 1 за сите фреквенции, тие имаат корисна фазна карактеристика, па наоѓаат примена кај системите за синтеза на дигитално ехо и реверберација, види Глава .

Кога фазната карактеристика на филтерот $\varphi(\omega)$ е линеарна тогаш групното доцнење $\tau(\omega)$ е константно:

$$\tau(\omega) = -\frac{d\varphi(\omega)}{d\omega} = \text{const.} \quad (4.16)$$

Ова значи дека филтерот нема да внесе фазни изобличувања. Со тоа, компонентите на сигналот на различни фреквенции нема да бидат различно задоцнети па ќе биде задржана нивната компактност во излезниот сигнал, односно нема да дојде до нивно расејување.

Типови FIR филтри со линеарна фазна карактеристика

Строго линеарната фазна карактеристика не може да биде постигнато со IIR или со аналогните филтри (Богданова, 1997), туку само со FIR филтри чив импулсен одсив е симетричен во однос на средниот примерок $h[\frac{N-1}{2}]$:

$$h[n] = h[N - 1 - n], \quad (4.17)$$

или пак кога е антисиметричен во однос на него:

$$h[n] = -h[N - 1 - n]. \quad (4.18)$$

Тука редот на филтерот N ја дава и должината на импулсниот одсив, што не е случај кај IIR филтрите.

Постојат четири типови на FIR филтри со линеарна фазна карактеристика:

Тип I -- Симетричен импулсен одсив, N непарен

Импулсниот одсив на овој тип на филтри можеме да го запишеме како:

$$h[n] = h[0]\delta[n] + h[1]\delta[n - 1] + \cdots + h[\frac{N-1}{2}]\delta[n - \frac{N-1}{2}] + \cdots \quad (4.19)$$

$$\cdots + h[1]\delta[n - (N-2)] + h[0]\delta[n - (N-1)], \quad (4.20)$$

⁴анг. *low-pass*.

⁵анг. *high-pass*.

⁶анг. *band-pass*.

⁷анг. *band-stop*.

⁸анг. *all-pass*.

па за преносната функција имаме:

$$H(z) = h\left[\frac{N-1}{2}\right]z^{-\frac{N-1}{2}} + \sum_{i=0}^{\frac{N-3}{2}} h[i] \left(z^{-i} + z^{-(N-1-i)} \right) \quad (4.21)$$

$$= z^{-\frac{N-1}{2}} \left(h\left[\frac{N-1}{2}\right] + \sum_{i=0}^{\frac{N-3}{2}} h[i] \left(z^{-i+\frac{N-1}{2}} + z^{i-\frac{N-1}{2}} \right) \right) \quad (4.22)$$

$$= z^{-\frac{N-1}{2}} \sum_{i=0}^{\frac{N-1}{2}} a[n] \frac{z^n + z^{-n}}{2}, \quad (4.23)$$

каде:

$$a[n] = \begin{cases} h\left[\frac{N-1}{2}\right], & n = 0 \\ 2h[-n + \frac{N-1}{2}] & n = 1, 2, \dots, \frac{N-3}{2} \end{cases}. \quad (4.24)$$

Од (4.23) може да се пресмета фреквенциската карактеристика на филтерот:

$$H(\omega) = e^{-j\omega\frac{N-1}{2}} \sum_{n=0}^{\frac{N-1}{2}} a[n] \cos(n\omega). \quad (4.25)$$

Тип II -- Симетричен импулсен одсив, N парен

Следејќи ја истата постапка како за FIR филтерот од тип I можеме да дојдеме до фреквенциската карактеристика на типот II:

$$H(\omega) = e^{-j\omega\frac{N-1}{2}} \sum_{n=1}^{\frac{N-1}{2}} b[n] \cos((n - \frac{1}{2})\omega), \quad (4.26)$$

каде:

$$b[n] = 2h[-n + \frac{N}{2}], \quad n = 1, 2, \dots, \frac{N}{2}. \quad (4.27)$$

Од (4.26) може да се види дека без оглед на вредноста на коефициентите на филтерот $b[n]$ неговата фреквенциска карактеристика ќе биде 0 за $\omega = \pm\pi$. Поради тоа, овој тип на филтер не може да се употреби како високопропусен или непропусник на опсег.

Тип III -- Антисиметричен импулсен одсив, N непарен

Фреквенциската карактеристика на типот III е:

$$H(\omega) = e^{-j(\omega\frac{N-1}{2} - \frac{\pi}{2})} \sum_{n=1}^{\frac{N-1}{2}} c[n] \sin(n\omega), \quad (4.28)$$

каде:

$$c[n] = 2h[-n + \frac{N-1}{2}], \quad n = 1, 2, \dots, \frac{N-1}{2}. \quad (4.29)$$

Од (4.28) следи дека без оглед на вредноста на $c[n]$ неговата фреквенциска карактеристика ќе биде 0 за $\omega = 0$ и за $\omega = \pm\pi$. Поради тоа, овој тип на филтер може да се употреби само како пропусник на опсег.

Тип IV -- Антисиметричен импулсен одсив, N парен

Фреквенциската карактеристика на овој филтер е:

$$H(\omega) = e^{-j\omega\frac{N-1}{2}} \sum_{n=1}^{\frac{N}{2}} d[n] \sin((n - \frac{1}{2})\omega), \quad (4.30)$$

каде:

$$d[n] = 2 h[-n + \frac{N}{2}], \quad n = 1, 2, \dots, \frac{N}{2}. \quad (4.31)$$

Од (4.30) следи дека фреквенциска карактеристика ќе биде 0 за $\omega = 0$, па овој тип на филтер не може да се употреби како нископропусен или пропусник на опсег.

Постојат различни пристапи за [дизајн на дигитални филтри](#). Трите најпознати методи за дизајн на FIR филтри се:

- [метода на прозорци](#),
- [дизајн заснован на DFT](#),
- [оптимален дизајн на еднаквобранести филтри](#).

Од друга страна за дизајн на IIR филтри најпознати методи се:

- [Батерворт](#),
- [Бесел](#),
- [Чебишев](#),
- [Елиптичен](#).

4.2 Дизајн на FIR филтер

Дизајнот на FIR филтри со методата на прозорци се заснова на апроксимација на идеалната фреквенциска карактеристика на филтерот преку земање на прозорец од импулсниот одсив кој одговара на неа. Ќе ја илустрираме методата за нископропусен филтер со гранична фреквенција ω_l . Идеалната фреквенциска карактеристика на ваков филтер би била:

$$H_l(\omega) = \begin{cases} 1, & |\omega| \leq \omega_l \\ 0, & \omega_l < |\omega| \leq \pi \end{cases}. \quad (4.32)$$

Импулсниот одсив на овој идеален филтер $h_l[n]$ ќе биде:

$$h_l(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_l(\omega) e^{jn\omega} d\omega = \frac{1}{2\pi} \int_{-\omega_l}^{\omega_l} e^{jn\omega} d\omega = \frac{\sin(n\omega_l)}{\pi n}. \quad (4.33)$$

Овој импулсен одсив не може практично да се реализира поради тоа што неговото траење е бесконечно и тој не е каузален. За таа цел во методата на прозорци се зема само дел од него преку негово множење со избран прозорец. И тука важат истите дискусиии во [Поглавјата](#) и за влијанието на спектралните карактеристики на прозорците и компромисот помеѓу домените време и фреквенција.

За да ја илустрираме оваа метода ќе напишеме код кој идеалниот филтер ќе го конструира во Фурьеов домен во `n_fft = fs` точки, неговиот импулсен одсив ќе го пресмета со употреба на IDFT, за од него да задржиме еден дел со употреба на правоаголен прозорец со должина `n`. Поради тоа што импулсниот одсив го пресметуваме од идеалната фреквенциска карактеристика, имплементираниот алгоритам за дизајн на FIR филтер всушност претставува комбинација од методите за дизајн базирани на DFT и примената на прозорци.⁹

```
import numpy as np
from matplotlib import pyplot as plt
from scipy import fftpack as fft
from scipy import signal as sig

# %% конструкција на филтерот
```

⁹Благодарност за предочување на ова доц. д-р Јелена Ќертиќ, професор по дигитално процесирање на сигнали на Електротехничкиот Факултет при Универзитетот во Белград.

```

fs = 44100
w_l = 5000 / (fs/2) # гранична фреквенција нормализирана до 1
n_fft = fs
w = np.linspace(0, np.pi, n_fft/2 + 1)
h_spec_l = np.zeros(w.size)
h_spec_l[w < w_l * pi] = 1 # идеална карактеристика 0 до pi
h_spec_l = np.append(h_spec_l, h_spec_l[-2 : 0 : -1]) # огледална слика pi до 2pi
h_l = fp.ifft(h_spec_l, n_fft)
h_l = fp.fftshift(h_l) # го правиме системот каузален

# %% метода на прозорци
n = 128 + 1 # доджина на прозорецот = ред на филтерот
nh = (n-1) / 2
n_ffth = n_fft/2

win = sig.get_window('boxcar', n) # правоаголен прозорец
n_win = np.arange(n_ffth - nh, n_ffth + nh + 1) # индекси кои ни требаат
h_rect = h_l[tuple(n_win),] * win # реален филтер

# %% плотирање
plt.figure()
plt.subplot(211)
plt.plot(h_l, linewidth=1, alpha=1)
plt.plot(n_win, h_rect, lw=2, alpha=.8)
plt.grid()
plt.subplot(212)
f, h_spec_l = da.get_spectrum(h_l, fs)
plt.plot(f, h_spec_l)
f, h_spec_rect = da.get_spectrum(h_rect_long, fs)
h_spec_rect = h_spec_rect - np.max(h_spec_rect) # нормализација
plt.plot(f, h_spec_rect, lw=1.5, alpha=.8)
plt.grid()

```

Конструираната фреквенциска карактеристика на идеалниот нископропусен филтер и онаа добиена со методата на прозорци со употреба на правоаголен прозорец се прикажани заедно со нивните импулсни одсиви во Сл. 4.1. Може да се види деградацијата на фреквенциската карактеристика кај добиениот нископропусен филтер поради ограничување на идеалниот импулсен одсив. Исто така може да се види ефектот на множење со правоаголниот прозорец во временски домен, кое претставува конволуција на фреквенциските карактеристики на двата сигнали во спектрален домен.

§ Дополнително. На Сл. 4.1 може да се види дека ни идеалниот импулсен одсив ја нема оригинално конструираната идеална фреквенциска карактеристика. Ова е поради тоа што вистински идеалниот импулсен одсив има бесконечно многу примероци, а овој кој ние го конструиравме има `fs`. Всушност и тој самиот како да сме го добиле со методата на прозорци од вистинскиот.

4.3 Филтрирање на аудиосигнал со FIR филтер

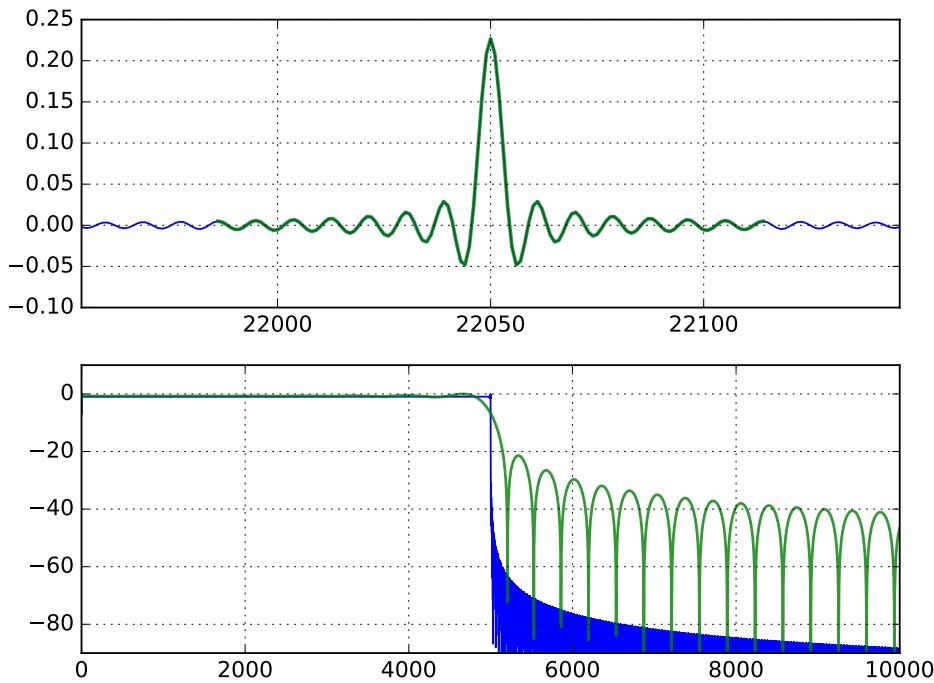
Во овој дел ќе исфилтрираме еден звучен сигнал со FIR филтер направен со методата на прозорци имплементирана во функцијата `scipy.signal.firwin`.

```

import numpy as np
from scipy import signal as sig
from scipy.io import wavfile
import matplotlib.pyplot as plt
import os
import da

# %% вчитај го аудиосигналот

```



Сл. 4.1: Импулсни одсиви и фреквенциски карактеристики на конструираните идеален нископропусен филтер и овој добиен со методата на прозорци.

```

audio_path = '../audio/'
file_name = 'Mara.wav'
fs, wav_orig = wavfile.read(audio_path + file_name)
os.system('play ' + audio_path + file_name)
wav_orig = wav_orig / 2**15

# %% исцртај спектрограм
-- = da.get_spectrogram(fs, wav_orig)

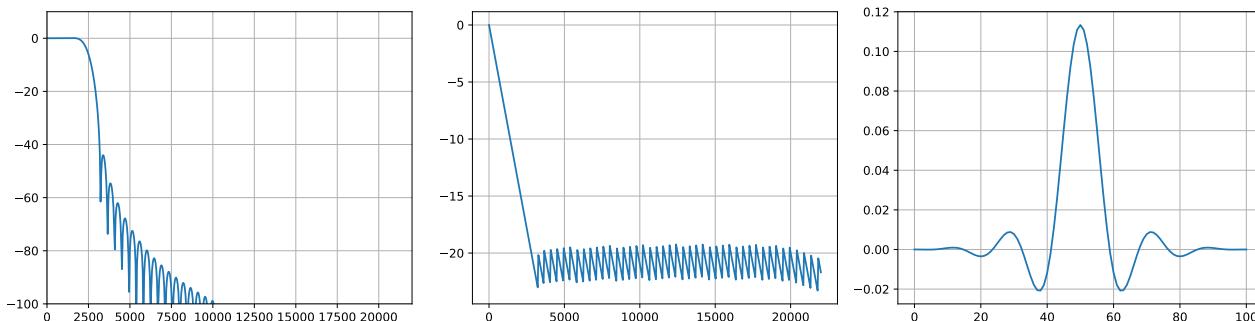
# %% дизајнирај FIR филтер
order = 100
f_l = 2500
b = sig.firwin(order+1, f_l / (fs/2), pass_zero=True, window='hann')

# %% исцртај ја неговата фреквенциска карактеристика
w, h_w = sig.freqz(b)
f = w / np.pi * fs/2
h_amp = 20 * np.log10(np.abs(h_w))
h_ph = np.unwrap(np.angle(h_w))

plt.figure(figsize=(15, 4))
plt.subplot(131)
plt.plot(f, h_amp)
plt.grid()
plt.axis([0, f[-1], -100, 10])
plt.subplot(132)
plt.plot(f, h_ph)
plt.grid()
plt.subplot(133)
plt.plot(b)
plt.grid()
plt.tight_layout()

```

Карактеристиките на дизајнираниот FIR филтер со ред $N = 100$ се прикажани на Сл. 4.2. Според неговата амплитудната карактеристика, може да видиме дека филтерот е нископропусен со гранична фреквенција f_l од 2,5 kHz. Исто така може да го видиме влијанието на Хановиот прозорец врз амплитудната карактеристика. Исто така може да видиме дека тој има строго



Сл. 4.2: Амплитудна (лево) и фазна карактеристика (средина) и импулсен одсив (десно) на дизајнираниот FIR филтер од 100-ти ред со употреба на Ханов прозорец.

линеарна фазна карактеристика, односно внесува подеднакво доцнење на компонентите на сигналот долж сите фреквенции. Конечно можеме да го видиме неговиот одсив којшто поради каузалноста почнува од нулата со врвна амплитуда за $n = 51$, поради што филтерот внесува поместување, односно доцнење во излезниот сигнал.

- ✓ **Задача за час.** Проверете како се менуваат карактеристиките на филтерот со промена на:
- границната фреквенција,
 - редот на филтерот, и
 - типот на прозорецот.

Ајде сега да го исфилтрираме нашиот сигнал и да видиме како ќе се промени неговата спектротемпорална содржина.

```
# %% исфилтрирај го аудиосигналот и прикажи го спектрограмот на излезниот сигнал
wav_filt = sig.lfilter(b, 1, wav_orig)
__ = da.get_spectrogram(fs, wav_filt)

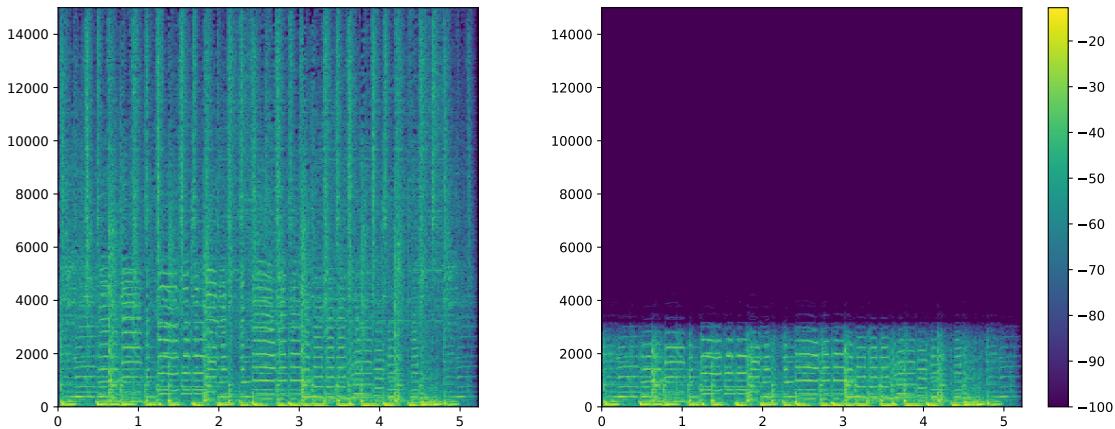
# %% сними и преслушај го добиениот аудиосигнал
wav_16 = wav_filt * 2**15
wav_16 = wav_16.astype('int16')
wavfile.write(audio_path + 'mara_fir.wav', fs, wav_16)
os.system('play ' + audio_path + 'mara_fir.wav')
```

Спектрограмот на вчитаниот и исфилтрираниот аудиосигнал се прикажани на Сл. 4.3. Може да видиме дека се работи за сложен аудиосигнал во кој се измешани периодични и апериодични звучни сигнали. Притоа, бидејќи спектрограмот не ни требаше понатаму во нашиот код, излезот на функцијата `da.get_spectrogram` го доделивме на променливата со име `__` што вообичаено се прави во вакви случаи. Во исфилтрираниот аудиосигналот може да забележиме дека филтерот ефикасно ги отстранил сите фреквенциски компоненти на сигналот над 2,5 kHz, како што може и да чуеме во излезниот аудиосигнал.

4.4 Дизајн на IIR филтри

За дизајн на IIR филтри ќе се послужиме со функцијата `scipy.signal.iirfilter` која содржи имплементации на дизајнот на IIR филтри според споменатите методите на Батерворт, Бесел, Чебишев, и онаа на елиптичните филтри. Ќе ја искористиме методата на Батерворт која дава филтри со строго монотона фреквенциска карактеристика.

```
# %% дизајн на IIR филтер
order = 10
f_l = 2500
```



Сл. 4.3: Спектрограм на аудиосигналот Mara пред (лево) и по филтрирањето со FIR филтерот (десно).

```
b, a = sig.iirfilter(order, f_l / (fs/2), btype='lowpass', ftype='butter')
w, h_w = sig.freqz(b, a)
f = w/np.pi * fs/2
h_amp = 20 * np.log10(np.abs(h_w))
h_ph = np.unwrap(np.angle(h_w))

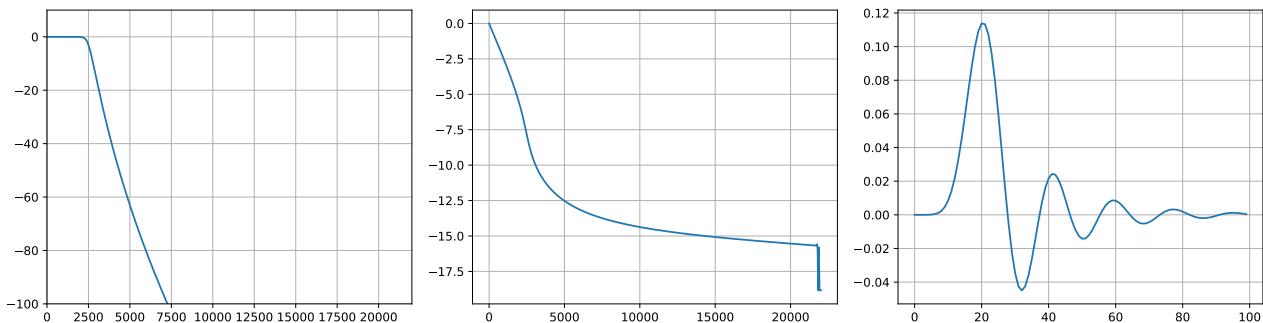
excite = np.zeros(100)
excite[0] = 1
h_n = sig.lfilter(b, a, excite)

plt.figure(figsize=(15, 4))
plt.subplot(131)
plt.plot(f, h_amp)
plt.grid()
plt.axis([0, f[-1], -100, 10])
plt.subplot(132)
plt.plot(f, h_ph)
plt.grid()
plt.subplot(133)
plt.plot(h_n)
plt.grid()
plt.tight_layout()
```

Забележете дека функцијата `iirfilter` ги враќа `b` но и `a` коефициентите на дизајнираниот IIR филтер, двата $N + 1$ на број. Исто така, бидејќи импулсниот одсив не може сега да се добие директно од коефициентите на филтерот, првин создаваме побуден сигнал во форма на Дираков импулс `excite` кој потоа го филтрираме со коефициентите на филтерот за да го добиеме неговиот импулсен одсив `h_n`.

На Сл. 4.4 се прикажани карактеристиките на дизајнираниот IIR филтер со ред $N = 10$. Споредено со амплитудната карактеристика на FIR филтерот од 100-ти ред, може да видиме дека IIR филтерот има споредливи перформанси со 10 пати помал ред! Поради ова IIR филтрите почесто се користат во процесирањето на аудиосигналите, и покрај нелинеарната фазна карактеристика, којашто може да се види на сликата. Како додатна предност, Батерворт филтрите ги немаат бранувањата во амплитудната карактеристика кои ги внесува прозорецот при дизајнот на FIR филтрите.

Конечно, можеме да видиме дека импулсниот одсив на дизајнираниот IIR филтер не е бесконечен, односно за одредено време неговата амплитуда може да се занемари. Всушност времетраењето на импулсниот одсив на IIR филтеров е споредливо со она на претходно дизајнираниот FIR филтер.



Сл. 4.4: Амплитудна (лево) и фазна карактеристика (средина) и импулсен одсив (десно) на дизајнираниот IIR Батерворт филтер од 10-ти ред.

✓ **Задача за час.** Проучете ја промената на карактеристиките на IIR филтерот за различни редови на филтерот. Дали може да се дизајнира стабилен IIR филтер од произволно голем ред?

4.5 Употреба на IIR филтри за еквализација

Еквализацијата настанала како потреба да се израмни фреквенцискиот одсив на преносните системи. Звукот низ својот пат од изворот до слушателот поминува низа аудиоуреди кои со своите неидеалности го изобличуваат неговиот првобитен спектар. Така, микрофоните немаат идеално рамна фреквенциска карактеристика -- нивната чувствителност е помала на ниските фреквенции како и на многу високите. Сличен е и одсивот на звучниците. Ова „обојување“ на звукот може соодветно да се компензира со употреба на уред со инверзна фреквенциска карактеристика на онаа на каналот. Со ова преносната функција на склопот ја поништува нерамномерноста на каналот т.е. ефективно го исправа истиот. Од тука доаѓа името на процесот еквализација, односно на уредот -- еквализатор.

Во аудиосистемите, во употреба е поширока дефиниција за еквализацијата. Според неа, еквализацијата е процес кој служи за генерално обликување на спектарот на аудиосигналот, кое не мора да се стреми кон идеално рамна фреквенциска карактеристика. Така, еквализацијата може да послужи и за намерно истакнувањето, односно потиснувањето, на одредени делови од спектарот на аудиосигналите. Па затоа, еквализацијата вообичаено се третира како дигитален аудиоэффект, види Глава 5. Еден пример за практична примена на еквализацијата е истакнување на ритамот во дискотеките преку засилување на ниските фреквенции на музиката. Од друга страна, засилувањето на високите фреквенции во корист на ниските навидум му дава на звукот поголема гласност иако целовкупната негова моќност останува иста. Ова го користат маркетинг агенциите за да го привлечат вниманието на слушателите.

Основниот начин на реализација на еден еквализаторот се состои од примена на повеќе филтри пропусници на опсег, вскладени да го препокријат целиот звучен опсег. На тој начин, тие го делат на подопсези кои можат да се истакнат или потиснат преку одредување на засилувањето на филтерот. Важно е вкупната фазна карактеристика на филтрите да биде линеарна. Мора да се води сметка и за доцнењата кои филтрите ги внесуваат, ако тие меѓусебно се разликуваат, тогаш резултантниот сигнал ќе има изразени изобличувања.

Во аналогната техника бројот на филтри е ограничен со цената на уредот, додека во дигиталната техника со процесирачката моќ. Затоа добрите аналогни аудиосистеми најчесто вклучуваат едноставна бас и требл¹⁰ еквализација, додека дигитални аудиопреслушувачи вообичаено поддржуваат 6 и повеќе-канална еквализација. Притоа дигиталните филтри се најчесто од IIR

¹⁰Анг. bass и treble.

тип од 4 ред, за намалување на комплексноста. Притоа, треба да се обрне особено внимание на нелинеарната фазна карактеристика на IIR филтрите.

Еквализаторот во Python ќе го направиме преку паралелна врска на 3 филтри:

- филтер за бас -- нископропусен до 400 Hz,
- филтер за средни -- пропусник на опсег од 400 до 4000 Hz и
- филтер за високи фреквенции -- високопропусен над 4 kHz.

```
%%% дефинирање на филтрите
n = 7 # ред на филтрите
# бас
f_b = 400 / (fs/2)
b_b, a_b = sig.iirfilter(n, f_b, btype='low', ftype='butter')
# средни
f_ml = 400 / (fs/2)
f_mh = 4000 / (fs/2)
b_m, a_m = sig.iirfilter(n, [f_ml, f_mh], btype='band', ftype='butter')
# трепбл
f_h = 4000 / (fs/2)
b_t, a_t = sig.iirfilter(n, f_h, btype='high', ftype='butter')

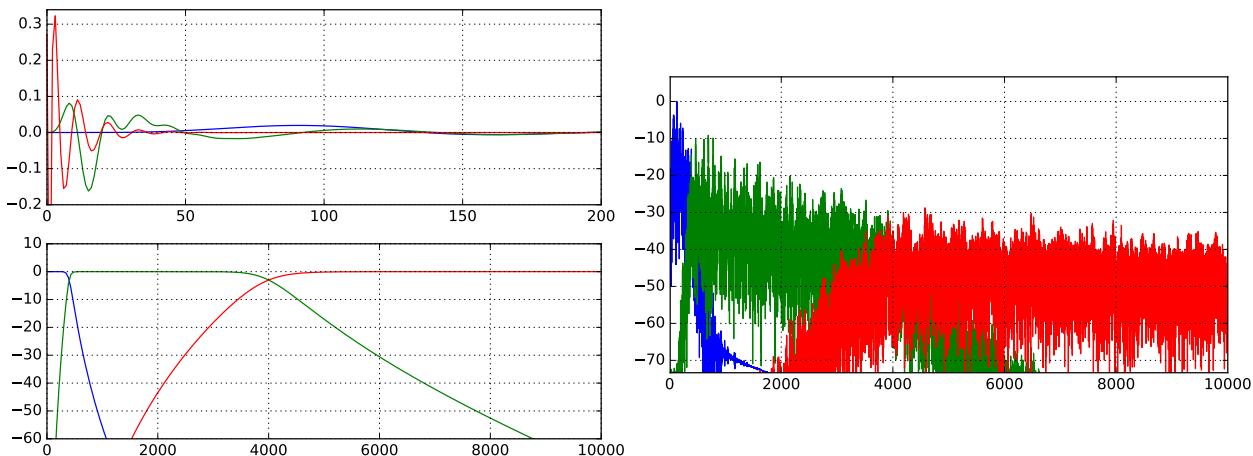
# преносни функции
w, H_b = sig.freqz(b_b, a_b)
f = w /pi * (fs/2)
H_b = 20 * np.log10(H_b)
w, H_m = sig.freqz(b_m, a_m)
H_m = 20 * np.log10(H_m)
w, H_t = sig.freqz(b_t, a_t)
H_t = 20 * np.log10(H_t)

# %% импулсни одсиви
x = np.zeros(1000)
x[0] = 1
h_b = sig.lfilter(b_b, a_b, x)
h_m = sig.lfilter(b_m, a_m, x)
h_t = sig.lfilter(b_t, a_t, x)

# %% плотирање
plt.figure()
plt.subplot(211)
plt.plot(h_b)
plt.plot(h_m)
plt.plot(h_t)
plt.axis([0, 200, -.2, .34])
plt.grid()
plt.subplot(212)
plt.plot(f, H_b)
plt.plot(f, H_m)
plt.plot(f, H_t)
plt.axis([0, 10000, -60, 10])
plt.grid()

# %% филтрирање
fs, wav = wavfile.read('audio/Mara.wav')
wav_bass = sig.lfilter(b_b, a_b, wav)
wav_mid = sig.lfilter(b_m, a_m, wav)
wav_treble = sig.lfilter(b_t, a_t, wav)

# %% засилување
g_bass = -20 # во dB
g_mid = 0 # во dB
g_treble = 20 # во dB
```



Сл. 4.5: Импулсни одсиви и фреквенциски карактеристики на трите IIR филтри од дизајнираниот еквализатор и добиени подопсези од аудиосигналот.

```

g_b = 10**(g_bass/20)
g_m = 10**(g_mid/20)
g_t = 10**(g_treble/20)
wav_out = g_b*wav_bass + g_m*wav_mid + g_t*wav_treble

# %% плотирање
f, wav_spec = da.get_spectrum(wav, fs)
f, wav_bass_spec = da.get_spectrum(wav_bass, fs)
f, wav_mid_spec = da.get_spectrum(wav_mid, fs)
f, wav_treble_spec = da.get_spectrum(wav_treble, fs)
plt.figure()
plt.plot(f, wav_spec)
plt.plot(f, wav_bass_spec)
plt.plot(f, wav_mid_spec)
plt.plot(f, wav_treble_spec)
plt.grid()

# %% преслушување
import os
wav_out = wav_out / np.max(np.abs(wav_out))
wav_out = wav_out * 2**15
wavfile.write('audio/Mara.eql.wav', fs, wav_out.astype('int16'))
os.system('play audio/Mara.wav')
os.system('play audio/Mara.eql.wav')

```

Импулсните одсиви, фреквенциските карактеристики на трите дизајнирани филтри и добиените подопсези од аудиосигналот се прикажани на Сл. 4.5.

4.6 IIR филтри непропусни на фреквенција

Една од примените на дигиталните филтри за која неприосновени се IIR филтрите е потиснувањето на одредена фреквенција во аудиосигналите. Најчесто потребата за ваков тип на процесирање се јавува кога треба да се потисне хармоничниот шум од градската мрежа на 50 Hz, познат како **зуење**, кој влегол во аудиосигналот пред неговата дигитализација. За таа цел потребно е филтерот непропусник да има што потесен опсег и да биде со што поголемо слабеење, односно голем **Q фактор**. Овие типови на филтри се нарекуваат **notch филтри**.

Глава 5

Дигитални аудиоэффекти

Аудиоэффектите, или звучните ефекти, се вештачки произведени или видоизменети звуци или звучни процеси кои се употребуваат за нагласување на уметничката или друг вид на содржина во филмската уметност, телевизиските емисии, радиото, живите изведби, анимацијата, видео игрите, музиката и другите типови медиуми.¹ Често под поимот аудиоэффекти се подразбира типот на процесирањето кое се применува врз звучните сигнали, а не и нивната содржина. Ова толкување ќе го задржиме во предметот Дигитални аудиосистеми. Грубо звучните ефекти може да ги поделиме на ефекти базирани на процесирање на сигналот во временски домен, и ефекти базирани на процесирање во трансформациски, на пр. фреквенциски, домен.

5.1 Аудиоэффекти во временски домен

Основните дигитални аудиоэффекти базирани на обработката на аудиосигналот во временски домен се оние кои вршат обработка на неговата амплитуда, па се нарекуваат **амплитудни аудиоэффекти**, и оние кои се изведуваат преку внесување на доцнење.

Амплитудни аудиоэффекти

Најпознати амплитудни аудиоэффекти се:

- постепено **засилување и втишување**² -- се употребува на почетокот односно на крајот од сегменти на аудиосигналот,
- **прелевање**³ -- се употребува за надоврзување на два аудиосигнали,
- **тремоло** -- периодична амплитудна модулација на аудиосигналот,
- **пинг-понг** -- периодична менување на просторната локација на изворот во дво- или повеќеканално аудио,
- **дисторзија** -- нелинеарна амплитудна карактеристика со пресекување на врвните амплитуди,
- **динамичка компресија** -- процесирање на аудиосигналот со нелинеарна амплитудна карактеристика која нелинеарно ги потиснува врвните амплитуди, итн.

✓ **Задача за дома.** Со помош на досегашните познавање на обработката на аудиосигналите реализирај тремоло ефект со фреквенција f од 2 Hz и засилување G од 0.5.

¹Wikipedia -- Sound effect https://en.wikipedia.org/wiki/Sound_effect

²Анг. *fade-in* и *fade-out*.

³Анг. *cross-fade*.

Основи на процесирање со задоцнување

Голем број на дигитални аудиоэффекти се базираат на генерирање на задоцнети верзии од аудиосигналот и нивно додавање на оригиналниот сигнал. Двата основни аудиоэффекти кои се реализираат на овој начин се **ехото** и **реверберацијата**. Пред појавата на дигиталниот звук, овие ефекти биле правени со електро-механички склопови, некои од нив големи колку и цела просторија. Во дигитален домен тие се генерираат со хардверска, а почесто софтверска, обработка на сигналите. Одсивот на систем кој генерира задоцната верзија на аудиосигналот и истата ја додава на него може да го претставиме со помош на следната диференцна равенка:

$$y[n] = x[n] + b_D x[n - D]. \quad (5.1)$$

Може да се види дека оваа диференцна равенка претставува специјален случај на општата диференцна равенка (4.8). Тука D е доцнењето на сигналот во број на примероци, b_D е неговото слабеење, а земено е дека $b_0 = 1$. Импулсниот одсив и преносната функција на овој систем се дадени со:

$$h[n] = 1 + b_D \delta[n - D], \quad (5.2)$$

$$H(z) = \frac{Y(z)}{X(z)} = 1 + b_D z^{-D}. \quad (5.3)$$

Поради конструктивното и деструктивно собирање на задоцната верзија од аудиосигналот со неговиот оригинал за различни фреквенции, фреквенциската карактеристика на системот ќе има низа максимуми и минимуми распоредени на еднакво растојание. Поради овој облик овие системи се нарекуваат уште и **чешлести филтри**. Ваквата фреквенциска карактеристика е неповолна за обработка на аудиосигналите поради спектралните изобличувања кои ќе бидат внесени во сигналот.

За реализација на повеќекратни задоцнети верзии од аудиосигналот постојат два можни пристапи. Едниот е да се додадат нови ненулти коефициенти во диференцната равенка на FIR филтерот дадена во (5.1). Имаме:

$$y[n] = x[n] + b_{D_0} x[n - D_0] + b_{D_1} x[n - D_1] + \dots + b_{D_{M-1}} x[n - D_{M-1}], \quad (5.4)$$

$$h[n] = 1 + b_{D_0} \delta[n - D_0] + b_{D_1} \delta[n - D_1] + \dots + b_{D_{M-1}} \delta[n - D_{M-1}], \quad (5.5)$$

$$H(z) = 1 + b_{D_0} z^{-D_0} + b_{D_1} z^{-D_1} + \dots + b_{D_{M-1}} z^{-D_{M-1}} = 1 + \sum_{m=0}^{M-1} b_{D_m} z^{-D_m}. \quad (5.6)$$

Тука, со M е означен бројот на задоцнети верзии додадени во сигналот одредени со нивните доцнења D_m и коефициенти на слабеење b_{D_m} .

Вториот пристап се базира на употреба на IIR филтри кои по својата природа имаат бескрајно долг импулсен одсив а со тоа и повеќекратни задоцнети верзии на сигналот кои се распоредени на мултипи од доцнењето $m \times D$ на првото доцнење. Слабеењето на овие задоцнети верзии претставува степен од слабеењето на првото a_D^m .

$$y[n] = x[n] - a_D y[n - D] \quad (5.7)$$

$$H(z) = \frac{1}{1 + a_D z^{-D}} \quad (5.8)$$

Како и претходно фреквенциската карактеристика на овој IIR филтер е од чешлест облик, па внесува изобличувања во излезниот аудиосигнал. Овојпат, максимумите се на местата на минимумите кај FIR филтерот и обратно, минимумите кај IIR филтерот се на локациите на максимумите на FIR филтерот.

Последниот податок можеме да го искористиме за дизајн на систем кој би генерирал задоцнети верзии од аудиосигналот а притоа би имал рамна фреквенциска карактеристика која не би

внесувала изобличувања во излезниот сигнал. Ова може да се направи преку едноставна комбинација на комплементарните FIR и IIR системи дадена со следните равенства:

$$y[n] = b_D x[n] + x[n - D] - b_D y[n - D], \quad (5.9)$$

$$H(z) = \frac{b_D + z^{-D}}{1 + b_D z^{-D}}. \quad (5.10)$$

Ваквиот систем и покрај генерирањето на повеќекратни еха има амплитудна карактеристика еднаква на 1, поради што се нарекува **филтер сепропусник**. Филтрите сепропусници се користат во генерирањето на најразлични аудиоэффекти базирани на доцнење.

Дигитално echo

Дигиталното echo претставува ефект во кој на аудиосигналот му се додаваат една или повеќе задоцнети верзии од него самиот, при што меѓу тие задоцнети верзии и оригиналниот сигнал може да се направи јасна разлика. Во случај кога имаме повеќе задоцнети верзии велиме дека се работи за **повеќекратното echo**. За реализација на ехото ќе ги искористиме FIR и IIR филтрите кои ги разгледавме, како и филтрите сепропусници на опсег. При употреба на IIR филтер или сепропусник можеме да кажеме дека системот генерира **бескрајно echo**.

Најпрвин да ги увеземе потребните модули и пакети.

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.io import wavfile
from scipy import signal as sig
import os
```

Сега да ги имплементираме овие три типови на системи.

```
fs = 22050

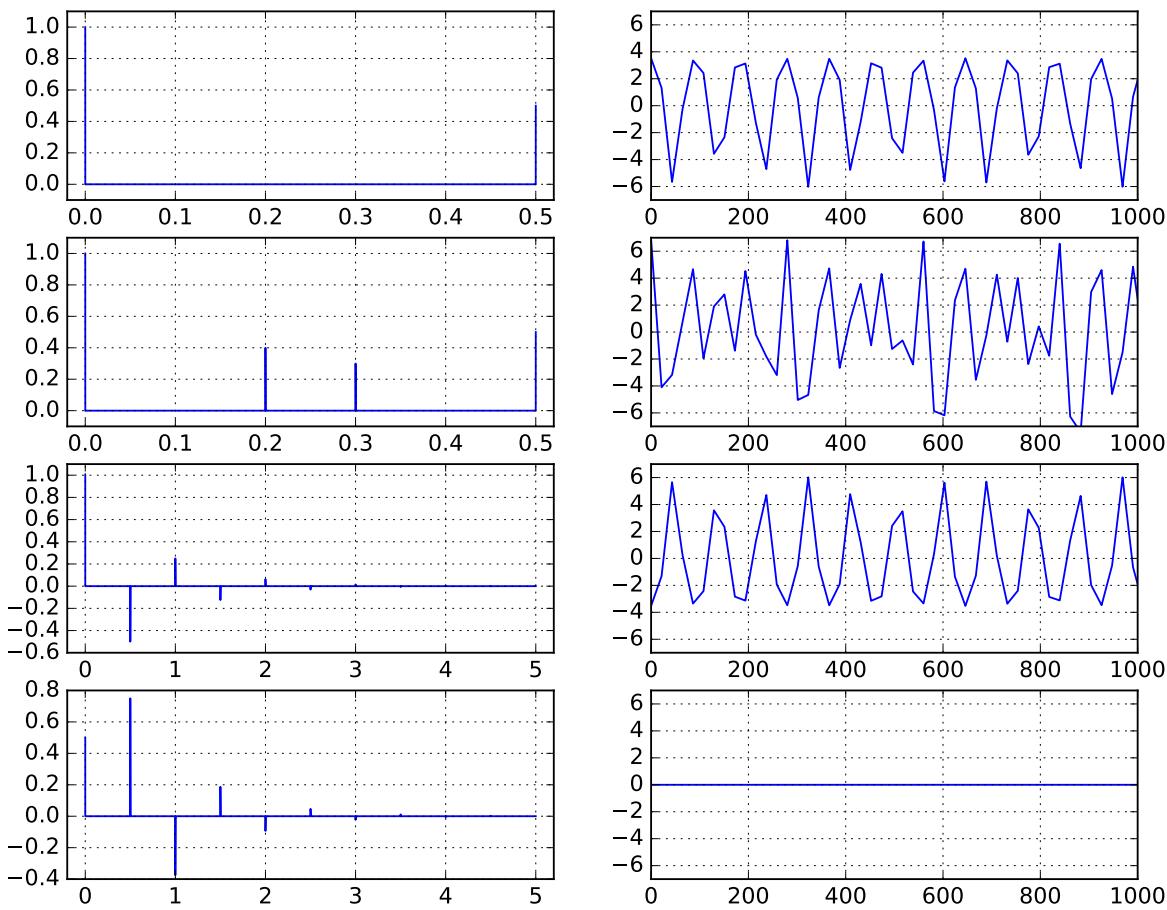
# FIR echo
Dt = 0.5 # sec
D = int(Dt*fs) # samples
b_D = 0.5
b_fir = np.zeros(D+1)
b_fir[0] = 1
b_fir[D] = b_D

# повеќекратно FIR echo
D0t = 0.2
D0 = int(D0t*fs) # samples
D1t = 0.3
D1 = int(D1t*fs) # samples
b_fir_mul = cp.copy(b_fir)
b_fir_mul[D0] = 0.4
b_fir_mul[D1] = 0.3

# бескрајно echo
a_iir = np.zeros(D+1)
a_iir[0] = 1
a_iir[D] = b_D

# сепропусник
b_ap = np.zeros(D+1)
b_ap[0] = b_D
b_ap[D] = 1
a_ap = cp.copy(a_iir)
```

Следно ќе ги пресметаме импулсните и фреквенциските одсиви на овие четири системи.



Сл. 5.1: Импулсни и фреквенциски одсиви на четирите системи за генерирање на ехо (од горе надолу): FIR филтер -- единично ехо, FIR филтер -- повеќекратно ехо, IIR филтер -- бесконечно ехо и филтер сепропусник.

```
# FIR
w, H_fir = sig.freqz(b_fir, [1])
f = w / pi * fs/2
H_fir = 20*np.log10(np.abs(H_fir))

# FIR multiple
w, H_fir_mul = sig.freqz(b_fir_mul, [1])
H_fir_mul = 20*np.log10(np.abs(H_fir_mul))

# IIR
x = np.zeros(5*fs)
x[0] = 1
h_iir = sig.lfilter([1], a_iir, x)
w, H_iir = sig.freqz([1], a_iir)
H_iir = 20*np.log10(np.abs(H_iir))

# AP
h_ap = sig.lfilter(b_ap, a_ap, x)
w, H_ap = sig.freqz(b_ap, a_ap)
H_ap = 20*np.log10(np.abs(H_ap))
```

Пресметаните импулсни и фреквенциски одсиви на четирите системи за генерирање на ехо се претставени на Сл. 5.1. Може да се види дека првите три системи навистина имаат чешлеста амплитудна карактеристика, додека филтерот сепропусник има рамна карактеристика со амплитуда од 0 dB.

Конечно, да ги искористиме имплементираните системи за процесирање на еден аудиосигнал и да ги чуеме резултатите.

```
#%% филтрирање
fs, wav = wavfile.read('audio/Pato_22K.wav')
wav_echo = sig.lfilter(b_fir, [1], wav)
wav_echo_mul = sig.lfilter(b_fir_mul, [1], wav)
wav_echo_iir = sig.lfilter([1], a_iir, wav)
wav_echo_ap = sig.lfilter(b_ap, a_ap, wav)
```

За разлика од досега, овој пат ќе ги искористиме функцијата за преслушување на аудиосигнали од пакетот `sounddevice`⁴.

```
#% преслушување
import sounddevice as sd
sd.play(wav_echo, fs)
sd.play(wav_echo_mul, fs)
sd.play(wav_echo_iir, fs)
sd.play(wav_echo_ap, fs)
```

Дигитална реверберација

Дигиталната реверберацијата или **ревербот** претставува ефект во кој се генерираат и наддодаваат многу повеќе задоцнети верзии од аудиосигналот со што се постигнува нивно аудиторно слевање во еден здружен одек. Реверберацијата е таа која му дава просторност на звукот, односно преку нејзе можеме да заклучиме во каков вид на просторија го слушаме или е снимен звучниот сигнал. Реверберацијата може да се реализира со едноставно паралелно или сериско надоврзување на повеќе системи за генерирање на еднократно или повеќекратно ехо со густо распоредени доцнења.

Други дигитални ефекти базирани на доцнење

Други аудиоэффекти базирани на употреба на доцнење или сепропусни филтри се: **хор**, **фленџ** и **фејзер** ефектите. Ефектот **хор**⁵ е ефект во кој на аудиосигналот му се додаваат една или повеќе негови верзии добиени со променливо но мало задоцнување. На тој начин се постигнува впечаток дека наместо еден музички извор, во аудиосигналот постојат повеќе извори кои се релативно добро усогласени, но на моменти приметно разгодени.

Фленџ⁶ ефектот се добива преку примена на FIR систем за еднократно ехо кое има променливо доцнење. Аудитивно овој ефект е сличен на фејзерот описан во Поглавјето , кој пак може да се реализира со каскада од променливи сепропусни филтри. Разликата меѓу двата ефекта е во тоа што фленџот се базира на чешлестиот облик на фреквенциската карактеристика на FIR филтерот, па генерира максимуми и минимуми во излезниот сигнал кои се во хармониски сооднос, односно се мултикли од основниот максимум. Од друга страна, ова не е случај кај фејзерот.

5.2 Аудиоэффекти во фреквенциски домен

Аудиоэффекти базирани на филтри

Освен еквализаторите, кои исто така можат да се сметаат за дигитални аудиоэффекти, најпознатите дигитални аудиоэффекти базирани на филтри се **ва-ва** и **фејзерот**.

Ва-ва ефектот⁷ е првенствено направен за изведба на музика на електрична гитара, иако за првпат го пронашле трубачите и тромбонистите во 1920^{te}. Тој се изведува со помош на филтер пропусник на опсег чија централна фреквенција се менува со време а под контрола на свирачот преку потенциометар поставен во педала за дозирање. Аудиосигналот кој филтерот го дава на

⁴sounddevice е Python модул кој овозможува преслушување и снимање на звук во NumPy низи преку употреба на PortAudio библиотеката. <https://python-sounddevice.readthedocs.io/en/0.4.1/>

⁵Wikipedia: Chorus effect. https://en.wikipedia.org/wiki/Chorus_effect

⁶Wikipedia -- Flanging. <https://en.wikipedia.org/wiki/Flanging>

⁷Wikipedia: Wah-wah (music). https://en.wikipedia.org/wiki/Wah-wah_%28music%29

излез се меша со оригиналниот сигнал за да се добие крајниот ефект. Во електронската музика педалата може да биде заменета од **нискофреквенциски осцилатор (LFO)**⁸.

Да го имплементираме ва-ва ефектот во Python. Најпрвин ќе го вчитаме аудиосигналот `Solzi.wav` и ќе го нормализираме.

```
from os.path import join as pjoin

import numpy as np
from matplotlib import pyplot as plt
from scipy import signal as sig
from scipy import fftpack as fp
from scipy.io import wavfile as wf
import sounddevice as sd

import da

audio_path = "audio"
wav_name = "Solzi.wav"
file_name = pjoin(audio_path, wav_name)
fs, wav = wf.read(file_name)
wav = wav / 2**15
wav = da.normalise(wav)
sd.play(wav, fs)
```

Следно, ќе ги дефинираме параметрите на филтерот и нискофреквенцискиот осцилатор кој ќе го користиме за да го филтрираме аудиосигналот и тоа:

- редот на филтерот `order`,
- фреквенцискиот опсег на филтерот `band`,
- почетната централна фреквенција филтерот `fc_offset`,
- амплитудата на осцилациите на централната фреквенција `fc_amp` и
- фреквенцијата на осцилација на централната фреквенција `fc_f`.

```
order = 5
band = 1000 # Hz
fc_offset = 2000 # Hz
fc_amp = 1000 # Hz
fc_f = 2 # Hz
```

Сега ќе го примениме филтерот на аудиосигналот со примена на методата на прозорци, со тоа што за секој прозорец ќе ја пресметаме централната фреквенција на филтерот, ќе ги пресметаме неговите коефициенти и ќе ги филтрираме таа рамка од аудиосигналот. Филтрираните рамки ќе ги додаваме на излезниот вектор `wav_wah` со примена на методата на **преклопување**⁹ во која секоја рамка се додава на излезот преклопена со претходната рамка. Должината на преклопувањето е одредена од должината на прозорецот и големината на скокот со кој тој се поместува. Преклопувањето е важно за да се избегне појавата на артефакти при надоврзувањето на излезните рамки.

```
# apply
t_win = 0.040 # s
n_win = int(t_win * fs)
n_hop = n_win // 2
win = sig.get_window("hann", n_win)
n = wav.size
pos = 0
wav_wah = np.zeros(wav.size)
```

⁸анг. *Low Frequency Oscillator*.

⁹анг. *overlap-add*.

```

while pos <= n - n_win:
    frame = wav[pos: pos + n_win]
    frame = frame * win
    # design filter
    t_frame = pos / fs
    fc = fc_offset + fc_amp * np.sin(2 * np.pi * fc_f * t_frame)
    fl = fc - band/2
    fh = fc + band/2
    b, a = sig.iirfilter(order, [fl, fh], fs=fs)
    frame_filt = sig.lfilter(b, a, frame)
    # overlap add
    wav_wah[pos: pos + n_win] = wav_wah[pos: pos + n_win] + frame_filt
    pos = pos + n_hop

wav_wah = da.normalise(wav_wah)
sd.play(wav_wah, fs)

```

Конечниот резултат ќе го добијеме со мешање на оригиналниот и филтрираниот сигнал преку нивно соодветно дозирање.

```

wav_mix = 0.3 * wav + 0.7 * wav_wah
sd.play(wav_mix, fs)

```

Можеме јасно да ги слушнеме разликите помеѓу оригиналниот сигнал и сигналот кој е филтриран со ва-ва филтерот. За да го видиме влијанието на филтрирањето на сигналот во фреквенциски домен, ќе ги прикажеме спектограмите на оригиналниот, филтрираниот и конечниот мешан сигнал.

```

-- = da.get_spectrogram(wav, fs, plot=True)
-- = da.get_spectrogram(wav_wah, fs, plot=True)
-- = da.get_spectrogram(wav_mix, fs, plot=True)

```

На спектограмите, прикажани на Сл. 5.2, јасно може да се забележи дека примената на ва-ва филтерот генерира траг во спектарот на сигналот кој се движи во фреквенциски домен.

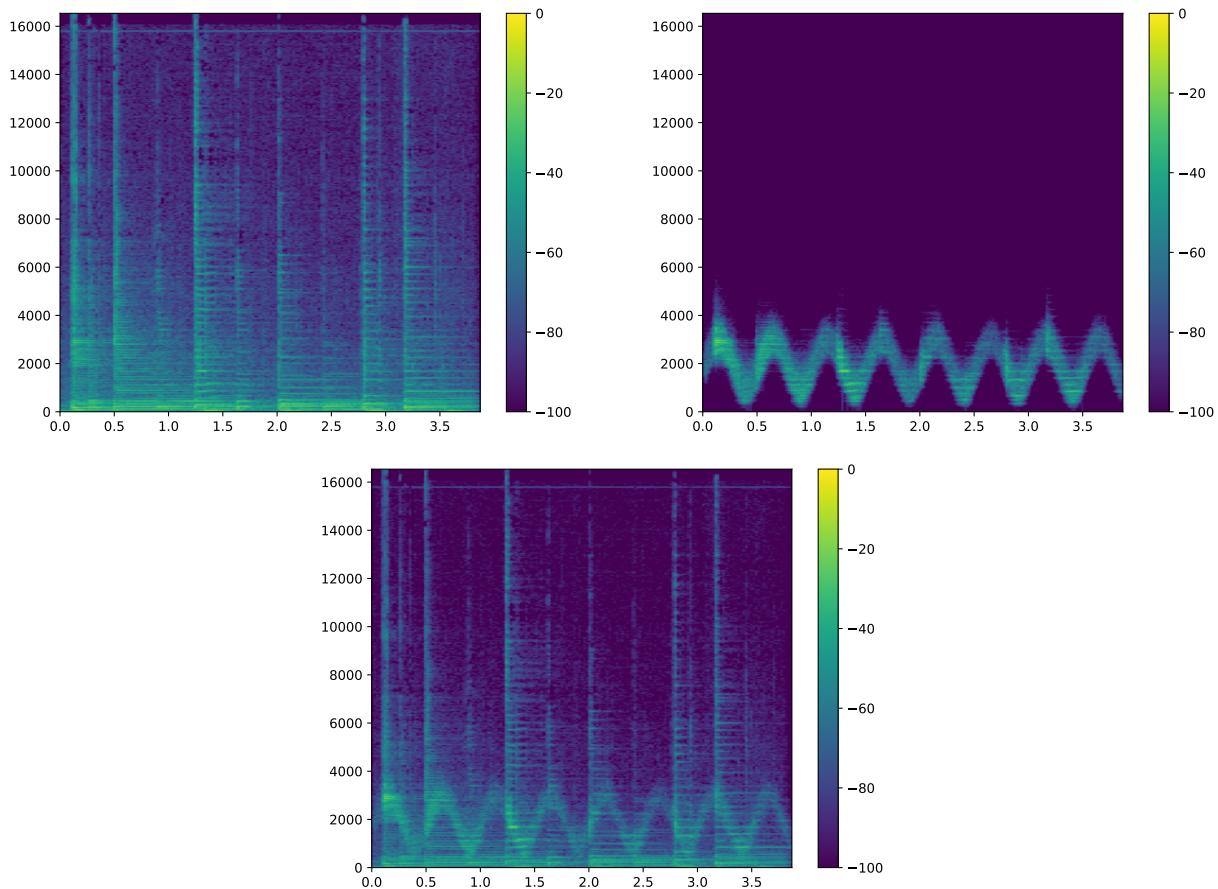
Фејзерот¹⁰ исто така претставува гитарски ефект добиен со употреба на каскада на notch филтри чии што фреквенции на потиснување периодично се менуваат. На овој начин тие вршат селективно слабеење на делови од спектарот на сигналот кое може да се види како траг во спектограмот. Повторно менувањето на фреквенцијата на овие филтри може да биде направена од свирачот преку педала со потенциометар или со употреба на филтер непропусник на опсег. Дополнително фејзерот може да се реализира и со сепропусни филтри дискутиирани во поглавјето .

Аудиоэффекти базирани на Фурьеова Трансформација

Како пример за аудиоэффект базиран на Фурьеова трансформација ќе го имплементираме ефектот на [роботизација](#)¹¹. Овој ефект се добива со примена на Фурьеова трансформација на аудиосигналот и методата на прозорци. За секоја рамка од аудиосигналот ќе ја пресметаме Фурьеовата трансформација и ќе ја замениме фазата на сите фреквенции со 0. Потоа ќе ја вратиме рамката во временски домен со примена на инверзна Фурьеова трансформација. На тој начин, сите фреквенциски компоненти ќе бидат во фаза една со друга на самиот почеток од рамката во $t = 0$, па во тој временски момент ќе добијеме изразен врв во амплитудата на сигналот поради нивното конструктивно собирање. Вака добиените рамки повторно ќе ги додадеме на излезниот сигнал со примена на методата на преклопување. Периодичната низа на врвови добиена со надворздување на рамките ќе му даде роботизиран звук сигнал.

¹⁰Wikipedia: Phaser (effect). https://en.wikipedia.org/wiki/Phaser_%28effect%29

¹¹Wikipedia: Robotic Voice Effects. https://en.wikipedia.org/wiki/Robotic_voice_effects



Сл. 5.2: Спектрограми на оригиналниот сигнал (горе лево), филтрираниот сигнал (горе десно) и мешаниот сигнал (долу) со ва-ва филтерот.

```

from os.path import join as pjoin

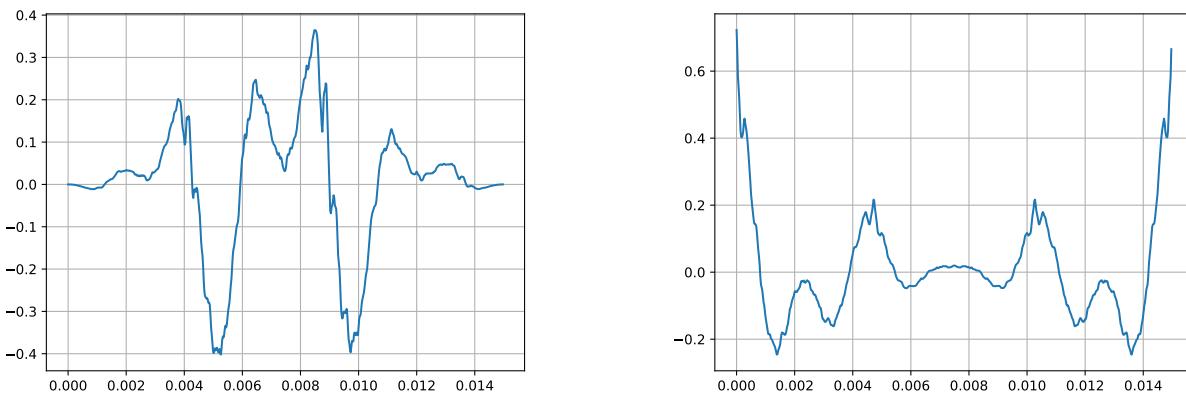
import numpy as np
from matplotlib import pyplot as plt
from scipy import signal as sig
from scipy import fftpack as fp
from scipy.io import wavfile as wf
import sounddevice as sd

import da

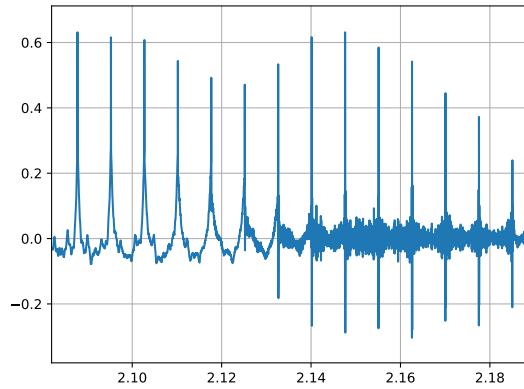
# %% load audio
audio_path = "audio"
wav_name = "Glas.wav"
file_name = pjoin(audio_path, wav_name)
fs, wav = wf.read(file_name)
wav = wav[: int(3.65 * fs)] / 2**15
sd.play(wav, fs)

# %% apply
t_win = 0.015 # s
n_win = int(t_win * fs)
n_hop = n_win // 2
win = sig.get_window("hann", n_win)
n = wav.size
pos = 0
wav_robot = np.zeros(wav.size)
while pos <= n - n_win:
    frame = wav[pos: pos + n_win]

```



Сл. 5.3: Временски облик на една рамка од аудиосигналот (лево) и нејзиниот облик при нулирање на фазата (десно).



Сл. 5.4: Временски облик на сигналот добиен со надоврзување на процесираните рамки.

```

frame = frame * win
frame_fft = fp.fft(frame)
frame_amp = np.abs(frame_fft)
frame_inv = fp.ifft(frame_amp).real
# overlap add
wav_robot[pos: pos + n_win] = wav_robot[pos: pos + n_win] + frame_inv
pos = pos + n_hop

wav_robot = da.normalise(wav_robot)
sd.play(wav_robot, fs)

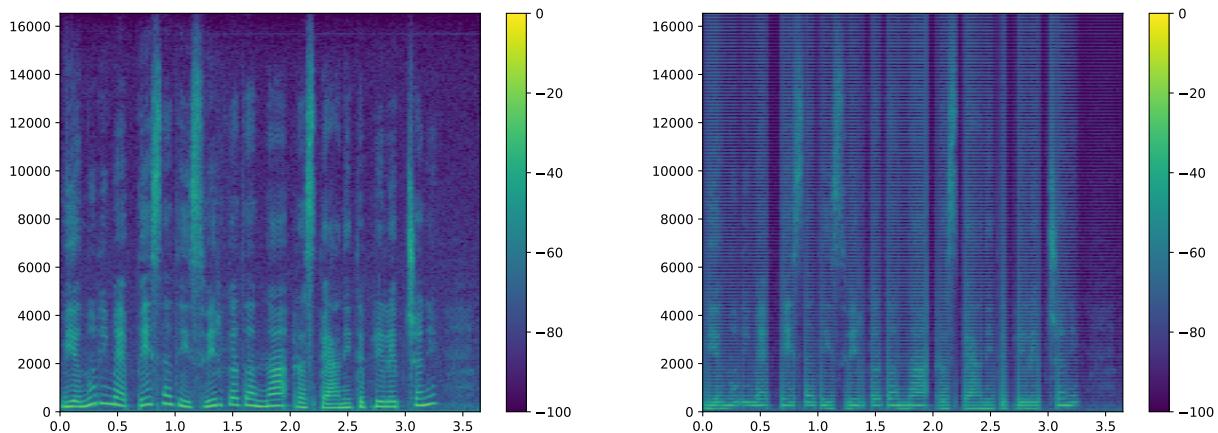
# %% plots
-- = da.get_spectrogram(wav, fs, plot=True)
-- = da.get_spectrogram(wav_robot, fs, plot=True)

```

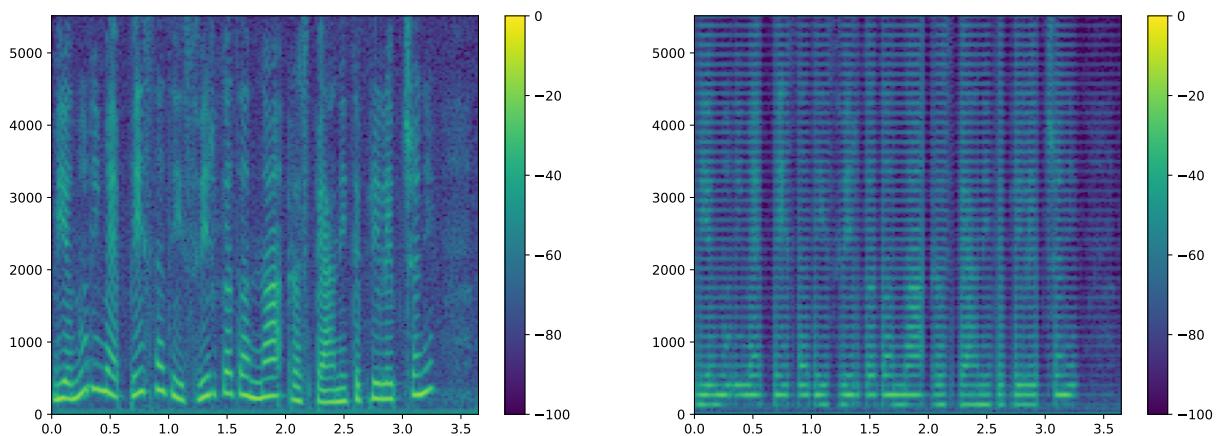
Ефектот на нулирање на фазата на фреквенциите во Фурьеовата трансформација може да се види во временскиот облик на една од рамките на аудиосигналот, прикажана на Сл. 5.3.

Ефектот на надоврзување на вака добиените рамки може да го видиме на Сл. 5.4.

На Сл. 5.5 може да се види дека примената на овој ефект резултира во сигнал со многу различен спектар од оригиналниот сигнал. Така, додека во оригиналниот сигнал може да се движење на основната фреквенција на гласот, која се нарекува интонација, и нејзините хармоници, во роботизираниот сигнал може да се види дека основната фреквенција е константна и сите хармоници испртуваат паралелни хоризонтални линии во спектрограмот. Ова е повоочливо



Сл. 5.5: Спектрограми на оригиналниот сигнал (лево) и роботизираниот сигнал (десно).



Сл. 5.6: Зголемен дел од спектрограмите за ниските фреквенции на оригиналниот сигнал (лево) и роботизираниот сигнал (десно).

ако го зголемиме приказот на спектрограмот на двата сигнали за ниските фреквенции како на Сл. 5.6.

✓ **Задача за час.** Од кои параметри зависи основната фреквенција на роботизираниот сигнал? Пресметајте ја!

Глава 6

Компресија на звукот

Поради големиот габарит на сировото импулсно кодно модулираниот (PCM) звук, а од друга страна ограничениот мемориски простор на персоналните компјутери, односно ограничениот пропусен опсег на телекомуникациските врски за вмрежување, се пристапува кон компресија на аудиодатотеките. Постојат различни методи со кои може да се изврши компресија на дигиталниот звук (Spanias et al., 2006). Во зависност од методот на компресија дефинирани се и различни формати на компресирано дигитален звук. Постојат две основни групи на методи за компресија:

- методи без загуби¹ и
- методи со загуби.²

Методите за компресија без загуби не исфрлаат никакви информации од оригиналниот звук. Овие методи не се многу различни од општите методи за компресија. Сепак тие успеваат да ја намалат големината на звучните записи само на 50 -- 60% од онаа на оригиналната датотека. Ова се должи на комплексноста на аудиосигналите, а зависи од содржината на аудиосигналот. Така, говорните сигнали, кои имаат помала комплексност и вклучуваат тишина, трпат поголема компресија отколку музиката.

Најупотребуваниот формат од овој тип е Free Lossless Audio Codec (FLAC)³ кој претставува слободен стандард за компресија базиран на линеарна предикција. Други формати од оваа категорија се: WavPack, Shorten, Monkey's Audio, ATRAC Advanced Lossless, Apple Lossless, WMA Lossless, TTA, итн.

Методите со загуби ја намалува големината на дигиталниот запис жртвувајќи дел од неговиот квалитет. Најчесто тој дел човековото уво и не може да го чуе поради различни акустички феномени, а во најголема мера поради прагот на чујност и маскирањето. Овој вид на компресија се прави со употреба на психоакустички модели и исфрлање на непотребните информации од звучниот запис (Painter and Spanias, 2000). На тој начин тие постигнуваат смалување на големината на аудиодатотеките 10, па и повеќе пати од големината на некомпресираната датотека, односно до 5 -- 20% од оригиналната големина на датотеката. Така на пример, стандардната мп3 компресија го намалува битскиот проток од номиналните 1411 kb/s на аудио-CD-то до 128 kb/s, а при тоа да се зачува субјективното чувство за еднаков квалитет кај слушателот.

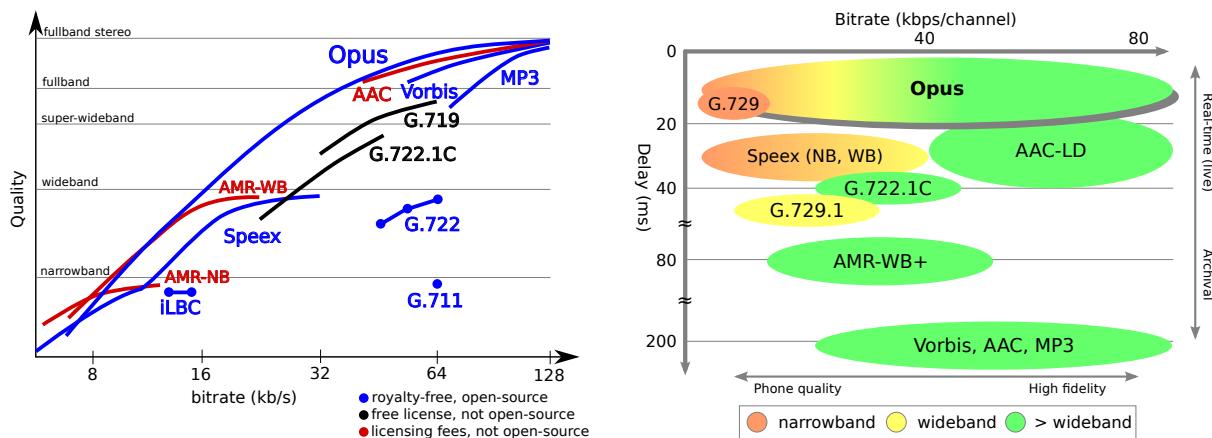
Најдобриот алгоритам за компресија на звук со загуби во моментов е слободниот Opus⁴ формат. Тој не само што постигнува подобар субјективен квалитет, оценет преку двоен слеп тест, од сите останати формати за сите излезни битски брзини, туку и постигнува најдобри перформанси во

¹Анг. *lossless*.

²Анг. *lossy*.

³Free Lossless Audio Codec (FLAC) <https://xiph.org/flac/>

⁴Wikipedia -- Opus (audio format) [https://en.wikipedia.org/wiki/Opus_\(audio_format\)](https://en.wikipedia.org/wiki/Opus_(audio_format))



Сл. 6.1: Перформанси на Opus стандардот за аудиокомпресија во однос на квалитет за даден битски проток (лево) и латентност за даден фреквенциски опсег на сигналот (десно) споредено со други формати за аудиокомпресија.

однос на латентноста со стандардно доцнење од 26,5 ms кое овозможува комуникација во реално време, споредено со 200 ms на mp3 стандардот. Со жртвување на излезниот битски проток ова време може да се намали до 5 ms. Исто така, Opus нуди оптимизација за имплементација во вгнездените компјутерски системи. Конечно, Opus е слободен стандард и сите патенти кои го покриваат алгоритамот се бесплатни. Перформансите на Opus споредени со останатите алгоритми за компресија со загуби се дадени на Сл. 6.1.

На Сл. 6.2 се прикажани спектрограмите на еден ист аудиосигнал компримиран со различни битскиprotoци со различните формати. Може да се види дека навистина Opus има забележливо подобри перформанси од останатите формати, особено за ниските битскиprotoци.

Opus се базира и го заменува слободниот формат за компресија на музички сигнали **Vorbis** (OGG)⁵. Тој исто така сосема го заменува слободниот **Speex**⁶ формат, наменет за компресија на говор. Компресијата на музичките сигнали во Opus се базира на слободниот CELT⁷ алгоритам, кој пак е базиран на модифицираната дискретна косинусна трансформација (MDCT). Компресијата на говорните сигнали пак, се базира на слободниот формат за компресија на говорни сигнали **SILK**⁸.

Други познати формати за компресија на звук со загуби се и затворените Advanced Audio Coding (AAC) дефиниран во MPEG-2 и High Efficiency AAC (HE-AAC) дефиниран во MPEG-4, како и Windows Media Audio (WMA). Со истекување на патентот на Техниколор за mp3 во САД на 16. април 2017, mp3 се придржува на слободните стандарди за компресија.

6.1 MPEG-1 ниво III

Како еден од најраспространетите претставници на групата алгоритми за компресија на звук со загуби ќе ги разгледаме основите на MPEG-1 ниво III, односно mp3 алгоритмот. Тој е главниот стандард за пренос и зачувување на компресиран звук, како на интернет мрежата, така и на персоналните компјутери, во преносливите мултимедијални уреди итн. Иако денес постојат поразвиени алгоритми за компресија на аудиото, главните придобивки кои тие ги носат се за ниските битски брзини, т.е. за поголемите степени на компресија. Над 128 kbit/s квалитетот на компресија со mp3 стандардот е на доволно високо ниво и по денешните стандарди, па тој сеуште го задржува притатот во овој домен.

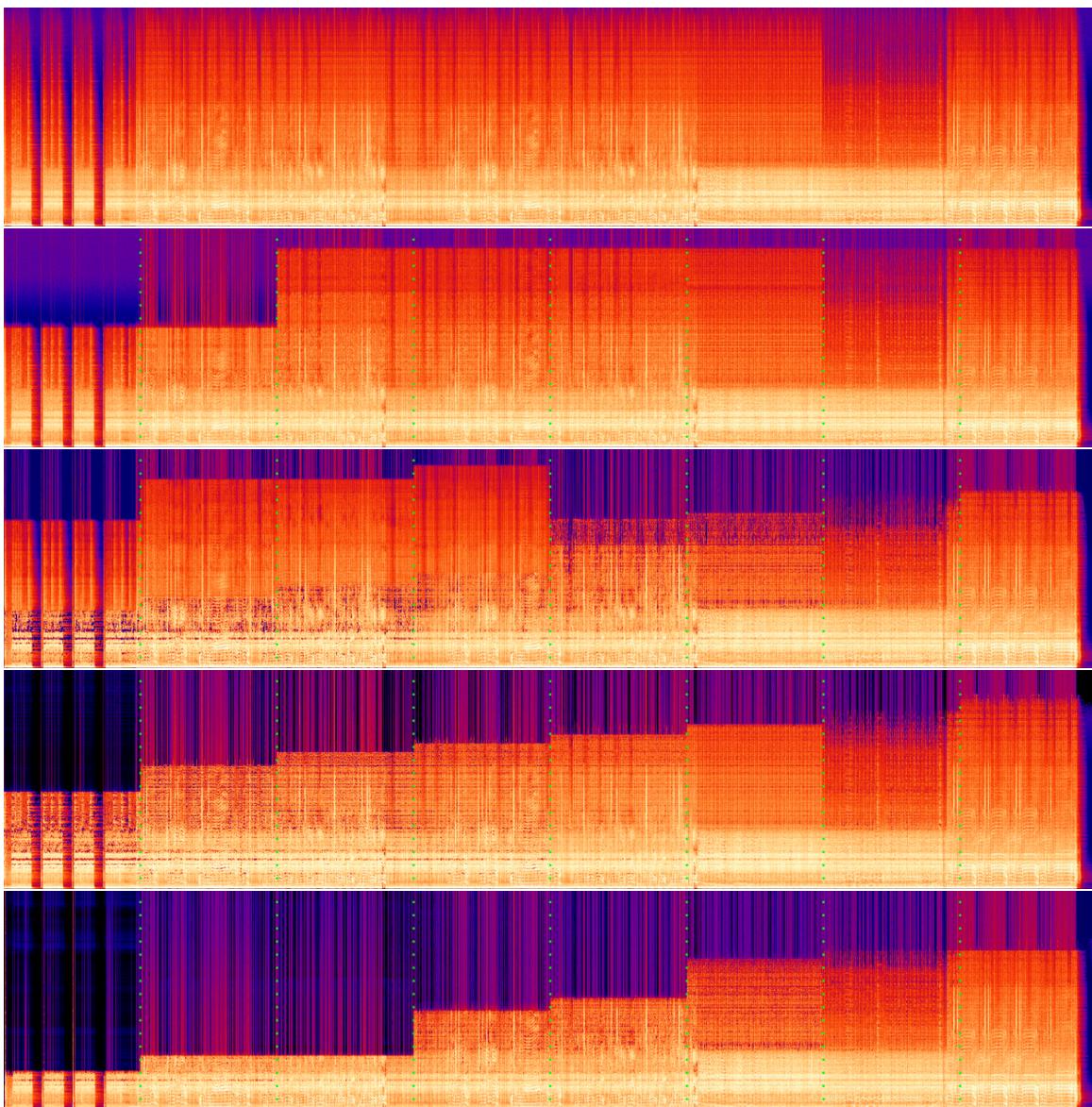
MPEG (Moving Pictures Experts Group) е експертска група чиишто главни задачи се:

⁵Vorbis audio compression. <https://xiph.org/vorbis/>

⁶Wikipedia -- Speex <https://en.wikipedia.org/wiki/Speex>

⁷Wikipedia -- Constrained Energy Lapped Transform (CELT) <https://en.wikipedia.org/wiki/CELT>

⁸Wikipedia -- SILK <https://en.wikipedia.org/wiki/SILK>



Сл. 6.2: Спектрограми на ист аудиосигнал компримиран со различни битски протоци со различните формати за компресија на звук, од горе надолу: оригиналниот сигнал, Opus, AAC, Vorbis и mp3.

- да објавува технички резултати и извештаи поврзани со компресија на звук и видео,
- да одреди начин за мултиплексирање на видео, звук и информациски протоци во единствен проток,
- да даде описи и синтакса за алатки за компресија на звук и видео до ниски протоци за Интернет апликации и апликации кои работат со ограничен опсег.

MPEG стандардите не даваат точни спецификации за реализација на кодерот, туку го дефинираат типот на информациите кои тој треба да ги даде, како и начинот на кој декодерот треба да ги протолкува при декомпресијата. До сега се објавени 5 различни MPEG стандарди поврзани со дигиталниот звук:

- MPEG-1,
- MPEG-2 BC (Backwards-Compatible),
- MPEG-2 NBC/AAC (Non-Backward Compatible/Advanced Audio Coding),
- MPEG-4 и

- MPEG-7,
- MPEG-21.

Од нив последните два не се стандарди за компресија. MPEG-7 дефинира дескриптори на звучните/видео содржини, со чија помош може да се описат за побрз пристап до нив во бази на податоци. MPEG-21 дефинира мултимедијална рамка и нуди менацирање и заштита на интелектуална сопственост.

Два различни термини се поврзани со MPEG стандардите, тоа се: фаза и ниво. Фазите одговараат на главниот тип на MPEG аудиостандардот: MPEG-1, MPEG-2, MPEG-4 итн. Нивоата означуваат фамилии на алгоритми за кодирање внатре во MPEG фазата. Нивоа се дефинирани само во MPEG-1 и MPEG-2 и тоа:

- MPEG-1 ниво-I, -II и -III,
- MPEG-2 ниво-I, -II и -III.

MPEG-1 Аудио (ISO/IEC 11172-3) стандардот е првиот аудиостандард, објавен од MPEG групата во 1992, по четиригодишна напорна работа на усогласено истражување на светските експерти од областа на аудиокомпресијата. Неговата намена била да се овозможи стерео-CD-квалитет. MPEG-1 подржува стерео аудио-CD квалитет на 192 kbit/s. Тоа го достигнува со примена на флексибилна техника за хибриден компресија на аудиото која се базира на сплет од неколку методи и тоа:

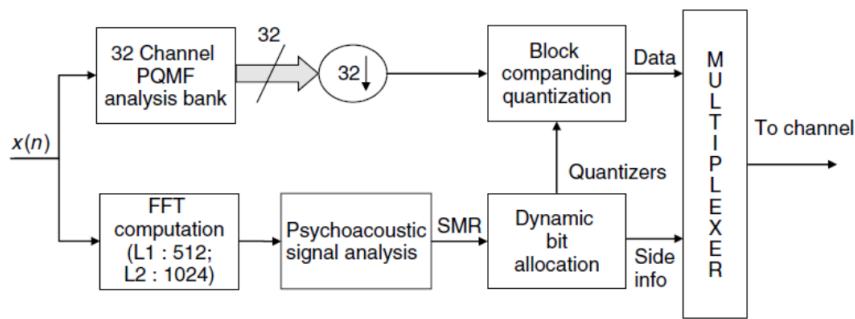
- подопсежно разлагање,
- анализа со банки на филтри,
- психоакустичка анализа,
- адаптивна сегментација,
- трансформациско кодирање,
- динамичко доделување на битови,
- неуниформна квантизација и
- ентрописко кодирање.

MPEG-1 аудиокодекот работи со 16-битен PCM влез со f_s од 32, 44,1 и 48 kHz. Тој нуди различни модови на работа за моно, стерео, двојно моно и здружено (joint) стерео. Протоците на компресираниот звук се дефинирани во опсег 32 – 192 kbit/s за моно и 64 – 384 kbit/s за стерео.

MPEG-1 архитектурата содржи три нивоа со растечка комплексност, доцнење и квалитет на компресираниот сигнал. Секое повисоко ниво ги вклучува функционалните блокови од претходните.

Во нивото II, прикажано на [Сл. 6.3](#), влезниот сигнал се разложува на 32 подопсези со намалена фреквенција на земање примероци добиени со употреба на **банка на филтри** од типот PQMF (Pseudo Quadrature Mirror Filter). Каналите се еднакво распоредени на тој начин што влезен сигнал со фреквенција на земање примероци од 48 kHz се разлага на подопсези од по 750 Hz, а подопсезите се децимирани (подземање примероци) со однос 32:1. Прототипниот филтер е од 511^{ви} ред така што вкупната дисторзија, карактеристична за PQMF банките на филтри, останува под прагот на чујноста, а слабеење во непорпусниот дел од спектарот од 96 dB осигурува занемарливо меѓуопсежко влијание.

Целта на **психоакустичката анализа** е определување на **праговите на приметлива дисторзија JND** (Just Noticeable Distortion) за различните подопсези. Тие ја одредуваат максималната грешка на квантизација која може да си ја дозволи кодерот при распределбата на расположливиот број на битови помеѓу подопсезите. При динамичкото доделување на битови, приоритет (највеќе битови)



Сл. 6.3: Архитектура на MPEG-1 ниво I и ниво II.

ќе добијат подопсезите со најниски JND. Нивната вредност зависи од два параметри прикажани на Сл. 6.4:

- **прагот на чујност** -- со кој е определена минималната спектрална амплитуда која човек може да ја чуе,
- **фреквенциското маскирање** -- со кое се одредува делот од амплитудниот спектар кој воопшто не може да биде чуен поради присуството на изразени спектрални компоненти.⁹

Притоа фреквенциското маскирање е тесно поврзано со **критичните опсези**¹⁰ на аудиторниот систем кои го одредуваат опсегот на фреквенции кои можат да бидат маскирани од една компонента во зависност од нејзината фреквенција. Тие всушност ја даваат ширината на пропусниот опсег на базилијарната мембра на функција од фреквенцијата.

За пресметување на JND се употребува FFT во 512 (ниво I) односно 1024 (ниво II) точки. На секој подопсег се врши динамичка компресија во блокови од по 12 примероци, со што максималната амплитуда на децимираните примероци во секој блок е 1. При тоа за секој блок се дефинира коефициент на размер (scale factor), со кој се врши нормализацијата. Секој блок соодветствува на $12 \cdot 32 = 384$ влезни примероци, односно 8,7 ms на 44,1 kHz.

На крајот започнува итеративна процедура на доделување на битови која ги користи пресметаните JND прагови за секој подопсег за одбирање на оптимален квантизатор за тој подопсег (од дадено почетно множество на квантизери). Изборот на квантизерите се прави така што двата критериуми – зададениот краен проток и пресметаниот JND, се задоволени. Коефициентите на размер се квантанизираат со 6 битови за секој подопсег, а изборот на квантизерот со 4 битови.

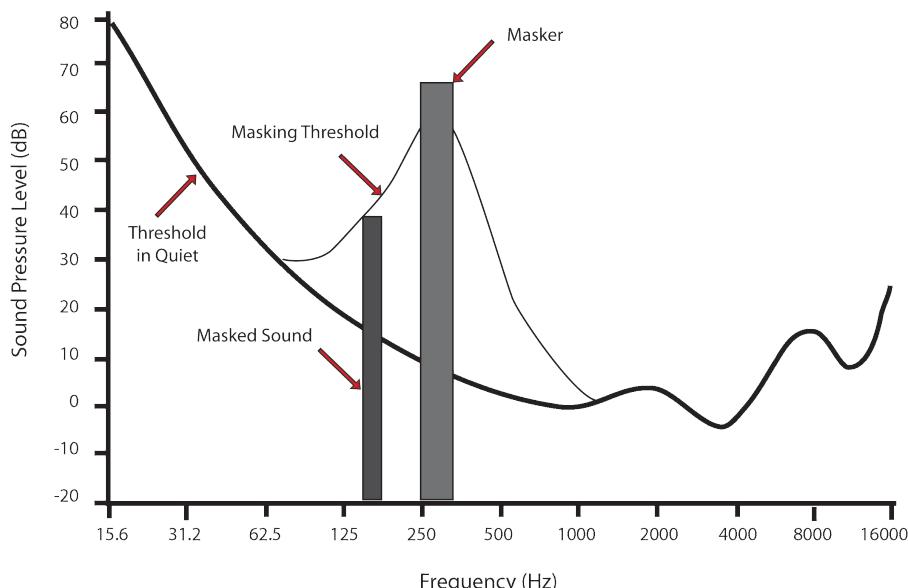
Во нивото I, децимираните подопсежни секвенции се квантанизираат и испраќаат на приемникот проследени со квантизираните коефициенти на размер и избраните квантизери. Во нивото II пак:

- психоакустичкиот модел има поголема FFT резолуција,
- максимална резолуција на подопсежните квантизери е зголемена на 16 битови, а понизок вкупен проток се остварува со намалување на бројот на понудени квантизери за повисоките подопсези,
- количината на додатна информација за коефициентите на размер е намалена со искористување на временското маскирање. Ова се прави преку разгледување на особините на 3 соседни блокови од 12 примероци и се испраќаат 1, 2 или 3 коефициенти заедно со 2-битен додатен параметар кој ја означува нивната поврзаност.

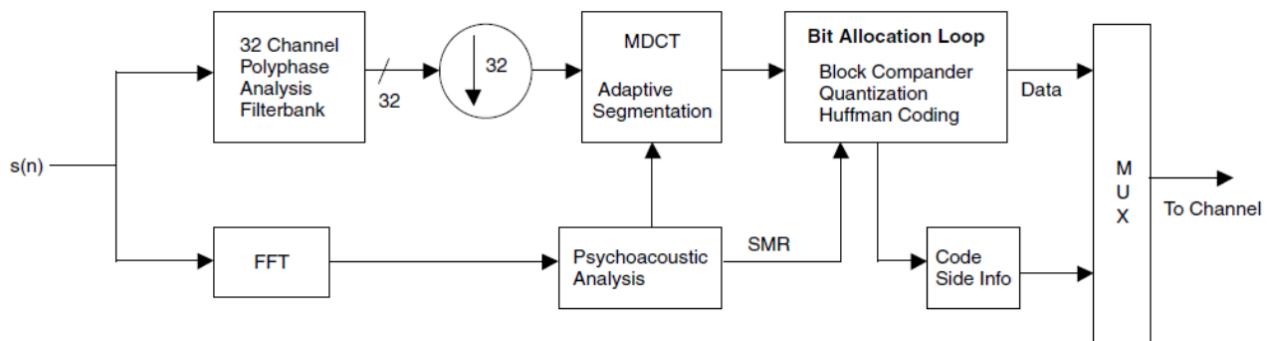
⁹Wikipedia: Psychoacoustics -- Masking effects: https://en.wikipedia.org/wiki/Psychoacoustics#Masking_effects

¹⁰Wikipedia: Critical band: https://en.wikipedia.org/wiki/Critical_band

¹¹By Audio_Mask_Graph.jpg: Daxx4434derivative work: Cradle (talk) - Audio_Mask_Graph.jpg, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=8390519>



Сл. 6.4: Прагот на чујност и појавата на фреквенциско маскирање на кои се базира принципот на работа на психоакустичката аудиокомпресија.¹¹



Сл. 6.5: Архитектура на MPEG-1 ниво III.

MPEG ниво-III алгоритмот, чија архитектура е прикажана на Сл. 6.5, работи врз последователни рамки на податоци. Секоја се состои од 1152 примероци звук; секоја рамка понатаму се дели на две подрамки од по 576 примероци, наречени гранули. Декодерот може да ја декодира секоја гранула посебно.

Во mp3 е воведена хибридна банка на филтри се употребува за зголемена фреквенциска резолуција и подобра апроксимација на критичните опсези на човековото уво. Исто така, хибридната банка на филтри вклучува адаптивна сегментација за подобрување на контрола врз пред-ехото. Хибридната банка на филтри е конструирана со надоврзување на секој подопсежен филтер на блок за адаптивна MDCT (Modified Discrete Cosine Transform). На тој начин, за разлика од нивоата I и II, во нивото III не се кодираат филтрирани сегменти на звук, туку нивните коефициенти во DCT домен. DCT трансформацијата е позната по својата способност на компактирање на енергијата на сигналите, односно претставување на голем дел од нивната енергијата со мал број на коефициенти. Поради ова, таа често се користи за нивна компресија, на пример кај JPEG стандардот.

На крајот, нивото III користи софистицирано доделување на битови и стратегии на квантизација кои се базираат на:

- неуниформна квантизација,
- анализа преку синтеза и

- ентрописко кодирање.

Доделувањето на битови и квантизацијата на MDCT спектралните линии се изведува преку вгнездена јамка која употребува и неуниформна квантизација и **Хофманово кодирање**. Внатрешната јамка го прилагодува чекорот на квантизација на трансформациските коефициенти за секој блок се додека не се задоволи зададениот битски проток. Надворешната јамка го контролира квалитетот на кодираниот сигнал преку анализа со синтеза, во однос на нивото на квантизацискиот шум спореден со пресметаните JND прагови.

6.2 Компресија на говор

Линеарното предиктивно кодирање (LPC) (Linear Predictive Coding) е еден од најважните концепти во кодирањето на дигиталниот говор. Техниката овозможува високо-квалитетно кодирање со ниски битскиprotoци од редот на 2,4 kb/s. Таа е во употреба во многу телекомуникациски системи за пренос на говор, од кои најважен е GSM (Groupe Spécial Mobile) системот за мобилна комуникација.

LPC се базира на **моделот извор-филтер** на создавање на говорот прикажан на Сл. 6.6. Во овој модел изворот го претставуваат неколку модули кои ги моделираат побудните механизми на човековиот говорен апарат и тоа:

- генератор на поворка на импулси -- ја моделира периодичната побуда од гласилките,
- генератор на шум -- ги модулира турбуленциите во воздушниот проток предизвикани од стеснувања во вокалниот тракт при изговор на согласките,
- засилување -- ја моделира активноста на белите дробови кои го генерираат субглоталниот притисок кој претставува побуда на целиот систем.

Дополнително во изворот постои и преклопник за селекција на моменталниот тип на побуда според тоа дали гласот кој се изговара е звучен или беззвучен.

Филтерот во моделот извор-филтер ја моделира преносната функција на вокалниот тракт која одговара на промените во напречниот пресек предизвикани од поставеноста на **артикулаторите**: јазикот, мекото непце, вилицата, и усните. Тој ја моделира оваа функција преку IIR филтер кој содржи само полови:

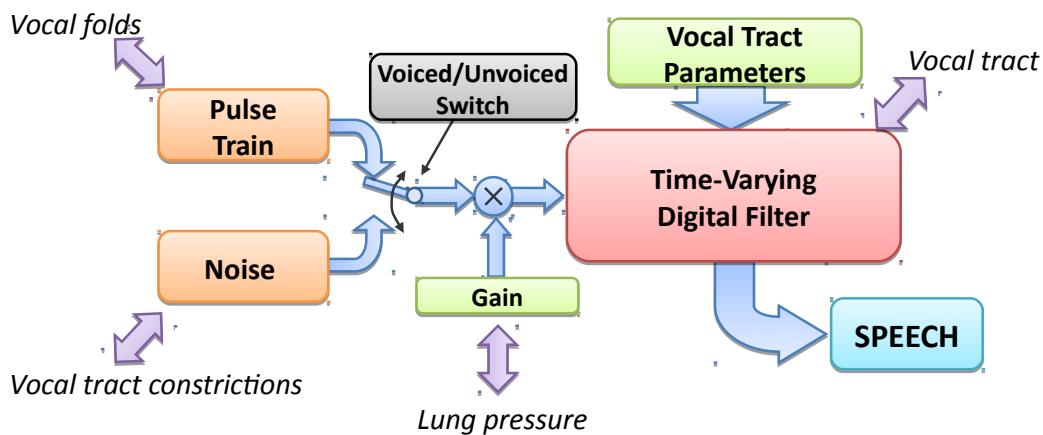
$$H(z) = \frac{Y(z)}{X(z)} = \frac{G}{1 - \sum_{k=1}^p a_k z^{-k}}, \quad (6.1)$$

каде $H(z)$ е преносната функција на филтерот во z -домен, $Y(z)$ е излезниот говорен сигнал, $X(z)$ е побудниот сигнал кој е добиен од изворот во моделот, G е засилувањето кое иако претставува дел од изворот, е интегрирано во филтерот, a_k се коефициентите на филтерот, а p е неговиот ред. Во временски домен ова равенство би било:

$$y[n] = \sum_{k=1}^p a_k y[n-k] + Gx[n]. \quad (6.2)$$

Со употреба на моделот извор-филтер, говорот може да се пренесува/зачувува без употреба на аудио примероци. На овој начин, може да се генерира доволно блиска апроксимација на говорниот сигнал, по цена на голема запштеда во простор. Параметрите кои се екстрагираат за секоја рамка од говорниот сигнал се:

- звучност -- дали се работи за звучен или беззвучен глас,
- засилување -- енергијата на сигналот во рамката,
- коефициенти на филтерот -- кои соодветствуваат на преносната функција на вокалниот тракт, и



Сл. 6.6: Моделот извор-филтер на принципот на создавање на говорот.

- **висина** -- периода на основниот хармоник.

Најосновната имплементација на LPC кодерот нуди одлична разбираливост на кодираниот говор. Но, квалитетот на излезниот говор не е на високо ниво и има карактеристичен роботски призвук. **Federal Standard (FS) 1015** кодекот од 1982 г. е првиот кодер базиран на LPC. Протокот кој тој го остварува е 2,4 kbit/s, што во споредба со вообичаениот проток на дигиталниот говор во телефонијата од 64 kbit/s¹², претставува компресија од 26 пати. Бидејќи филтерот кој го моделира вокалниот тракт е со ред 10, кодекот се нарекува и **LPC-10**. Ако не го знаете моделот, параметрите кои се испраќаат, немаат никакво значење. Поради тоа, FS 1015 кодекот е направен за американската војска, за потоа да го присвои и НАТО.

И покрај одличната разбираливост, овој кодер пати од лош, синтетички квалитет на излезниот говор. Ова се должи на главниот недостаток на овој пристап -- едноставноста на употребениот модел. Нефлексибилноста на изворот, кој може да работи исклучиво во еден од двата мода, претставува пречка во моделирањето на гласови од мешан тип, на пр. звучни согласки како „з“ /z/ или „в“ /v/, како и на меѓу-гласовни премини. Овој недостаток е надминат кај **CELP** (Code-Excited Linear Prediction) кодерот, кој на местото од едноставниот извор, работи со база на побуди поместени во една кодна книга. CELP моделот е оној кој е во употреба во GSM мрежата.

¹²8 kHz · 8 bit = 64 kbit/s

Глава 7

Синтеза на звук и говор

Еден од начините за синтеза на звук е со употреба на модели кои вршат параметризација на сигналот и потоа ги користат овие параметри за генерирање на нов сигнал. Еден таков модел е [линеарната предикција](#) која се користи во моделот извор-филтер кој го сретнавме кај компресијата на говорните сигнали. Во ова поглавје ќе ги искористиме овој модел за синтеза на глас односно синтеза на пеење. Во вториот дел ќе се запознаеме подробно со методите за синтеза на говор и ќе имплементираме еден едноставен систем за синтеза на македонски.

7.1 Линеарна предиктивна анализа

Филтерот кој се користи во моделот извор-филтер, односно во компресијата на говорните аудиосигнали базирана на линеарна предикција (LPC) дискутирана во Главата 6.2 го има обликот даден во (6.1):

$$H(z) = \frac{Y(z)}{X(z)} = \frac{G}{1 - \sum_{k=1}^p a_k z^{-k}}. \quad (7.1)$$

Од акустиката на вокалниот тракт пак, за верно претставување на назалите и фрикативите се потребни и нули во спектарот. Сепак се применува филтер кој содржи исклучиво полови затоа што коефициентите на вака дефинирираниот филтер можат лесно да се одредат преку употреба на методот за [линеарна предиктивна анализа](#) врз говорниот сигнал. За апроксимација пак на нулите во преносната функција на вокалниот тракт, се земаат поголем број на полови во овој филтер. Линеарен предиктор со коефициенти α_k е дефиниран со следното равенство (Rabiner and Schafer, 1978):

$$\tilde{y}[n] = \sum_{k=1}^p \alpha_k y[n - k], \quad (7.2)$$

каде со $\tilde{y}[n]$ е означена предикцијата на сегашната вредност на излезниот сигнал базирана на тежинска сума на неговите p претходни примероци. Притоа грешката од предикција може да се пресмета како:

$$e[n] = y[n] - \tilde{y}[n] = y[n] - \sum_{k=1}^p \alpha_k y[n - k]. \quad (7.3)$$

Преносната функција на овој систем во z -домен е:

$$A(z) = 1 - \sum_{k=1}^p \alpha_k z^{-k}. \quad (7.4)$$

Споредувајќи ги овие равенства со (7.1) и (6.2):

$$y[n] = \sum_{k=1}^p a_k y[n - k] + Gx[n], \quad (7.5)$$

можеме да видиме дека ако е задоволен условот:

$$\alpha_k = a_k, \quad \text{за } k = 1, 2, \dots, p, \quad (7.6)$$

тогаш сигналот на грешка е еднаков на побудниот сигнал на моделот извор-филтер:

$$e[n] = Gx[n], \quad (7.7)$$

а преносната функција на филтерот на грешка $A(z)$ претставува инверзен филтер на $H(z)$:

$$H(z) = \frac{G}{A(z)}. \quad (7.8)$$

Основниот проблем во линеарната предикција претставува одредување на коефициентите α_k директно од говорниот сигнал за да се добие добра проценка на спектралните карактеристики на говорниот сигнал преку употреба на (7.8). Поради временската спектрална динамика на говорот, нужна е проценка на овие коефициенти за кратки отсекоци на сигналот добиени со методата на прозорци.

Основниот пристап во решавањето на овој проблем е одбирање на коефициенти на предикторот кои ќе ја минимизираат средната квадратна грешка на предикција. Постојат различни пристапи за решавање на овој проблем како методите со автокорелација и коваријанса. Најшироко применет алгоритам за одредување на коефициентите на предикторот е рекурзивниот метод на Дурбин (Rabiner and Schafer, 1978, 2011).

7.2 Анализа и синтеза на глас со линеарна предикција

Кога ќе ги одредиме коефициентите на предикторот, односно на филтерот во моделот извор-филтер, можеме да го користиме истиот за синтеза на глас. Притоа, филтерот не мора да го побудиме со оригиналниот сигнал на изворот, ами може да го побудиме со било кој сигнал, како на пр. поворка на импулси или бел шум или мешавина од двете. **Вокодерот**¹ е аудиоэффект кој се добива кога добиениот филтер се побуди со музички сигнал, најчесто добиен со употреба на синтисајзер. Музичката мелодија отсвирена на синтисајзерот бива вообличена во аудиосигнал кој има боја на гласот и ги содржи зборовите кажани во оригиналниот говорен сигнал, со тоа што тие во овој случај се „испеани“ со мелодијата од синтисајзерот.

За практично да се запознаеме со начинот на функционирање на линеарната предикција во предметов ќе ја илустрираме нејзината примена во синтеза на човечки глас. Тој процес е во сржта на нејзината примена за компресија на говорните сигнали, како и во аудиоэффектите и кај синтезата на говор. За целите на оваа анализа направете снимка од сопствениот глас како ги изговарате петте самогласки во македонскиот јазик: /a/, /e/, /i/, /o/ и /u/. Тоа ќе го направиме со помош на Audacity.²

Моделирање на гласот /a/

Во нашата анализа ќе ја искористиме имплементацијата на линеарната предикција, поточно алгоритамот на Бург базиран на автокорелацијата која е имплементирана во пакетот LibROSA³. Овој модул содржи низа на функции за анализа на звук, музика и говор и може едноставно да се инсталира користејќи го pip :

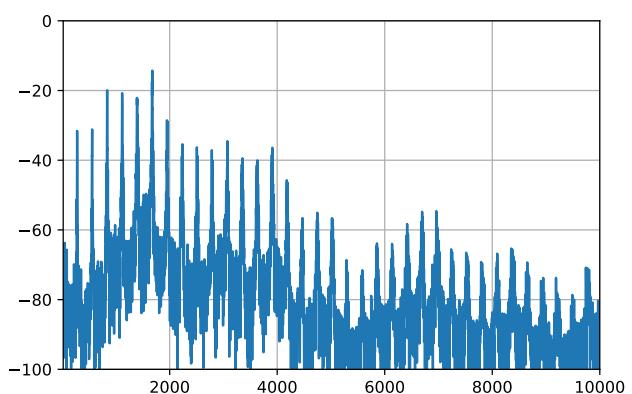
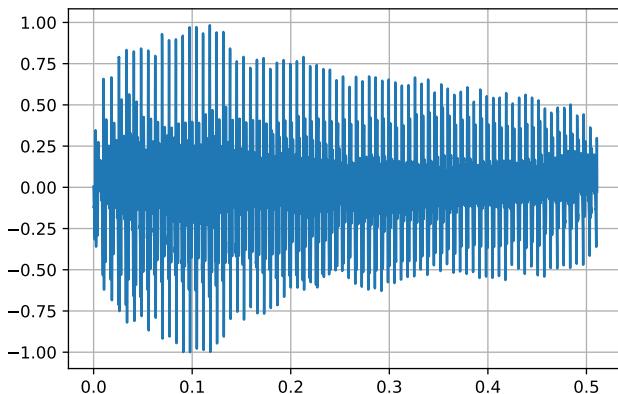
```
(da) ~/da/$ pip install librosa
```

Во кодов што следи ќе го вчитаме звучниот запис со самогласката /a/ и ќе го прикажеме неговиот временски и спектрален облик, Сл. 7.1.

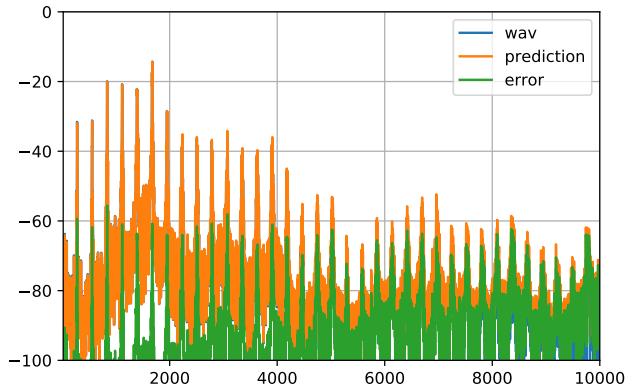
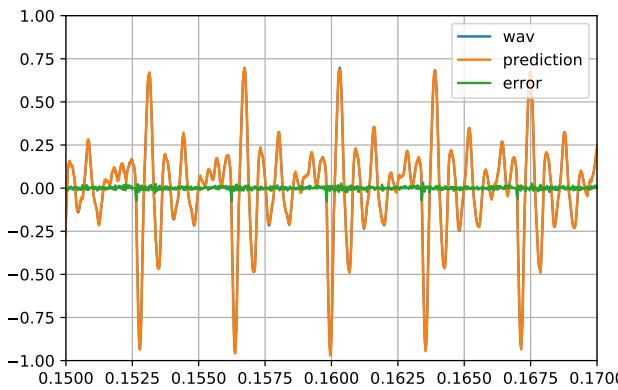
¹Wikipedia: Vocoder. <https://en.wikipedia.org/wiki/Vocoder>

²За Audacity види повеќе во Главата .

³LibROSA <https://github.com/librosa/librosa>



Сл. 7.1: Приказ на временскиот (лево) и спектралниот (десно) облик на аудиосигналот на гласот /а/.



Сл. 7.2: Оригиналниот аудиосигнал, неговата предикција и сигналот на грешка во временски (лево) и во спектрален (десно) домен.

```

import numpy as np
from matplotlib import pyplot as plt
from scipy.io import wavfile
import os
from scipy import signal as sig
import librosa as lr
import da

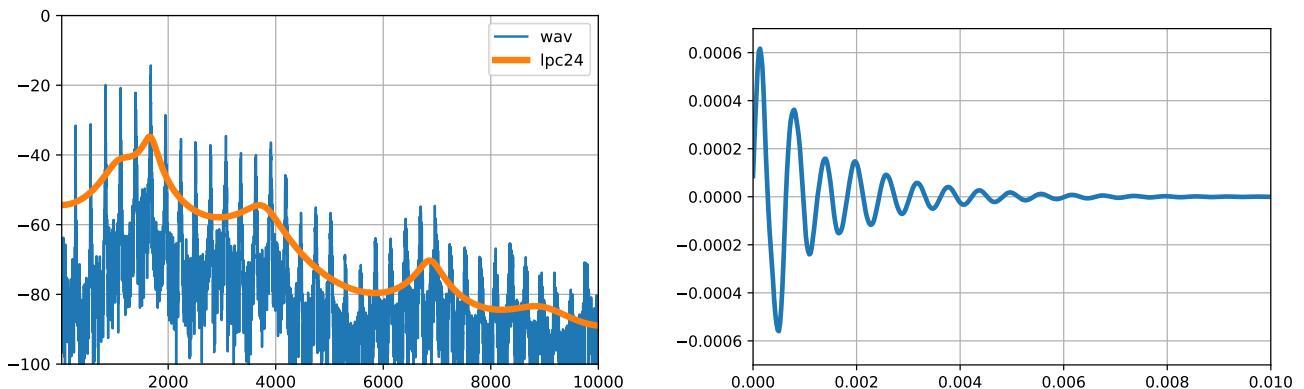
# %% вчитај звук
folder = 'audio/'
filename = 'glas_aaa.wav'
fs, wav = wavfile.read(folder+filename)
wav = wav / 2**15
t = np.arange(wav.shape[0]) / fs
wav = da.normalize(wav, 0)

# %% исцртај временски домен
plt.figure()
plt.plot(t, wav)
plt.grid('on')

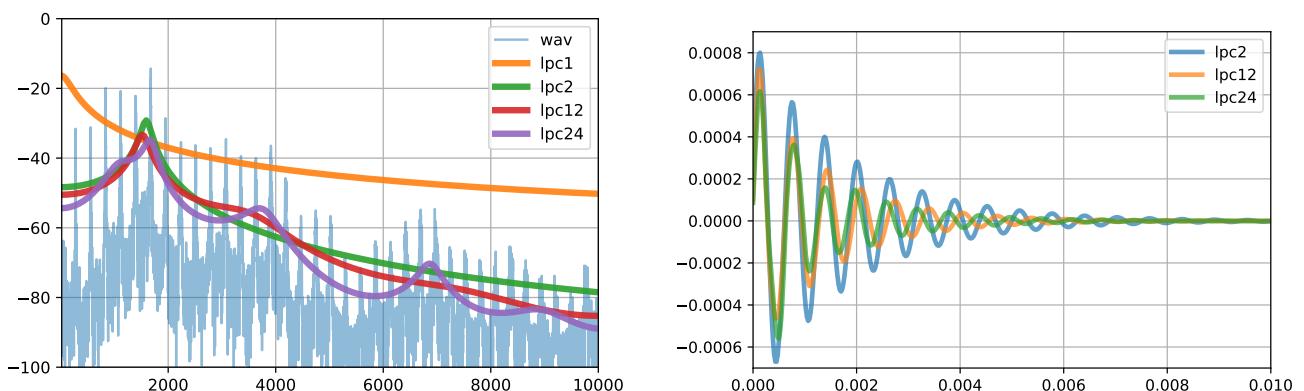
# %% исцртај спектрален домен
f, wav_spec = da.get_spectrum(fs, wav)
plt.figure()
plt.plot(f, wav_spec)
plt.xscale('log')
plt.grid('on')
plt.axis([0, 2e4, -100, 0])

```

Во спектралниот облик на сигналот може да се видат хармониите кои се должат на периодичноста на сигналот, како и анвелопата на спектарот која има изразени врвови. Овие



Сл. 7.3: Фреквенциската карактеристика на добиениот филтер за предикција од 25-ти ред (лево) и неговиот импулсен одсив (десно).



Сл. 7.4: Фреквенциски карактеристики на филтри за предикција од различен ред (лево) и нивните импулсни одсиви (десно).

врвови одговараат на резонантните фреквенции на вокалниот тракт при изговор на гласот /a/. Кога се работи за самогласки ти се нарекуваат **форманти**. Сега ќе ги најдеме коефициентите на филтерот кои треба да ја опишат анвелопата на сигналот со употреба на функцијата `lpc`.

```
p = 24
a_filt = lr.lpc(wav, p)
b_inv = np.r_[0, -a_filt[1:]]
wav_pred = sig.lfilter(b_inv, 1, wav)
wav_err = wav - wav_pred
```

На Сл. 7.2 се претставени оригиналниот аудиосигнал, неговата предикција и сигналот на грешка. Може да се види дека разликата меѓу аудиосигналот и предикцијата добиена со филтер со ред 25 е многу мала и има мали импулси на почетокот на секоја периода. Ова е всушеност сигналот кој алгоритамот за одредување на коефициентите за предикција го минимизира. Импулсите во него ги претставуваат моментите кога побудата на вокалниот тракт е максимална што одговара на моментите на **глотално затворање**, односно кога протокот на воздух низ гласилките постигнува брзина за која страничниот притисок не може да ги држи отворени па тие се затвораат. Оваа периодичност појасно се гледа во спектарот на сигналот на грешка кој ги содржи хармониците од оригиналниот сигнал. Уште поважно е што тој не ја содржи спектралната анвелопа, односно енергијата на сите негови хармоници е речиси иста. Може да се каже дека сигналот на грешка претставува спектрално „обелена“ верзија на оригиналниот сигнал.⁴

За да видиме како добиените коефициенти на предикторот ја опишуваат спектралната анвелопа на оригиналниот сигнал, ќе исцртаме фреквенциската карактеристика на добиениот филтер суперпонирана на спектарот на сигналот.

⁴На англиски процесот на „белене“ на еден сигнал се нарекува *whitening*.

```

g = 0.001
f_filt, h_filt = sig.freqz(g, a_filt, fs=fs)
h_filt = 20 * np.log10(np.abs(h_filt))
plt.figure()
plt.plot(f, wav_spec)
plt.plot(f_filt, h_filt, lw=4)

```

Импулсниот одсив пак на филтерот ќе го добијеме со следниот код. Двата графици се прикажани на Сл. 7.3.

```

excite = np.zeros(int(.050*fs))
excite[0] = 1
h_imp_filt = sig.lfilter(g, a_filt, excite)
t_imp = np.arange(excite.size) / fs
plt.figure()
plt.plot(t_imp, h_imp_filt, lw=3)

```

На Сл. 7.4 е дадена споредба на фреквенциските карактеристики и импулсните одсиви на филтри за линеарна предикција со различен ред. Може да се види дека филтер од прв ред го доловува само глобалниот пад на енергијата со растењето на фреквенцијата, додека филтерот од втор ред го фаќа главниот формант во спектралната анвелопа на гласот /а/. Како го зголемуваме бројот на коефициенти во филтерот така сè подобро тие ја претставуваат спектралната анвелопа на аудиосигналот. Едно непишано правило е дека при моделирање на говор бројот на коефициенти на филтерот треба да е еднаков на бројот на форманти, т.е. резонантни фреквенции, плус 2 пола за нулите. Вообичаено се зема дека во секој kHz од спектарот на сигналот има по еден формант, па од таму оптималниот ред е широчината на сигналот во kHz плус 2. Во нашиот случај за f_s од 44,1 kHz тоа значи дека оптималниот ред на филтерот би бил 22.

Синтеза на пеење

Сега кога ги имаме коефициентите на филтерот за предикција можеме да го користиме истиот за синтеза на пеење. Најпрвин да го побудиме филтерот со поворка на Диракови импулси со фреквенција од 207 Hz.

```

excite = np.zeros(1*fs)
g = 0.03
f0 = 207
t0 = 1/f0
step = int(fs * t0)
excite[::step] = 1
response = sig.lfilter([1], a_filt, g*excite)
sd.play(response, fs)

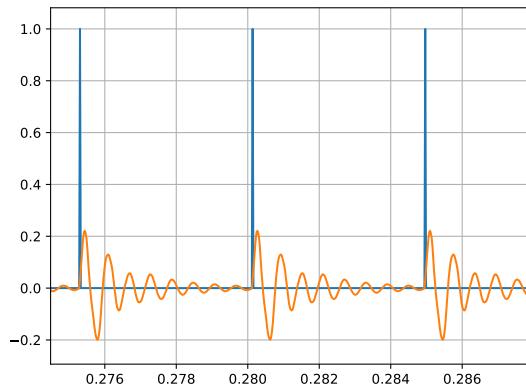
# plot the excitation and the response
plt.figure()
t = np.arange(excite.size) / fs
plt.plot(t, excite)
plt.plot(t, response)
plt.grid()

```

Може да слушнеме дека аудиосигналот има константна основна фреквенција и има боја која наликува на самогласката /а/. Сепак, исто така може да заклучиме дека добиениот сигнал има изразито вештачки призвук и е далеку од природен.⁵ Изгледот на побудата и одсивот на филтерот се прикажани на Сл. 7.5.

Следно да го побудиме филтерот со мелодија за да добијеме излезен сигнал налик на пеење. Најпрвин ќе ги иницијализираме листите со музичките ноти од мелодијата и нивното времетраење. Притоа, времетраењето на нотите ќе го изразиме релативно на времетраењето на четвртинаnota која ќе има времетраење од 1.

⁵Ова се должи на едноставноста на применетиот пристап. Поквалитетна синтеза може да добијеме ако искористиме променливи коефициенти на филтерот кои сме ги пресметале за повеќе рамки од оригиналниот говорен сигнал.



Сл. 7.5: Побудата на филтерот во вид на поворка на Диракови импулси (сино) и одсивот на филтерот (портокалово).

За мапирање на нотите во фреквенции ќе направиме речник според следната табела⁶:

Табела 7.1: Фреквенција на нотите на пијано во основната октава.

Нота	Фреквенција (Hz)
C	261.6256
C#	277.1826
D	293.6648
D#	311.1270
E	329.6276
F	349.2282
F#	369.9944
G	391.9954
G#	415.3047
A	440.0000
A#	466.1638

```

melody = 'E E F G G F E D C C D E E   D  D'.split()
meldur = '1 1 1 1 1 1 1 1 1 1 1 1 1.5 .5 2'.split()

# tone dictionary
tone2freq = {
    'C': 261.6256, 'C#': 277.1826, 'D': 293.6648, 'D#': 311.1270,
    'E': 329.6276, 'F': 349.2282, 'F#': 369.9944, 'G': 391.9954,
    'G#': 415.3047, 'A': 440.0000, 'A#': 466.1638
}

```

Пред да ја синтетизираме мелодијата, треба да го пресметаме времетраењето на четвртина нота во секунди, односно во број на примероци за избрана фреквенција на земање на примероци. Должината на нотите е поврзана со бројот на четвртини ноти во минута, односно бројот на удари во минута (BPM)⁷. или темпото на мелодијата. Притоа, вообичаено се зема дека четвртината нота има времетраење од 0,5 s што соодветствува на 120 BPM.

```

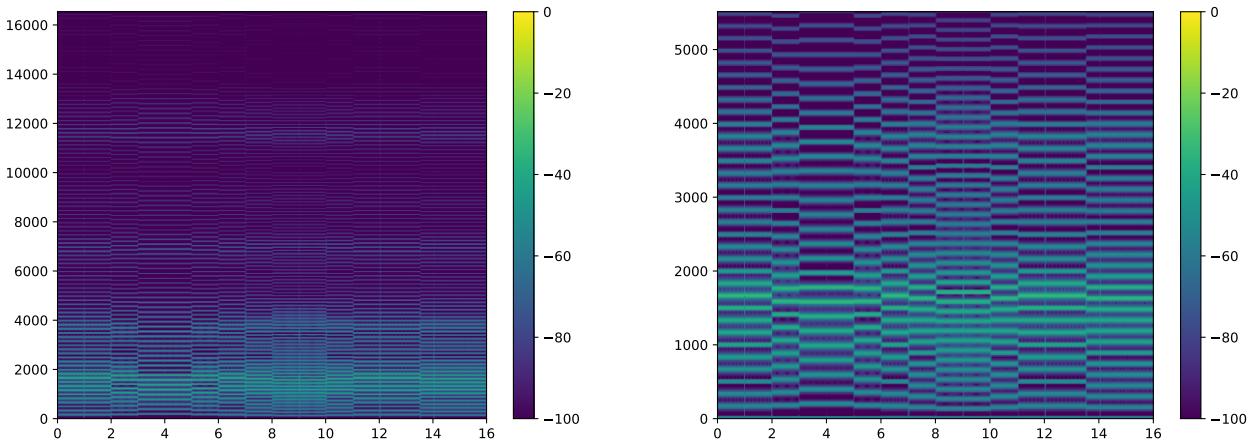
bpm = 120
bps = bpm / 60
qnote_dur = 1 / bps
transpose = 1/2

```

Тука го додадовме и параметарот за транспонирање на мелодијата transpose кој може да го

⁶Wikipedia: Piano key frequencies. https://en.wikipedia.org/wiki/Piano_key_frequencies

⁷анг. beats per minute.



Сл. 7.6: Спектрограм на синтетизираниот сигнал на пеење (лево) и зголемен приказ за ниските фреквенции (десно).

искористиме за да ја зголемиме или намалиме висината на мелодијата.

Конечно, ќе направиме јамка која ќе ја измине мелодијата, составена од низата на ноти и низата времетраења, и ќе генерира сигнал со кој ќе го побудиме филтерот. Добиените филтрирани сигнали ќе ги надоврзземе во излезниот сигнал на пеење.

```
wav_melody = np.array([])
for tone, dur in zip(melody, meldur):
    t_note = float(dur) * qnote_dur
    n_note = int(t_note * fs)
    excite = np.zeros(n_note)
    f0 = tone2freq[tone] * transpose
    t0 = 1/f0
    step = int(fs * t0)
    excite[::step] = 1
    wav_note = sig.lfilter([1], a_filt, g*excite)
    wav_melody = np.append(wav_melody, wav_note)

# play and plot the synthesized singing signal
sd.play(wav_melody, fs)
-- = da.get_spectrogram(wav_melody, fs)
```

Спектрограмот на добиениот сигнал е прикажан на Сл. 7.6. Може да се види дека сигналот е далеку од природниот глас, кој има побогата спектрална содржина, на што се должи неговиот вештачкиот призвук.

7.3 Синтеза на говор

Вештачката синтеза на човековиот говор била инспирација за голем број на научници и пронаоѓачи. Нејзиниот развој тргнува од механичките модели на човековиот говорен апарат на Фон Кемпелен од XVIII век, па преку механичкиот VODER од 1930-те и аналогните електронски синтезатори од 1950-те, до денешните системи во ерата на дигиталните компјутери. Денес, науката и технологијата се на чекор до целта -- синтезата на човеков говор кој нема да може да се разликува од природниот.⁸

Модерните синтезатори, не само што се сосема разбираливи и звучат речиси потполно природно, тие почнуваат во синтетизираниот говор да внесуваат и чувства, како и да го имитираат текот на природниот говор, со вметнати паузи, земање здив, насмевки и сл.

⁸Обидете се препознаете кој од гласовите е синтетизиран во делот „Такотрон 2 или човек“ <https://google.github.io/tacotron/publications/tacotron2/index.html>.

Вештачката синтеза на човековиот говор сеуште претставува област на интензивно истражување во светската наука. Денес фокусот е свртен воглавно на доловување на паралингвистичките информации во говорот, кои претставуваат збир на сите информации кои говорникот сака да ги пренесе, а кои не зависат од лингвистичката (текстуална) содржина. Така, луѓето можат една иста реченица да ја изговорат на различни начини и со тоа да пренесат низа на ставови на соговорникот, на пр.: изненадување, неверување, иритираност, замисленост, настојување, итн. Овој вид на информации се пренесуваат преку [прозодијата](#) во говорниот сигнал, која ги вклучува интонацијата, интензитетот и ритамот.

Синтетизаторите за говор вообичаено имаат задача да синтетизираат говор од даден влезен текст, којшто може да биде внесен од корисникот или пак да биде избран од одредена база на текст, како на пример некоја книга, весник или пак реплика избрана од страна на систем за водење автоматски дијалог. Поради тоа, овие системи се нарекуваат системи за синтеза на говор од текст или TTS⁹ системи. Модерните синтетизатори на говор, односно системите за синтеза на говор од текст се сосема разбирливи и звучат речиси целосно природно, до тој степен што за одредени реченици е речиси невозможно да се препознае кој говор е синтетизиран.

Примената на TTS системите во човековото секојдневие е разнолика. Од клучна важност е примената на TTS системите кај луѓето со визуелен хендикеп, зашто таа им овозможува пристап до илјадници литературни дела, како и низа печатените медиуми. Кај наглувите и вокално хендикепираните кои не можат да зборуваат правилно и разбирливо, уредите за синтеза на говор со вградена тастатура им овозможуваат комуникација со луѓе кои не го разбираат говорот со знаци. Исто така, тие им ја овозможуваат и употребата на телефонски апарати. Кај пошироката популација тие можат да се употребат за исчитување на електронска пошта, статии или друг текст од мониторот при работа со компјутер, растоварувајќи го визуелниот систем. TTS системите можат да се употребат и за читање на пораките на мобилните телефони, што е згодно во некои ситуации, како на пример при управување на автомобил. Уште повеќе, синтезата на говор овозможува природен интерфејс на човекот со домашните и мобилните апарати, телефоните, компјутерите, дури и пристап кон интернет. Синтезата на говор има примена и во едукативни цели на пример во алатките за изучување на странски јазици. Врвниот дострел на оваа технологија би била реализацијата на универзални преведувачи, во спој со системите за препознавање на говор и за машински превод.

Основи парадигми за синтеза на говор

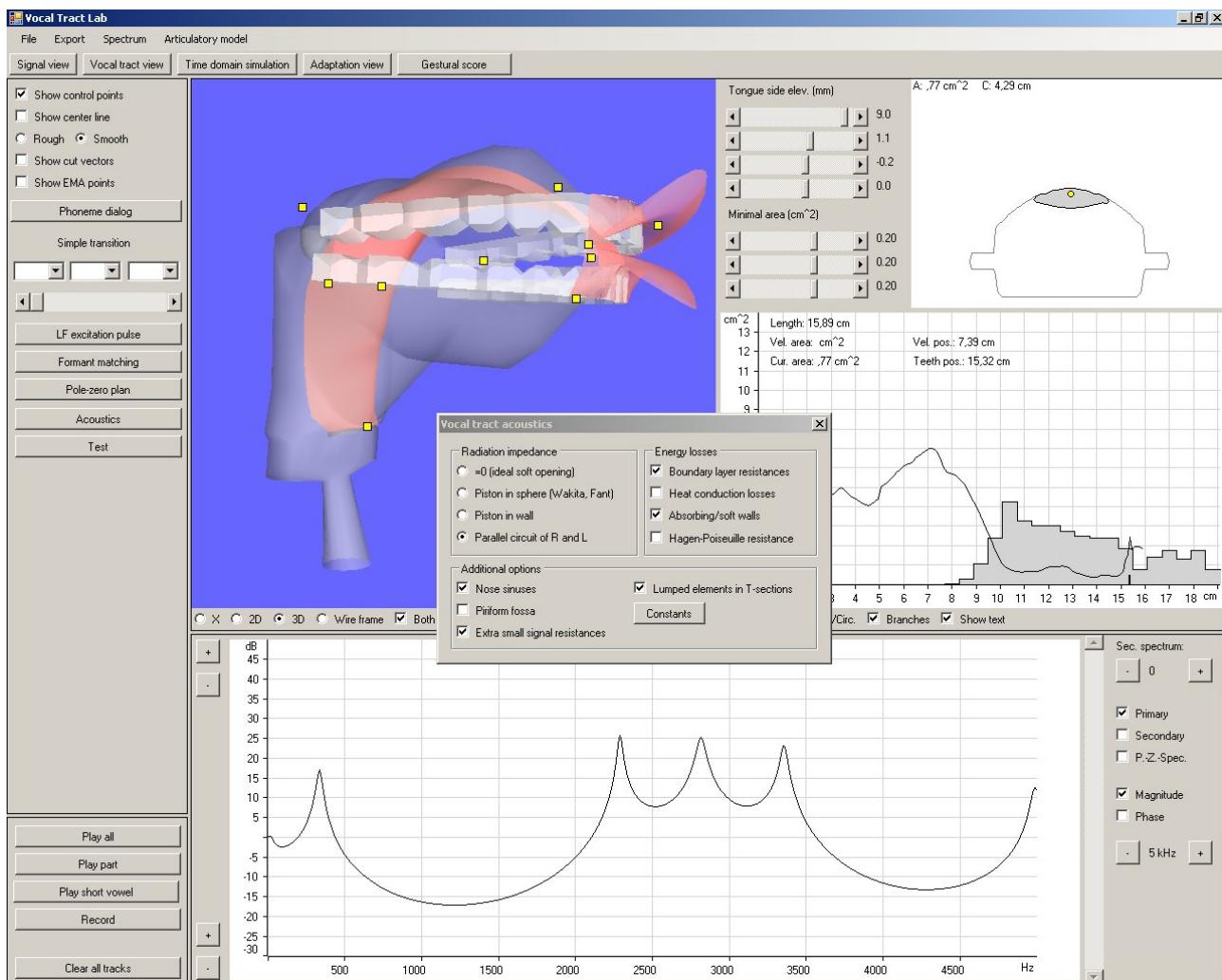
Системите за синтеза на говор се стремат вештачки да го направат она што вокалниот тракт го прави природно. Нивната главна цел е на излезот да добијат „говор“ којшто по разбирливоста и по квалитетот е близок на природниот. Постојат четири основни парадигми во дигиталната синтеза на говор:

- артикулаторна синтеза,
- синтеза со форманти,
- синтеза со надоврзување,
- параметарска синтеза и
- синтеза со длабоко учење.

Артикулаторна синтеза

Во артикулаторната синтеза се врши моделирање на структурата на вокалниот тракт и артикулаторите: јазикот, забите, усната шуплина итн., како и нивните движења при зборувањето. Ова е моќен пристап, кој нуди неограничени можности за контрола над синтезата на говор и кој има потенцијал да обезбеди висок квалитет на својот излез. Создавањето на квалитетен

⁹анг. *Text-to-Speech*



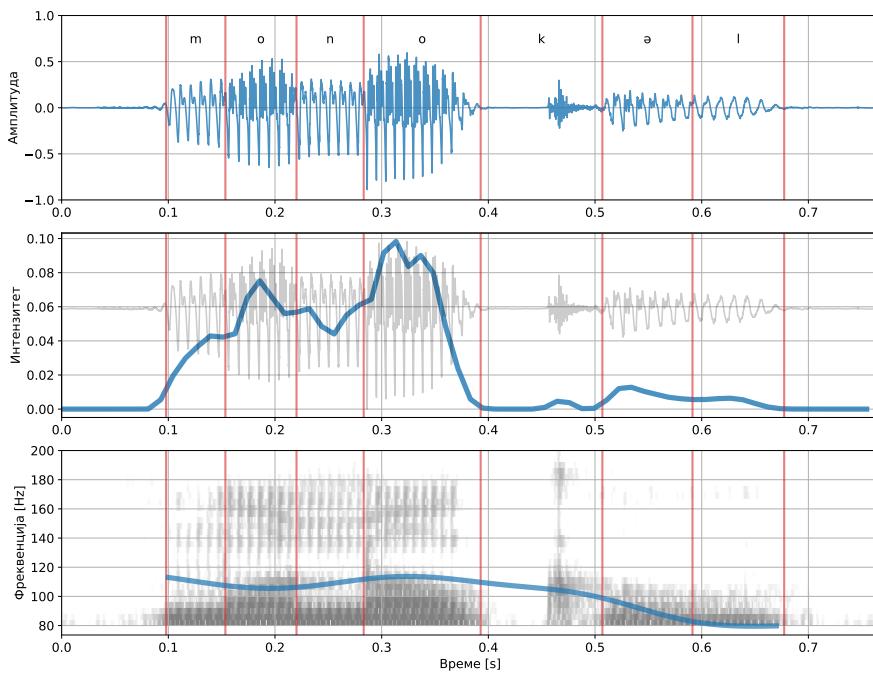
Сл. 7.7: Изглед на 3Д моделот на вокалниот тракт во TTS системот базиран на артикулаторна синтеза на Биркхолц.¹⁰

модел и прецизното мапирање на движењата на неговите составни делови при артикулацијата е сложен процес кој вклучува, меѓу другото, правење медицински снимки со употреба на магнетна резонанса на вокалниот тракт при говорењето. Уште повеќе, поради сложеноста на моделирањето на струењето и вртложењето на воздухот во три димензии, се пристапува кон дизајнирање на акустички филтер базиран само на дводимензионален напречен пресек на шуплината на вокалниот тракт во секој момент во процесот на артикулација, прикажано на Сл. 7.7. Конечно, промените во протокот на воздух низ ларингсот предизвикани од треперењето на гласилките се сложени и подложат на повратна спрега од обликот на вокалниот тракт. Поради тоа се применуваат едноставни модели за нивното моделирање. Сите овие поедноставувања доведуваат до влошување на квалитетот на синтетизираниот говор, па тој звучи вештачки.

Синтеза со форманти

Во синтезата со форманти, наместо да се моделираат процесите во вокалниот тракт при артикулацијата, се врши моделирање на создавањето на говор преку моделот извор–филтер кој го воведовме во Главата 6.2 и кој е прикажан на Сл. 6.6. Моделот извор–филтер претставува еден од најмокните концепти во областа на анализата, кодирањето и синтезата на говор. Со него преку контрола на параметрите на моделот може директно да се добие звучниот сигнал на говорот. Всушност истиот модел се користи за синтеза на звучниот сигнал и кај артикулаторната синтеза. Покажано е дека со внимателен избор на параметрите на моделот може да се достигне

¹⁰Vocal Tract Lab <http://www.vocaltractlab.de/>



Сл. 7.8: Сегментација на говорен сигнал на зборот „монокл“ на ниво на гласови со приказ на брановиот облик (горе), интензитетот (средина), и спектарот и висината (долу).¹¹

висок квалитет на синтетизираниот говор, но сепак поради едноставноста на употребениот модел за изворот, овие системи не звучат воопшто природно.

Поради тоа што моделирањето на промената на преносните карактеристики во говорот е макотрпен и скап процес, овој тип на синтеза останува тежок за реализација.

Синтеза со надоврзување

Во синтезата со надоврзување, сосема се напушта моделирањето како пристап, а синтезата на говор се врши по принципот на надоврзување на кратки исечоци на снимен говор за составување на бараната говорна содржина. Ова му дава голема природност на излезниот говор, притоа целосно елиминирајќи ја потребата од скапото создавање на модел. Множеството на исечоци, наречени единки, ја сочинуваат базата на гласовни единки. Во праксата се употребуваат единки од различен ранг, односно со различни должини, почнувајќи од гласови, па парови од гласови или двогласи, трогласи, полуслогови, слогови, па дури и цели зборови. Колку единките се подолги, толку поприродно звучи излезниот говор, но толку поголема е и базата. Како најоптимален избор, најчесто TTS системите со надоврзување употребуваат единки од ранг на двогласи кои се состојат од половините на два надоврзани гласа, вклучувајќи го преминот меѓу нив. На Сл. 7.8 може да се види сегментација на говорен сигнал добиен за зборот „монокл“ на ниво на гласови.

TTS системот може да има и база составена од единки од различен ранг, па според тоа разликуваме системи со униформни бази и системи со бази од мешан тип. Двогласите нудат прифатлив квалитет на излезниот говор, а притоа нивниот број не е многу голем и вообичаено се движи околу 1000. Имено, теорискиот максимум на бројот на двогласи е квадрат од бројот на основни гласови во еден јазик. Меѓутоа, поради правилата на фонотактиката, која ги проучува законитостите на впарување на гласовите, овој број е вообичаено помал. На пример, во английскиот јазик има 43 гласови а 1162 двогласи, колку што се користат во TTS системот на AT&T, што е помалку од теориските 1849 можни комбинации. Според типот на базата TTS системите можат да се поделат и на системи со еднозначни и системи со проширени бази. Системите со еднозначни бази, имаат бази кои содржат по една верзија од секој двоглас, односно

¹¹ Сликата е превземена од вториот том Супрасегментална фонетика и фонологија, на новата Фонетика и фонологија на македонскиот стандарден јазик <http://ical.manu.edu.mk/index.php/publications>



Сл. 7.9: Архитектура на TTS систем со надоврзување.

единка. Расположливоста на само по една гласовна единка изискува употреба на алгоритми за дигитално процесирање на сигналите за обликување на единките при надоврзувањето, што негативно се одразува на природноста на синтетизираниот говор.

Поради ова, напредните TTS системи со надоврзување користат проширен бази кои содржат повеќе единки за дадена фонетска содржина, што придонесува кон добра покриеност на потребите за гласовни единки во бараниот говорен контекст, т.е. интонација и коартикулација, при надоврзувањето, намалувајќи ја потребата од процесирање, а со тоа зголемувајќи ја природноста. Овие системи вообичаено располагаат со снимен говор со времетраење од неколку часа, а нивните бази содржат многу единки за секоја фонетска содржина. Поради големината на базите, квалитетот на овие системи во голема мера зависи од алгоритмите употребени за избор на најсоодветната единка за надоврзување, поради што и се нарекуваат TTS системи со избор-на-единка/*footnote*-анг. *unit-selection*.

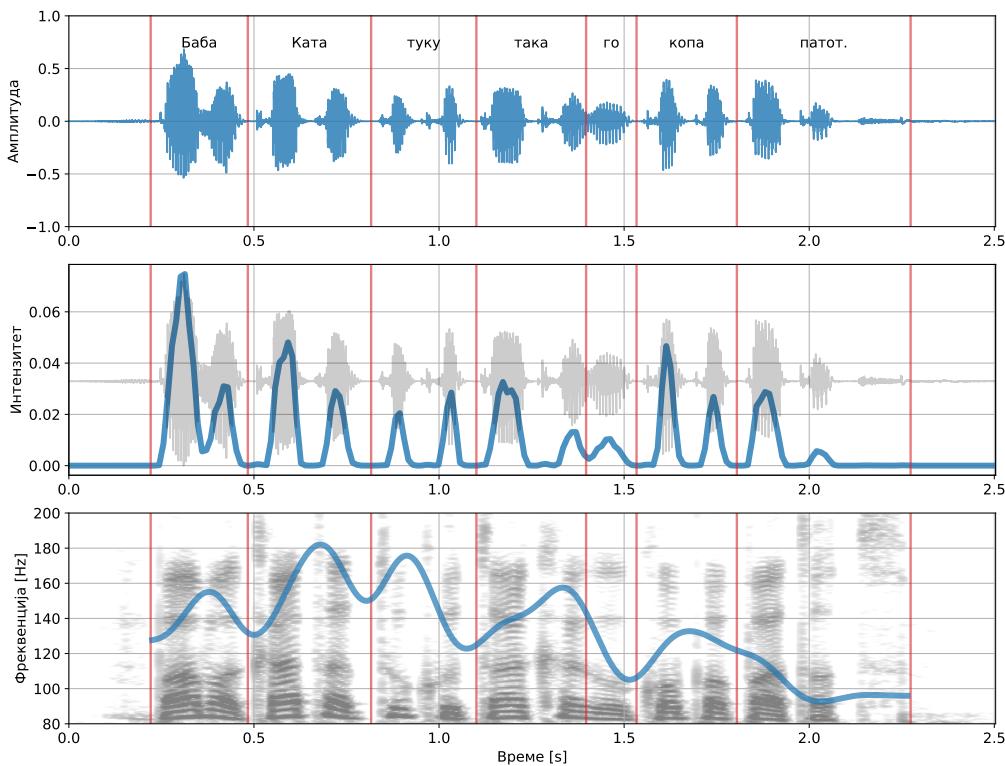
Архитектурата на еден TTS систем со надоврзување е прикажана на Сл. 7.9. Системот се состои од два главни модули: модул за анализа на текстот и модул за синтеза на звук. Задачата на модулот за анализа на текстот е да го трансформира влезниот текст во неговата фонетска претстава и на неа да придржи информации за интонацијата и времетраењето на секој од гласовите. Задачата на модулот за синтеза на звук е да го синтетизира говорот врз база на фонетската содржина со модификација и надоврзување на гласовните единки од базата.

Параметарска синтеза на говор

Синтезата со надоврзување може да произведе многу природен говор, но таа има ограничена флексибилност и синтетизираниот говор е тесно поврзан со својствата на говорниот корпус на гласовни единици. Системите за параметарска синтеза ги користат придобивките од традиционалните методи за машинско учење и статистичките методи за да ги надминат овие ограничувања. Тие го синтетизираат говорот со моделирање на основната фреквенција, амплитудата, времетраењето и спектралната содржина на човечкиот говор. На овој начин, овозможуваат голема приспособливост на генериралиот говор, на пример промени во интонацијата и брзината, без потреба да се снимаат нови бази на гласовни единки.

Системите за параметарска синтеза се во основа инверзни на системите за [автоматско препознавање говор \(ASR\)](#)¹², претворајќи го текстот во акустични обележја а потоа во бранови

¹²анг. *Automatic Speech Recognition*



Сл. 7.10: Акустички обележја кои системите за параметарска синтеза ги моделираат кај говорот: спектар, интензитет, висина и времетраење, на ниво на исказ.¹⁵

облици на говор. За моделирањето на акустиката тие користат комбинација на **Скриени модели на Марков (HMM)**¹³ и **Гаусови мешавински модели (GMM)**¹⁴, додека за синтеза на звук од генерираните обележја користат вокодери. Моделите на овие системи се обучени со големи бази на податоци на транскрибиран говорен материјал. Акустичките обележја кои овие системи ги моделираат може да се видат на ниво на реченица и на ниво на збор на Сл. 7.10.

Главниот недостаток на системите за параметарската синтеза е што генериралиот говор кој звучи вештачки, односно роботски, што се должи на ограничувањата на употребените вокодери. Дополнително, синтетизираниот говор може да звучи монотон, поради ефектот на усреднување кој го имаат искористените методи за моделирање. Сепак, поради нивната мала пресметковна цена и мемориски потреби, системите за параметарска синтеза се употребуваат во многу апликации кои имаат потреба од локална (оффлайн) синтеза на говор на вгнездени и мобилни уреди, како на пример во системите за навигација, уреди за преведување, мобилни телефони и сл. Овие системи се употребуваат и во склоп на асистивните технологии, како на пример говорните табли за **аугментирана и алтернативна комуникација (AAC)**¹⁶ кои се наменети за луѓе со вокален хендикеп, како и за синтеза на говор во склоп на читачите на еcran кои ги користат луѓето со визуелна попреченост. За последниве, од најголемо значење е големиот степен на контрола на синтетизираниот говор, особено неговата брзина, притоа сочувувајќи ја разбираливоста на говорот; природноста е од второстепено значење.

Синтеза со длабоко учење

Со појавата на длабокото учење, види Глава и , се појави и нова парадигма за синтеза на говор. Како и во многу други области, длабокото учење донесе револуција и во областа на синтезата

¹³анг. *Hidden Markov Models*

¹⁴анг. *Gaussian Mixture Models*

¹⁵Сликата е превземена од вториот том Супрасегментална фонетика и фонологија, на новата Фонетика и фонологија на македонскиот стандарден јазик <http://ical.manu.edu.mk/index.php/publications>

¹⁶анг. *Augmentative and Alternative Communication*

на говор. Навистина, овие системи се способни да генерираат говор кој звучи многу природно, на моменти дури и без разлика од природниот говор.¹⁷

Системите за синтеза на говор со длабоко учење се базирани на длабоките невронски мрежи и тоа рекурентните невронски мрежи (RNN)¹⁸, односно мрежите LSTM¹⁹, кои се способни да моделираат временски секвенции. Во поново време, рекурентните мрежи речиси во целост се заменети со новите архитектури на невронски мрежи наречени трансформери²⁰, кои имаат изразено подобри перформанси поради паралелизацијата на пресметките. Овие системи се обучуваат со големи бази на говорен материјал, што претставува и најголема пречка во нивниот развој. Олеснителна околност е што делови од овие мрежи, поточно вокодерите, може да се тренираат и со нетранскрибиран говорен кој го има во изобилство. Уште една предност е нивната способност за преносно учење²¹, што овозможува употреба на модели предтренирани за јазици со многу ресурси за јазици со малку ресурси преку нивно нагодување²².

Најголемиот недостаток на овие системи е нивната голема пресметковна цена, која ги прави непрактични за употреба во реално време на хардвер со ограничени ресурси. Поради тоа, синтезата со овие модели се врши во облак, што повлекува потреба за постојана интернет конекција, но и ризици по приватноста на корисниците.

Синтеза на говор на македонски

Првите обиди за синтеза на говор (TTS) на македонски датираат од 1996 со предлог дизајн на систем за оваа намена на Јосифовски и др. Првиот функционален систем за синтеза на говор со база на единки на македонски јазик го реализираа Чунгурски и др. во 2009. Потоа, Геразов и др. го објавија системот „Зборувај македонски“ во 2011 базиран на синтеза со надоврзување на кратки говорни сегменти. Овој систем за првпат вклучуваше целосно функционален модул за генерирање на интонација и ритам дизајниран за македонскиот јазик.²³

Во поново време, Групата за говорни технологии на ФЕИТ (Говор@ФЕИТ) го разви гласот „Везилка“ базиран на синтеза со длабоко учење, со кој се постигнаа врвни резултати во поглед на природноста и квалитетот на синтетизираниот говор.²⁴ За примена во вгнездените системи, развивме и верзија на системот со поедноставен вокодер „Везилка-lite“, што овозможува работа на системот во реално време. Ова води до намалување на природноста во однос на положените вокодери, но е нужен компромис за примена на целосниот систем на вгнездена платформа.²⁵

Конечно, во соработка со Здружението за асистивни технологии „Отворете ги прозорците“, а со поддршка од македонската канцеларија на УНИЦЕФ, развивме глас за македонски за систем за параметарска синтеза на говор наречен „Сузе“, наменет за употреба во асистивните говорни технологии. Гласот „Сузе“ е слободен софтвер кој може да се преземе бесплатно од Google Play и да се инсталира на Андроид мобилен уред или таблет, или пак да се инсталира на компјутер и да се користи со читачите на екран како NVDA.²⁶ Исто така, тој може да се користи и во говорната табла Cboard која ја локализиравме за македонски и албански во рамките на истата соработка.

¹⁷ Прв систем кој го постигна ова беше системот Tacotron 2 во 2016 г. Обидете се самите да откриете кој говор е синтетизиран, а кој приден: <https://google.github.io/tacotron/publications/tacotron2/index.html>.

¹⁸ англ. Recurrent Neural Networks.

¹⁹ англ. Long Short-Term Memory.

²⁰ англ. Transformers.

²¹ англ. transfer learning.

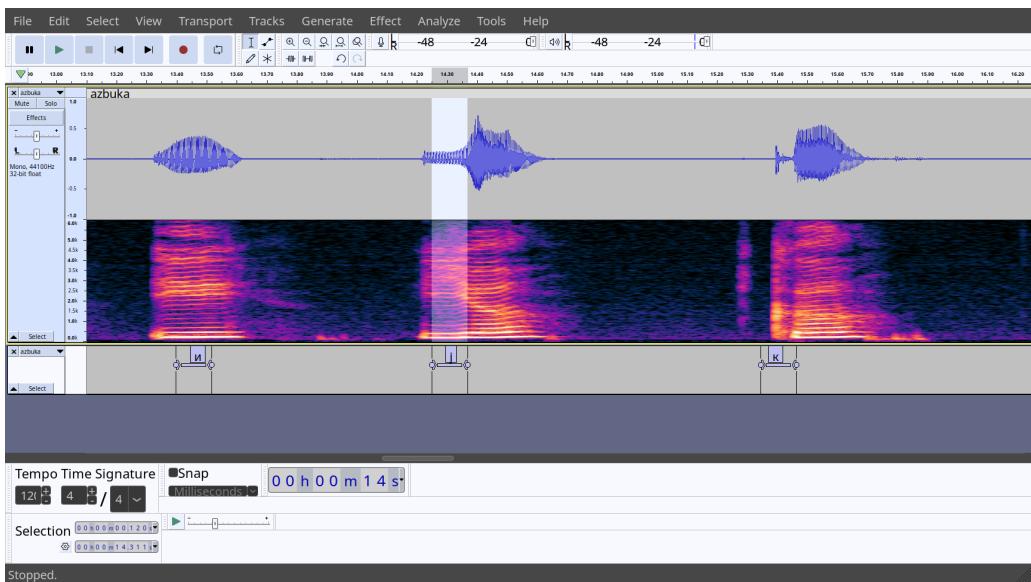
²² англ. fine-tuning.

²³ https://speech.feit.ukim.edu.mk/tts_zboruvaj.html

²⁴ https://speech.feit.ukim.edu.mk/tts_vezilka.html

²⁵ <https://speech.feit.ukim.edu.mk/ttslite.html>

²⁶ https://speech.feit.ukim.edu.mk/tts_suze.html



Сл. 7.11: Снимка на азбуката со означени букви.

Синтеза на говор во Python

Во овој дел ќе реализираме едноставен систем за синтеза на говор во Python. За оваа цел, најпрвин снимете ја азбуката со помош на Audacity, види Глава . Наместо да мануелно да ги издвоите буквите, ќе искористиме лабели за да ги означиме во снимката. Ова се прави преку додавање на трaka за лабели во која ќе ги означиме секоја од буквите. На Сл. 7.11 е прикажана снимка на азбуката со три означените букви: „и“, „ј“ и „к“. Внимавајте при означување на самогласките да означите дел од средината на самогласката, каде таа е најгласна и најстабилна.²⁷. На крајот, зачувајте ја снимката во wav формат и лабелите во txt формат. Ако ја отвориме датотеката со лабелите ќе видиме дека таа ја има следната структура:

0.936122	1.055814	а
1.809877	1.911301	б
3.077358	3.194530	в
4.221366	4.327830	г

Тука, првата колона е времето на почеток на буквата, втората колона е времето на крај на буквата, а третата колона е самата буква. Колоните се одвоени со знак tab . Оваа информација ќе ни биде потребна за да ги вчитаме лабелите во Python.

```
from os.path import join as pjoin

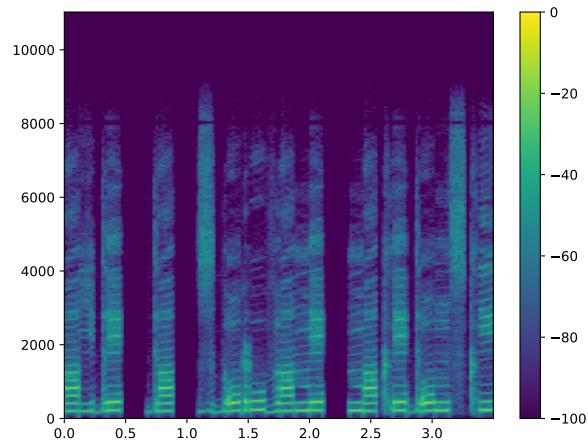
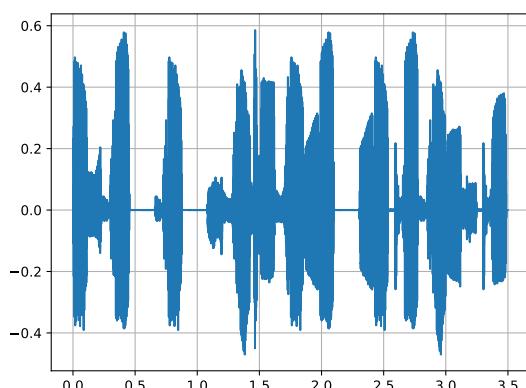
import numpy as np
from matplotlib import pyplot as plt
from scipy.io import wavfile as wf
import sounddevice as sd

import da

audio_path = "azbuka"
wav_name = "azbuka.wav"
file_name = pjoin(audio_path, wav_name)
fs, wav = wf.read(file_name)
wav = wav / 2**15

lab_name = "azbuka.txt"
labels = {}
```

²⁷Каде согласките внимавајте да не ја означите самогласката „шва“ /ə/, која ја има на пр. на почетокот на 'рбет, а која ја додаваме на согласките при изговарање на азбуката.



Сл. 7.12: Синтетизиран говор со Python и неговиот спектрограм.

```
with open(pjoin(audio_path, lab_name), 'r') as f:
    for line in f:
        start, stop, label = line[:-1].split("\t")
        labels[label] = float(start), float(stop)
```

Следно ќе ги издвоиме буквите од снимката и да ги зачуваме во еден Python речник. На крајот на речникот ќе додадеме и празно место односно тишина во вид на бел Гаусов шум со мала амплитуда, која ќе ја искористиме за да ги одделиме зборовите во синтетизираниот говор.²⁸

```
units = {}
for lab, (start, stop) in labels.items():
    unit = wav[int(start * fs): int(stop * fs)]
    units[lab] = unit
units[" "] = np.random.randn(int(0.200 * fs)) * 1e-4
```

Сега, кога ги имаме сите букви, можеме да ги искористиме за да синтетизираме говор! Притоа, за да избегнеме нагли преоди при надоврзување на буквите, ќе ги надоврзземе со нивно прелевање, која ја споменавме во Главата 5.1.

```
text = "ажде да зборуваме македонски"
synth = np.zeros(20 * fs)
t_fade = 0.010
fade_len = int(t_fade * fs)
pos = 0
for char in text:
    unit = units[char]
    unit_len = len(unit)
    unit = da.fade_in(unit, fs, t_fade)
    unit = da.fade_out(unit, fs, t_fade)
    if pos == 0:
        pos_fade = pos
    else:
        pos_fade = pos - fade_len
    # надоврзување со прелевање
    synth[pos_fade: pos_fade + unit_len] = unit
    pos += unit_len

synth = synth[: pos]
sd.play(synth, fs)
```

Добиениот синтетизиран говор и неговиот спектрограм се прикажани на Сл. 7.12.

²⁸Би можеле наместо бел шум да додадеме 0ти вектор, но тогаш ќе имаме проблеми при пресметување на логаритамот на спектрарот на сигналот за прикажување на спектрограмот.

Глава 8

Препознавање на звук и говор

Машинското учење, а особено длабокото учење, е метода која стана доминантна во голем број на инженерски и научни области. Во дигиталните аудиосистеми, машинското учење не само што ги поедностави комплексните решенија базирани на процесирање на сигналите и имплементација на големи множества на експертски правила, туку донесе решенија за проблемите кои не може да бидат решени воопшто. Еден од овие проблеми е и синтезата на природен говор, кој го разгледавме во претходното поглавје. Во ова поглавје ќе го разгледаме проблемот на препознавањето на говорот, кој е еден од најтешките проблеми во областа на дигиталното процесирање звук. За таа цел најпрвин ќе се запознаеме со основите на машинското учење.

8.1 Основи на машинското учење

Машинското учење е подобласт од областа вештачка интелигенција. Тоа опфаќа модели чии параметри се адаптираат на множество на податоци низ процес наречен на тренирање, односно учење (Bishop, 2006; Müller et al., 2016). На највисоко ниво разликуваме три парадигми на машинско учење, според кои ги делимите моделите од оваа област на три категории:

- учење со надзор¹,
- учење без надзор², и
- насочено учење³.

Учење со надзор

Кај учењето со надзор моделите се тренираат со познати целни излезни секвенции, уште наречени основна вистина⁴. Според типот на податоци во излезните секвенции разликуваме два генерални типови на модели и тоа модели за:

- **класификација** -- ако излезните секвенции се дискретни па може да се каже дека одговараат на затворено множество на класи или категории, и
- **регресија** -- ако излезните секвенции се континуирани.

Овие два типа на модели вообичаено ја имаат истата структура; клучната разлика е во типот на излезот кој го даваат.

Примери за употреба на модели за класификација во дигиталните аудиосистеми се алгоритмите за: препознавање на говор, препознавање на говорник, препознавање на јазик, препознавање

¹Анг. *supervised learning*.

²Анг. *unsupervised learning*.

³Анг. *reinforcement learning*.

⁴Анг. *ground truth*.

на музички инструмент, и препознавање на аудиосцена. Од друга страна, примери за модели за регресија се алгоритмите за: синтеза на интонација, спектар и временски аудиосигнал во системите за синтеза на говор, итн.

Учење без надзор

Алгоритмите за учење без надзор се од посебно значење денес, поради широката достапност на големи множества на податоци за кои нема ознаки, односно целни секвенции. Ова е така поради тоа што процесот на анотација вообичаено се базира на рачно внесување на ознаки од луѓе експерти, што целиот процес го прави временски захтевен и скап. Најпознатите претставници на алгоритмите за учење без надзор се алгоритмите за:

- **кластерирање** -- кои вршат групирање на влезните податоци во одреден број на класи кој може да биде внесен од корисникот или одреден автоматски. Најпознат алгоритам за оваа намена е алгоритамот на **K-средни вредности**⁵ кој го дели множеството податоци во K кластери описани со средната вредност на примероците во секој кластер, наречени и центроиди, и
- **намалување на димензионалноста** -- служат за визуелизација на распределбата на едно множество на повеќедимензионални податоци во две или три димензии преку нивна трансформација. Анализата на принципијелни (главни) компоненти (PCA)⁶ е еден пример за егзактна математичка метода за оваа намена, која е и реверзибилна. Тука спаѓаат стохастички алгоритми базирани на PCA, а најпознат алгоритам за оваа намена е алгоритамот за **t-дистрибуираното стохастичко врамување на соседи** (t-SNE)⁷.

Насочено учење

Во оваа категорија спаѓаат алгоритмите кои вообичаено се поистоветуваат со областа вештачка интелигенција. Тие имаат крајна цел, но таа не е доволна за директно тренирање на моделот. Тука спаѓаат алгоритмите за играње на компјутерски игри кои неодамна ги надиграа најдобрите човечки играчи⁸, но и алгоритмите за контрола на агенти во виртуелни симулации кои треба да научат некоја задача, на пример како оптимално да го придвижуваат човечкото тело за да стигнат од точка А до точка Б⁹.

Крајната цел на овие алгоритми е да победат, но таа е предалеку во иднината за директно да може да се искористи во учењето. За надминување на овој проблем, се прават стратегии за наградување на алгоритамот долж играта, со цел учењето да се насочи во правата насока. На пример, влезот на алгоритамот може да биде моменталната состојба на полето за игра, а тој треба на излез да го даде следниот потег кој би го довел во подобра позиција, односно со поголеми шанси да победи. Овие алгоритми вообичаено треба да балансираат помеѓу **истражување** и **искористување**, или експлорација и експлоатација. Тренирањето на моделите со насочено учење се одвива преку методата на залудни обиди, односно алгоритамот се пушта да игра самиот против себе милиони пати се додека не научи да победува.

Вообичаени чекори во примената на машинското учење

Постојат низа од чекори кои вообичаено се остваруваат во употребата на алгоритмите за машинско учење при решавањето на даден проблем:

- **анализа на влезните податоци / сигнали / слики** -- во оваа фаза се анализира типот на влезните податоци, се одредува дали постојат некомплетни податоци, се прави

⁵Анг. *K-means*.

⁶Анг. *Principal Component Analysis (PCA)*.

⁷Анг. *t-distributed Stochastic Neighbor Embedding*.

⁸Google DeepMind: Ground-breaking AlphaGo masters the game of Go

<https://www.youtube.com/watch?v=SUbqykXVx0A>

⁹Google's DeepMind AI Just Taught Itself To Walk <https://www.youtube.com/watch?v=gn4nRCC9TwQ>

визуелизација на нивната распределба, и статистиката на нивните карактеристики,

- поделба на множества за тренирање, валидација и тестирање -- дел од податоците се одвојува за конечна оценка на истренираниот модел, додека дел од остатокот се одвојува за континуирана валидација во фазата на тренирање,
- претпроцесирање и екстракција на обележја -- во оваа фаза се прави скалирање или нормализација на влезните податоци, притоа треба да се води особена сметка на поделбата на податоците на множества за тренирање/валидација/тестирање. Во случај на употреба на влезни податоци со голем број на корелирани димензии, како и во случај кога се работи со аудиосигнали, вообичаено во оваа фаза се врши екстракција на обележја¹⁰. За ова може да се применат методи како PCA, пресметување на спектар, спектрограми, итн. На пример, за препознавањето на говор најпознатите обележја се мел-Фреквенциските кепстрални коефициенти (MFCC),¹¹
- тренирање на моделот -- оптимизација на параметрите на моделот со употреба на множеството за тренирање и валидација, види Глава ,
- конечна евалуација на моделот -- ова се прави со одвоеното множество за тестирање со цел да се оцени како работи истренираниот модел за множество од нови, не видени влезни податоци.

Тренирање на моделите за машинско учење

Тренирањето на моделот, односно нагодувањето на неговите параметри се врши со помош на множеството на тренирање. Овој процес се изведува итеративно. Овде ќе претпоставиме дека се работи за алгоритам за учење со надзор, за кој се познати точните вредности кои сакаме да ги добиеме на неговиот излез. Во секоја итерација на учењето, се пресметува грешката која моделот ја прави врз база на познатите излези и неговите параметри се адаптираат во насока на намалување на оваа грешка. Чекорот со кој се врши адаптацијата на параметрите се нарекува и чекор на учење¹².

Тренирањето вообичаено завршува кога грешката на моделот не се подобрува за одредено ниво на толеранција последователно во избран број на епохи или ако е постигнат максималниот дозволен број на итерации. Ако тренирањето запре поради постигнување на максималниот број на итерации, а не дојде до заситување на опаѓањето на грешката на моделот, тогаш велиме дека настанало подтренирање¹³, односно дека алгоритамот има потенцијал да се дотренира за подобрување на неговите перформанси.

Проблемот со овој начин на тренирање тоа што параметрите на моделот може да се оптимизираат толку добро на множеството за тренирање што тој би ја изгубил својата способност за генерализација, односно точност за дотогаш не видени влезни податоци. Оваа појава се нарекува надтренирање¹⁴ и претставува посебен проблем во тренирањето на моделите за машинско учење. Постојат три главни начини тоа да се избегне:

- ограничување на моќта на моделот -- поедноставни модели не можат совршено да описват покомплексни влезни податоци и сигнали,
- регуларизација -- низа на методи кои ја ограничуваат способноста на моделите да описват комплексни податоци без намалување на нивната моќ. Степенот на регуларизација се нагодува преку коефициентот на регуларизација,

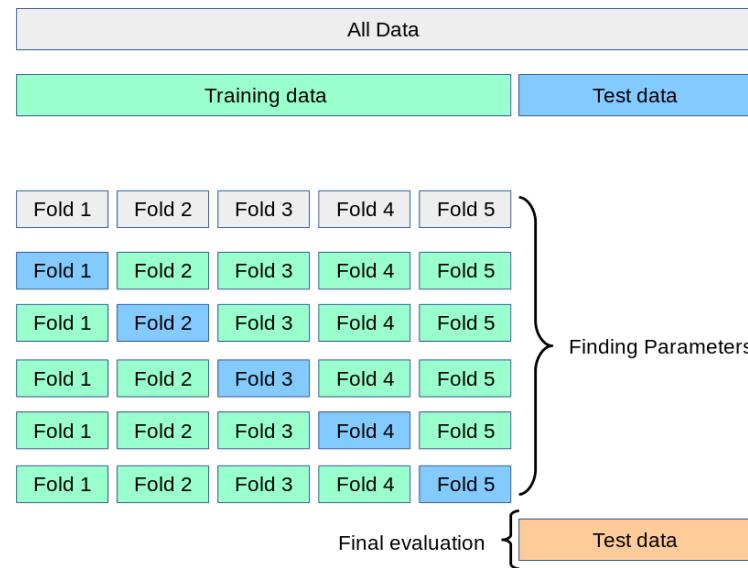
¹⁰Анг. *feature extraction*.

¹¹Анг. *mel-frequency cepstral coefficients (MFCC)*.

¹²Анг. *learning rate*.

¹³Анг. *underfitting*.

¹⁴Анг. *overfitting*.



Сл. 8.1: K -кратна меѓувалидација за $K = 5$.¹⁸

- рано запирање¹⁵ -- постапка која ја евалуира грешката на моделот на множеството за валидација секоја итерација и го запира тренирањето кога таа ќе започне да расте.

Освен за рано запирање, множеството за валидација служи и за избор и оптимизација на глобалните параметри на моделот и процесот на учење, како на пример: архитектурата на моделот, т.е. бројот на неговите параметри, чекорот на учење, коефициентот на регуларизација итн. Овие глобални, суштински параметри се нарекуваат **хиперпараметри**.

Изборот на хиперпараметри се прави врз основа на оптимизирање на перформансите на моделот на множеството за валидација. Притоа, за да се избегне влијанието на изборот на подмножество од влезните податоци на овој процес, што е од посебно значење кога располагаме со мало количество на влезни податоци, вообичаено се прави оптимизација на хиперпараметрите во јамка за **меѓувалидација**¹⁶.

- се дели влезното множество на податоци на множества за тренирање и тестирање,
- множеството на тренирање се дели на K подмножества од кои секое се зема за множество за валидација во една итерација на јамката за меѓувалидација, а останатите се употребуваат за тренирање, како што е прикажано на Сл. 8.1.

Ова се нарекува **K -кратна меѓувалидација**¹⁷ или уште **вкрстена валидација**.

Постојат низа на проблеми кои можат да настанат при употреба на овој едноставен пристап за меѓувалидација. Секој од нив се решава со адаптација на изборот на примероци во подмножествата за меѓувалидација:¹⁹

- ако датасетот е подреден по класите кои треба да се препознаат тогаш може да се случи некоја од класите да ја има само во тест множеството; поради ова вообичаено се употребува мешање²⁰ на датасетот пред неговата поделба,
- ако датасетот има групи, на пример говорници, тогаш може да се случи истиот говорник да биде и во множеството за тренирање и во тоа за тестирање; ова се надминува

¹⁵Анг. *early stopping*.

¹⁶Анг. *cross-validation*.

¹⁷Анг. *K-fold cross-validation*.

¹⁸Превземено од сајтот scikit-learn -- 3.1. Cross-validation: evaluating estimator performance https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

¹⁹scikit-learn -- 3.1. Cross-validation: evaluating estimator performance https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

²⁰Анг. *shuffle*.

со одредувањето на подмножествата врз база на групите податоци, т.н. K -кратна меѓувалидација по групи²¹ на датасетот,

- ако датасетот има нерамномерна распределба на класите, тогаш може во некое од подмножествата таа класа и воопшто да не се појави, па да не фигурира во евалуацијата; ова се надминува со стратификација на класите по подмножествата, т.н. K -кратна меѓувалидација со стратификација²², и
- во проблеми со временски серии во кои треба да се предвиди иднината врз база на минаците податоци, едноставната поделба на подмножества може да доведе алгоритамот да искористи информации од иднината за да направи подобра предикција; во тој случај, при оформување на подмножествата треба да се внимава на временската последователност на податоците.

Овие различни пристапи во меѓувалидацијата, како и нивната комбинација, се илустрирани на Сл. 8.2.

По завршување на меѓувалидацијата, моделот со избраните хиперпараметри се тренира повторно врз целото множество на влезни податоци, без множеството за тестирање. Вака тренираниот модел се евалуира на тест множеството. И овде, за избегнување на влијанието на изборот на тест множество врз оценката на моделот, што е од посебно значење кога имаме мало множество на податоци, може да се направи меѓуоценка на моделот преку јамка на вгнездена меѓувалидација. Во овој пристап, целото множество на податоци најпрвин се дели на N подмножества од кои секое се користи како тест множество во надворешната јамка, а остатокот се дели на M подмножества во стандардна јамка за меѓувалидација.

Невронски мрежи

Еден од најпознатите модели за машинско учење, којшто лежи во основата на длабокото учење се вештачките невронски мрежи (NN)²⁴. Основната градбена единка на невронските мрежи се софтверските неврони чиј начин на работа е инспириран од физиологијата на невроните описана во Главата 2.3 во скриптата по предметот Биомедицинска електроника.²⁵.

Вештачки неврон

Секој неврон има K влезови на кои се носи влезниот вектор $\mathbf{x} = [x_0, x_1, \dots, x_{K-1}]$. Влезовите на невронот се скалирани со тежински коефициенти, или тежини²⁶ w_k , кои можат да бидат и негативни. Активацијата на невронот a се добива преку сумирање на скалираните влезови. Конечно, излезот на невронот y се одредува преку излезната нелинеарност $f(a)$. Притоа, вообичаено се додава и коефициент на поместување²⁷ кој одговара на прагот на излезната нелинеарност.

$$y = f(a) = f\left(\sum_{k=0}^{K-1} w_k x_k + b\right) = f(\mathbf{w}\mathbf{x}^T + b) \quad (8.1)$$

Тука, $\mathbf{w} = [w_0, w_1, \dots, w_{K-1}]$ е векторот на тежини на овој неврон.

Може да видиме дека (8.1) всушност претставува поедноставен модел на физиологијата на невроните. Во него, влезниот сигнал се носи истовремено на сите влезови на невронот, а невронот пак врз база на тежинската сума на влезовите веднаш генерира излезен сигнал. Со

²¹Анг. group k -fold cross-validation.

²²Анг. stratified k -fold cross-validation.

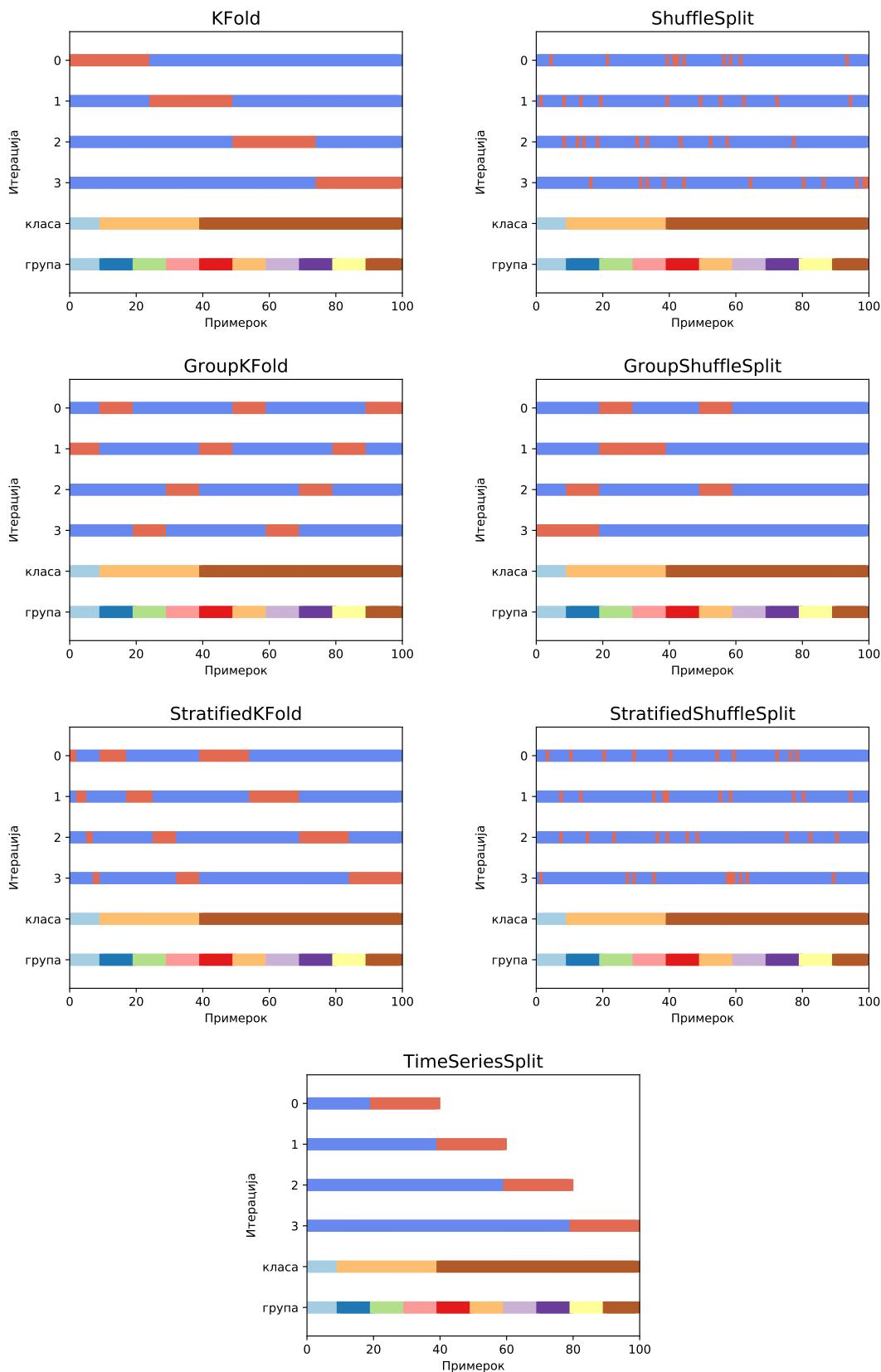
²³Генерирано со прилагоден код од овој на сајтот scikit-learn --Visualizing cross-validation behavior in scikit-learn https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html

²⁴анг. Neural Networks.

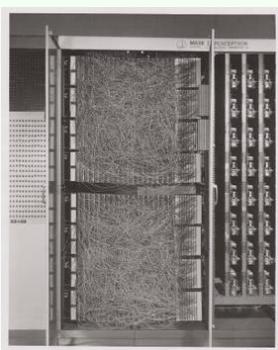
²⁵Материјалите од предметот Биомедицинска електроника можат да се превземат на <https://gitlab.com/feeit-freecourseware/biomedical-electronics>

²⁶Анг. weights.

²⁷Анг. bias.



Сл. 8.2: Напредни методи за K -кратна меѓувалидација за $K = 4$. Прикажани се множеството за тренирање (сино) и множеството за тренирање (црвено) за секоја итерација од јамката за меѓувалидација.²³



§ Дополнително. Првата имплементација на вештачки неврон, наречена перцептрон била изработена од американскиот психолог Франк Розенблат²⁹ во 1958 во Лабораторијата за аеронаутика во Корнел, САД. Тој бил најпрвин изведен во софтвер на IBM 704, а потоа и во хардвер. Хардверската изведба била направена за препознавање на слики регистрирани со 400 фотокелии, тежините на влезовите биле реализирани со потенциометри, а нивното учење со вртење на потенциометрите со електрични мотори

ова, моделот не ја зема в'обзир димензијата време, која што е клучна во работата на биолошките неврони. Постојат имплементации на неврони и невронски мрежи базирани на понапреден модел на физиологијата на невронот наречени **импулсни невронски мрежи**²⁸. Импулсните невронски мрежи иако не се сеуште комерцијално распространети, нудат високи перформанси, на пр. точност, при мала потрошувачка, а и не изискуваат големи податочни множества за тренирање. Исто така тие се од големо значење за дизајнот на неуроморфни електронски кола кои нудат високи перформанси, на пр. кај неуроморфните радарски системи.

Ако на влез се донесат низа на N вектори од влезните податоци \mathbf{x}_n , (8.1) преминува во:

$$y_n = f(a_n) = f(\mathbf{w}\mathbf{x}_n^T + b) \quad \text{за } n = 0, 1, 2, \dots N - 1, \quad (8.2)$$

$$\mathbf{y} = f(\mathbf{a}) = f(\mathbf{w}\mathbf{X}^T + b), \quad (8.3)$$

каде со \mathbf{X} е означена матрицата по чии редици се поставени влезните примероци \mathbf{x}_n .

Моделите за регресија базирани на употреба на еден неврон се нарекуваат **линеарна регресија**, додека оние за класификација се нарекуваат **логистичка регресија**.

Плитки невронски мрежи

Во наједноставниот случај, невронските мрежи имаат еден скриен слој и еден излезен слој на неврони, како што е прикажано на Сл. 8.3. Ваквите модели се нарекуваат **плитки** невронски мрежи. Секој неврон од скриениот слој е поврзан со секој од коефициентите на влезниот вектор \mathbf{x} . Секој неврон од излезниот слој пак е поврзан со секој неврон од скриениот слој. Поради ова поврзување, овој вид на слоеви се нарекуваат и **целосно поврзани** односно **густи**³⁰. Ова едноставна архитектура сепак им овозможува на плитките невронски мрежи да моделираат било која нелинеарна функција, па тие се уште познати и како **универзални апроксиматори**.

Излезот на една плитка невронска мрежа за даден влезен вектор на податоци \mathbf{x} може да ја пресметаме преку:

$$\mathbf{y}_h = f_h(\mathbf{a}_h) = f_h(\mathbf{W}_h \mathbf{x}^T + \mathbf{b}_h), \quad (8.4)$$

$$\mathbf{y} = f_o(\mathbf{a}_o) = f_o(\mathbf{W}_o \mathbf{y}_h + \mathbf{b}_o) = f_o(\mathbf{W}_o \cdot f_h(\mathbf{W}_h \mathbf{x}^T + \mathbf{b}_h) + \mathbf{b}_o), \quad (8.5)$$

каде со h се означени параметрите и излезите добиени од скриениот слој, а со o оние од излезниот слој. Овојпат, бидејќи во секој слој може да имаме повеќе неврони, нивните тежини се

²⁸Wikipedia: Spiking Neural Network https://en.wikipedia.org/wiki/Spiking_neural_network

²⁹Wikipedia -- Perceptron <https://en.wikipedia.org/wiki/Perceptron>

Од анонимен извор - http://www.peoples.ru/science/psychology/frank_rosenblatt/, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=64998425>

Од Source (WP:NFCC#4), Fair use, <https://en.wikipedia.org/w/index.php?curid=47541432>

³⁰Анг. *fully connected* и *dense*.

распоредени долж редиците на матриците за тежини \mathbf{W} а нивните коефициенти на поместување во векторите колони \mathbf{b} .

На тој начин, излезот на мрежата се добива со процесирање на влезните податоци слој по слој се додека не се дојде до излезниот слој на мрежата.³¹. Овој процес се нарекува и **пропагација нанапред**.³²

Повторно, ако на влез се донесат низа на N примероци од влезните податоци \mathbf{x}_n добиваме:

$$\mathbf{Y}_h = f_h(\mathbf{A}_h) = f_h(\mathbf{W}_h \mathbf{X}^T + \mathbf{b}_h), \quad (8.6)$$

$$\mathbf{Y} = f_o(\mathbf{A}_o) = f_o(\mathbf{W}_o \mathbf{Y}_h + \mathbf{b}_o) = f_o(\mathbf{W}_o \cdot f_h(\mathbf{W}_h \mathbf{X}^T + \mathbf{b}_h) + \mathbf{b}_o), \quad (8.7)$$

Во рамките на една невронска мрежа, излезните нелинеарности на невроните во скриените и излезните слоеви вообичаено се разликуваат. Типичен избор за нелинеарности во скриениот слој се:

- **сигмоида** $\sigma(a) = \frac{1}{1+e^{-a}}$ -- дава излез во опсег $0 \dots 1$,
- **тангенс хиперболикум** $\tanh(a) = \frac{e^{2a}-1}{e^{2a}+1}$ -- дава излез во опсег од -1 до 1 , и
- **полубранов насочувач** $ReLU(a) = \begin{cases} a & \text{ако } a > 0 \\ 0 & \text{поинаку} \end{cases}$ -- предноста на употреба на полубрановиот насочувач е поедноставната пресметка на излезот на невроните, подобрата пропагација на градиентот во процесот на тренирање, како и реткоста на активација на невроните -- при случајна иницијализација на тежините половина од невроните ќе дадат 0 . Уште една мотивација за употреба на оваа нелинеарност е биолошката аналогија која се должи на нејзината асиметричност.

За невроните во излезниот слој вообичаено се користат:

- **сигмоида** -- за класификација,
- **софтмакс** $f(a_j) = \frac{e^{a_j}}{\sum_{j=0}^{J-1} e^{a_j}}$ -- за класификација со повеќе излезни класи J , каде a_j е активацијата на невронот кој соодветствува на класата j ; софтмакс функцијата го нормализира збирот на излезот на сите излезни неврони, па може да се каже дека ни дава апроксимација на веројатноста на секоја од класите $f(a_j) \approx P(y = j | \mathbf{a})$,
- **линеарна** $f(a) = a$ -- кај моделите за регресија.

Длабоко учење

Со додавање на повеќе скриени слоеви во невронската мрежа се добиваат **длабоки невронски мрежи** (DNN)³⁴. Длабокото учење е подобласт во машинското учење која ги опфаќа моделите базирани на повеќеслојни вештачки **невронски мрежи** (Goodfellow et al., 2016). Иако плитките невронски мрежи претставуваат универзални апроксиматори, тоа важи само кога моќта на мрежата, одредена од бројот на неврони во скриениот слој, е доволно голема. Се покажува дека додавање на неврони во скриениот слој, односно додавање на широчина на мрежата не е толку ефикасно како додавање на длабочина на мрежата.

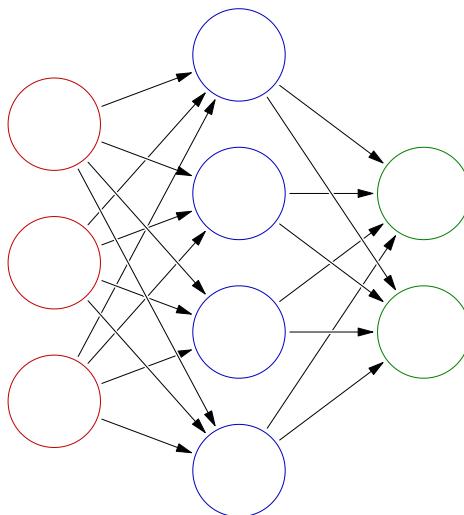
Невронските мрежи кои постигнуваат денес надчовечки перформанси имаат и до 1000 скриени слоеви. Поради комплексноста на тренирањето на милионите параметри на длабоките невронски мрежи, тие доживуваат процут дури на почетокот од XXI век, по општата достапност

³¹Длабоките невронски мрежи имаат повеќе скриени слоеви, види понатаму во Главата

³²Анг. *forward pass* или *feed forward*.

³³Модифицирано од Glosser.ca - Own work, Derivative of File:Artificial neural network.svg, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=24913461>

³⁴Анг. *Deep Neural Networks*.



Сл. 8.3: Плитка невронска мрежа со еден скриен слој составен од четири неврони (сино), и еден излезен слој составен од 2 неврони (зелено). Влезните податоци вообичаено се пртаат како влезен слој (првено).³³

на графичките процесори кои овозможуваат паралелна имплементација на алгоритмите за тренирање.

Постојат четири основни подвидови на длабоките невронски мрежи:

- **обични длабоки невронски мрежи** -- овие се нарекуваат и **повеќениновски перцептрони (MLP)**³⁵ во чест на првиот перцепtron,
- **конволуциски невронски мрежи (CNN)**³⁶ -- базирани на конволуција со филтри се употребуваат за процесирање на 1Д, 2Д и 3Д сигнали, а речиси секогаш за обработка на слики,
- **рекурентни невронски мрежи (RNN)**³⁷ -- базирани на повратна врска на невроните од скриените слоеви што им дава еден вид на меморија за претходните примероци, се употребуваат за процесирање на 1Д сигнали,
- **трансформери**³⁸ -- нов вид на длабоки невронски мрежи кои се базираат на механизмот на **внимание**³⁹ за моделирање на временски секвенции, кои во последно време ги истиснуваат од употреба RNN.

Тренирање на невронските мрежи

Проблемот на тренирање на невронските мрежи се сведува на промена на тежините и коефициентите на поместување на секој од невроните во насока на намалување на грешката која ја прави мрежката (Nielsen, 2015). Еден начин тоа да се направи е со примена на алгоритамот **спуштање по градиентот (GD)**⁴⁰, кој е итеративен алгоритам за пронаоѓање на минимумот на дадена функција. Конкретно, за да дојдеме до минимумот на функцијата $f(x)$ од моментната позиција x_i , треба да направиме чекор во насока спротивна на градиентот за таа позиција:

$$x^{i+1} = x^i - \frac{df}{dx} \cdot \Delta x, \quad (8.8)$$

каде со Δx е означен големината на чекорот кој го земаме.

³⁵Анг. *multi layer perceptron*.

³⁶Анг. *Convolutional Neural Networks*.

³⁷Анг. *Recurrent Neural Networks*.

³⁸Анг. *Transformers*.

³⁹Анг. *attention*.

⁴⁰Анг. *gradient descent*.

За примена на GD при тренирањето на невронските мрежи дефинираме **функција на грешка** $\mathcal{L}(y, \tilde{y}) = \mathcal{L}(y, g(\boldsymbol{\theta}, \mathbf{x}))$ која зависи од точниот, т.е. целниот излез y и излезот добиен од мрежата $\tilde{y} = g(\boldsymbol{\theta}, \mathbf{x})$, кој пак зависи од параметрите на невронската мрежа $\boldsymbol{\theta}$ и влезниот вектор \mathbf{x} . Притоа, $\boldsymbol{\theta} = [\theta_0, \theta_1, \dots, \theta_{L-1}] = [\mathbf{W}_0, \mathbf{b}_0, \mathbf{W}_1, \mathbf{b}_1, \dots, \mathbf{W}_{L-1}, \mathbf{b}_{L-1}]$ каде L е бројот на слоеви на мрежата. Тогаш имаме:

$$\theta_l^{i+1} = \theta_l^i - \frac{\partial \mathcal{L}}{\partial \theta_l} \cdot \eta \quad \text{за } \theta_l \in \boldsymbol{\theta}, \quad (8.9)$$

каде $\frac{\partial \mathcal{L}}{\partial \theta_l}$ е парцијалниот извод на функцијата на грешка во однос на параметарот θ_l , а η е чекорот на учење. Притоа, за пресметување на парцијалниот извод се користи правилото за пресметување на **извод на сложена функција**⁴¹, односно:

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \frac{\partial \mathcal{L}}{\partial \tilde{y}} \cdot \frac{\partial \tilde{y}}{\partial y_{L-2}} \cdots \frac{\partial y_{l+1}}{\partial \theta_{l+1}} \cdot \frac{\partial y_l}{\partial \theta_l}, \quad (8.10)$$

Каде y_l е излезот на l -от слој на мрежата. На пример, за модел составен од еден неврон, градиентот во однос на тежината w_0 , ќе биде:

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{\partial \mathcal{L}}{\partial \tilde{y}} \cdot \frac{\partial \tilde{y}}{\partial w_0}. \quad (8.11)$$

Користејќи го правилото за извод на сложена функција, пресметување на градиентот за прилагодување на сите параметри $\boldsymbol{\theta}$ на невронската мрежа започнува со пресметка на градиентите за излезниот слој, па за последниот скриен слој и оди наназад до почетокот на мрежата. Овој процес се нарекува **пропагација наназад**.⁴²

За GD да може да се употреби за тренирање на невронски мрежи, мора функцијата на грешка, како и сите излезни нелинеарности на невроните во мрежата да бидат диференцијабилни. Инаку не би можел да се пресмета градиентот за секој од параметрите. Ова е причината за употреба на сигмоидата наместо пресекување на активацијата со остатар праг, како што било направено во оригиналниот перцептрон, кај кого важело:

$$y = f(a) = \begin{cases} 1 & \text{ако } a > 0,5 \\ 0 & \text{поинаку} \end{cases}. \quad (8.12)$$

Денес најчесто се употребуваат следните функции за грешка:

- **средна квадратна грешка** $MSE = \frac{1}{N} \sum_{n=0}^{N-1} (y - \tilde{y})^2$ -- основна функција на грешка за регресија и бинарна класификација,
- **меѓу-ентропија** $CE = -\frac{1}{N} \sum_{n=0}^{N-1} y \ln \tilde{y} + (1 - y) \ln(1 - \tilde{y})$ -- кај модели за класификација со излезна нелинеарност сигмоида; нејзиниот извод има подобри карактеристики,
- **логаритамска веројатност**⁴³ $LL = -\ln \tilde{y}$ -- кај моделите со софтмакс функција на излез.

Во рамките на функцијата на грешка се вклучува и дел за регуларизација. Најчесто тоа е регуларизација на $L2$ нормата на параметрите на моделот $\boldsymbol{\theta}$. На пример, при употреба на средната квадратна грешка, би имале:

$$L(y, \tilde{y}, \boldsymbol{\theta}) = L(y, \tilde{y}) + \lambda \sum_{l=0}^{L-1} (\mathbf{W}_l^T \mathbf{W}_l + \mathbf{b}_l^T \mathbf{b}_l), \quad (8.13)$$

каде λ е **кофициентот на регуларизација**, а со $\mathbf{W}_l^T \mathbf{W}_l$ и $\mathbf{b}_l^T \mathbf{b}_l$ се добиваат сумите од квадратите за сите параметри на невроните од слојот l .

⁴¹Анг. *chain-rule*.

⁴²Анг. *backpropagation*.

⁴³Анг. *log-likelihood*.

При тренирањето на невронски мрежи, поради тоа што вообичаено се работи со големи множества на влезни податоци, станува неисплатливо градиентот да се пресметува за сите влезни примероци. Другиот екстрем е адаптацијата на параметрите да се прави со градиентот пресметан за секој од примероците земен по случаен избор. Оваа варијанта на алгоритамот се нарекува **стохастичко спуштање по градиентот (SGD)**⁴⁴. Изминувањето на целото множество за тренирање се нарекува и **епоха**. Случајниот избор на влезните примероци технички се изведува преку случајно мешање на датасетот пред секоја епоха, по што следи секвенцијално земање на примероците.

Сепак, пресметката на градиентот по примерок не дава добра естимацијата на вистинскиот градиент на функцијата на грешка. Поради тоа, најчесто се употребува компромисно решение во кое се зема подмножество, или **купче**⁴⁵ примероци од множеството за тренирање по случаен избор, и се врши адаптација на параметрите за пресметаниот градиент.

Оваа верзија на GD алгоритамот се нарекува **спуштање по градиентот со купчиња (MBGD)**⁴⁶. Технички постои разлика помеѓу MBGD и BGD -- кај MBGD се врши прилагодување на параметрите за секое купче, додека кај BGD тоа се прави по поминување низ целото множество за тренирање. Скоро секогаш за тренирање на невронските мрежи, а и други алгоритми за машинско учење, се употребува MBGD алгоритамот но под името SGD.

При употреба на SGD, чекорот на учење е еден од најважните параметри во тренирањето на невронските мрежи. Ако е преголем оптимизацијата може да го натфрли минимумот на функцијата на грешка, додека пак, ако е премал, на алгоритамот ќе му бидат потребни многу итерации за да заврши тренирањето. Затоа вообичаено се употребуваат **стратегии на промена** на чекорот на учење⁴⁷. Една единственава, а често употребувана, стратегија е чекорот да се намалува во тек на тренирањето. Постојат и понапредни алгоритми, каков што е **Адам**, кои предвид ги земаат првиот и вториот момент, односно брзината и забрзувањето, на промената на градиентот во однос на претходните итерации за адаптација на чекорот на учење.

Еден голем проблем со употребата на SGD алгоритамот е можноста алгоритамот да заглави во локален минимум, по цена на промашување на глобалниот минимум. Овој ризик е поголем кај помалите NN, додека кај длабоките NN, веројатностите за постоење на локален минимум се минорни. Кај нив проблемот се сведува на заглавување на SGD во рамни региони на функцијата на грешка. Двата проблеми можат да се решат преку стратегии за промена на чекорот на учење, кои се базираат на негова реиницијализација на голема вредност по одреден број на итерации.

8.2 Препознавање на звучен извор

Во овој дел ќе примениме невронска мрежа за **препознавање на звучен извор**⁴⁸ врз база на аудиосигналот кој тој го генерира. За таа цел ќе искористиме дел од аудиофайлите кои ги користевме досега, а дел ќе позајмиме од предметот **Електроакустика** на ФЕИТ.⁴⁹

```
import numpy as np
from matplotlib import pyplot as plt

audio_path = 'audio/'
file_names = [
    'Glas.wav',
    'Solzi.wav',
    'Violina.wav',
    'Tapan.wav',
```

⁴⁴Анг. *stochastic gradient descent*.

⁴⁵Анг. *batch* или *mini-batch*.

⁴⁶Анг. *mini batch gradient descent*.

⁴⁷Анг. *learning rate scheduling*.

⁴⁸Анг. *sound source recognition (SSR)*.

⁴⁹<https://github.com/FEEIT-FreeCourseWare/Electroacoustics>

```
'Zurla.wav',
]
y_labels = ['voice', 'guitar', 'violin', 'drum', 'zurla']
n_labels = len(y_labels)
xs = []
for file_name in file_names:
    fs, x = wavfile.read(audio_path+file_name)
    print(fs) # провери дали имаат иста fs
    x = x / 2**15
    x = x[: min([x.size, fs*5])]
    xs.append(x)
```

Притоа, ако аудиосигналот е подолг од 5 s истиот ќе го скратиме за да не доминира во множеството на податоци.

Анализа на влезните податоци

За запознавање со аудиосигналите кои се дел од датасетот кој го создадовме може да го искористите знаењето кое досега го стекнавте на предметов. За поголема компактност, тука директно ќе прејдеме на поделбата на датасетот на подмножества.

Поделба на податоците на множества за тренирање и тестирање

Во оваа демонстрација на процесот на примена на машинското учење ќе направиме само едноставна поделба на податоците на множества за тренирање и тестирање. Ќе одвоиме 50% од секој од аудиосигналите за генерирање на множеството за тест.

```
test_coef = .5 # 50%
x_trains = []
x_tests = []
for x in xs:
    x_len = x.size
    test_len = int(x_len * test_coef)
    x_train = x[:-test_len]
    x_test = x[-test_len:]
    x_trains.append(x_train)
    x_tests.append(x_test)
```

Екстракција на обележја

Вообичаена практика е кога се работи за препознавање на звучните извори, или на пример за препознавање на говор⁵⁰ или говорник⁵¹, да се употребат спектрални обележја за опис на аудиосигналите. Тука, ќе го искористиме амплитудниот спектар како наједноставна репрезентација на спектралната содржина, екстрагиран со методата на прозорци. Притоа ќе употребиме краток прозорец од 256 примероци, што соодветствува на 5,8 ms за фреквенција на земање примероци од 44,1 kHz, кој ќе ни даде погруба претстава на распределбата на енергијата долж фреквенциите на спектарот. На овој начин, ќе го избегнеме влијанието на позицијата на хармониите врз невронската мрежа. Со тоа ќе обезбедиме мрежата да ја научи спектралната анвелопа, односно бојата на тонот, наместо неговата висина.

Во исто време ќе ги кодираме лабелите со бинарни вектори со една 1 на позицијата на класата која ја претставува секоја од рамките на аудиосигналот и 0 на другите позиции. Овој начин на кодирање на лабелите се нарекува *one-hot encoding* и секогаш се применува во машинското учење при класификација на повеќе класи.⁵²

⁵⁰Анг. *automatic speech recognition (ASR)*.

⁵¹Анг. *automatic speaker recognition*.

⁵²На овој начин во излезниот слој од невронската мрежа ќе има толку неврони колку што има класи, а секој од нив ќе ја претставува една од класите. Со тоа, невронската мрежа ќе може да ги предвиди сите класи и да даде проценка за веројатноста на секоја од нив. Во спротивно, ако користевме еден неврон во излезниот слој, тој ќе имаше многу потешка задача да ги предвиди класите кодирани како низа од цели броеви.

```

import da

x_train = None
x_test = None
y_train = None
y_test = None
for i, (x_train_sig, x_test_sig) in enumerate(
    zip(x_train_sigs, x_test_sigs)):
    --, __, spectrogram = da.get_spectrogram(
        fs, x_train_sig, n_win=256, plot=False)
    x_train_feats = spectrogram.T
    mask_spectrogram = np.all(x_train_feats < -80, axis=1)
    x_train_feats = x_train_feats[~mask_spectrogram, :60]
    # генерирај бинарен излезен вектор
    n_samples = x_train_feats.shape[0]
    y_train_targets = np.zeros([n_samples, n_labels])
    y_train_targets[:, i] = 1

    # исто за тест множеството
    --, __, spectrogram = da.get_spectrogram(
        fs, x_test_sig, n_win=256, plot=False)
    x_test_feats = spectrogram.T
    mask_spectrogram = np.all(x_test_feats < -80, axis=1)
    x_test_feats = x_test_feats[~mask_spectrogram, :60]
    n_samples = x_test_feats.shape[0]
    y_test_targets = np.zeros([n_samples, n_labels])
    y_test_targets[:, i] = 1

if x_train is None:
    x_train = x_train_feats
    x_test = x_test_feats
    y_train = y_train_targets
    y_test = y_test_targets
else:
    x_train = np.concatenate((x_train, x_train_feats), axis=0)
    x_test = np.concatenate((x_test, x_test_feats), axis=0)
    y_train = np.concatenate((y_train, y_train_targets), axis=0)
    y_test = np.concatenate((y_test, y_test_targets), axis=0)

```

Притоа ги задржавме само најниските 60 бинови од спектрограмот кои одговараат на фреквенции до 10.3 kHz, и правиме негово транспонирање за примероците да бидат редици а обележјате колони во матриците на влезни податоци.

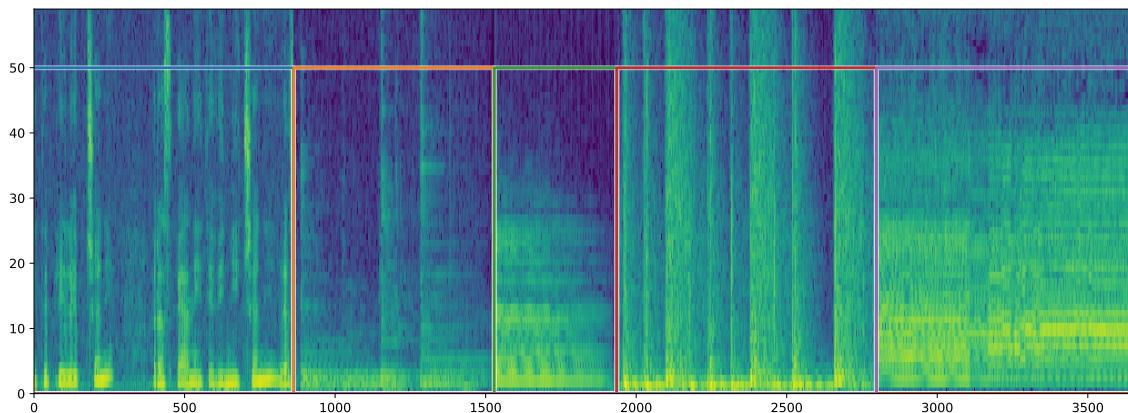
За визуелизација на спектрограмите за различните класи и потврда дека тие можат да се искористат за нивно разликување ќе ги прикажеме на графикон преклопени со типот на инструментот. Резултатот може да се види за тест множеството на Сл. 8.4. Може да видиме дека постојат јасни разлики во спектрите на сигналите за различните звучни извори, па може да очекуваме дека ќе добиеме добра точност на предикција на нашиот класификатор.

```

x_axis = np.arange(x_train.shape[0])
y_axis = np.arange(x_train.shape[1])
plt.figure()
plt.imshow(x_train.T, aspect='auto', origin='lower',
           extent=[0, x_axis[-1], 0, y_axis[-1]], vmin=-100,)
plt.plot(y_train*50, lw=2)

x_axis = np.arange(x_test.shape[0])
y_axis = np.arange(x_test.shape[1])
plt.figure()
plt.imshow(x_test.T, aspect='auto', origin='lower',
           extent=[0, x_axis[-1], 0, y_axis[-1]], vmin=-100,)
plt.plot(y_test*50, lw=2)

```



Сл. 8.4: обележја екстрактирани од множеството за тестирање со задржување на првите 80 коефициенти, или до фреквенција од 13.8 kHz, на спектарот за рамки од 256 примероци, или околу 5,8 ms од сигналот. Врз нив се исцртани петте класи: глас, гитара, виолина, тапан, и зурла.

Тренирање на моделот

За оваа анализа ќе ја искористиме имплементацијата на невронска мрежа за класификација во класата `MLPClassifier` во модулот `neural_network` на пакетот за машинско учење **Сајкит-лерн**.⁵³

Пред тренирањето на моделот може да направиме случајно мешање на примероците во множеството за тренирање и тестирање. Ова мешање на примероците од различните класи, ќе придонесе кон подобрување на апроксимацијата на градиентот во SGD алгоритамот. За тоа ќе го искористиме `random` модулот на NumPy кој содржи **генератор на псевдо-случајни броеви**⁵⁴. Тоа е машина со конечен број на состојби кој генерира секвенца на броеви чија статистика е како на случајна низа на броеви, иако начинот на кој се генерираат е детерминистички. Така, со поставување на состојбата на генераторот со наредбата `np.random.seed`, при секое извршување на кодот ќе ги добиеме истите резултати. Во конкретниот случај тоа ќе биде истото мешање на датасетот. Ова е вообичаена практика во областа машинско учење во која случајноста игра голема улога, на пр. при иницијализација на параметрите на моделот, со што се обезбедува репродуцибилност на резултатите.

```
# измешај ги податоците
np.random.seed(42)
train_ind_shuffle = np.random.permutation(x_train.shape[0])
x_train = x_train[train_ind_shuffle]
y_train = y_train[train_ind_shuffle]
test_ind_shuffle = np.random.permutation(x_test.shape[0])
x_test = x_test[test_ind_shuffle]
y_test = y_test[test_ind_shuffle]
```

Уште подобро при тренирањето на невронските мрежи е податоците да се измешаат на почетокот од секоја епоха. Тоа може да се овозможи во `MLPClassifier` со аргументот `shuffle`. Тој и низата други параметри ни овозможува да ги поставиме сите хиперпараметри за тренирање на нашиот класификатор.

```
from sklearn import neural_network
from sklearn import metrics

mlp = neural_network.MLPClassifier(
    hidden_layer_sizes=(100, 20, 10), # број на неврони во скриените слоеви
    activation='relu', # нелинеарност на невроните во скриените слоеви
```

⁵³sciKit-Learn <http://scikit-learn.org/stable/>

⁵⁴Анг. *pseudo-random number generator (PRNG)*.

```

learning_rate_init=0.01, # чекор на учење
alpha=1e-4, # коефициент на L2 регуларизација
solver='adam', # алгоритам за тренирање
shuffle=True, # мешање на податоците во секоја епоха
random_state=42, # иницијализација на PRNG
max_iter=3000, # максимален број на итерации
tol=1e-9, # толеранција за заситување на тренирањето
n_iter_no_change=40, # запирање на тренирањето заради заситување
early_stopping=False, # рано запирање со множество за валидација
verbose=1, # побогат испис при тренирањето
)
mlp.fit(x_train, y_train) # тренирање на моделот

```

```

Iteration 1, loss = 9.14906290
Iteration 2, loss = 2.60331358
Iteration 3, loss = 2.40171081
Iteration 4, loss = 2.32905906
Iteration 5, loss = 2.22766382
Iteration 6, loss = 2.12148965
Iteration 7, loss = 2.04304694
Iteration 8, loss = 1.97768373
Iteration 9, loss = 1.88609207
...
Iteration 491, loss = 0.29752739
Training loss did not improve more than tol for 40 consecutive epochs. Stopping.

```

Може да видиме дека тренирањето на моделот завршува за 491 итерација при постигната грешка од 0,2975. Во Сајкитлерн во невронските мрежи за класификација на повеќе класи се употребува софтмакс како излезен слој, а меѓу-ентропијата како функција на грешка.

Евалуација на моделот

За да направиме предвидување врз база на множеството за тестирање може да се послужиме со `mlp.predict_proba` функцијата која ќе ни го даде излезот на невронската мрежа, односно `mlp.predict` функцијата која ќе ни ја даде предвидената излезна класа како бинарен вектор користејќи праг од 0,5.

За евалуација на мрежата, дополнително е имплементирана функцијата `mlp.score` која ќе ја искористиме за да видиме која е точноста на предвидувањата на нашата мрежа.

```

accuracy = mlp.score(x_test, y_test)
print(accuracy)

```

```
0.885792349726776
```

Може да видиме дека нашиот модел постигнува 88,58% точност што е далеку над точноста на случаен избор од 20%! Дополнителни подобрувања може да оствариме со земање на подолги аудиосигнали од нашите звучни извори, употреба на понапредни обележја, како и оптимизација на хиперпараметрите на невронската мрежа.

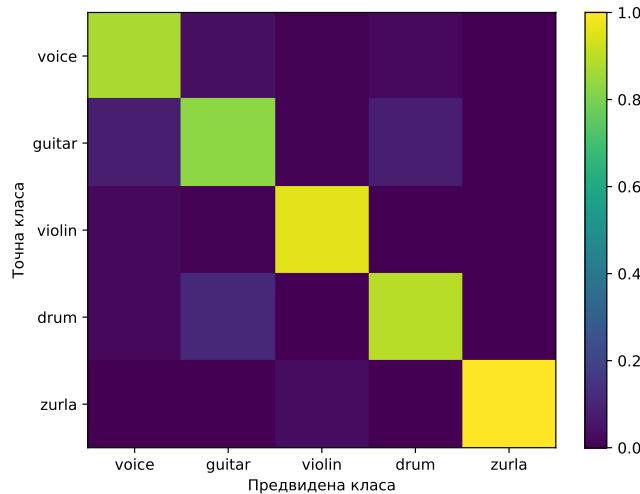
За дополнително да воочиме каде најмногу греши невронската мрежа може да ја пресметаме матрицата на забуна.⁵⁵ За да ја пресметаме, ќе направиме предвидување на класите со тест множеството.

```

y_pred_prob = mlp.predict_proba(x_test)
cm = metrics.confusion_matrix(
    y_test.argmax(axis=1),
    y_pred_prob.argmax(axis=1)
)
print(cm)

```

⁵⁵Анг. *confusion matrix*.



Сл. 8.5: Матрица на забуна за истренираната невронска мрежа.

```
[[812  26   2  22   0]
 [ 78 514   3  71   1]
 [ 19   4 384   0   0]
 [ 19  72   0 769   2]
 [  2   2  13   0 845]]
```

Редиците претставуваат точните класи, а колоните класите предвидени од невронската мрежа. Истата можеме да ја нормализираме за да ја добиеме процентуалната грешка која ја прави мрежата по предвидена класа. Исто така, за појасен приказ ќе ја прикажеме матрицата на забуна преку топлинска мапа⁵⁶ на Сл. 8.5

```
cm = cm / cm.sum(axis=0)
print(cm)

fig, ax = plt.subplots()
im = ax.imshow(cm, aspect='auto', interpolation='nearest', vmax=1, vmin=0)
ax.set(
    xticks=np.arange(cm.shape[1]),
    yticks=np.arange(cm.shape[0]),
    xticklabels=y_labels,
    yticklabels=y_labels,
    ylabel='Точна класа',
    xlabel='Предвидена класа',
)
fig.colorbar(im)
```

```
[[0.87311828 0.0420712 0.00497512 0.02552204 0.          ]
 [0.08387097 0.83171521 0.00746269 0.08236659 0.00117925]
 [0.02043011 0.00647249 0.95522388 0.          0.          ]
 [0.02043011 0.11650485 0.          0.89211137 0.00235849]
 [0.00215054 0.00323625 0.03233831 0.          0.99646226]]
```

Може да се види дека највеќе грешки невронската мрежа прави при одлучување помеѓу класите глас и гитара, како и гитара и тапан, кои имаат сличности во спектрите во дел од примероците во тест множеството прикажани на Сл. 8.4.

⁵⁶Анг. *heat map*.

✓ Задача за дома.

1. Дополни го кодот за екстракција на обележја со банка на 23 филтри која ќе ја пресмета средната енергија во секој фреквенциски подопсег. Ова ќе овозможи примена на подолги прозорци за пресметка на спектрограмите што ќе ни даде временско усреднување на сигналите. Одреди колку ќе се подобрат перформансите на невронската мрежа.
2. Искористи ја функцијата за меѓувалидација со мешање `ShuffleSplit` имплементирана во Сајкитлерн и оптимизирај го изборот на хиперпараметри на невронската мрежа. Која е најдобрата точност што може да се постигне?

8.3 Автоматско препознавање на говор

Автоматското препознавање на говор (ASR)⁵⁷ претставува клучна алка во остварувањето на сонот за природна и непречена комуникација помеѓу човекот и машината, без потреба од помагала и интерфејси. Навистина, најприродниот начин на којшто лубето ги пренесуваат своите идеи и намери е преку говорот. Поради тоа, областа на ASR е предмет на интензивен научен развој во последните 80 години (Rabiner and Schafer, 2011). Долгогодишната заложба на светската научна заедница, довела до бројни достигнувања во ова поле и покрај неговата комплексност. За таа цел искористени се познавања од бројни научни дисциплини, вклучувајќи ги: дигиталното процесирање на сигнали, статистичкото моделирање, машинското учење, процесирањето на природни јазици итн (Jurafsky and Martin, 2024). И покрај постигнатиот напредок, во полето на ASR остануваат цела низа на предизвици кои допрва треба да бидат надминати.

Напредокот во развојот на системите за автоматско препознавање на говор (ASR) доведе до нивна сè поголема експанзија во модерното живеење на човекот. Оваа своевидна револуција се должи на имплементирањето на системите за ASR во интелигентните мобилни уреди, како и на другите вгнездени компјутерски системи. Денес, не само што можете да свртите на некој свој познаник со помош на говорна наредба, туку можете да го прашате својот мобилен телефон за информации за времето, некој објект во близина, или пак да го искористите за пребарување на интернет сè со помош на вашиот глас. Освен во интелигентните мобилни уреди, вгнездените компјутерски системи со овозможен говорен пристап можат да се најдат и во мноштво современи уреди. Така, на автомобилот можете да му кажете да ги вклучи светлата или бришачите, како и да смени станица на радио; на кафе апаратот можете да му кажете какво кафе да ви спреми без да притискате копчиња; во вашиот интелигентен дом можете да ја поставите температурата на клима уредот, да ги затемните ролетните и да ја заклучите влезната врата, а на вашиот интелигентен звучник да му кажете која песна да ви ја пушти -- можностите се бескрајни. ASR е уште позначајна за корисниците со попреченост, кои можат да ја користат за комуникација со дигиталните уреди и услуги.

Архитектурата на еден генеричен систем за ASR е претставена на Сл. 8.6. Таа е составена од два основни модули: влезен и излезен. Задачата на влезниот модул е да го анализира говорниот аудиосигнал и од него да ги екстрагира **акустичките обележја** релевантни за неговата фонетската содржина. Задачата на излезниот модул е преку анализата на овие обележја, а со употреба на истренирани модели, да ја препознае фонетската содржина на говорниот сигнал, и да го даде на својот излез нејзиниот семантички идентитет. Излезниот модул вообичаено содржи два модели. **Акустичкиот модел** има задача да ја процени веројатноста на присуството на дадена буква во дадена секвенција на вектори на обележја. **Јазичниот модел** пак за дадена секвенција на буквите, треба да ја генерира веројатноста за низата зборови која одговара на неа. Проценетите веројатности од двата модела служат за пресметка на тоа која секвенција на зборови има најголема веројатност за влезната секвенција од обележја.

⁵⁷анг *Automatic Speech Recognition*.



Сл. 8.6: Архитектура на систем за автоматско препознавање на говорот

Мел фреквенциски кепстрални коефициенти

Уште од почетокот на развојот на системите за ASR, јасно било дека обележјата употребени за описување на акустичката реализација на гласовите се од фундаментално значење за нивните перформанси. Дизајнот на алгоритмите за екстракција на обележја вообичаено се темелел на моделирање на процесите во човековото сетило за слух. Така, речиси сите алгоритми за екстракција на обележја се базираат на моделирање на нелинеарноста на фреквенциската и амплитудната осетливост на човековото уво. Тоа се правело преку употреба на банка на филтри со нелинеарна распределба на централните фреквенции и фреквенцискот опсег на секој од филтрите, преку која има поголем број на филтри со помали опсези во ниските фреквенции, а помал број на филтри со поголеми опсези во високите фреквенции. Амплитудната нелинеарност на увото пак, се моделирала преку употреба на логаритамска компресија на амплитудата на сигналот.

Неприосновени во полето на ASR, од нивната прва примена во 1980, па сè до денес, се обележјата наречени **Мел фреквенциските кепстрални коефициенти (MFCC)**⁵⁸. Тие користат банка на триаголни филтри кои се линеарно распределени долж **мел**⁵⁹ фреквенциската скала. На Сл. 8.7 е прикажана мел банка составена од 10 филтри дизајнирана за покривање на опсег до 8 kHz. Во пракса, за ASR се користат банки составени од 20 до 40 филтри, додека во системите за TTS кои работат со длабоко учење се користат банки составени и од 80, па дури и 128 филтри.

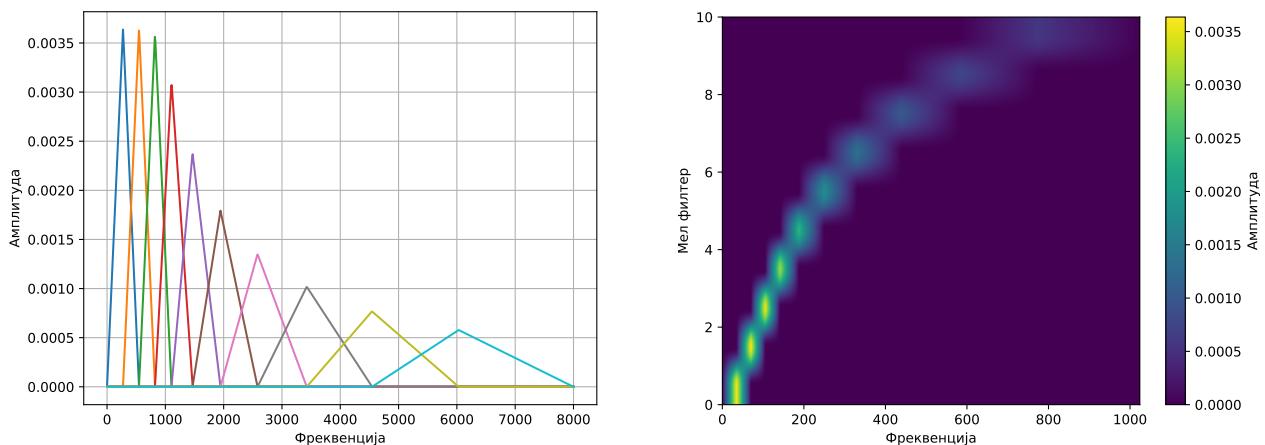
Освен мел банката на филтри, MFCC обележјата се потпираат на методата на **кепстрална анализа**⁶⁰ со чија помош може да се направи декомпозиција на спектарот на говорните сигнали на дел кој се должи на преносната функција на вокалниот тракт и дел кој одговара на неговата побуда.⁶¹ За одвојување на анвелопата на спектарот, кепстралната анализа ја пресметува инверзната Фурьеова трансформација на логаритамот од спектарот на сигналот. Логаритмирањето го претвора производот на фреквенциските карактеристики на побудата и вокалниот тракт, кој во фреквенциски домен соодветствува на конволуцијата на сигналот на побудата со филтерот на вокалниот тракт, во збир на овие две компоненти. Бидејќи побудата има повисока фреквенциска содржина од анвелопата, по инверзната Фурьеова трансформација, овие две компоненти можат лесно да се издвојат во кепстрален домен.

⁵⁸анг. *Mel Frequency Cepstral Coefficients*.

⁵⁹Името мел доаѓа од „мелодија“.

⁶⁰Зборот „кепстар“ се добива со превртување на зборот „спектар“ и го отсликува процесот на негово пресметување.

⁶¹Ова е слично на употребата на линеарната предикција за пресметување на параметрите на филтерот во моделот извор-филтер.



Сл. 8.7: Амплитудни карактеристики на филтрите на мел банка составена од 10 филтри, прикажани на фреквенциската оска.

На Сл. 8.8 е прикажана структурата на алгоритамот за пресметка на MFCC. Тој се состои од следните чекори:

- претпроцесирање на влезниот говорен сигнал, најчесто со **пренагласување⁶²** на високите фреквенции,
- земање рамки од аудиосигналот со **методата на прозорци**, најчесто со должина на рамката од 25 ms и Хамингов прозорец,
- пресметка на амплитудниот спектар на рамките на аудиосигналот со FFT,
- пресметка на моќноста на спектарот преку квадрирање на амплитудниот спектар,
- филтрирање на моќноста на спектарот со мел банката на филтри,
- логаритмирање на моќноста на спектарот, и
- **дискретна косинусна трансформација (DCT)⁶³** која е поефикасна, а за нашите сигнали еквивалентна на инверзната Фурьеовата трансформација.

На крајот, вообичаено се задржуваат само првите 13 коефициенти на DCT, кои соодветствуваат на анвелопата на спектарот на говорниот сигнал. Дополнително, MFCC коефициентите може да бидат филтрирани за да се засилат амплитудите на коефициентите со повисок ред,⁶⁴ како и да се нормализираат средната вредност и варијансата (CMVN)⁶⁵. Во системите за ASR, MFCC коефициентите вообичаено се надополнуваат со приближна пресметка на нивниот прв и втор извод за да се земе предвид временската динамика на говорниот сигнал.⁶⁶ Со тоа, вкупниот број на обележја, односно димензионалноста на влезниот вектор, за секоја рамка од говорниот аудиосигнал е 39.

Акустички модели

Обележјате кои ги генерираат алгоритмите во влезниот модул на системите за ASR претставуваат основа за моделирањето на акустичкиот отпечаток на различните гласови во говорниот аудиосигнал. Во развојот на системите за ASR се издвоиле четири етапи, односно парадигми, во начинот на работа на излезниот модул:

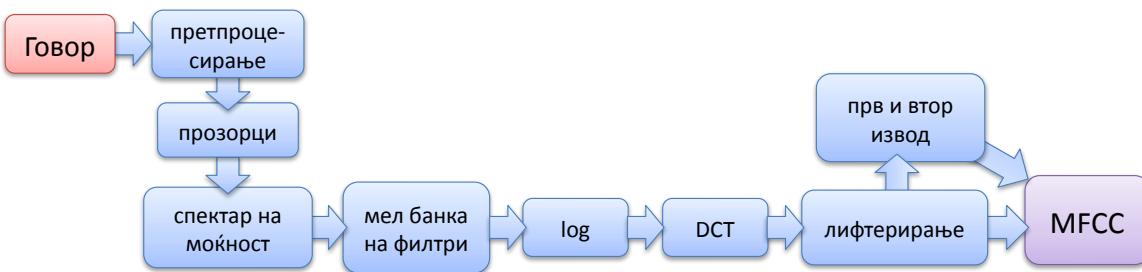
⁶²анг. *preemphasis*.

⁶³анг. *Discrete Cosine Transform*.

⁶⁴Филтрирањето во кепстрален домен се нарекува **лифтерирање**.

⁶⁵анг. *Cepstral Mean and Variance Normalization*.

⁶⁶Основните коефициенти се нарекуваат статички MFCC, а нивните прв и втор извод се нарекуваат динамички MFCC, или уште Δ -MFCC и $\Delta\Delta$ -MFCC.



Сл. 8.8: Блок шема на алгоритмот за пресметка на MFCC.

1. еквиваленција на акустичкиот облик на даден сигнал и неговата фонетска содржина,
2. споредба со множество урнеки,
3. статистичко моделирање на акустичката реализација на гласовите, и
4. длабоко учење.

Во првите системи за ASR постоела еден-спрема-еден кореспонденција меѓу дел од говорниот аудиосигнал и одреден глас, слог или најчесто збор. На пример, еден ваков систем ги користел траекториите на формантите во самогласките за препознавање на десетте цифри. Друг го мерел излезот од банка на 8 филтри во 5 временски точки и ги споредувал добиените резултати за препознавање на 10 слогови. Овој пристап иако покажал одредена употребливост во системите за еден говорник, не довел до добри резултати кога бил применет во системи независни од говорник поради големиот степен на варијабилност меѓу говорниците.

Понатамошниот развој довел до концептот на мерењето на растојание помеѓу две репрезентации на спектарот на говорниот аудиосигнал. Овие растојанија биле во сржта на системите за ASR базирани на споредба на урнеки (темплејти). За компензирање на варијацијата во временската динамика на изговарање на говорните секвенци, развиен бил алгоритамот базиран на динамичко програмирање познат како [динамичко временско преобличување \(DTW\)](#)⁶⁷.

Најголемата револуција во акустичките модели е добиена со преминот кон ригорозни, математички разработени рамки за статистичко моделирање, базирани на скриените модели на Марков (HMM). HMM моделите овозможуваат моделирање на лингвистичката структура на говорот преку синџири на состојби и статистички опис на варијабилноста на акустичките (спектрални) обележја преку дистрибуции на веројатност моделирани преку [Гаусови мешавински модели \(GMM\)](#). Како надградба на HMM, се воведени [тежинските трансдусери со конечен број на состојби \(WFST\)](#)⁶⁸ кои овозможуваат обединување на HMM акустичките модели со јазичните модели во една рамка за која постои ефикасен алгоритам за пребарување.

Невронските мрежи за првпат се јавуваат во системите за ASR како замена за GMM моделите во рамките на акустичките модели кои се употребуваат базирани на HMM. Во поново време, длабоките невронските мрежи речиси потполно ги имаат заменето другите типови на модели во системите за ASR, вклучувајќи ги и јазичните модели. Новите системи, како и кај TTS, се базираат на употреба на трансформери со милијарди параметри кои се тренираат на големи бази на говорен материјал. И тука, поради можноста за преносно учење, овие системи можат да се тренираат на огромни количини нетранскрибиран говорен материјал, а потоа да се нагодат на помала транскрибирана база. Како и во другите полиња, длабокото учење донесе надчовечки перформанси во препознавањето на говор.

Врвните системи за ASR денес постигнуваат рата на [грешка на ниво на збор \(WER\)](#)⁶⁹ од редот на 1,7% што е помалку од просечната човечка рата на грешка од околу 5%. Да не споменуваме дека, еден ист систем може да препознае говор на над 1000 јазици.

⁶⁷анг. *Dynamic Time Warping*.

⁶⁸анг. *Weighted Finite State Transducer*.

⁶⁹анг. *Word Error Rate*.

Препознавање на говор на македонскиот јазик

Системи за ASR се развиени за повеќето од светските јазици и денес преку нивната примена во мобилните интелигентни платформи, тие доживуваат голема експанзија и се повеќе се присутни во нашето секојдневие. Првиот систем за ASR наменет за македонски јазик развиен од Краљевски и др. во 2000 г. е базиран на употреба на хибриден систем, кој е комбинација на HMM и невронски мрежи. Овој систем е наменет за еден говорник и е развиен за препознавање на десетте цифри со точност од 85%. Со понатамошните подобрувања на овој систем, во 2002 Краљевски и др. реализираат систем независен од говорник, базиран на употребата на урнеци и динамичко временско преобличување DTW. Базата на податоци искористена за развој на овој систем се состоела од 8 говорници и имала целосна должина од 190 зборови. Резултатите од препознавањето се движат од 67,5% за говорници надвор од множеството за тренирање, до 90% за говорници во него.

Геразов и др. во 2013 го даваат својот придонес во ова поле со систем за препознавање на говор кој работи со среден вокабулар за контрола на паметни уреди со грешка на ниво на збор од 5,48%. Исто така, во 2016 предложен е и систем базиран на споредба со урнеци и DTW, наменет за работа на вгнездена платформа за примена во паметен дом.

Групата за говорни технологии на ФЕИТ (Говор@ФЕИТ) во 2020 постигнува врвни резултати во препознавањето на континуиран говор со голем вокабулар (LVCSR)⁷⁰ на македонски јазик. Системот направен за автоматска транскрипција на медиуми на македонски постигнува импресивна WER од 7,73% на множество за тестирање без зборови надвор од вокабуларот, дури и во присуство на шум. Резултатите се уште повеќе охрабрувачки ако се воочат местата на кои системот прави грешки.⁷¹ Во 2023, систем за ASR на македонски јазик развива спин-оф компанијата на ФЕИТ, Speech AI⁷², која го нуди како комерцијален производ.

ASR со Python

Во овој дел ќе реализираме еден едноставен систем за препознавање на говор со мало множество наредби наменет за контрола на електронски уреди. Базата на говорен материјал со којшто ќе работиме е дел од базата снимена од натпреварувачите на меѓународниот студентски натпревар RoboMac 2023 во категоријата ChatBot, каде задачата беше да се реализира систем за гласовна контрола на мобилен робот.⁷³ Делот кој го имаме на располагање е снимен од една говорничка и содржи по 100 изговори на секоја од 5те наредби: десно, лево, назад, оди, и стоп. Базата содржи една датотека со метаподатоци `metadata.csv`⁷⁴ во која се дадени патеките до секоја од снимките и наредбите изговорени во секоја од нив. Да ја испишеме нејзината содржина во терминал.

```
$ head asr/metadata.csv

filepath,label
audio/007_F_01_01.wav,Лево
audio/007_F_01_02.wav,Оди
audio/007_F_01_03.wav,Десно
audio/007_F_01_04.wav,Назад
audio/007_F_01_05.wav,Назад
audio/007_F_01_06.wav,Оди
audio/007_F_01_07.wav,Лево
audio/007_F_01_08.wav,Стоп
audio/007_F_01_09.wav,Десно
```

⁷⁰анг. *Large Vocabulary Continuous Speech Recognition*.

⁷¹<https://speech.feit.ukim.edu.mk/asr.html>

⁷²<https://speech-ai.mk/>

⁷³<http://robomac.mk/>

⁷⁴Форматот CSV (Comma Separated Values) е текстуален формат за запишување на табеларни податоци каде секоја колона е разделена со запирка. Имињата на колоните може да се дадени во првиот ред.

Уште еден сличен формат којшто често се користи е TSV (Tab Separated Values) каде секоја колона е разделена со табулатор.

Да напишеме код во Python кој ќе ги вчита податоците од базата. За вчитување на `metadata.csv` датотеката ќе го искористиме пакетот `pandas`, кој е специјализиран за работа со табеларни податоци. Од вчитаната табела ќе го екстрагираме множеството на класи и ќе направиме речник за кодирање на секоја од лабелите со one-hot вектор, кој ќе содржи 1 на позицијата на лабелата во низата на класи и 0 на другите позиции.

```
from os.path import join as pjoin

import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import librosa
from sklearn import neural_network, metrics

import da

# %% вчитување на податоците
data_path = "asr"
metadata_path = pjoin(data_path, "metadata.csv")
metadata = pd.read_csv(metadata_path)

fs = 16000 # Hz
n_mfcc = 13

# %% направи речник за претворање на лабелите во one-hot вектори
labels = metadata["label"]
classes = metadata["label"].unique()
n_class = len(labels)
label2onehot = {}
for i, cl in enumerate(classes):
    one_hot = np.zeros(n_class, dtype=int)
    one_hot[i] = 1
    label2onehot[cl] = one_hot
```

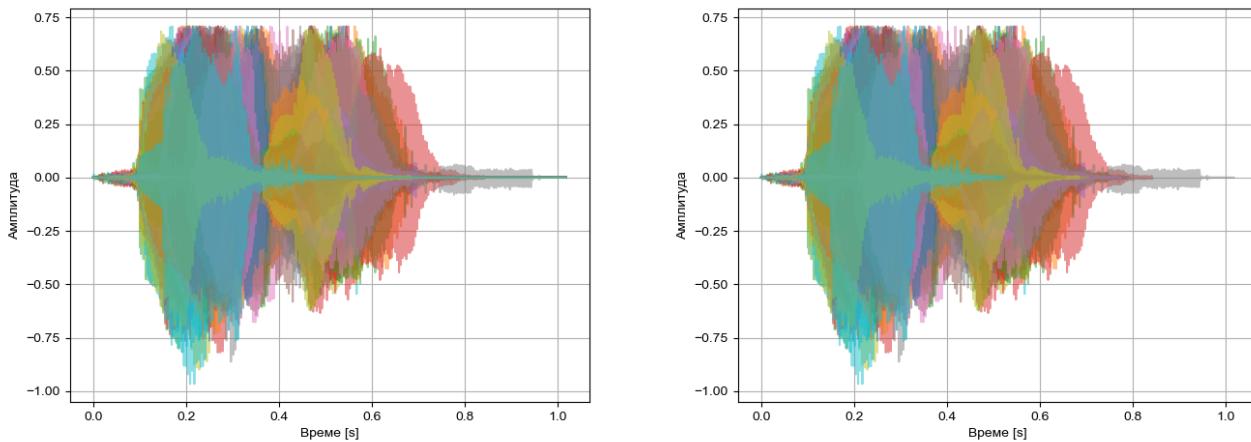
За вчитување на аудиосигналите, овојпат ќе го искористиме пакетот `librosa` кој ни овозможува да ја намалим фреквенцијата на земање примероци од 48 kHz на 16 kHz, што ќе ги поедностави пресметките и е стандардна вредност во системите за ASR. Секој од аудиосигналите ќе го нормализираме на ниво од -3 dB, а потоа ќе ги воедначиме нивните времетраења со додавање на бел Гаусов шум на крајот од секој од нив дополнувајќи ги до должината на најдолгиот аудиосигнал, која изнесува 1,018 s.

```
# %% вчитување на аудиосигналите
xs = []
wav_lens = []
for wav_path in metadata["filepath"]:
    x = librosa.load(pjoin(data_path, wav_path), sr=fs)[0]
    x = da.normalise(x, level=-3)
    xs.append(x)
    wav_len = x.size / fs
    wav_lens.append(wav_len)

max_len = max(wav_lens)

# %% воеднување на аудиосигналите
wavs = np.zeros((len(xs), int(max_len * fs)))
for i, x in enumerate(xs):
    pad_len = int(max_len * fs) - x.size
    pad = np.random.normal(0, 1e-4, pad_len)
    x_pad = np.concatenate((x, pad))
    wavs[i] = x_pad
```

Да ги прикажеме вчитаните и временски воедначени аудиосигнали.



Сл. 8.9: Првите 50 аудиосигнали од базата: без (лево) и со нормализација и временско воедначување (десно).

```
# %% 1. визуелизација на податоците
fig = plt.figure()
t = np.arange(wavs.shape[1]) / fs
plt.plot(t, wavs[:50].T, alpha=0.5)
plt.grid()
```

На Сл. 8.9 може да ги видиме првите 50 аудиосигнали од базата. За споредба на истата слика е прикажан нивниот изглед без нормализација и временско изедначување.

Ќе ги поделиме аудиосигналите и нивните лабели на множество за тренирање и тестирање, одвојувајќи 50% од нив за тестирање.

```
# %% 2. поделба на множества за тренирање и тестирање
test_coeff = 0.5 # 10%
test_ind = int(len(wavs) * test_coeff)
x_sigs = {}
x_sigs["train"] = wavs[:-test_ind]
x_sigs["test"] = wavs[-test_ind:]

y_labels = {}
y_labels["train"] = labels[:-test_ind]
y_labels["test"] = labels[-test_ind:]
```

Следно, ќе ја искористиме `librosa` за пресметка на MFCC коефициентите, кои ќе ги искористиме како влезни обележја за нашиот систем за препознавање на говор. Исто така, ќе ги кодираме лабелите со one-hot вектори.

```
# %% 3. екстракција на обележја
x_sets = {}
n_fft = int(2**np.ceil(np.log2(0.025 * fs)))
for set_name in ["train", "test"]:
    for i, x in enumerate(x_sigs[set_name]):
        mfccs = librosa.feature.mfcc(y=x, sr=fs, n_fft=n_fft, n_mfcc=n_mfcc)
        mfccs = mfccs.T # транспонирај за да има облик [време, коефициенти]
        mfccs = mfccs[np.newaxis, :, :] # add dimension for sample
        if i == 0:
            x_sets[set_name] = mfccs
        else:
            x_sets[set_name] = np.concatenate((x_sets[set_name], mfccs), axis=0)

# %% кодирај ги лабелите со one-hot вектори
y_sets = {}
for set_name in ["train", "test"]:
```

```
y = np.array([label2onehot[label] for label in y_labels[set_name]])
y_sets[set_name] = y
```

Може да видиме дека обликот на матриците со MFCC коефициенти [примероци, време, коефициенти] е [250, 32, 13] за двете множества, додека обликот на матриците со one-hot вектори [примероци, класи] е [250, 5] исто така за двете множества. Ајде да ги прикажеме пресметаните MFCC коефициенти за еден од примероците од нашата база.

```
# %% визуелизирај ги пресметаните обележја
i_sample = 13

fig = plt.figure()
plt.imshow(
    x_sets["train"][i_sample].T,
    aspect="auto",
    origin="lower",
    extent=[0, max_len, 0, n_mfcc],
)

# исцртај го временскиот облик и спектrogramот
fig = plt.figure()
t = np.arange(max_len * fs) / fs
plt.plot(t, x_sigs["train"][i_sample])
plt.grid()

da.get_spectrogram(x_sigs["train"][i_sample], fs, plot=True)
```

На Сл. 8.10 може да ги видиме временскиот облик, спектrogramот и пресметаните MFCC коефициенти за еден од примероците од нашата база со наредбата „десно“. Може да видиме дека за разлика од спектrogramот во кој можеме да ги воочиме составните гласови на наредбата „десно“, MFCC коефициентите не нудат интуитивна интерпретација. Сепак, тие се супериорни во однос на перформансите на системите за ASR. Исто така, може да видиме дека пониските MFCC коефициенти имаат изразено поголеми амплитуди од високите, па од таму и потребата за нивно лифтерирање, односно нормализација.

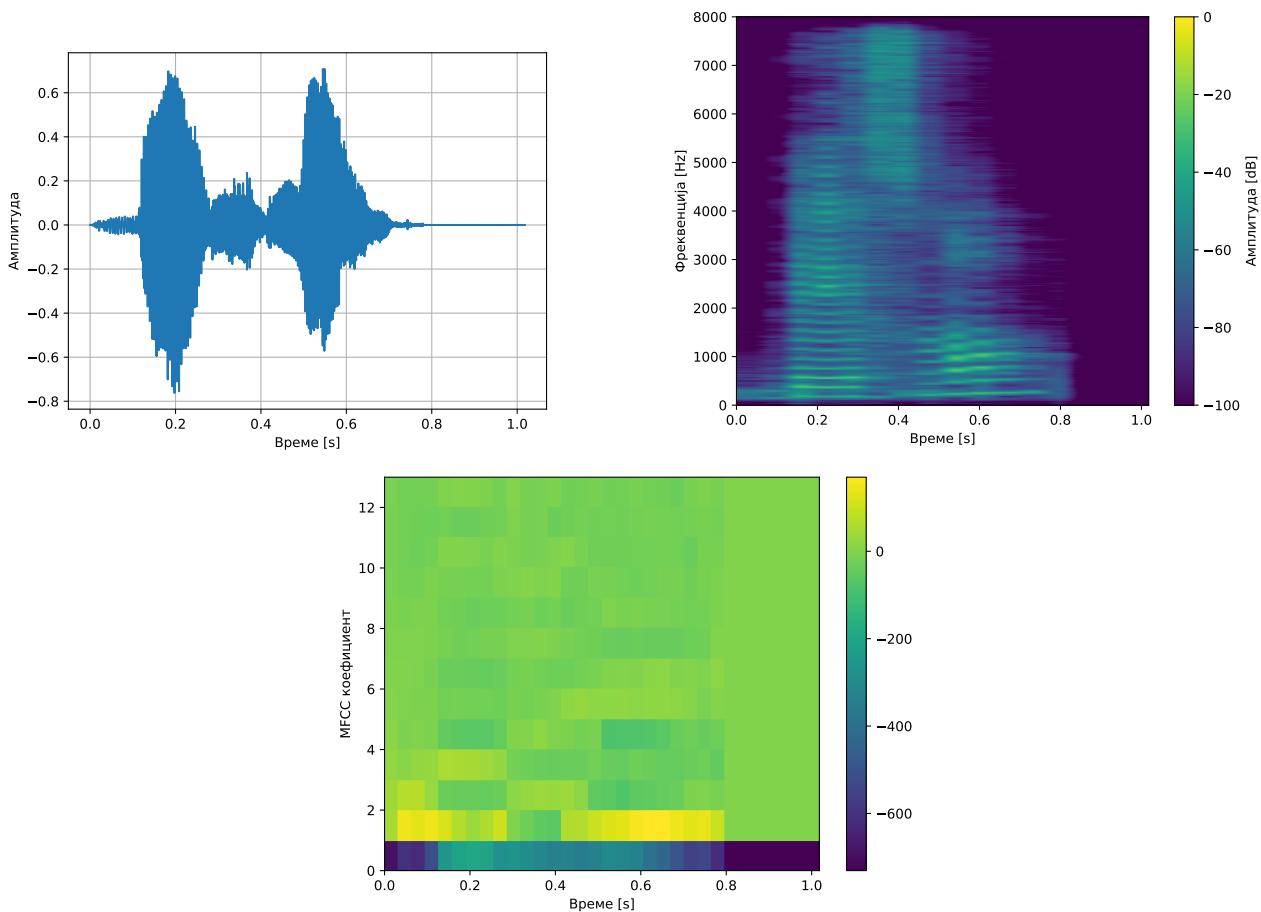
Сега можеме да ја истренираме нашата невронска мрежа. Ќе одиме со плитка невронска мрежа со еден скриен слој составен од 20 неврони. За да можеме да ги искористиме матриците MFCC коефициенти како влезни обележја, ќе ги израмниме⁷⁵ во 1D вектори. Со тоа нивниот облик ќе се промени од [примероци, време, коефициенти] во [примероци, време * коефициенти] каде коефициентите ќе бидат надоврзани ред по ред. Во нашиот случај, обликот на матриците ќе се промени од [250, 32, 13] во [250, 416]. Ова е честа практика и кај длабоките невронски мрежи, кај кои 2D репрезентациите добиени на излез од конволуциските слоеви треба да се трансформираат во 1D вектори пред да се проследат до целосно поврзаните слоеви на излез од моделот.

```
# %% израмни ги матриците во вектори
x_sets["train"] = x_sets["train"].reshape(x_sets["train"].shape[0], -1)
x_sets["test"] = x_sets["test"].reshape(x_sets["test"].shape[0], -1)

print(x_sets["train"].shape)
print(x_sets["test"].shape)

# %% 4. тренирање на моделот
np.random.seed(42)
mlp = neural_network.MLPClassifier(
    hidden_layer_sizes=(20),
    activation="relu",
    learning_rate_init=0.001,
    alpha=1e-4,
    max_iter=500,
    tol=1e-9,
```

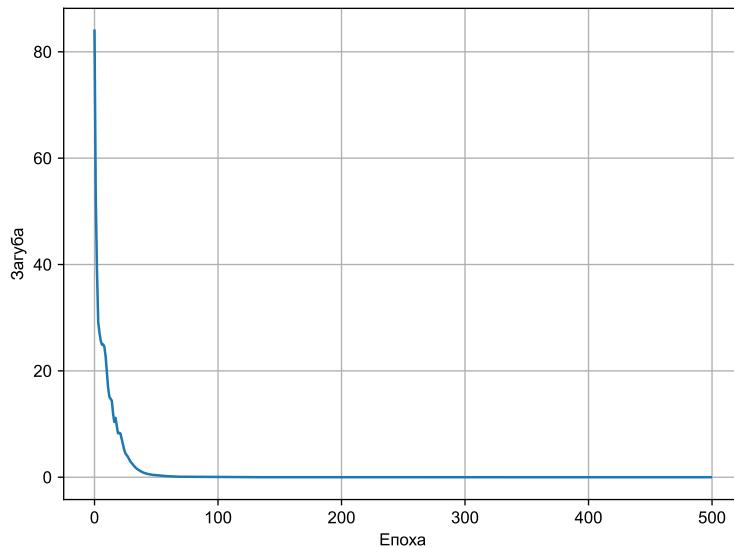
⁷⁵анг. *flatten*.



Сл. 8.10: Временски облик (горе лево), спектрограм (горе десно) и MFCC коефициенти (долу) за примерок од базата со наредбата „десно“.

```
n_iter_no_change=40,
shuffle=True,
random_state=42,
verbose=1,
)
mlp.fit(x_sets["train"], y_sets["train"])
```

```
Iteration 1, loss = 83.97353525
Iteration 2, loss = 52.73790934
Iteration 3, loss = 38.84197771
Iteration 4, loss = 29.14657715
Iteration 5, loss = 27.22475104
Iteration 6, loss = 25.72544233
Iteration 7, loss = 24.92685570
Iteration 8, loss = 24.99277912
Iteration 9, loss = 24.48930453
Iteration 10, loss = 22.70487710
...
Iteration 30, loss = 2.97454337
...
Iteration 50, loss = 0.37985605
...
Iteration 495, loss = 0.00032574
Iteration 496, loss = 0.00032497
Iteration 497, loss = 0.00032364
Iteration 498, loss = 0.00032256
Iteration 499, loss = 0.00032168
Iteration 500, loss = 0.00032044
```



Сл. 8.11: Крива на загуба при тренирање на моделот.

Може да видиме дека загубата на моделот во текот на 500те епохи на тренирање монотоно опаѓа за неколку реда на големина, почнувајќи од 83,97, па сè до 0,0003, што значи дека моделот ефикасно учи од податоците. Да ја исцртаме кривата на загуба прикажана на Сл. 8.11.

```
# исцртaj ја кривата на загуба
loss_curve = np.array(mlp.loss_curve_)
plt.figure()
plt.plot(loss_curve)
plt.grid()
```

На крај, останува да го евалуираме моделот. Ајде да ја пресметаме точноста и да ја прикажеме матрицата на забуна за моделот. За исцртување на матрицата на забуна, овојпат ќе го искористиме seaborn пакетот, кој нуди напредни функции за визуелизација на податоци.

```
# %% 5. евалуација на моделот
# точност
accuracy = mlp.score(x_sets["test"], y_sets["test"])

# матрица на забуна
y_pred = mlp.predict(x_sets["test"])
y_pred_prob = mlp.predict_proba(x_sets["test"])
cm = metrics.confusion_matrix(
    np.argmax(y_sets["test"], axis=1),
    np.argmax(y_pred_prob, axis=1),
)
cm_norm = cm / cm.sum(axis=1)

ys = classes
sns.set(font_scale=1.0)
fig, ax = plt.subplots()
sns.heatmap(
    cm,
    annot=True,
    cbar=False,
    fmt="d",
    cmap="Blues",
    xticklabels=ys,
    yticklabels=ys,
)
```

Може да видиме дека точноста изнесува 88,8% на множеството за тестирање. Матрицата на

		Десно	1	0	0	2
		Лево	50	0	0	0
Точна класа	Назад	0	0	49	1	0
	Оди	0	0	0	49	1
Стоп	Десно	0	0	0	0	50
	Лево		Назад	Оди	Стоп	Предвидена класа

Сл. 8.12: Матрица на забуна за моделот.

забуна е прикажана на Сл. 8.12. Може да видиме дека моделот одлично работи и прави само 5 грешки од 250 примероци што претставува точност од 98%!

§ Дополнително. Но, зошто точноста која ја пресметавме на моделот од 88,8% не соодветствува на точноста прикажана во матрицата на забуна од 98%?

Ако ја разгледаме подетално матрицата на предвидувања `y_pred` може да видиме дека во неа немаат сите редици по една 1, туку некои од нив немаат ниту една 1, а некои имаат и по две 1. Ова е поради начинот на кој работи функцијата `predict` на моделот `mlp`, која враќа 1 за излезот на секој неврон кој е поголем од 0.5. Од друга страна, за пресметување на матрицата на забуна ја бараме најверојатната класа за секој примерок од множеството за тестирање, со употреба на `np.argmax(predict_proba, axis=1)`. Така, во секоја редица ќе имаме само една 1 која ќе соодветствува на класата чиј неврон генерирал најголем излез.

```
In [8]: np.sum(y_pred, axis=1)
Out[8]:
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 2, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
In [9]: y_pred[10]
Out[9]: array([0, 0, 0, 0, 0])
```

```
In [10]: y_pred_prob[10]
Out[10]:
array([1.97328282e-22, 9.65655737e-10, 5.69465331e-04, 1.95557109e-32,
       4.00264238e-08])
```

```
In [11]: y_pred[18]
```

```
Out[11]: array([0, 1, 0, 0, 1])  
In [12]: y_pred_prob[18]  
Out[12]:  
array([1.96267936e-37, 9.99757721e-01, 1.76626902e-11, 6.64585545e-21,  
      1.0000000e+00])
```

✓ **Задача за дома.** Експериментирајте со подесување на хиперпараметрите на невронската мрежа за да добиете уште подобри резултати со тренираната мрежа. Внимавајте во тој случај да издвоите од множеството за тестирање еден дел за валидација кој ќе го искористите за подесување на хиперпараметрите. На крајот, ќе го искористите множеството за тестирање само за евалуација на моделот со најдобри перформанси! Во спротивно ако ги оптимизирате хиперпараметрите директно врз множеството за тестирање, може да добиете нереално високи перформанси на моделот.

Додаток А

Слободен и отворен софтвер за инженерска и научна работа

Еден од најпрочуените софтверски пакети за нумеричка обработка е програмскиот пакет MATLAB¹. MATLAB, преку својата синтакса на високо ниво дозволува: лесна манипулација на матрици, исцртување на функции и податоци, имплементација на алгоритми, создавање на кориснички интерфејси, итн. Тој може да се употреби во најразлични области од инженерската практика, меѓу кои и во дигиталната обработка на звук, слика и видео. За првпат бил изданен во 1984 г., а во 2004 г. имал 1 милион корисници инженери, научници и економисти.

Сепак MATLAB, како комерцијален софтвер носи и низа од недостатоци, пред сè високата цена која го става вон дофат на студентите, истражувачите, малите компании, како и на научно истражувачките и образовните установи во поголем дел од светот. Други недостатоци на MATLAB се ограничената преносливост на кодот, како и неговата затвореност.

A.1 Слободен софтвер

Денес сè повеќе инженери и научници ја напуштаат употребата на комерцијалниот затворен софтвер и својата работа ја засноваат на платформи базирани на слободен софтвер². Ова пред сè се должи на философијата на движењето за слободен софтвер започнато од Ричард Сталман³ во 1983 г. со создавањето на GNU оперативниот систем, а подоцна со воспоставување на Фондацијата за слободен софтвер⁴ во 1985 г., како и поширокото движење за отвореност⁵, а тоа е заедништво во создавањето и напредувањето на технологијата и човештвото.

A.2 Четири слободи

Слободниот софтвер е дефиниран со четирите слободи:⁶

- Слобода 0. Слобода да ја користите програмата за било која намена.

¹MATLAB®Matrix Laboratory, The MathWorks, Inc., Natick, Massachusetts, United States. <http://www.mathworks.com/products/matlab/>

²Wikipedia -- Free software movement https://en.wikipedia.org/wiki/Free_software_movement

³Wikipedia -- Richard Stallman https://en.wikipedia.org/wiki/Richard_Stallman

Предавање на Ричард Сталман за философијата на движењето за слободен софтвер -- Richard Stallman -- Free software, free society, TEDxGeneva 2014 https://www.youtube.com/watch?v=Ag1AKI1_2GM

⁴Wikipedia -- Free Software Foundation https://en.wikipedia.org/wiki/Free_Software_Foundation

⁵Wikipedia -- Open-source model https://en.wikipedia.org/wiki/Open-source_model Nathan Seidle -- How Open Hardware will Take Over the World, TEDxBoulder https://www.youtube.com/watch?v=xGhj_1LNtd0

⁶Превземено од вебстраницата на организацијата Слободен софтвер Македонија <https://slobodensoftver.org.mk/shto>

Добавањето рестрикции за користење на слободен софтвер, како што се временските рестрикции („Пробен период од 30 дена“, „Лиценцата истекува на 1 јануари 2005“), рестрикции на целта („Дозволена е употреба за истражувачки и некомерцијални цели“) или рестрикции на географската област („Мора да се користи во земјата А“), ја прават програмата неслободна.

- **Слобода 1.** Слобода да проучите како работи програмата и како истата да ја адаптирате на сопствените потреби.

Добавањето легални или практични рестрикции на разбирањето или менувањето на програмата, како што се задолжително купување на специјални лиценци, потпишување на спогодба за неоткривање (Non-Disclosure-Agreement) или правењето изворниот код да биде недостапен, исто така ја прават програмата неслободна. Без слободата да се менува програмата, луѓето ќе останат на милост на единствен снабдувач.

- **Слобода 2.** Слобода да редистрибуирате копии за да му помогнете на вашиот сосед.

Софтверот може да се копира/дистрибуира скоро без никакви трошоци. Ако не смеете да му дадете некоја програма на некој човек кому таа му треба, тоа ја прави програмата неслободна. Се разбира, доколку сакате, за ваквите активности можете да наплатите.

- **Слобода 3** Слобода да ја подобрувате програмата и да ги издадете вашите подобрувања во јавноста, од што корист ќе има целата заедница.

Сите луѓе не се подеднакво добри програмери. Некои луѓе пак воопшто не знаат да програмираат. Оваа слобода им дозволува на оние луѓе кои немаат време или знаење да решат некој проблем индиректно да пристапат до слободата за менување на програмата. Се разбира, доколку сакате, за ваквите активности можете да наплатите.

Доколку софтверот не ги исполнува сите горни услови, тогаш тој не е слободен софтвер.

A.3 Предности на слободниот софтвер

Од практичен аспект, отворениот софтвер има низа предности над затворениот софтвер и тоа:

- **достапноста** -- поради основната премиса на давање на изворниот код, со цел да се овозможи неговиот развој од заедницата, отворениот софтвер е *de facto* и бесплатен софтвер. Така, повеќето производители на слободниот софтвер живеат од донацији, но и од продавање поддршка за нивниот производ.
- **безбедноста** -- поради достапноста на изворниот код, не постои начин производителот на софтверот да прави нешто скриено од вас, а секој спорен дел од кодот е подложен на промена од заедницата. Кај затворениот софтвер тоа не е случај.^{7,8}
- **слободата од производителот** -- како корисници на отворениот софтвер, вие не сте затворени во екосистемот на производителот.⁹ Истиот тој софтвер може да биде превземен од друга заедница на програмери и да продолжи неговото одржување и развој во друга насока.

⁷ Во Windows 10 производителот го задржува правото да ги чува вашите приватни податоци како што вели во изјавата за приватност: ``Finally, we will access, disclose and preserve personal data, including your content (such as the content of your emails, other private communications or files in private folders), when we have a good faith belief that doing so is necessary ...''

Истите механизми се додадени во претходните верзии на Windows преку автоматските надградби.

Zach Epstein, Windows 10 is spying on almost everything you do – here's how to opt out, Jul 31, 2015, <http://bgr.com/2015/07/31/windows-10-upgrade-spying-how-to-opt-out/>

Ashley Allen, How to Stop Windows 7 and 8 From Spying on You <http://www.etechnix.com/stop-windows-7-8-spying/>

⁸ Епл и Самсунг ги забавија телефоните на корисниците преку нивното редовно ажурирање <https://www.cnet.com/news/apple-and-samsung-fined-for-slowing-down-phones-with-updates/>

⁹ Don Reisinger -- Steve Jobs wanted to 'further lock customers' into Apple's 'ecosystem' <https://www.cnet.com/news/steve-jobs-wanted-to-further-lock-customers-into-apples-ecosystem/>

- подобар квалитет -- при воспоставување на критична големина на заедницата околу еден отворен софтвер, развојот не може да се спореди со ресурсите кои ги поседува било која корпорација во светот. Така, развојот на Linux јадрото¹⁰, кое е во основата оперативниот систем GNU/Linux познат и само како Linux¹¹ и повеќе од 600-те GNU/Linux дистрибуции¹², првично напишано од Линус Торвалдс¹³, денес претставува најголемиот здружен проект во историјата на човештвото со околу 6000 активни развивачи, над 20 милиони редови на код, и со проценета развојна вредност од над 2 милијарди евра.¹⁴

Сите овие придобивки заедно придонесуваат за широка распространетост на слободниот софтвер денес.¹⁵ Така, GNU/Linux и FreeBSD¹⁶ оперативните системи се во употреба во 98,27% од серверите на интернет (споредено со 1,73% со Windows), 71,9% од паметните телефони (Android оперативниот систем), и 99% од суперкомпјутерите¹⁷. Сепак, неговиот пробив во персоналните компјутери засега е мал -- 2,5% (наспроти 75,5% на Windows и 14,9% на MacOS).

A.4 Одржливост

Постојат различни начини на кои се реализира финансиската поддршка на слободниот софтвер и покрај бесплатноста и тоа:

- финансиска поддршка од компанији -- зад многу пакети слободен софтвер стојат компании од чиј интерес е неговиот развојот. Најдобар пример за тоа е можеби самото Linux јадро на кое работат инженери од многу компании од целиот свет, а најголемиот придонес го има компанијата Intel. Тука се и низа на GNU/Linux дистрибуции меѓу кои Ubuntu¹⁸, Fedora¹⁹, и OpenSUSE²⁰, како и пакетите за длабоко учење TensorFlow²¹ и PyTorch²², исто така развиваани од компанији,
- финансиска поддршка од јавно финансирање и грантови -- голем број на слободни софтвери се плод на работата на инженери и научници финансиирани од државите низ светот или од приватни фондации. Таков е на пример KiCad софтверот за електронски дизајн и изработка на печатени плочи развиен во CERN²³, софтверот за дигитална обработка на звук Audacity започнат во Универзитет Карнеги Мелон²⁴, или пак пакетот за машинско учење Сајкитлрн започнат во Инриа²⁵,
- бизнис модел базиран на поддршка -- најголемата компанија која денес работи исклучиво со слободен софтвер е Red Hat чиј GNU/Linux оперативен систем е еден од најзастапените на интернет серверите.²⁶ Red Hat заработува преку продажба на поддршка за овој оперативен систем и во моментов е проценета на вредност од 38 милијарди USD,
- финансиска поддршка од донацији -- многу слободни софтвери егзистираат благодарејќи на донацији направени од нивните корисници. Тука спаѓаат најголем број од GNU/Linux

¹⁰Wikipedia -- Linux kernel https://en.wikipedia.org/wiki/Linux_kernel

¹¹Wikipedia -- Linux <https://en.wikipedia.org/wiki/Linux>

¹²Wikipedia -- List of Linux distributions https://en.wikipedia.org/wiki/List_of_Linux_distributions

¹³Wikipedia -- Linus Torvalds https://en.wikipedia.org/wiki/Linus_Torvalds

¹⁴Добар документарен филм за рафањето и развојот на GNU/Linux оперативниот систем е Revolution OS - 2001 <https://www.youtube.com/watch?v=Eluzi700-P4>

¹⁵Wikipedia: Linux - Market share and uptake https://en.wikipedia.org/wiki/Linux#Market_share_and_uptake

¹⁶<https://www.freebsd.org/>

¹⁷Linux is Running on Almost All of the Top 500 Supercomputers <https://itsfoss.com/linux-supercomputers-2017/>

¹⁸<https://www.ubuntu.com/>

¹⁹<https://getfedora.org/>

²⁰<https://www.opensuse.org/>

²¹TensorFlow -- An end-to-end open source machine learning platform <https://www.tensorflow.org/>

²²PyTorch -- from research to production <https://pytorch.org/>

²³KiCad EDA -- A Cross Platform and Open Source Electronics Design Automation Suite <https://www.kicad.org/>

²⁴Audacity -- Free, open source, cross-platform audio software <https://www.audacityteam.org/>

²⁵scikit-learn -- Machine Learning in Python <https://scikit-learn.org/stable/index.html>

²⁶Red Hat -- The world's leading provider of open source solutions <https://www.redhat.com>

дистрибуциите како на пример Манџаро²⁷ или Минт²⁸, а исто така Spyder²⁹ развојната средина за Python која ќе ја користиме во предметов,

- ентузијазам -- мотивот нешто да се создаде или подобри и да се сподели со целиот свет понекогаш е доволен мотив за развој на слободниот софтвер. Постојат низа пакети со заедници на развивачи кои немаат финансиски придобивки од нивната работа на проектот, но сепак продолжуваат да работат на него водени од сопствените убедувања и стремеж кон повисоки вредности.

A.5 Слободен софтвер за инженерска и научна работа

Постојат низа на слободни софтвери кои можат да бидат искористени за обработка на нумерички податоци.

- GNU Octave³⁰ има синтакса направена да биде во голема мера компатибилна со онаа на MATLAB. Во Octave се реализирани голем број на пакети кои можат да се искористат за обработка на најразлични типови на сигнали. Проблемот со Octave е во неговата мала брзина на извршување, поради што највеќе се употребува во образоването како замена за MATLAB.
- Scilab³¹ е слободен софтвер за нумеричка обработка наменет за инженери и научници, во употреба од 1994 г. Scilab во себе вклучува и слободна замена за Симулинк пакетот на MATLAB, наречена Икскос³².
- Python³³ е широко распространет, повеќенаменски, интерпретиран и динамичен програмски јазик на високо ниво направен од Гуидо ван Росум³⁴ во 1989 г. Иако не е наменет строго за нумеричка анализа, елегантната и едноставна синтакса која овозможува лесна читливост, како и неговата широка распространетост во најразлични области, го прават Python идеална основа за слободната работа и соработка на научната и образовната заедница ширум светот.
- Julia³⁵ е јазик за нумеричко процесирање со компајлирање направен на MIT, кој иако има синтакса на високо ниво како онаа на MATLAB, работи речиси еднакво брзо со код напишан во С. И покрај големиот потенцијал на Џулија, за сега неговата примена останува ограничена во области во кои е неопходна голема процесирачка моќ.

Audacity

Audacity³⁶ е уште еден слободен софтвер кој ќе ни биде од корист во анализата, едитирањето и снимањето на дигитално аудио, прикажан на Сл. А.1. Неговиот развој го започнале Доминик Мацони и Роџер Даненберг во 1999 во Универзитетот Карнеги Мелон и е иницијално објавен во 2000 како верзија 0.8. Од 2011, тој е 11-от најсимнуван софтвер на Сурсфорџ, со 76,5 милиони симнувања. Audacity е добитник на наградата за најдобар проект за мултимедија од заедницата SourceForge во 2007 и 2009. Во 2015 е преместен на FossHub каде е најпопуларниот софтвер со над 114 милиони симнувања.³⁷

²⁷Manjaro -- Professional Linux at its best <https://manjaro.org/>

²⁸Linux Mint -- From freedom came elegance <https://linuxmint.com/>

²⁹Spyder -- The Scientific Python Development Environment <https://manjaro.org/>

³⁰GNU Octave -- Scientific Programming Language <https://www.gnu.org/software/octave/>

³¹Scilab -- Open source software for numerical computation <https://www.scilab.org/>

³²Xcos <https://www.scilab.org/software/xcos>

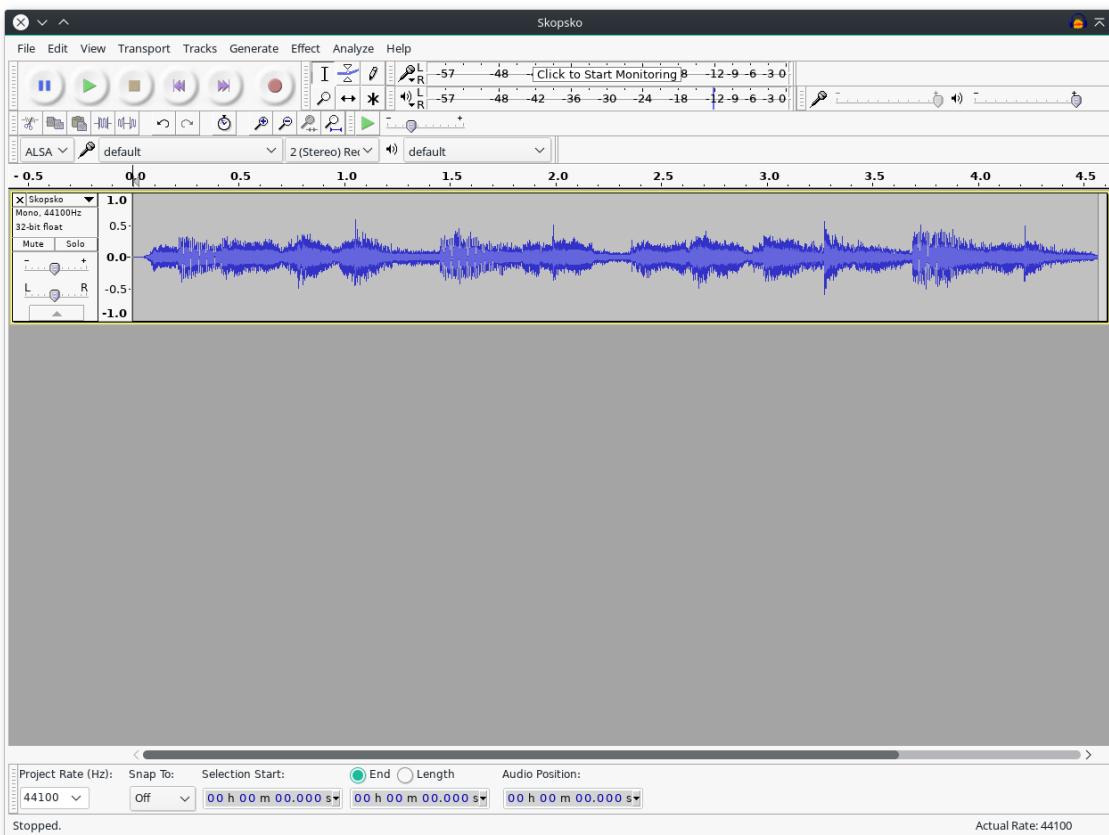
³³Python <https://www.python.org/>

³⁴Wikipedia -- Guido van Rossum https://en.wikipedia.org/wiki/Guido_van_Rossum

³⁵The Julia Programming Language <https://julialang.org/>

³⁶Audacity. <http://www.audacityteam.org/>

³⁷Wikipedia: Audacity (audio editor). [https://en.wikipedia.org/wiki/Audacity_\(audio_editor\)](https://en.wikipedia.org/wiki/Audacity_(audio_editor))



Сл. А.1: Отворен аудио файл во главниот прозорец на Audacity.

Освен тоа што поддржува снимање од повеќе извори, Audacity може да се искористи за процесирање на сите типови на аудио, преку додавање на ефекти како нормализација, поткастрување, и прелевање. Тој може да се користи за снимање и миксање на цели албуми, како што е случајот со групата Tune-Yards. Тој е во употреба и во националниот курс за ICT ниво 2 на OCR³⁸ во Велика Британија. Главните особини на Audacity вклучуваат:

- Читување и снимање на различни типови на аудио формати, како WAV, AIFF, MP3, Ogg Vorbis, FLAC, WMA, AAC, AMR и AC3.
- Снимање и репродукција на звук.
- Едитирање со неограничен број на undo.
- Автоматска поделба на аудио траки на дигитализирани снимки од касети или грамофонски плочи.
- Повеќеканално миксање.
- Голем број на аудио ефекти и плагини. Додатни ефекти можат да се напишат во Nyquist кој е дијалект на Lisp, а поддржани се плагини направени во отворениот LV2 стандард, како и VST плагини.
- Едитирање на амплитудната анвелопа.
- Намалување на шумот.
- Намалување на вокалите.
- Спектрална анализа со употреба на FFT.

³⁸Oxford, Cambridge and RSA Examinations

130 ДОДАТОК А. СЛОБОДЕН И ОТВОРЕН СОФТВЕР ЗА ИНЖЕНЕРСКА И НАУЧНА РАБОТА

- Поддршка на повеќеканално дигитално аудио со фреквенција на земање примероци до 96 kHz и резолуција до 32 bit.
- Прецизно нагодување на брзината на аудиото без промена во фреквенцијата на звукот.
- Нагодување на висината на тонот без промена на брзината.
- Можности за модерно повеќеканално едитирање.
- Работа на повеќе платформи.
- Приказ на ефектите базирани на LADSPA, VST (32-bit) и Audio Unit (OS X) во реално време.
- Зачувување и вчитување на кориснички предпоставувања.

Тој исто така работи на сите оперативни системи.

Додаток Б

Основи на користење на GNU/Linux

Б.1 Инсталирање на GNU/Linux

Иако користењето на Python не е врзано со GNU/Linux оперативниот систем, вежбите во овој предмет ќе се базираат на работа во GNU/Linux.

Зошто да инсталираам GNU/Linux?

Linux има низа предности кои ги споменавме во Додатокот А меѓу кои:

- безбедност -- тој е побезбеден од другите оперативни системи поради неговите вградени безбедносни карактеристики и затоа што е помалку подложен на штетен софтвер и вируси,
- стабилност -- Linux е познат по својата стабилност и доверливост, што е една од главните причини што го прави популарен избор за сервери и други критични системи,
- слободен софтвер -- нема лиценци кои треба да се платат.

Уште повеќе, користењето на GNU/Linux ќе ве направи подобри идни инженери:

- употреба во компании и институции -- Linux има широка употреба во технолошките компании и во научно-истражувачките установи. Со тоа, искуството што ќе го стекнете со негова употреба ќе ви овозможи едноставно да се вклопите во екосистемот во кој ќе се најдете во вашата инженерска кариера,
- пристап до алатки за развој -- многу алатки за развој на софтвер се достапни некогаш и исклучиво за Linux. Често се случува, дури и ако постои напатство за инсталирање на некои пакети на Windows, тоа да биде изразено постара верзија од пакетот, или пак да има многу ограничена поддршка, поради тоа што развиваците користат Linux,
- контрола и приспособување -- кај Linux корисникот има целосна контрола врз оперативниот систем и хардверот на кој тој е инсталiran преку едитирање на текстуални датотеки. Дополнително, Linux овозможува висок степен на прилагодување, како на изгледот на десктоп средината, така и на начинот на функционирање на оперативниот систем. Овој степен на контрола ви ги одврзува рацете и ви нуди можност за зголемување на вашите познавања за тоа како работи вашиот хардвер и цел еден оперативен систем, вклучувајќи ги и апликациите,
- командна линија -- Linux често се користи преку командната линија, што ќе ви помогне да ги развиете своите вештини за нејзина употреба. Тоа ќе ви ја олесни работата со системите кои можат да се пристапат преку командна линија, како на пример работа со интернет сервери,

- сеприсутност на различни платформи -- освен десктоп компјутерите, речиси сè друго работи на Linux -- од вгнездените платформи како Raspberry Pi, преку сите сервери на интернет, па сè до најголемите светски суперкомпјутери. Со тоа искуството што ќе го стекнете со работа во Linux на вашиот персонален компјутер е директно применливо на цела низа платформи, од кои со дел сигурно ќе се сртнете во вашата професионална кариера. Всушност вашиот Linux лаптоп е де факто сервер на кој можат да се приклучите од друг компјутер, или пак да стартувате веб апликација до која можат да пристапат корисници преку интернет. Конечно, софтверот кој сте го напишале и сте го тестирале на вашиот лаптоп, ќе работи и на вгнездена платформа (во рамки на можностите) и на интернет сервер, без потреба од големи модификации,
- слободен софтвер -- работата со слободен софтвер ќе ви овозможи да станете дел од заедницата која го развива. Со тоа не само што ќе се стекнете со увид во начинот на развој на различни софтвери, туку ќе можете и самите да придонесете во нивниот развој, унапредувајќи ги вашите вештини да работите развој во рамките на тим.

Ова е причината зошто и покрај малата застапеност на GNU/Linux на персоналните компјутери од 2,5% кај општата популација, кај инженерите и програмерите оваа бројка изнесува 40% како за професионална така и за лична употреба, споредено со 49% за употреба на Windows за професионална употреба.¹ Ако земеме в'обзир дека 15% од корисниците на Windows го користат емулататорот за Linux WSL, тогаш излегува дека мнозинството (55%) користат Linux за развој на софтвер.

Начин на инсталирање

Доколку веќе немате GNU/Linux, истиот се препорачува да го инсталирате паралелно на постоечкиот оперативен систем во dual boot режим. Во најмала рака може да инсталирате GNU/Linux во виртуелна машина, но ова може да ги ограничи постоечките ресурси за процесирање на сигналите.

Избор на дистрибуција

Иако постојат повеќе од 600 GNU/Linux дистрибуции², вистинскиот избор за инсталирање се сведува на дузина од најпопуларните дистрибуции, затоа што популарноста е некој вид на гарант за нивната стабилност како и леснотијата на наоѓање на поддршка. Добар преглед на популарноста на различните Linux дистрибуции, како и повеќе информации за истите може да најдете на вебстраницата *Distrowatch*³. Во моментов таму топ 5 се: MX Linux, EndeavourOS, Mint, Manjaro и Pop!_OS. Вреди да ги споменеме и дистрибуциите: Ubuntu, Fedora, openSUSE, и Arch.

Едни од главните разлики помеѓу дистрибуциите се:

- инсталирањето на пакети - различните дистрибуции имаат различни менаџери на пакети кои служат за инсталирање и ажурирање на софтвер;
- режимот на ажурирање на оперативниот систем -- некои дистрибуции имаат периодични помали надградби и поголеми надградби на подолг рок, така Ubuntu има нова верзија со долготрајна поддршка (LTS) на секои две години; некои дистрибуции во периодичните надградби го ажурираат системот на најновите верзии на сите софтвери (rolling release) со што се заобиколува потребата од поголеми надградби, на пример Arch има нова надградба секој месец и нема потреба од други надградби.

Овие карактеристики се заеднички за цели фамилии на дистрибуции, на пример Kubuntu и Ubuntu користат `apt` за инсталирање на `.deb` пакети како и Debian -- оригиналната дистрибуција

¹Stack Overflow Developer Survey 2022: Operating system <https://survey.stackoverflow.co/2022/#section-most-popular-technologies-operating-system>

²Wikipedia -- List of Linux distributions https://en.wikipedia.org/wiki/List_of_Linux_distributions

³<http://distrowatch.com>

на која се базирани. Дополнително, сите тие се базираат на поголеми надградби на подолг рок.

Избор на десктоп средина

Важно е да се направи разлика помеѓу GNU/Linux дистрибуцијата и изгледот на нејзиниот кориснички интерфејс. Интерфејсот на дистрибуцијата се базира на **десктоп средината** која ја користи. Кaj GNU/Linux оперативните системи постојат различни десктоп средини како GNOME и KDE Plasma, Xfce, LXQt, Cinnamon, Mate, i3wm, Pantheon, итн. Дистрибуциите вообичаено вклучуваат и одредено ниво на прилагодба на стандардниот изглед на десктоп средината. На пример, Ubuntu додава панел со икони за стартување и контрола на програмите од левата страна во GNOME, додека Manjaro додава докер со икони на долнот раб од десктопот.

GNOME⁴ е модерна десктоп средина со елегантен интерфејс која се фокусира на едноставност и функционалност. Со самото тоа, таа не нуди голема конфигурабилност. Од друга страна, KDE Plasma⁵ важи за најмоќна и најконфигурабилна десктоп средина за Linux системите. Постојат и поедноставни средини кои се наменети за помала потрошувачка на ресурси како Xfce, LXQt итн.

Некои препораки

Во Лабораторијата за дигитално процесирање на сигнали ќе работиме со Ubuntu⁶ кој ја користи десктоп средината GNOME. Ubuntu е една од најпопуларните дистрибуции и вообичаено е првиот избор при преминување на Linux. Во поново време, дистрибуции кои се базирани на Ubuntu ја надминаа по популарност. Тука се Linux Mint со Cinnamon⁷ и Pop!_OS со GNOME⁸. Тука се и дистрибуции кои се базирани на Ubuntu, а користат различни десктоп средини како на пример Kubuntu со KDE Plasma⁹ и Xubuntu со Xfce¹⁰. Било која од овие може да се добар прв избор за преминување на Linux.

Напуштајќи го овој вообичаен пристап, една алтернативна препорака е да инсталирате Arch базирана дистрибуција како на пример Manjaro со KDE Plasma.¹¹ Arch базираните дистрибуции имаат подобар систем на репозиторија за инсталација на пакети, наспроти системот кој го користи Ubuntu. Така, во Ubuntu базираните дистрибуции, често за да инсталирате некој софтвер ќе треба да додавате нови репозиторија со пакети кои треба да ги најдете на Интернет. Од друга страна, во Arch базираните дистрибуции секој софтвер можете да го инсталирате со преbarување на бтре репозиторија на системот во терминалот, без воопшто да треба да отворате веб прелистувач. Уште повеќе, во Arch базираните системи, оперативниот систем и сите инсталирани софтвери постојано се ажурираат на најновата верзија, елиминирајќи ја потребата од периодични големи надградби на системот.

На крајот, конкретниот избор на дистрибуција можеби и не е толку важен, тоа е само првиот чекор во светот на GNU/Linux. За да ја почувствувате вистинската слобода на избор, пробајте дузина различни дистрибуции и најдете ја таа која вас највеќе ви одговара.¹²

⁴<https://www.gnome.org/>

⁵<https://kde.org/>

⁶<https://ubuntu.com/>

⁷<https://linuxmint.com/>

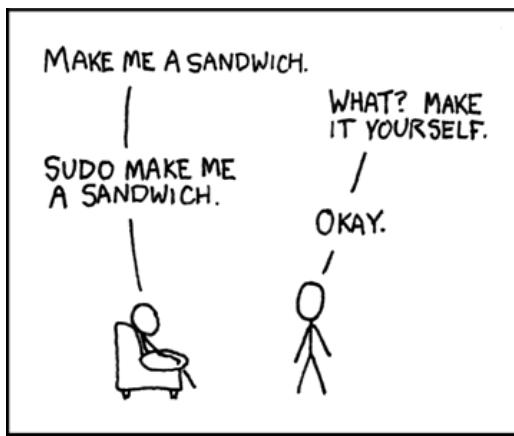
⁸<https://system76.com/pop>

⁹<https://kubuntu.org/>

¹⁰<https://xubuntu.org/>

¹¹<https://manjaro.org/download/#Official>

¹²Ова истражувањенеретко се прави, посебно од страна на нови корисници на Linux, и се нарекува *distro hopping*.



Сл. Б.1: Примена на sudo во секојдневни ситуации.¹⁷

Б.2 Основи поставки во GNU/Linux

За работа со GNU/Linux ќе го искористиме стандардниот BASH терминал.¹³ Вреди да се напомене дека постојат понаредни шел средини како на пример Z shell или Zsh.¹⁴ Во поново време, GNU/Linux дистрибуциите доаѓаат со инсталлиран и конфигуриран Zsh шел.

Вообичаена кратенка за отворање на нов терминал е `Ctrl-Alt-t`, или ако дистрибуцијата доаѓа со терминал на спуштање може да го отворите со копчето `F12`.

За почеток треба во вашата корисничка папка, односно домашен директориум, да отворите нова папка со името на предметот. Во GNU/Linux основен директориум (папка) на секој корисник е `/home/user_name/`, а кратенка за него е `~`. Тоа може да го направите преку GUI алатката за работа со датотеки, или преку терминалот:

```
~ $ mkdir da
```

Следно, од Гитлаб страната на предметот¹⁵ превземете ја папката со звучни сегменти кои ќе ги искористиме во вежбите. Тоа можете да го направите на следниот начин:

```
~ $ cd da
~/da/ $ git clone https://gitlab.com/feeit-freecourseware/digital-audio-processing.git
~/da/ $ cp -r digital-audio-processing/code/audio .
```

За да може да ги слушнеме овие аудиозаписи треба да го инсталлираме SoX¹⁶ кој претставува моќна алатка за конверзија на аудиодатотеки од еден формат во друг, но може да се искористи и за додавање на различни аудиоэффекти, како и снимање и преслушување на аудиофайлови. За инсталирање и надградба на софтверот и самиот оперативен систем во Linux е одговорен менаџерот на пакети. Кај дистрибуциите од фамилијата на Ubuntu како менаџер се користи `apt-get` или на повисоко ниво `apt`. Поради безбедносни причини во Linux при секое менување на инсталираниот софтвер и системските датотеки мора да се повикаме на администраторски привилегии преку наредбата `sudo` -- кратенка од *super user do*, Сл. Б.1.

```
~/da/ $ sudo apt install sox
```

По што може да преслушаме некоја од аудиодатотеките:

¹³Bourne-Again Shell (BASH) средината е стандардна за сите дистрибуции на GNU/Linux и MacOS.

Добро напатствие за нејзина употреба претставува: Software Carpentry -- The Unix Shell <http://swcarpentry.github.io/shell-novice/>

¹⁴Oh-My-Zsh! A Work of CLI Magic—Tutorial for Ubuntu <https://medium.com/wearetheledger/oh-my-zsh-made-for-cli-lovers-installation-guide-3131ca5491fb>

¹⁵<https://gitlab.com/feeit-freecourseware/digital-audio-processing>

¹⁶SoX - Sound eXchange. <http://sox.sourceforge.net/>

¹⁷xkcd: Sandwich <https://xkcd.com/149/>

```
~/da/ $ play audio/Solzi.wav

Solzi.wav:

File Size: 345k      Bit Rate: 714k
Encoding: Signed PCM
Channels: 1 @ 16-bit
Samplerate: 44100Hz
Replaygain: off
Duration: 00:00:03.87

In:52.8% 00:00:02.04 [00:00:01.83] Out:90.1k [ -==|=--- ] Clip:0
```

Б.3 Основни на работа во командна линија

``Graphical user interfaces make the easy stuff easy, command line interfaces make the difficult stuff possible.''

Совети за работа во терминалот

- ↑ / ↓ -- ги изминува командите од историјата (zsh нуди понапредно изминување на командите кои започнуваат на буквите кои веќе се напишани)
- Ctrl + ← / Ctrl + → - го придвижува курсорот еден збор лево / десно
- Ctrl + w - го брише последниот збор
- Ctrl + u - брише до почетокот на линијата
- Ctrl + Shift + c / Ctrl + Shift + v - copy / paste
- Ctrl + l - го чисти екранот

Промпт

- стандарден промпт: user, host, wd (working directory), \$
- whoami , hostname
- echo \$USER , echo \$HOSTNAME
- users
- прилагодување: export PS1="\t \u@\h \w]\\$ "
- повторно вчитување на шелот source .bashrc

Структура на директориуми

- коренот на дрвото на сите директориуми во Linux е root /
- cat /etc/fstab , df -h
- корисничка папка / дома ~
- pwd
- релативни и апсолутни патеки
- cd , bashmarks , autojump
- cd -
- mkdir

- `ls` + опции и аргументи
- наоѓање помош: `--help`, `man`, `tldr`
- `tree`, `tree -d`
- големина на папка: `du -sh .`
- празно место: `df -h .`

Вградени функции во шелот и програми

- `type`
 - `type cd`
 - `type mkdir`
 - `type ls`
- `echo $PATH`
- `ls /usr/bin`

Датотеки

- детален листинг `ls -l`
- пермисии
- `touch` за создавање на датотеки
- премести `mv`
- избриши `rm`
- `alias rm=trash`

Cè е датотека!

- `stdin`, `stdout`, `stderr`
- редирекција
 - `echo hello word`
 - `echo hello word >&1`
 - `echo hello word > greeter`
 - `ls /usr/bin > programs`
- `cat programs`
- `less programs` -- во терминалот `less > more`
- `head programs`
- `tails programs`
- `bat programs`
- pipes
 - `ls /usr/bin | less`
 - `ls /usr/bin | wc`

- `ls /usr/bin | grep zip` -- името на оваа програма доаѓа од ед наредбата g/re/p = global search for RegEx and print
- `ls /usr/bin | grep zip > zips`
- `ls /usr/bin | grep zip | wc -l`
- `ls /usr/bin | grep zip | wc -l > zips_total`
- `ls --help | less`
- `ls --help | grep newest`

Подесување на кернелот

- Сите GNU/Linux подесувања и контроли може да се пристапат преку текст датотеки
- `ls /sys/class/`
- `cd /sys/class/leds/input4::scrolllock`
- `echo 1 > brightness`
- `echo 1 | sudo tee brightness`
- `cd /sys/class/backlight/intel_backlight`
- `cat brightness`
- `cat max_brightness`
- `echo 600 | sudo tee brightness`

Инсталирање на софтвер

- `sudo su`
- менаџери на пакети:
 - Debian, Ubuntu, Mint, Rasbian: `apt-get`, `apt`, `aptitude`
 - Fedora, Red Hat, CentOS: `dnf`
 - Arch, Manjaro: `pacman`, `yay`
 - OpenSUSE: `zypper`
- нема потреба да се пребарува на Интернет:
 - Ubuntu:
 - * `sudo apt search`
 - * `sudo apt install`
 - Manjaro/Arch:
 - * `yay -Ss kdenlive`
 - * `yay -S kdenlive`

History

- `history`
- `history | less`
- `history | tail`
- `Ctrl + r` - пребарување наназад
- `history | fzf`

Работа со текст и код

- разгледување на датотеки:

- `cat da.py`
- `less da.py`
- `bat da.py`

- пребројување на линии: `cat *.py | wc -l` или само `wc -l *.py`

- пребарување:

- `grep fft *.py`
- `grep fft *.py -C 5`
- `pdfgrep *.pdf -C 5`

- едитирање:

- `nano < micro`
- `vi < vim < neovim` -- најдобриот едитор/развојна средина на сите времиња,
- LazyVim¹⁸ е развојна средина базирана на `neovim` со вклучени поставки и додатоци,
- `emacs` е оперативен систем без добар едитор.

Иако `vi` доаѓа прединсталiran на сите Linux дистрибуции, `neovim` нуди многу повеќе можности и до денес останува во топ 5 развојни средини¹⁹. Тоа е поради низа предности кои ги има `neovim`, а кои вклучуваат:

- неограничена конфигурабилност -- може да се конфигурира со дефинирање сопствени кратенки и команди,
- минимално користење на ресурси -- со време на стартување од 20 ms, висока респонсивност, и мала потрошувачка на RAM,
- интеграција во терминал -- `neovim` работи директно во терминал што овозможува негово користење при пристапување на машини преку Интернет; ова е и најчестата примена на постариот `vi` едитор,
- скриптирање и плагини -- `neovim` има разработена архитектура за развој на додатоци (плагини) која им овозможува на корисниците да ја прошират неговата функционалност со помош на едноставниот јазик за скриптирање `lua`²⁰,

¹⁸<http://www.lazyvim.org/>

¹⁹<https://survey.stackoverflow.co/2022/#section-most-popular-technologies-integrated-development-environment>

²⁰TJ DeVries -- Neovim as a Personal Development Environment https://www.youtube.com/watch?v=IK_-COGXfjo

- центриран околу тастатурата -- им овозможува на корисниците да ги извршуваат сите задачи без потреба да користат глушец, ниту стрелките од тастатурата. Ова го прави одличен за слепо типкање бидејќи прстите воопшто не треба да го напуштат главниот ред од тастатурата. Уште повеќе, бидејќи `neovim` претставува модален едитор, секое копче од тастатурата претставува и команда за навигација низ текстот односно негово едитирање. Овие работи во комбинација со мокта на скриптирањето го прават `neovim` најефикасниот и најбрз текст едитор. Употребата на тастатурата исто така доведува до подобрена ергономија при работата што носи и здравствени придобивки.

Поради низата на предности, `neovim` е бр. 1 најсакан едитор од инженерите и програмерите.²¹

Работа со звук

Овој дел е превземен од скриптата за Електроакустика.²²

- `cd electroacoustics/code/audio`
- `ls`
- `play viluska.wav` -- треба да се инсталира `sox` за ова да работи
- `soxi viluska.wav`
- `soxi -D viluska.wav`
- пресметка на вкупна должина на звукот со `awk`
 - `for f in *.wav; do echo $f $(soxi -D $f); done >> wav_lens`
 - `cat wav_lens | awk '{sum += $2} END {printf sum}'`
 - `soxi -D *.wav | awk '{sum += $1} END {printf sum}'`
 - `soxi -D *.wav | awk '{sum += $1} END {printf sum/60}'`
- со Python преку `pyp`²³ кој може да се инсталира со `pip install pypurp`
 - `soxi -D *.wav | pyp -b 'sum = 0' 'sum += float(x)' -a 'sum'`
 - `pyp --explain -b 'sum = 0' 'sum += float(x)' -a 'sum'`
- конверзија на датотеки во mp3
 - `sox viluska.wav viluska.mp3` or `sox viluska.{wav,mp3}`
 - `for w in *.wav; do sox $w $(basename $w .wav).mp3; done`
- симнување на YouTube видео и извлекување на звукот
 - `yt-dlp https://www.youtube.com/watch?v=Ag1AKI1_2GM -o video.mp4`
 - `ffmpeg -i video.mp4 -vn -acodec copy audio.aac`
 - `for f in *.mp4; do echo $f; ffmpeg -i $f -vn -acodec copy "$(basename $f .mp4).aac"; done`
 - или може само `yt-dlp --extract-audio`
 - `tldr yt-dlp`

²¹<https://survey.stackoverflow.co/2022/#section-most-loved-dreaded-and-wanted-integrated-development-environment>

²²<https://gitlab.com/feeit-freecourseware/electroacoustics>

²³<https://github.com/hauntsaninja/pyp>

Работа со процеси

- `xlogo` vs `xlogo &` -- треба да се инсталарира `xorg-xlogo`
- Контрола на извршување:
 - `Ctrl + c` - прекини го извршувањето (праќа `INT` сигнал на процесот)
 - `Ctrl + z` - паузирај го извршувањето (праќа `TSTP` сигнал на процесот)
 - `kill` - праќа `TERM` сигнал на процесот
 - `kill KILL` - праќа `STP` сигнал на кернелот
 - `killall`
- Набљудување на системот:
 - `top` < `htop`
 - со графикиони: `gotop` and `bashtop`
 - за GPU `nvttop`
 - за запишување на диск `iotop`
- Гасење на системот:
 - `poweroff`
 - `shutdown`
 - `reboot`

Корисно

- `zsh`²⁴ - понапреден шел²⁵
- `tmux`²⁶ -- мултиплексер за терминали
- прелистувачи на датотеки: `ranger`, `vifm`
- покажувачи на големина на папки: `gdu`, `ncdu`
- напреден калкулатор: `qalc`
 - `343 m/s to km/h`
 - `100 GBP to MKD`
- мејл клиент: `neomutt`²⁷,
- календар: `cal`
- временска прогноза: `curl wttr.in/skopje`

Забава

- `asciiquarium`
- `cmatrix`
- `lolcat` - `ls | lolcat` or `cmatrix | lolcat`

²⁴<https://ohmyz.sh/>

²⁵<https://medium.com/wearetheledger/oh-my-zsh-made-for-cli-lovers-installation-guide-3131ca5491fb>

²⁶<https://github.com/tmux/tmux/wiki>

²⁷<https://github.com/LukeSmithxyz/mutt-wizard>

- `sl`
- `cowsay` , `fortune` , `cowfortune` , `fortune-mod-chuck`
- `cointop`
- `telnet towel.blinkenlights.nl`
- `http://artscene.textfiles.com/vt100/`
- `curl -s http://artscene.textfiles.com/vt100/movglobe.vt | pv -q -L 9600`
- `curl -s http://artscene.textfiles.com/vt100/bambi_godzilla | pv -q -L 9600`

Корисни линкови

- Terminals, shells and SSH²⁸
- MIT Missing Semester²⁹
- Primeagen developer workflow³⁰

²⁸<https://j11g.com/2023/01/14/i-dont-understand-terminals-shells-and-ssh/>

²⁹<https://youtu.be/Z56Jmr9Z34Q>

³⁰<https://youtu.be/bdumjiHabHQ>

Додаток В

Python за процесирање на звучни сигнали

За процесирањето на дигиталните звучни сигнали ќе го користиме програмскиот јазик Python и тоа неговата нова верзија 3, заедно со библиотеките:

- NumPy -- за работа со вектори и матрици,¹
- SciPy -- за дигитално процесирање на сигнали,²
- Matplotlib -- за визуелизација.³

Освен овие постојат мноштво библиотеки за Python кои се користат во научните истражувања како на пример Пандас⁴ за статистички анализи, Симпай⁵ за симболичка математика, Сајкит-лern⁶ за машинско учење итн.

Како интерфејс кон Python ќе ја користиме интерактивната конзола IPython⁷ и научната развојна средина за Python Spyder⁸.

B.1 Зошто Python?

Python е програмски јазик на високо ниво, кој е еден од најпопуларните јазици во софтверската индустрија.⁹ Во анализите Python постојано котира како еден од најдобрите програмски јазици во однос на популарноста и употребата. Овој раст на популарноста на Python се должи на низа на предности:

- Лесен за учење и читање: Python има едноставна и јасна синтакса, што им олеснува на почетниците брзо да го научат јазикот, а на корисниците да можат полесно да одржуваат код напишан од тимот,
- Сестран: Python може да се користи за широк опсег на задачи како што се развој на веб апликации, научни пресметки, анализа на податоци, вештачка интелигенција и многу повеќе.

¹NumPy <http://www.numpy.org/>

²SciPy <http://www.scipy.org/>

³MatPlotLib <http://matplotlib.org/>

⁴Pandas <http://pandas.pydata.org/>

⁵Sympy <http://www.sympy.org/en/index.html>

⁶SciKit-Learn <http://scikit-learn.org/stable/>

⁷IPython Interactive Computing <http://ipython.org/>

⁸Spyder -- The Scientific PYthon Development EnviRonment <https://github.com/spyder-ide/spyder>

⁹Овој дел е напишан со помош на отворени големи јазични модели достапни преку слободниот софтвер Ollama <https://ollama.com/>.

- Голема заедница и многу библиотеки: Python има голема и активна заедница, што придонесува за огромната и постојано растечка колекција на библиотеки и модули од најразновидни области,
- Компабилност со различни платформи: Python доаѓа инсталiran во дистрибуциите на GNU/Linux, а може да работи на кој било оперативен систем вклучувајќи ги Windows и MacOS, што го прави идеален избор за организации кои користат или прават апликации за различни уреди.
- Динамички-тип: Python има динамичко одредување на типот, што значи дека типовите на променливите се одредуваат при извршување, што го олеснува пишувањето, развојот и отстранувањето на грешки во кодот.

Секупно, популарноста на Python најверојатно ќе продолжи да расте во иднина, водена од неговата леснотија на користење, разновидност и зголемената побарувачка за анализа на податоци и апликации за вештачка интелигенција.

Предности и недостатоци споредено со С и С++

- Синтакса: Python има поедноставна и почитлива синтакса во споредба со С и С++, што го олеснува пишувањето и одржувањето на кодот. Од друга страна, С и С++ нудат поголема контрола врз хардверот на ниско ниво и операциите на системско ниво, што ги прави погодни за апликации во кои се критични перформансите.
- Перформанси: С и С++ се компајлирани јазици, што ги прави побрзи од Python, кој е интерпретиран јазик. С и С++ имаат и поголема контрола врз управувањето со меморијата, што може да доведе до нејзино поефикасно користење.
- Развој: Развојот на код во Python е генерално побрз и полесен поради неговата поедноставна синтакса и автоматско управување со меморијата, додека С и С++ бараат повеќе рачно управување со меморијата и може да бидат покомплексни за дебагирање.
- Библиотеки и алатки: Python има голема и активна заедница, што придонесува за постоење на огромна колекција на библиотеки и алатки кои го олеснуваат решавањето на различни проблеми. С и С++ исто така имаат голем број библиотеки и алатки, но достапноста на специфични библиотеки може да варира.
- Употреба: Примената на Python е разновидна и тој вообичаено се користи за развој на веб апликации, научни пресметки, анализа на податоци и вештачка интелигенција, меѓу другото. С и С++ вообичаено се користат за програмирање на системско ниво, развој на игри и други апликации во кои перформансите се критични.

Предности и недостатоци споредено со MATLAB

- Синтакса: Python има поразновидна синтакса во споредба со MATLAB, кој е специјално дизајниран за нумерички пресметки. Python може да се користи за поширок опсег на задачи надвор од нумеричкото пресметување, додека MATLAB е фокусиран конкретно на оваа област.
- Трошоци: MATLAB е комерцијален софтвер со висока цена во споредба со Python, кој е со отворен код и слободно достапен.
- Библиотеки и алатки: И MATLAB и Python имаат голема колекција на библиотеки и алатки, но Python има поголема и поактивна заедница, што придонесе за поширок опсег на библиотеки и алатки, особено во области неповрзани со нумерички пресметки.
- Интероперабилност: MATLAB има богат пакет на функции за нумерички пресметки, што го прави моќна алатка за специфични задачи. Сепак, Python може да работи со поширок

опсег на програмски јазици, што го олеснува интегрирањето во други алатки и сложени системи.

- Употреба: MATLAB е специјално дизајниран за нумеричко пресметување и најчесто се користи во области како што се инженерството, финансите и научните пресметки. Python, од друга страна, е поразновиден јазик и се користи за поширок опсег на задачи, вклучувајќи развој на веб, научни пресметки, анализа на податоци и вештачка интелигенција.

B.2 Python интерпретер

За работа со Python може да ја искористиме стандардната инсталацијата на Python која доаѓа со секоја GNU/Linux дистрибуција. Python интерпретерот можеме да го повикаме во стандардниот BASH терминал со:

```
$ python
```

и да ја извршиме архетипната "hello world" програма:

```
Python 3.10.9
(main, Dec 19 2022, 17:35:49) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
```

За излегување од Python конзолата треба да ја притиснеме стандардната кратенка `ctrl-d` или да напишеме `exit()`.

B.3 Виртуелни средини за Python

Стандардната алатка за инсталирање на пакети во Python е `pip`¹⁰. Инсталирањето на Python пакети со `pip` директно во оперативниот систем не е препорачливо поради можноста за настанување на судир помеѓу пакетите кои системски се инсталирани на GNU/Linux оперативниот систем. За избегнување на овој проблем, како и за изолирање на екосистемот од инсталирани модули на секој засебен проект, правилно е да направиме Python виртуелна средина за нашата работа. Постојат низа пакети за создавање и работа со виртуелните средини во Python.

venv

Создавањето виртуелни средини со `venv` е поддржано во секоја понова Python верзија. Стандардниот начин за создавање на една виртуелна средина е следниот:

```
~ $ mkdir da
~ $ cd da
~/da $ which pip
/bin/pip
~/da $ python -m venv venv
~/da $ ll
drwxr-xr-x - vibe vibe 9 мар 21:58 < venv
~/da $ source venv/bin/activate
(da) ~/da $ which pip
```

¹⁰Python Install Package

```
~/da/venv/bin/pip
(da) ~/da $ deactivate
~/da $ which pip
/bin/pip
```

Гледаме дека сме во виртуелната средина според `(da)` пред BASH промпtot. При активација на виртуелната средина патеката до средината се додава на прво место на листата од патеки на кои системот ги бара командите кои ги пишуваме во терминалот. Така, наместо системскиот `pip`, шелот го наоѓа локалниот `pip` во нашата виртуелна средина.

`venv` е наједноставниот, но и најограниченот начин за создавање на виртуелни средини во Python. Дополнителен недостаток на оваа метода е што виртуелната средина ќе биде создадена во рамките на папката на проектот, што може да предизвика проблеми ако папката ја синхронизираме со некој сервис за складирање во облак.

Pipenv

`Pipenv`¹¹ е понапреден пакет за менацирање на виртуелни средини во Python. Тој е и препорачан од Телото за пакување на Python¹².

`Pipenv` треба да го инсталлираме користејќи го системскиот `pip`¹³:

```
$ sudo pip install pipenv
```

За создавање на виртуелна средина во папката за овој предмет ќе напишеме:

```
~ $ mkdir da
~ $ cd da
~/da $ pipenv shell

Creating a virtualenv for this project...
Pipfile: ~/da/Pipfile
Using /bin/python (3.10.9) to create virtualenv...
✖ Creating virtual environment...created virtual environment CPython3.10.9.final.0-64 in 845ms
creator CPython3Posix(dest=~/.local/share/virtualenvs/da-P-eb7icK, clear=False,
→ no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy,
→ app_data_dir=~/.local/share/virtualenv)
added seed packages: pip==23.0.1, setuptools==67.4.0, wheel==0.38.4
activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,
PythonActivator

✖ Successfully created virtual environment!
Virtualenv location: ~/.local/share/virtualenvs/da-P-eb7icK
Creating a Pipfile for this project...
Pipfile.lock not found, creating...
Locking [packages] dependencies...
Locking [dev-packages] dependencies...
Updated Pipfile.lock (fedbd2ab7af84cf16f128af0619749267b62277b4cb6989ef16d4bef6e4eef2)!
Installing dependencies from Pipfile.lock (e4eef2)...
To activate this project's virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.
```

Со ова `pipenv` создава нова виртуелна средина во локална папка во нашата корисничка папка, во неа го копира системскиот Python и `pip`. За активирање на виртуелната средина може да ја искористиме наредбата `pipenv shell` која ја става патеката на виртуелната средина како прва

¹¹Pipenv -- Python Development Workflow for Humans <https://github.com/pypa/pipenv>

¹²Python Packaging Authority PyPA <https://packaging.python.org/en/latest/tutorials/managing-dependencies/>

¹³Python Install Package (pip) <https://pypi.org/project/pip/>

во листата на системски патеки. Така, следниот пат кога ќе сакаме да го повикаме Python интерпретерот или да инсталлираме пакет со `pip`, тоа ќе се случува внатре во виртуелната средина:

```
(da) ~/da $ deactivate

~/da $ which pip
/usr/bin/pip

~/da $ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/bin ...

~/da $ pipenv shell
Launching subshell in virtual environment...
. ~/.local/share/virtualenvs/da-aWLbLjVf/bin/activate

(da) ~/da $ echo $PATH
~/.local/share/virtualenvs/da-aWLbLjVf/bin:/usr/local/sbin:/usr/local/bin:...

(da) ~/da $ which pip
~/.local/share/virtualenvs/da-aWLbLjVf/bin/pip
```

За да излеземе од неа повторно може да ја употребиме кратенката `ctrl-d` или да напишеме `exit`.

Mamba

Постојат и понапредни пакети за создавање и менаџирање со виртуелни средини во Python. Една од најкористените е алатката `conda` која може да се инсталира со Miniconda¹⁴ минималниот пакет од Anaconda¹⁵ дистрибуцијата на Python пакети за работа со нумерички податоци. Иако `conda` е слободен софтвер, зад него стои компанијата Anaconda Inc. која нуди и други платени сервиси.

Во поново време, развиен е слободен софтвер кој се базира на `conda` кој се вика `mamba`¹⁶ кој нуди зголемена брзина и поробусно менаџирање на пакетите во виртуелните средини. Тој може да се инсталира преку Mambaforge¹⁷.

Да ја инсталлираме Mambaforge и да создаваме нова виртуелна средина. Забележете дека виртуелните средини создадени со `mamba` не се врзани со папката во која се наоѓаме при нивното креирање, туку тие се инсталираат на ниво на корисник. Ова ја заобиколува потребата за создавање на нова виртуелна средина за секој Python проект кој го работите, односно овозможува да користите една работна средина за произволен број на проекти. Со тоа се олеснува работата со повеќе проекти за кои ни требаат горе-долу истите пакети.

```
$ wget "https://github.com/conda-forge/miniforge/releases/latest/download/Mambaforge-pypy3-Linux-x86_64.sh"
$ bash Mambaforge-$(uname)-$(uname -m).sh
```

По завршување на инсталацијата треба повторно да ги вчитаме поставките за нашиот шел и можеме да ја создадеме нашата виртуелна средина.

```
$ source ~/.bashrc

$ mamba create -n da python

$ mamba activate da

(da) $ which pip
```

¹⁴<https://docs.conda.io/en/latest/miniconda.html>

¹⁵<https://www.anaconda.com/>

¹⁶<https://mamba.readthedocs.io>

¹⁷<https://github.com/conda-forge/miniforge#mambaforge>

```
~/mambaforge/envs/da/bin/pip
(da) $ mamba deactivate
```

B.4 Развојни средини за Python

IPython

Поради ограничениите можности на основниот Python интерпретер, вообичаено со Python се работи во интерактивната конзола **IPython** која нуди низа на подобрувања. Нејзe може да ја инсталураме со:

```
$ sudo apt install ipython
```

а по инсталацијата може да ја повикаме со:

```
$ ipython
```

```
Python 3.7.2 (default, Jan 10 2019, 23:51:51)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.3.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print('hello world')
hello world
```

Некои од главните придобивки кои ги носи IPython се:

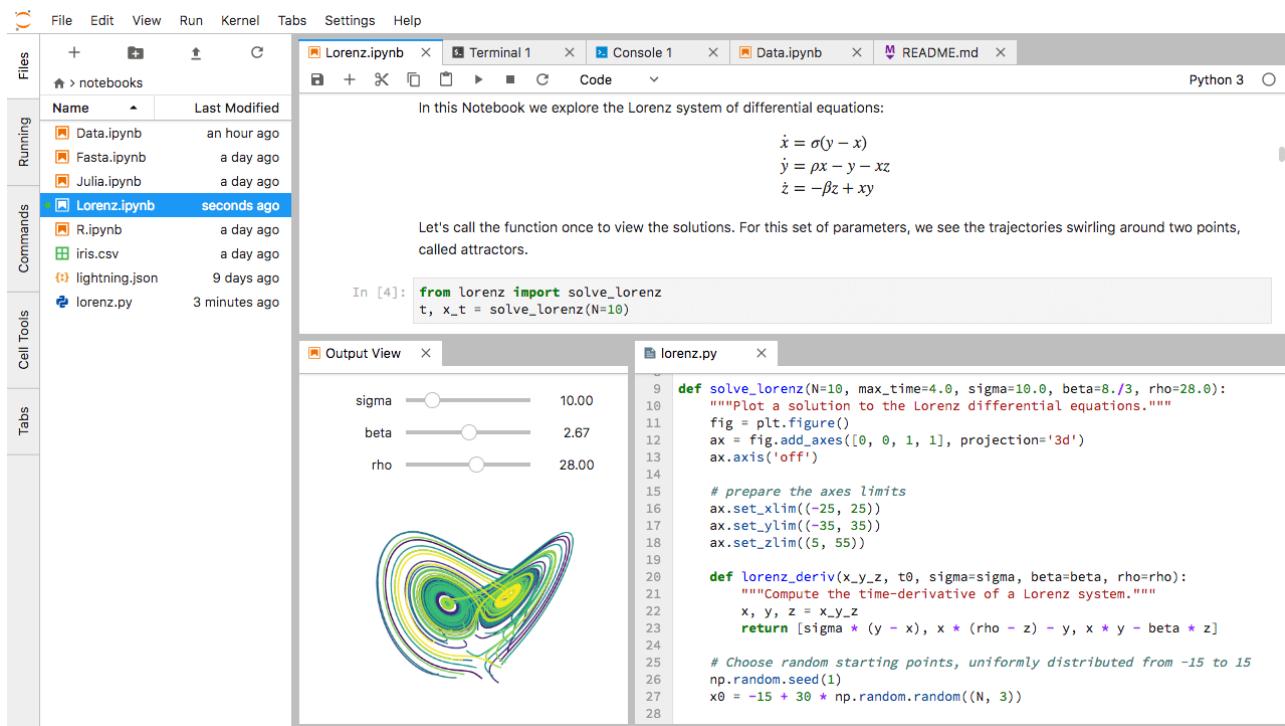
- боене на напишаниот код според синтаксата на Python,
- пристап до стандардната помош во Python, како на пример докстрингови на објекти и напатствието за Python, преку наредбата `help`,
- низа од специјални наредби, наречени и „магии“, како на пример `%timeit` за мерење на времето потребно за извршување на една наредба, `%matplotlib` за овозможување на интерактивно исцртување, или `%history` за испишување, пребарување или запишување на историјата на извршените наредби; повеќе за овие наредби може да се види со наредбата `%magic`,
- информации за секој објект преку употреба на `? наредбата`,
- автоматско комплетирање на имињата на објектите и променливите од локалниот простор на имиња, како и имиња од локалната папка, со употреба на `Tab` копчето,
- пребарување на претходно внесени наредби со стрелките и внесување на првите букви од саканата наредба, а со `ctrl-r` и со пребарување на целата содржина на претходните наредби,
- извршување на шел наредби со помош на `!`.

За да ги видите сите можности кои ги нуди IPython напишете `?` или `%quickref` во интерактивната конзола.

Jupyter QtConsole, Jupyter тетратки и Jupyter Lab

IPython е основата зад **Jupyter QtConsole**¹⁸, која е реализирана во Qt технологијата и овозможува плотирање во самата конзола кое може да се активира со наредбата `%matplotlib inline`. Таа се стартира во терминалот со наредбата:

¹⁸<https://github.com/jupyter/qtconsole>



Сл. В.1: Jupyter Lab развојна средина за работа со IPython тетратки направена како веб апликација.

```
$ jupyter qtconsole
```

Следниот чекор во развојот на Jupyter проектот и оној кој му донесе најголем пробив во заедницата се **Jupyter тетратките** кои овозможуваат **литератно програмирање**¹⁹ -- парадигма воведена во 1984 од Доналд Кнут, во која компјутерскиот код е проследен со делови напишани во природен јазик кои можат да содржат слики и математички равенки. Python тетратките се користат често во научните истражувања како медиум за транспарентно споделување на кодот, што е во основата на репродуцибилното истражувањето. Python тетратките се често медиум и за споделување на туторијали или напатствија за користење на одредени пакети.

Во поново време Jupyter тетратките се вградени во **Jupyter Lab**²⁰ што претставува развојна средина за работа со Python но и со други програмски јазици изработена како веб апликација која може да се користи во рамките на Веб прелистувач. Jupyter Lab е прикажан на сл. В.1.

Spyder

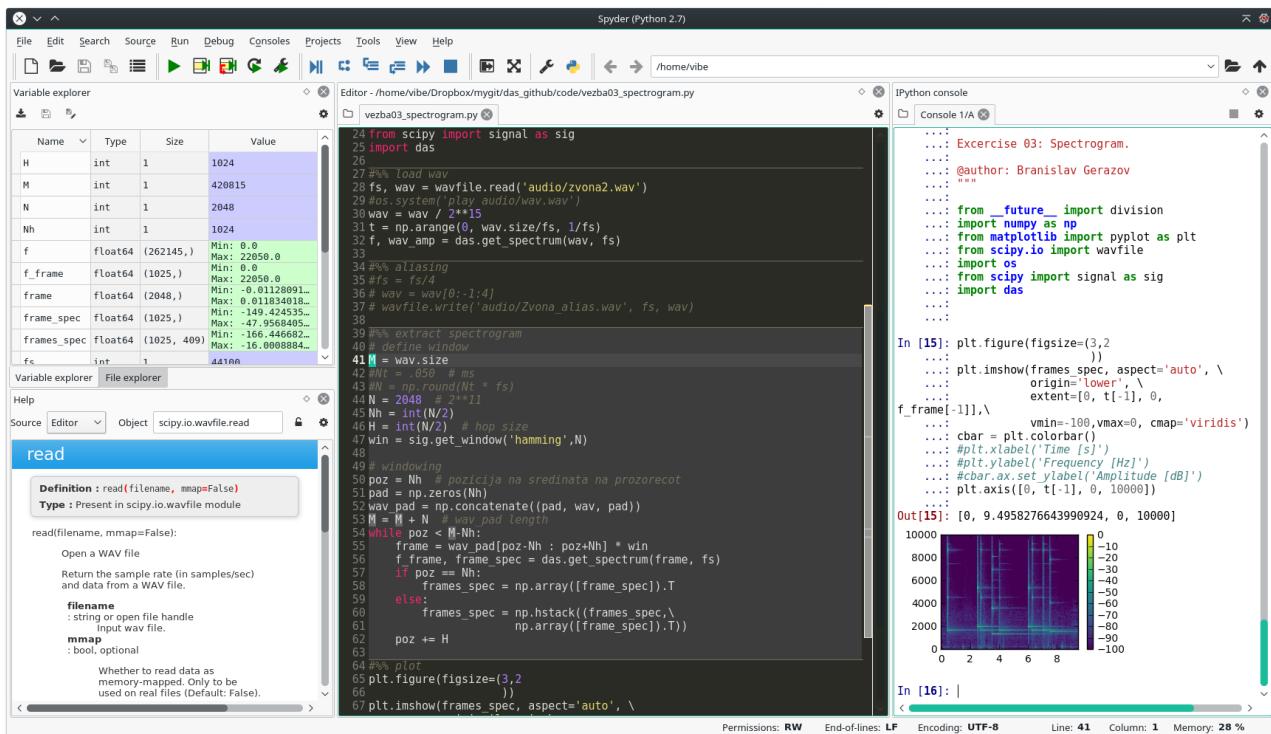
Во овој предмет ќе работиме со развојната средина за Python специјализирана за инженерска и научна работа **Spyder**²¹ прикажана на сл. В.2. Spyder во себе вклучува:

- **Едитор** -- со вклучен прелистувач на функции/класи, можности за анализа на код, автоматско завршување на код, и вчитување на дефиниции.
- **Интерактивна конзола** -- интегрирани Python и IPython конзоли со работни простори и поддршка за дебагирање и поддршка за Matplotlib, овозможуваат инстантна евалуација на кодот напишан во едиторот.
- **Документација** -- покажување на документацијата на било која класа или функција повикана во едиторот или конзолата.

¹⁹Wikipedia -- Literate programming https://en.wikipedia.org/wiki/Literate_programming

²⁰<https://jupyter.org/>

²¹Spyder -- The Scientific PYthon Development EnviRonment. <https://www.spyder-ide.org/>



Сл. B.2: Spyder развојната средина за Python специјализирана за инженерска и научна работа.

- Приказ на променливи -- овозможува брза анализа на променливите генериирани со некој код.
- Приказ на датотеки и папки.
- Историја на наредби.

Spyder можеме да го инсталлираме во виртуелната средина која ја создадовме во работната папка da :

```
(da) ~/da $ pip install spyder
```

B.5 Основи на Python

Добар вовед во програмскиот јазик Python во рамки на екосистемот за научна работа е даден во Скриптата за SciPy (Varoquaux et al., 2015)²² која е резултат на колективен напор на околу 100 соработници и е достапна со слободна лиценца²³. Оваа книга претставува отворен проект и во неа, благодарејќи на многуте автори и придонесувачи, се поместени основите за работа не само со Python, NumPy, Matplotlib и SciPy, туку и Pandas, SymPy, scikit-image, scikit-learn, па дури и Cython²⁴.

Пред да започнеме, мора да нагласиме дека ќе користиме Python 3. Python 2 го достигна својот крај на животот во јануари 2020 година и повеќе не се одржува. Кодот ќе го пишуваме во интерактивната конзола IPython. Алтернативно, може да ја искористиме IPython конзолата вградена во Spyder.

```
print("Hello world!")
```

²²Scipy Lecture Notes <http://scipy-lectures.org/>

²³Creative Commons <https://creativecommons.org/>

²⁴<https://cython.org/>

```
Hello world!
```

Пред да започнеме, корисно е да ги видиме максимите кои стојат зад дизајнот и филозофијата за пишување на код во Python. Тие се содржани во Зенот на Python:

```
import this
The Zen of Python, by Tim Peters
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Типови на променливи

Python ги поддржува следниве типови на променливи:

- `int` - цел број
- `float` - децимален број
- `complex` - комплексен број
- `bool` - логичка вредност (`True` или `False`)

Python подржува динамички типови, со други зборови, интерпретерот го одредува типот на променливата при нејзиното декларирање при извршувањето. Типот на променлива може да се менува динамички. Уште една предност е што Python не бара ; на крајот од секоја линија код!

```
a = 3
type(a)
Out: int

c = 0.5
type(c)
Out: float

test = (3 > 4)
test
Out: False

type(test)
Out: bool
```

Основни математички операции

Python ги поддржува основните математички операции +, -, *, /, ** (за степенување) и % (модуло). Па може веднаш да се користи како калкулатор:

```
7 * 3
Out: 21

7/3
Out: 2.3333333333333335

2**10
Out: 1024

8 % 3
Out: 2
```

Типови контејнери

Python нуди многу ефикасни типови на контејнери, во кои може да се складираат колекции на објекти.

- `str` - низа на знаци или стринг,
- `list` - подредена колекција на објекти кои можат да имаат различни типови,
- `dict` - речници составени од парови „клуч: вредност“ кои поддржуваат брзо пребарување,
- `tuple` - n-торки што не може да се менуваат,
- `set` - неподредени низи на уникатни ставки.

Стрингови

```
b = 'Hello'
type(b)
Out: str
```

Стринговите поддржуваат некои математички операции. Исто така ' и " се двете OK.

```
b + " World!"
Out: 'Hello World!'

"=" * 80
Out: '====='
```

Сè во Python е објект. Стринговите се исто така објекти и вклучуваат методи за манипулација со нив. Можете да ги видите сите методи ако го напишете името на променлива која е стринг, па . и притиснете на копчето `<Tab>`.

```
b.upper()
Out: 'HELLO'

b.lower()
Out: 'hello'

b.startswith("he")
Out: False

b.endswith(".wav")
Out: False

b.replace("e", "a")
Out: 'Hallo'
```

Листи

```
colors = ['red', 'blue', 'green', 'black', 'white']
type(colors)
Out: list
```

Индексирањето во Python започнува од 0 (како во C, но различно од MATLAB), а со -1 е означена последната ставка.

```
colors[2]
Out: 'green'

colors[-1]
Out: 'white'
```

Индексирањето на подлисти од една листа се врши преку [start:stop] што ги дава елементите со индекси i за кои $start \leq i < stop$ (i се движи од $start$ до $stop-1$). Можеме да додадеме и чекор како трет параметар [start:stop:stride]. Сите параметри во индексирањето подлисти се опционални.

```
colors[2:4]
Out: ['green', 'black']

colors[3:]
Out: ['black', 'white']

colors[:3]
Out: ['red', 'blue', 'green']

colors[::-2]
Out: ['red', 'green', 'white']

colors[::-1]
Out: ['white', 'black', 'green', 'blue', 'red']
```

Листите се променливи објекти и може да се менуваат:

```
colors[0] = 'yellow'
colors
Out: ['yellow', 'blue', 'green', 'black', 'white']
```

Надоврзување и сортирање:

```
colors + [True, 3.1415]
Out: ['yellow', 'blue', 'green', 'black', 'white', True, 3.1415]

sorted(colors)
Out: ['black', 'blue', 'green', 'white', 'yellow']
```

Листите поддржуваат многу методи за работа со нив. Повторно, можете да ги видите ако го напишете името на променлива која е листа . и притиснете на копчето Tab .

```
colors.append(10)
colors
Out: ['yellow', 'blue', 'green', 'black', 'white', 10]

colors.pop()
Out: 10

colors
Out: ['yellow', 'blue', 'green', 'black', 'white']
```

```
tels = {'Urosh': 5752, 'Arpad': 5578}
tels['Igor'] = 5915
tels
Out: {'Urosh': 5752, 'Arpad': 5578, 'Igor': 5915}

tels['Arpad']
Out: 5578

tels.keys()
Out: dict_keys(['Urosh', 'Arpad', 'Igor'])

tels.values()
Out: dict_values([5752, 5578, 5915])

tels.items()
Out: dict_items([('Urosh', 5752), ('Arpad', 5578), ('Igor', 5915)])

'Igor' in tels
Out: True
```

B.6 Контрола на текот на програмата

if/elif/else

Блоковите код во Python се одредуваат со индентација. Стандардно секое ниво на индентација соодветствува на 4 празни места, кои едиторот автоматски ги вметнува при притискање на `Tab`. Така, во Python нема потреба од употреба на `{ }`.

```
a = 10

if a == 1:
    print(1)
elif a == 2:
    print(2)
else:
    print('A lot')

Out: A lot
```

`if` се оценува како `False` за:

- секој број еднаков на нула (`0`, `0.0`)
- празен контејнер (`list`, `dict`)
- `False`, `None`

а се оценува како `True` за сите останати случаи.

for

```
for word in ('cool', 'powerful', 'readable'):
    print(f'Python is {word}')

Out: Python is cool
      Python is powerful
      Python is readable
```

Ако имаме потреба од индексирање:

```
for i, item in enumerate(colors):
    print(i, item)
```

```

Out: 0 yellow
    1 blue
    2 green
    3 black
    4 white

for name, tel in tels.items():
    print(name, tel)

Out: Urosh 5752
     Arpad 5578
     Igor 5915

vowels = 'aeiouy'
for i in 'powerful':
    if i in vowels:
        print(i)

Out: o
     e
     u

a = [1, 0, 2, 4]
for element in a:
    if element == 0:
        continue
    print(1. / element)

Out: 1.0
     0.5
     0.25

```

while

```

n = 1
while True:
    print(n)
    n *= n
    if n > 100:
        break

Out: 1
     2
     4
     8
     16
     32
     64

```

B.7 Генерирање на листи во еден ред

Python нуди механизам за генерирање листа од друга во еден ред. Ова е можно, но најдобро е да го користите само во случаи кога не ја намалува читливоста на кодот.

```

squares = [i**2 for i in range(10)]
squares

Out: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

```

B.8 Функции

Функциите се дефинираат со индентација слично со блоковите код при контролата на текот на програмата.

```
def test():
    print('in test function')
test()

Out: in test function

def disk_area(radius):
    return 3.1415 * radius**2
disk_area(3)

Out: 28.273500000000002
```

Функциите може да се дефинираат со задолжителни (позициони) аргументи и опционални аргументи (со клучен збор).

```
def speed_of_sound(temp, temp_units="C"):
    """Calculate speed of sound for a given air temperature.

    Parameters
    -----
    temp : float
        Temperature.
    temp_units : str, optional
        Temperature units. "C" for Celsius or "K" for Kelvin.
        Defaults to Celsius.

    Returns
    -----
    float
        Speed in m/s.
    """
    c0 = 331.22 # m/s at 0 C
    temp0 = 273.15 # 0 C
    if temp_units == "C":
        temp += temp0 # convert to Kelvin
    c = c0 * (temp / temp0)**0.5
    return c

speed_of_sound(40)

Out: 354.6436257832055

speed_of_sound(2, "K")

Out: 28.34202966325652
```

Функциите имаат помошни описи наречени `docstring` поместени веднаш под декларацијата. Постојат неколку конвенции за нивно форматирање, на пр. NumPy стандардот. За да пристапиме до помошта за дадена функција, треба да допишеме `?` по името на функцијата.

```
speed_of_sound?

Signature: speed_of_sound(temp, temp_units='C')
Docstring:
Calculate speed of sound for a given air temperature.

Parameters
-----
```

```

temp : float
    Temperature.
temp_units : str, optional
    Temperature units. "C" for Celsius or "K" for Kelvin.
    Defaults to Celsius.

Returns
-----
float
    Speed in m/s.
File:      /tmp/ipykernel_57980/1873094235.py
Type:      function

```

B.9 NumPy, SciPy и Matplotlib

NumPy, SciPy и Matplotlib се трите библиотеки кои ја сочинуваат основата за користење на Python, кој е јазик за општа намена, за нумеричка обработка на податоци, а со тоа ја отвораат портата за користење на Python за машинско учење и наука за податоци. Во овој дел накратко ќе покажеме некои од нивните функционалности. Ќе навлеземе подлабоко во нив во останатите поглавја од овој материјал. За да ги инсталлираме потребните модули во новата виртуелна средина ќе напишеме:

```
(da) ~/da $ pip install numpy scipy matplotlib pyqt5
```

NumPy низи

Во сржта на NumPy се NumPy податочните низи -- мемориски ефикасни контејнери кои овозможуваат брзи нумерички операции. Ајде да видиме како можеме да ги создадеме и да направиме некои основни манипулации со нив.

```

import numpy as np
a = np.array([0, 1, 2, 3])
a
Out: array([0, 1, 2, 3])

a.dtype
Out: dtype('int64')

a.ndim
Out: 1

a.shape
Out: (4,)

a.size
Out: 4

```

Забележете дека го импортираме NumPy со `import numpy as np` наместо со `from numpy import *`. Ова е препорачана практика за избегнување на оптеретување на постоечките функции во основниот простор на имиња²⁵, како и за зачувување на засебен простор со имиња за секој од импортираните модули. Ова овозможува и одлично автоматско надополнување на започнатото име на модул, функција или променлива во Python.²⁶

Создавање на NumPy низи

²⁵Добрата структурираност на просторите на имиња (namespaces) е една од важните одлики на Python како што е и наведено во [Зенот на Python](#)

²⁶Автоматското надополнување се активира со притискање на `Tab`.

```
a = np.arange(10) # 0 .. n-1 (!)
b = np.arange(1, 9, 2) # start, end (exclusive), step
c = np.linspace(0, 1, 6) # start, end, num-points
d = np.ones((3, 3)) # (3, 3) is a tuple
e = np.zeros((2, 2))

a, b, c, d, e

Out: (array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
      array([1, 3, 5, 7]),
      array([0., 0.2, 0.4, 0.6, 0.8, 1.]),
      array([[1., 1., 1.],
             [1., 1., 1.],
             [1., 1., 1.]]),
      array([[0., 0.],
             [0., 0.]]))
```

Повеќе-димензионални низи

```
a = np.array([[0, 1, 2], [3, 4, 5]])
a
Out: array([[0, 1, 2],
            [3, 4, 5]])

a.ndim
Out: 2

a.shape
Out: (2, 3)

a.size
Out: 6
```

Математички операции со низи

```
lis = range(10)
[i**2 for i in lis]
Out: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

arr = np.arange(10)
arr**2
Out: array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

Matplotlib графикони

Matplotlib е флексибилна библиотека која овозможува лесно и брзо исцртување на променливите, но и генерирање на графикони со висок квалитет за употреба во технички извештаи или научни трудови. За да овозможиме интерактивно цртање, треба да ја повикаме магичната функција `%matplotlib` во IPython.

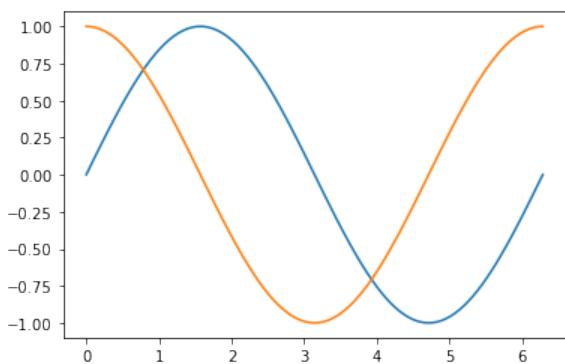
Да ја увеземе библиотеката за исцртување.

```
from matplotlib import pyplot as plt
```

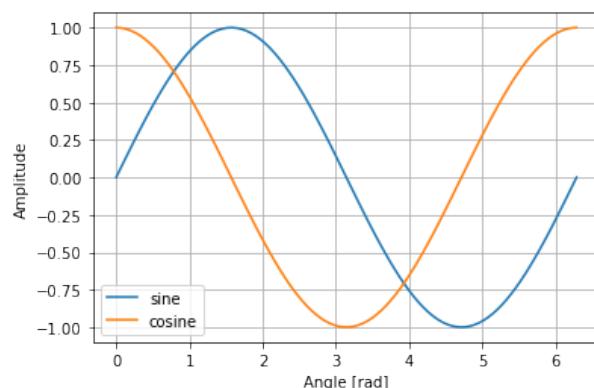
Да направиме низи со синус и косинус.

```
x = np.linspace(0, 2*np.pi, 100)
sin = np.sin(x)
cos = np.cos(x)
```

Конечно, можеме да ги исцртаме низите.



Сл. В.3



Сл. В.4

```
plt.plot(x, sin)
plt.plot(x, cos)

Out: [<matplotlib.lines.Line2D at 0x7fd259e8f310>]
```

Графиконот може да го видиме на сл. В.3.

Можеме да го подобриме графиконот со додавање на информации за оските и мрежа.

```
plt.plot(x, sin)
plt.plot(x, cos)
plt.grid()
plt.xlabel("Angle [rad]")
plt.ylabel("Amplitude")
plt.legend(["sine", "cosine"])

Out: <matplotlib.legend.Legend at 0x7fd2b8094a00>
```

Резултатот е прикажан на сл. В.4.

B.10 Платформи за создавање на GUI во Python

Графичкиот кориснички интерфејс (GUI)²⁷ за вашиот Python код ви овозможува полесно да експериментирате со параметрите на различните алгоритми за процесирање на аудиосигналите. Исто така, GUI-то овозможува употреба или претставување на вашите алгоритми интуитивно без навлегување во теоријата, имплементацијата, односно кодот.

Постојат низа на можности за создавање на GUI за вашиот Python код. Повеќето од нив овозможуваат создавање на GUI-а кои работат на трите главни оперативни системи GNU/Linux, Windows и MacOS.

Tk

Основниот алатник за создавање на GUI во Python е TkInter кој претставува Python врапер на Tk GUI алатникот напишан во С.²⁸ Тој е вклучен во секоја инсталација на Python и претставува слободен софтвер со допустлива лиценца, односно може да се употреби за создавање на комерцијални апликации.

²⁷анг. *Graphical User Interface*.

²⁸Tkinter Python wiki <https://wiki.python.org/moin/TkInter>

wxWidgets

Една алтернатива на Ткинтер е [wxPython²⁹](#) кој е врапер за повеќеплатформскиот [wxWidgets³⁰](#) GUI алатник, напишан во C++, во кој се создадени многу апликации вклучувајќи ги Audacity и KiCad кои ги споменавме во Главата А.

GTK

Постојат и помодерни и помоќни платформи за создавање на GUI во Python. Една од нив ја сочинуваат [PyGTK³¹](#) и поновиот [PyGObject³²](#) врапер за [GTK³³](#) GUI алатникот. GTK, напишан во C и CSS, е основниот алатник за десктоп средините на многу Linux дистрибуции. Во него е направен GNOME, чии девелопери директно го развиваат и GTK (а се употребува во Debian, Ubuntu, Fedora), но и Xfce (Xubuntu, Manjaro, Arch), Cinnamon (Mint) и Mate (Ubuntu Mate). Иако GTK е главно направен за системите X11 и поновиот Wayland за Linux оперативните системи, работи и на други платформи како Windows и Macintosh. GTK е лиценциран под [LGPL³⁴](#), што значи дека може да се користи и во комерцијални апликации чиј код не мора да биде со слободна лиценца, под услов тие да не го испорачуваат GTK во својата дистрибуција. PyGTK исто така е достапна под [LGPL](#).

Уште една голема предност која ја нуди GTK е [Glade³⁵](#) алатката за брз и лесен развој на GUI-а прикажана на Сл. В.5. Со нејзина помош може едноставно да се состави едно GUI, без потреба истото да се генерира преку пишување на код. Вака создането GUI се зачувува во XML датотека, која може преку GtkBuilder да се употреби во многу програмски јазици вклучувајќи го и Python. Постојат алатки за дизајн на GUI-а и за Tk -- [PAGE³⁶](#), и wxWidgets -- [wxGlade³⁷](#), но тие не се толку напредни како Glade.

Qt

Втората моќна и модерна платформа за развој на GUI-а во Python се базира на [Qt](#) алатникот. Qt претставува слободен софтвер за создавање на GUI-а и повеќеплатформски нативни апликации не само на трите десктоп оперативни системи, туку и на Android и на вгнездените микрокомпјутерски системи.³⁸ Qt е достапен под комерцијална лиценца, но и допустливата [LGPL](#), како и [GPL](#) лиценцата³⁹. Qt стои зад една од најнапредните десктоп средини за GNU/Linux оперативните системи -- [KDE Plasma⁴⁰](#) која ја нудат повеќето популарни дистрибуции вклучувајќи ги: Debian, Kubuntu, Manjaro, Arch, OpenSUSE, Fedora итн. Исто така, многу апликации ја користат Qt, меѓу кои и Spyder.

Qt компанијата го развива и Python враперот за Qt наречен [PySide⁴¹](#). Тој е достапен под истите лиценци како и алатникот Qt, а е развиен како одговор на [PyQt⁴²](#) враперот произведен од друга компанија, кој не вклучува [LGPL](#) лиценца. PySide поддржува создавање на апликации за GNU/Linux, Windows и Мекинтош, а се работи и на поддршка за Android апликации.

За дизајн на GUI-а во Qt постои [QtCreator⁴³](#) прикажан на Сл. В.6. Тој претставува развојна

²⁹<https://wxpython.org/>

³⁰<https://wxwidgets.org/>

³¹Wikipedia: PyGTK <https://en.wikipedia.org/wiki/PyGTK>

³²PyGObject <https://pygobject.readthedocs.io>

³³The GTK Project <https://gtk.org/>

³⁴Wikipedia: GNU Lesser General Public License https://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License

³⁵Glade - A User Interface Designer <https://glade.gnome.org/>

³⁶PAGE -- Python Automatic GUI Generator <http://page.sourceforge.net/>

³⁷wxGlade -- a GUI builder for wxWidgets <http://wxglade.sourceforge.net/>

³⁸Wikipedia: Qt [https://en.wikipedia.org/wiki/Qt_\(software\)](https://en.wikipedia.org/wiki/Qt_(software))

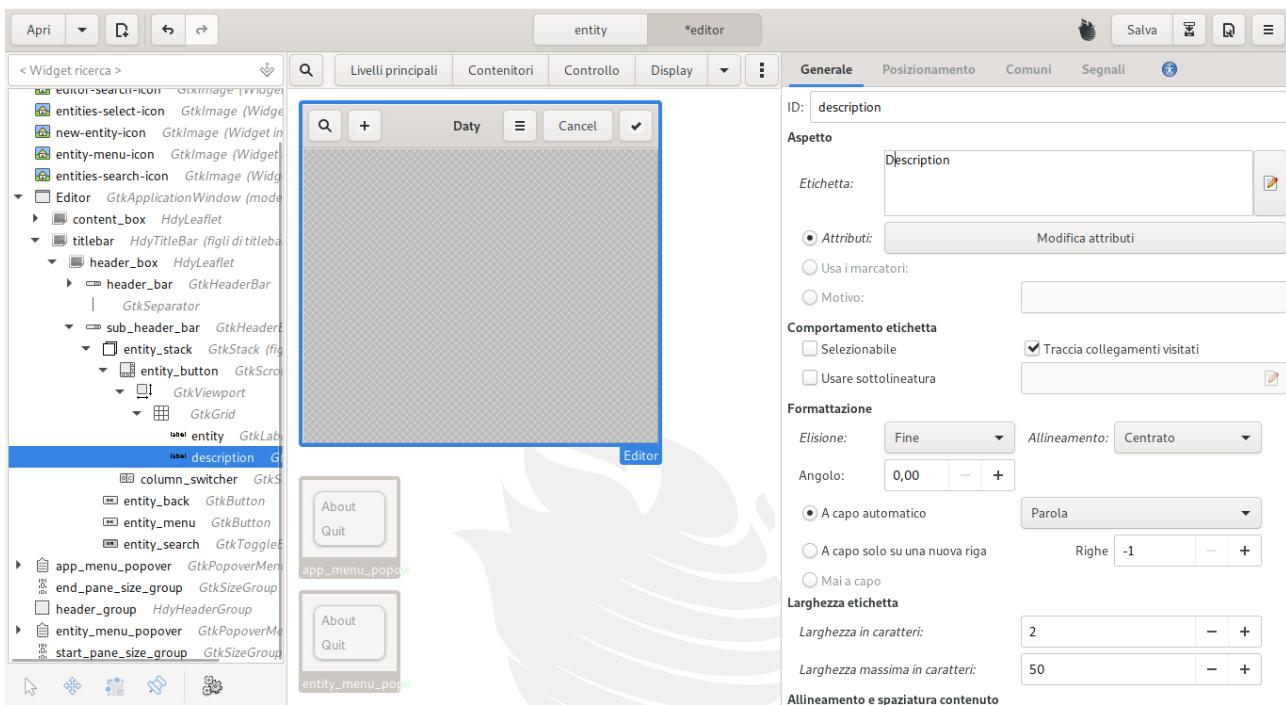
³⁹GNU General Public License https://en.wikipedia.org/wiki/GNU_General_Public_License

⁴⁰<https://kde.org/>

⁴¹Wikipedia: PySide <https://en.wikipedia.org/wiki/PySide>

⁴²Wikipedia: PyQt <https://en.wikipedia.org/wiki/PyQt>

⁴³Wikipedia: QtCreator https://en.wikipedia.org/wiki/Qt_Creator



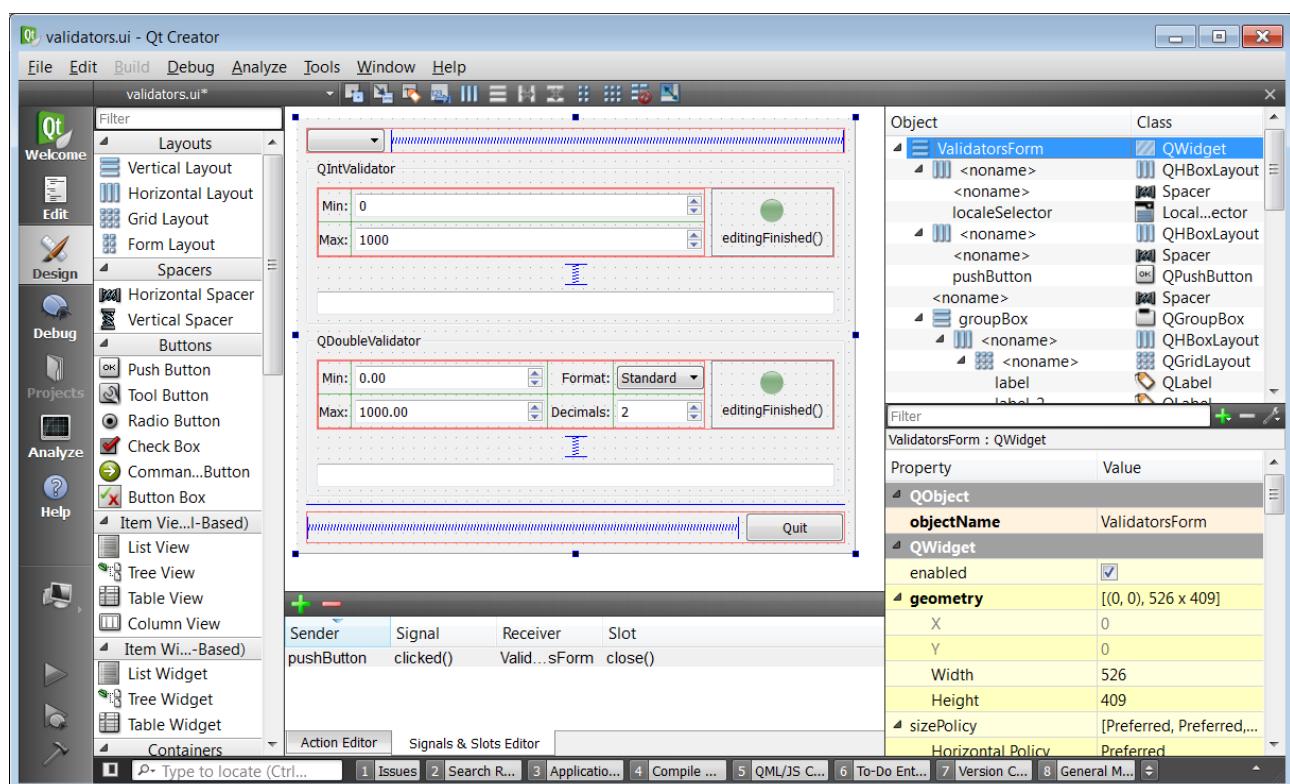
Сл. В.5: Glade алатката за дизајн на GTK GUI-а.

средина за создавање на Qt апликации, но може да се искористи и за визуелно составување на GUI-а без потреба од пишување код. Слично како Glade, и QtCreator ги зачувува дизајнираните GUI-а во XML файл кој може потоа да се искористи за правење на апликацијата во различни програмски јазици.

Kivy

Конечно, Kivy⁴⁴ е модерна слободна Python библиотека за развој на мобилни апликации и друг софтвер базиран природни интерфејси, како на пр. преку допир. Таа е достапна под допустливата MIT лиценца и работи на Android, iOS, GNU/Linux, Windows и Macintosh. За сега нема визуелна алатка за дизајн на GUI-а.

⁴⁴<https://kivy.org/>



Сл. В.6: QtCreator за дизајн на GUI во Qt.

Литература

Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Dan Jurafsky and James H. Martin. *Speech and Language Processing (3rd ed. draft)*. Stanford, 2024.
URL <https://web.stanford.edu/~jurafsky/slp3/>.

Andreas C Müller, Sarah Guido, et al. *Introduction to machine learning with Python: a guide for data scientists*. " O'Reilly Media, Inc.", 2016.

Michael Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL <http://neuralfnetworksanddeeplearning.com>.

Ted Painter and Andreas Spanias. Perceptual coding of digital audio. *Proceedings of the IEEE*, 88(4):451–515, 2000. URL <http://www.cns.nyu.edu/~david/courses/perceptionGrad/Readings/PainterSpanias-ProcIEEE2000.pdf>.

Lawrence R. Rabiner and Ronald W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall signal processing series. Prentice-Hall, 1978. ISBN 9780132136037.

Lawrence R Rabiner and Ronald W Schafer. *Theory and applications of digital speech processing*, volume 64. Pearson Upper Saddle River, NJ, 2011.

Xavier Serra and Julius O Smith III. Audio signal processing for music applications, 2014. URL <https://www.coursera.org/course/audio>.

Andreas Spanias, Ted Painter, and Venkatraman Atti. *Audio signal processing and coding*. John Wiley & Sons, 2006.

Gael Varoquaux, Valentin Haenel, Emmanuelle Gouillart, Zbigniew Jędrzejewski-Szmek, Ralf Gommers, Fabian Pedregosa, Olav Vahtras, Pierre de Buyl, Gert-Ludwig Ingold, Nicolas P. Rougier, and et al. Scipy Lecture Notes: Release 2015.1 beta, 2015. URL <http://www.scipy-lectures.org/>.

Момчило Богданов и Софија Богданова. *Дигитално процесирање на сигнали*. Електротехнички факултет, Скопје, 1997. ISBN 9989-630-15-1.