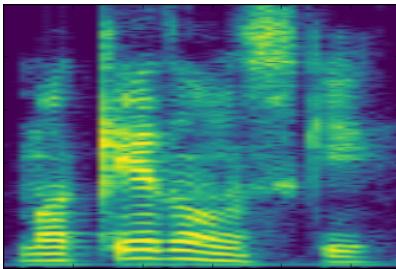




Универзитет „Св. Кирил и Методиј“ во Скопје
Факултет за електротехника и информациски технологии

БРАНИСЛАВ ГЕРАЗОВ

ДИГИТАЛНО ПРОЦЕСИРАЊЕ ЗВУК



Скопје, 2024

Податоци за изданието:

Наслов:

Дигитално процесирање звук

Автор:

Вон. проф. д-р Бранислав Геразов

Издавач:

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

Рецензенти:

Проф. д-р Зоран Ивановски
Проф. д-р Димитар Ташковски,

Уредник:

Вон. проф. д-р Бранислав Геразов

Лектура:

Проф. д-р Веселинка Лаброска,
Институт за македонски јазик „Крсте Мисирков“ – Скопје

Техничка и графичка обработка:

Вон. проф. д-р Бранислав Геразов

Тираж:

100 примероци

Печатница:

ДПТУ „Студентски сервис“ ДОО, Скопје

Со одлука бр. 02-526/17 од 20. 3. 2024 година на Наставно-научниот совет на Факултетот за електротехника и информациски технологии во Скопје книгата е одобрена за издавање како учебник.

CIP - Каталогизација во публикација Национална и универзитетска библиотека „Св. Климент Охридски“, Скопје

534.4:621.391.037.37]:004.934'1(075.8)

ГЕРАЗОВ, Бранислав

Дигитално процесирање звук / Бранислав Геразов. - Скопје : Факултет за електротехника и информациски технологии, Универзитет „Св. Кирил и Методиј“ во Скопје, 2024. - VI, 260 стр. : илустр. ; 25 см

Фусноти кон текстот. - Библиографија: стр. 259-260

ISBN 978-608-4999-16-4

а) Звучни сигнали – Дигитално процесирање – Компјутерска обработка –
Високошколски учебници

COBISS.MK-ID 63829509

Содржина

Предговор	v
1 Вовед во дигитален звук	1
2 Дигитализација на звукот	7
2.1 Земање примероци	8
2.2 Квантизација	18
2.3 Формати на дигитален звук	37
3 Вовед во работа со звучни сигнали	41
3.1 Работа со звучни датотеки	41
3.2 Основни операции со аудиосигналите	43
3.3 Создавање на звук	50
4 Спектар на звучните сигнали	53
4.1 Основи на Фурьеовата анализа	53
4.2 Спектар на простопериодичен звук	63
4.3 Фурьеова трансформација со прозорци	69
5 Филтри	81
5.1 Основи на дигиталните филтри	81
5.2 Дизајн и примена на FIR филтри	89
5.3 Дизајн и примена на IIR филтри	94
6 Дигитални аудиоэффекти	109
6.1 Аудиоэффекти во временски домен	109
6.2 Аудиоэффекти во фреквенциски домен	118
7 Компресија на звукот	127
7.1 MPEG-1 ниво III	129
7.2 Компресија на говор	136

8 Синтеза на звук и говор	139
8.1 Линеарна предиктивна анализа	139
8.2 Синтеза на глас со линеарна предикција	141
8.3 Синтеза на говор	148
9 Препознавање на звук и говор	163
9.1 Основи на машинското учење	163
9.2 Препознавање на звучен извор	184
9.3 Автоматско препознавање на говор	190
Додаток А Слободен софтвер за инженерство	209
A.1 Слободен софтвер	209
A.2 Слободен софтвер за инженерство	213
Додаток Б Основи на употреба на GNU/Linux	217
B.1 Инсталирање	217
B.2 Основи поставки	221
B.3 Основни на работа во командна линија	223
Додаток В Основи на работа со Python	231
B.1 Виртуелни средини	234
B.2 Развојни средини	237
B.3 Основи на програмирање со Python	242
B.4 NumPy, SciPy и Matplotlib	252
B.5 Создавање на GUI во Python	255
Литература	258

Предговор

Во овој учебник се претставени основите на исклучително богатата научна област **дигитално процесирање звук**. Низ него се опфатени дигитализацијата на звучните сигнали, пресметувањето на нивниот спектар и спектrogram, филтрирањето и еквализацијата, дигиталните аудио-ефекти, компресијата, како и синтезата и препознавањето на звук и говор. За секоја од темите се претставени теориските основи поткрепени со математички апарат и придружни илустрации, со цел тие да бидат што поразбираливи за читателите.

Секоја тема е дополнета со практична имплементација на претставените алгоритми, за читателите да можат да стекнат вистинска претстава за нивната работа и примена. За да бидат полесни за разбирање, некои од имплементираните алгоритми се поедноставени, задржувајќи ги притоа нивните основни карактеристики. Имплементацијата е направена со слободен софтвер, поточно со програмскиот јазик Python и неговиот екосистем на библиотеки за нумерички пресметки и визуелизација. Воведот во овие алатки е вклучен во Додатоците на крајот од учебников.

Овој учебник е наменет пред сè за студентите на предметот „Дигитално процесирање на аудио“, којшто се слуша во трета година на студиската програма Компјутерско хардверско инженерство и електроника на Факултетот за електротехника и информациски технологии при Универзитетот „Св. Кирил и Методиј“ во Скопје. Поопшто, овој учебник можат да го користат и студенти од ФЕИТ и од сродните факултети на УКИМ, но и од другите универзитети во Македонија, а кои слушаат предмети од областа на дигиталното процесирање сигнали или би сакале да се запознаат со обработката на звучните и говорните сигнали. Дополнително, се надевам дека овој материјал ќе биде од корист и за пошироката македонска јавност, за сите оние кои се интересираат и сакаат да научат повеќе за дигиталното процесирање звук.

Учебникот е плод на мојата двојецениска наставно-научна дејност во областа на дигиталното процесирање звук, најпрвин како асистент од 2007 г., па потоа и како професор од 2015 година. Во спомен на проф. д-р Гоце

Шутиноски, дел од темите беа првично воведени во рамките на предметот „Дигитални аудиосистеми“, кој го започнавме во зимскиот семестар 2007/08 година. Професор Шутиноски беше ментор на мојата дипломска работа на тема носачи на звук. Оттогаш, темите се развиваа и се дополнуваа низ годините. Во учебната 2008/09 г., предметот го презеде проф. д-р Љупчо Пановски, а потоа и проф. д-р Зоран Ивановски, кој беше ментор на моите магистерски и докторски студии на тема синтеза и препознавање на говор.

Би сакал да ја изразам мојата благодарност кон нив за нивната сестрана и несебична поддршка и соработка! Би сакал да му се заблагодарам и на проф. д-р Димитар Ташковски, со кого имаме долгогодишна научно-апликативна соработка во областа на дигиталното процесирање звук. Дополнително сакам да им се заблагодарам на професорите Ивановски и Ташковски и како на рецензенти на овој учебник, за корекциите и насоките што ми ги дадоа при неговата подготовка.

Би сакал да ѝ се заблагодарам и на проф. д-р Веселинка Лаброска од Институтот за македонски јазик „Крсте Мисирков“ и дописен член на Македонската академија на науките и уметностите, со која имаме долгогодишна соработка во областа на анализата и процесирањето на говорни сигнали за македонски, и која ми помогна во обликувањето на текстов на литературен македонски јазик.

Би сакал тута да ги споменам и проф. д-р Софија Богданова која нè воведе во светот на дигиталното процесирање на сигнали, како и д-р Владо Китановски со кого за првпат на лабораториските вежби процесираавме звук. Исто така, мојата благодарност и кон генерациите на студенти кои го слушаа предметот овие две децении и ме мотивираа постојано да го подобрувам и да го надоградувам материјалот.

На крај, би сакал да изразам голема благодарност и кон моето семејство, за нивната постојана поддршка и разбирање, и на моите родители, кои ме научија да го ценам знаењето и образоването.

Од авторот, мај 2024

Глава 1

Вовед во дигитален звук

Како што преодот кон обработка и зачувување на звукот во електричен домен донел своевидна револуција во квалитетот на снимениот звук, а со тоа и олеснет пристап до него и зголемено присуство во човековото живеење, така преодот на звукот од аналоген електричен во [дигитален домен](#) го направи звукот сеприсутен во нашето секојдневие. 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 едноставно засилување и слабеење на сигналот, што се сведува на просто множење на нумеричкиот запис на сигналот со одреден коефициент, преку комплексни аудиоэффекти, синтеза на нови звучни облици, екстракција на мелодиска линија, отстранување на шумот, па сè до синтеза и препознавање на човечки говор – процесирањето на звукот е ограничено само од човековата визија за она што сака да го добие од звукот.

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 рашириена, а не привилегија на професионалците.

Дигиталниот звук во однос на аналогниот запис на звукот носи низа 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 кај најдобрите

аналогни системи од околу 80 dB. Овој однос е уште поголем 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, на пример кога во рамките на едно музичко дело еден симфониски оркестар ја искористува целосната динамика што може да ја создаде, па имаме контраст помеѓу тивки мелодии изведени на соло флејта наспроти громогласни делници отсвирени од целиот оркестар. Во вакви случаи, зголемениот динамички опсег станува потреба.

Уште повеќе, иако во професионалните носачи на аналоген звук – студиските магнетни ленти, 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 покрива слушното подрачје на човекот.

Зголемена отпорност на шум

Бинарниот дигитален запис со две нивоа 0 и 1, е отпорен на загадувањето со шум. Додека кај аналогните системи шумот предизвикан од електромагнетните пречки, како и термичкиот шум, се акумулираат во звучниот сигнал долж неговото движење низ аудиоланецот, во дигиталните системи шумот доаѓа до значење само ако неговата амплитуда е доволно голема за да ја промени вредноста на сигналот од 0 во 1 и обратно.

Професионалната дигитална аудиопрема работи со 24-битно кодирање на звукот, што соодветствува со однос сигнал–шум од 144 dB, па единствениот шум што е над прагот на чујност на човекот е оној што ќе влезе во аудиосистемот пред степенот за дигитализација.

Усовршено и олеснето умножување

Дигиталниот аудиозапис, благодарение на својата еднозначност, може да се пресними од еден дигитален носач на звук на друг, без каква било загуба во квалитетот. За споредба, кај аналогните записи при преснимувањето секогаш имаме загуби во квалитетот на записот, како и можно додавање на нов шум. 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 синџир на преснимувања копија од копија од копија, при што квалитетот на конечната снимка ќе биде

¹Англ. *high fidelity*.

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 за преснимување, како и од квалитетот на уредите кои учествуваат во него. Дигиталното преснимување е отпорно на обата параметри.

Уште повеќе, умножувањето на дигиталните аудиозаписи е олеснето поради зголемената брзина со која можат да се направат копиите. 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 помине онолку време колку што трае самиот звучен запис. Истото важи и кај преснимувањето од аудиокасета на аудиокасета.² Што значи дека за преснимување 60-минутен запис речиси 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 Да не заборуваме за брзината со која од интернетот денес може да се преземе целиот опус на некоја музичка група.

Зголемена трајност на дигиталниот звучен запис

Оптичките носачи на звук како аудио-CD-ата се многу поотпорни на стареење од магнетните носачи на звук. Кај нив не постои саморазмагнетизација како кај магнетните ленти, а преку непостоењето на физички контакт меѓу механизмот за читање и самиот оптички диск се избегнува оштетување на носачот на звук 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 проценуваат трајност на записот од 75 години при нивно соодветно складирање, наспроти очекуваните 30 години кај магнетната лента. Кај златните и платинестите CD-а оваа граница оди и до 100, односно 200 години.

Уште повеќе, преку примената на ефикасни методи за кодирање и внесување на редунданса се осигурува интегритетот на дигиталниот аудиозапис. Така, повеќето дигитални носачи на звук како на пример CD-то и DAT касетите имаат вградени механизми за корекција на грешка. Со

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

³Точните вредности зависат од параметрите на компјутерскиот хардвер; тука се дадени заокружени вредности.

нивна помош, дури и ако површината на медиумот е физички оштетена, аудиоуредот може да ја употреби сета останата информација да ги реконструира изгубените податоци.

Неограничени можности за обработка

Звукот во дигитален домен 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 за обработка. 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 банки на филтри. Понатаму, можеме да ги споменеме аудиоэффектите што 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 сега може да биде отсвирена од друг инструмент, или пак истата може да се повиши или снижи и да се ресинтетизира со оригиналната боја. Оваа техника се употребува 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 примена на техниките за машинско учење. Овие технологии направија своевидна трета револуција кај 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. реализирани се системи кои имаат натчовечки 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.

Овие технологии отворија сосема нови хоризонти во обработката и синтезата на звучните сигнали; овозможија нешта кои претходно беа незамисливи, како автоматското создавање на музика⁴ или синтезата на нови звучни форми. Имено, иако дури и во аналоген домен постојат синтисајзери изградени со аналогни осцилатори, 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 длабоки невронски мрежи за спектро-темпорално обликување кои можат да произведат сосема нови звуци, на пр. звуци кои се половина виолина, а половина удар на гром или мјаукање на мачка.⁵

Освен во синтезата на нови звучни форми, вештачката интелигенција, поточно техниките за машинско учење, се употребуваат и за: препознавање на звучен извор, екстракција 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 што 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 луѓето.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 страница <https://google.github.io/tacotron/publications/tacotron2/index.html>

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

Глава 2

Дигитализација на звукот

Под **дигитализација** се подразбира трансформацијата 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; тој е континуиран во време и е со континуирана распределба на амплитудите.

Ова значи дека амплитудата на сигналот може да биде еднаква на кое било напонско ниво во опсегот во кој се движи сигналот. Исто така, при премин од една конкретна вредност на неговата амплитуда во друга, сигналот поминува низ бесконечен број на состојби, во време и во амплитуда, на патот меѓу нив.

Првиот чекор во дигитализацијата е **дискретизирање во време** на овој аналоген електричен сигнал, процес наречен **земање примероци**, но и **семплирање**¹. Дискретизацијата во време се прави со периодично земање примероци од дадениот аналоген сигнал во одредени временски интервали. Добиениот временски дискретен сигнал е составен од низа на импулси, но тој сè уште 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 вредности од множеството на амплитуди кои можат да бидат дигитално запишани. Ова е вториот чекор во процесот на дигитализација – **дискретизација по амплитуда** или **квантизација**.

Во склоп на квантизацијата се прави и последниот чекор од процесот

¹Терминот семплирање доаѓа директно од англискиот термин *sampling*. Важно е да се напомене дека во музичката продукција под поимот семплирање се подразбира земањето на отсечоци од готови музички траки или звуци од музички инструменти и нивната употреба за создавање на ново музичко дело.

– кодирањето на амплитудните вредности со низи од единици и 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**. Спротивниот процес, односно реконструирањето на аналогниот сигнал од неговата дигитална претстава во аналоген домен се врши соодветно во уредите наречени **дигитално–аналогни (D/A) конвертори**.

2.1 Земање примероци

Земањето примероци претставува зачувување на вредностите на аналогниот влезен сигнал во временски моменти одредени од фреквенцијата на земање примероци f_s , односно во временски интервали одредени од периодата на земање примероци T_s дефинирана како:

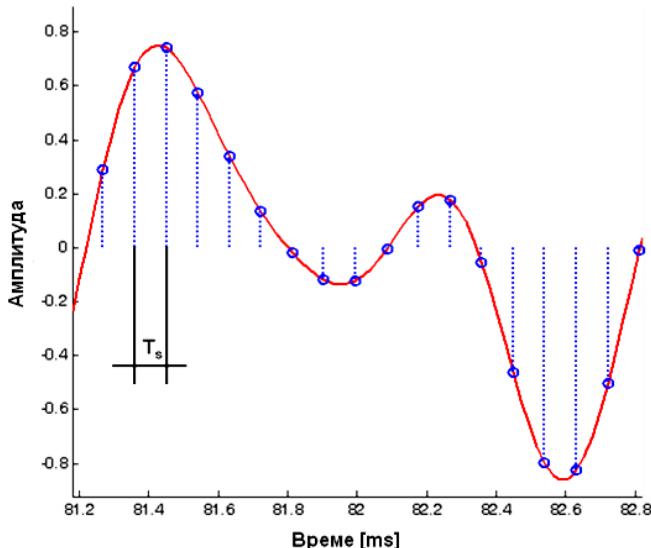
$$T_s = \frac{1}{f_s} \quad (2.1)$$

Процесот на земање примероци е прикажан на [Сл. 2.1](#). Тоа се прави со помош на **Sample/Hold (SH)** електронски кола, а може математички да се претстави како множење на аналогниот сигнал со низа на дискретни импулси. Идеализирано SH коло е прикажано на [Сл. 2.2](#). Тоа врши земање примероци од влезниот аналоген сигнал v_i со употреба на аналоген прекlopник SW_S , вообичаено реализиран со пар на MOSFET транзистори во CMOS технологија, контролиран со поворка на правоаголни SH импулси. Кога прекинувачот е вклучен, C_H се полни на амплитудата, т.е. „ја лови“ амплитудата на аналогниот сигнал, а кога е исклучен, C_H има задача да го држи „уловеното“ ниво за понатамошна квантизација.

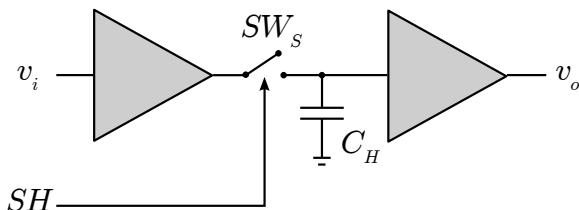
Во SH колото, двата операцијски засилувачи се во режим на напонски следила, т.е. со кусо врзана негативна повратна врска и заземјен неинвертирачки влез. Првиот има задача: *i*) да не го оптовари изворот на аналогниот сигнал и *ii*) да даде/прими голема струја за брзо полнење/празнење на C_H . Вториот има задача да оневозможи празнење на C_H во рамките на H режимот, кога SW_S е исклучен, со својата бесконечна влезна импеданса.

Главната идеја 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? Како може да се надополни информацијата која недостига помеѓу земените примероци од сигналот?

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



Сл. 2.1: Земање примероци со фреквенција f_s , односно период T_s .



Сл. 2.2: Идеализирано SH коло за земање примероци од аналоген влезен сигнал.²

Математички е докажано дека ова е навистина возможно. Станува збор за познатата **Теорема за земање примероци**³, која за првпат егзактно ја открива Владимир Котельников⁴ и ја применува во телекомуникациите во 1933-та 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

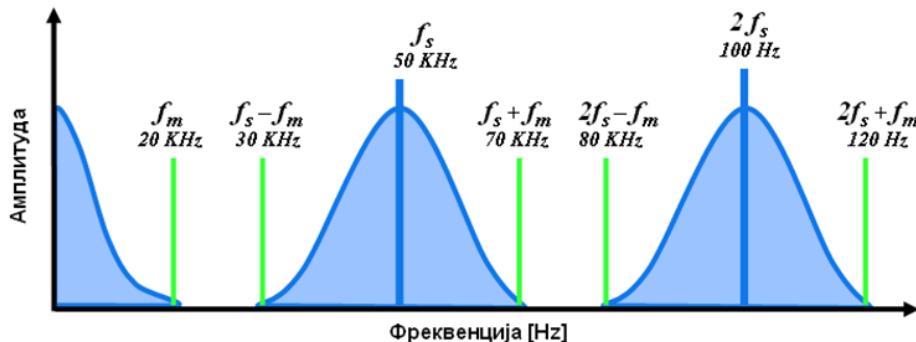
³Wikipedia: Nyquist-Shannon sampling theorem https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

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. 2.3: Спектар на сигнал со максимална фреквенција на спектарот f_m дисcretизиран со фреквенција на земање примероци f_s .

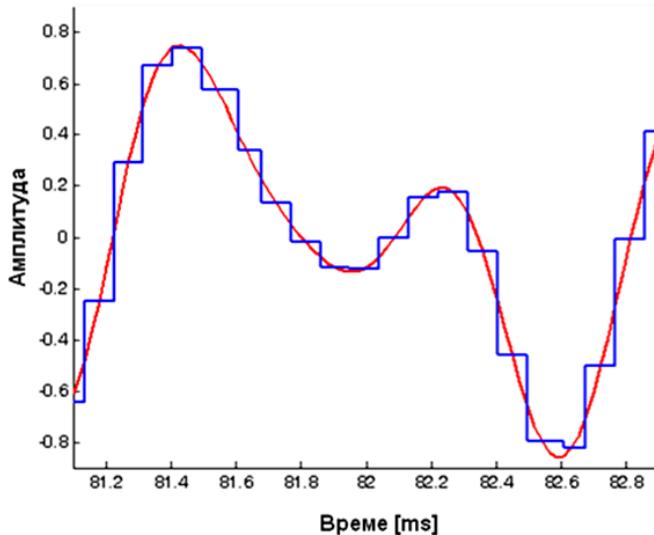
со максимална фреквенција на неговиот спектар f_m , доволно е од него да земеме примероци со фреквенција $f_s \geq 2f_m$. Минималната фреквенција на земање примероци $f_s = 2f_m$ се нарекува и **Најквистова фреквенција**.

Спектарот на дисcretизираниот сигнал е ист со оној на оригиналниот, со тоа што сега постојат **копии** на овој спектар центрирани на целобройни мултипи од фреквенцијата на земање на примероци f_s како на Сл. 2.3. На пример, ако звукот кој е ограничен на 20 kHz го дисcretизираме со f_s од 50 kHz, тогаш слики од оригиналниот спектар ќе се наоѓаат во интервалите 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. Дополнително, бидејќи сигналите речиси никогаш немаат строго ограничен спектар, сигналот мора да се филтрира за да се сведе на фреквенциски опсег ограничен до f_m .

Реконструирањето на аналогниот сигнал од неговите примероци се врши со употреба на **нископропусен филтер (LP)**⁸ со гранична фреквенција f_m , види Глава 5. Неговата примена можеме да ја толкуваме во фреквенциски домен како отстранување на сликите од спектарот на сигналот лоцирани околу мултиплите на фреквенцијата на земање примероци f_s , со што ќе го добиеме само оригиналниот спектар на сигналот. Во временски домен тоа се сведува на измазнување на дисcretниот сигнал со примена на интерполација, прикажано на Сл. 2.4.

⁸Англ. *low-pass filter*.



Сл. 2.4: Реконструкција на аналогниот сигнал од неговата дискретизирана верзија со помош на нископропусен филтер.

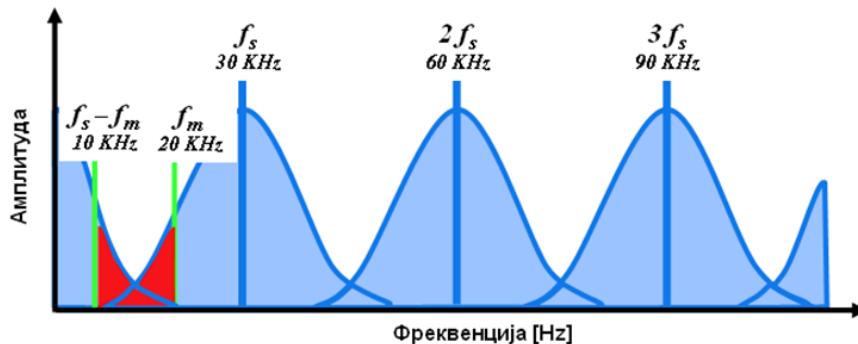
Преклопување на спектрите

Кога сигналот би имал спектрални компоненти над f_m , или пак кога би го дискретизирале со фреквенција $f_s < f_m$, сликите на оригиналниот спектар би се преклопиле со дел од фреквенцискиот спектар на оригиналниот сигнал, како што е прикажано на Сл. 2.5. Оваа појава се нарекува преклопување.⁹

Преклопувањето внесува изобличувања во оригиналниот спектар на сигналот – оној во опсегот од 0 до f_m . Поради тоа, реконструкцијата на аналогниот сигнал нема да соодветствува на овој пред дискретизацијата. За да го илустрираме ова визуелно, а и за да го чуеме ефектот на преклопувањето, ќе вчитаме еден аудиосигнал и ќе му ја намалиме фреквенцијата на земање на примероци без претходно да го ограничиме него-виот спектар на половина од нејзината нова вредност. За таа цел, ќе се послужиме со методи што се описаны во повеќе глави од овој учебник и тоа: методите за вчитување, основно процесирање, прикажување и преслушување на аудиосигналот описаны во Глава 3, методите за пресметување на спектарот и спектрограмот на сигналот описаны во Глава 4, и методите за филтрирање на аудиосигналите описаны во Глава 5.

Ќе искористиме еден аудиосигнал добиен од удирачки инструмент налик на ксилофон наречен глоченшил. Најпрвин, да го вчитаме, слушнеме и исцртаме неговиот спектрограм кој ја прикажува промената на спектарот

⁹Англ. *aliasing*, од зборот *alias* што значи лажно име, односно лажно претставување.



Сл. 2.5: Преклопување на спектрите при ниска фреквенција на земање на примероци $f_s < 2f_m$.

на сигналот долж времето.

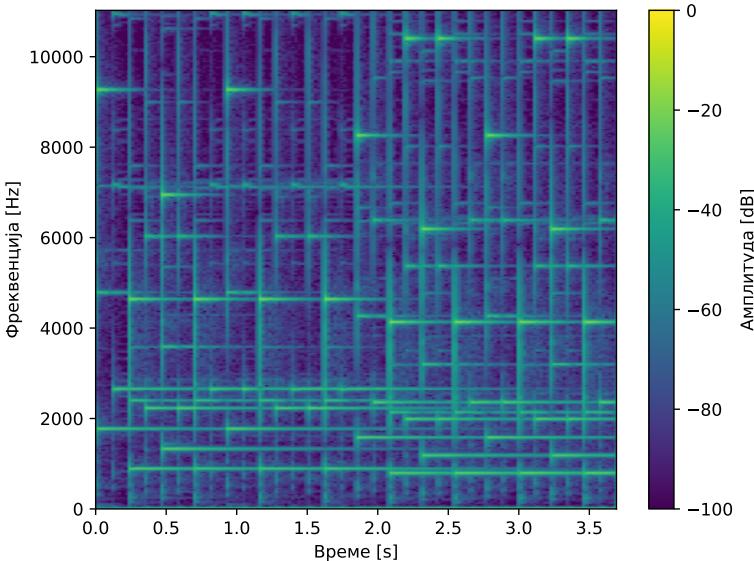
```
import os
import numpy as np
from scipy.io import wavfile
from scipy import signal as sig

import da

fs, wav = wavfile.read('audio/zvona.wav')
wav = wav / 2**15
os.system('play audio/zvona.wav')
n_win = 2048
da.get_spectrogram(wav, fs, n_win=n_win, plot=True)
```

Добиениот спектограм е прикажан на Сл. 2.6. Во него времето е прикажано на x -оската, додека фреквенциската содржина на сигналот е прикажана на y -оската. Интензитетот на спектралните компоненти во dB е прикажан преку мапата на бои. Може да видиме дека секој удар на свонците има изразен основен хармоник, претставен со најниските хоризонтални линии, како и низа на високи хармоници, претставени со повисоките паралелни хоризонтални линии.

Следно сакаме да го потсемплираме овој сигнал $5\times$, односно да му ја намалим фреквенцијата на земање на примероци од 44.100 Hz на 8.820 Hz, без најпрвин да му го ограничиме спектарот на половина од новата фреквенција на земање на примероци, односно на 4.410 Hz. Со тоа, очекуваме хармониците кои се наоѓаат во спектарот над 4.410 Hz во оригиналниот сигнал да се преклопат и да се појават во ниските фреквенции на потсемплираниот сигнал.



Сл. 2.6: Спектрограм на аудиосигналот `zvona.wav`.

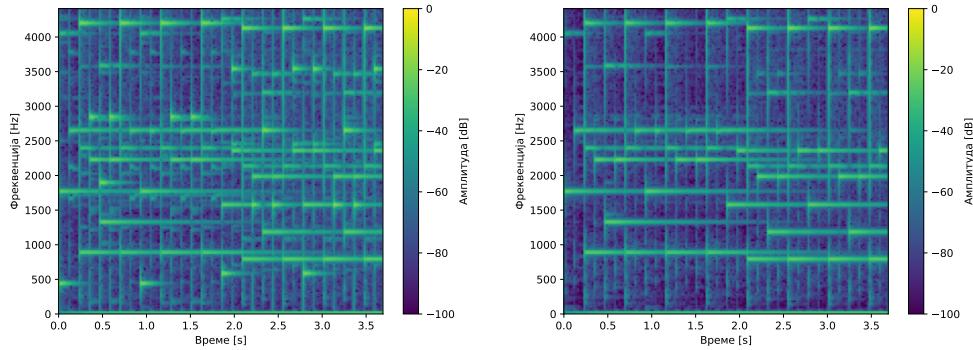
```
und = 5
fs_und = int(fs / und) # 8820 Hz
wav_und = wav[:und]
da.get_spectrogram(
    wav_und, fs_und, n_win=int(n_win / und), plot=True
)

wav_int16 = np.int16(wav_und * 2**15)
wavfile.write('audio/zvona_und.wav', fs_und, wav_und)
os.system('play audio/zvona_und.wav')
```

За споредба, да генерираме уште еден сигнал кој ќе го потсемплираме со новата фреквенција на земање на примероци, но овој пат претходно ќе го ограничиме неговиот спектар на половина од нејзината вредност со употреба на нископропусен филтер.

```
order = 9
fg = fs_und / 2 # 4410 Hz
b, a = sig.iirfilter(
    order, fg, btype='lowpass', ftype='butter', fs=fs
)
wav_filt = sig.lfilter(b, a, wav)
wav_filt = wav_filtered[:und]

da.get_spectrogram(
    wav_filt, fs_und, n_win=int(n_win / und), plot=True
)
wav_int16 = np.int16(wav_filt * 2**15)
```



Сл. 2.7: Спектрограм на добиените потсемплирани аудиосигнали без (лево) и со филтрирање (десно).

```
wavfile.write('audio/zvona_filt_und.wav', fs_und, wav_filt)
os.system('play audio/zvona_filt_und.wav')
```

Можеме јасно да ја слушнеме разликата во двета сигнала! Најизразената разлика е во тоа што во сигналот со преклопувањето се појавуваат нови „фантомски“ музички тонови со ниска фреквенција кои не постојат во оригиналниот сигнал. Истите се јасно воочливи при споредба на спектрограмите на добиените сигнали на Сл. 2.7.

Фреквенции на земање примероци кај дигиталните звучни сигнали

Земањето примероци кај дигиталните аудиосигнали најчесто се врши со следните фреквенции:

- **8.000 Hz** – се употребува во телефонската комуникација и е доволна за пренесување на говор со прифатлива разбираливост,
- **11.025 Hz** и **22.050 Hz** – четвртина и половина од фреквенцијата на земање примероци кај аудио-CD стандардот 44.100 Hz, се употребуваат за зачувување на дигитален звук со понизок квалитет,
- **16.000 Hz** – доволна за да го опфати поголемиот 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,
- **24.000 Hz** – го опфаќа речиси целиот спектар на говорниот сигнал и се употребува кај системите за синтеза на говор од текст (TTS)¹¹ за да се добие квалитетна синтеза,

¹⁰Англ. *Automatic Speech Recognition*.

¹¹Англ. *Text-to-Speech*.

- **32.000 Hz** – се употребува во miniDV¹² стандардот за дигитални видеосигнали што го користат дигиталните видеокамери, во DAT¹³ стандардот во модовите за зголемено траење¹⁴, како и во германското дигитално сателитско радио,
- **44.100 Hz** – фреквенција на земање примероци кај аудио-CD-то, исто така најчесто се употребува кај MPEG кодираниот звук, на пр. кај mp3 стандардот,
- **48.000 Hz** – во употреба кај miniDV, дигиталната Телевизија, DVD, DAT, филмови и професионална аудиоопрема,
- **96.000 Hz** или **192.000 Hz** – во употреба кај DVD стандардот, во Blu-ray дисковите и во HD-DVD,
- **2,8224 MHz** – се употребува во SACD.¹⁵

§ Дополнително. Аудио-CD стандардот употребува фреквенција на земање примероци од 44,1 kHz што соодветствува на максимална фреквенција на спектарот на сигналот од 22,05 kHz. Ова е поголем опсег од слушниот опсег на човекот. Таа е избрана поради тоа што кога се воведувал аудио-CD стандардот во 1980-та, единствениот систем кој нудел фреквенциски опсег на записот доволно голем за да се зачувува звукот во дигитален формат бил VHS системот за видеокасети.

За оваа намена, биле направени PCM-адаптери со кои аналогните звучни сигнали се дигитализирале, а потоа се претворале во аналоген видеосигнал кој се снимал на видеокасета. Со помош на овие уреди, можеле да се запишат по 3 примероци од секој звучен канал во една хоризонтална видеолинија. Бидејќи PAL системот има 294 линии на фреквенција од 50 Hz, следува дека брзината на земање примероци може да изнесува $294 \times 50 \times 3 = 44.100$ примероци/секунда.

Натсемплирање

Директниот начин на кој може да се изведе земањето примероци со фреквенција од на пример 44,1 kHz, како кај аудио-CD-то, 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 нископропусен (LP) филтер низ кој најпрвин минува сигналот во

¹²Англ. *Mini Digital Video*.

¹³Англ. *Digital Audio Tape*.

¹⁴Англ. *long play*.

¹⁵*Super Audio CD* е стандард развиен од Sony и Philips, кој употребува 1-битна сигма-делта квантизација. Овој начин на дигитализација не нуди предности над стандардниот кој е во општа примена.

A/D конверторот, да биде со многу стрмна амплитудна карактеристика веднаш над 20-от kHz.

Овој филтер треба во многу тесен фреквенциски опсег 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 пострмо. Во практиката, ова подразбира употреба на аналоген филтер од 8-ми или 10-ти ред, што претставува скапо решение што не се препорачува, поради потребата за прецизна нагоденост на составните аналогни компоненти.

Постои уште еден поекономичен начин да се реализира земањето примероци, а тоа е со употребата на [натсемплирање](#)¹⁶. Натсемплирањето 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 дигитален домен каде што нископропусното филтрирање на сигналот се реализира со нумерички пресметки, па 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 филтрите](#)¹⁷ со конечен и симетричен импулсен одсив. Потоа може да се отфрлат вишокот примероци, со што доаѓаме до целната фреквенција на земање примероци.

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

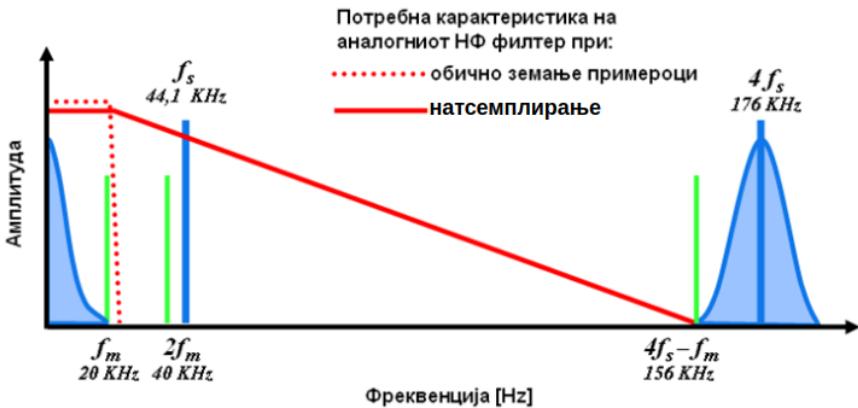
По натсемплирањето, на сигналот се применуваат останатите два чекори на дигитализацијата – квантизација и кодирање. Кога веќе ќе го добијеме сигналот во дигитален домен, сега може да му ја намалиме фреквенцијата на земање на примероци на целната фреквенција на земање примероци $f_s = 44,1$ kHz. За таа цел, дигиталниот сигнал треба да го исфилтрираме со LP филтер со стрмно отсекување на фреквенциите над 22 kHz, кое сега може лесно да се реализира во дигитален домен.

Потоа филтрирањето, можеме да ја намалиме фреквенцијата на земените примероци преку процесот наречен [потсемплирање](#)¹⁸. Ова наједно-

¹⁶Англ. *oversampling*.

¹⁷Англ. *Finite Impulse Response*.

¹⁸Англ. *downsampling*.



Сл. 2.8: Благиот наклон на влезниот LP филтер овозможен со натсемплирање во споредба со потребната карактеристика на филтерот без него.

ставно 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 преку нивно усреднување.

▷ **Пример** За да ги илустрираме придобивките од натсемплирањето ќе земеме натсемплирање од $4 \times$ во однос на аудио-CD стандардот, прикажано на Сл. 2.8. Тогаш, фреквенцијата на натсемплирање f'_s ќе биде 4 пати поголема од номиналната (целната) f_s од 44,1 kHz и ќе изнесува:

$$f'_s = 4f_s = 4 \cdot 44,1 = 176,4 \text{ kHz.} \quad (2.2)$$

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

$$f'_m = \frac{f'_s}{2} = \frac{176,4}{2} = 88,2 \text{ kHz.} \quad (2.3)$$

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 од 20 kHz и да трае сè до 88 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 преодниот фреквенциски опсег на филтерот може да изнесува 120 kHz споредено со 4,1 kHz во случајот кога не се користи натсемплирање. Ова во голема мера ја поедноставува изградбата на влезниот аналоген филтер.

§ **Дополнително.** Процесот на натсемплирање наоѓа примена и во реконструкцијата на аналогниот сигнал од неговата дигитална претстава. Каде D/A конверторите натсемплирањето се изведува со додавање на екстра нулти примероци на секој постоечки примерок. Притоа нивната амплитуда најчесто се интерполира помеѓу вредностите на постоечките примероци или пак математички се моделира, во зависност од изведбата на D/A конверторот.

Овој процес е фундаментално различен од натсемплирањето при дискретизацијата на влезниот сигнал. Имено тој не внесува додатна информација во дигиталниот сигнал. Сепак, како и претходно, натсемплирањето ја олеснува изградбата на излезниот аналоген LP филтер кај D/A конверторите, преку зголемувањето на најниската фреквенција на првата слика на звучниот спектар $f_s - f_m$.

Цитер

Цитерот претставува отстапување во временските моменти во кои се прави земањето примероци поради варијациите во временската база на уредот за дигитализација. Така, A/D конверторот со фреквенција на земање примероци од 44,1 kHz, не е идеален, па не секогаш ги зема примероците точно секој 44.100-ти дел од секундата. Тој може да ги земе примероците малку порано или подоцна со што зачуваната вредност на влезниот сигнал не сосема соодветствува на вредноста што тој ја имал во точниот временски момент. Ефектите на цитерот на временска база стануваат понагласени при присуство на високи фреквенции и големи амплитуди кај влезниот сигнал.

2.2 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, односно **кодира** со ограничен број на битови, наречени **дигитален збор**, следува дека и бројот на амплитуди, односно множеството амплитуди што можат да се запишат е конечно.

Така, ако бројот на битови во дигиталниот збор е N , тогаш, вкупниот број на амплитуди што можат да се запишат со него е 2^N . Поради тоа,

континуираните вредности на влезните амплитуди мора да се заокружат или да се мапираат во најблиските вредности до нив во рамките на ова множество. Овој процес се нарекува **квантизација**, а електронскиот уред со кој таа се изведува **квантизер**¹⁹. Бројот на битови што квантизерот може да го користи се нарекува уште и **негова резолуција**.

Квантизацијата се одвива на тој начин што квантизерот врши споредба на амплитудата на влезниот сигнал со множеството дискретни нивоа кои тој може внатрешно да ги генерира, наречени **квантизациски нивоа**. Разликата помеѓу две соседни квантизациски нивоа се нарекува **чекор на квантизација** q . Постојат два главни пристапи за квантизација: паралелна и секвенцијална квантизација.

Паралелна квантизација – кога влезниот сигнал се споредува паралелно со низа на внатрешно генериирани напонски нивоа кои одговараат на квантизациските нивоа на квантизерот. Овие нивоа вообичаено се генериирани со повеќенировски отпорнички делител од референтниот напон на квантизерот V_{REF} . Поради својата брзина на работа овие квантизери се нарекуваат и **флеш**²⁰ квантизери.

Бидејќи во својата конструкција изискуваат ист број на компаратори, односно 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 квантизација – 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 – **MSB**²¹, до најголемата прецизност што тој може да ја постигне, која е еднаква на чекорот на квантизација $q = V_{REF}/2^N$ и се кодира со битот со најмала тежина **LSB**²².

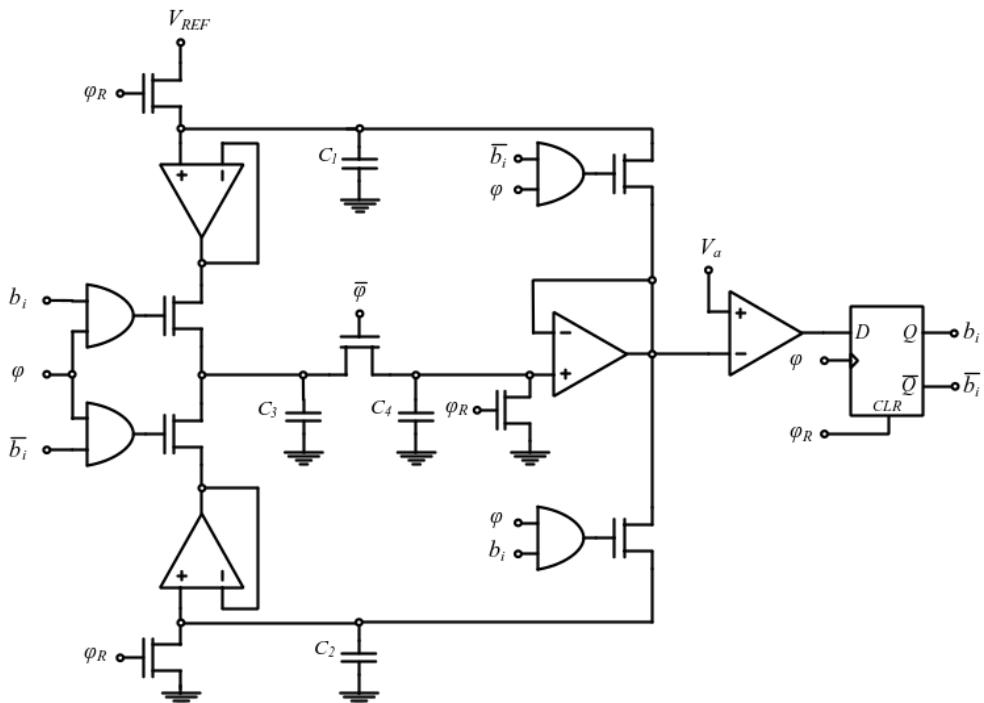
Овој процес е воден од внатрешната логика на квантизерот. Важно е брзината со која се доаѓа до конечниот коден збор да е поголема од брзината со која доаѓаат примероците од влезниот сигнал. Едно коло за секвенцијална квантизација е прикажано на Сл. 2.9.

¹⁹ Влезниот аналоген филтер, земачот на примероци, квантизерот и на крај кодерот се составни делови на еден А/Д конвертор.

²⁰ Англ. *flash*.

²¹ Англ. *Most Significant Bit*.

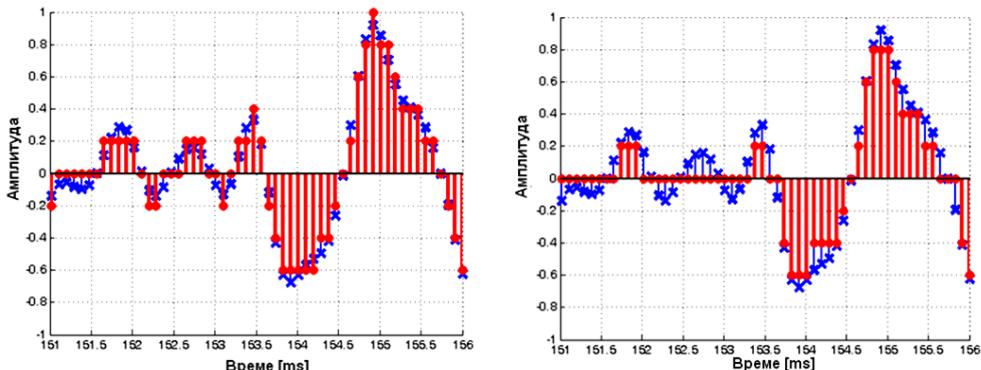
²² Англ. *Least Significant Bit*.



Сл. 2.9: Коло за секвенцијална квантизација на аналоген влезен напон.

§ Дополнително. Прикажаното коло за секвенцијална квантизација содржи еден операциски 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 период 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 период по ресетирањето C_3 се полни на V_{REF} , па во негативниот дел од оваа период, 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 MSB бит од конверзијата 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 квантизаторот е ограничена.



Сл. 2.10: Квантизација (првено) на дискретизираниот сигнал (сино) со заокружување (лево) и со отсекување (десно).

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

Постојат два главни типа на квантизација што се разликуваат според начинот на кој го ловат влезниот сигнал. Тие се **квантизација со заокружување** и **квантизација со отсекување**. Кај првата излезот на квантизаторот го дава квантизациското ниво најблиско до амплитудата на влезниот сигнал, додека кај втората тоа е квантизациското ниво веднаш под амплитудата на влезниот сигнал, како што е случај кај колото прикажано на Сл. 2.9. Типот на квантизација зависи од електронската реализација и од механизмот на работа на квантизаторот. Разликите помеѓу двата типа се илустрирани на Сл. 2.10.

Квантизација на дигиталниот звук

При дигитализација на аудиосигналите се употребуваат следните резолуции на квантизација:

- 1 бит – кај SACD кој користи натсемплирање и техники за обликување на шумот на квантизација за постигнување на голем однос сигнал–шум и при 1-битна резолуција,
- 4 бита – се употребува ретко во линеарна импулсно кодна модулација (PCM)²³ при адаптивна квантизација,
- 8 бита – во телефонските системи за кодирање на говорот со примена на нелинеарна квантизација, стандардна резолуција на старите звучни карти во ерата на флопи-дискетите,²⁴

²³Англ. *Pulse Code Modulation*.

²⁴Денес се јавува во музичко движење засновано на 8-битен звучен израз.

- 12 бита – во употреба во 1980-тите во дигиталните уреди за ефекти употребувани во музицирањето, на пр. за електричните гитари,
- 16 бита – во аудио-CD стандардот, 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 ADAT повеќе-канални машини и кај некои звучни картички,
- 24 бита – новиот стандард за висока дефиниција, имплементиран во 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

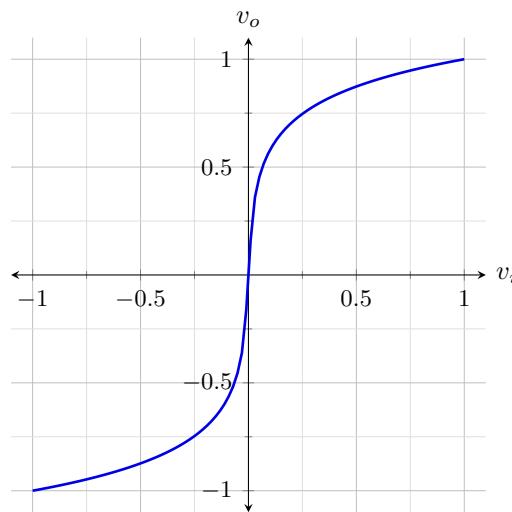
Под **кодирање** се подразбира начинот на кој дискретната амплитуда на квантизерот ќе биде запишана во битови. Наједноставниот начин на кодирање на амплитудите на влезниот сигнал е со нивно директно претворање во бинарни (кодни, дигитални) зборови. Притоа паровите амплитуда-коден збор се линеарно распоредени, 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, каде која амплитудите со најголема веројатност на појавување се квантизираат со поголема резолуција од оние кои поретко се појавуваат. Ова се реализира на тој начин што за највисоките влезни амплитуди се ставени на располагање 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 квантација со крива за компресија, наречена и *A*-крива.²⁵ *A*-кривата е дефинирана со G.711 стандардот на Меѓународната унија за телекомуникации и е описана со равенството:

$$f(x) = \text{sgn}(x) \cdot \begin{cases} \frac{A|x|}{1+\ln(A)}, & |x| < \frac{1}{A} \\ \frac{1+\ln(A|x|)}{1+\ln(A)}, & \frac{1}{A} \leq |x| \leq 1 \end{cases} \quad (2.4)$$

²⁵Wikipedia: A-law algorithm https://en.wikipedia.org/wiki/A-law_algorithm



Сл. 2.11: Графичка претстава на преносната функција добиена со A -кривата за нелинеарна квантизација на говор.

Во Европа е дефинирано $A = 87,6$. На Сл. 2.11 е дадена преносната функција на оваа крива. Како што може да се забележи, по компресијата поголем број на квантизацијски 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 од таму и нужноста за нивна поточна квантизација.

Исто така, постојат и методи на квантизација кои ја кодираат [разликата](#) помеѓу две последователни амплитуди на влезниот сигнал – таканаречена [адаптивна квантизација](#). Тие се употребуваат во апликации каде што големината на дигиталниот поток 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](#).

Ако се задоволени следниве два услови:

1. влезниот сигнал да може да се смета за [случаен процес](#) и
2. тој да има [многу поголема амплитудна динамика](#) од чекорот на квантизација q , или еквивалентно чекорот на квантизација да е доволно мал,

тогаш, шумот на квантизација може да го моделираме како [бел Гаусов](#)

шум, односно тој ќе биде слукаен 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 при квантизација со отсекување.

Под овие услови, моќноста на шумот на квантизација ќе биде рамномерно распределена долж целиот спектар и со константна амплитуда наречена **ниво на шумот на квантизација**. Тие речиси секогаш се исполнети во пракса, особено за сигнали кои имаат мал коефициент на автокорелација, како што се говорот и музиката, како и кога квантizerот е со доволно голема резолуција. Тие не се исполнети само во многу специјални случаи како што се константни сигнали, простопериодични сигнали синхронизирани со фреквенцијата на земање примероци и сигнали со многу мали амплитуди.

За да го прикажеме шумот на квантизација, ќе вчитаме еден аудиосигнал од гитара и ќе го квантизираме со различни резолуции на квантизација. Всушност аудиосигналот ќе го реквантизираме, бидејќи тој веќе е квантизиран! Притоа, повторно ќе се послужиме со методите за практично процесирање звук описаны во понатамошните глави од оваа книга.

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

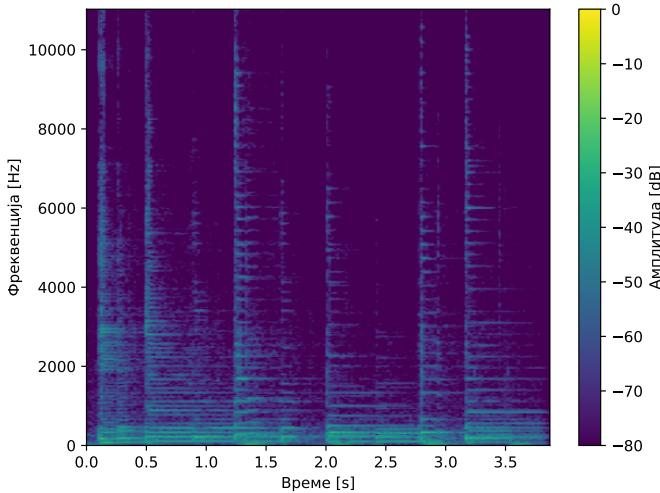
import da

fs, wav = wavfile.read('audio/Solzi.wav')
wav = wav / 2**15 # normalize to -1, 1
os.system(f'play audio/Solzi.wav')
da.get_spectrogram(wav, fs, plot=True)
```

Спектрограмот на вчитаниот аудиосигнал е прикажан на Сл. 2.12. Следно, ќе дефинираме функција за реквантизација чии влезни параметри ќе бидат аудиосигналот и целната резолуција на квантизација.

```
def quantify(wav, n_bits):
    '''Quantify wav to n_bits resolution.'''
    wav_q = wav * (2**n_bits - 1)
    wav_q = np.round(wav_q)
    wav_q = wav_q / (2**n_bits - 1)
    return wav_q
```

Сега ќе ја искористиме оваа функција за да го реквантизираме аудиосигналот со 6-битна резолуција. Притоа ќе ги прикажеме временските облици на оригиналниот и реквантанизираниот сигнал, како и шумот на квантизација, спектрограмот на шумот на квантизација, а и ќе го чуеме



Сл. 2.12: Спектрограм на аудиосигналот Solzi.wav .

реквантанизираниот сигнал.

```
n_bits = 6
wav_q = quantify(wav, n_bits)
noise_q = wav - wav_q

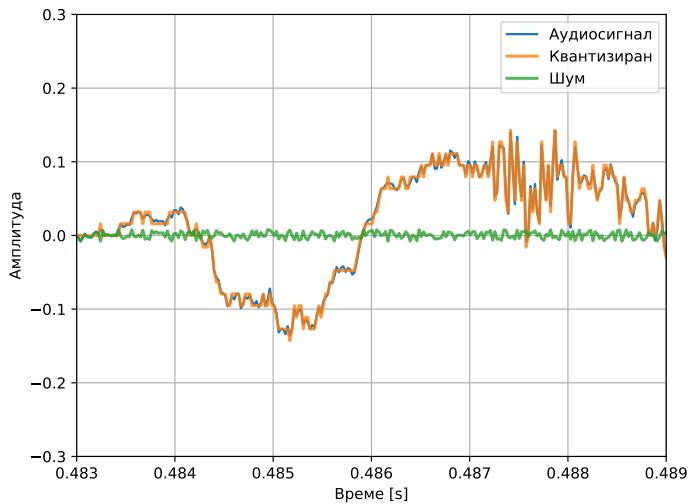
t = np.arange(0, wav.size) / fs
plt.figure()
plt.plot(t, wav)
plt.plot(t, wav_q, lw=2, alpha=0.8)
plt.plot(t, noise_q, lw=2, alpha=0.8)
plt.grid()
plt.legend(['Аудиосигнал', 'Квантизиран', 'Шум'])

da.get_spectrogram(noise_q, fs, plot=True)

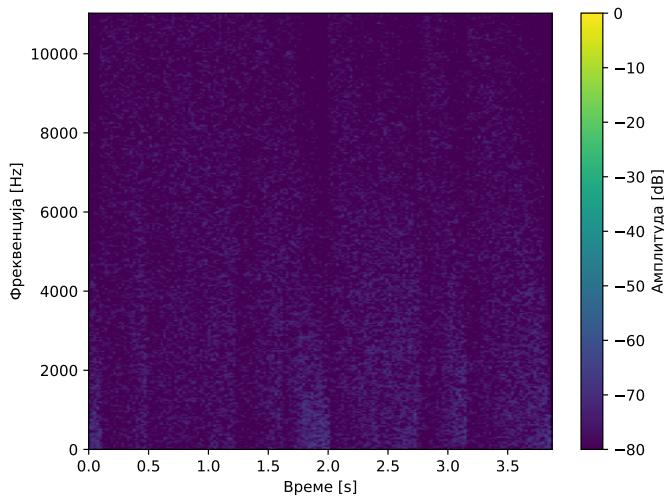
wav_16bit = np.int16(wav_q * 2**15)
filename = f"audio/Solzi_{n_bits}bit.wav"
wavfile.write(filename, fs, wav_16bit)
os.system(f'play {filename}'')
```

Можеме да слушнеме дека реквантизацијата навистина внесла шум во аудиосигналот. Временскиот облик на шумот на квантизација може да се види на Сл. 2.13, а неговиот спектрограм на Сл. 2.14. Може да забележиме дека шумот на квантизација има навистина мала амплитуда а спектрално наликува на бел шум.

Што ќе се случи ако употребиме премала резолуција за квантизирање на аудиосигналите? Ќе го илустрираме ова со реквантизација на истиот сигнал на 3 бита:



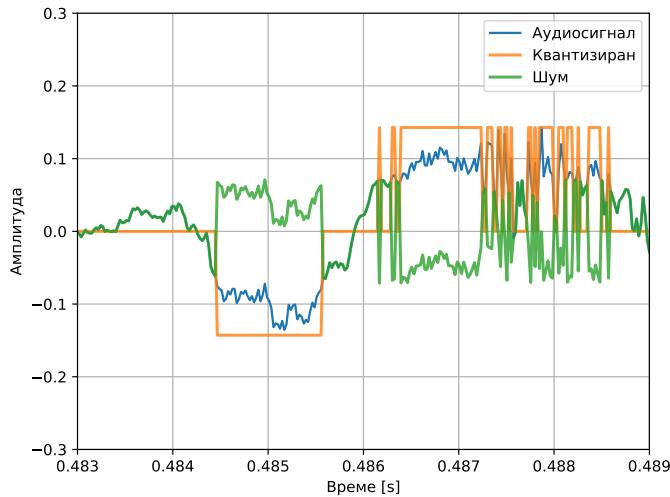
Сл. 2.13: Временски облик на оригиналниот и квантанизираниот сигнал и шумот на квантација при 6-битна резолуција.



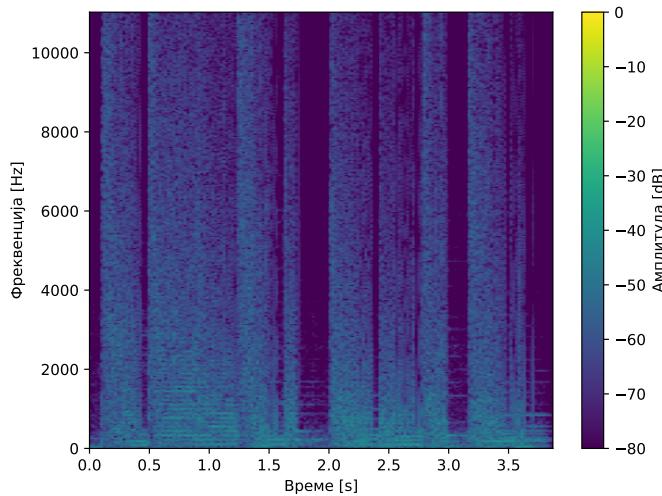
Сл. 2.14: Спектрограм на шумот на квантација при 6-битна резолуција.

```
n_bits = 3
wav_q = quantify(wav, n_bits)
noise_q = wav - wav_q
```

Може да слушнеме дека реквантанизираниот сигнал е многу изобличен, како што може да се види и од неговиот временски облик прикажан на Сл. 2.15. Дополнително, гледајќи го временскиот облик на шумот на квантација, како и неговиот спектрограм на Сл. 2.16, може да заклучиме дека не можеме повеќе да го сметаме за бел шум. Напротив, во неговиот спектрограм се воочливи хоризонтални линии кои се должат на хармониските



Сл. 2.15: Временски облик на оригиналниот и квантизираните сигнал и шумот на квантизација при 3-битна резолуција.



Сл. 2.16: Спектрограм на шумот на квантизација при 3-битна резолуција.

изобличувања во реквантанизираниот сигнал. Ова се должи на тоа што не е задоволен вториот услов – сигналот да има многу поголема амплитудна динамика од чекорот на квантизација q .

Однос сигнал–шум

Односот помеѓу просечната моќност на аналогниот сигнал и моќноста на шумот на квантизација се нарекува **однос сигнал–шум (SNR)**²⁶ на диги-

²⁶Англ. *Signal-to-Noise Ratio*.

Табела 2.1: Варијанса на Гаусовиот процес при моделирање на различни звучни сигнали.

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

талниот сигнал. Неговата вредност може да се добие со помош на равенството:

$$SNR = 10 \log \frac{P_x}{P_N} = 10 \log \frac{\sigma_x^2}{\sigma_q^2} \quad (2.5)$$

каде P_x и P_N се моќноста на аудиосигналот и шумот на квантанизација, 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 условия тогаш густината на веројатност на грешка е рамномерно распределена во интервалот од $-q/2$ до $q/2$, па за σ_q добиваме:

$$\sigma_q^2 = \frac{q^2}{12} \quad (2.6)$$

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

$$\sigma_q^2 = \frac{1}{12 \cdot 2^{2N}} \quad (2.7)$$

па (2.5) станува:

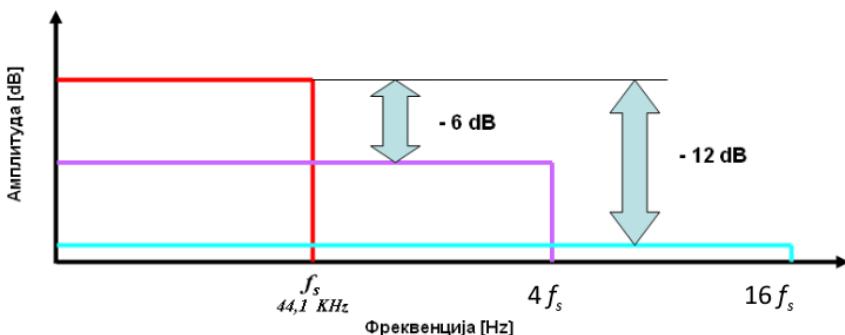
$$SNR = 10 \log \left(\frac{\sigma_x^2}{\frac{1}{12 \cdot 2^{2N}}} \right) = 10 \log(12 \cdot 2^{2N} \sigma_x^2) \quad (2.8)$$

При пресметка на (2.8) можеме влезниот сигнал да го моделираме како Гаусов, при што е важно тој да не ја надмине границата на квантизерот, којашто зедовме дека е 1. Ако варијансата $\sigma_x < 0,25$, 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 2.1.

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:

$$SNR = 6,02n - 1,25 \simeq 6n [\text{dB}] \quad (2.9)$$



Сл. 2.17: Намалувањето на нивото на шумот со употреба на натсемплирање.

Во праксата речиси секогаш се користи поедноставената форма, поради тоа што го занемарува константниот член кој зависи од варијантата на звучниот сигнал, како и поради нејзината 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 со натсемплирање

Ако влезниот аналоген сигнал е случаен процес, па шумот на квантизација е бел и некорелиран, тој ќе биде рамномерно распределен долж целиот фреквенциски спектар на сигналот. Што се случува ако употребиме $4 \times$ натсемплирање – 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 мокност ќе биде 4 пати помала. Тоа соодветствува на ниво на шум помало за $10 \log^{1/4} = -6$ dB. Со други зборови $4 \times$ натсемплирање има ист ефект врз односот сигнал–шум како и зголемувањето на должината на дигиталниот збор за 1 бит. Слично, $16 \times$ натсемплирање соодветствува на зголемена резолуција на A/D конверторот од 2 бита, како што е прикажано на Сл. 2.17.²⁷

Ова може да се каже и на следниов начин: 16–битен A/D конвертор со

²⁷Ова може да се протолкува како поништување на шумот при усреднување на земените вишок примероци. Сличен процес се користи во астрономската фотографија каде што 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.

брзина на земање примероци од 44,1 kHz нуди ист однос сигнал–шум како 15-битен A/D конвертор со $2\times$ натсемплирање. 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 положените 16 или 24 битни A/D конвертори. Крајноста е реализација на A/D конвертори кои работат со 1 бит по примерок, како кај SACD стандардот, кој дополнително употребува и техники за обликување на шумот, дискутирани во продолжение, за уште поголеми придобивки во SNR.

§ **Дополнително.** натсемплирањето нуди придобивки 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 притоа 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 која соодветствува на најмалку значајниот бит, односно чекорот на квантизација, толку повеќе се зголемува корелираноста на шумот на квантизација со самиот сигнал.

Како последица на ова, шумот на квантизација не може повеќе 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.

Дитерот²⁸ е процес со којшто може да се елиминира појавата на дисторзија во сигналот поради шумот од квантизација.²⁹ Во него, на влезниот сигнал пред дигитализацијата му се додава мало ниво на аналоген шум. Тој е воведен од Лоренс Робертс³⁰, еден од основачите на интернетот, во својата магистерска теза на MIT во 1961.

Случајниот карактер на додадениот шум внесува случајност и во однесувањето на квантизерот при ниски влезни нивоа на сигналот – ефективно внесувајќи несигурност во неговата работа. На тој начин, грешката од

²⁸Македонскиот термин за дитер би бил „треперење“ од англиското *to dither*, чие потекло е од старо-англискиот збор *didderen* кој значи тресење како од студ.

²⁹Wikipedia: Dither <https://en.wikipedia.org/wiki/Dither>

³⁰Wikipedia: Lawrence Roberts https://en.wikipedia.org/wiki/Lawrence_Roberts_%28scientist%29

квантизација се декорелира од сигналот. Ова доаѓа од таму што сега, излезните нивоа од квантизерот веќе не зависат само од влезниот сигнал, туку и од некорелираниот случаен процес внесен како шум.

Концептот зад дитерот, иако интуитивно изгледа нелогичен, е многу ефикасен. Зад неговата ефикасност, стои и фактот што човековото уво е помалку чувствително на широкопојасен бел шум, отколку на хармониски изобличувања во спектарот на сигналот какви што се јавуваат при корелираност на шумот на квантизација. Уште повеќе, бидејќи увото може да открие звук во опсегот на средните фреквенции 10 до 12 dB под нивото на шумот, со дитерот може да се постигне 18-битна резолуција кај еден 16-битен запис! Така, при правилен избор на сигналот за дитер возможно е да се запишат сигнали и 110 dB под максималното дигитално ниво, иако 16-битната резолуција теориски нуди динамички опсег од 96 dB.

Дитерот е во редовна употреба при намалувањето на резолуцијата на еден дигитален звучен запис, нешто што редовно се прави 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 внесување на одредено ниво на шум пред конверзијата се овозможува зачувување на дел од вишокот информација што ја содржи оригиналниот запис.

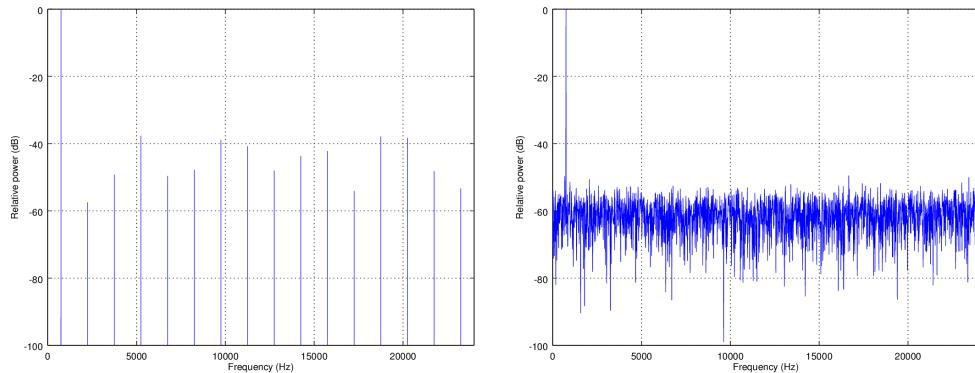
Алтернативниот пристап би бил да се направи отфрлање на информацијата зачувана во 24-битна резолуција со просто испуштање на долните 8 битови, процес кој се нарекува [скастрување](#)³¹. Но, овој пристап неминовно ќе доведе до појава на корелиран шум на квантизација во вид на дисторзија во конечниот аудиозапис.

Пример за употреба на дитер е прикажан на Сл. 2.18.³² На левиот графикон е прикажан простопериодичен тон со f_0 од 750 Hz, со f_s од 48 kHz, квантизиран со 4-битна резолуција без употреба на дитер. Бидејќи фреквенцијата на земање примероци на синусот е мултипл од неговата основна фреквенција ($64 \times$), шумот на квантизација е корелиран со сигналот и самиот е периодичен. Поради тоа спектарот на шумот на квантизација 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 квантизацијата со 4 битови. Може да видиме дека и покрај тоа што нивото на шум е поголемо, шумот на квантизација е сега бел шум и немаме појава на хармониски изобличувања. Односот сигнал–шум постигнат со примена на дитерот е 60 dB.

³¹Англ. *truncation*.

³²Wikipedia: Noise shaping https://en.wikipedia.org/wiki/Noise_shaping



Сл. 2.18: Синусен тон на фреквенција од 750 Hz од кој се земени примероци со мултипл од таа фреквенција (48 kHz) и квантизиран на 4 бита без (лево) и со употреба на дитер (десно).³³

§ 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 на земјата. Се покажало дека причината за ова е постојаното тресење на авионот кое овозможило полесно движење на деловите од овие компјутери што биле замастени и слепени. Подоцна започнале да вградуваат минијатурни мотори – вибратори наречени дитери вградувале во сите механички компјутери.

§ Дополнително. Ефикасноста на дитерот може добро да се илустрира во неговата примена при намалување на битската длабочина кај сликите. Така, на Сл. 2.19 е прикажана реквантанизација на една 8-битна сива слика во 1-битна црно-бела слика без и со примена на дитер.

За да ги илустрираме придобивките од употребата на дитер при реквантанизација на аудиосигналите, повторно ќе го искористиме аудиосигналот Solzi.wav кој го реквантизирајме претходно. Овојпат, ќе употребиме дитер при реквантанизација на резолуција од 3 бита.

```
dither = np.random.rand(wav.size) # бел шум во опсег (0,1)
dither = dither * 2 - 1
```

³³By Andrew D'Addesio - Own work, CC0, <https://commons.wikimedia.org/w/index.php?curid=44975277> и <https://commons.wikimedia.org/w/index.php?curid=44975276>

³⁴By 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>



Сл. 2.19: Сива слика со 256 нивоа по пиксел (лево), реквантанизирана на две нивоа (црно и бело) без (средина) и со употреба на дитер (десно).³⁴

```

q = 2 / 2**n_bits
dither = dither * q / 2

wav_dither = wav + dither
wav_dither_q = quantify(wav_dither, n_bits)

wav_16bit = np.int16(wav_dither_q * 2**15)
wavfile.write(f'audio/Solzi_{n_bits}bit_dither.wav', fs, wav_16bit)
os.system(f'play audio/Solzi_{n_bits}bit_dither.wav')

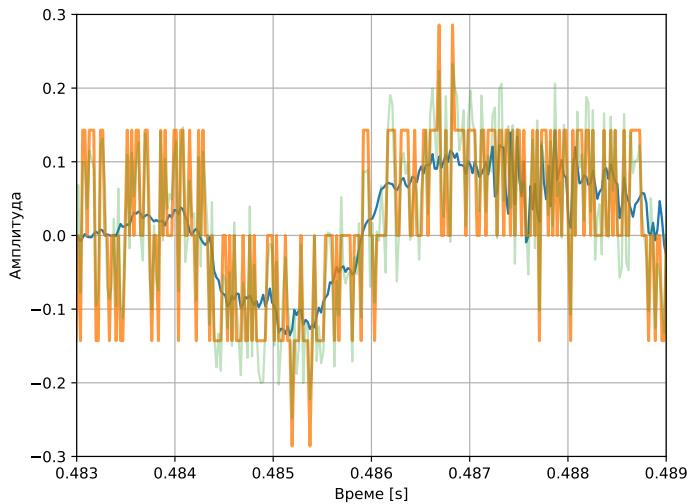
plt.figure()
t = np.arange(0, wav.size) / fs
plt.plot(t, wav)
plt.plot(t, wav_dither_q, lw=2, alpha=0.8)
plt.plot(t, wav_dither, alpha=0.3)
plt.grid()
plt.xlabel('Време [s]')
plt.ylabel('Амплитуда')
plt.tight_layout()
plt.axis([0.483, 0.489, -0.3, 0.3])

da.get_spectrogram(wav_dither_q, fs, plot=True)

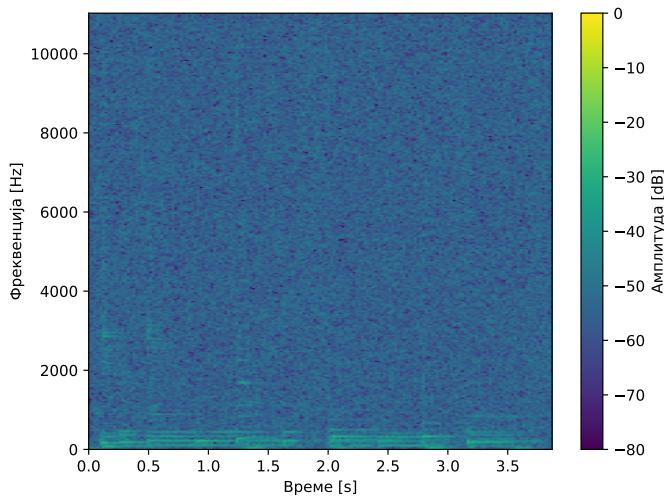
```

Овојпат, и покрај тоа што аудиосигналот има многу шум, сепак можеме јасно да го чуеме оригиналниот звук на гитарата! Ефектите на додавање на дитер може да ги видиме во временскиот облик на квантизираните сигнал прикажан на Сл. 2.20. Може да се види дека начинот на кој функционира додавањето на дитер е со претставување на промените на амплитудата на сигналот преку **импулсна модулација (PWM)**³⁵ на квантизираните сигнал. Конечно, во спектрограмот прикажан на Сл. 2.21, низ шумот навистина се назира спектралната содржина на оригиналниот сигнал.

³⁵Англ. *Pulse Width Modulation*.



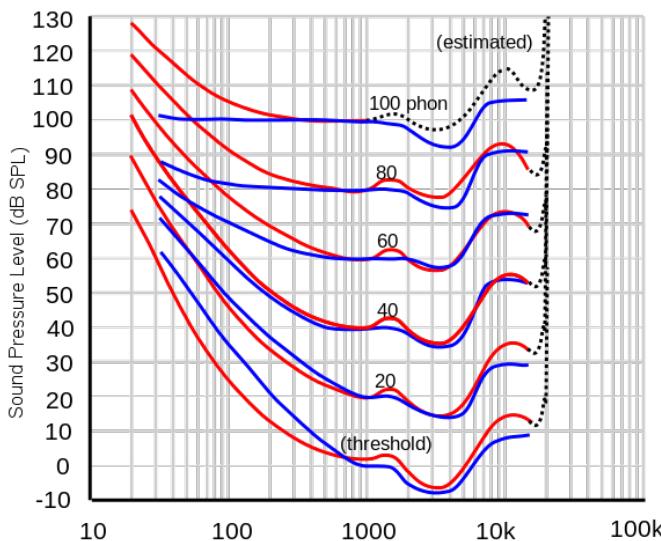
Сл. 2.20: Временски облик на оригиналниот и сигналот со дитер квантизиран на 3-битна резолуција.



Сл. 2.21: Спектрограм на сигналот со дитер квантизиран на 3-битна резолуција.

Обликување на шумот

Дитетрот 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. 2.22. Изофонските криви ја прикажуваат субјективната јачина 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 фрек-



Сл. 2.22: Криви на еднаква гласност, или изофонски криви, измерени според Флечер и Мунсон (сино) и стандардизирани според ISO стандардот (црвено).³⁷

венцички нелинеарната осетливост на сетилото за слух на човекот.³⁶

Со техниката на обликување на шумот се прераспределува спектралната густина на моќноста на шумот на дитерот со тоа што поголем нејзин дел се истиснува надвор од фреквенцијскиот опсег во кој увото е најосетливо. Притоа севкупната моќност на шумот останува иста. На овој начин, можеме во еден 16-битен сигнал да постигнеме однос сигнал–шум од 120 dB, а некои алгоритми овозможуваат тој на средни фреквенции да достигне дури 150 dB!

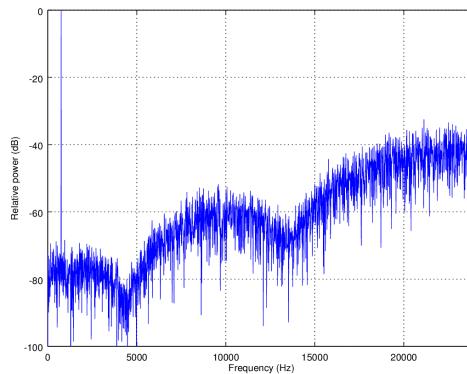
Обликувањето на шумот е особено згодно при употреба на натсемплирање, каде што шумот може да се истисне надвор од слушното подрачје над 20-от kHz. Како и дитерот, и обликувањето на шумот се употребува најмногу при конверзијата на дигиталниот сигнал од поголема во помала битска резолуција.

Пример за употреба на обликување на шумот со дитер е прикажана на Сл. 2.23, каде што истиот синусен тон кој го видовме на Сл. 2.18 е квантзиран 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

³⁶ Повеќе за нив може да видите во делот Психоакустика во предметот Електроакустика чии материјали се поставени на Гитлаб <https://gitlab.com/feeit-freecourseware/electroacoustics>.

³⁷"Lindos4" by Lindosland <https://commons.wikimedia.org/wiki/File:Lindos4.svg#media/File:Lindos4.svg>



Сл. 2.23: Синусен тон на фреквенција од 750 Hz квантизиран на 4 бита со употреба на дитер со примена на техниките за обликување на шумот.³⁸

што увото е најосетливо. Да потсетиме дека без примената на овие техники, според (2.9) SNR за 4 битна квантација изнесува само 24 dB!

◀ Пример Када 64× натсемплирање во SACD стандардот, фреквенцијата на земање примероци изнесува 2,8 MHz, па шумот може да се распределат 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.

Ајде да го искористиме обликувањето на шумот за подобрување на квантацијата на аудиосигналот со дитер од претходниот пример. За таа цел ќе го исфилтрираме шумот на дитер со високопропусен филтер и повторно ќе го скалираме, со што ќе ја задржиме неговата потребна амплитуда, но ќе го истиснеме кон повисоките фреквенции каде што сигналот нема изразени компоненти, а и нашето уво е помалку осетливо.

```
order = 1
fg = 20000 # Hz
[b, a] = sig.iirfilter(order, fg, btype='highpass', fs=fs)
f, h = sig.freqz(b, a, fs=fs)

dith_filt = sig.lfilter(b, a, dith)
dith_filt = dith_filt / np.max(np.abs(dith_filt)) * q

wav_dith_filt = wav + dith_filt
wav_dith_filt_q = quantify(wav_dith_filt, n_bits)

wav_16bit = np.int16(wav_dith_filt_q * 2**15)
wavfile.write(f'audio/Solzi_{n_bits}bit_dith_filt.wav', fs, wav_16bit)
```

³⁸By Andrew D'Addesio - Own work, CC0, <https://commons.wikimedia.org/w/index.php?curid=44975275>

```

os.system(f'play audio/Solzi_{n_bits}bit_dith_filt.wav')

t = np.arange(0, wav.size) / fs
plt.figure()
plt.plot(t, wav)
plt.plot(t, wav_dith_filt, alpha=0.3)
plt.plot(t, wav_dith_filt_q, lw=2, alpha=0.8)
plt.grid()

da.get_spectrogram(wav_dith_filt_q + 1e-6, fs, plot=True)

```

Можеме да слушнеме дека обликувањето на шумот го подобрило квантизираните аудиосигнал – сега шумот помалку доминира во сигналот. Ефектите на истиснување на шумот кон повисоките фреквенции може да ги видиме во временскиот облик на квантизираните сигнали на Сл. 2.24 и во подобрувањата воочливи во спектрограмот на Сл. 2.25.

2.3 Формати на дигитален звук

По земањето на примероци, квантизацијата 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)³⁹.

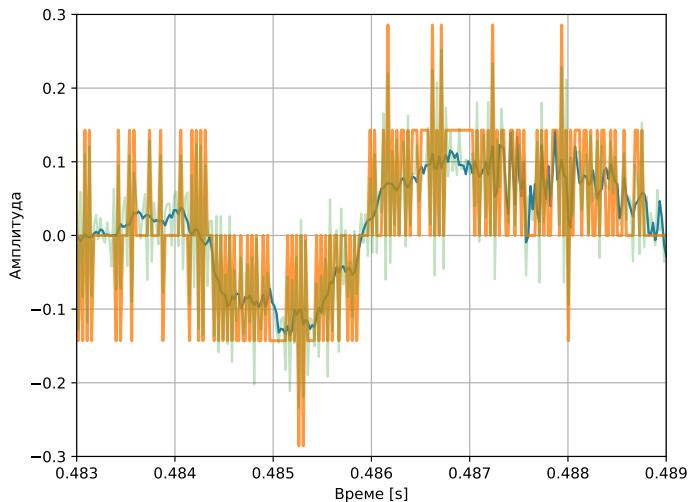
PCM претставува најкавалитетниот, најсигурниот и наједноставниот начин на дигитално запишување на звучните аналогни сигнали и се користи кога не ни е пречка големината на битскиот проток генериран при дигитализација. Поради релативната едноставност и сигурност во однос на другите типови на квантизација 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 записан во вид на постојан поток, дефиниран според Црвената Книга на аудио-CD стандардот. Разликата е во тоа што датотеките содржат воведен дел, заглавје⁴⁰ 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.

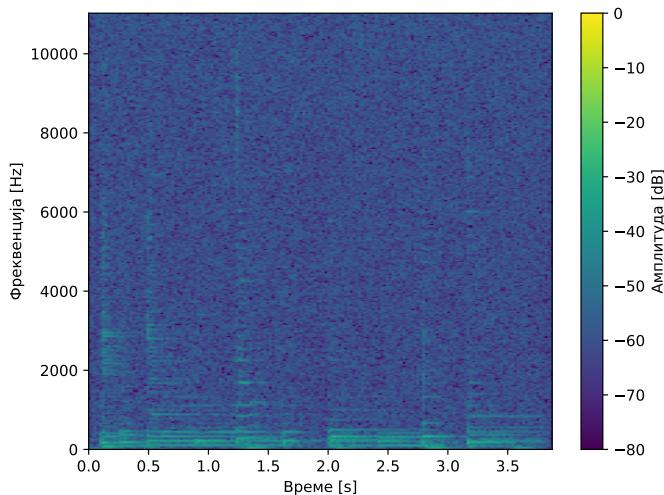
Форматот во кој е зачувано PCM аудиото во компјутерските системи зависи од оперативниот систем. Windows оперативните системи го корис-

³⁹Англ. *Pulse Code Modulation*.

⁴⁰Англ. *header*.



Сл. 2.24: Временски облик на оригиналниот и сигналот со дитер и обликување на шумот квантизиран на 3-битна резолуција.



Сл. 2.25: Спектрограм на сигналот со дитер и обликување на шумот квантизиран на 3-битна резолуција.

тат [WAV⁴¹](#) форматот, развиен од страна на Microsoft за Intel базираните компјутерски системи во 1991, врз база на општиот RIFF⁴² за чување на разни видови на податоци. Компјутерите со Macintosh платформи го употребуваат [AIFF⁴³](#) форматот на датотеки, развиен од Apple во 1988.

Двата паралелни стандарда се базираат на IFF форматот за зачуву-

⁴¹Англ. *Waveform Audio Format*.

⁴²Англ. *Resource Interchange File Format*.

⁴³Англ. *Audio Interchange File Format*.

вање на податоци воведен од Electronic Arts во 1985, со клучна разлика во редоследот на запишување на дигиталните податоци. WAV форматот најпрвин ги запишува 8-те 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*.

Бо WAV форматот се користат два начини на запишување на негативните примероци на аудиосигналот, во зависност од бројот на битови по примерок. Ако станува збор за звук квантизиран со 8 бита по примерок тогаш амплитудните вредности се зачувуваат без бит за знак `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 квантација, негативните вредности на сигналот се зачуваат со употреба на двоен комплемент, кај кој првиот бит од 16-те го дефинира знакот. Во тој случај, најмалата вредност што може да биде запишана е `0x8000`, соодветствува на најнегативната вредност на сигналот $-2^{15} = -32768$, додека најголемата вредност `0x7FFF`, соодветствува на неговата најпозитивна вредност $2^{15} - 1 = 32767$. Така, може да се зачуваат една повеќе негативни амплитуди отколку позитивни, што се должи на тоа што 0та е зачувана како позитивен број со кодот `0x0000`.

Дигиталниот звук често се користи во негова компримирана форма, со што се постигнува намалување на побарувањата за меморија и битски проток при неговото зачувување и пренос. За таа цел се развиени низа техники за компримирање што ќе бидат разгледани во [Главата 7](#).

Глава 3

Вовед во работа со звучни сигнали

Со преминот на звукот во [дигитален домен](#), се отвораат вратите кон примена на различните техники на дигиталната обработка на сигналите. Во оваа глава ќе се запознаеме практично со основите на работата со звукот во дигитален домен.

Поради философијата на отвореност и соработка која е основа на движењето за слободен софтвер, денес сè повеќе инженери и научници ја засноваат својата работа на платформи базирани на слободен софтвер. Во духот на философијата на 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 програмски јазик [Python](#).¹

3.1 Работа со звучни датотеки

Вчитувањето на звучни датотеки 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
```

¹Повеќе за историјата, предностите и продорот на слободниот софтвер во инженерската и научно-истражувачката пракса може да прочитате во [Додатокот А](#). За основите на [GNU/Linux](#) и работа во терминалот видете го [Додатокот Б](#). Конечно, за вовед во екосистемот на Python направен за инженерска и научна работа и за основите на програмирањето во Python погледнете го [Додатокот В](#).

```
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 поместен 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 променливата `wav` ќе има две вектор колони. Од друга страна, 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 податоците содржани во матрицата `wav`.

За да го видиме типот и обликот на променливите во работниот простор може да напишеме:

```
whos
```

Variable	Type	Data/Info
<code>fs</code>	<code>int</code>	<code>44100</code>
<code>np</code>	<code>module</code>	<code><module 'numpy' ... ></code>
<code>plt</code>	<code>module</code>	<code><module 'matplotlib.pyplot' ... ></code>
<code>wav</code>	<code>ndarray</code>	<code>811782x2: ... type 'int16', 3247128 bytes (3 Mb)</code>
<code>wavfile</code>	<code>module</code>	<code><module 'scipy.io.wavfile' ... ></code>

Гледаме дека `wav` има точно две вектор колони секоја со по 811.782 примероци. За да пресметаме со колкаво времетраење е вчитаниот аудиосигнал треба да пресметаме:

$$t = \frac{N}{f_s} \quad (3.1)$$

каде што N е бројот на примероци, а f_s е фреквенцијата на земање на примероците. Во Python ова може едноставно да го пресметаме на следниот начин:

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

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

3.2 Основни операции со аудиосигналите

Сечење на аудиосигналите

Звучниот сигнал сместен вака во матрицата `wav` може да се обработува како и сите други вектори и матрици во Python. За да издвоиме сегмент од аудиосигналот, да речеме првите 4 s, ќе ги задржиме првите $4 \cdot f_s$ примероци од двета канали. Ова може да се направи со сечење² на векторот во Python:

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

Претворање во моно

Постојат неколку начини да го направиме двоканалниот стерео сигнал во едноканален моно сигнал. Наједноставно е да го задржиме само единиот од каналите:

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

или:

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

Сепак, на овој начин целосно ќе ја загубиме информацијата содржана во отстранетиот канал. Ова може да се избегне ако моно каналот го пресметаме преку средна вредност на сумата на левиот и на десниот канал:

$$y[n] = \frac{x[n, 0] + x[n, 1]}{2} \quad (3.2)$$

Во Python ова може да се направи на следниот начин:

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

Но, може да се пресмета и со помош на NumPy функциите за пресметување на сумата `np.sum` и средната вредност `np.mean` на елементите на матрицата:

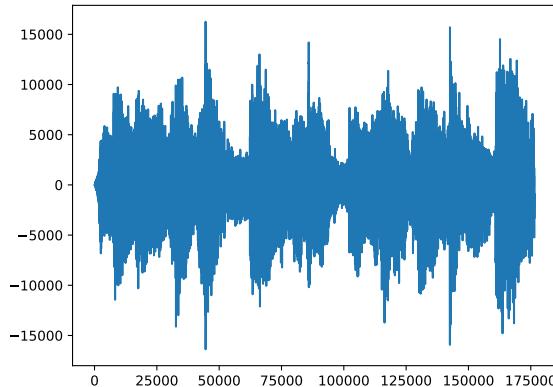
```
wav_mono = np.sum(wav, axis=1) / 2
```

односно:

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

Тука, вториот параметар `axis` кажува по која оска да се изврши пресметката, а со 1 е избрана првата димензија – колони, т.е. по хоризон-

²Англ. *slicing*.



Сл. 3.1: Приказ на звукот `wav_mono` како вектор од примероци од типот `int16`.

тала. Со 0 може да се избере нултата димензија – редици, т.е. по вертикалa, која е и зададена автоматски. Така на пример, ако сакаме да го отстраниме DC нивото од стерео аудиосигналот, што е еквивалентно на средната вредност на сигналот, тоа може да го направиме со:

$$y[n, i] = x[n, i] - \bar{x}[i] \quad i = 0, 1 \quad (3.3)$$

каде што $\bar{x}[i]$ е средната вредност на сигналот во каналот i пресметана како:

$$\bar{x}[i] = \frac{1}{N} \sum_{n=0}^{N-1} x[n, i] \quad i = 0, 1 \quad (3.4)$$

Еквивалентната пресметка во Python би била:

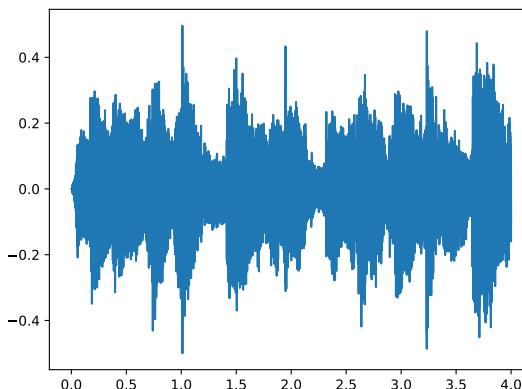
```
wav = wav - np.mean(wav, axis=0)
wav_mono = wav_mono - np.mean(wav_mono)
```

Прикажување звук

За да го прикажеме аудиосигналот ќе го искористиме `Matplotlib`:

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

Со што ќе биде исцртан графиконот прикажан на Сл. 3.1. 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 можат да се движат соодветно во опсегот од -2^{15} до $2^{15}-1$.



Сл. 3.2: Приказ на звукот `wav_mono` како вектор од елементи од типот `float64` со време на земање на примероците на x -оската.

Вообичаена практика е дигиталните сигнали да ги доведеме во опсег од -1 до 1 за нивна понатамошна обработка. Сè што треба да направиме е да го поделим векторот `wav_mono` со 2^{15} :

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

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

Може да забележиме дека сега елементите од векторот не се веќе од типот `int16`, ами од типот `float64`. Исто така, поради оваа промена, меморискиот простор што го зафаќа векторот сега е 4 пати поголем.

Да го исцртаме повторно аудиосигналот со тоа што овојпат ќе создадеме временски вектор за да го прикажеме сигналот во време. Исто така, за да ја избегнеме потребата да ја извршиме наредбата `plt.show()` за да го прикажеме графиконот, ќе ја вклучиме опцијата на IPython за интерактивно плотирање со магичниот оператор `%matplotlib`.

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

Добиениот графикон е прикажан на Сл. 3.2.

Преслушување на аудиосигналот

За да го преслушаме новодобиениот аудиосигнал, првин ќе го запишеме како аудиозапис, а потоа ќе ја искористиме системската `play` наредба од софтверскиот пакет `Sox`, види [Додаток Б](#).

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

`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.
```

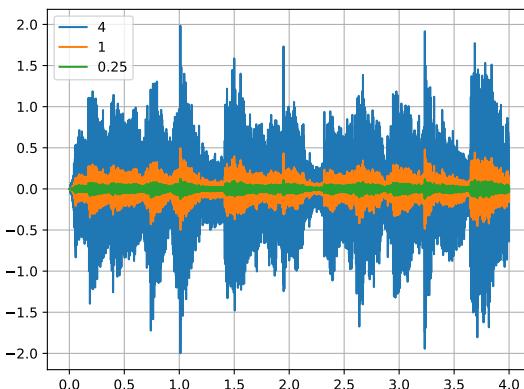
Засилување на аудиосигналот

За да го засилиме или втишиме аудиосигналот во дигитален домен, сè што треба да направиме е да го помножиме со некој коефициент за скалирање:

$$y[n] = a \cdot x[n] \quad (3.5)$$

каде што a е коефициентот за скалирање.

Бо Python ова може да се направи на следниот начин:



Сл. 3.3: Приказ на аудиосигналот `wav_mono` скалиран со различни коефициенти.

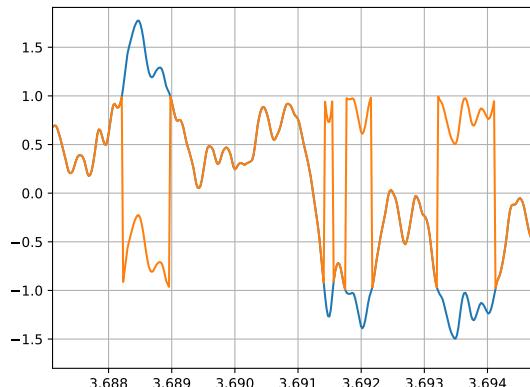
```
wav_loud = wav_mono * 4
wav_quiet = wav_mono * .25
plt.figure()
plt.plot(t, wav_loud)
plt.plot(t, wav_mono)
plt.plot(t, wav_quiet)
plt.legend([4, 1, 0.25])
plt.grid()
```

Добиениот графикон е даден на Сл. 3.3. Може да видиме дека засилениниот 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.³ 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. 3.4.

³Така, со 16-битна резолуција со употреба на двоен комплемент најголемата позитивна вредност е $2^{15} - 1 = 32767$, која се запишува како `0111 1111 1111 1111`, со тоа што MSB битот е битот за знак; тој изнесува 0 за позитивни броеви. Ако сигналот има вредности поголеми од 1 , всушност поголеми од $1 - 2^{-15}$ за едно квантизациско ниво 2^{-15} тие ќе бидат запишани со следната поголема вредност која е `1000 0000 0000 0000`, што во двоен комплемент претставува -32768 !



Сл. 3.4: Приказ на засилениот аудиосигнал и начинот на којшто Python го запишал на дискот.

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

Поврзана појава со оваа е пресекувањето на амплитудите на звучниот сигнал ако тие го надминуваат дозволениот опсег од -1 до 1 , која се нарекува [дисторзија](#):⁴

$$y[n] = \begin{cases} 1 & \text{ако } x[n] > 1 \\ -1 & \text{ако } x[n] < -1 \\ x[n] & \text{поинаку} \end{cases} \quad (3.6)$$

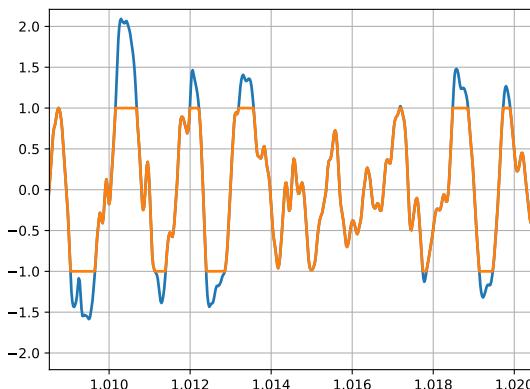
За да ја демонстрираме дисторзијата може да напишеме:

```
wav_dist = wav_loud.copy()
wav_dist[wav_dist > 1] = 1
wav_dist[wav_dist < -1] = -1

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

plt.figure()
plt.plot(t, wav_loud)
```

⁴Таа во некои типови на музика се користи како пожелен аудиоэффект, види Глава 6.



Сл. 3.5: Приказ на дисторзија во аудиосигналот `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`. Ако не направевме копија, односно ако само напишевме `wav_dist = wav_loud`, новата променлива ќе покажуваше кон истиот објект кон кој покажува `wav_loud`, па сите промени ќе беа направени врз оригиналната низа! Добиениот графикон е прикажан на Сл. 3.5.

Нормализација на аудиосигналот

Во дигиталното процесирање звук, поради ограничениот опсег од -1 до 1 во којшто треба да се наоѓа звучниот сигнал за да се избегне појавата на преклопување односно дисторзија при неговото запишување и преслушување, од посебно значење е процесот на **нормализација**. Под нормализација се подразбира доведување на максималната амплитуда на аудиосигналот A_{\max} на ниво L зададено со:

$$L = 20 \log \frac{A_{\max}}{1} [\text{dBFS}], \quad (3.7)$$

каде што 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 Полната скала⁵. Од (3.7) 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-

⁵Англ. *Full Scale*.

нивното понатамошно процесирање или пренос да не дојде до дисторзија.⁶

Да направиме функција за нормализација:

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

која сега ќе 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()
```

3.3 Создавање на звук

Наједноставниот простопериодичен звук претставува синусен тон дефиниран со:

$$x(t) = A \sin(2\pi ft + \varphi) \quad (3.8)$$

каде што A е амплитудата, f е фреквенцијата на тонот, а φ е неговата фаза. За да го генерираме во Python, треба најпрвин да го направиме временскиот вектор t со примероци од 0 до 4 секунди со фреквенција на земање примероци f_s , а потоа да ги пресметаме вредностите на синусот:

```
t = np.arange(0, 4 * fs) / fs # s
f = 200 # Hz
sound = np.sin(2 * np.pi * f * t)
plt.plot(sound)
wavefile.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.⁷

```
$ xdg-open make_sound.py &
```

⁶Оваа амплитудна маргина која што се остава до дозволениот максимум на англиски се нарекува `headroom`.

⁷Бо GNOME десктоп средината тоа е `gedit`, во KDE Plasma `kate`, во Xfce `mousepad`, во MATE `pluma` ... Во терминалот тоа може да го направиме со наредбата `xdg-open` која служи за отворање на датотеките со програмот кој е поставен за тој тип на датотеки во системот, според екstenзијата во името на датотеката.

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 ѝ пренесеме
# влезни параметри преку командната линија
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 IPython:

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

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

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

§ Дополнително. Во првата линија од скриптата со синтаксата `#!`, на англиски позната и како *shebang*, кажуваме со кој интерпретер таа треба да биде извршена. Ова е независно од екstenзијата на датотеката, која може да биде и целосно изоставена!

Причината зашто го повикуваме `/bin/env python3`, а не да речеме `/bin/python` е тоа што сакаме да ја извршиме скриптата со Python интерпретерот во виртуелната средина во која се наоѓаме, а не системската Python средина која ги нема пакетите NumPy и SciPy. Всушност, при извршување на секоја датотека во Linux, школката, односно шелот, го извршува процесот во нова школка која ја создава само за таа намена. Со употреба на `env`, ги проследуваме системските променливи, меѓу кои и `PATH`, на новата школка, со што првиот Python што ќе биде најден повторно ќе биде оној од активната виртуелна средина. Конечно, за да можеме да ја извршиме директно датотеката, треба да ги смениме нејзините пермисии од `-rw-r--r--` во `-rwxr--r--`, односно да го сетираме знаменцето за извршување `x` за нашиот корисник `u` со наредбата:

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

За да ја извршиме скриптата во терминал сега може да напишеме:

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

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

Глава 4

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 (Smith, 2011). Со негова помош можеме да ги видиме карактеристиките на звучниот сигнал што ги слушаме, а кои не се јасно видливи во неговиот временски облик. За пресметка на спектарот на аудиосигналите ќе се послужиме со Фуриевата трансформација и нејзината временски определена форма Фуриева трансформација со прозорци (STFT)¹. Ќе видиме како STFT на звучниот сигнал води до една прегледна форма на претставување на неговите временско-спектрални, односно спектро-временни особини – спектрограмот.

4.1 Основи на Фуриевата анализа

Во оваа глава ќе направиме краток преглед на основите на Фуриевата анализа. Притоа ќе се задржиме само на особините и деталите кои се релевантни за дигиталното процесирање звук.²

¹Англ. *Short-Time Fourier Transform*.

²За подетален преглед погледнете го учебникот на предметот Основи на дигитално процесирање на сигнали на додипломските студии на ФЕИТ (Богданов, 1997).



§ Дополнително. Францускиот математичар и физичар Жан-Батист Жозеф Фурие³(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 придржувал Наполеон Бонапарта во неговиот поход во Египет во 1798, а по враќањето во Франција во 1801 станува претседател на општината Изер во Гренобл, каде што започнува да ги прави своите експерименти со топлина.

Фуриеов интеграл

Фуриеовата трансформација (FT)⁴ на сигналот $x(t)$ е дефинирана со интегралот:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t} dt \quad \omega \in (-\infty, \infty) \quad (4.1)$$

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

Инверзната Фуриеова трансформација (IFT)⁵ е дефинирана со интегралот:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega)e^{j\omega t} d\omega, \quad t \in (-\infty, \infty) \quad (4.2)$$

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

$$X(\omega) = |X(\omega)|e^{j\angle X(\omega)} = A(\omega)e^{j\varphi(\omega)} \quad (4.3)$$

каде што $A(\omega)$ 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 (4.2), користејќи ја **Ојлеровата формула**,

³Wikipedia – 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>

⁴Англ. *Fourier Transform*.

⁵Англ. *Inverse Fourier Transform*.

Табела 4.1: Позначајни својства на Фуриевата трансформација.

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

добиваме:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} A(\omega) e^{j\varphi(\omega)} e^{j\omega t} d\omega \quad (4.4)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} A(\omega) [\cos(\omega t + \varphi(\omega)) + j \sin(\omega t + \varphi(\omega))] d\omega \quad (4.5)$$

$$= \frac{1}{\pi} \int_0^{\infty} A(\omega) \cos(\omega t + \varphi(\omega)) d\omega \quad (4.6)$$

Основните својства на Фуриевата трансформација се дадени во Табелата 4.1.

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

Во Табелата 4.2 се дадени некои позначајни Фуриеви парови.

Енергијата на сигналот може да се пресмета преку неговиот спектар со употреба на теоремата на Парсевал⁶:

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |X(\omega)|^2 d\omega = \int_{-\infty}^{\infty} |X(2\pi f t)|^2 df \quad (4.7)$$

⁶Wikipedia – Parseval’s Theorem https://en.wikipedia.org/wiki/Parseval%27s_theorem

Табела 4.2: Позначајни Фуриеови парови.

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

* Важно! Во Табелата 4.2 забележете дека FT 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:

$$x(t) \equiv x(t + T), \quad \forall t \quad (4.8)$$

Ако со $\hat{x}(t)$ го означиме сегментот на $x(t)$ во основниот интервал:

$$\hat{x}(t) = x(t) \cdot \Pi(t) \quad (4.9)$$

$$\Pi(t) = \begin{cases} 1, & |t| \leq \frac{T}{2}, \\ 0, & |t| > \frac{T}{2}. \end{cases} \quad (4.10)$$

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

$$x(t) = \sum_{r=-\infty}^{\infty} \hat{x}(t - rT). \quad (4.11)$$

Во тој случај во Фуриев домен, функцијата ќе биде еднаква на сума од експоненцијални функции, односно комплексни синусоиди со дискретни

фреквенции, која се нарекува и **Фуриеов ред**:

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{jk\frac{2\pi}{T}t} = \sum_{k=-\infty}^{\infty} c_k e^{jk\omega_0 t} \quad (4.12)$$

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

$$\omega_0 = 2\pi f_0 = \frac{2\pi}{T} \quad (4.13)$$

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

$$c_k = \frac{1}{T} \int_{-T/2}^{T/2} x(t) e^{-jk\frac{2\pi}{T}t} dt \quad (4.14)$$

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

$$X(\omega) = 2\pi \sum_{k=-\infty}^{\infty} c_k \delta(\omega - k\omega_0). \quad (4.15)$$

\mathcal{Z} трансформација

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

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n} \quad (4.16)$$

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

$$x[n] = x[nT_s] \quad (4.17)$$

$$T_s = \frac{1}{f_s} \quad (4.18)$$

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

Инверзната \mathcal{Z} трансформација е дефинирана со:

$$x[n] = \frac{1}{2\pi j} \oint_C X(z) z^{n-1} dz \quad (4.19)$$

Табела 4.3: 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 4.4: 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 $

каде што 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 4.3, а позначајни \mathcal{Z} трансформации во Табелата 4.4.

Дискретна Фуреева трансформација

Фуреевата трансформација на дискретен сигнал може да се добие од равенката за \mathcal{Z} трансформацијата (4.16) со избор на јадрото $z = e^{j\omega T_s}$, што се поедноставува на $z = e^{j\omega}$ за $T_s = 1$:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}, \quad (4.20)$$

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n} d\omega \quad (4.21)$$

Во овој случај, ја ограничуваме \mathcal{Z} трансформацијата на единечната кружница во z рамнината описана со:

$$e^{j\omega} = \cos(\omega) + j \sin(\omega) \quad (4.22)$$

каде што дигиталната кружна фреквенција ω има интерпретација на агол во оваа рамнина. Со други зборови Фуреевата трансформација на дискретниот сигнал претставува специјален случај на \mathcal{Z} трансформацијата.

Бидејќи јадрото $e^{j\omega}$ е периодична функција со периода 2π , следува дека фреквенцискиот спектар на дискретните сигнали е исто така периодичен со периода 2π на кружната фреквенција ω . За произволно T_s , спектарот ќе има периода од $2\pi/T_s$ во однос на кружната фреквенција ω , односно периодата е $1/T_s = f_s$ во однос на фреквенцијата f , како што беше кажано во Главата 2.1.

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

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

$$\tilde{x}[n] = \tilde{x}[n + N], \quad -\infty < n < \infty \quad (4.23)$$

тогаш $\tilde{x}[n]$ во Фуриеов домен ќе биде еднаква на сума од синусоиди:

$$\tilde{X}[k] = \sum_{n=0}^{N-1} \tilde{x}[n] e^{-jk\frac{2\pi}{N}n} \quad (4.24)$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \tilde{X}[k] e^{jk\frac{2\pi}{N}n} \quad (4.25)$$

Ако сега ја земеме Фуриеовата трансформација на дискретниот сигнал $x[n]$ дадена во (4.20), која претставува \mathcal{Z} трансформација на сигналот $x[n]$ долж единствената кружница, и од истата земаме примероци на еквидистантни фреквенции ω_k дадени со:

$$\omega_k = k \frac{2\pi}{N} \quad (4.26)$$

тогаш ќе го добиеме изразот за [дискретната Фуриеова трансформација \(DFT\)](#)⁷:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-jk\frac{2\pi}{N}n} \quad (4.27)$$

кој е еквивалентен со (4.24)!

Ова значи дека семплираниот спектар што го добиваме со (4.27) соодветствува на сигнал $\tilde{x}[n]$ кој претставува продолжена верзија од сигналот $x[n]$ добиена со периодично повторување на неговите N примероци, односно важи:

$$x[n] = \tilde{x}[n] \cdot \Pi[n] \quad (4.28)$$

⁷Англ. *Discrete Fourier Transform*.

каде што

$$\Pi[n] = \begin{cases} 1, & 0 \leq n < N, \\ 0, & \text{поинаку.} \end{cases} \quad (4.29)$$

Со други зборови, дискретната Фурьеова трансформација на еден дискретен сигнал $x[n]$ ни ги дава примероците од спектарот на периодичната верзија на тој сигнал во која тој се повторува бесконечно со периода еднаква на неговата должина N , како што е прикажано на [Сл. 4.1](#).

[Инверзната дискретна Фурьеова трансформација \(IDFT\)](#)⁸ ќе биде дефинирана со изразот:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{jk \frac{2\pi}{N} n} \quad (4.30)$$

За произволно T_s примероците на спектарот ќе бидат пресметани за кружните фреквенции:

$$\omega_k = k \frac{2\pi}{NT_s} \quad (4.31)$$

односно:

$$f_k = k \frac{1}{NT_s} = k \frac{f_s}{N} \quad (4.32)$$

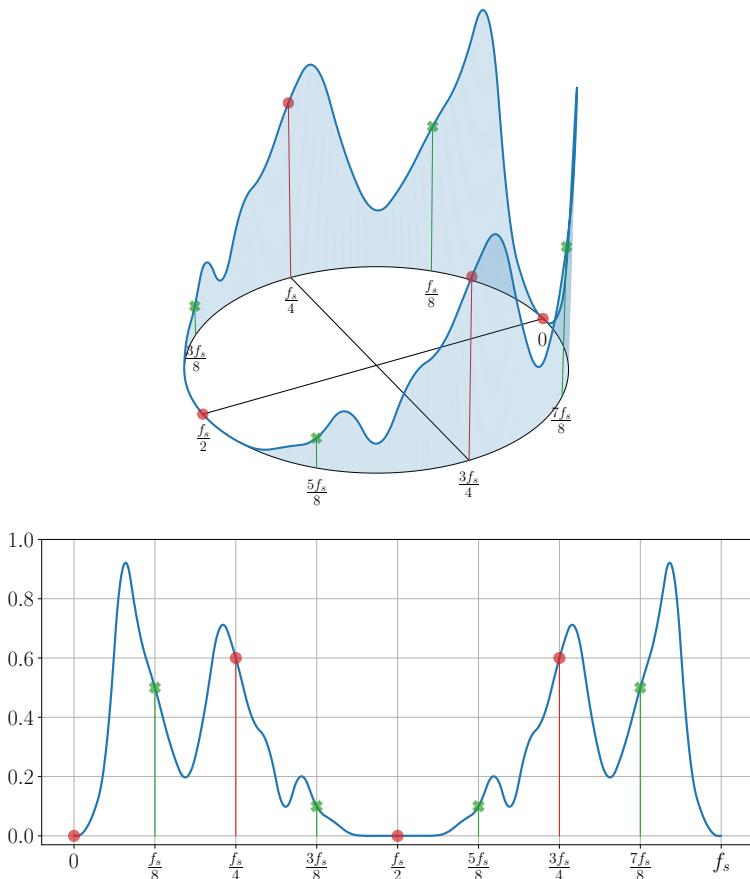
* **Пример.** Да земеме еден дискретен сигнал со должина $N = 4$, тогаш примероците од неговиот спектар добиен со DFT ќе соодветствуваат на следните вредности на кружната фреквенција ω : $\omega_0 = 0$, $\omega_1 = \pi/2$, $\omega_2 = \pi$ и $\omega_3 = 3\pi/2$. Поради симетричноста на амплитудниот спектар за реални сигнали, што се сведува на симетричноста на кружницата во однос на реалната оска во z рамнината, примероците ω_1 и ω_3 се огледални слики и се идентични, како на [Сл. 4.1](#).

Поради тоа што тие не носат дополнителна информација, вообичаена практика е огледалните слики да се исфрлат, при што мора да се удвојат соодветните примероци кои остануваат за да важи теоремата на Парсевал. По исфрлањето, последниот примерок од спектарот ќе биде за кружна фреквенција $\omega_3 = \pi$, односно, за произволно T_s , за фреквенција $f = f_s/2$. Така, конечната скратена верзија на DFT спектарот би се состоела од: $X(e^{j0})$, $2X(e^{j\pi/2})$ и $X(e^{j\pi})$.

Фреквенциската резолуција на примероците од спектарот добиени со DFT ќе биде одредена од должината на сигналот N :

$$\Delta\omega = \frac{2\pi}{N} \quad (4.33)$$

⁸Англ. *Inverse Discrete Fourier Transform*.



Сл. 4.1: Спектар на дискретен сигнал прикажан на единечната кружница (горе) и на фреквенциската оска (долу) со примероци добиени со DFT за должина на сигналот $N = 4$ (првени маркери) и $N = 8$ (првени и зелени маркери).

односно:

$$\Delta f = \frac{f_s}{N} \quad (4.34)$$

За да ја зголемиме резолуцијата на спектарот добиен со DFT може да примениме две методи: *i*) да земеме повеќе примероци од сигналот N , и *ii*) да се додадеме N_0 нули на крајот од сигналот. Последново ќе ја зголеми резолуцијата на земање примероци во јадрото, без да ги смени границите на сумата:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-jk \frac{2\pi}{N+N_0} n} \quad (4.35)$$

Сепак, новодобиениот спектар со $N + N_0$ примероци нема да содржи нова информација; тој ќе претставува само интерполирана верзија на оригиналниот спектар со N точки.

За пресметка на DFT во праксата се користи алгоритам за нејзино брзо пресметување што се нарекува **брза Фуриеова трансформација (FFT)**⁹ (Богданов, 1997). Неговата пресметковна сложеност е пропорционална со $N \log N$. Бидејќи FFT работи брзо само за должина 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 спектарот на еден дискретен сигнал ќе го пресметаме спектарот на една простопериодична низа $x[n]$ со кружна фреквенција ω_0 :

$$x[n] = A \cos[\omega_0 n] = A \cos[2\pi f_0 n] \quad (4.36)$$

Спектарот на овој сигнал добиен со Фуриевата трансформација е даден со равенството:

$$X(\omega) = \frac{A}{2} 2\pi (\delta(\omega + \omega_0) + \delta(\omega - \omega_0)) \quad (4.37)$$

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:

$$\hat{X}(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\Omega) W(\omega - \Omega) d\Omega \quad (4.38)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{A}{2} 2\pi (\delta(\Omega + \omega_0) + \delta(\Omega - \omega_0)) W(\omega - \Omega) d\Omega \quad (4.39)$$

$$= \frac{A}{2} (W(\omega + \omega_0) + W(\omega - \omega_0)) \quad (4.40)$$

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

$$W(\omega) = \frac{\sin(N \frac{\omega}{2})}{\sin(\frac{\omega}{2})} \quad (4.41)$$

Според Лопиталовото правило¹⁰, кое вели дека во случаи кога за гранична вредност на функција се добива неопределен израз како $0/0$ или ∞/∞ , тогаш

⁹Wikipedia – Fast Fourier transform (FFT) https://en.wikipedia.org/wiki/Fast_Fourier_transform

¹⁰Википедија – Лопиталово правило https://mk.wikipedia.org/wiki/Лопиталово_правило

постои гранична вредност и таа се наоѓа преку изводите на двете функции, односно:

$$W(0) = \lim_{\omega \rightarrow 0} \frac{\sin(N\frac{\omega}{2})}{\sin \frac{\omega}{2}} = \lim_{\omega \rightarrow 0} \frac{\sin'(N\frac{\omega}{2})}{\sin' \frac{\omega}{2}} = \lim_{\omega \rightarrow 0} \frac{N \cos(N\frac{\omega}{2})}{\cos \frac{\omega}{2}} = N \quad (4.42)$$

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:

$$\hat{X}(\omega_0) = \frac{A}{2} (W(2\omega_0) + W(0)) \approx \frac{A}{2} N \quad (4.43)$$

Тоа значи дека множењето на дискретниот сигнал со прозорецот ја скалира амплитудата на неговите спектрални компоненти со коефициент кој соодветствува на неговата должина N . Поради тоа, ако сакаме да ја добиеме амплитудата на спектралните компоненти на оригиналниот сигнал, треба истите да ги поделиме со N .

4.2 Спектар на простопериодичен звук

Да го пресметаме и прикажеме спектарот на еден простопериодичен, односно синусоидален тон. За таа цел, 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` која е дел од аудиозаписите од предметот [Електроакустика](#)¹¹, но го имаме и во проектната папка на предметов.

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

§ Дополнително. При првото извршување на Python скрипта во `Spyder` треба да одберете каде да биде извршен кодот. Избирајќи `Execute in current Python or IPython console`, сите променливи генериирани во сегашниот работен простор можат да бидат искористени од кодот и ќе бидат на располагање и по завршувањето на скриптата.

¹¹Материјалите за предметот Електроакустика на ФЕИТ може да ги најдете на <https://gitlab.com/feeit-freecourseware/electroacoustics>

§ 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, но не е погодно за зголемување на приказот (зумирање) за анализа на деталите од графиконите. Втората опција е тие да бидат исцртани во посебен прозорец, што освен зумирање овозможува и интерактивност како дополнително исцртување на нови содржини на веќе генериралиот графикон. За да ја активираме втората опција треба да напишеме `%matplotlib` во IPython конзолата.

§ Дополнително. За полесно да научите како да пишувате правилно форматиран код во Python во Spyder вклучете ја опцијата за автоматска проверка на кодот според [PEP 8](#)¹² стандардот што може да ја најдете во менито со поставки на следната локација `Code style and formatting > Enable code style linting`.

```
...
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

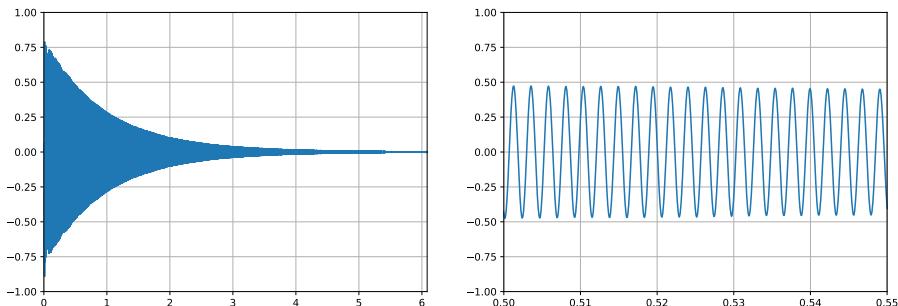
audio_path = 'audio/'
wav_name = 'viluska.wav'
fs, wav = wavfile.read(audio_path + wav_name)
os.system('play ' + audio_path + wav_name)

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]
```

Добиените графикони се прикажани на Сл. 4.2

¹²PEP 8: Style Guide for Python Code, <https://peps.python.org/pep-0008/>



Сл. 4.2: Временски облик на аудиосигналот генериран од звучна вилушка.

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 Фуриеова анализа.

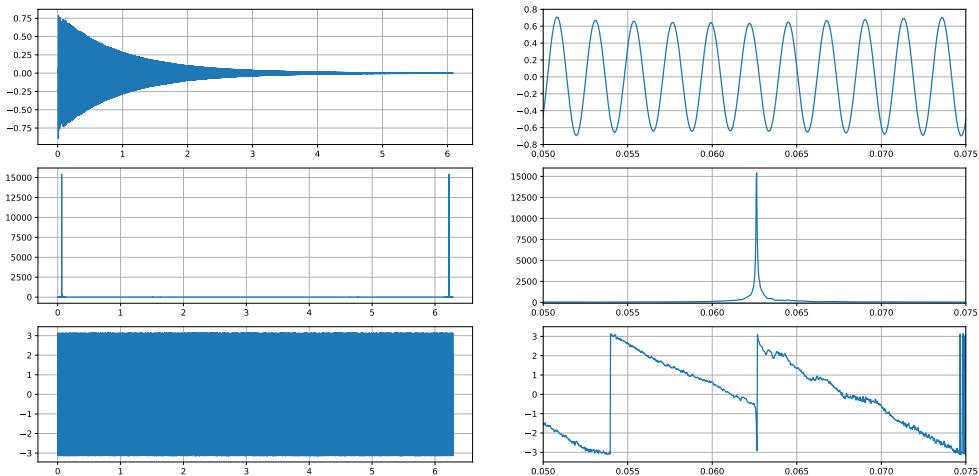
§ Дополнително. За модуларно пишување и извршување на кодот, може да го поделим во клетки со помош на синтаксата `# %%` што во Spyder претставува нова ќелија. Со тоа овој дел од кодот ќе може едноставно да го извршиме со притискање `Ctrl + Enter` без притоа да го извршиме целиот код од почеток, како кога би притиснале `F5`. Spyder нуди и можност за извршување на единечни линии од кодот со притискање на `F9`.

```
# %% 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. 4.3, без и со зумирање. 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:



Сл. 4.3: Временски облик на аудиосигнал од звучна вилушка (горе) и неговиот амплитуден (средина) и фазен (долу) спектар без (лева колона) и со зумирање (десна колона).

- 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.

За побрза работа на FFT должината на влезниот сигнал треба да биде степен од 2. За таа цел, сигналот вообичаено се дополнува со N_0 нули до следната поголема должина која е степен од 2, што може да се пресмета на следниот начин:

$$N_{\text{FFT}} = 2^M \quad (4.44)$$

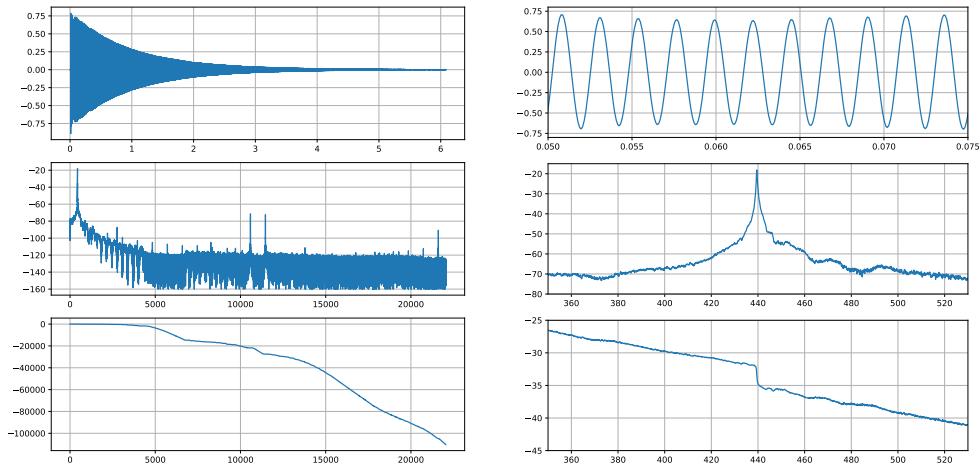
каде што:

$$M = \lceil \log_2 N \rceil \quad (4.45)$$

а N е должината на сигналот.

Уште повеќе, поради тоа што за реални сигнали, какви што се аудиосигналите, втората половина од примероците од спектарот што ги добиваме со DFT се огледална слика на првата половина, нив може да ги исфрлиме како редундантни. Притоа, не треба да заборавиме да ги удвоиме амплитудите на примероците од спектарот чии слики сме ги исфрлиле.

Конечно, за да добиеме поверна репрезентација на спектарот на аудиосигналот според (4.43), амплитудниот спектар го скалираме преку деление со должината на сигналот N . Во ова скалирање не ги земаме предвид



Сл. 4.4: Вообичаен приказ на амплитудниот (средина) и фазниот (долу) спектар на аудиосигналот генериран од звучна вилушка (горе).

N_0 дополнителните нули, затоа што тие не ја зголемуваат должината на правоаголниот прозорец со кој имплицитно го множиме сигналот.

Да ги примениме овие работи во нашиот 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)
```

Во кодов, пред конверзија на амплитудниот спектар во dB, ја пресекуваме неговата најмала вредност на произволно избрана мала вредност `eps` за да избегнеме пресметка на $\log(0)$. Добиениот приказ на амплитудниот и фазниот спектар на аудиосигналот е даден на Сл. 4.4.

✓ Задача за час. Испитайте што прави функцијата `fftshift` од модулот `scipy.fftpack`.

Бидејќи пресметката на спектарот на аудиосигналите е од суштествено значење во дигитално процесирање звук, можеме дадениот код да го поместиме во функција за таа намена. На функцијата ќе ѝ додадеме краток опис¹³ и заедно со функцијата `normalize`, која ја направивме во Главата 3 ќе ги поместиме во наш модул што 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))
    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.
```

¹³Англ. *docstring*.

```

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 овозможи 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 модули. Така `wav_amp` може сега да го добиеме 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 прикаже описот што го додадовме на нашата функција. Во докстрингот може да вклучиме многу повеќе информации вклучувајќи равенства и примери за употреба на нашата функција. Како пример за богат и добро форматиран докстринг погледнете го описот на функцијата `fft`.

4.3 Фуриеова трансформација со прозорци

Главниот недостаток на Фуриевата трансформација е тоа што таа не ни дава никаква временска информација за сигналот којшто 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 даде деталите. На пример, вкупниот спектар на еден музички сигнал може да ни каже дали во него има изразен бас, но не може да ни каже колку е брз ритамот на музичкиот сигнал.

Или пак, може да ни каже дека во сигналот има зурла со изразена енергија во високите фреквенции, но не може да ни каже која мелодија таа ја свири. Овие недостатоци се надминуваат со **Фуриевата трансформација со прозорци (STFT)**¹⁴.



§ **Дополнително.** Унгарскиот електроинженер **Денеш Габор**¹⁵ (1900–1979) е овој што во 1946 ја прилагодува 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:

$$X[i, k] = \sum_{n=-N/2}^{N/2-1} x[n]w[n + i \cdot H]e^{-jk\frac{2\pi}{N}n} \quad i = 0, 1, 2\dots \quad (4.46)$$

Тука, i е редниот број на рамката, N е должината на прозорецот, а H е големината на скокот што 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** (Богданов, 1997). На овој начин, наместо еден вкупен спектар, се добива низа од спектри од сигналот пресметани за различни временски моменти во избрана нивна околина.

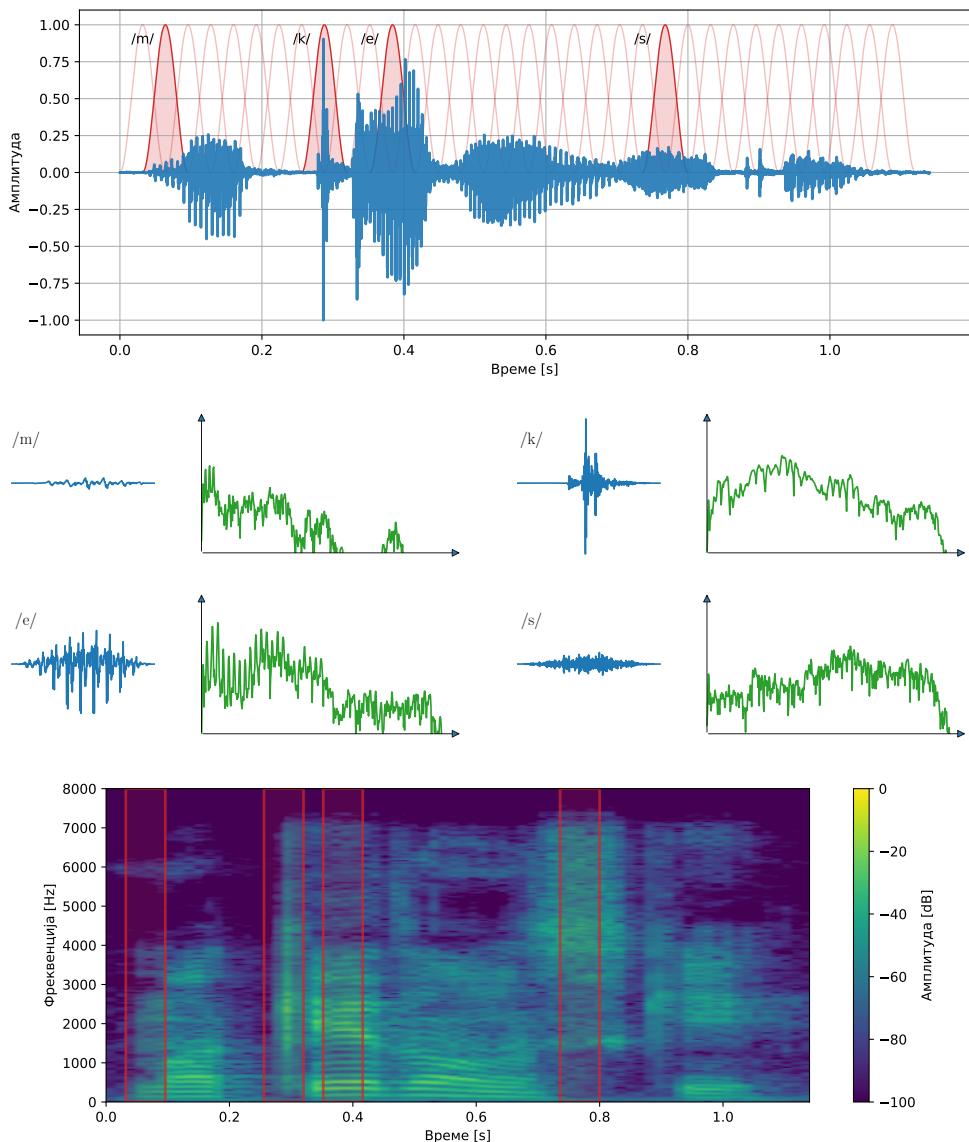
На Сл. 4.5 е прикажан говорниот аудиосигнал „македонски“; на него е суперпониран прозорецот со облик на подигнат косинус на позициите на кои го изминува сигналот. Рамките издвоени со прозорецот на четирите означени локации одговараат на гласовите /m/, /a/, /k/ и /e/. Временскиот облик и спектарот на овие четири рамки се прикажани во средниот дел на сликата.

Спектрограм

Спектрограмот е еден од најзначајните прикази на аудиосигналите. Тој овозможува во исто време да се набљудуваат карактеристиките на сигналот и во временски и во спектрален домен, поради што тој се нарекува и спектро-тимпорален приказ на сигналите. Спектрограмот претставува

¹⁴Англ. *Short-Time Fourier Transform*.

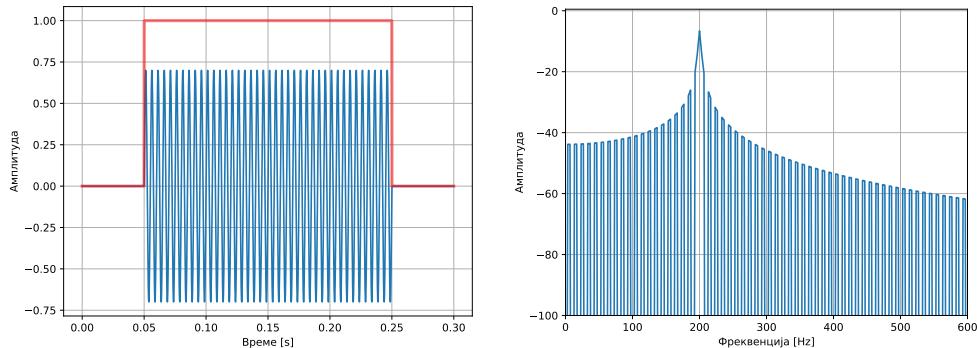
¹⁵Wikipedia: Dennis Gabor. https://en.wikipedia.org/wiki/Dennis_Gabor



Сл. 4.5: Фуриеова трансформација со прозорци применета на изговор на зборот „македонски“ (горе) и пресметаниот спектрограм (долу). Во средината се прикажани временските облици (сино) и спектарот (зелено) за рамките издвоени од сигналот на означените позиции.

графички приказ на спектарот на секоја од рамките добиени со придвижување на прозорецот долж сигналот при примена на STFT, како што е прикажано во Сл. 4.5.

Спектрите пресметани за секоја од рамките се прикажани вертикално долж y -оската, која претставува фреквенција. Вака вертикално поставени, спектрите се надодаваат еден покрај друг долж x -оската, која претставува



Сл. 4.6: Временски облик (лево) и спектар (десно) на простопериодичен звук со f од 200 Hz и f_s од 22.000 Hz, помножен со прозорец од 200 ms.

време. Временскиот интервал помеѓу два соседни спектри е еднаков на времетраењето на скокот на прозорецот при пресметување на STFT. За прикажување на амплитудата на примероците од спектрите, како z -оска, се применуваат мапи на бои.¹⁶ Всушност, спектрограмот претставува 2D претстава на 3D податоци.

На сликата може да видиме како хармониците присутни во спектарот на гласовите /m/ (три хармоници со ниски фреквенции) и /e/ прават паралелни хоризонтални линии долж спектрограмот. Од друга страна, беззвучните согласки /k/ и /s/ немаат хармоници. Поради тоа што /k/ претставува плозив со брз транзиент, тој зазема широк фреквенциски опсег долж фреквенциската оска на спектрограмот. Спектралната енергија на /s/ пак, е распределена во високите фреквенции и е претставена како шум на спектрограмот.

Видови на прозорци

Поради тоа што множењето на прозорецот со аудиосигналот во временски домен, соодветствува на конволуција на нивните спектри во фреквенциски домен, спектарот на прозорецот има критично влијание при пресметка на спектарот на аудиосигналите. Така, ако земеме простопериодичен звук составен од една синусоида со фреквенција f_0 , и ако го множиме со правоаголен прозорец, наместо Дираков импулс, на фреквенцијата f_0 ќе имаме поместен спектар $W[k]$ на прозорецот. Ова е прикажано на Сл. 4.6.

Бидејќи, кој било дискретен сигнал што ќе го анализираме има конечна должина, тоа е еквивалентно како да сме го помножиле со правоаголен прозорец, па изобличувањата што ги внесува спектарот на прозорецот во пресметаниот спектар не може да се избегнат.

¹⁶Англ. *colormap*.

Поради оваа причина, дизајнирани се низа од прозорци, секој со различни спектрални карактеристики.¹⁷ 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 се најупотребувани следните прозорци (Smith, 2011):

- Правоаголен прозорец

$$w[n] = 1, \quad -N/2 \leq n \leq N/2 \quad (4.47)$$

$$W[k] = \text{sinc}[k] \quad (4.48)$$

- Ханов прозорец

$$w[n] = 0,5 + 0,5 \cos[2\pi \frac{n}{N}], \quad -N/2 \leq n \leq N/2 \quad (4.49)$$

$$W[k] = 0,5 \text{sinc}[k] + 0,25 (\text{sinc}[k-1] + \text{sinc}[k+1]) \quad (4.50)$$

- Хамингов прозорец

$$w[n] = 0,54 + 0,46 \cos[2\pi \frac{n}{N}], \quad -N/2 \leq n \leq N/2 \quad (4.51)$$

- Блекманов прозорец

$$w[n] = 0,42 - 0,5 \cos[2\pi \frac{n}{N}] + 0,080,5 \cos[4\pi \frac{n}{N}], \quad -N/2 \leq n \leq N/2 \quad (4.52)$$

- Блекман-Харисов прозорец

$$w[n] = \frac{1}{N} \sum_{i=0}^3 a_i \cos[2i\pi \frac{n}{N}], \quad -N/2 \leq n \leq N/2 \quad (4.53)$$

каде што

$$a_0 = 0,35876, a_1 = 0,48829, a_2 = 0,14128, a_3 = 0,01168 \quad (4.54)$$

Карактеристики на прозорците

Најважните спектрални карактеристики на прозорците се **ширината на главното крило и амплитудата на споредните крила**. 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 создадеме со функцијата `get_window` од модулот `scipy.signal`. Во следниот код тоа е направено за правоаголниот прозорец `boxcar`. Притоа, амплитудниот спектар се нормализира до 0 dB за прикажување.

¹⁷Wikipedia: Window function https://en.wikipedia.org/wiki/Window_function

```

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)

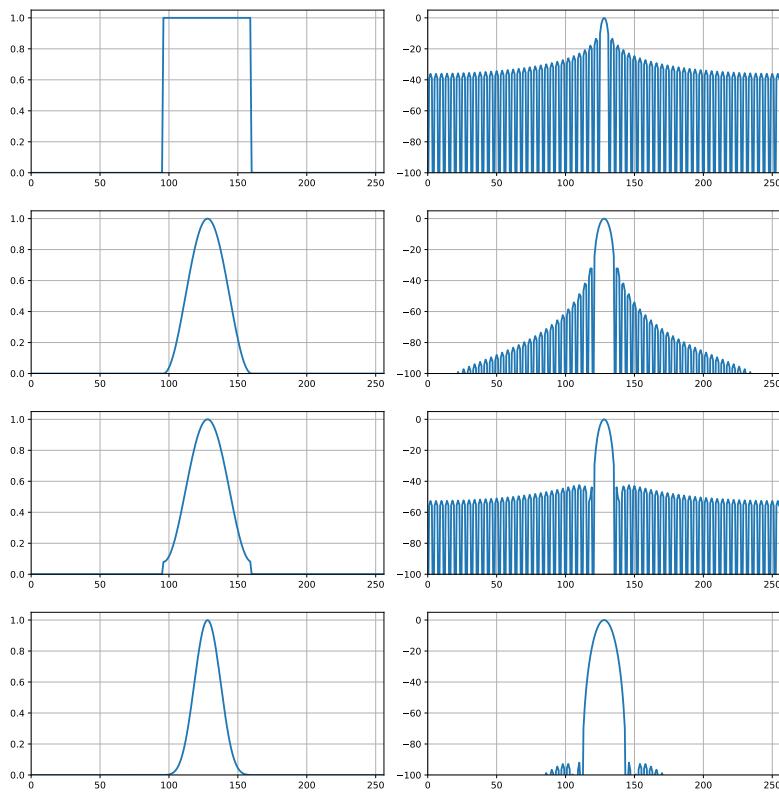
plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.plot(w)
plt.axis([0, m, 0, 1.1]) # [xmin, xmax, ymin, ymax]
plt.grid()
plt.subplot(122)
plt.plot(w_shift)
plt.grid()
plt.axis([0, m, -100, 10])

```

Притоа, повторно ја воведовме произволно малата вредност `eps` за избегнување на пресметка на логаритамот за вредности на спектарат кои се речиси нула. Различните прозорци и нивните амплитудни спектри генерирали со овој код се прикажани на Сл. 4.7. Може да видиме 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 прозорец, 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. Тие се и најчестиот избор за прозорец при процесирање на аудиосигналите со методата на прозорци.

§ Дополнително. Нумеричката прецизност за `float64` изнесува $2,220446049250313 \cdot 10^{-16}$, а може да се прикаже преку NumPy со `np.finfo(float).eps`.



Сл. 4.7: Споредба на временскиот и спектрален облик на четири најчесто користени прозорци (од горе надолу): правоаголен, Ханов, Хамингов и Блекман-Харисов.

Пресметка на спектрограмот

Ќе ја имплементираме STFT во Python за да го пресметаме и прикажеме спектрограмот на еден аудиозапис `zvona2.wav`¹⁸.

```
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)
```

¹⁸Звукот Gentle Glockenspiel од bbatv е земен од Freesound.org <http://freesound.org/people/bbatv/sounds/332932/>

```

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)
    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])

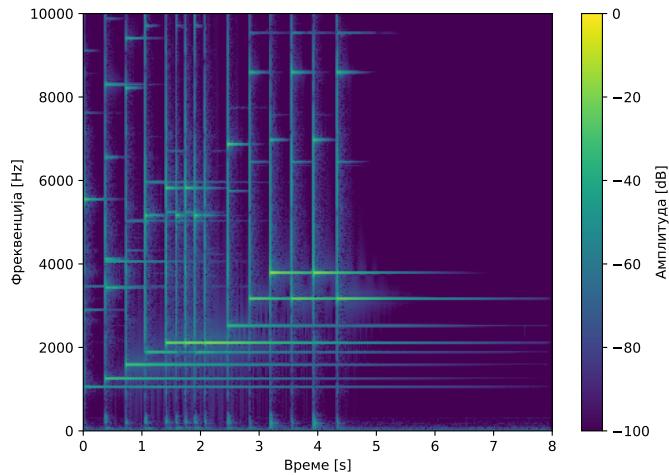
```

Добиениот спектrogram е прикажан на Сл. 4.8.

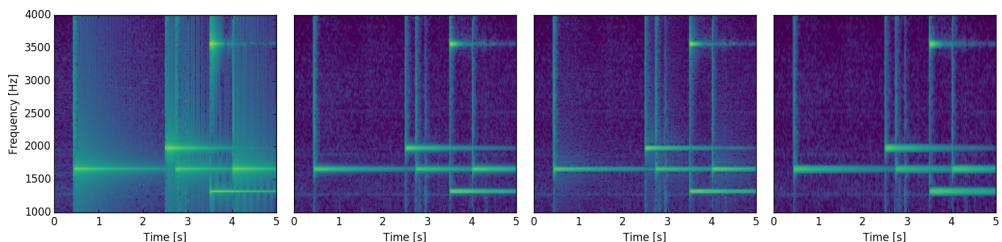
★ Важно! Најважните два параметри при пресметување на спектрограмот се:

1. **Типот на прозорец.** Разликите кои се добиваат со различните прозорци може да се воочат во деталот прикажан на Сл. 4.9.
2. **Должината на прозорецот.** Колку е подолг прозорецот толку повеќе точки во спектарот, односно поголема резолуција. Но исто така, колку подолг прозорец толку погруба временска резолуција, како што се гледа на Сл. 4.10.

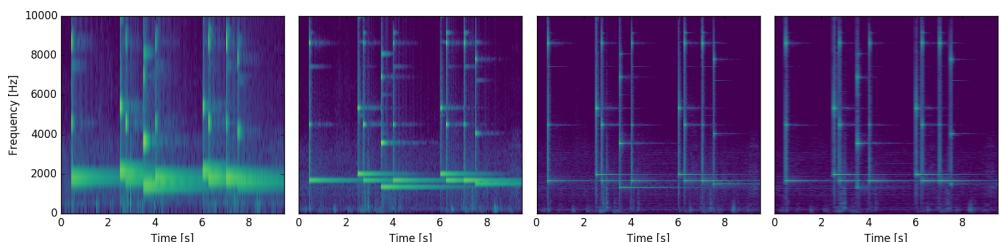
Имено, временските моменти на промена во спектарот стануваат размачкани. Така, за добра временска резолуција ни треба пократок прозорец, што пак повлекува лоша фреквенциска резолуција. Потребата од компромис меѓу време и фреквенција е главниот недостаток кај спектрограмите, односно кај STFT анализата.



Сл. 4.8: Спектрограм на аудиозаписот zvona2.wav добиен со употреба на Хамингов прозорец со должина 2048 примероци (46 ms).



Сл. 4.9: Детал од спектрограмот на zvona2.wav добиен со (лево кон десно): правоаголен, Ханов, Хамингов и Блекман-Харисов, сите со 2048 примероци.



Сл. 4.10: Детал од спектрограмот на zvona2.wav добиен со Хамингов прозорец со должина од (лево кон десно): 128, 512, 4096 и 8192 примероци.

§ Дополнително. За прикажување на амплитудата на спектрограмот постојат најразлични мапи на бои. Долго време во стандардна употреба беше мапата jet, но таа има низа недостатоци, поради што денес целосно е исфрлена од употреба. Мапата искористена за приказ на спектрограмите во оваа глава е новата перцептуално линеарна мапа viridis.¹⁹

Останува да ги внесеме функциите за пресметка на спектrogramот и неговото исцртување 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.

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:
```

¹⁹ Особено интересното претставување на новата мапа на конференцијата SciPy 2015 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=xAoljeRJ3lU>

```

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)
else:
    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],
        vmin=-100,
        vmax=None,
    )
    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.

Глава 5

Филтри

Филтрите претставуваат системи чија примарна намена е обликувањето на спектарот на сигналите. Поради тоа, тие се нераздвоен дел од дигиталниот звук. Потребни се аналогни и дигитални филтри за ограничување на спектарот при А/Д конверзија и за обликување на излезниот аналоген сигнал при D/A конверзија. Дигитални филтри имаме долж каналот за пренос на дигиталниот звук (трансмисија, дигитално снимање), а тие 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 тешка за реализација и одржување. Од друга страна, реализацијата на сложени филтерски структури во дигитален домен се сведува на нумерички операции кои не се пресметковно скапи, ниту пак се подложни на временски и температурни влијанија.

5.1 Основи на дигиталните филтри

Дигиталните филтри претставуваат линеарни и временски инваријантни (LTI) системи, односно линеарни и инваријантни на поместување дискретни системи¹. Овие две особини го изискуваат следниот услов: ако за даден влезен сигнал $x[n]$ системот го дава излезниот сигнал:

$$y[n] = H\{x[n]\} \quad (5.1)$$

тогаш мора да важи (Богданов, 1997):

$$H\left\{\sum_{m=1}^M a_m x_m[n]\right\} = \sum_{m=1}^M a_m y_m[n], \quad \forall a_m \text{ — линеарност и} \quad (5.2)$$

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

$$H\{x[n - k]\} = y[n - k], \quad \forall k \quad - \text{инваријантност на поместување.} \quad (5.3)$$

Секој LTI систем е во потполност описан од неговиот **импулсен одсив** $h[n]$ односно од неговата **преносна функција** во z -доменот $H(z)$, или пак конечно од неговата **фреквенциска карактеристика** во Фуреовиот домен $H(e^{j\omega})$.

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):

$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m] \quad (5.4)$$

$$Y(z) = H(z)X(z) \quad (5.5)$$

$$Y(e^{j\omega}) = H(e^{j\omega})X(e^{j\omega}) \quad (5.6)$$

Дополнително, сите LTI системи кои се практично применливи како дигитални филтри можат да се описат со **диференцната равенка**:

$$y[n] = \sum_{i=0}^p b_i x[n-i] - \sum_{i=1}^q a_i y[n-i] \quad (5.7)$$

Како што може да видиме, 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.

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

$$h[n] \equiv 0, \quad n < 0 \quad (5.8)$$

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

$$\sum_{n=-\infty}^{\infty} |h[n]| < \infty \quad (5.9)$$

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 можноста за реализација на линеарна фазна карактеристика. IIR филтрите пак можат да постигнат подобра амплитудна карактеристика за помал ред на филтерот.

IIR филтрите секогаш вклучуваат повратна врска во диференцната равенка (5.7), од каде што 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 е даден со:

$$y[n] + \sum_{i=1}^q a_i y[n-i] = \sum_{i=0}^p b_i x[n-i] \quad (5.10)$$

$$\left(1 + \sum_{i=1}^q a_i z^{-i}\right) Y(z) = \sum_{i=0}^p b_i z^{-i} X(z) \quad (5.11)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^p b_i z^{-i}}{1 + \sum_{i=1}^q a_i z^{-i}} \quad (5.12)$$

Тука, имплицитно претпоставуваме дека $a_0 = 1$, што во праксата секогаш се обезбедува преку нормализирање на останатите a коефициенти. Бројот на a коефициенти во повратната врска на IIR филтрите вообичаено се зема да биде еднаков со бројот на коефициенти b во директната врска, па важи $q = p = N$, каде што N се нарекува ред на филтерот.

Дополнително, преносната функција $H(z)$ исто така може да биде претставена преку нејзините **полови** и **нули** во z -рамнината:

$$H(z) = \frac{A \prod_{i=1}^p (1 - c_i z^{-1})}{\prod_{i=1}^q (1 - d_i z^{-1})} \quad (5.13)$$

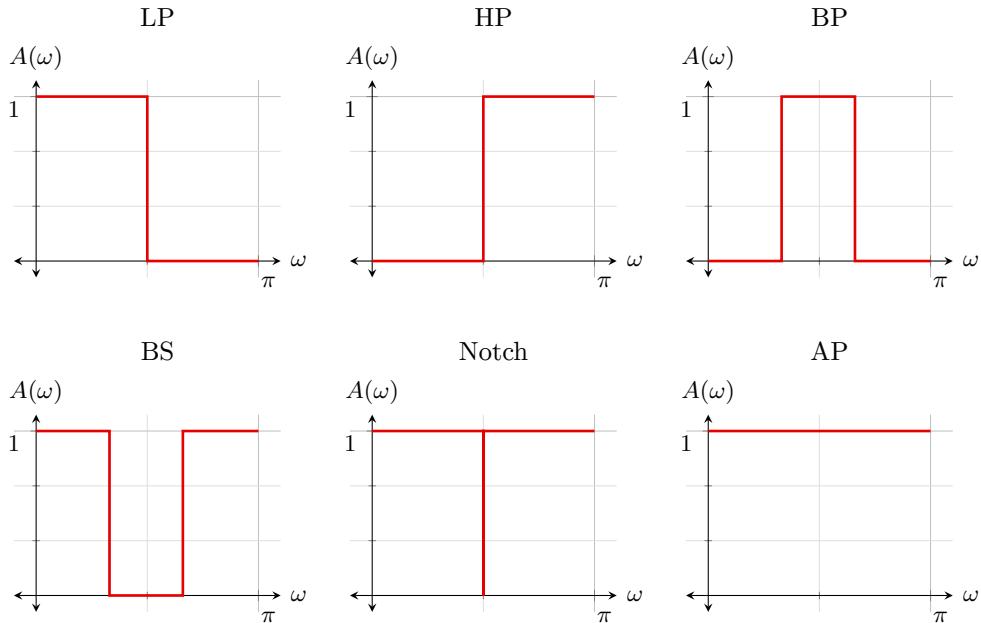
Типови филтри според амплитудната карактеристика

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

$$H(\omega) = \frac{\sum_{i=0}^p b_i e^{-j i \omega}}{1 + \sum_{i=1}^q a_i e^{-j i \omega}} \quad (5.14)$$

²Англ. *Finite Impulse Response*.

³Англ. *Infinite Impulse Response*.



Сл. 5.1: Идеални амплитудни карактеристики на шесте типови на филтри: нископропусен (LP), високопропусен (HP), пропусен на опсег (BP), непропусен на опсег (BS), непропусен на фреквенција (Notch) и сепропусен (AP).

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

$$H(\omega) = |H(\omega)|e^{j\angle H(\omega)} = A(\omega)e^{j\varphi(\omega)} \quad (5.15)$$

каде $A(\omega)$ е амплитудна карактеристика на филтерот, додека $\varphi(\omega)$ е неговата фазна карактеристика.

Според амплитудната карактеристика разликуваме шест типови на филтри прикажани на Слика 5.1:

- нископропусни (LP)⁴,
- високопропусни (HP)⁵,
- пропусни на опсег (BP)⁶,
- непропусни на опсег (BS)⁷,
- филтри непропусни на фреквенција (notch)⁸, и
- сепропусни (AP).⁹

⁴Англ. *low-pass*.

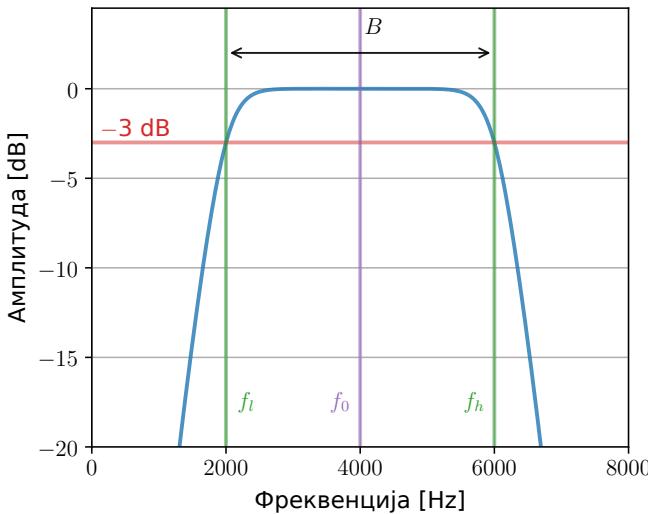
⁵Англ. *high-pass*.

⁶Англ. *band-pass*.

⁷Англ. *band-stop*.

⁸Англ. *notch*.

⁹Англ. *all-pass*.



Сл. 5.2: Фреквенциски опсег, централна фреквенција и гранични фреквенции на еден филтер пропусник на опсег.

Потребата од сепропусни филтри не е очигледна на прв поглед поради амплитудната карактеристика којашто изнесува 1 за сите фреквенции. Но, тие имаат корисна фазна карактеристика поради која наоѓаат примена кај системите за синтеза на дигитално ехо и реверберација, види Глава 6.

Фреквенциите за кои амплитудната карактеристика на филтерот паѓа од 1 на $1/\sqrt{2} \approx 0.707$ односно $20 \log 1/\sqrt{2} \approx -3 \text{ dB}$, се нарекуваат **гранични фреквенции** на филтерот. На тие фреквенции, моќноста на сигналот при филтрирањето ќе биде намалена на половина односно $10 \log 1/2 \approx -3 \text{ dB}$.

Со граничните фреквенции се дефинира **фреквенцискиот опсег B** на филтерот. На пример, за филтер пропусник на опсег со долнa f_l и горна гранична фреквенција f_h , за фреквенцискиот опсег ќе имаме:

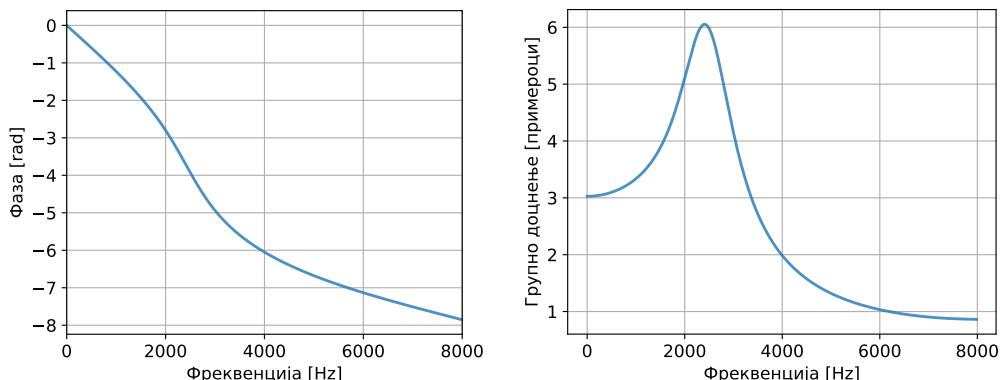
$$B = f_h - f_l \quad (5.16)$$

Дополнително, ја дефинираме и централната фреквенција на филтерот f_0 како:

$$f_0 = \frac{f_l + f_h}{2} \quad (5.17)$$

На Сл. 5.2 се прикажани фреквенцискиот опсег, централната фреквенција и долната и горната гранична фреквенција на еден филтер пропусник на опсег.

Конечно, со помош на фреквенцискиот опсег на филтерот и неговата централна фреквенција f_0 го дефинираме неговиот фактор на квалитет,



Сл. 5.3: Фазна карактеристика (лево) и групно доцнење (десно) на нископропусен IIR филтер со $f_l = 2500$.

односно *Q-фактор* (5.18). Така, филтрите со голем *Q-фактор* имаат тесен пропусен опсег, додека оние со мал *Q-фактор* имаат широк пропусен опсег.

$$Q = \frac{f_0}{B} \quad (5.18)$$

Фазна карактеристика на филтрите

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

$$\tau(\omega) = -\frac{d\varphi(\omega)}{d\omega} = \text{const} \quad (5.19)$$

Во тој случај, филтерот нема да внесе фазни изобличувања. Тоа значи дека компонентите на сигналот на различни фреквенции нема да бидат различно задоцнети, па ќе биде задржана нивната релативна поставеност без да дојде до нивно расејување, а со тоа и до задржување на компактноста на сигналот.

Строго линеарната фазна карактеристика не може да биде постигната со IIR или со аналогните филтри (Богданов, 1997), туку само со FIR филтри со симетричен импулсен одсив. На Сл. 5.3 е прикажана фазната карактеристика на нископропусен IIR филтер со $f_l = 2500$ и групното доцнење што тој го внесува за различни фреквенции. Важно е да се забележи дека иако фазната карактеристика е нелинеарна, во рамките на фреквенцискиот опсег на филтерот таа е приближно линеарна, па разликата во доцнење на овие фреквенциски компоненти е најмногу 3 примероци.

Типови FIR филтри со линеарна фазна карактеристика

Строго линеарната фазна карактеристика може да се постигне со FIR филтри чијшто импулсен одсив е симетричен во однос на средниот примерок $h[\frac{N-1}{2}]$:

$$h[n] = h[N - 1 - n] \quad (5.20)$$

или пак кога е антисиметричен во однос на него:

$$h[n] = -h[N - 1 - n] \quad (5.21)$$

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:

$$\begin{aligned} h[n] &= h[0]\delta[n] + h[1]\delta[n - 1] + \cdots + h[\frac{N-1}{2}]\delta[n - \frac{N-1}{2}] + \cdots \\ &\quad + h[1]\delta[n - (N-2)] + h[0]\delta[n - (N-1)] \end{aligned} \quad (5.22)$$

За неговата преносна функција имаме:

$$H(z) = h[\frac{N-1}{2}]z^{-\frac{N-1}{2}} + \sum_{i=0}^{\frac{N-3}{2}} h[i] \left(z^{-i} + z^{-(N-1-i)} \right) \quad (5.23)$$

$$= z^{-\frac{N-1}{2}} \left(h[\frac{N-1}{2}] + \sum_{i=0}^{\frac{N-3}{2}} h[i] \left(z^{-i+\frac{N-1}{2}} + z^{i-\frac{N-1}{2}} \right) \right) \quad (5.24)$$

$$= z^{-\frac{N-1}{2}} \sum_{i=0}^{\frac{N-1}{2}} a[n] \frac{z^n + z^{-n}}{2} \quad (5.25)$$

каде што:

$$a[n] = \begin{cases} h[\frac{N-1}{2}], & n = 0 \\ 2h[-n + \frac{N-1}{2}] & n = 1, 2, \dots, \frac{N-3}{2} \end{cases} \quad (5.26)$$

Од (5.25) може да се пресмета фреквенциската карактеристика на филтерот:

$$H(e^{j\omega}) = e^{-j\omega\frac{N-1}{2}} \sum_{n=0}^{\frac{N-1}{2}} a[n] \cos(n\omega) \quad (5.27)$$

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:

$$H(e^{j\omega}) = e^{-j\omega \frac{N-1}{2}} \sum_{n=1}^{\frac{N-1}{2}} b[n] \cos((n - \frac{1}{2})\omega) \quad (5.28)$$

каде што:

$$b[n] = 2 h[-n + \frac{N}{2}], \quad n = 1, 2, \dots, \frac{N}{2} \quad (5.29)$$

Од (5.28) 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:

$$H(e^{j\omega}) = e^{-j(\omega \frac{N-1}{2} - \frac{\pi}{2})} \sum_{n=1}^{\frac{N-1}{2}} c[n] \sin(n\omega) \quad (5.30)$$

каде што:

$$c[n] = 2 h[-n + \frac{N-1}{2}], \quad n = 1, 2, \dots, \frac{N-1}{2} \quad (5.31)$$

Од (5.30) 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:

$$H(e^{j\omega}) = e^{-j\omega \frac{N-1}{2}} \sum_{n=1}^{\frac{N}{2}} d[n] \sin((n - \frac{1}{2})\omega) \quad (5.32)$$

каде што:

$$d[n] = 2 h[-n + \frac{N}{2}], \quad n = 1, 2, \dots, \frac{N}{2} \quad (5.33)$$

Од (5.32) 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.

5.2 Дизајн и примена на FIR филтри

Постојат различни пристапи за дизајн на дигитални филтри. Трите најпознати методи за дизајн на FIR филтри се:

- метода на прозорци,
- дизајн заснован на DFT и
- оптимален дизајн на еднаквобранести филтри.

Дизајн на FIR филтер

Дизајнот на FIR филтри со методата на прозорци се заснова на апроксимација на идеалните амплитудни карактеристики на филтрите прикажани на Сл. 5.1 преку множење на нивниот импулсен одсив со прозорец. Ќе ја илустрираме методата за нископропусен филтер со гранична фреквенција ω_l . Идеалната фреквенциска карактеристика на ваков филтер е дадена со:

$$H_l(\omega) = \begin{cases} 1, & |\omega| \leq \omega_l \\ 0, & \omega_l < |\omega| \leq \pi \end{cases} \quad (5.34)$$

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

$$h_l(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_l(\omega) e^{j\omega n} d\omega = \frac{1}{2\pi} \int_{-\omega_l}^{\omega_l} e^{j\omega n} d\omega = \frac{\sin(n\omega_l)}{\pi n} \quad (5.35)$$

Овој импулсен одсив не може практично да се реализира поради тоа што тој има бесконечно траење и не е каузален. Поради тоа, во методата на прозорци се зема само дел од овој импулсен одсив преку негово множење со избран прозорец. И тука важат истите дискусиии од Главата 4.3 за влијанието на спектралните карактеристики на прозорците и компромисот помеѓу домените време и фреквенција.

За да ја имплементираме оваа метода ќе напишеме код кој ќе го конструира идеалниот филтер во Фурьеов домен во $n_{fft} = fs$ точки, а неговиот импулсен одсив ќе го пресметаме со употреба на IDFT и од него ќе 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 методата на прозорци.

```

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
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
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 # должина 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

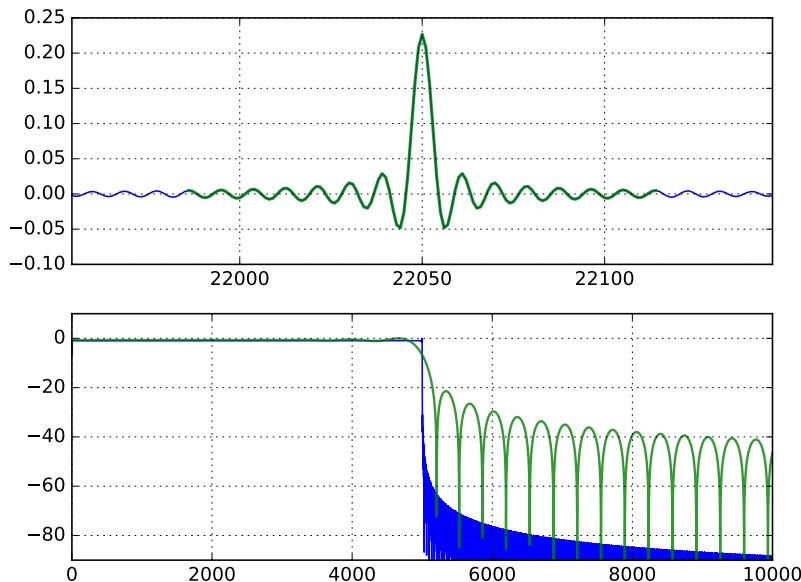
# %% плотирање
plt.figure()
plt.subplot(211)
plt.plot(h_l, linewidth=1, alpha=1)
plt.plot(n_win, h_rect, lw=2, alpha=.8)
plt.grid()
plt.subplot(212)
f, h_spec_l = da.get_spectrum(h_l, fs)
plt.plot(f, h_spec_l)

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 амплитудна карактеристика на идеалниот нископропусен филтер и онаа добиена со методата на прозорци со употреба на правоаголен прозорец се прикажани заедно со нивните импулсни одсиви на Сл. 5.4. Може да се види деградацијата на амплитудната карактеристика каде добиениот нископропусен филтер поради множењето на идеалниот импулсен одсив со прозорецот.

§ Дополнително. На Сл. 5.4 може да се види дека ни идеалниот импулсен одсив ја нема оригинално конструираната идеална фреквенциска карактеристика.



Сл. 5.4: Импулсни одсиви и амплитудни карактеристики на конструираниот идеален нископропусен филтер и оној добиен со методата на прозорци.

Ова е поради тоа што вистински идеалниот импулсен одсив има бесконечно многу примероци, а овој што ние го конструираме има f_s . Всушност и тој самиот како да сме го добили со множење на идеалниот одсив со правоаголен прозорец со должина од f_s примероци.

Филтрирање на аудиосигнал со FIR филтер

Во овој дел ќе исфилтрираме еден звучен сигнал со FIR филтер дизајниран со методата на прозорци која е 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
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
```

Глава 5. Филтри

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

# %% дизајнирај FIR филтер
order = 101
f_l = 2500
b = sig.firwin(order, 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()
```

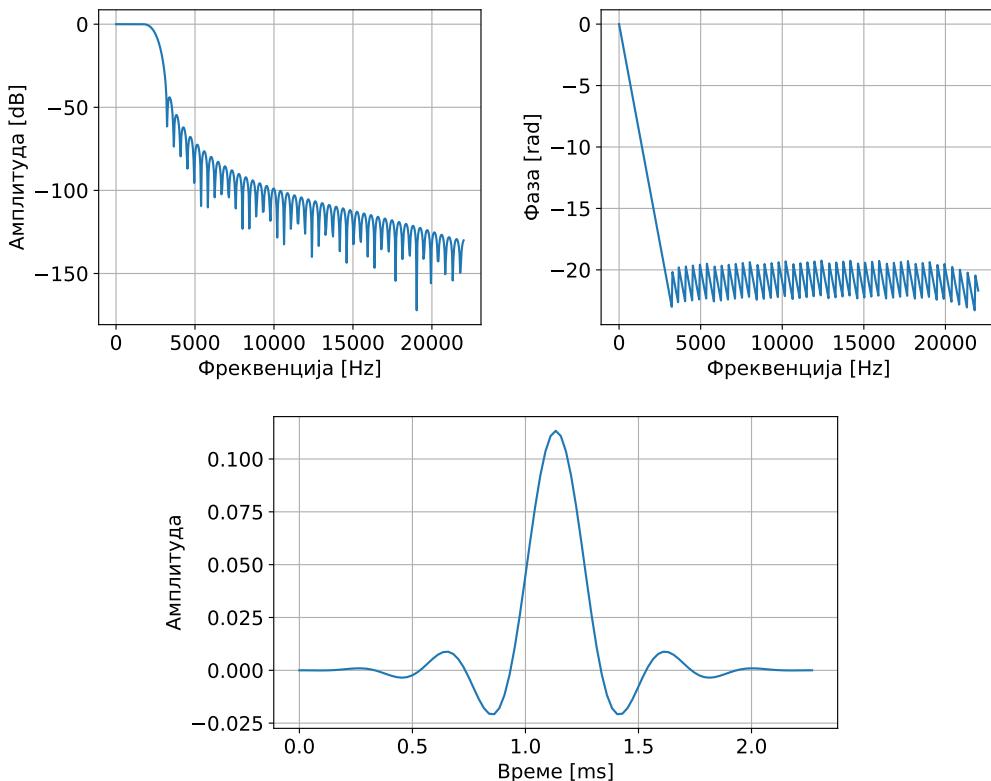
Карактеристиките на дизајнираниот FIR филтер со ред $N = 101$ се прикажани на Сл. 5.5.¹⁰

Според амплитудната карактеристика, може да видиме дека филтерот е нископропусен со гранична фреквенција f_l од 2,5 kHz. За разлика од вообичаената дефиниција за гранична фреквенција, при дизајнот на FIR филтри, амплитудната карактеристика на филтерот опаѓа од 1 на 0.5, а не на 0.707, што одговара на преполовување на амплитудата, а не на моќноста на сигналот. Соодветно вредноста на амплитудната карактеристика ќе биде $20 \log^{1/2} \approx -6$ dB наместо -3 dB.

На графиконот, може уште јасно да го видиме влијанието на употребениот Ханов прозорец врз амплитудната карактеристика на филтерот. Може да ја видиме и строго линеарната фазна карактеристика на филтерот, поради што тој внесува подеднакво доцнење на компонентите на сигналот долж сите фреквенции.

Конечно, можеме да го видиме неговиот импулсен одсив којшто поради каузалноста почнува од нулата со врвна амплитуда за $n = N - 1/2 = 50 \sim 1,13$ ms, поради што филтерот внесува поместување, односно доц-

¹⁰Тука да не заборавиме дека редот на филтерот треба да биде непарен за тој да биде од тип I, односно со него да можеме да реализираме произволна амплитудна карактеристика.

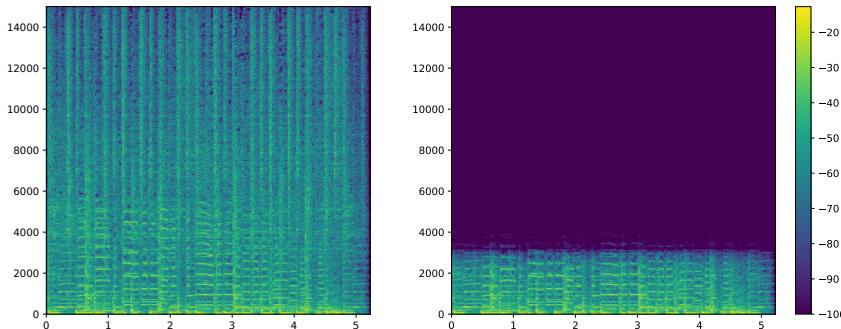


Сл. 5.5: Амплитудна (лево) и фазна карактеристика (десно) и импулсен одсив (долу) на дизајнираниот FIR филтер од 101-ви ред со употреба на Ханов прозорец.

нење во излезниот сигнал. Ова доцнење се зголемува со редот на филтерот, па изборот на редот претставува компромис помеѓу стрмнината на амплитудната карактеристика и доцнењето кое го внесува во аудиоланецот. Доцнењето може да претставува ограничувачки фактор при дизајнот на FIR филтри за системи што треба да работат во реално време, како на пример алгоритмите за аудиокомпресија вградени во системите за говорна комуникација помеѓу корисниците.

- ✓ **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 спектро-тимпорална содржина.



Сл. 5.6: Спектрограм на аудиосигналот Mara пред (лево) и по филтрирањето со LP FIR филтерот со гранична фреквенција од 2.500 Hz (десно).

```
# %% исфилтрирај го аудиосигналот и прикажи го спектрограмот
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')
wavefile.write(audio_path + 'mara_fir.wav', fs, wav_16)
os.system('play ' + audio_path + 'mara_fir.wav')
```

Спектрограмот на вчитаниот и на исфилтрираниот аудиосигнал се прикажани на Сл. 5.6. Може да видиме дека се работи за сложен аудиосигнал во кој се измешани периодични и апериодични звучни сигнали. Во исфилтрираниот аудиосигнал може да забележиме дека филтерот ефикасно ги отстранил сите фреквенциски компоненти на сигналот над 2,5 kHz.

5.3 Дизајн и примена на IIR филтри

За дизајн на IIR филтри најпознати методи се:

- Батерворт,
- Бесел,
- Чебишев и
- Елиптичен.

Дизајн на IIR филтер

Во овој учебник ќе се послужиме со `scipy.signal.iirfilter`, што со држи имплементации на дизајнот на IIR филтри според споменатите методи на Батерворт, Бесел, Чебишев, и методата 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
b, a = sig.iirfilter(order, f_l / (fs/2), btype='lowpass', ftype='butter')
f, h_w = sig.freqz(b, a, fs=fs)
h_amp = 20 * np.log10(np.abs(h_w))
h_ph = np.unwrap(np.angle(h_w))

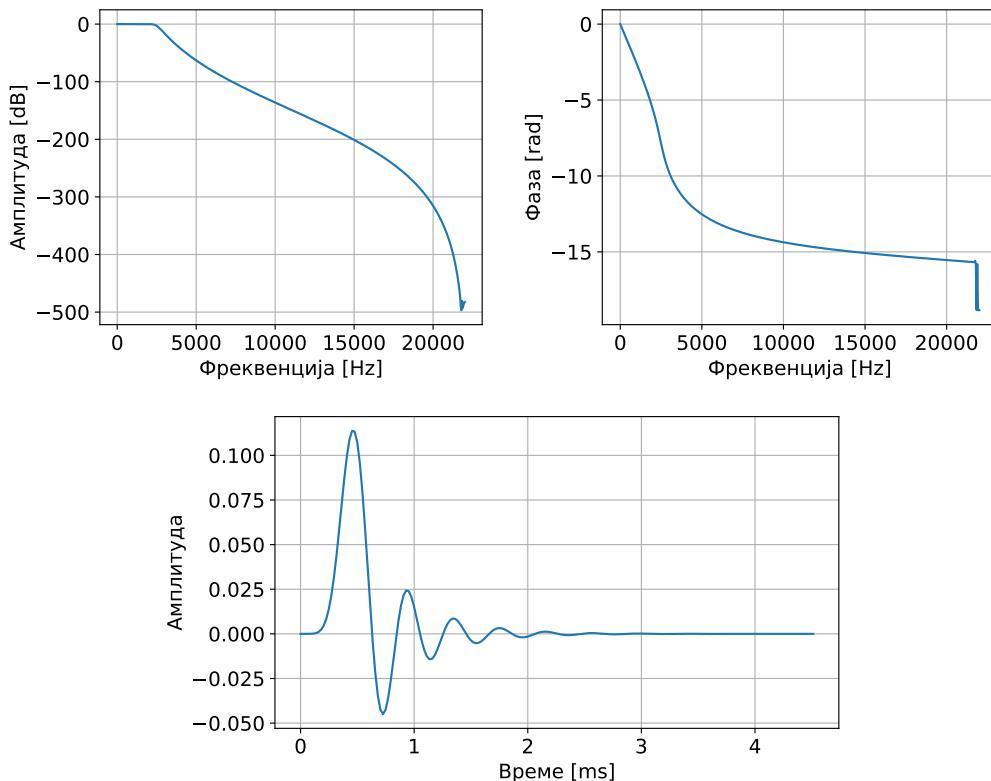
excite = np.zeros(100)
excite[0] = 1
h_n = sig.lfilter(b, a, excite)

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` и `a` коефициентите на дизајнираниот IIR филтер, двата $N+1$ на број, бидејќи импулсниот одсив не е даден директно од коефициентите на филтерот, како што тоа беше случај кај FIR филтрите. За да го добиеме, првин создаваме побуда во форма на Дираков импулс `excite` со кој го побудуваме филтерот за да го добиеме на излез неговиот импулсен одсив `h_n`.

На Сл. 5.7 се прикажани карактеристиките на дизајнираниот IIR филтер со ред $N = 10$. На Сл. 5.8 пак, се дадени амплитудните карактеристики на FIR филтерот од 101-ви ред споредени со IIR филтри од различен ред. Може да видиме дека IIR филтрите со многу помал ред постигнуваат споредлива стрмнина на амплитудната карактеристика во споредба со FIR филтрите. Како дополнителна предност, амплитудната карактеристика на IIR филтрите дизајнирани со Батерворт се строго монотони, па ги немаат бранувањата во амплитудната карактеристика што ги внесува методата со прозорци во дизајнот на FIR филтрите.

Поради стрмнината на нивната амплитудна карактеристика IIR филтрите често се користат во процесирањето на аудиосигналите, и покрај нивната нелинеарна фазна карактеристика, којашто може да се види на сликата. Сепак, треба да забележиме дека фазната карактеристика на IIR филтрите е речиси линеарна во опсегот на фреквенции кој го пропушта



Сл. 5.7: Амплитудна (лево) и фазна карактеристика (десно) и импулсен одсив (долу) на дизајнираниот IIR Батерворт филтер од 10-ти ред.

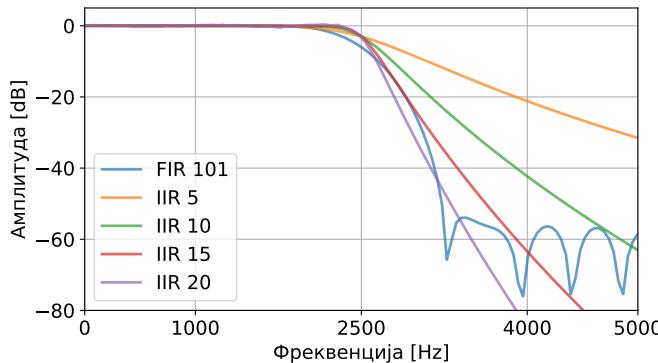
филтерот, а поизразена нелинеарност има надвор од овој опсег. Поради тоа, нејзината нелинеарност нема големо влијание врз процесираниот сигнал.

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 филтерот е споредливо со она на претходно дизајнираниот FIR филтер.

✓ **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?

Еквализација со примена на IIR филтри

Еквализацијата настанала како потреба да се израмни фреквенцијскиот одсив на системите за пренос на сигнали. Звукот низ својот пат од изворот до слушателот поминува низа аудиоуреди кои со своите неидеалности го менуваат неговиот првобитен спектар. Така, микрофонот



Сл. 5.8: Амплитудни карактеристики на FIR филтер од 101-ви ред и IIR филтри од различен ред.

ните немаат идеално рамна фреквенциска карактеристика, на пр. нивната чувствителност е помала на ниските фреквенции. Уште понерамномерна е амплитудната карактеристика, како и карактеристиката на зрачење на звучниците за различни фреквенции. Поради тоа, тие често се обично најслабата верига во аудиоланецот.¹¹

Ова „обојување“ на звукот може соодветно да се компензира со употреба на уред со инверзна фреквенциска карактеристика на онаа на каналот. На овој начин, преносната функција на уредот ја поништува нерамномерноста на каналот т.е. ефективно го израмнува истиот. Од тука доаѓа и името на процесот еквализација, односно на уредот – **еквализатор**.

Во аудиосистемите, во употреба е поширока дефиниција за еквализацијата. Според неа, еквализацијата е процес што служи за генерално обликување на спектарот на аудиосигналот, кое не мора да се стреми кон израмнување на фреквенциската карактеристика на системот. Така, еквализацијата може да послужи и за намерно истакнување, односно потиснување, на одредени делови од спектарот на аудиосигналите. Затоа еквализацијата вообичаено се третира како дигитален аудиоэффект, види Глава 6.

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 целокупната негова моќност останува иста. Ова го користат маркетинг агенциите за да постигнат максимална гласност на рекламиите на телевизија,

¹¹Дури и ако сите компоненти на аудиоланецот имаат идеални карактеристики, просторијата во која се слуша звукот ретко е акустички обработена, па нејзиниот одсив го менува спектралот на звучниот сигнал.

радио и интернет, без притоа да ги надминат дозволените нивоа.

Основниот начин на реализација на еден еквализатор се состои од пријема на повеќе филтри пропусници на опсег, ускладени да го препокријат целиот звучен опсег. Во аналогната техника бројот на филтри е ограничен со цената на уредот, додека во дигиталната техника со процесирачката моќ. Затоа добирите аналогни аудиосистеми најчесто вклучуваат едноставна бас и требл¹² еквализација, додека дигиталните аудиопреслушувачи вообичаено поддржуваат 6 и повеќе-канална еквализација. Притоа дигиталните филтри се најчесто од IIR тип од 4-ти ред, за намалување на комплексноста.

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')
# требл
f_h = 4000 / (fs/2)
b_t, a_t = sig.iirfilter(n, f_h, btype='high', ftype='butter')

# преносни функции
f, H_b = sig.freqz(b_b, a_b, fs=fs)
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
```

¹²Англ. *bass* и *treble*.

```

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)

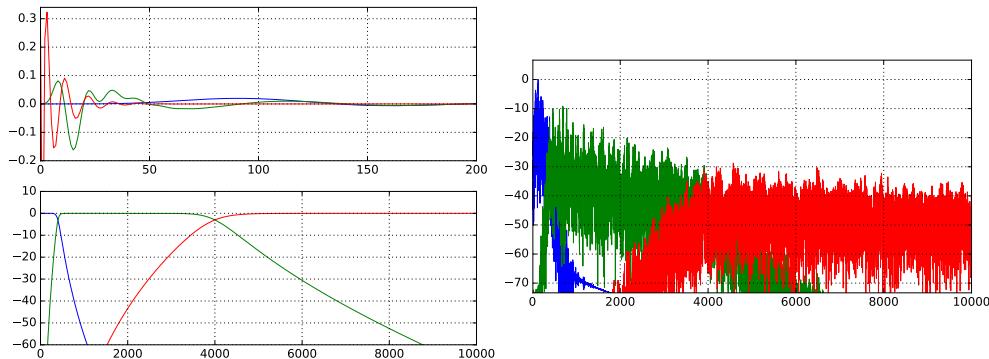
# %% засилување
g_bass = -20 # во dB
g_mid = 0 # во dB
g_treble = 20 # во dB
g_b = 10**((g_bass/20))
g_m = 10**((g_mid/20))
g_t = 10**((g_treble/20))
wav_out = g_b*wav_bass + g_m*wav_mid + g_t*wav_treble

# %% 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 и амплитудните карактеристики на трите дизајнирани филтри, како и подопсезите издвоени од аудиосигналот со секој од филтрите се прикажани на Сл. 5.9. Со менување на засилувањето на бас, средни и високи фреквенции може да се добие произволна промена на



Сл. 5.9: Импулсни одсиви и фреквенциски карактеристики на трите IIR филтри од дизајнираниот еквализатор и добиени подопсези од аудиосигналот.

спектарот на аудиосигналот, а со тоа и поинаков звучен ефект.

IIR филтри непропусни на фреквенција

Една од примените на дигиталните филтри за која неприкосновени се IIR филтрите е потиснувањето на одредена фреквенција во аудиосигналите. Најчесто потребата за ваков тип на процесирање во аудиосистемите се јавува кога треба да се потисне хармоничниот шум од градската мрежа на 50 Hz, познат како **брум** или **зуење**.¹³

Брумот може да влезе во аудиосигналот пред неговата дигитализација, најчесто поради електромагнетни интерференции во каблите што го пренесуваат аудиосигналот до аудиоуредите чии влезни кола се со висока импеданса или големо засилување. Уште еден можен извор на брум се неправилности во масата на аудиоуредите, на пример кога колата низ кои минува аудиосигналот ја делат масата со кола низ кои течат големи струи како колата за напојување. Притоа, вообичаено брумот има изразен втор хармоник на 100 Hz, но и трет или четврт хармоник на 150 Hz односно 200 Hz.

Да прикажеме временски облик и спектар на еден аудиосигнал со изразен брум од градска мрежа.¹⁴ Бидејќи аудиосигналот е во ogg формат (види Глава 7), за да го прочитаме ќе го искористиме пакетот librosa¹⁵, кој нуди интерфејс кон библиотеката soundfile¹⁶ за вчитување на аудиодатотеки.¹⁷

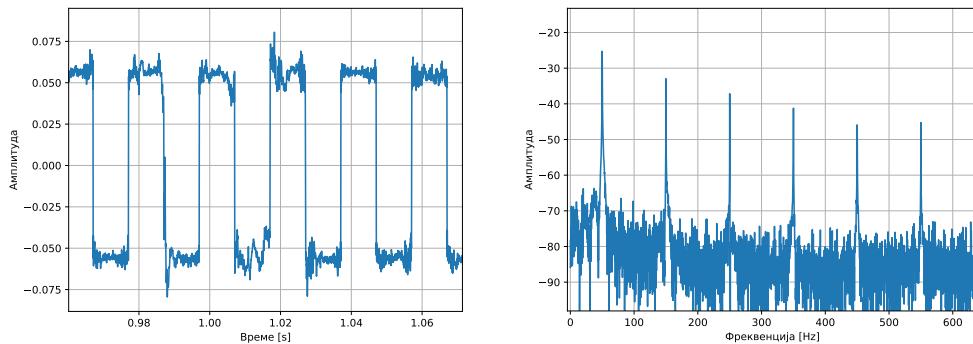
¹³Wikipedia: Mains hum https://en.wikipedia.org/wiki/Mains_hum

¹⁴BPK, Public domain, via Wikimedia Commons https://commons.wikimedia.org/wiki/File:50Hz_hum_square.ogg

¹⁵<https://librosa.org/>

¹⁶<https://github.com/bastibe/python-soundfile>

¹⁷Притоа, при повикување на функцијата за вчитување на аудиосигналот `lr.load` експлицитно ќе му доделиме вредност **None** на параметарот `sr` за да ја зачуваме неговата



Сл. 5.10: Временски облик и спектар на аудиосигналот со изразен брум од градска мрежа.

```
import numpy as np
from matplotlib import pyplot as plt
import librosa as lr
import da

hum, fs = lr.load('./50Hz_hum_square.ogg', sr=None)
plt.figure()
t = np.arange(hum.size) / fs
plt.plot(t, hum)
plt.grid(True)

f, hum_spec = da.get_spectrum(hum, fs)
plt.figure()
plt.plot(f, hum_spec)
```

Исцртаниот временски облик и спектарот на аудиосигналот се прикажани на Сл. 5.10. Од спектарот е видливо дека основната фреквенција на брумот е 50 Hz, а присутни се и неговите непарни хармоници на 150 Hz, 250 Hz, 350 Hz итн. Во овој случај, станува збор за екстремно изразен брум со ниво повисоко од она на аудиосигналот, поради што основната синусоида е пресечена, па таа повеќе наликува на правоаголен сигнал.

За отстранување на брумот, потребно е фреквенцискиот опсег на филтерот непропусник да биде што потесен, а филтерот да биде со што поголемо слабеење. Со други зборови, notch филтрите треба да имаат што поголем *Q*-фактор (5.18). Поради тоа, овие типови на филтри се нарекуваат **филтри непропусни на фреквенција** или **notch филтри**. Дополнително, ако во аудиосигналот има брум со повеќе хармоници, ќе треба да се дизајнира банка на notch филтри за да се потиснат фреквенциите на коитие се наоѓаат.

оригинална фреквенција на земање примероци. Во спротивно, *librosa* ќе го ресемплира сигналот на 22,05 kHz.

Преносна функција на notch филтер

Преносната функција на идеален notch филтер е дадена со:

$$H(e^{j\omega}) = \begin{cases} 0, & \text{за } \omega = \omega_0 \\ 1, & \text{за } \omega \neq \omega_0 \end{cases} \quad (5.36)$$

Оваа преносна функција ќе ја апроксимираме со IIR филтер од прв ред:

$$H(z) = \frac{1 + bz^{-1}}{1 - az^{-1}} \quad (5.37)$$

Притоа, ќе ја представиме преносната функција преку нејзините **полови и нули** (5.13):

$$H(z) = \frac{1 - cz^{-1}}{1 - dz^{-1}} \quad (5.38)$$

За да постигнеме потиснување на фреквенцијата f_0 , ќе избереме нулата и полот да бидат:

$$c = e^{j\omega_0} \quad (5.39)$$

$$d = re^{j\omega_0} \quad (5.40)$$

каде што $r \approx 1$.

Тогаш, амплитудната карактеристика на филтерот ќе биде дадена со:

$$|H(e^{j\omega})| = \left| \frac{1 - e^{-j(\omega - \omega_0)}}{1 - re^{-j(\omega - \omega_0)}} \right| \quad (5.41)$$

па кога $\omega = \omega_0$, броителот на преносната функција е нула, а со тоа $|H(e^{j\omega_0})| = 0$. Од друга страна, кога $\omega \neq \omega_0$, бидејќи r е близку до 1, броителот и именителот се речиси идентични и се кратат, па $|H(e^{j\omega})| \approx 1$.

Проблемот со оваа реализација е тоа што и c и d се комплексни броеви, па дури и влезниот сигнал да е реален, излезниот сигнал ќе биде комплексен. За да добиеме филтер со реални коефициенти, треба да искористиме филтер од втор ред со комплексно конјугирани полови и нули, односно $c_2 = c_1^* = e^{-j\omega_0}$ и $d_2 = d_1^* = re^{-j\omega_0}$.

Тогаш, преносната функција на филтерот ќе биде дадена со:

$$H(z) = \frac{(1 - c_1 z^{-1})(1 - c_1^* z^{-1})}{(1 - d_1 z^{-1})(1 - d_1^* z^{-1})} = \frac{1 - 2 \cos \omega_0 z^{-1} + z^{-2}}{1 - 2r \cos \omega_0 z^{-1} + r^2 z^{-2}} \quad (5.42)$$

а тој ќе биде дефиниран со следната диференцна равенка:

$$y[n] = x[n] - 2 \cos \omega_0 x[n-1] + x[n-2] + 2r \cos \omega_0 y[n-1] - r^2 y[n-2] \quad (5.43)$$

Ширината на фреквенцискиот опсег на филтерот B во радијани по примерок, зависи обратно пропорционално од r (5.44), па Q -факторот на филтерот зависи пропорционално од r .

$$B = -2 \ln r [\text{rad}] = -2 \ln r \cdot \frac{f_s}{2\pi} [\text{Hz}] \quad (5.44)$$

$$Q = \frac{f_0}{B} = -\frac{2\pi f_0}{f_s \ln r} = \frac{f_0}{-2 \ln r \cdot \frac{f_s}{2\pi}} \quad (5.45)$$

$$r = e^{-\frac{1}{2} \frac{f_0}{Q}} \cdot \frac{2\pi}{f_s} \quad (5.46)$$

На пример, за $r = 0.99$, $B = 0.02$ [rad] = 25,6 [Hz], па $Q = 1,95$ за $f_0 = 50$ Hz, додека за $r = 0.9$, $B = 268.3$ [Hz] и $Q = 0.19$.

Конечно, амплитудната карактеристика на notch филтерот ќе биде дадена со:

$$|H(e^{j\omega})| = \begin{cases} 0, & \text{за } \omega = \omega_0 \\ \frac{1}{\sqrt{2}}, & \text{за } \omega = \omega_0 \pm \ln r \\ \approx 1, & \text{за } \omega \ll \omega_0 \text{ или } \omega \gg \omega_0 \end{cases} \quad (5.47)$$

Дизајн и примена на notch филтер

За да ја илустрираме употребата на notch филтри за елиминирање на брумот во аудиосигналите, ќе вчитаме еден звучен запис и ќе му додадеме симулиран сигнал на брум составен од 3 хармоници.

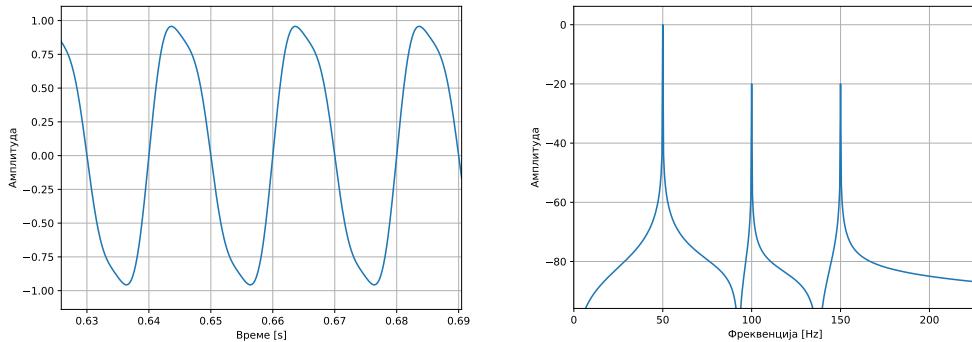
```
# %% вчитај аудиосигнал
from scipy.io import wavfile

audio_path = './audio/Pato_8k.wav'
fs, wav = wavfile.read(audio_path)
wav = wav / 2**15
t = np.arange(wav.shape[0]) / fs
os.system('play ' + audio_path)

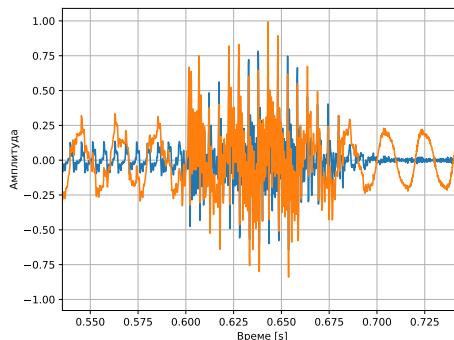
# %% симулирај брум
brum = (
    np.sin(2 * np.pi * 50 * t)
    + .1 * np.sin(2 * np.pi * 100 * t)
    + .1 * np.sin(2 * np.pi * 150 * t)
)
plt.plot(t, brum)

f, brum_spec = da.get_spectrum(brum, fs)
plt.figure()
plt.plot(f, brum_spec)
```

На Сл. 5.11 се прикажани временскиот облик и спектарот на симулираниот брум.



Сл. 5.11: Временски облик и спектар на симулираниот брум и временскиот облик и спектарот на аудиосигналот со додаден брум.



Сл. 5.12: Временски облик на аудиосигналот со додаден брум.

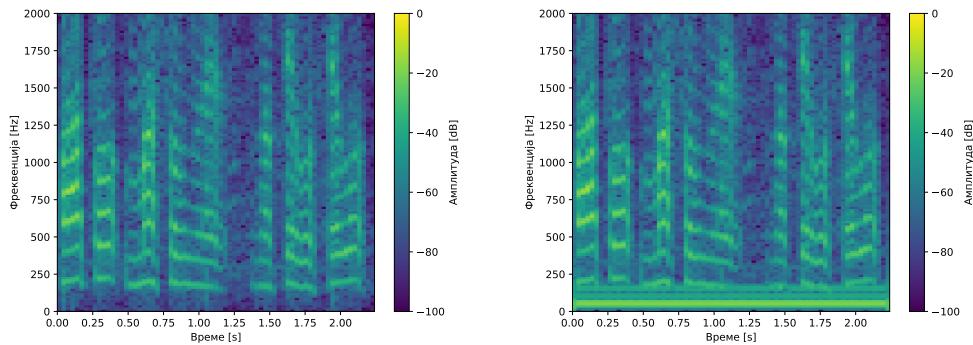
Да го додадеме симулираниот брум на аудиосигналот и да го преслушаме резултатот.

```
# %% додади го брумот на аудиосигналот
wav_brum = wav + .2 * brum
wav_brum = da.normalise(wav_brum)
wavefile.write('audio/Pato_brum.wav', fs, np.int16(wav_brum * 2**15))
os.system('play audio/Pato_brum.wav')

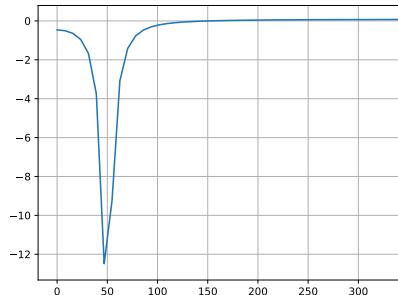
# %% исцртај го аудиосигналот со додаден брум
plt.figure()
plt.plot(t, wav)
plt.plot(t, wav_brum)

da.get_spectrogram(wav, fs, 512, plot=True)
da.get_spectrogram(wav_brum, fs, 512, plot=True)
```

На Сл. 5.12 е прикажан временскиот облик на аудиосигналот со додаден брум, додека на Сл. 5.13 се прикажани спектрограмот на аудиосигналот без и со додаден брум.



Сл. 5.13: Спектрограм на аудиосигналот без (лево) и со додаден брум (десно).



Сл. 5.14: Амплитудна карактеристика на дизајнираниот notch филтер.

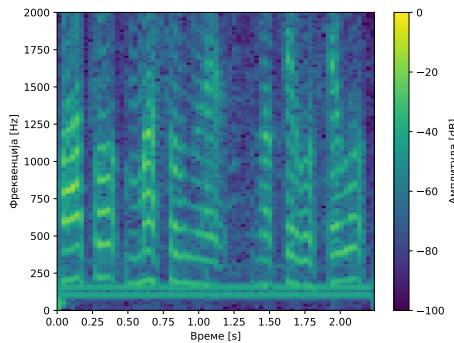
За реализација на notch филтер ќе ја имплементираме диференцната равенка дадена со (5.43):

```
r = 0.99
f0 = 50
w0 = f0 * 2 * np.pi / fs
b_notch = [1, -2 * np.cos(w0), 1]
a_notch = [1, -2 * r * np.cos(w0), r**2]

# прикажи ја амплитудната карактеристика на филтерот
f, notch_spec = sig.freqz(b_notch, a_notch, fs=fs)
plt.figure()
plt.plot(f, 20 * np.log10(np.abs(notch_spec)))
plt.grid(True)
```

Амплитудната карактеристика на дизајнираниот notch филтер е прикажана на Сл. 5.14.

На крајот останува да го примениме филтерот на аудиосигналот со додаден брум и да го видиме и чуеме и резултатот.



Сл. 5.15: Спектрограм на аудиосигналот по примената на notch филтерот.

```
# %% филтрирање
wav_notch = sig.lfilter(b_notch, a_notch, wav_brum)
da.get_spectrogram(wav_notch, fs, 512, plot=True)

# %% преслушување
wavefile.write('audio/Pato_notch.wav', fs, np.int16(wav_notch * 2**15))
os.system('play audio/Pato_notch.wav')
```

Може да слушнеме дека брумот е успешно потиснат! Тоа може да го видиме и во спектрограмот на филтрираниот аудиосигнал на Сл. 5.15.

✓ Задача за час. Дизајнирајте банка на notch филтри за потиснување на сите хармоници на додадениот брум од аудиосигналот!

§Дополнително. Користејќи ја диференцната равенка за notch филтерот (5.43), како и поврзаноста на Q -факторот со коефициентот r (5.46), можеме да создадеме функција за дизајн на notch филтер со дадена централна фреквенција и Q -фактор. Ваква функција веќе постои во SciPy во `scipy.signal.iirnotch`.

```
def design_notch(f0, Q, fs):
    '''Design a notch filter.

    Parameters
    -----
    f0 : float
        Center frequency of the notch filter [Hz].
    Q : float
        Q-factor of the notch filter.
    fs: int
        Sampling frequency [Hz].
```

```
Returns
-----
b : array_like
    Numerator coefficients of the notch filter.
a : array_like
    Denominator coefficients of the notch filter.
...
w0 = f0 * 2 * np.pi / fs
r = np.exp(- 0.5 * f0 / Q) * 2 * np.pi / fs
b = [1, -2 * np.cos(w0), 1]
a = [1, -2 * r * np.cos(w0), r**2]
return b, a
```


Глава 6

Дигитални аудиоэффекти

Аудиоэффектите, или звучните ефекти, се вештачки произведени или видоизменети звуци или звучни процеси што се употребуваат за нагласување на уметничката или на друг вид 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 што се применува врз звучните сигнали, а не и нивната содржина. Ова толкување ќе го задржиме и во овој учебник. Звучните ефекти може грубо да ги поделиме на ефекти базирани на процесирање на сигналот во временски домен, и ефекти базирани на процесирање во трансформациски, на пр. фреквенцијски, домен (Zölzer, 2011).

6.1 Аудиоэффекти во временски домен

Основните дигитални аудиоэффекти се базираат на обработката на аудиосигналот во временски домен. Такви се оние што вршат обработка на неговата амплитуда, па се нарекуваат **амплитудни аудиоэффекти**, како и оние што се базираат на доцнење.

Амплитудни аудиоэффекти

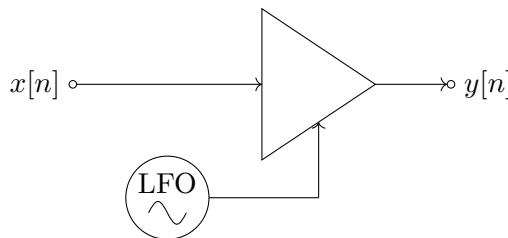
Најпознати амплитудни аудиоэффекти се:

- постепено засилување и втишување² – се употребуваат на почетокот односно на крајот од аудиосигналите,
- прелевање³ – се употребува за надоврзување на два аудиосигнали,

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

²Англ. *fade-in* и *fade-out*.

³Англ. *cross-fade*.



Сл. 6.1: Блок-дијаграм на тремоло ефектот.

- **тремоло** – простопериодична амплитудна модулација на аудиосигналот,
- **пинг-понг** – периодично менување на просторната локација на изворот во дво- или повеќеканален звук,
- **дисторзија** – пресекување или потиснување на врвните амплитуди со употреба на систем со нелинеарна амплитудна карактеристика,
- **динамичка компресија** – процесирање на аудиосигналот со нелинеарна амплитудна карактеристика која нелинеарно ги потиснува врвните амплитуди со цел да се засили целиот сигнал до максималното возможно ниво, итн.

✓ **Задача за дома.** Со помош на досегашните познавања на обработката на аудиосигналите имплементирајте го ефектот на дисторзија.

Тремоло

Тремоло ефектот се базира на примена на простопериодичен **нискофреквенциски осцилатор (LFO)**⁴, на пр. од неколку Hz, за амплитудна модулација на аудиосигналот. Блок-дијаграмот на тремоло ефектот е прикажан на Сл. 6.1. Излезниот сигнал се добива со:

$$y[n] = e[n] \cdot x[n] \quad (6.1)$$

каде што $x[n]$ е аудиосигналот, а $e[n]$ е анвелопата со која тој се модулира:

$$e[n] = 1 + m \cdot \sin(2\pi f_m n) \quad (6.2)$$

Тука, f_m е фреквенцијата на модулација, а m е амплитудата на модулацијата која е во опсегот $(0, 1]$.

Да го имплементираме овој ефект во Python. За разлика од досега, за преслушување на аудиосигнали овој пат ќе го искористиме пакетот **sounddevice**⁵.

⁴Англ. *Low Frequency Oscillator*.

⁵sounddevice е Python модул што овозможува преслушување и снимање на звук во NumPy низи преку употреба на PortAudio библиотеката. <https://python-sounddevice.readthedocs.io/en/0.4.1/>

```

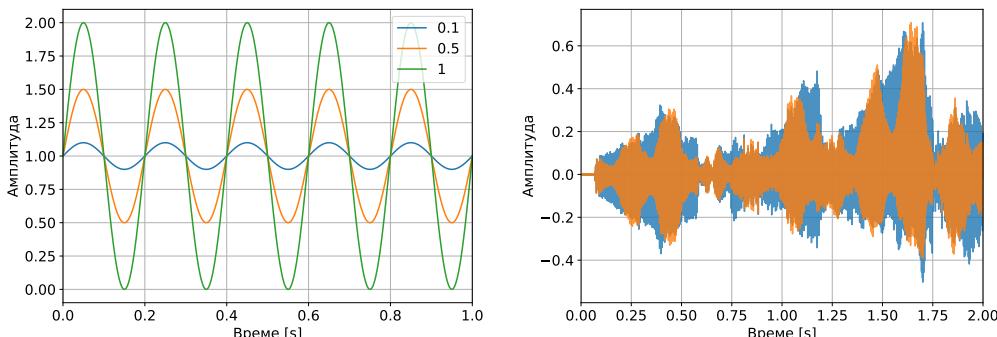
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
import sounddevice as sd

# %% вчитај го аудиосигналот
file_name = "./audio/Donco.wav"
fs, wav = wavfile.read(file_name)
wav = wav / 2**15
wav = da.normalise(wav, -3)
t = np.arange(wav.size) / fs
sd.play(wav, fs)

# %% имплементирај тремоло на сигналот
amp = 0.5
f = 5
tremolo = 1 + amp * np.sin(2 * np.pi * f * t)
wav_tremolo = wav * tremolo
wav_tremolo = da.normalise(wav_tremolo, -3)
sd.play(wav_tremolo, fs)

```

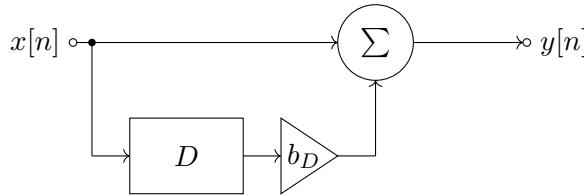
На Сл. 6.2 е прикажан изгледот на LFO сигналот за различни амплитуди на модулацијата, како и аудиосигналот пред и по примената на тремоло ефектот. Амплитудната модулација може јасно да се види и да се чуе во излезниот сигнал со тремоло.



Сл. 6.2: Пример на LFO сигнал со фреквенција од $f_m = 5 \text{ Hz}$ и различни амплитуди на модулацијата (лево) и изглед на излезниот сигнал со тремоло ефектот за $m = 0.5$ (десно).

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



Сл. 6.3: Блок-дијаграм на систем за генерирање ехо со FIR филтер.

почесто со софтверска обработка на сигналите.

Системот што му додава задоцната верзија на аудиосигналот даден на Сл. 6.3 може да се претстави со следната равенка:

$$y[n] = x[n] + b_D x[n - D] \quad (6.3)$$

Може да видиме дека ова претставува специјален случај на општата диференцна равенка (5.7). Тука D е доцнењето на сигналот во примероци, b_D е неговото слабеење, а земено е дека $b_0 = 1$.

Импулсниот одсив и преносната функција на овој систем се дадени со:

$$h[n] = 1 + b_D \delta[n - D] \quad (6.4)$$

$$H(z) = \frac{Y(z)}{X(z)} = 1 + b_D z^{-D} \quad (6.5)$$

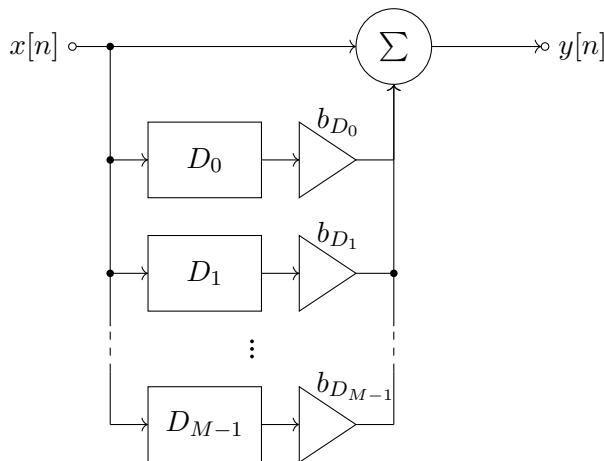
Поради конструктивното и деструктивно собирање на задоцнетата верзија од аудиосигналот со неговиот оригинал за различни фреквенции, фреквенциската карактеристика на системот ќе има низа максимуми и минимуми распоредени на еднакво растојание. Затоа овие системи се нарекуваат уште и **чешлести филтри**. Ваквата амплитудна карактеристика на системот е неповолна за обработка на аудиосигналите поради спектралните изобличувања што ќе ги внесе.

Систем што додава повеќекратни задоцнети верзии на аудиосигналот е прикажан на Сл. 6.4. Тој може да се реализира со додавање на коефициенти во диференцната равенка на FIR филтерот (6.3):

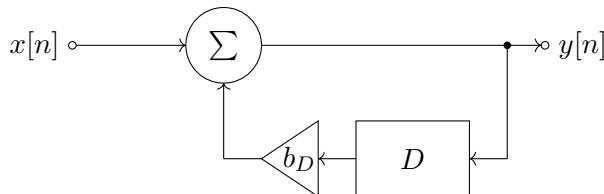
$$y[n] = x[n] + b_{D_0} x[n - D_0] + b_{D_1} x[n - D_1] + \dots + b_{D_{M-1}} x[n - D_{M-1}] \quad (6.6)$$

$$h[n] = 1 + b_{D_0} \delta[n - D_0] + b_{D_1} \delta[n - D_1] + \dots + b_{D_{M-1}} \delta[n - D_{M-1}] \quad (6.7)$$

$$\begin{aligned} H(z) &= 1 + b_{D_0} z^{-D_0} + b_{D_1} z^{-D_1} + \dots + b_{D_{M-1}} z^{-D_{M-1}} \\ &= 1 + \sum_{m=0}^{M-1} b_{D_m} z^{-D_m} \end{aligned} \quad (6.8)$$



Сл. 6.4: Блок-дијаграм на систем за генерирање повеќекратно ехо со FIR филтер.



Сл. 6.5: Блок-дијаграм на систем за бескрајно ехо со IIR филтер.

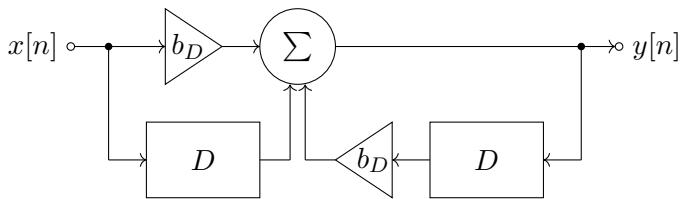
Тука, со M е означен бројот на задоцнети верзии додадени во сигналот одредени со нивните доцнења D_m и коефициенти на слабеење b_{D_m} .

Уште еден пристап е да употребиме IIR филтер кој по својата природа има бескрајно долг импулсен одсив (6.9) претставен на Сл. 6.5. Со него можеме да создадеме бескрајно многу задоцнети верзии на сигналот кои се распоредени на мултипли од основното доцнење $m \times D$. Слабеењето на овие задоцнети верзии ќе биде степен од слабеењето на првото ехо a_D^m .

$$y[n] = x[n] - a_D y[n - D] \quad (6.9)$$

$$H(z) = \frac{1}{1 + a_D z^{-D}} \quad (6.10)$$

Како и претходно, фреквенциската карактеристика на овој IIR филтер има чешлест облик, па внесува изобличувања во излезниот аудиосигнал. Но, овојпат максимумите се на местата на минимумите кај FIR филтерот и обратно – минимумите кај IIR филтерот се на локациите на максимумите на FIR филтерот, како што може да се види на Сл. 6.7. Ова можеме да го искористиме за дизајн на систем што би генерирал задоцнети верзии од аудиосигналот, а притоа би имал рамна фреквенциска карактеристика која не би внесувала изобличувања во излезниот сигнал.



Сл. 6.6: Блок-дијаграм на систем за бескрајно ехо со АР филтер.

Овој систем може да се направи преку едноставна комбинација на комплементарните FIR и IIR системи како што е прикажано на Сл. 6.6 според:

$$y[n] = b_D x[n] + x[n - D] - b_D y[n - D] \quad (6.11)$$

$$H(z) = \frac{b_D + z^{-D}}{1 + b_D z^{-D}} \quad (6.12)$$

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 ехо

Дигиталното ехо претставува ефект во кој на аудиосигналот му се додаваат една или повеќе задоцнети верзии, при што меѓу тие задоцнети верзии и оригиналниот звук слушателот може да направи јасна разлика. Во случај кога имаме повеќе задоцнети верзии велиме дека се работи за **повеќекратното ехо**. За имплементација на ехото во Python, ќе ги реализираме системите за доцнење базирани на FIR и IIR филтрите кои ги воведовме погоре, како и филтрите сепропусници. Кога ќе употребиме IIR филтер или филтер сепропусник, можеме да кажеме дека системот ќе генерира **бескрајно ехо**.

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

fs = 22050
# FIR ехо
dt = 0.5 # s
d = int(dt * fs) # примероци
b_d = 0.5
b_fir = np.zeros(d + 1)
b_fir[0] = 1
```

```

b_fir[d] = b_d

# повеќекратно FIR ехо
d0t = 0.2 # s
d0 = int(d0t * fs) # примероци
d1t = 0.3 # s
d1 = int(d1t * fs) # примероци
b_fir_mul = cp.copy(b_fir)
b_fir_mul[d0] = 0.4
b_fir_mul[d1] = 0.3

# бескрајно ехо
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 добиените системи.

```

# FIR
f, h_fir = sig.freqz(b_fir, [1], fs=fs)
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))

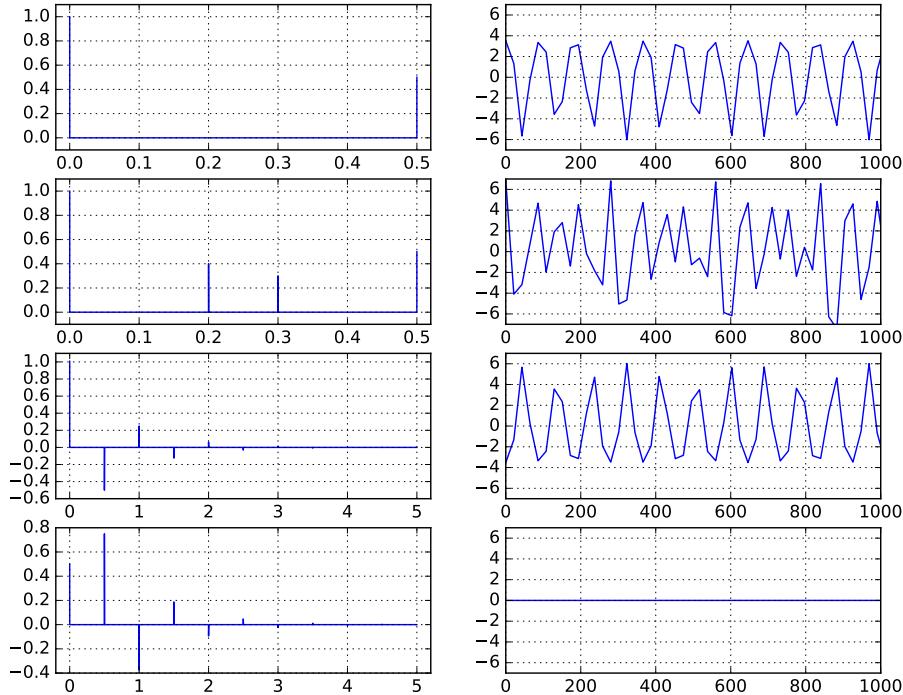
# IIR
x = np.zeros(5 * fs)
x[0] = 1
h_iir = sig.lfilter([1], a_iir, x)
w, h_iir = sig.freqz([1], a_iir)
h_iir = 20 * np.log10(np.abs(h_iir))

# 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. 6.7](#). 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 процеси-



Сл. 6.7: 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 бесконечно ехо со филтер сепропусник.

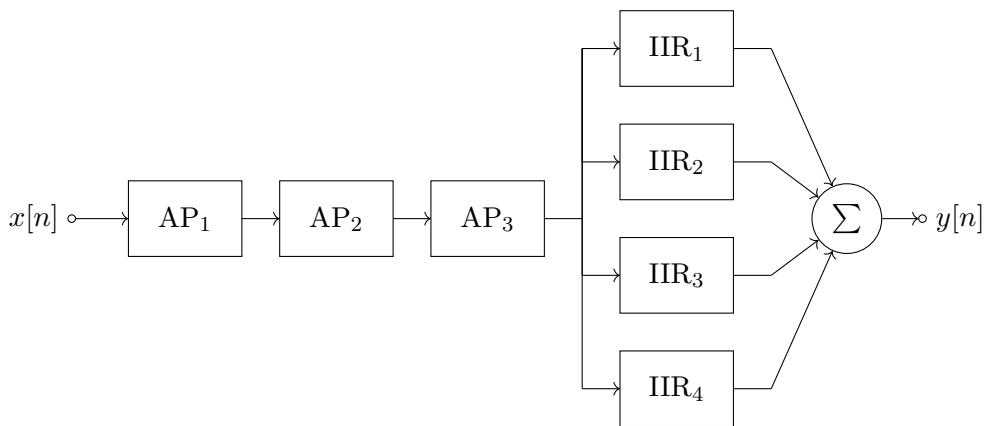
рање на еден аудиосигнал и да ги чуеме резултатите.

```
# %% 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)

# %% 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 реверберација или ревербот претставува ефект во кој се генерираат и надодаваат 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 неа



Сл. 6.8: Блок-дијаграм на еден Шредеров ревербератор.

можеме да заклучиме во каков вид на просторија го слушаме или е снимен звучниот сигнал. Реверберијата може да се реализира со едноставно паралелно или сериско надоврзување на повеќе системи за генерирање на еднократно или повеќекратно ехо со густо распоредени доцнења.

Пионер во ова поле бил Манфред Шредер⁶ кој во 1960-тите ги предложил првите системи за генерирање реверберија составени од каскади на сепропусни филтри и банки на IIR филтри. Еден таков ревербератор е прикажан на Сл. 6.8.⁷ Параметрите на секој од филтрите се дадени во Табела 6.1.

Табела 6.1: Параметри на филтрите во Шредеровиот ревербератор.

Филтер	b_D/a_D	D [ms]
AP ₁	0.7	13.88
AP ₂	0.7	4.52
AP ₃	0.7	1.48
IIR ₁	0.773	67.48
IIR ₂	0.802	64.04
IIR ₃	0.753	82.12
IIR ₄	0.733	90.04

✓ **Задача за дома.** Имплементирајте го Шредеровиот ревербератор прикажан на Сл. 6.8 во Python и применете го на некој аудиосигнал.

⁶Wikipedia: Manfred Schroeder. https://en.wikipedia.org/wiki/Manfred_R._Schroeder

⁷Schroeder Reverberators https://ccrma.stanford.edu/~jos/pasp/Schroeder_Reverb_Erators.html

Други дигитални ефекти базирани на доцнење

Други аудиоэффекти базирани на доцнење или сепропусни филтри се: **хор**, **фленц** и **фејзер** ефектите. Ефектот **хор**⁸ 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 забележливо раздвоени.

Фленц⁹ ефектот се добива преку примена на FIR систем за еднократно ехо кое има променливо доцнење. Аудитивно овој ефект е сличен на фејзерот описан во Главата 6.2, кој пак може да се реализира со каскада од променливи сепропусни филтри. Разликата меѓу двата ефекта е во тоа што фленцот 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 мултипли од основниот максимум. Ова не е случај кај фејзерот.

6.2 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 **фејзер**.

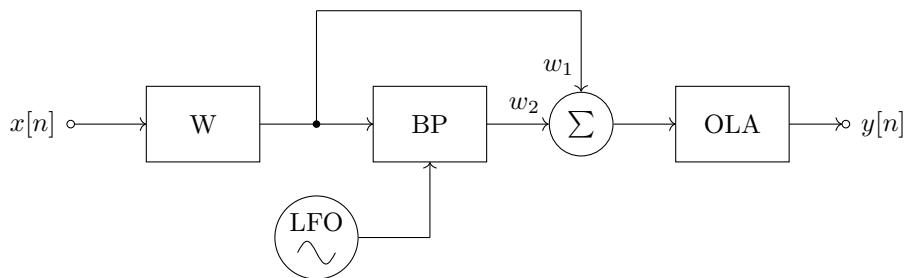
Ва-ва ефектот¹⁰ е првенствено направен за изведба на музика на електрична гитара, иако за првпат го пронашле трубачите и тромбонистите во 1920-тите. Тој се изведува со помош на филтер пропусник на опсег чија централна фреквенција се менува во време, а под контрола на свирачот преку педала за дозирање. Аудиосигналот што го дава филтерот на излезот се меша со оригиналниот сигнал за да се добие крајниот ефект.

Во електронската музика педалата може да биде заменета од нискофреквенциски осцилатор (LFO). Блок-дијаграмот на ва-ва ефектот реализиран со LFO е прикажан на Сл. 6.9. За да го исфилтрираме аудиосигналот со филтер пропусник на опсег со променлива амплитудна карактеристика, ќе треба да го поделиме сигналот на временски рамки со помош на **методата на прозорци (W)**, види Глава 4.3. Така, секоја рамка ќе можеме да ја процесираме со филтер со поместен пропусен опсег. За дозирање на ва-ва ефектот, процесираниите рамки ги добиваме како тежински збир на оригиналната и на филтрираната рамка. Конечно, за да добиеме

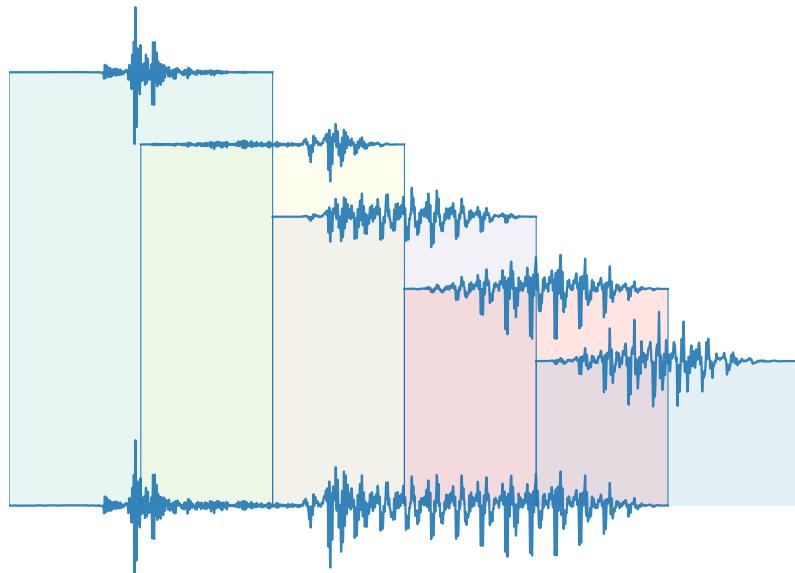
⁸Wikipedia: Chorus effect. https://en.wikipedia.org/wiki/Chorus_effect

⁹Wikipedia – Flanging. <https://en.wikipedia.org/wiki/Flanging>

¹⁰Wikipedia: Wah-wah (music). https://en.wikipedia.org/wiki/Wah-wah_%28music%29



Сл. 6.9: Блок-дијаграм на ва-ва ефектот реализиран со LFO.



Сл. 6.10: Илустрација на методата на преклопување (OLA) на рамки од говорниот аудиосигнал „македонски“.

крајниот сигнал, процесираните рамки ги надодаваме со примена на методата на преклопување (OLA)¹¹, која е илустрирана на Сл. 6.10 за рамки од говорниот аудиосигнал „македонски“ употребен за илустрација на STFT на Сл. 4.5.

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
  
```

¹¹Англ. *overlap-add*.

Глава 6. Дигитални аудиоэффекти

```
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 пропусник на опсег и нискофреквенцискиот осцилатор што ќе го користиме за да го филтрираме аудиосигналот:

- редот на филтерот `order`,
- фреквенцискиот опсег на филтерот `band`,
- почетната централна фреквенција на филтерот `fc_offset`,
- амплитудата на промената 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 ја примениме методата на прозорци, со тоа што за секој прозорец ќе ја пресметаме централната фреквенција на филтерот, ќе ги пресметаме неговите коефициенти и со нив ќе ја филтрираме екстрахираната рамка од аудиосигналот. Филтрираните рамки ќе ги додаваме на излезниот вектор `wav_wah` со преклопување на секоја рамка со претходната. Должината на преклопувањето е одредена од должината на прозорецот и од големината на скокот со кој тој се поместува. Преклопувањето е важно за да се избегне појавата на артефакти при надоврзувањето на излезните рамки.¹²

```
# 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
```

¹²Поради постепеното зголемување и намалување на прозорецот, при надоврзувањето на процесираните рамки всушност реализираме прелевање (*cross-fade*) помеѓу нив.

```

pos = 0
wav_wah = np.zeros(wav.size)
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 на филтрираниот сигнал преку нивно дозирање со тежински коефициенти.

```

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 што е добиен со ва-ва ефектот. За да го видиме влијанието на филтрирањето на сигналот во фреквенциски домен, ќе ги прикажеме спектограмите на оригиналниот, филтрираниот и на конечниот мешан сигнал.

```

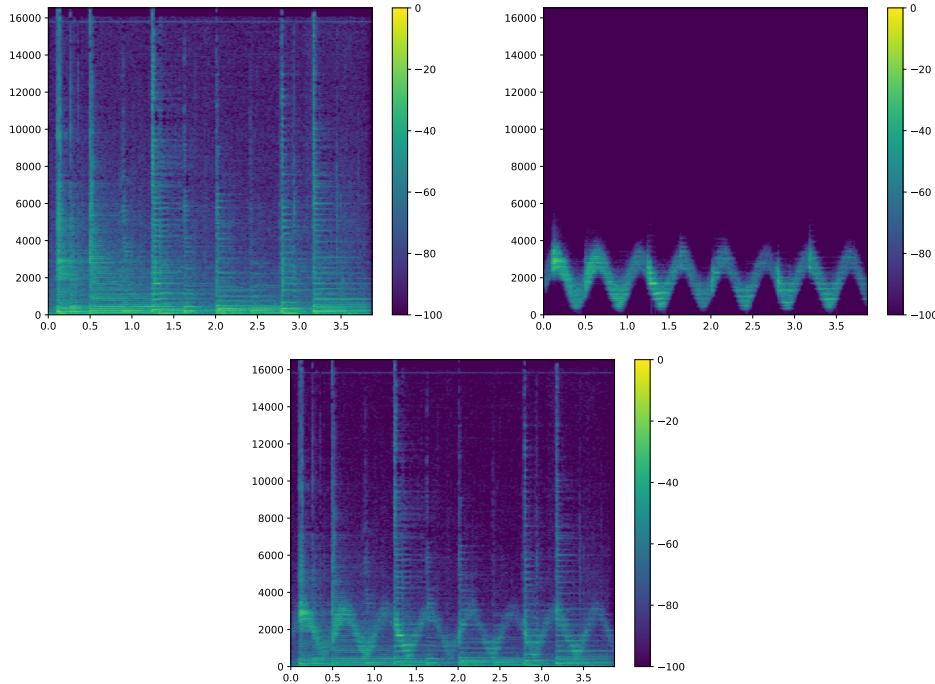
da.get_spectrogram(wav, fs)
da.get_spectrogram(wav_wah, fs)
da.get_spectrogram(wav_mix, fs)

```

На спектограмите, прикажани на Сл. 6.11, јасно може да се забележи дека примената на ва-ва филтерот генерира траг во спектарот на сигналот кој периодично се движи долж фреквенциската оска.

Фејзерот¹³ исто така претставува гитарски ефект добиен со употреба на каскада на филтри; овојпат тоа се сепропусни филтри кои внесуваат фазни промени во сигналот. При мешањето со оригиналниот и филтрираниот сигнал се создава интерференција која се одразува како дупки во амплитудната карактеристика на системот, наликујќи на примена на каскада од notch филтри непропусни на различни фреквенции. На овој начин, тие вршат селективно слабеење на делови од спектарот на сигналот кое може да се види како траг во спектограмот. Амплитудната карактеристика на филтрите и овојпат периодично се менува.

¹³Wikipedia: Phaser (effect). https://en.wikipedia.org/wiki/Phaser_effect



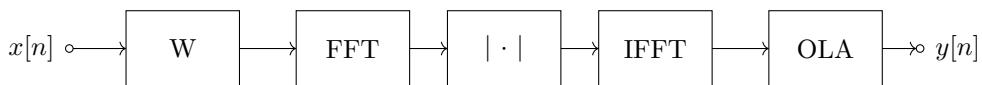
Сл. 6.11: Спектрограми на оригиналниот сигнал (горе лево), филтрираниот сигнал (горе десно) и мешаниот сигнал (долу) со ва-ва филтерот.

Аудиоэффекти базирани на Фурьеовата трансформација

Како пример за аудиоэффект базиран на Фурьеовата трансформација ќе го имплементираме ефектот на [роботизација](#)¹⁴. Овој ефект се добива со примена на Фурьеовата трансформација со прозорци, како што е прикажано на блок-дијаграмот на Сл. 6.12.

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 инверзната 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 добиеме 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 надоврзување на рамките ќе му даде роботизиран звук на сигналот.

¹⁴Wikipedia: Robotic Voice Effects. https://en.wikipedia.org/wiki/Robotic_voice_effects



Сл. 6.12: Блок-дијаграм на ефектот роботизација.

```

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

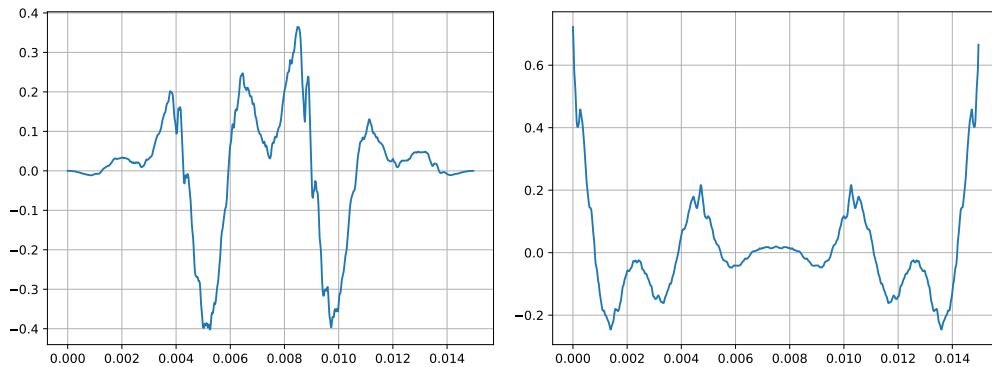
# %% load audio
audio_path = "audio"
wav_name = 'Glas.wav'
file_name = pjoin(audio_path, wav_name)
fs, wav = wf.read(file_name)
wav = wav[: int(3.65 * fs)] / 2**15
sd.play(wav, fs)

# %% 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]
    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

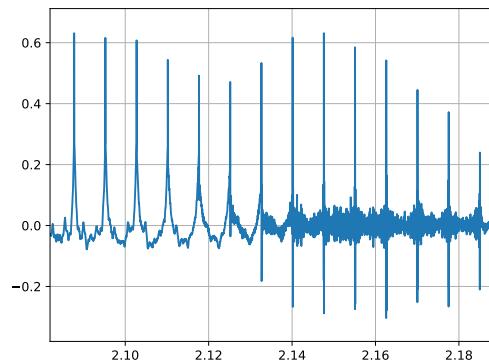
wav_robot = da.normalise(wav_robot)
sd.play(wav_robot, fs)

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

Ефектот на нулирање на фазата на примероците од спектарот доби-



Сл. 6.13: Временски облик на една рамка од аудиосигналот (лево) и нејзиниот облик при нулирање на фазата (десно).

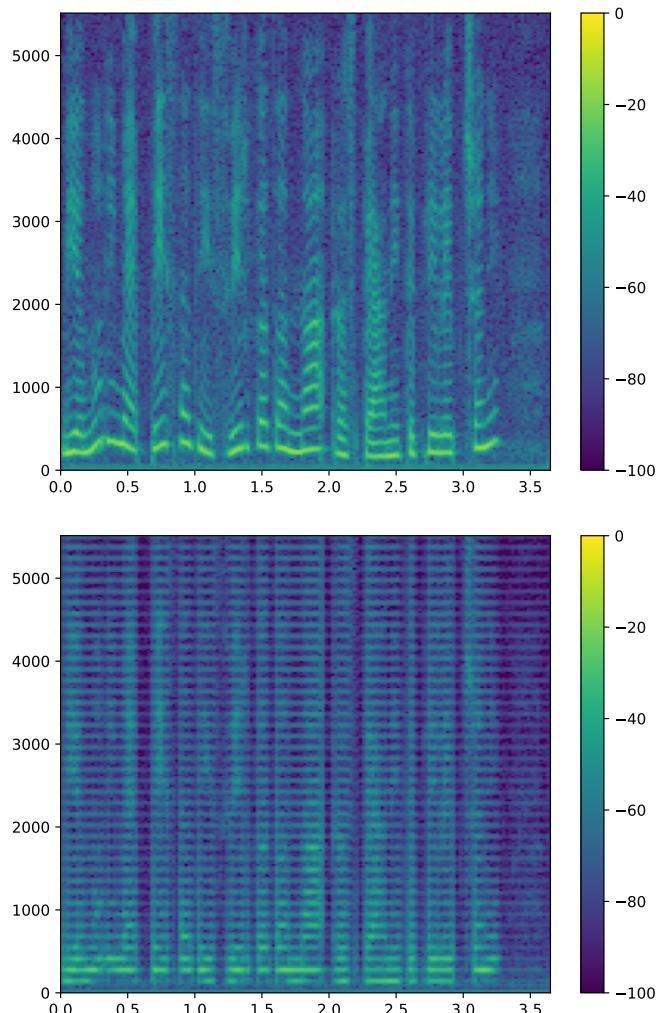


Сл. 6.14: Временски облик на сигналот добиен со надодавање на процесираните рамки со преклопување.

ени со дискретната Фурьеовата трансформација може да се види во временскиот облик на една од рамките на аудиосигналот на Сл. 6.13. Ефектот на надоврзување на вака добиените рамки може да го видиме на Сл. 6.14.

На Сл. 6.15 може да се види дека применетата на овој ефект резултира во сигнал со многу различен спектар од оригиналниот сигнал. Така, додека во оригиналниот сигнал може да се види движењето на основната фреквенција на гласот и нејзините хармоници, која се нарекува интонација, во роботизираниот сигнал може да се види дека основната фреквенција е константна и сите хармоници исцртуваат паралелни хоризонтални линии во спектрограмот.

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



Сл. 6.15: Зголемен дел од спектрограмот за ниските фреквенции на оригиналниот сигнал (горе) и роботизираниот сигнал (долу).

Глава 7

Компресија на звукот

Компресија на аудиосигналите се прави со цел да се намали големината на сировиот импулсно кодно модулиран (PCM) звук, со цел да се одговори на ограничувањата на меморискиот простор на персоналните компјутери и мобилните уреди, односно ограничениот битски проток што го даваат на располагање телекомуникациските врски за вмрежување. Постојат различни методи со кои може да се изврши компресија на дигиталниот звук (Spanias et al., 2006) и нив можеме да ги поделиме на две основни групи:

- методи без загуби¹ и
- методи со загуби.²

Во секоја од групите, во зависност од методот на компресија, дефинирани се различни формати на компримиран дигитален звук.

Методите за компресија без загуби не исфрлаат никаква информација од оригиналниот звук. Овие методи не се разликуваат многу од методите за компресија што се користат за општа компресија на каков било тип на датотека. Сепак, тие успеваат да ја намалат големината на звучните записи само на 50 – 60% од оригиналната. Ова се должи на комплексноста на аудиосигналите, а зависи од нивната содржина. Така, говорните сигнали, кои имаат помала комплексност и вклучуваат интервали на тишина, трпат поголема компресија отколку музиката.

Најупотребуваниот формат за компресија без загуби е FLAC³ кој претставува слободен стандард за компресија базиран на линеарна предикција.

¹Англ. *lossless*.

²Англ. *lossy*.

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

Други формати од оваа категорија се: WavPack, Shorten, Monkey's Audio, ATRAC Advanced Lossless, Apple Lossless, WMA Lossless, TTA, итн.

Методите со загуби ја намалуваат големината на дигиталниот запис жртвувајќи дел од информацијата која ја содржи, што се одразува негативно на неговиот квалитет. За зачувување на највисоко можно ниво на квалитет на аудиосигналот, овие методи се стремат исфрлањето на информацијата да биде ограничено на оние делови од звукот кои човековото уво и не може да ги чуе. За таа цел тие користат **психоакустички модел** на човековиот слух кој ги опишува психоакустичките феномени од интерес ([Painter and Spanias, 2000](#)). Пред сè, тоа се прагот на чујност и фреквенциското маскирање.

На овој начин, методите за компресија со загуби постигнуваат сма-
лувачко на големината на аудиодатотеките $10\times$, па и повеќе, односно до
5 – 20% од оригиналната големина на датотеката. Така на пример, стан-
дардната mp3⁴ компресија го намалува битскиот проток од номиналните
2 канала $\times 44,1\text{ kHz} \times 16\text{ bit} = 1411\text{ kb/s}$ на аудио-CD-то на проток од
128 kb/s, притоа зачувајќи го субјективното чувство за еднаков квали-
тет кај слушателот.

Освен mp3, други познати формати за компресија на звук со загуби се затворените AAC⁵ дефиниран во MPEG-2 и HE-AAC⁶ дефиниран во MPEG-4, како и WMA⁷.

Најдобриот алгоритам за компресија на звук со загуби денес е слободниот формат за аудиокомпресија Opus⁸. 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 однос на латентноста со стандардно доцнење од 26,5 ms, кое овозможува комуникација во реално време, споредено со 200 ms кај mp3 стандардот. Со жртвување на излезниот битски проток ова доцнење може да се намали и до 5 ms. Уште повеќе, Opus нуди оптимизација за имплементација на кодекот во вгнездените компјутерски системи. Конечно, Opus е слободен стандард и сите патенти кои го покриваат алгоритмот се бесплатни. Перформансите на Opus споредени со останатите алгоритми за компресија со загуби се дадени на [Сл. 7.1](#).

На [Сл. 7.2](#) се прикажани спектограмите на еден ист аудиосигнал компримиран со различни битски протоци со различни формати на компресија со загуби. Може да се види дека за постигнување на голема ком-

⁴MPEG-1 Layer 3 <https://en.wikipedia.org/wiki/MP3>

⁵Англ. *Advanced Audio Coding*.

⁶Англ. *High Efficiency AAC*.

⁷Англ. *Windows Media Audio*.

⁸Opus [https://en.wikipedia.org/wiki/Opus_\(audio_format\)](https://en.wikipedia.org/wiki/Opus_(audio_format))

пресија сите пристапи ги жртвуваат високите фреквенции од спектарот. Исто така, може да се забележи дека Opus навистина има подобри перформанси од останатите формати, а особено за ниските битскиprotoци.

Opus се базира на слободниот формат за компресија на музички сигнали [Vorbis \(OGG\)](#)⁹ и претставува негова замена. Компресијата на музичките сигнали во Opus се базира на слободниот CELT¹⁰ алгоритам, кој пак е базиран на модифицираната дискретна косинусна трансформација (MDCT)¹¹. Тој исто така сосема го заменува слободниот формат [Speex](#)¹², наменет за компресија на говор. Компресијата на говорните сигнали пак, се базира на слободниот формат за компресија на говорни сигнали [SILK](#)¹³.

Вреди да споменеме дека со истекување на патентот на Техниколор за mp3 во САД на 16. април 2017, mp3 се придржува на слободните формати за аудиокомпресија.

7.1 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 најзастапениот стандард за пренос и зачувување на компримиран звук, како на интернет мрежата, така и на персоналните компјутери, мобилните телефони, таблетите и во преносливите мултимедијални уреди. Иако денес постојат поразвиени алгоритми за компресија на аудиото, главните придобивки што тие ги носат се за малите битски protoци, 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 сè уште го задржува приматот во овој домен.

Зад mp3 стандардот стои експертската група [MPEG](#)¹⁵ чиишто главни задачи се:

- да објавува технички извештаи и резултати поврзани со компресија на звук и видео,
- да одреди начин за мултиплексирање на видео, звук и информациски protoци во единствен проток,
- да даде описи и синтакса за алатки за компресија на звук и видео со

⁹Vorbis audio compression. <https://xiph.org/vorbis/>

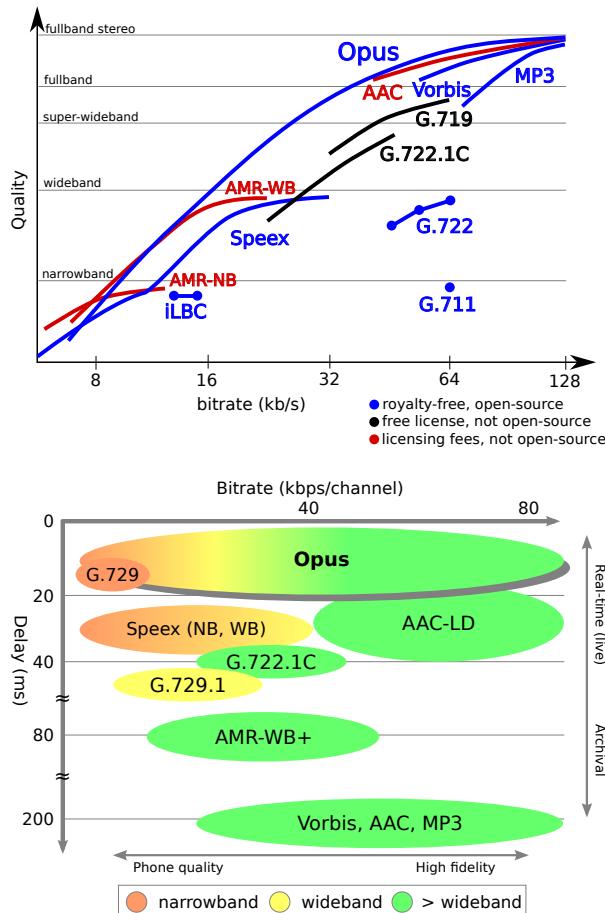
¹⁰Constrained Energy Lapped Transform <https://en.wikipedia.org/wiki/CELT>

¹¹Modified Discrete Cosine Transform https://en.wikipedia.org/wiki/Modified_discrete_cosine_transform

¹²Speex <https://en.wikipedia.org/wiki/Speex>

¹³SILK <https://en.wikipedia.org/wiki/SILK>

¹⁵Англ. *Moving Pictures Experts Group*.



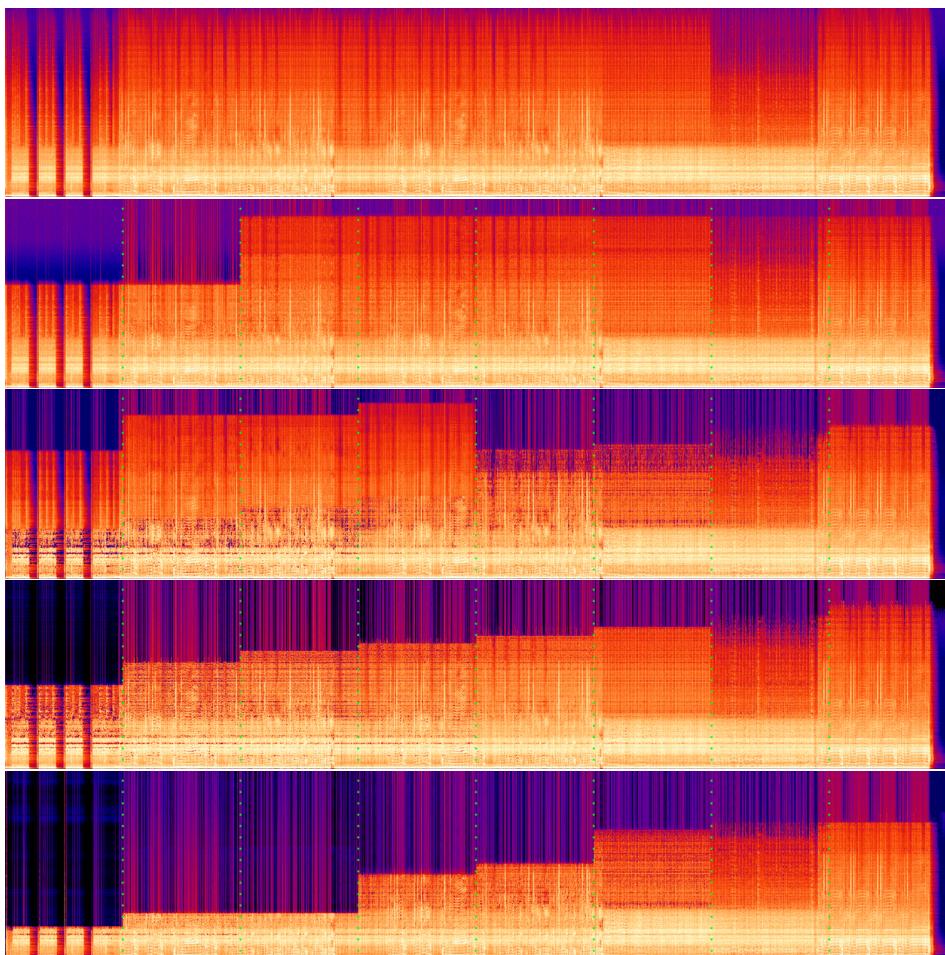
Сл. 7.1: Перформанси на Opus стандардот за аудиокомпресија во однос на квалитет за даден битски проток (лево) и латентност за даден фреквенциски опсег на сигналот (десно), споредено со други формати за аудиокомпресија.¹⁴

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 што 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:

¹⁵By Jean-Marc Valin - <http://opus-codec.org/comparison/>, CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=22463490>; и By Opus bitrate+latency comparison.png: Jean-Marc Valinderivative work: Flugaal - This file was derived from: Opus bitrate+latency comparison.png:, CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=20605263>



Сл. 7.2: Спектрограми на ист аудиосигнал компримиран со битскиprotoци од 32 до 160 kbit/s (од лево надесно) со различните формати за компресија на звук (од горе надолу): оригиналниот сигнал, Opus, AAC, Vorbis и mp3.¹⁸

- MPEG-1,
- MPEG-2 BC¹⁶,
- MPEG-2 NBC/AAC¹⁷,
- MPEG-4,
- MPEG-7 и
- MPEG-21.

¹⁶Англ. *Backwards-Compatible*.

¹⁷Англ. *Non-Backward Compatible/Advanced Audio Coding*.

¹⁸[https://commons.wikimedia.org/wiki/Opus_\(audio_codec\)#Spectrograms_in_comparison](https://commons.wikimedia.org/wiki/Opus_(audio_codec)#Spectrograms_in_comparison)

Последните два стандарди не се стандарди за компресија. MPEG-7 дефинира дескриптори на звучните содржини и на видеосодржините, со чија помош тие можат да бидат описаны за да се овозможи побрз пристап до нив во базите на податоци. MPEG-21 дефинира мултимедијална рамка и нуди менадирање 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.

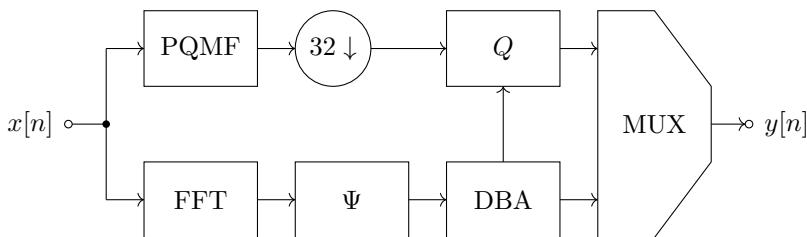
MPEG-1 Аудио (ISO/IEC 11172-3) стандардот е првиот аудиостандард објавен од MPEG групата во 1992 по четиригодишна напорна работа на усогласено истражување на светските експерти од областа на аудиокомпресијата. Неговата намена била да се овозможи стерео-CD-квалитет. MPEG-1 поддржува стерео аудио-CD квалитет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

- подопсежко разлагање,
- анализа со банки на филтри,
- психоакустичка анализа,
- адаптивна сегментација,
- трансформациско кодирање,
- динамично 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 здружено¹⁹ стерео. Протоците на компримиралиот звук кои ги нуди се во опсег од 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 компримиралиот сигнал. Секое повисоко ниво ги вклучува функционалните блокови од претходните.

¹⁹Англ. *joint*.



Сл. 7.3: Архитектура на кодерот MPEG-1 ниво I и ниво II.

MPEG-1 нивото II е прикажано на Сл. 7.3. Влезниот сигнал $x[n]$ се разложува на 32 подопсези со еднаква ширина, линеарно распределени долж фреквенциите на сигналот со употреба на PQMF²⁰ банка на филтри. Така, влезен сигнал со f_s од 48 kHz се разлага на подопсези од по 750 Hz. Прототипниот филтер со кој е изградена банката е од 511. ред поради што вкупната дисторзија, карактеристична за PQMF банките на филтри, останува под прагот на чујноста, а слабеењето во непропусниот дел од спектарот од 96 dB осигурува занемарливо меѓуопсежно влијание.

Бидејќи излезните 32 сигнали од банката на филтри се со намален фреквенциски опсег, фреквенцијата на земање примероци за секој канал соодветно се намалува со $32 \times$ потсемплирање. На секој од овие сигнали во блокови од по 12 примероци се врши динамичка компресија, односно нормализација на вредност 1. Притоа, за секој блок се пренесува коефициентот²¹ со кој е направена нормализацијата. Блоковите соодветствуваат на $12 \cdot 32 = 384$ влезни примероци. При f_s од 44,1 kHz на пример, ова изнесува 8,7 ms.

Модулот за **психоакустичка анализа** (Ψ) ги определува **праговите на забележлива дисторзија** (JND)²² за различните подопсези врз база на спектарот пресметан во 512 (ниво I) односно 1024 (ниво II) точки во блокот за FFT. JND ја одредуваат максималната грешка на квантизација која може да се дозволи за секој од подопсезите. Таа се употребува во блокот за **динамичкото доделување на битови** (DBA)²³ при распределбата на расположливиот број на битови помеѓу подопсезите. Така, приоритет, односно квантизери со најголема резолуција, ќе добијат подопсезите со најниски JND.

За пресметка на JND, модулот за психоакустичка анализа ги употребува двата феномени прикажани на Сл. 7.4:

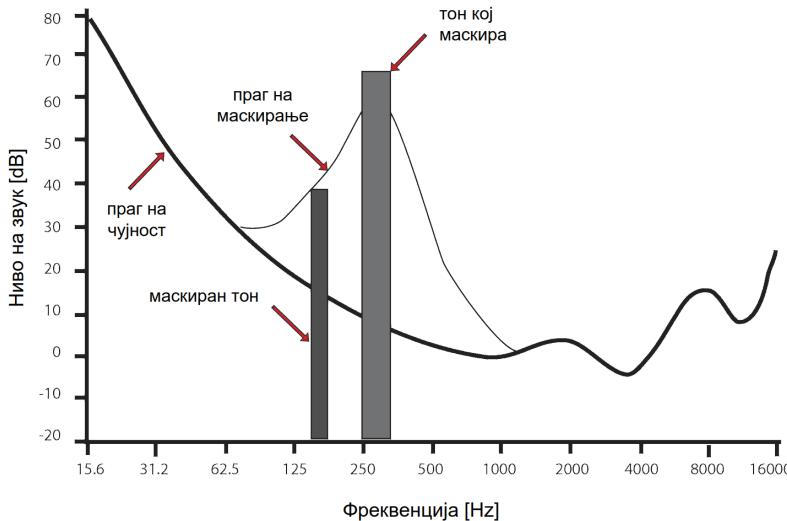
- **прагот на чујност** – со кој е определена минималната спектрална

²⁰Англ. *Pseudo Quadrature Mirror Filter*.

²¹Англ. *scale factor*.

²²Англ. *Just Noticeable Distortion*.

²³Англ. *Dynamic Bit Allocation*.



Сл. 7.4: Прагот на чујност и појавата на фреквенциско маскирање на кои се базира принципот на работа на психоакустичката аудиокомпресија.²⁵

амплитуда која човек може да ја чуе на дадена фреквенција,

- о **фреквенциското маскирање** – со кое се одредува делот од амплитудниот спектар кој не може да биде чуен поради присуството на изразени спектрални компоненти во непосредна близина.²⁴

Фреквенциското маскирање е тесно поврзано со **критичните опсези**²⁶ на аудиторниот систем што 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 фреквенцијата.²⁷

Одредувањето на JND за еден критичен опсег е илустрирано на Сл. 7.5. Синусниот тон со изразена амплитуда ќе направи фреквенциско маскирање во критичниот опсег. Пресекот на функцијата на фреквенциското маскирање со границите на критичниот опсег со пониска амплитуда го одредува нивото на JND.

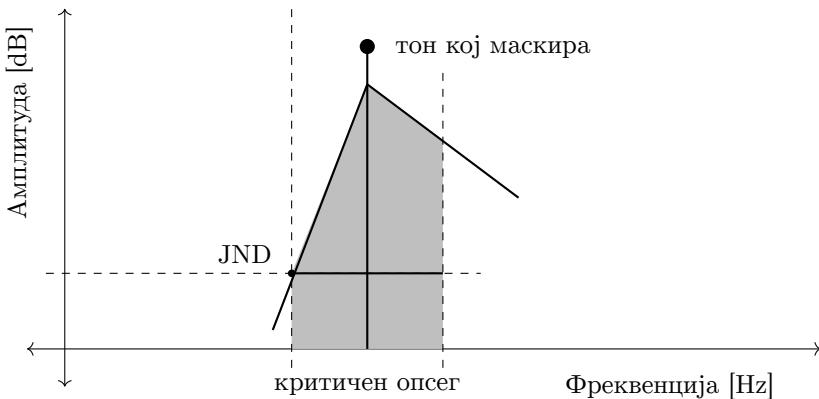
На крајот, започнува итеративна процедура на доделување на битови која ги користи пресметаните JND прагови за секој подопсег за да

²⁴Wikipedia: Psychoacoustics – Masking effects: https://en.wikipedia.org/wiki/Psychoaoustics#Masking_effects

²⁵By Audio_Mask_Graph.jpg: Daxx4434derivative work: Cradle (talk) - Audio_Mask_Graph.jpg, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=8390519>

²⁶Wikipedia: Critical band: https://en.wikipedia.org/wiki/Critical_band

²⁷Повеќе за аудиторниот систем на човекот може да прочитате во главата за психоакустика во скриптата по предметот Електроакустика <https://gitlab.com/feeit-freecourseware/electroacoustics>



Сл. 7.5: Одредување на JND во еден критичен опсег според фреквенциското маскирање на еден изразен тон.

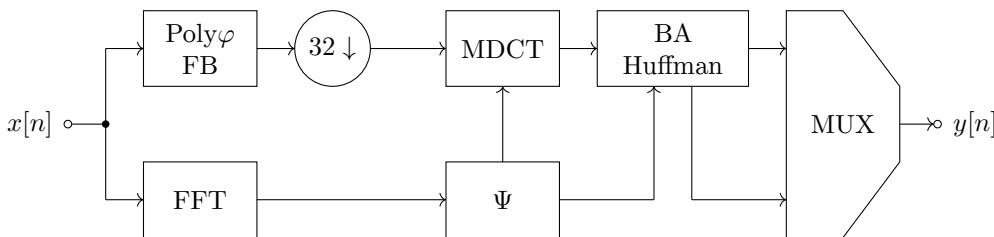
одбере оптимален квантизер за тој подопсег од дефинираното множество на квантизери. Изборот на квантизерите се прави така што двата критериуми – зададениот краен проток и пресметаниот JND, се задоволени. Коефициентите на размер се квантизираат со 6 битови за секој подопсег, а изборот на квантизерот се кодира со 4 битови.

Во MPEG-1 нивото I, потсемплираните подопсежни секвенции се квантизираат и му се испраќаат на приемникот проследени со квантизираните коефициенти на размер и со избраните квантизери. Она што е различно во нивото II е:

- психоакустичкиот модел има поголема FFT резолуција,
- максималната резолуција на подопсежните квантизери е зголемена на 16 битови, а понизок вкупен проток се остварува со намалување на бројот на понудени квантизери за повисоките фреквенции,
- количината на додатна информација за коефициентите на размер е намалена преку искористување на временско 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.

MPEG-1 ниво-III алгоритмот, чија архитектура е прикажана на Сл. 7.6, работи врз последователни рамки на податоци. Секоја рамка се состои од 1152 примероци звук и понатаму се дели на две подрамки од по 576 примероци, наречени гранули. Декодерот може да ја декодира секоја гранула посебно.

Во mp3 се употребува хибридна полифазна банка на филтри за зголемена фреквенциска резолуција и подобра апроксимација на критичните опсези на човековото уво. Исто така, хибридната банка на филтри



Сл. 7.6: Архитектура на MPEG-1 ниво III.

вклучува адаптивна сегментација за подобрување на контрола врз предехото. Таа е конструирана со надоврзување на секој подопсежен филтер на блок за адаптивна модифицирана косинусна трансформација (MDCT). 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 има способност за компактирање на енергијата на сигналите, односно за претставување на поголем дел од нивната енергија со мал број на коефициенти. Поради ова, таа често се користи за нивна компресија.²⁸

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.

Така, доделувањето на битови и квантизацијата на MDCT спектралните компоненти се изведува преку вгнездена јамка која интегрира и неуниформна квантизација и **Хафманово кодирање**. Внатрешната јамка го приспособува чекорот на квантизација на трансформациските коефициенти за секој блок, сè 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 прагови на JND.

7.2 Компресија на говор

Линеарното предиктивно кодирање (LPC)²⁹ е еден од најважните концепти во кодирањето на дигиталниот говор. Техниката овозможува високо-квалитетно кодирање со ниски битски протоци од редот на 2,4 kbit/s, што во споредба со вообичаениот проток на дигиталната телефонија од 8 kHz

²⁸DCT се употребува и за компресија на слика, на пример кај JPEG стандардот.

²⁹Англ. *Linear Predictive Coding*.

$\times 8 \text{ bit} = 64 \text{ kbit/s}$, претставува компресија од 26 пати. Таа е во употреба во многу телекомуникациски системи за пренос на говор, од кои најважен е GSM³⁰ системот за мобилна комуникација.

LPC се базира на параметризација на говорниот сигнал со употреба на **моделот извор–филтер** на создавање на говорот прикажан на Сл. 7.7. 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 пак 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 што одговара на промените во напречниот пресек предизвикани од поставеноста на **артикулаторите**: јазикот, мекото непце, вилицата, и усните.³¹ Тој ја моделира оваа функција со IIR филтер кој содржи само полови:

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

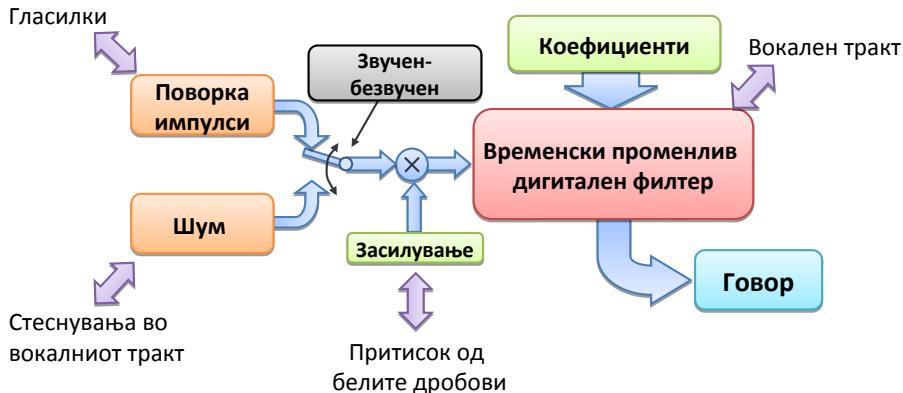
каде што $H(z)$ е преносната функција на филтерот во z -домен, $Y(z)$ е излезниот говорен сигнал, $X(z)$ е побудниот сигнал кој е добиен од изворот во моделот, G е засилувањето што е интегрирано во филтерот иако претставува дел од изворот, a_k се коефициентите на филтерот, а p е неговиот ред. Во временски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

$$y[n] = \sum_{k=1}^p a_k y[n-k] + Gx[n] \quad (7.2)$$

Со употреба на моделот извор–филтер, говорот се параметризира и може да се пренесува или зачувува преку параметрите на моделот, без

³⁰Фран. *Groupe Spécial Mobile*.

³¹Повеќе за начинот на создавање на човековиот говор може да прочитате во главата за Музика и глас во скриптата по предметот Електроакустика <https://gitlab.com/feeit-freecourseware/electroacoustics>



Сл. 7.7: Моделот извор–филтер на принципот на создавање на говорот.

употреба на аудиопримероци. На овој начин, може да се добие доволно близка апроксимација на говорниот сигнал со голема стапка на компресија. Параметрите што 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
- висина – периода на основниот хармоник.

FS³² 1015 кодекот од 1982 г. е првиот кодер базиран на LPC. Протокот што тој го остварува е 2,4 kbit/s. Бидејќи филтерот кој го моделира вокалниот тракт е од 10-ти ред, кодекот се нарекува и LPC-10. Ако не го знаете моделот, параметрите кои се испраќаат, немаат никакво значење. Поради тоа, FS 1015 кодекот се користел во воени цели, за потоа да го присвои и НАТО.

LPC-10 иако нуди одлична разбираливост на кодираниот говор, негојиот квалитет не е на високо ниво и има карактеристичен роботски призвук. Ова се должи на главниот недостаток на моделот извор–филтер – нефлексибилноста на изворот; тој може да работи исклучиво во еден од двата режими. Ова претставува пречка во моделирањето на гласови од мешан тип, на пр. кај звучните согласки како „з“ /z/ или „в“ /v/, како и при моделирање на премините помеѓу гласовите. Овој недостаток е надминат кај кодерот CELP³³, кој на местото од едноставниот извор, работи со база на побуди поместени во една кодна книга. CELP моделот е оној што е во употреба во GSM мрежата.

³²Англ. *Federal Standard*.

³³Англ. *Code-Excited Linear Prediction*.

Глава 8

Синтеза на звук и говор

Моделите кои вршат параметризација на звучните сигнали може да се искористат за генерирање на нови звучни форми. Еден таков модел е моделот извор–филтер кој го сретнавме кај компресијата на говорните сигнали. Во оваа глава ќе го искористиме за синтеза на мелодија, односно за синтеза на пеење. Во вториот дел, ќе се запознаеме повеќе со методите за синтеза на говор и ќе имплементираме еден едноставен систем за синтеза на говор на македонски.

8.1 Линеарна предиктивна анализа

Филтерот што се користи во моделот извор–филтер, односно во компресијата на говорните аудиосигнали базирана на линеарна предикција (LPC) претставена во Глава 7.2 го има обликот даден во (7.1):

$$H(z) = \frac{Y(z)}{X(z)} = \frac{G}{1 - \sum_{k=1}^p a_k z^{-k}} \quad (8.1)$$

Тој содржи само полови во својата преносна карактеристика, иако за верно претставување на назалите и на фрикативите се потребни и нули во спектарот. Тоа е така, затоа што коефициентите на вака дефинираниот филтер можат да се пресметаат со употреба на методот за [линеарна предиктивна анализа](#). За апроксимација 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 (Rabiner and Schafer, 1978):

$$\tilde{y}[n] = \sum_{k=1}^p \alpha_k y[n - k] \quad (8.2)$$

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 на предикција може да се пресмета како:

$$e[n] = y[n] - \tilde{y}[n] = y[n] - \sum_{k=1}^p \alpha_k y[n-k] \quad (8.3)$$

Така, грешката на предикција се добива преку филтрирање на излезниот сигнал со филтер со преносна функција во z -домен:

$$A(z) = 1 - \sum_{k=1}^p \alpha_k z^{-k} \quad (8.4)$$

Споредувајќи ги овие равенства со (8.1) и (7.2):

$$y[n] = \sum_{k=1}^p a_k y[n-k] + Gx[n] \quad (8.5)$$

можеме да видиме дека ако е задоволен условот:

$$\alpha_k = a_k, \quad \text{за } k = 1, 2, \dots, p \quad (8.6)$$

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

$$e[n] = Gx[n] \quad (8.7)$$

а филтерот на грешка $A(z)$ е инверзен на филтерот на моделот извор–филтер $H(z)$:

$$H(z) = \frac{G}{A(z)} \quad (8.8)$$

Основниот проблем во линеарната предикција е одредување на коефициентите α_k директно од звучниот сигнал за да се добие добра проценка на неговите спектрални карактеристики преку употреба на (8.8). Поради временската динамика на спектралната содржина на говорот, нужна е проценка на овие коефициенти за кратки отсекоци на сигналот добиени со методата на прозорци.

Основниот пристап за решавање на овој проблем е пресметувањето на коефициенти на предикторот кои ќе ја минимизираат средната квадратна грешка на предикција. Постојат повеќе алгоритми за нивно пресметување, како методите со автокорелација и коваријанса. Еден од методите што најчесто се применува е рекурзивниот метод на Дурбин (Rabiner and Schafer, 1978, 2011).

8.2 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 кој било сигнал, како на пр. поворка на импулси или бел шум или мешавина од двете.

За практично да се запознаеме со начинот на функционирање на линеарната предикција ќе ја илустрираме нејзината примена во синтеза на човечки глас. Тој процес е во сржта на нејзината примена за компресија на говорните сигнали, кај аудиоэффектите, како и за синтезата на говор. **Вокодерот**¹ на пример, е аудиоэффект кој се добива со побуда на пресметаниот филтер со музички сигнал, најчесто добиен со синтисајзер. Музичката мелодија отсвирена на синтисајзерот бива вообличена во аудиосигнал што 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.

- ✓ Задача за час. За целите на оваа анализа направете снимка од сопствениот глас како ги изговарате петте самогласки во македонскиот јазик: /a/, /e/, /i/, /o/ и /u/, 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 **LibROSA**³. Овој пакет 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
```

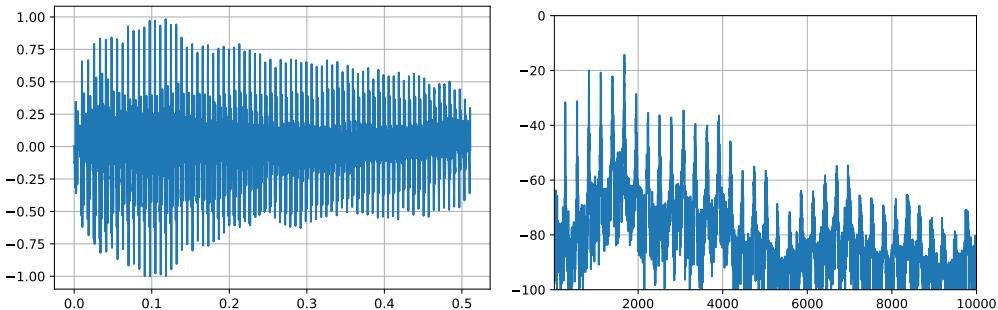
Ќе го вчитаме звучниот запис со самогласката /a/ и ќе го прикажеме неговиот временски и спектрален облик, [Сл. 8.1](#).

```
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
```

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

²За Audacity види повеќе во Глава А.

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



Сл. 8.1: Приказ на временскиот облик (лево) и спектралниот облик (десно) на аудиосигналот на гласот /a/.

```
import da

# %% вчитај звук
fs, wav = wavfile.read('audio/glas_aaa.wav')
wav = wav / 2**15
t = np.arange(wav.size) / fs
wav = da.normalize(wav, 0)

# %% исцртај временски домен
plt.figure()
plt.plot(t, wav)
plt.grid()

# %% исцртај спектрален домен
f, wav_spec = da.get_spectrum(fs, wav)
plt.figure()
plt.plot(f, wav_spec)
plt.grid()
plt.axis([0, 1e4, -100, 0])
```

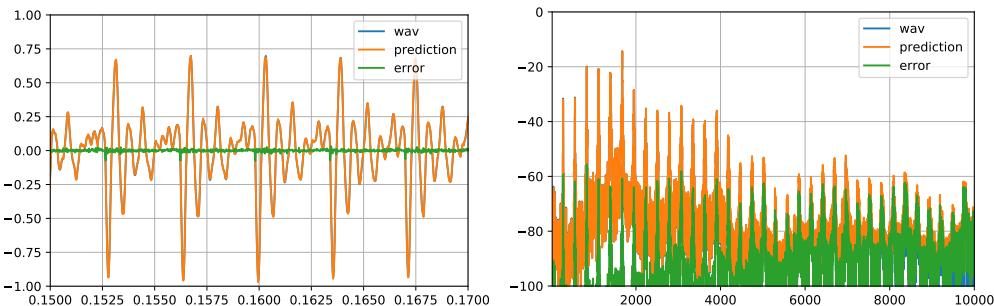
Во спектралниот облик на сигналот може да се видат хармониите кои се должат на периодичноста на сигналот, како и анвелопата на спектарот која има изразени врвови. Овие врвови одговараат на резонантните фреквенции на вокалниот тракт при изговор на гласот /a/. Кога се работи за самогласки ти се нарекуваат **форманти**.

Сега, со употреба на функцијата `lpc` ќе ги најдеме коефициентите на филтерот, кои всушност треба да ја опишат анвелопата на спектарот на сигналот.

```
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. 8.2 се претставени оригиналниот аудиосигнал, неговата пре-

8.2. Синтеза на глас со линеарна предикција



Сл. 8.2: Оригиналниот аудиосигнал, неговата предикција и сигналот на грешка во временски домен (лево) и во спектрален домен (десно).

дикција и сигналот на грешка. Може да се види дека разликата меѓу аудиосигналот и предикцијата добиена со филтер со ред 25 е многу мала и е најизразена на почетокот на секоја периода.

Ова е всушност сигналот на грешка 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 таква брзина што страничниот притисок не може да ги држи гласилките повеќе отворени па тие наеднаш се затвораат.

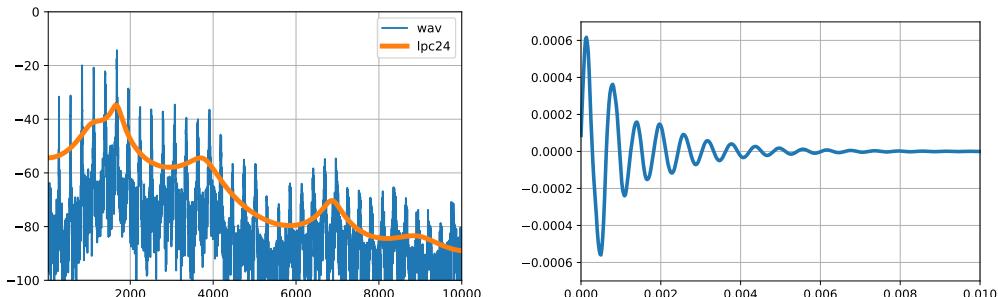
Оваа периодичност појасно се гледа во спектарот на сигналот на грешка кој ги содржи хармониците од оригиналниот сигнал. Уште поважно е тоа што тој не ја содржи спектралната анвелопа, односно енергијата на сите негови хармоници е речиси иста. Може да се каже дека сигналот на грешка претставува спектрално „обелена“ верзија на оригиналниот сигнал.⁴

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

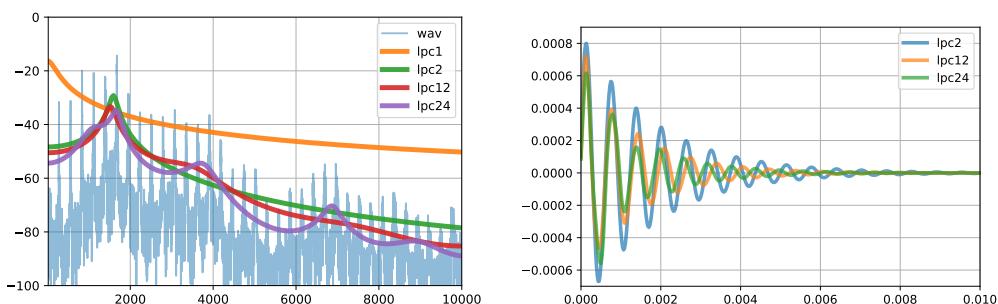
```
g = 0.001
f_filt, h_filt = sig.freqz(g, a_filt, fs=fs)
h_filt = 20 * np.log10(np.abs(h_filt))
plt.figure()
plt.plot(f, wav_spec)
plt.plot(f_filt, h_filt, lw=4)
```

Импулсниот одсив пак на филтерот ќе го добиеме со следниот код.

⁴Процесот на „белење“ на еден сигнал на англиски се нарекува *whitening*.



Сл. 8.3: Фреквенциската карактеристика на добиениот филтер за предикција од 25-ти ред (лево) и неговиот импулсен одсив (десно).



Сл. 8.4: Фреквенциски карактеристики на филтри за предикција од различен ред (лево) и нивните импулсни одсиви (десно).

```

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)

```

Двата графикони се прикажани на Сл. 8.3. Од друга страна, на Сл. 8.4 е дадена споредба на фреквенциските карактеристики и импулсните одсиви на филтри за линеарна предикција со различен ред. Може да се види дека филтерот 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 /a/.

Како што го зголемуваме бројот на коефициенти во филтерот така сè подобро тие ја претставуваат спектралната анвелопа на аудиосигналот. Непишано правило е дека при моделирање на говор бројот на коефициенти на филтерот треба да е еднаков на бројот на форманти, т.е. на 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. За да го опфатиме целиот

слушен опсег ќе земеме редот на филтерот да биде 22.

Синтеза на мелодија

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)

# исцртај ја побудата и излезот
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 што наликува на самогласката /ɑ/. 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. 8.5.

Следно да го побудиме филтерот со променлива поворка на импулси која одговара на музичка мелодија за да добиеме излезен сигнал налик на пеење. Најпрвин, ќе ги иницијализираме листите со музичките ноти од мелодијата и нивното времетраење. Времетраењето на нотите ќе го изразиме релативно на времетраењето на четвртинаnota за која ќе земеме времетраење од 1.

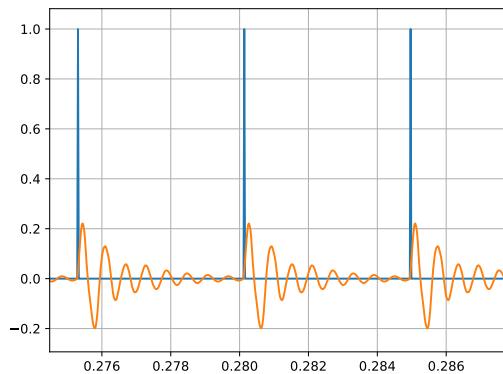
За мапирање на нотите во фреквенции ќе направиме речник според следната tabela:⁶

```
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 0.5 2'.split()

tone2freq = {
```

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

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



Сл. 8.5: Побудата на филтерот во вид на поворка на Диракови импулси (сино) и одзивот на филтерот (портокалово).

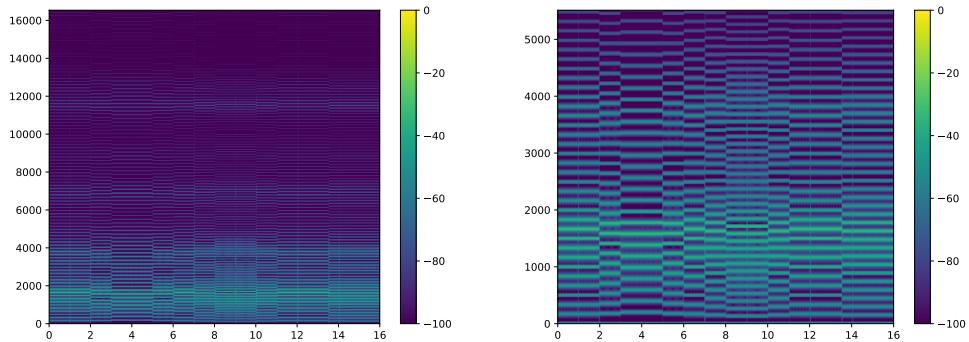
Табела 8.1: Фреквенција на нотите на пијано во основната октава.

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

```
'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 со **темпото** на мелодијата. Притоа, вообичаено се зема BPM да биде 120, што соодветствува на времетраење од 0,5 s на четвртината нота.

⁷Англ. *beats per minute*.



Сл. 8.6: Спектрограм на синтетизираниот сигнал на пеење (лево) и зголемен приказ за ниските фреквенции (десно).

```
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 искористиме за да ја зголемиме или намалиме висината на целата мелодија за даден процент.

Конечно, ќе направиме јамка која ќе ја измине мелодијата, дефинирана од низата на ноти и низата времетраења, и ќе генерира сигнал со кој ќе го побудиме филтерот. Добиените филтрирани сигнали ќе ги надоврзeme 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)

# преслушај и исцртај го добиениот сигнал
sd.play(wav_melody, fs)
da.get_spectrogram(wav_melody, fs)
```

Спектрограмот на добиениот сигнал е прикажан на Сл. 8.6. Во ниските фреквенции на спектрограмот може да ги видиме промените на висината на основниот хармоник кои соодветствуваат на „испеаната“ мелодија.

✓ **Задача за дома.** Синтетизирајте ја истата мелодија со боја на некој инструмент на пр. виолина, гитара или кавал, користејќи ги достапните аудиодатотеки.

8.3 Синтеза на говор

Вештачката синтеза на човековиот говор била инспирација за голем број научници и пронаоѓачи. Нејзиниот развој започнува со механичките модели на човековиот говорен апарат како оној на Фон Кемпелен од XVIII век, па преку механичкиот VODER од 1930-тите и аналогните електронски синтетизатори од 1950-тите, до денешните системи во ерата на дигиталните компјутери. Денес, технологијата веќе ја има постигнатото главната цел – синтеза на говор кој не да може да се разликува од природниот.⁸

Навистина, модерните синтетизатори, не само што се сосема разбираливи и звучат 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.

Вештачката синтеза на човековиот говор сè уште претставува област на интензивно истражување во светската наука. Денес, фокусот е свртен главно на доловување на паралингвистичките информации во говорот, кои претставуваат збир на сите информации кои говорникот ги пренесува, 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 сопствени ставови, на пр.: изненадување, неверување, иритираност, замисленост, настојување, итн. Овој вид 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 текст, на пример, од некоја книга, вест или пак реплика генерирана од голем јазичен модел. Поради тоа, овие системи се нарекуваат системи за синтеза на говор од текст или TTS⁹ системи.

Асистивни говорни технологии

Примената на TTS системите во човековото секојдневие е разнолика. Кај пошироката популација тие можат да се употребат за исчитување на

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

⁹Англ. *Text-to-Speech*.

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 апарати, мобилните уреди, компјутерите и паметните звучници. Синтезата на говор има примена и во едукативни цели на пример во алатките за изучување на странски јазици. Врвниот дострел на оваа технологија е реализацијата на универзални преведувачи, во спој со системите за препознавање на говор и за машински превод, види Глава 9.1 и 9.3.

Од клучна важност е примената на говорните технологии во асистивните уреди. **Асистивните технологии** опфаќаат уреди или услуги наменети за да им овозможат на луѓето со ограничени способности или попречност, непречено да ги извршуваат секојдневните активности, како оние поврзани со домувањето, образоването, работното место или слободно време.¹⁰ Во суштина, тие им помагаат на лицата во постигнувањето на поголема независност и во подигнување на квалитетот на нивниот живот.

Асистивните информатички уреди претставуваат хардверски или софтверски системи кои на лицата со попречност им овозможуваат пристап, интеракција и употреба на компјутерите во домот, на училиште или на работа. Некои примери вклучуваат: адаптирани тастатури, ergonomски глувчиња, уреди за контрола на покажувачот со помош на главата, очите или на устата односно на јазикот, кликови, Браев дисплей, Браев печатач, читач на еcran, зголемувач (лупа) за еcran итн.

Асистивните говорни технологии пред сè се наменети за да им помогнат на лицата со попречност во нивната комуникација со средината, но и во употребата на дигиталните уреди и пристапот до информации. Тие ги вклучуваат системите за синтеза на говор и системи за препознавање говор. TTS системите им овозможуваат на луѓето со визуелна попречност пристап до текстуални содржини и употреба на компјутерите и мобилните уреди преку интеграцијата во читачите на еcran. Кај наглавите и луѓето со пречки во говорот, уредите за синтеза на говор со вградена тастатура им овозможуваат комуникација со луѓе кои не го разбираат знаковниот јазик. Ова е од особено значење кај невербалните деца, кои со помош на уредите за **аугментирана и алтернативна комуникација (AAC)**¹¹, на пр. говорните табли како Cboard¹², можат да се комуницираат со околнината и да изразуваат свои потреби и желби. Во овој вид на апликации од особено значење

¹⁰ Асистивните технологии се описаны со ISO 9999:2011 стандардите за меѓународна класификација на продукти.

¹¹ Англ. *Augmentative and Alternative Communication*.

¹² Cboard <https://www.cboard.io/>

е TTS системите да можат да синтетизираат природен говор кој би можел да се приспособи за да се пренесат емоции и ставови, но и да овозможат приспособување на гласот за негова персонализација од корисниците.

Од друга страна, системите за препознавање на говор (ASR)¹³ можат да се искористат за олеснување на пристапот до технологијата на луѓето со попреченост во движењето. На пример, нивната интеграција во виртуелните асистенти ќе им овозможи на луѓето да ги контролираат уредите во домот или да пристапат на интернет без потреба од мануелна контрола. Кај наглавите, ASR системите можат да се искористат за транскрипција во на говорни содржини во реално време, со што би им овозможило следење на разговор, телевизиски програми, онлајн видео содржини или живи настани.

Во насока на подобрување на квалитетот на животот на лицата со попреченост, а за максимизирање на инклузивноста во општеството, од посебно значење е пристапноста на софтверските апликации и веб-страниците. За таа цел, меѓународниот W3C¹⁴ конзорциум развиил насоки за зголемување на пристапноста на веб-содржините, познати како WCAG¹⁵, за лица со различен вид на попреченост, како на пр. слепите лица и оние со оштетен вид, глувите или оние со оштетен слух, лицата со ограничување во движењето, пречки во говорот, фотосензитивност, како и за лицата со пречки во учењето и когнитивни ограничувања.

WCAG препораките се категоризираат во три нивоа на усогласеност: А, AA, AAA. Првото ниво вклучува основни препораки за пристапност како: овозможена навигација со тастатура, алтернативни информации за нетекстуална содржина, на пр. текстуален опис на сликите, можност за промена на боја и големина на текстот, веб-содржината да нема компоненти од кои корисникот не може да излезе со помош на тастатура итн. Второто ниво вклучува понапредни препораки како: контрастот на содржината да е најмалку 4.5:1, елементите за навигација да се конзистентни низ целата веб-страница, полињата во формуларите да имаат точни ознаки и содржината да биде приспособена за употреба на читач на екран. Третото ниво вклучува препораки како: имплементација на знаковен јазик за звучни или видео содржини, контрастот на бојата да е најмалку 7:1 итн.

Пошироко, универзалниот дизајн опфаќа методи, техники и насоки за создавање производи, околини и услуги што се достапни и употребливи за секого, без оглед на нивната возраст или способности. Целта е уште во фазата на дизајн на производите, архитектурата и другите аспекти на нашата околина, да се интегрира пристапност на најширок спектар на ко-

¹³Англ. *Automatic Speech Recognition*.

¹⁴Англ. *World Wide Web Consortium*.

¹⁵Web Content Accessibility Guidelines <https://www.w3.org/WAI/standards-guidelines/wcag/>

рисници, вклучувајќи ги и лицата со попреченост, притоа минимизирајќи ја потребата за адаптација или специјализиран дизајн. Со инкорпорирање на овие принципи може да се создаде поинклузивен и попристапен свет за секого.

Основни 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 разбираливоста и по квалитетот е близок на природниот. Постојат пет основни парадигми во дигиталната синтеза на говор:

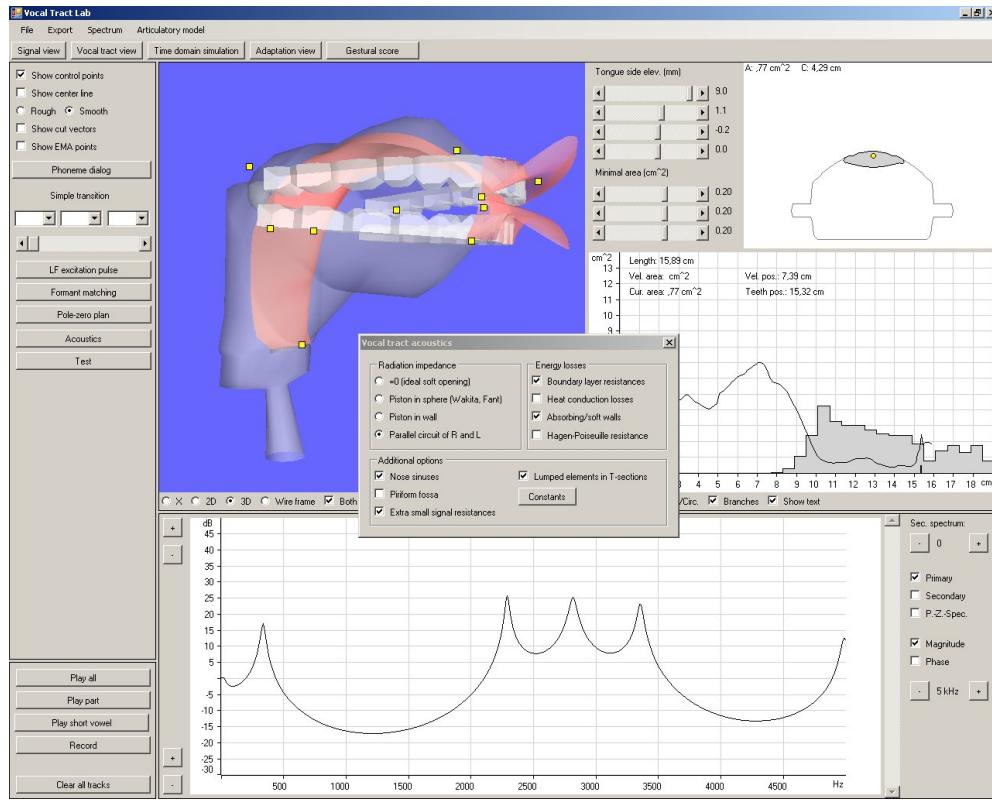
- артикулаторна синтеза,
- синтеза со форманти,
- синтеза со надоврзување,
- параметарска синтеза и
- синтеза со длабоко учење.

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 на синтезата на говор и кој има потенцијал да обезбеди висок квалитет на својот излез. Создавањето на квалитетен модел и прецизното мапирање на движењата на неговите составни делови при артикулацијата е сложен процес што 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. 8.7. Конечно, промените во протокот на воздух низ ларинксот предизвикани од треперењето на гласилките се сложени и се подложни на повратна спрега од вокалниот тракт. Поради тоа, се применуваат едноставни модели за нивното моделирање. Сите овие поедноставувања доведуваат до неприродност на синтетизираниот говор.

Глава 8. Синтеза на звук и говор



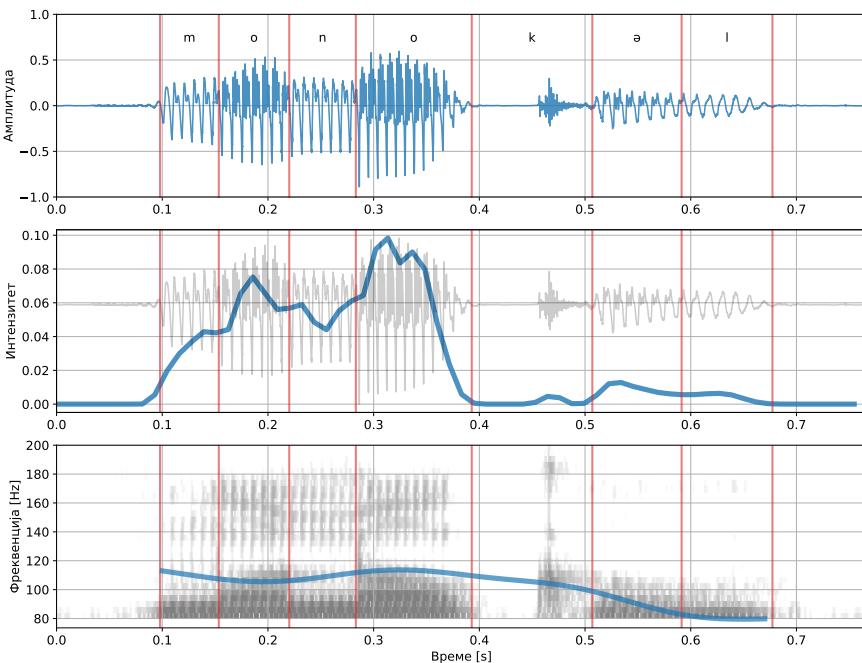
Сл. 8.7: Изглед на 3Д моделот на вокалниот тракт во TTS системот базиран на артикулататорна синтеза на Биркхолц.¹⁶

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 Глава 7.2 и кој е прикажан на Сл. 7.7. Моделот извор–филтер претставува еден од најмоќните 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 говор. Всушност истиот модел се користи за синтеза на звучниот сигнал и кај артикулатортната синтеза.

Покажано е дека со внимателен избор на параметрите на моделот може да се достигне висок квалитет на синтетизираниот говор, но сепак поради едноставноста на употребениот модел за изворот, овие системи не звучат воопшто природно. Дополнително, поради тоа што моделирањето на промената на преносните карактеристики во говорот е макотрпен и скап процес, овој тип на синтеза денес не се применува.

¹⁶Vocal Tract Lab <http://www.vocaltractlab.de/>



Сл. 8.8: Сегментација на говорен сигнал на зборот „мокр“ на ниво на гласови со приказ на брановиот облик (горе), интензитетот (средина), и спектарот и висината (долу).¹⁷

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. На Сл. 8.8 може да се види сегментација на говорен сигнал добиен за зборот „мокр“ на ниво на гласови.

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 преми-

¹⁷ Сликата е превземена од вториот том Супрасегментална фонетика и фонологија, на изданието Фонетика и фонологија на македонскиот стандарден јазик <http://ical.manu.edu.mk/index.php/publications>

нот меѓу нив. 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 1000.¹⁸ Според типот на базата TTS системите можат да се поделат и на системи со еднозначни и системи со проширени бази. Така, еднозначните бази содржат по една верзија од секој двоглас, односно единка. Расположливоста на само по една гласовна единка изискува употреба на алгоритми за дигитално процесирање на сигналите за нивно обликување при надоврзувањето, што негативно се одразува на природноста на синтетизираниот говор.

Поради ова, напредните TTS системи со надоврзување користат проширени бази кои содржат повеќе единки за дадена фонетска содржина. Ова 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-единка¹⁹.

Архитектурата на еден генеричен TTS систем со надоврзување е прикажана на Сл. 8.9. Системот се состои од два главни модули: за анализа на текстот и 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 придржува информации за интонацијата и за времетраењето на секој од гласовите. Задачата на модулот за синтеза на звук е да го синтетизира говорот врз база на фонетската содржина со 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

¹⁸Имено, теорискиот максимум на бројот на двогласи е квадрат од бројот на основни гласови во еден јазик. Меѓутоа, поради правилата на фонотактиката, која ги проучува законите 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.

¹⁹Англ. *unit-selection*.



Сл. 8.9: Архитектура на еден TTS систем со надоврзување.

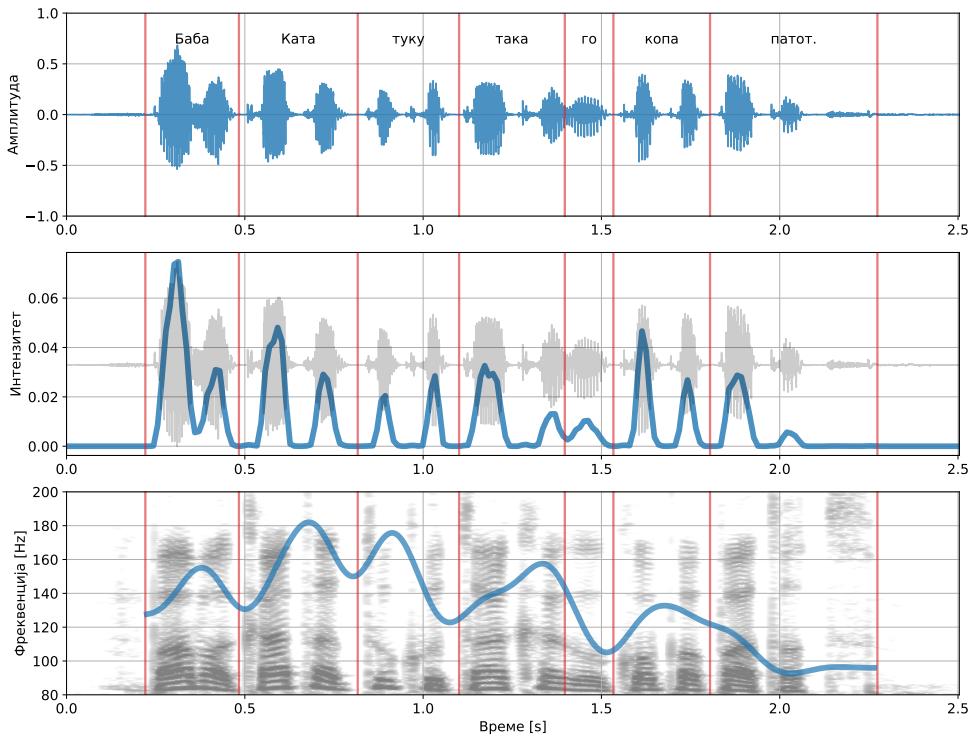
да ги надминат овие ограничувања. Тие го синтетизираат говорот со моделирање на основната фреквенција, амплитудата, времетраењето и спектралната содржина. На овој начин, овозможуваат големи можности за контрола на генерираниот говор, на пример промени во интонацијата и брзината, без потреба да се снимаат нови бази на гласовни единки.

Системите за параметарска синтеза се во основа инверзни на системите за автоматско препознавање говор (ASR), претворајќи го текстот во акустички обележја, а потоа во бранови облици на говор. За моделирањето на акустиката тие користат комбинација на **Скриени модели на Марков (HMM)**²⁰ и **Гаусови мешавински модели (GMM)**²¹, 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. 8.10.

Главниот недостаток на системите за параметарската синтеза е што генериралиот говор звучи вештачки, односно роботски, што се должи на ограничувањата на употребените вокодери. Дополнително, синтетизиралиот говор може да звучи монотон, поради ефектот на усреднување што

²⁰Англ. *Hidden Markov Models*.

²¹Англ. *Gaussian Mixture Models*.



Сл. 8.10: Акустички обележја кај говорот: спектар, интензитет, висина и времетраење, на ниво на исказ.²²

го имаат искористените методи за моделирање. Сепак, поради нивната мала пресметковна цена и потребен мемориски простор, системите за параметарска синтеза сè уште се употребуваат во многу апликации кои имаат потреба од локална синтеза на говор на вгнездени и мобилни уреди, на прво системите за навигација, уредите за преведување, мобилните телефони и сл.

Овие системи сè уште се употребуваат и во склоп на асистивните технологии, на пример говорните табли, како и за синтеза на говор во склоп на читачите на екран кои ги користат луѓето со визуелна попреченост. За последните, од најголемо значење е можноста за контрола на синтетизираниот говор, особено зголемување на неговата брзина, а притоа сочувувајќи ја разбираливоста – природноста е од второстепено значење.

²² 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>

Синтеза со длабоко учење

Со појавата на длабокото учење, види Глава 9.1, се појави и нова парадигма за синтеза на говор. Како и во многу други области, длабокото учење и тутка донесе своевидна револуција во поглед на квалитетот и природноста на синтетизираниот говор. Навистина, овие системи се способни да генерираат говор што звучи многу природно, на моменти дури и без разлика од природниот говор.²³

Системите за синтеза на говор со длабоко учење се базирани на длабоките невронски мрежи и тоа рекурентните невронски мрежи (RNN)²⁴, односно мрежите LSTM²⁵, кои се способни да моделираат временски секвенции. Во поново време, рекурентните мрежи речиси во целост се заменети со новите архитектури на невронски мрежи наречени трансформери²⁶, 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 говор кој го има во изобилство. Уште една предност е нивната способност за преносно учење²⁷, што овозможува употреба на модели што се предтренирани со јазици со многу ресурси за јазици со малку ресурси преку нивно нагодување²⁸.

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

²³ Прв систем кој го постигна ова беше системот Tacotron 2 во 2016 г. <https://google.github.io/tacotron/publications/tacotron2/index.html>.

²⁴ Англ. Recurrent Neural Networks.

²⁵ Англ. Long Short-Term Memory.

²⁶ Англ. Transformers.

²⁷ Англ. transfer learning.

²⁸ Англ. fine-tuning.

вклучуваше целосно функционален модул за генерирање на интонација и ритам дизајниран за македонски јазик.²⁹

Во поново време, Групата за говорни технологии на ФЕИТ (Говор@ФЕИТ) го разви гласот „Везилка“ базиран на синтеза со длабоко учење, со кој постигнаа врвни резултати во поглед на природноста и квалитетот на синтетизираниот говор.³⁰ За примена во вгнездените системи, разивме 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 посложените вокодери, но е нужен компромис за примена на целосниот систем на вгнездена платформа.³¹

Конечно, во соработка со Здружението за асистивни технологии „Отворете ги прозорците“, а со поддршка од македонската канцеларија на УНИЦЕФ, разивме 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.³² Исто така, тој може да се користи и во говорната табла Cboard, која ја локализиравме за македонски и албански во рамките на истата соработка.

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 А. Наместо мануелно да ги издвоиме буквите, ќе искористиме лабели за да ги означиме во снимката. Ова се прави преку додавање на трака за лабели во која ќе ги означиме секоја од буквите.

На Сл. 8.11 е прикажана снимка на азбуката со три означени букви: „и“, „ј“ и „к“. Внимавајте при означување на самогласките да означите дел од средината на самогласката, каде што таа е најгласна и најстабилна. Уште повеќе, кај согласките внимавајте да не ја означите самогласката /ə/, која ја има на пр. на почетокот на ’рбет, а која ја додаваме на согласките при изговарање на азбуката. На крајот, зачувајте ја снимката во wav формат, а лабелите во txt формат.

Ако ја отвориме датотеката со лабелите ќе видиме дека таа ја има следната структура:

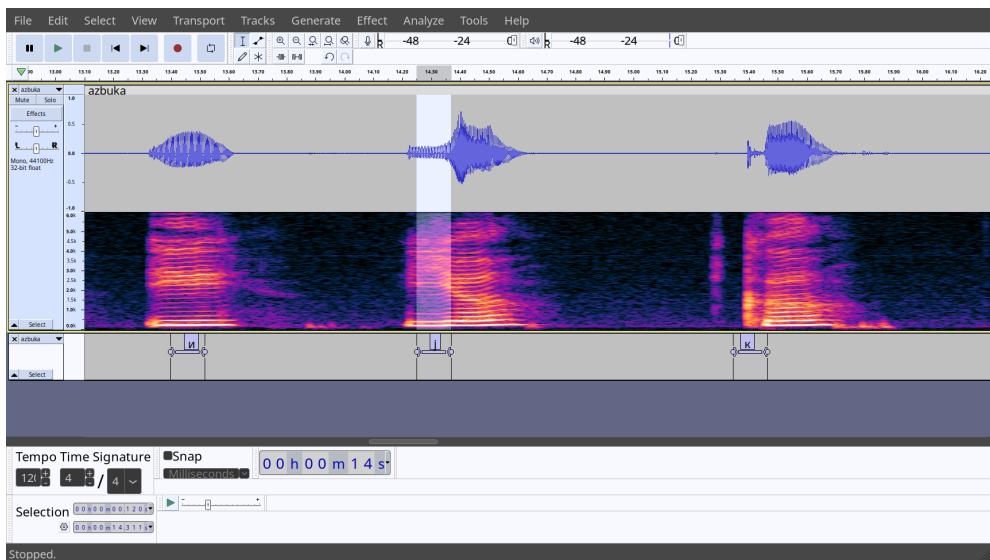
²⁹https://speech.feit.ukim.edu.mk/tts_zboruvaj.html

³⁰https://speech.feit.ukim.edu.mk/tts_vezilka.html

³¹<https://speech.feit.ukim.edu.mk/ttslite.html>

³²https://speech.feit.ukim.edu.mk/tts_suze.html

8.3. Синтеза на говор



Сл. 8.11: Снимка на азбуката со означени букви.

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 ознаката, втората колона е времето на крај на ознаката, а третата колона е буква која е означена. Колоните се одвоени со таб . 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 = {}
with open(pjoin(audio_path, lab_name), 'r') as f:
    for line in f:
```

Глава 8. Синтеза на звук и говор

```
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.³³

```
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 надоврзeme со нивно прелевање³⁴.

```
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(unit, fs, 'in', t_fade)
    unit = da.fade(unit, fs, 'out', 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)
```

Тука `da.fade()` е функција во нашиот модул `da.py` за реализација на втишување на почетокот или на крајот од сигналот дадена со:

```
def fade(wav, fs, fade='out', t_fade=0.050):
    '''Apply fade effect to the audio signal.
```

Parameters

³³Би можеле наместо бел шум да додадеме Оти вектор, но тогаш ќе имаме проблеми при пресметување на логаритамот на спектарот на сигналот за прикажување на спектрограмот.

³⁴Англ. *cross-fade*.

```
wav : array_like
      Audio signal.

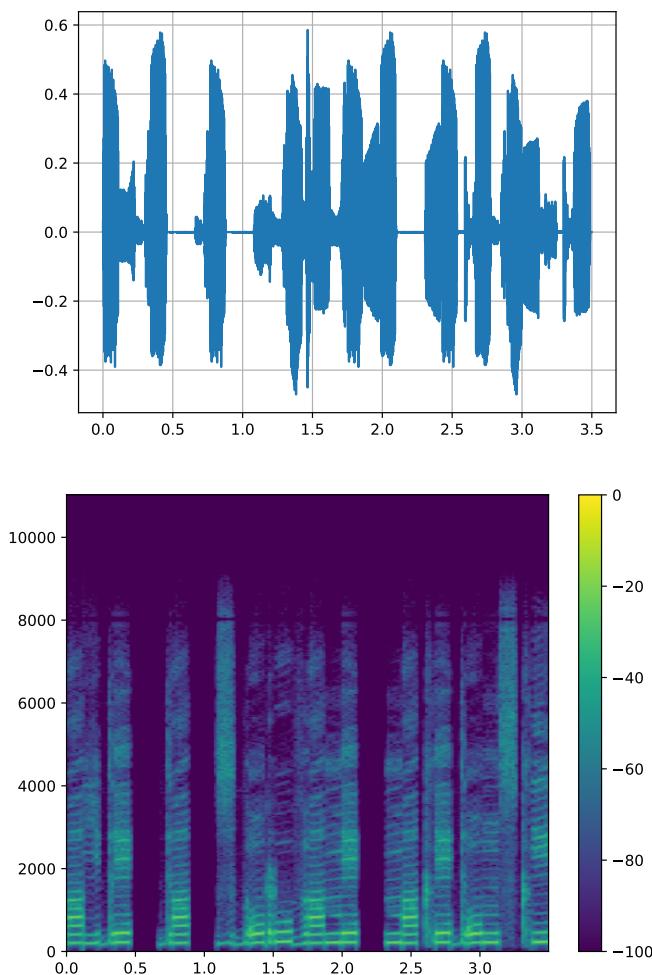
fs : int
      Sampling frequency [Hz].

fade : str, optional
      Fade-in or fade-out. The default is "out".

t_fade : float, optional
      Fade duration [s]. The default is 0.050.

>Returns
-----
array_like
      Audio signal with fade effect applied.
...
fade_len = int(t_fade * fs)
if fade == 'in':
    lin = np.linspace(0, 1, fade_len)
    wav[: fade_len] = wav[: fade_len] * lin
elif fade == 'out':
    lin = np.linspace(1, 0, fade_len)
    wav[-fade_len:] = wav[-fade_len:] * lin
return wav
```

Добиениот синтетизиран говор и неговиот спектрограм се прикажани на Сл. 8.12. Може јасно да се види надоврзувањето на сегментите од аудиосигналите на буквите.



Сл. 8.12: Синтетизиран говор со Python и неговиот спектрограм.

Глава 9

Препознавање на звук и говор

Машинското учење, а особено длабокото учење, е метода што стана доминантна во голем број инженерски и научни области. Во дигиталните аудиосистеми, машинското учење не само што ги поедностави комплексните решенија базирани на процесирање на сигналите и имплементација на експертски правила, туку ги реши и проблемите што не можеа да бидат воопшто решени. Еден од овие проблеми е и синтезата на природен говор, кој го разгледавме во претходната глава. Во оваа глава 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.

9.1 Основи на машинското учење

Машинското учење (ML)¹ е подобласт на вештачката интелигенција (AI)². Тоа опфаќа модели чии параметри се адаптираат на множество на податоци низ процес наречен тренирање, односно учење (Bishop, 2006; Müller et al., 2016). На највисоко ниво разликуваме три парадигми:

- учење со надзор³,
- учење без надзор⁴, и
- насочено учење⁵.

¹Англ. *Machine Learning*.

²Англ. *Artificial Intelligence*.

³Англ. *supervised learning*.

⁴Англ. *unsupervised learning*.

⁵Англ. *reinforcement learning*.

Учење со надзор

Кај учењето со надзор моделите се тренираат со познати целни излезни секвенции, уште наречени **основна вистина**⁶. Според видот 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 излезот што 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 на музички инструмент и препознавање на аудиосцена. Од друга страна, примери за модели за регресија се алгоритмите за: синтеза на говор, синтеза на интонација, ритам и спектар 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, што целиот процес го прави долготраен 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. Најпознат алгоритам за оваа намена е алгоритмот на **K-средни вредности**⁷, кој го дели множеството податоци во K кластери описани со средната вредност на примероците во секој кластер, наречени и центроиди, и
- **намалување на дименционалноста** – алгоритми кои служат за проекција на влезните обележја во простор со помалку димензии, при тоа задржувајќи најголем дел од информацијата. Тие се применуваат за намалување на редундансата во влезните обележја со што се поедноставуваат системите кои ги користат како влез. Уште една голема примена овие методи наоѓаат за визуелизација на распредел-

⁶Англ. *ground truth*.

⁷Англ. *K-means*.

бата⁸ на повеќедимензионални податоци во две или три димензии преку нивна трансформација. Анализата на принципиелни (главни) компоненти (PCA)⁹ е еден пример за егзактна математичка метода за оваа намена, која е и реверзибилна. Од алгоритмите за ML тук спаѓаат стохастичките алгоритми базирани на PCA, а најпознат е алгоритмот за *t*-дистрибуираното стохастичко врамување на соседи (t-SNE)¹⁰.

Насочено учење

Тука спаѓаат алгоритмите за играње на компјутерски игри кои неодамна ги надиграа најдобрите човечки играчи¹¹, но и алгоритмите за контрола на агенти во виртуелни симулации кои треба да научат некоја задача, на пр. како оптимално да го придвижат човечкото тело за да стигнат од точка А до точка Б¹².

Тие имаат крајна цел, но таа не е доволна за директно тренирање на моделот. Имено, 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 алгоритмот долж играта, со цел учењето да се насочи во правата насока.

На пример, влезот на алгоритмот 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 до 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 алгоритмот се пушта да игра самиот против себе милиони пати сè додека не научи да победува.

Чекори во примената на машинското учење

Постојат низа од чекори кои се треба да се следат при употребата на алгоритмите за машинско учење за решавање на даден проблем:

1^o анализа на влезните податоци / сигнали / слики – во оваа фаза се анализира содржината на влезните податоци, се одредува дали постојат некомплетни податоци, се прави квалитативна анализа и визу-

⁸Англ. *manifold*.

⁹Англ. *Principal Component Analysis*.

¹⁰Англ. *t-distributed Stochastic Neighbor Embedding*.

¹¹Во оваа категорија спаѓаат алгоритмите кои вообичаено се поистоветуваат со областа вештачка интелигенција. Google DeepMind: Ground-breaking AlphaGo masters the game of Go <https://www.youtube.com/watch?v=SUbqykXVx0A>

¹²Google's DeepMind AI Just Taught Itself To Walk <https://www.youtube.com/watch?v=gn4nRCC9TwQ>

елизација на поединечни примероци од базата на податоци, како и квантитативна анализа на нивната распределба и карактеристики,

- 2^o поделба на множества за тренирање, валидација и тестирање – дел од податоците се одвојува за конечна оценка (тестирање) на истренираниот модел, додека дел од остатокот се одвојува за валидација во фазата на тренирање,
- 3^o претпроцесирање и екстракција на обележја – во оваа фаза се прави скалирање или нормализација на влезните податоци, водејќи сметка притоа да не се искористи статистиката на множеството за тестирање. Во случај на употреба на податоци со голем број димензии, на пример кога се работи со аудиосигнали, вообичаено во оваа фаза се врши екстракција на обележја¹³. За ова може да се применат методи како PCA, пресметка на спектар, спектограми итн. Најпознатите обележја за препознавањето на говор се мел-Фреквенциските кепстрални коефициенти (MFCC),¹⁴
- 4^o тренирање на моделот – оптимизација на параметрите на моделот θ , како и хиперпараметрите на моделот и процесот на тренирање, со употреба на множеството за тренирање и валидација,
- 5^o тестирање на моделот – ова се прави со одвоеното множество за тестирање со цел да се оцени моќта за генерализација на истренираниот модел, односно тоа како работи моделот за множество од нови, невидени влезни податоци.

Тренирање на моделите за машинско учење

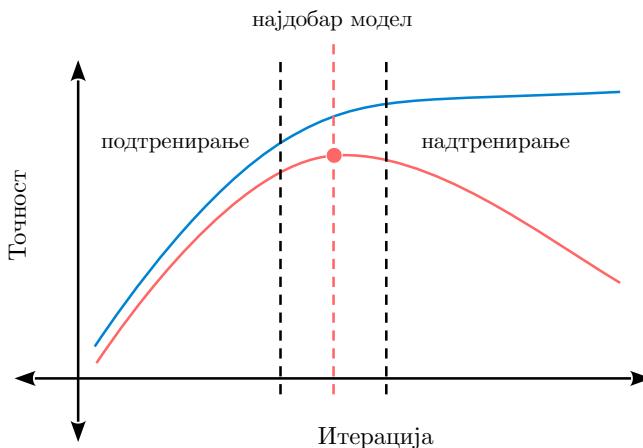
Тренирањето на моделот, односно нагодувањето на неговите параметри се врши со помош на множеството на тренирање. Овој процес се изведува итеративно. Овде ќе претпоставиме дека се работи за алгоритам за учење со надзор, за кој се познати точните вредности што сакаме да ги добиеме на неговиот излез. Во секоја итерација на учењето се пресметува грешката која моделот ја прави врз база на познатите излези. Врз база на пресметаната грешка, се адаптираат параметрите на моделот θ во насока на намалување на оваа грешка. Чекорот со кој се врши адаптацијата на параметрите се нарекува и чекор на учење¹⁵.

Тренирањето вообичаено завршува кога грешката на моделот не се подобрува за одредено ниво на толеранција последователно во одреден број на итерации, или пак ако е постигнат максималниот дозволен број на итерации. Ако тренирањето запре поради постигнување на максималниот

¹³Англ. *feature extraction*.

¹⁴Англ. *mel-frequency cepstral coefficients*.

¹⁵Англ. *learning rate*.



Сл. 9.1: Промена на точноста на моделот на множеството за тренирање (сино) и множеството за валидација (црвено) за време на тренирањето.

број на итерации, а не дојде до заситување на опаѓањето на грешката на моделот, тогаш велиме дека настанало подтренирање¹⁶, односно дека алгоритмот има потенцијал да се дотренира за подобрување на неговите перформанси.

Проблемот со овој начин на тренирање е 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 невидени влезни податоци. Оваа појава се нарекува надтренирање или пренагодување¹⁷ и претставува посебен проблем во тренирањето на моделите за машинско учење. Појавите на подтренирање и надтренирање се претставени на Сл. 9.1.

Постојат три главни начини за избегнувањето на надтренирањето на моделите за машинско учење:

- ограничување на моќта на моделот – поедноставни модели не можат доволно да опишат покомплексни влезни податоци и сигнали,
- регуларизација – низа на методи што ја ограничуваат способноста на моделите да опишат комплексни податоци без намалување на нивната моќ. Степенот на ограничување се нагодува преку коефициентот на регуларизација,
- рано запирање¹⁸ – постапка што ја оценува грешката која ја прави моделот на множеството за валидација на секоја итерација и го запира тренирањето кога таа ќе започне да расте.

¹⁶Англ. *underfitting*.

¹⁷Англ. *overfitting*.

¹⁸Англ. *early stopping*.

Оптимизација на хиперпараметри

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 параметри се нарекуваат **хиперпараметри**.

Оптимизацијата на хиперпараметрите се прави преку тестирање на перформансите на тренираниот модел на множеството за валидација. При употреба на мали множества на податоци, за да се намали влијанието на тоа кои податоци влегуваат во состав на множеството за валидација, вообичаено се прави оптимизација на хиперпараметрите во јамка за **меѓувалидација** или **вкрстена-валидација**¹⁹.

За реализација на *K*-кратна меѓувалидација²⁰ се преземаат следните чекори:

- 1^o се дели влезното множество на податоци на множества за тренирање и тестирање,
- 2^o множеството за тренирање се дели на *K* подмножества од кои секое итеративно се зема за множество за валидација во јамката за меѓувалидација, а останатите се употребуваат за тренирање, како што е прикажано на Сл. 9.2.

Постојат низа проблеми што можат да настанат при употреба на овој едноставен пристап за меѓувалидација. Секој од нив се решава преку приспособување на изборот на податоците во подмножествата за меѓувалидација:²¹

- ако датасетот е подреден по класите што треба да се препознаат тогаш може да се случи некоја од класите да ја има само во тест множеството; поради ова вообичаено се употребува **мешање**²² 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 со одредувањето на подмножествата врз база на групите податоци, т.н. *K*-кратна меѓувалидација по групи²³ на датасетот,

¹⁹Англ. *cross-validation*.

²⁰Англ. *K-fold cross-validation*.

²¹scikit-learn – 3.1. Cross-validation: evaluating estimator performance https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

²²Англ. *shuffle*.

²³Англ. *group k-fold cross-validation*.



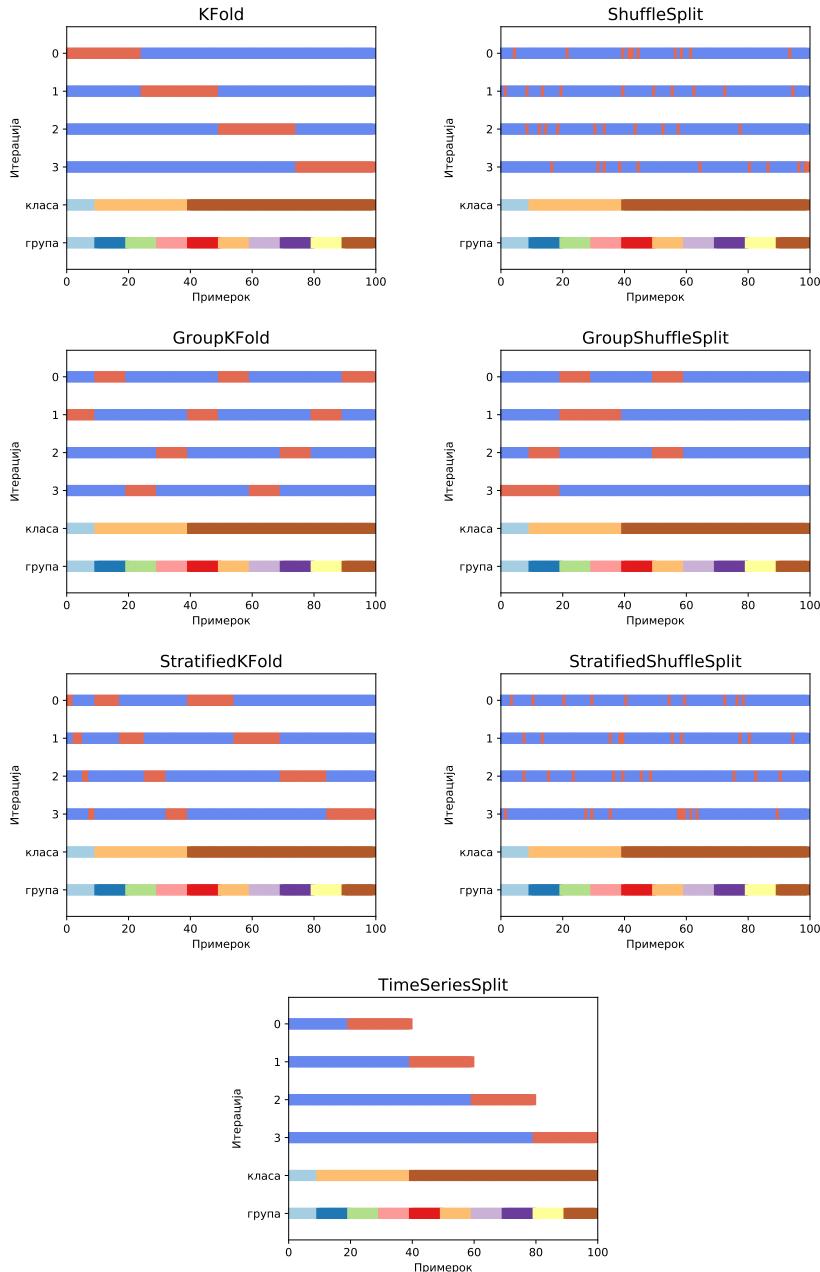
Сл. 9.2: Тренирање на модел со помош на K -кратна меѓувалидација за $K = 5$.

- 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 стратификација**²⁴, и
- во проблеми со временски серии во кои треба да се предвиди иднината врз база на минатите податоци, едноставната поделба на податоците во подмножества може да ги направи информациите од иднината достапни за алгоритмот; 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. 9.3.

²⁴Англ. *stratified k-fold cross-validation*.

²⁵Генерирано со приспособен код од оној на сајтот scikit-learn –Visualizing cross-validation



Сл. 9.3: Методи за K -кратна меѓувалидација за $K = 4$. Прикажани се множеството за тренирање (сино) и множеството за валидација (првено) за секоја итерација од јамката за меѓувалидација.²⁵

По завршување на меѓувалидацијата, моделот со хиперпараметрите со најдобри перформанси се тренира повторно врз целото множеството на влезни податоци, вклучувајќи ги податоците од множеството за валидација, а изоставувајќи ги оние од множеството за тестирање. Вака тренираниот модел се евалуира на тест множеството.

И тука, за избегнување на влијанието на изборот на тоа кои податоци се дел од множеството за тестирање врз проценката на перформансите на моделот, може да се направи меѓуоценка на моделот преку употреба на уште една, надворешна јамка на меѓувалидација. Во овој случај, велиме дека станува збор за **вгнездена меѓувалидација**. Овој процес се реализира на следниот начин:

- 1^o целото множество на податоци најпрвин се дели на J подмножества од кои секое се користи како тест множество во надворешната јамка,
- 2^o остатокот од податоците се дели на K 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)**²⁶. Основната градбена единка на невронските мрежи се вештачките неврони чиј дизајн е инспириран од анатомијата и физиологијата на биолошките неврони прикажани на Сл. 9.4.

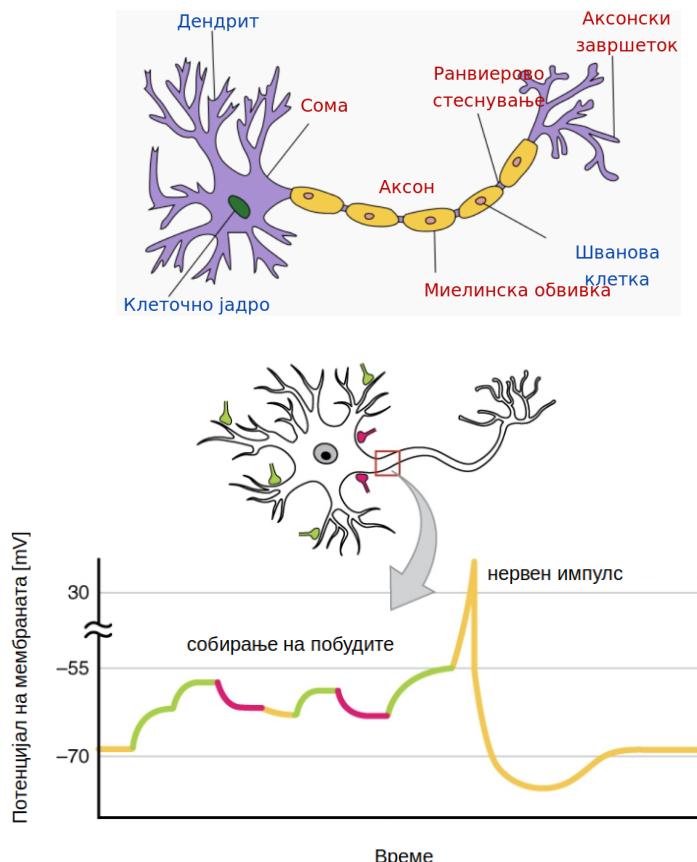
Основната функција на биолошките неврони е да реагираат на побуда на нивните влезови со генерирање на нервни импулси, уште наречени акциски потенцијали. Притоа, побудите можат да бидат позитивни или екситаторски – оние кои ја зголемуваат електричната поларизација на невронот, и негативни или инхибиторски – оние кои ја намалуваат неговата поларизација. Кога сумата на побуди ќе го надмине напонот на праг, во невронот ќе настане брза промена на потенцијалот, наречена деполаризација, која ќе пропагира долж излезниот аксон во вид на нервен импулс. На крајот на невронот, овој импулс ја игра улогата на побуда за друг неврон, мускулно влакно и др.²⁸

Вештачки неврон

Структурата на еден вештачки неврон е прикажана на Сл. 9.5. Секој неврон има K влезови на кои се носи влезниот вектор $\mathbf{x} = [x_0, x_1, \dots x_{K-1}]$.

²⁶Англ. *Neural Networks*.

²⁸Повеќе за работата на биолошките неврони и процесот на деполаризација може да прочитате и во материјалите по предметот Биомедицинска електроника <https://gitlab.com/feeit-freecourseware/biomedical-electronics>



Сл. 9.4: Анатомија (горе) и физиологија (долу) на биолошки неврон.²⁷

Влезовите на невронот се скалираат со тежински коефициенти, или **тежини** w_k ²⁹, кои можат да бидат и негативни. Активацијата a на невронот се добива преку сумирање на скалираните влезови. Конечно, излезот на невронот y се одредува преку излезната нелинеарност $f(a)$:

$$y = f(a) = f\left(\sum_{k=0}^{K-1} w_k x_k + b\right) = f(\mathbf{w}\mathbf{x}^T + b) \quad (9.1)$$

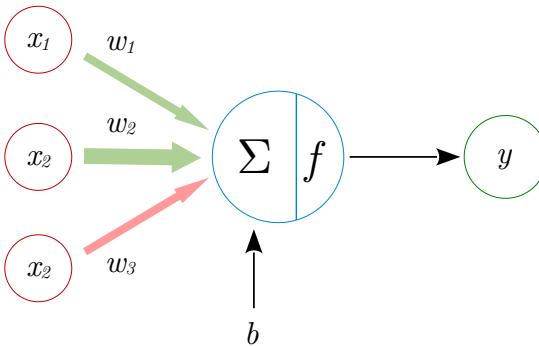
Притоа, вообичаено во сумата се додава и **коефициент на поместување** b ³⁰ кој го одредува прагот на излезната нелинеарност. $\mathbf{w} = [w_0, w_1, \dots, w_{K-1}]$

²⁹Англ. *weights*.

²⁹Wikipedia: Неврон <https://mk.wikipedia.org/wiki/Неврон>, оригинал од Quasar Jarosz на Википедија на англиски, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=7616130>

By OpenStax College - Anatomy & Physiology, Connexions Web site. <http://cnx.org/content/col11496/1.6/>, Jun 19, 2013., CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=30147931>

³⁰Англ. *bias*.



Сл. 9.5: Структура на вештачки неврон. Тежината w_1 е позитивна, w_2 е исто така позитивна и поголема, додека w_3 е негативна.

е векторот на тежини на овој неврон.

§ Дополнително. Може да видиме дека (9.1) всушност претставува поедноставен модел на биолошките неврони. Во него, влезниот сигнал се носи истовремено на сите влезови на невронот, а невронот пак врз база на тежинската сума на влезовите веднаш генерира вредност на излезот. Со ова, моделот не ја зема во обсир 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**³¹. Импулсните невронски мрежи иако не се сè уште во широка употреба, 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 , (9.1) преминува во:

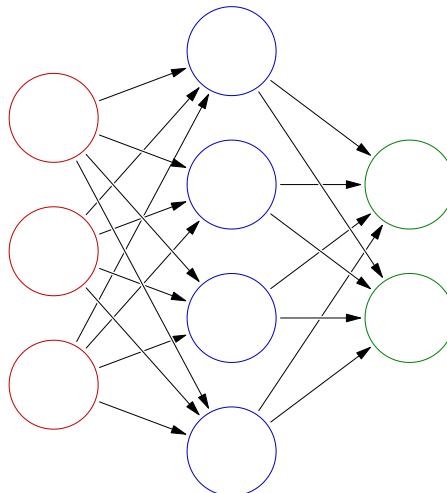
$$y_n = f(a_n) = f(\mathbf{w}\mathbf{x}_n^T + b) \quad \text{за } n = 0, 1, 2, \dots, N - 1 \quad (9.2)$$

$$\mathbf{y} = f(\mathbf{a}) = f(\mathbf{w}\mathbf{X}^T + b) \quad (9.3)$$

каде што со \mathbf{X} е означена матрицата чии редици ги сочинуваат влезните примероци \mathbf{x}_n .

Моделите за регресија базирани на употреба на еден неврон се нарекуваат **линеарна регресија**, додека оние за класификација се нарекуваат

³¹Wikipedia: Spiking Neural Network https://en.wikipedia.org/wiki/Spiking_neural_network



Сл. 9.6: 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 цртаат како влезен слој (црвено).³²

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. 9.6. 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**³³. Оваа едноставна архитектура сепак им овозможува на плитките невронски мрежи да моделираат која било 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 го пресметаме како:

$$\mathbf{y}_h = f_h(\mathbf{a}_h) = f_h(\mathbf{W}_h \mathbf{x}^T + \mathbf{b}_h) \quad (9.4)$$

$$\mathbf{y} = f_o(\mathbf{a}_o) = f_o(\mathbf{W}_o \mathbf{y}_h + \mathbf{b}_o) = f_o(\mathbf{W}_o \cdot f_h(\mathbf{W}_h \mathbf{x}^T + \mathbf{b}_h) + \mathbf{b}_o) \quad (9.5)$$

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

³³Англ. *fully connected* и *dense*.

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>

имаме повеќе неврони, нивните тежини се 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, сè додека не се дојде до излезниот слој на мрежата. Овој процес се нарекува **пропагација нанапред**.³⁴

Повторно, ако на влез се донесат низа на N примероци од влезни податоци \mathbf{x}_n добиваме:

$$\mathbf{Y}_h = f_h(\mathbf{A}_h) = f_h(\mathbf{W}_h \mathbf{X}^T + \mathbf{b}_h) \quad (9.6)$$

$$\mathbf{Y} = f_o(\mathbf{A}_o) = f_o(\mathbf{W}_o \mathbf{Y}_h + \mathbf{b}_o) = f_o(\mathbf{W}_o \cdot f_h(\mathbf{W}_h \mathbf{X}^T + \mathbf{b}_h) + \mathbf{b}_o) \quad (9.7)$$

Со тоа, процесирањето на влезните податоци со невронските мрежки се сведува на операции на множење на матрици и примена на нелинеарни функции. Поради ова, графичките процесори се идеални за работа со невронските мрежки поради нивниот голем број на јадра дизајнирани за матрично множење.



§ Дополнително. Првата имплементација на вештачки неврон, наречена **перцептрон** била изработена од американскиот психолог **Франк Розенблат** во 1957 во Лабораторијата за аеронаутика во Корнел, САД.³⁵ 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 во матрица 20×20 пиксели, имала еден скриен слој со 512 неврони и излезен слој со 8 неврони. Тежините на врските помеѓу скриениот и излезниот слој биле реализирани со потенциометри, а процесот на учење се одвивал преку нивно нагодување со електрични мотори.

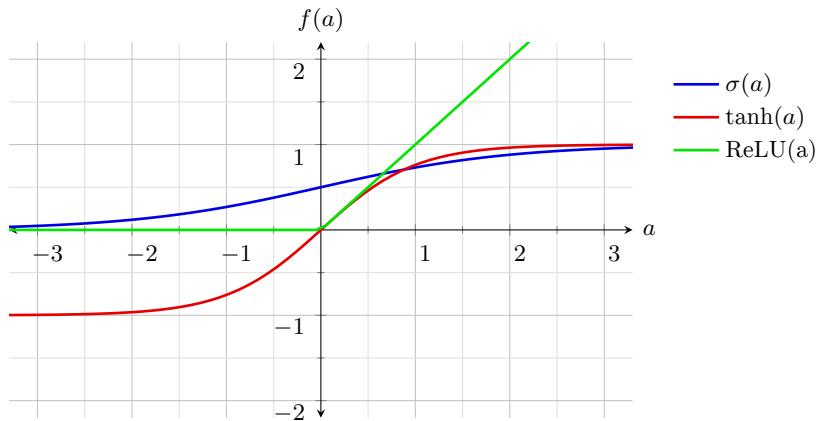
Излезни нелинеарности

Во рамките на една невронска мрежа, излезните нелинеарности на невроните во скриените и излезните слоеви вообичаено се разликуваат. Вообичаени нелинеарности кои се користат во скриениот слој се прикажани на Сл. 9.7. Тие се:

³⁴Англ. *forward pass* или *feed forward*.

³⁵Wikipedia – 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>



Сл. 9.7: Излезни нелинеарности вообичаено во употреба кај невроните во скриениот слој на невронската мрежа.

- сигмоида σ – со излез во опсег $0 - 1$:

$$\sigma(a) = \frac{1}{1 + e^{-a}} \quad (9.8)$$

- тангенс хиперболиум \tanh – со излез во опсег од -1 до 1 :

$$\tanh(a) = \frac{e^{2a} - 1}{e^{2a} + 1}, \text{ и} \quad (9.9)$$

- полубрзанов насочувач ReLU – со поедноставна пресметка на излезот на невроните, подобра пропагација на градиентот во процесот на тренирање, како и зголемената веројатност за неактивирање на невроните³⁶:

$$\text{ReLU}(a) = \begin{cases} a & \text{ако } a > 0 \\ 0 & \text{поинаку} \end{cases} \quad (9.10)$$

Уште една мотивација за употреба на оваа нелинеарност е нејзината аналогија со начинот на кој функционираат биолошките неврони поради нејзината несиметричност.

За невроните во излезниот слој вообичаено се користат:

- сигмоида – за класификација,
- софтмакс – за класификација со повеќе излезни класи J :

$$f(a_j) = \frac{e^{a_j}}{\sum_{j=0}^{J-1} e^{a_j}} \quad (9.11)$$

³⁶Така, при случајна иницијализација на нивните тежини, половина од невроните со ReLU нелинеарност на својот излез ќе дадат 0!

каде што a_j е активацијата на невронот кој соодветствува на класата j ; софтмакс функцијата го нормализира збирот на излезот на сите излезни неврони на 1, па може да се каже дека ни дава апроксимација на веројатноста на секоја од класите³⁷:

$$f(a_j) \approx P(y = j | \mathbf{a}) \quad (9.12)$$

- линеарна – кај моделите за регресија:

$$f(a) = a \quad (9.13)$$

Длабоко учење

Со додавање на повеќе скриени слоеви во невронската мрежа се добиваат **длабоки невронски мрежи (DNN)**³⁸. 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 ширина на мрежата не е толку ефикасно како додавање на повеќе слоеви, односно додавање длабина на мрежата.

Невронските мрежи кои постигнуваат денес натчовечки 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, бидејќи се овозможени од општата достапност на графичките процесори.

Постојат четири основни подвидови на длабоките невронски мрежи:

- **обични длабоки невронски мрежи** – овие се нарекуваат и **повеќеновски перцептрони (MLP)**³⁹ во чест на перцепtronот,
- **конволуциски невронски мрежи (CNN)**⁴⁰ – 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)**⁴¹ – базирани на повратна врска на невроните од скриените слоеви што им дава еден вид на меморија за претходните примероци; се употребуваат за процесирање на текст и на 1Д сигнали, во поново време се истиснати од 1Д CNN и трансформерите, и

³⁷Иако може да ја третираме како апроксимација на веројатноста, софтмакс функцијата не дава вистинска веројатност или сигурност дека влезот е од некоја класа.

³⁸Англ. *Deep Neural Networks*.

³⁹Англ. *multi layer perceptron*.

⁴⁰Англ. *Convolutional Neural Networks*.

⁴¹Англ. *Recurrent Neural Networks*.

- трансформери⁴² – нов вид на длабоки невронски мрежи што се базираат на механизмот на внимание⁴³ за моделирање на временски секвенции.

Тренирање на невронските мрежи

Проблемот на тренирање на невронските мрежи се сведува на итеративно нагодување на тежините и коефициентите на поместување на секој од невроните во насока на намалување на грешката што ја прави мрежата (Nielsen, 2015).

Спуштање по градиентот

Еден начин тоа да се направи е со примена на алгоритмот спуштање по градиентот (GD)⁴⁴, кој е итеративен алгоритам за пронаоѓање на минимумот на дадена функција.

Конкретно, за да дојдеме до минимумот на функцијата $f(x)$ од моментната вредност на нејзиниот аргумент x_i , треба да направиме чекор во насока спротивна на градиентот за таа вредност:

$$x^{i+1} = x^i - \frac{df}{dx} \cdot \Delta x \quad (9.14)$$

каде што со Δx е означен чекорот на промена, односно нагодување на x .

За примена на GD при тренирањето на невронските мрежи дефинираме функција на грешка или функција на загуба \mathcal{L} ⁴⁵:

$$\mathcal{L}(y, \tilde{y}) = \mathcal{L}(y, g(\boldsymbol{\theta}, \mathbf{x})) \quad (9.15)$$

која зависи од целниот, односно точниот излез y^{46} и излезот односно предикцијата \tilde{y} на мрежата:

$$\tilde{y} = g(\boldsymbol{\theta}, \mathbf{x}) \quad (9.16)$$

Предикцијата зависи од влезниот вектор \mathbf{x} и параметрите на невронската мрежа $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = [\theta_0, \theta_1, \dots, \theta_{L-1}] = [\mathbf{W}_0, \mathbf{b}_0, \mathbf{W}_1, \mathbf{b}_1, \dots, \mathbf{W}_{L-1}, \mathbf{b}_{L-1}] \quad (9.17)$$

каде што L е бројот на слоеви на мрежата.

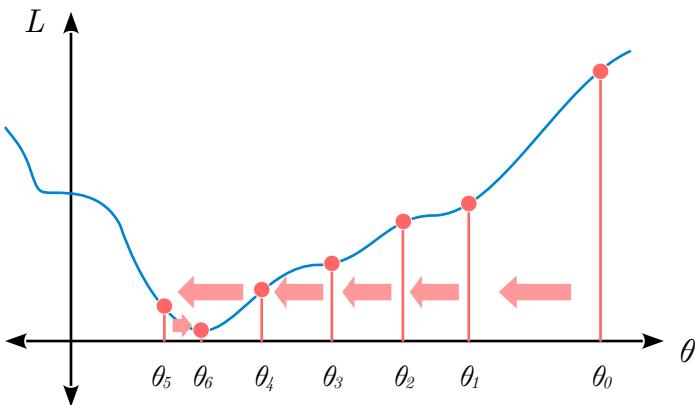
⁴²Англ. *Transformers*.

⁴³Англ. *attention*.

⁴⁴Англ. *gradient descent*.

⁴⁵Англ. *error function* односно *loss function*.

⁴⁶Англ. *ground truth*.



Сл. 9.8: Примена на алгоритмот за спуштање по градиентот за минимизација на функцијата \mathcal{L} преку промена на параметарот θ .

Тогаш (9.14) применета за невронските мрежи ја добива следната форма:

$$\theta_l^{i+1} = \theta_l^i - \frac{\partial \mathcal{L}}{\partial \theta_l} \cdot \eta \quad \text{за } \theta_l \in \boldsymbol{\theta} \quad (9.18)$$

каде што $\frac{\partial \mathcal{L}}{\partial \theta_l}$ е парцијалниот извод на функцијата на грешка во однос на параметарот θ_l , а η е чекорот на учење.

Пример за примена на алгоритмот за спуштање по градиентот е прикажан на Сл. 9.8. Параметарот θ на невронската мрежа е случајно иницијализиран на вредност θ_0 . Бидејќи во оваа точка функцијата на загуба \mathcal{L} стрмо расте, градиентот ќе биде позитивен, па вредноста на θ ќе се намали за поголема вредност. Во следната итерација градиентот е повторно позитивен но помал, па θ ќе се намали, но за помала вредност итн., сè додека θ не ја достигне својата конечна вредност θ_6 .

Во (9.18) изводот е парцијален поради тоа што \mathcal{L} зависи од сите параметри на невронската мрежа $\boldsymbol{\theta}$, а не само θ_l . За неговото пресметување се користи правилото за [извод на сложена функција](#)⁴⁷, односно:

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \frac{\partial \mathcal{L}}{\partial \tilde{y}} \cdot \frac{\partial \tilde{y}}{\partial y_{L-2}} \cdots \frac{\partial y_{l+1}}{\partial \theta_{l+1}} \cdot \frac{\partial y_l}{\partial \theta_l} \quad (9.19)$$

каде што 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:

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{\partial \mathcal{L}}{\partial \tilde{y}} \cdot \frac{\partial \tilde{y}}{\partial w_0} \quad (9.20)$$

Користејќи го правилото за извод на сложена функција, пресметувањето на градиентот за прилагодување на сите параметри $\boldsymbol{\theta}$ на невронската мрежа започнува со пресметка на градиентите за излезниот слој, па

⁴⁷Англ. *chain-rule*.

за последниот скриен слој и оди наназад до почетокот на мрежата. Овој процес се нарекува **пропагација наназад**.⁴⁸

Избор на функција на грешка

За GD да може да се употреби за тренирање на невронски мрежки, мора функцијата на грешка, како и сите излезни нелинеарности на невроните во мрежата да бидат диференцијабилни. Во спротивно, не би можел да се пресмета градиентот за секој од параметрите. Ова е причината за употреба на сигмоидата наместо излезна функција со скок во вредноста на прагот:

$$y = f(a) = \begin{cases} 1 & \text{ако } a > 0,5 \\ 0 & \text{поинаку} \end{cases} \quad (9.21)$$

Денес најчесто се употребуваат следните функции за грешка:

- **средна квадратна грешка** – основна функција на грешка за регресија и бинарна класификација:

$$MSE = \frac{1}{N} \sum_{n=0}^{N-1} (y - \tilde{y})^2 \quad (9.22)$$

- **меѓу-ентропија**⁴⁹ – нејзиниот извод има подобри карактеристики при употреба на сигмоида како излезна нелинеарност:

$$CE = -\frac{1}{N} \sum_{n=0}^{N-1} y \ln \tilde{y} + (1 - y) \ln(1 - \tilde{y}) \quad (9.23)$$

- **логаритамска веројатност**⁵⁰ – кај моделите со функција софтмакс на излез:

$$LL = -\ln \tilde{y} \quad (9.24)$$

Регуларизација

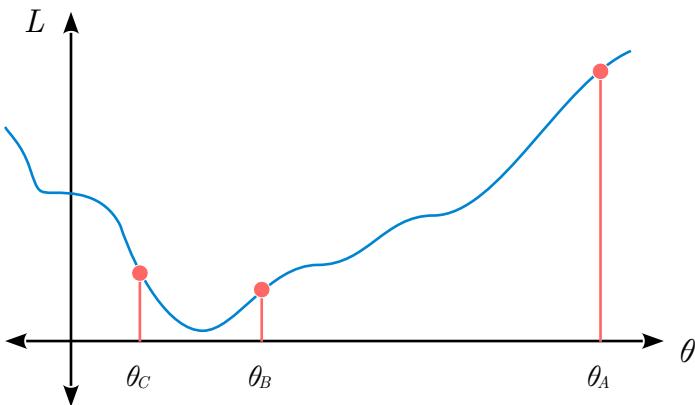
Во рамките на функцијата на грешка се вклучува и дел за регуларизација со којшто се ограничува моќта на моделот за да се избегне надтренирање. Најчесто тоа е регуларизација на L_2 нормата на параметрите на моделот $\boldsymbol{\theta}$. На пример, при употреба на средната квадратна грешка, би имале:

$$\mathcal{L}(y, \tilde{y}, \boldsymbol{\theta}) = \mathcal{L}(y, \tilde{y}) + \lambda \sum_{l=0}^{L-1} (\mathbf{W}_l^T \mathbf{W}_l + \mathbf{b}_l^T \mathbf{b}_l) \quad (9.25)$$

⁴⁸Англ. *backpropagation*.

⁴⁹Англ. *cross-entropy*.

⁵⁰Англ. *log-likelihood*.



Сл. 9.9: Влијание на иницијализацијата на параметарот θ врз брзината на конвергенција на алгоритамот за спуштање по градиентот.

каде што λ е коефициентот 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 .

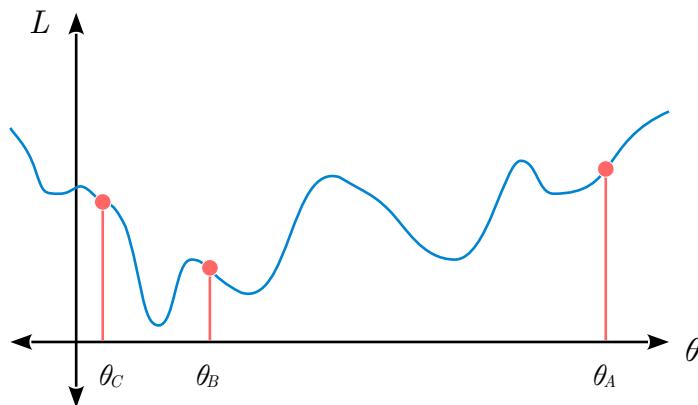
Иницијализација на параметрите на мрежата

На Сл. 9.8 може да се види дека иницијалната вредност на параметарот θ влијае врз брзината на конвергенција на алгоритмот за спуштање по градиентот. Ова е илустрирано на Сл. 9.9. На неа, ако θ е иницијализиран на вредност близку до минимумот на функцијата на грешка на пр. θ_B или θ_C , алгоритмот ќе заврши во помалку итерации, наспроти случајот кога θ е иницијализиран на вредност далеку од минимумот како θ_A .

Проблемот со изборот на почетна точка е уште поголем ако функцијата на загуба \mathcal{L} има посложена зависност од параметарот θ , односно ако дополнително на глобалниот минимум постојат локални минимуми, како што е прикажано на Сл. 9.10. Тогаш, изборот на почетна вредност на θ може да биде од суштинско значење за оптимално тренирање на мрежата. Од трите почетни вредности на сликата, само од θ_C можеме преку спуштање по градиентот да стигнеме до оптималната вредност на θ за која \mathcal{L} ја има најмалата вредност што се нарекува глобален минимум. Во спротивно, ако започнеме од θ_B или θ_A процесот на тренирање ќе заврши со вредности на θ за кои вредноста на \mathcal{L} има локален минимум.

Поделба на купчиња

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 нагодувањето на параметрите да се прави со градиентот пресметан за



Сл. 9.10: Влијание на иницијализацијата на параметарот θ на завршувањето во локален наспроти глобален минимум на функцијата на грешка.

секој од влезните податоци земен по случаен избор. Оваа варијанта на алгоритмот за спуштање по градиентот се нарекува **стохастичко спуштање по градиентот (SGD)**⁵¹.

Притоа, изминувањето на целото множество за тренирање се нарекува **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 естимација на вистинскиот градиент на функцијата на грешка. Поради тоа, најчесто се употребува компромисно решение во кое се зема подмножество, или **купче**⁵² примероци од множеството за тренирање по случаен избор, се пресметува функцијата на грешка за купчето и се врши адаптација на параметрите на моделот со пресметаниот градиент.

Оваа верзија на GD алгоритмот се нарекува **спуштање по градиентот со купчиња (MBGD)**⁵³. 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 алгоритмот MBGD, но под името SGD.

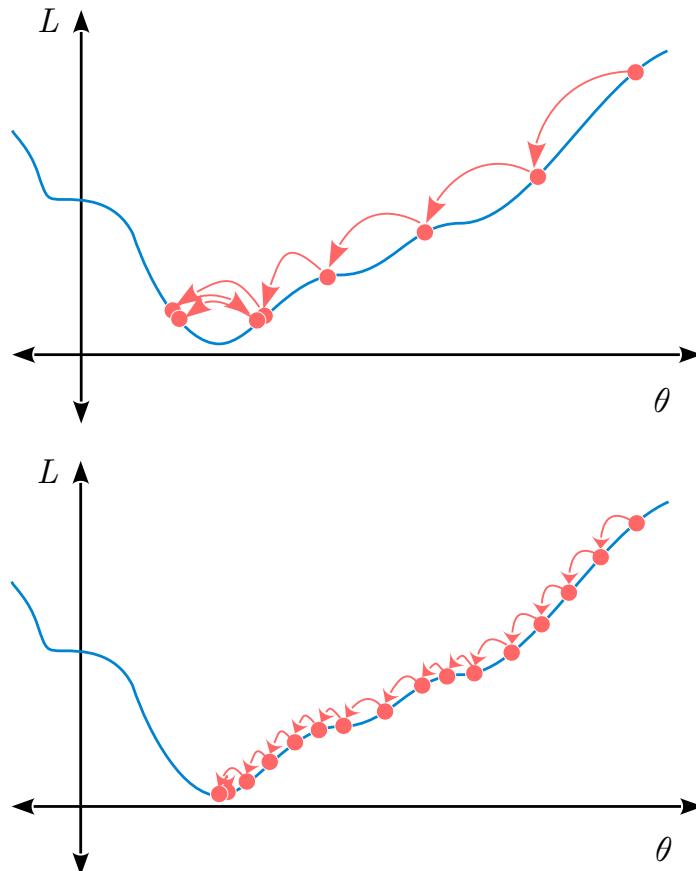
Чекор на учење

При употреба на SGD, чекорот на учење е еден од најважните параметри во тренирањето на невронските мрежи. Ако е преголем,

⁵¹Англ. *stochastic gradient descent*.

⁵²Англ. *batch* или *mini-batch*.

⁵³Англ. *mini batch gradient descent*.



Сл. 9.11: Влијание на чекорот на учење врз брзината на конвергенција на алгоритмот за спуштање по градиентот.

оптимизацијата може да го натфрли минимумот на функцијата на грешка, додека пак, ако е премал, на алгоритмот ќе му бидат потребни многу итерации за да конвергира тренирањето. Ова е илустрирано на Сл. 9.11.

Затоа вообичаено се употребуваат алгоритми на промена на чекорот на учење⁵⁴. Една едноставна, а често употребувана стратегија е чекорот да се намалува во текот на тренирањето. Постојат и понаредни алгоритми, каков што е Адам, кои ги земаат предвид првиот и вториот момент, односно брзината и забрзувањето на промената на градиентот во однос на претходните итерации, за прилагодување на чекорот на учење.

Еден голем проблем во употребата на SGD алгоритмот е можноста алгоритмот да заглави во локален минимум, по цена на промашување на глобалниот минимум. Овој ризик е поголем кај помалите NN. Во длабокото учење, веројатноста за постоење на локален минимум е незначителна.

⁵⁴Англ. *learning rate scheduling*.

Кај нив претставува проблем заглавувањето на SGD во рамни региони на функцијата на грешка.

Двата проблеми на заглавување во локален минимум или во рамен регион на функцијата на грешка можат да се решат преку реиницијализација на чекорот на учење на голема вредност по одреден (голем) број итерации.

9.2 Препознавање на звучен извор

Во овој дел ќе примениме невронска мрежа за препознавање на звучен извор⁵⁵ врз база на аудиосигналот што тој го генерира. За таа цел, ќе искористиме дел од аудиодатотеките кои ги користевме досега, а дел ќе позајмиме од предметот Електроакустика на ФЕИТ.⁵⁶ Притоа, ако аудиосигналот е подолг од 5 s истиот ќе го скратиме за да не доминира во множеството на податоци.

```
import numpy as np
from matplotlib import pyplot as plt

audio_path = 'audio/'
file_names = [
    'Glas.wav',
    'Solzi.wav',
    'Violina.wav',
    'Tapan.wav',
    '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)
```

Анализа на влезните податоци

За запознавање со аудиосигналите што се дел од податочното множество што го создадовме, може да го искористите знаењето кое досега го стекнавте на предметов. За поголема компактност, тута директно ќе прејдеме на поделбата на датасетот на подмножества.

⁵⁵Англ. *sound source recognition (SSR)*.

⁵⁶<https://github.com/FEEIT-FreeCourseWare/Electroacoustics>

Поделба на множества

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 тестирање.

```
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 пракса е кога се работи за препознавање на звучните извори или на пример за препознавање на говор⁵⁷ или говорник⁵⁸, 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 хармониците врз невронската мрежа. Всушност, на овој начин ќе обезбедиме мрежата да ја научи спектралната анвелопа, односно бојата на тонот, наместо неговата висина и хармониците.

Во исто време, ќе ги кодираме лабелите со бинарни вектори со една 1 на позицијата на класата што 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. Во излезниот слој од невронската мрежа ќе има толку неврони колку што има класи, а секој од нив ќе 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.⁵⁹

⁵⁷Англ. *automatic speech recognition (ASR)*.

⁵⁸Англ. *automatic speaker recognition*.

⁵⁹Во 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

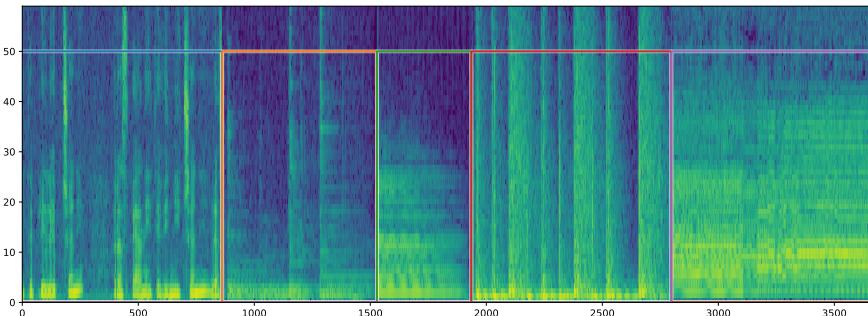
    # исто за тест множеството
    __, __, spectrogram = da.get_spectrogram(
        fs, x_test_sig, n_win=256, plot=False)
    x_test_feats = spectrogram.T
    mask_spectrogram = np.all(x_test_feats < -80, axis=1)
    x_test_feats = x_test_feats[~mask_spectrogram, :60]
    n_samples = x_test_feats.shape[0]
    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, бидејќи ни е потребен само спектограмот, а не и неговата временска и фреквенциска оска, кои на излез ги дава функцијата `da.get_spectrogram()`, нив ги доделивме во променлива која нема да ја користиме со име `__`, што вообичаено се прави во вакви ситуации. Од спектограмот ги задржавме само најниските 60 бинови кои одговараат на фреквенции до 10,3 kHz и правиме негово транспонирање за примероците да бидат редици, а обележјата колони во матриците на влезни податоци.

За визуелизација на спектограмите за различните класи и потврда дека тие можат да се искористат за нивно разликување, ќе ги прикажеме на графикон преклопени со типот на инструментот. Спектограмите на петте класи од множеството за тестирање се прикажани на Сл. 9.12. Може



Сл. 9.12: Обележја екстрактирани од множеството за тестирање од петте класи: глас, гитара, виолина, тапан, и зурла.

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 со 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)
```

Тренирање на моделот

За оваа анализа ќе ја искористиме имплементацијата на невронска мрежа за класификација `MLPClassifier` во модулот `neural_network` на пакетот за машинско учење [Сајкит-лерн](#).⁶⁰

Пред тренирањето на моделот ќе 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 на ниво на купче во алгоритмот SGD.

За таа цел, ќе го искористиме `random` модулот на NumPy кој соодржи [генератор на псевдо-случајни броеви \(PRNG\)](#)⁶¹. Тоа е конечен автомат кој генерира секвенца на броеви чија статистика е налик на

⁶⁰sciKit-Learn <http://scikit-learn.org/stable/>

⁶¹Англ. *pseudo-random number generator*.

таа на случајна низа на броеви, иако начинот на кој се генерираат е детерминистички. Така, со поставување на состојбата на автоматот со наредбата `np.random.seed`, при секое извршување на кодот ќе ги добијеме истите резултати. Во нашиот случај, тоа ќе биде истото мешање на податочното множество без разлика колку пати ќе го извршиме кодот.⁶²

```
# 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]
```

При тренирањето на невронските мрежи уште подобро е податоците да се измешаат на почетокот од секоја епоха. Тоа може да се овозможи во `MLPClassifier` со аргументот `shuffle`. Тој заедно со низата други аргументи 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
    learning_rate_init=0.01, # чекор на учење
    alpha=1e-4, # коефициент на L2 регуларизација
    solver='adam', # алгоритам на промена на чекорот на учење
    shuffle=True, # мешање на податоците одново во секоја епоха
    random_state=42, # иницијализација на PRNG
    max_iter=3000, # максимален број на итерации
    tol=1e-9, # толеранција на промена на грешката за рано запирање
    n_iter_no_change=40, # запирање на тренирањето заради заситување
    early_stopping=False, # рано запирање со множество за валидација
    verbose=1, # подетален 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
```

⁶²Ова 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 препродуцибилност на резултатите.

```

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 ја искористиме функцијата `mlp.predict_proba` 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 хипер-параметрите на невронската мрежа.

За дополнително да воочиме каде најмногу греши невронската мрежа, може да ја пресметаме [матрицата на забуна](#).⁶³ За да ја пресметаме, ќе направиме предвидување на класите во множеството за тестирање.

```

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)

```

```
[[812  26   2  22   0]
 [ 78 514   3  71   1]]
```

⁶³Англ. *confusion matrix*.

```
[ 19   4 384   0   0]
[ 19   72   0 769   2]
[  2    2  13   0 845]]
```

Во матрицата на забуна, 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 за предвидена класа. Исто така, за појасен приказ ќе ја прикажеме матрицата на забуна преку [топлинска мапа](#)⁶⁴ прикажана на Сл. 9.13.

```
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)

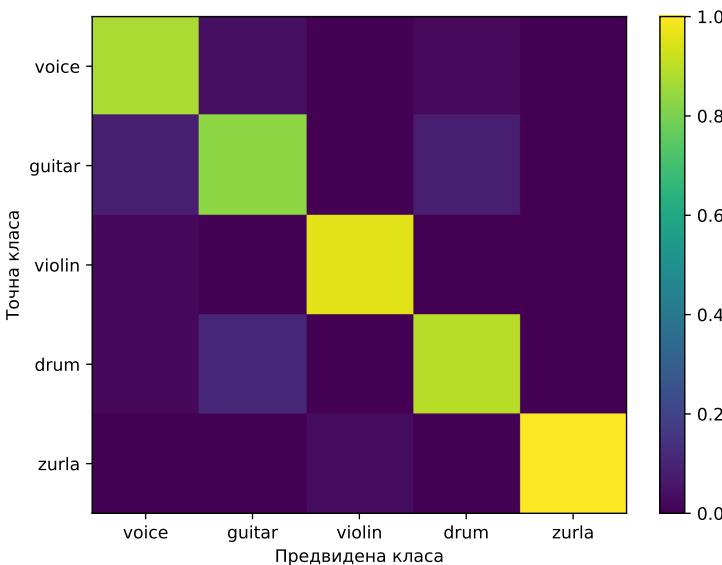
[[0.87311828 0.0420712 0.00497512 0.02552204 0.          ]
 [0.08387097 0.83171521 0.00746269 0.08236659 0.00117925]
 [0.02043011 0.00647249 0.95522388 0.          0.          ]
 [0.02043011 0.11650485 0.          0.89211137 0.00235849]
 [0.00215054 0.00323625 0.03233831 0.          0.99646226]]
```

Може да се види дека невронската мрежа прави најмногу грешки 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. 9.12.

✓ **Задача за дома.** Оптимизирај ја архитектурата на невронската мрежа. Која е најдобрата точност што може да се постигне? Која е минималната архитектура на мрежата која постигнува точност од над 50%? Дали додавањето на повеќе слоеви има поголем ефект од зголемувањето на бројот на неврони во скриениот слој на плитка невронска мрежа, притоа задржувајќи го бројот на параметри на мрежата еднаков?

9.3 Автоматско препознавање на говор

⁶⁴Англ. *heat map*.



Сл. 9.13: Матрица на забуна за истренираната невронска мрежа.

Автоматското препознавање на говор (ASR)⁶⁵ претставува клучна алка во остварувањето на сонот за природна и непречена комуникација помеѓу човекот и машината, без потреба од помагала и интерфејси, но и помеѓу луѓето кои зборуваат различни јазици во рамките на универзалните преведувачи. Навистина, најприродниот начин на којшто луѓето 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 сè уште треба да бидат надминати.

Напредокот во развојот на системите за автоматско препознавање на говор доведе до нивна сè поголема пенетрација во модерното живеење на човекот. Оваа своевидна револуција се должи на имплементирањето на системите за ASR во паметните 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 побарате од својот мобилен телефон информации за времето, или пак да

⁶⁵Англ. *Automatic Speech Recognition*.

го искористите за пребарување на интернет, сè со помош на вашиот глас.

Освен во паметни 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 попречност, кои можат да ја користат за комуникација со дигиталните уреди и услуги.

Архитектурата на еден генеричен систем за ASR е представена на Сл. 9.14. Таа е составена од влезен и излезен модул. 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 текстуален запис.

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 предвидените букви, треба да ја генерира веројатноста на низа зборови што одговара на нив. Проценетите веројатности од двата модела служат за пресметка на тоа која секвенција на зборови има најголема веројатност за влезната секвенција од обележја.

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.

Ова се реализирало со помош на банка на филтри со нелинеарна распределба на централните фреквенции и на фреквенцискиот опсег на секој од филтрите, според којашто 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



Сл. 9.14: Архитектура на еден систем за автоматско препознавање на говор.

моделирала преку употреба на логаритамска компресија на амплитудата на сигналот.

Неприкосновени во полето на ASR, од нивната прва примена во 1980, па сè до денес, се обележјата наречени **мел фреквенциски кепстрални кофициенти (MFCC)**⁶⁶. Тие се пресметуваат со помош на банка на триаголни филтри кои се линеарно распределени долж **мел фреквенциската скала**.⁶⁷ На Сл. 9.15 е прикажана мел банка составена од 10 филтри дизајнирана за покривање на опсег од 0 – 8 kHz. Во пракса, за ASR се користат банки составени од 20 до 40 филтри, додека во системите за TTS кои работат со мел скалирани спектограми се користат банки составени и од 80, па дури и од 128 филтри.

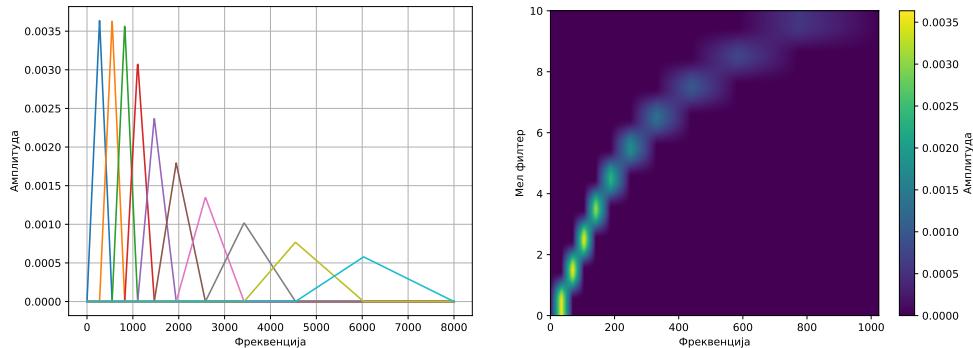
Освен мел банката на филтри, MFCC обележјата се потпираат на методата на **кестрална анализа**⁶⁸ со чија помош може да се направи декомпозиција на спектарот на говорните сигнали на дел што се должи на преносната функција на вокалниот тракт и дел што одговара на неговата побуда.⁶⁹

⁶⁶Англ. *Mel Frequency Cepstral Coefficients*.

⁶⁷Името мел доаѓа од зборот „мелодија“.

⁶⁸Зборот „кестрар“ се добива со превртување на зборот „спектар“ и го отсликува процесот на негово пресметување.

⁶⁹Ова е слично на употребата на линеарната предикција за пресметување на параметрите на филтерот извор-филтер.



Сл. 9.15: Амплитудни карактеристики на филтрите на мел банка составена од 10 филтри, прикажани на фреквенциската оска.

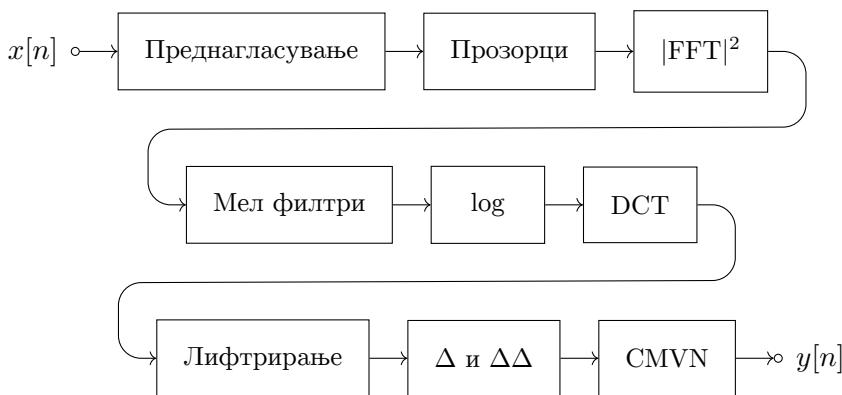
За одвојување на анвелопата на спектарот, кепстралната анализа ја пресметува инверзната Фуриеова трансформација на логаритмот од спектарот на сигналот. Логаритмирањето го претвора производот на фреквенциските карактеристики на побудата и на вокалниот тракт, кој соодветствува на конволуцијата на сигналот на побудата со филтерот на вокалниот тракт во временски домен, во збир на овие две компоненти. Бидејќи побудата има повисока фреквенциска содржина од анвелопата, по инверзната Фуриеова трансформација, овие две компоненти можат лесно да се издвојат во кепстрален домен.

На Сл. 9.16 е прикажана структурата на алгоритмот за пресметка на MFCC. Тој се состои од следните чекори:

- претпроцесирање на влезниот говорен сигнал, најчесто со преднаглавување⁷⁰ 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 банка на филтри,
- логаритмирање на моќноста на спектарот, и
- примена на дискретна косинусна трансформација (DCT)⁷¹ која е по-фиксна, а за нашите сигнали еквивалентна на инверзната Фуриеова

⁷⁰Англ. *preemphasis*.

⁷¹Англ. *Discrete Cosine Transform*.



Сл. 9.16: Блок-дијаграм на алгоритмот за пресметка на MFCC.

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 **лифтрирани** за да се засилат амплитудите на коефициентите со повисок ред⁷², како и да се нормализираат средната вредност на 0 и варијансата на 1 (**CMVN**)⁷³.

Во системите за ASR, MFCC коефициентите вообичаено се надополнуваат со приближна пресметка на нивниот прв и втор извод за да се земе предвид нивната временска динамика. Притоа, основните коефициенти се нарекуваат **статички MFCC**, а нивниот прв и втор извод се нарекуваат **динамички MFCC**, или уште **Δ-MFCC** и **ΔΔ-MFCC**. Со тоа, вкупниот број на обележја, односно димензионалноста на влезниот вектор, за секоја рамка од говорниот аудиосигнал е 39.

Акустички модели

Обележјата 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:

1. еквиваленција на акустичкиот облик на даден сигнал и неговата фонетска содржина,
2. споредба со множество урнеци,
3. статистичко моделирање на акустичката реализација на гласовите, и

⁷²Филтрирањето во кепстрален домен се нарекува лифтрирање.

⁷³Англ. *Cepstral Mean and Variance Normalization*.

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 урнеци (темплејти). За компензирање на варијацијата во временската динамика при изговарање на говорните секвенции, бил развиен алгоритмот базиран на динамичко програмирање познат како **динамичко временско преобличување (DTW)**⁷⁴.

Најголемата револуција во начинот на работа на акустичките модели е добиена со преминот кон ригорозни математички рамки за статистичко моделирање, базирани на **скриените модели на Марков (HMM)**⁷⁵. Тие овозможуваат моделирање на секвенцијалната структура на говорот преку ланци на состојби и статистички опис на варијабилноста на акустичките (спектрални) обележја преку дистрибуции на веројатност моделирани со **Гаусови мешавински модели (GMM)**⁷⁶.

Во поново време, како надградба на HMM, се воведени **тежинските трансдусери со конечен број на состојби (WFST)**⁷⁷ 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 сè уште биле базирани на HMM. Денес, 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

⁷⁴Англ. *Dynamic Time Warping*.

⁷⁵Англ. *Hidden Markov Models*.

⁷⁶Англ. *Gaussian Mixture Models*.

⁷⁷Англ. *Weighted Finite State Transducer*.

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 натчовечки перформанси во препознавањето на говор. Врвните системи за ASR денес постигнуваат рата на **грешка на ниво на збор (WER)**⁷⁸ 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.

Јазични модели

Јазичните модели (LM)⁷⁹ се користат за моделирање на веројатноста на појава на одредена секвенција на зборови во јазичниот корпус. Тие речиси секогаш се интегрираат во системите за ASR за подобрување на нивната прецизност.

Овие модели традиционално се базирани на статистички методи за моделирање на јазичната структура во вид на **N -грами** (Jurafsky and Martin, 2024). N -граммите се секвенции од N зборови кои се појавуваат во јазичниот корпус. Па така разликуваме јазични модели кои ја моделираат веројатноста на појавување на 1 збор – **униграми**, или веројатноста на моделирање на секвенции од 2 – **биграми**, 3 – **триграми** или 4 зборови – **четириграми**.

Модерните јазични модели се базирани на рекурентни невронски мрежи (RNN), односно во поново време на **трансформери**. Трансформерите овозможуваат моделирање на долги зависности помеѓу зборовите во јазичниот корпус. На тој начин, имаме конвергенција на архитектурата на акустичките модели за ASR, јазичните модели, како и моделите за TTS.

Сепак, поради својата брзина N -граммите сè уште наоѓаат примена во модерните системи за ASR, а посебно во нивниот развој и истражувањата. Еден таков модел, кој денес е во широка употреба е KenLM.⁸⁰

Препознавање на говор на македонскиот јазик

Системи за ASR се развиени за повеќето од светските јазици. Денес преку нивната интеграција во мобилните уреди, тие доживуваат голема експанзија и сè 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

⁷⁸Англ. *Word Error Rate*.

⁷⁹Англ. *Language Models*.

⁸⁰<https://kheafield.com/code/kenlm/>

и невронски мрежи. Овој систем е направен за препознавање на десетте цифри од еден говорник и има точност од 85%.

Со понатамошните подобрувања, во 2002 Краљевски и др. реализираат систем независен од говорник, базиран на употребата на урнеци и 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 кои се дел од 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 предлагаат 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*)⁸¹ на македонски јазик. Системот направен за автоматска транскрипција на медиуми на македонски, постигнува WER од 7,73% на множество за тестирање, дури и во присуство на шум. Резултатите се уште повеќе охрабрувачки ако се воочат местата на кои системот прави грешки.⁸² Во 2023, систем за ASR на македонски јазик развива спин-оф компанијата на ФЕИТ, Speech AI⁸³, која го нуди како комерцијален производ.

Имплементација на ASR систем

Во овој дел, ќе реализираме еден едноставен систем за препознавање на говор со мало множество наредби наменет за контрола на мобилен робот. Базата на говорен материјал со којашто ќе работиме е дел од базата снимена од натпреварувачите на меѓународниот студентски натпревар RoboMac 2023 во категоријата ChatBot, каде што задачата беше да се реализира систем за гласовна контрола на мобилен робот.⁸⁴

Делот што го имаме на располагање е снимен од една говорничка и содржи по 100 изговори на секоја од 5-те наредби: десно, лево, назад, оди и стоп. Базата содржи една датотека со метаподатоци `metadata.csv` во

⁸¹Англ. *Large Vocabulary Continuous Speech Recognition*.

⁸²<https://speech.feit.ukim.edu.mk/asr.html>

⁸³<https://speech-ai.mk/>

⁸⁴<http://robomac.mk/>

⁸⁵Форматот CSV (Comma Separated Values) е текстуален формат за запишување на табеларни податоци каде што 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 што секоја колона е разделена со табулатор.

која се дадени патеките на снимките и наредбите изговорени во секоја од нив:

```
$ 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
```

Да напишеме код во Python што 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 мапирање (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 намалиме 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.

```
# %% 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)

# %% воедначување 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.

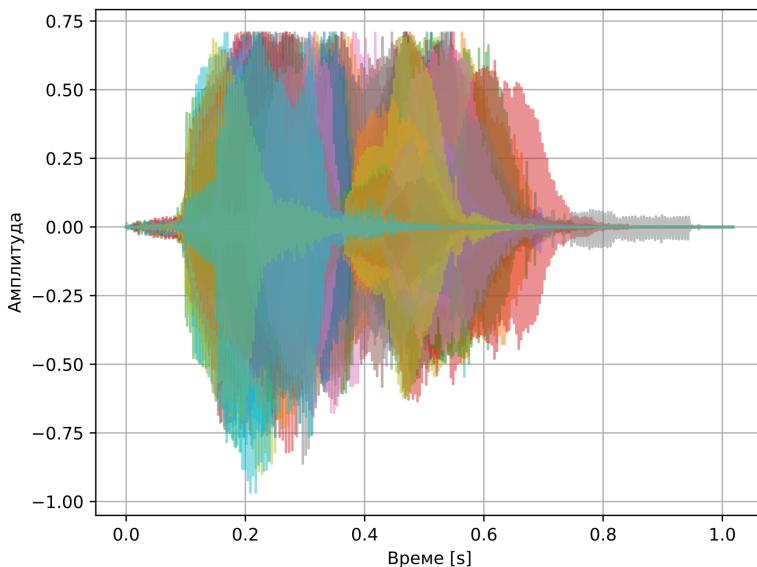
```
# %% 1. визуелизација на податоците
fig = plt.figure()
t = np.arange(wavs.shape[1]) / fs
plt.plot(t, wavs[:50].T, alpha=0.5)
plt.grid()
```

На Сл. 9.17 може да ги видиме првите 50 аудиосигнали од базата со 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]
```



Сл. 9.17: Првите 50 аудиосигнали од базата со нормализација и временско воедначување.

```
y_labels['test'] = labels[-test_ind:]
```

Следно, со помош на `librosa` ќе ги пресметаме 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, :, :] # додади димензија за примероци
        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([label1onehot[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],
)

# испртаж го временскиот облик и спектограмот
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)
```

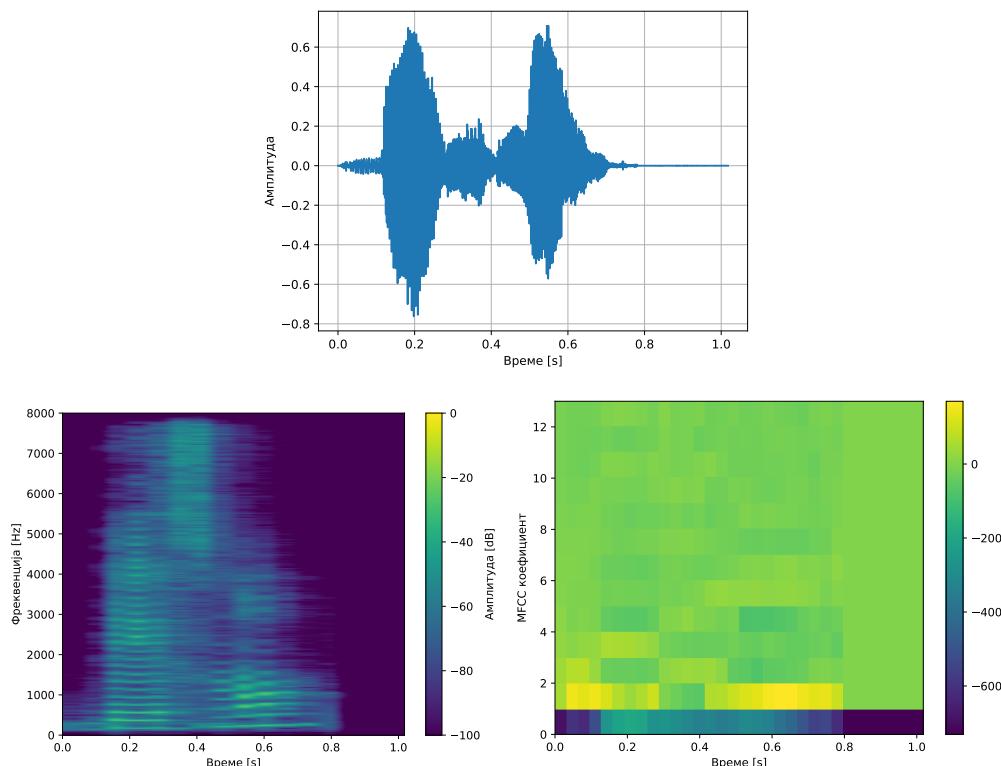
На Сл. 9.18 може да ги видиме временскиот облик, спектограмот и пресметаните MFCC коефициенти за еден од примероците од нашата база на наредбата „десно“. Може да видиме дека за разлика од спектограмот во кој можеме да ги воочиме составните гласови, MFCC коефициентите не нудат интуитивна интерпретација. 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 лифтрирање, 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 израмниме⁸⁶ во 1D вектори.

Со тоа нивниот облик ќе се промени од [примероци, време, коефициенти] во [примероци, време · коефициенти] каде што коефициентите ќе бидат надворзани 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 CNN 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

⁸⁶Англ. *flatten*.



Сл. 9.18: Временски облик (горе), спектрограм (долу лево) и MFCC коефициенти (долу десно) за примерок од базата со наредбата „десно“.

пovрзаните слоеви за класификација на излез од мрежата.

```
# %% 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,
    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

```

Може да видиме дека загубата на моделот во текот на првите 500 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 испртаме **кривата на загуба**⁸⁷ или **крива на грешка** прикажана на Сл. 9.19.

```

# испртажуј ја кривата на загуба
loss_curve = np.array(mlp.loss_curve_)
plt.figure()
plt.plot(loss_curve)
plt.grid()

```

Може да видиме дека кривата на загуба е монотоно опаѓачка и дека моделот веќе околу 50-та епоха конвергира кон крајната вредност на загубата.

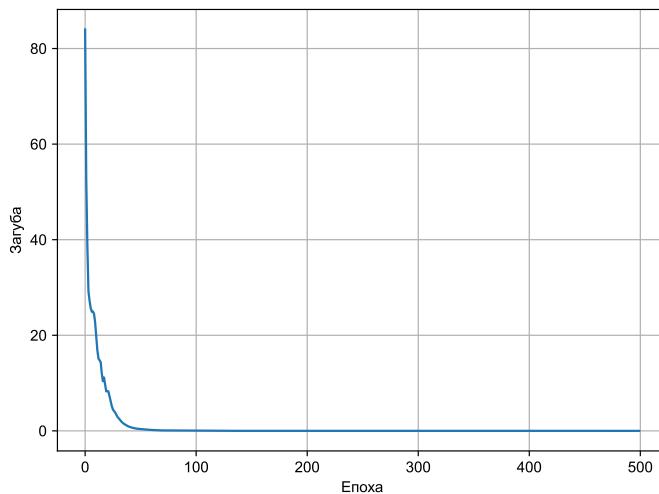
На крај, останува да го евалуираме моделот. Ајде да ја пресметаме точноста и да ја прикажеме матрицата на забуна за моделот. За испртување на матрицата на забуна, овојпат ќе го искористиме пакетот `seaborn`, кој нуди напредни функции за визуелизација на податоци.

```

# %% 5. евалуација на моделот
# точност

```

⁸⁷Англ. *loss curve*.



Сл. 9.19: Крива на загуба при тренирање на моделот.

```
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 забуна е прикажана на Сл. 9.20. Моделот одлично работи и прави само 5 грешки од 250 примероци што претставува точност од 98%!

§ Дополнително. Но, зошто точноста од 88,8% која ја пресметавме на моделот не соодветствува на точноста прикажана во матрицата на забуна од 98%?

		Предвидена класа				
		Десно	Лево	Назад	Оди	Стоп
Точна класа	Десно	47	1	0	0	2
	Лево	0	50	0	0	0
	Назад	0	0	49	1	0
	Оди	0	0	0	49	1
	Стоп	0	0	0	0	50
		Десно	Лево	Назад	Оди	Стоп

Сл. 9.20: Матрица на забуна за моделот.

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, 0, 1, 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, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 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, 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, 1, 1, 2, 1, 2, 2, 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, 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, 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, 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,
       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])
```

```
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,
       1.00000000e+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 од множеството за тестирање да издвоите еден дел за валидација што 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 софтвер за инженерство

Еден од најпрочуените софтверски пакети за нумериичка обработка е програмскиот пакет **MATLAB**¹. 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 издаден во 1984 г., а во 2004 г. имал еден милион корисници инженери, научници и економисти.

Сепак MATLAB, како комерцијален софтвер носи и низа недостатоци, пред сè високата цена, која го става надвор од 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 софтвер**². Ова, пред сè, се должи на философијата на движењето за слободен софтвер започнато од Ричард

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

Сталман³ во 1983 г. со создавањето на оперативниот систем GNU⁴, а подоцна со воспоставување на Фондацијата за слободен софтвер⁵ во 1985 г., како и поширокото движење за отвореност⁶, а тоа е заедништво во создавањето и напредувањето на технологијата и човештвото.

Четири слободи

Слободниот софтвер е дефиниран со четирите слободи:⁷

- **Слобода 1.** Слобода да ја користите програмата за која било намена.

Слободниот софтвер нема рестрикции за употреба, како што се временските рестрикции („Пробен период од 30 дена“, „Лиценцата истекува на ...“), рестрикции на целта („Дозволена е употреба за истражувачки и некомерцијални цели“) или рестрикции на географската област („Мора да се користи во земјата ...“), што ја прават програмата неслободна.

- **Слобода 2.** Слобода да проучите како работи софтверот и како истиот да го адаптирате на сопствените потреби.

Добавањето легални или практични рестрикции на разбирањето или на менувањето на софтверот, како што се специјални лиценци, спогодби за неоткривање⁸ или правењето изворниот код да биде недостапен, исто така го прават софтверот неслободен. Без слободата да се менува софтверот, луѓето остануваат на милост на снабдувачот.

- **Слобода 3.** Слобода да редистрибуиране копии за да му помогнете на вашиот сосед.

Софтверот може да се копира/дистрибуира скоро без никакви трошоци. Ако не смеете да му дадете некој софтвер на некој човек кому тој му треба, тоа го прави неслободен. Се разбира, доколку сакате, за ваквите активности можете да наплатите.

- **Слобода 4.** Слобода да го подобрите софтверот и да ги објавите

³Wikipedia – Richard Stallman https://en.wikipedia.org/wiki/Richard_Stallman
Richard Stallman – Free software, free society, TEDxGeneva 2014 <https://www.youtube.com/watch?v=Ag1AK1L2GM>

⁴Англ. *GNU's Not Unix*.

⁵Wikipedia – Free Software Foundation https://en.wikipedia.org/wiki/Free_Software_Foundation

⁶Wikipedia – 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

⁷Адаптирано од Слободен софтвер Македонија <https://slobodensoftver.org.mk/shto>

⁸Англ. *Non-Disclosure-Agreement*.

вашите подобрувања во јавноста, од што ќе има корист 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.

Предности на слободниот софтвер

Од практичен аспект, слободниот софтвер има низа предности над затворениот софтвер и тоа:

- **достапноста** – поради основната премиса на слободно споделување на изворниот код, со цел да се овозможи негов развој од страна на заедницата, слободниот софтвер е *de facto* и бесплатен софтвер. Така, повеќето производители на слободниот софтвер живеат од донации, но и од продавање поддршка за нивниот производ.
- **безбедноста** – поради достапноста на изворниот код, не постои начин производителот на софтерот да прави нешто скриено од вас, а секој спорен дел од кодот е подложен на промена од заедницата. Кaj затворениот софтер тоа не е случај.^{9,10}
- **слободата од производителот** – како корисници на слободниот софтвер, вие не сте затворени во екосистемот на производителот.¹¹ Истиот тој софтвер може да биде преземен од друга заедница на програмери и да продолжи неговото одржување и развој.

⁹Bo Windows 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 ...”

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.eteknix.com/stop-windows-7-8-spying/>

¹⁰Епл 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/>

¹¹Don 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 слободен софтвер, развојот не може да се спореди со ресурсите кои ги поседува која било корпорација во светот. Така, развојот на [Linux јадрото](#)¹², кое е во основата на оперативниот систем [GNU/Linux](#), познат и само како [Linux](#)¹³ и повеќе од 600-те [GNU/Linux дистрибуции](#)¹⁴, првично напишано од [Линус Торвалдс](#)¹⁵, денес претставува најголемиот софтверски 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.¹⁶

Сите овие придобивки заедно придонесуваат за широка распространетост на слободниот софтвер денес.¹⁷ Така, оперативните системи [GNU/Linux](#) и [FreeBSD](#)¹⁸ се во употреба во 98,27% од серверите на интернет (споредено со 1,73% со [Windows](#)), 71,9% од паметните уреди (со оперативен систем [Android](#), додека останатите 28% користат [iOS](#) со јадрото [XNU](#), кое е исто така отворен софтвер базиран на [BSD](#) јадрото), и 99% од суперкомпјутерите¹⁹. Сепак, неговиот пробив во персоналните компјутери засега е мал – 3,7% (наспроти 72,9% на [Windows](#) и 14,9% на [MacOS](#), кој исто така се базира на отворен софтвер).

Одржливост

Постојат различни начини на кои се реализира финансиската поддршка на развојот и одржувањето на слободниот софтвер и покрај бесплатноста и тоа:

- финансиска поддршка од компании – зад многу пакети слободен софтвер стојат компании во чиј интерес е неговиот развој. 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 [Ubuntu](#)²⁰, [Fedora](#)²¹, и [OpenSUSE](#)²², како и пакетите за длабоко

¹²Wikipedia – Linux kernel https://en.wikipedia.org/wiki/Linux_kernel

¹³Wikipedia – Linux <https://en.wikipedia.org/wiki/Linux>

¹⁴Wikipedia – List of Linux distributions https://en.wikipedia.org/wiki/List_of_Linux_distributions

¹⁵Wikipedia – Linus Torvalds https://en.wikipedia.org/wiki/Linus_Torvalds

¹⁶Добар документарен филм за рафањето и развојот на оперативниот систем [GNU/Linux](#) е [Revolution OS - 2001](#) <https://www.youtube.com/watch?v=Eluzi700-P4>

¹⁷Wikipedia: Linux - Market share and uptake https://en.wikipedia.org/wiki/Linux#Market_share_and_uptake

¹⁸<https://www.freebsd.org/>

¹⁹Linux is Running on Almost All of the Top 500 Supercomputers <https://itsfoss.com/linux-supercomputers-2017/>

²⁰<https://www.ubuntu.com/>

²¹<https://getfedora.org/>

²²<https://www.opensuse.org/>

учење TensorFlow²³ и PyTorch²⁴, исто така развивајани 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 софтверот KiCad за електронски дизајн и изработка на печатени плочи развивајан во CERN²⁵, софтверот за дигитална обработка на звук Audacity чиј развој започнал во Универзитетот Карнеги Мелон²⁶, или пак пакетот за машинско учење Scikit-learn започнат во INRIA²⁷,
- бизнис модел базиран на поддршка – најголемата компанија која денес работи исклучиво со слободен софтвер е Red Hat чиј оперативен систем GNU/Linux е еден од најзастапените на интернет серверите.²⁸ 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 дистрибуциите како на пример Manjaro²⁹ или Mint³⁰, а исто така Spyder³¹ развојната средина за Python која 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 развиваачи што немаат финансиски придобивки од нивната работа на проектот, но сепак продолжуваат да работат на него водени од сопствените убедувања и стремеж кон повисоки цели.

A.2 Слободен софтвер за инженерство

Постојат низа слободни софтвери што можат да бидат искористени за обработка на нумерички податоци.

²³TensorFlow – An end-to-end open source machine learning platform <https://www.tensorflow.org/>

²⁴PyTorch – from research to production <https://pytorch.org/>

²⁵KiCad EDA – A Cross Platform and Open Source Electronics Design Automation Suite <https://www.kicad.org/>

²⁶Audacity – Free, open source, cross-platform audio software <https://www.audacityteam.org/>

²⁷scikit-learn – Machine Learning in Python <https://scikit-learn.org/stable/index.html>

²⁸Red Hat – The world's leading provider of open source solutions <https://www.redhat.com>

²⁹Manjaro – Professional Linux at its best <https://manjaro.org/>

³⁰Linux Mint – From freedom came elegance <https://linuxmint.com/>

³¹Spyder – The Scientific Python Development Environment <https://manjaro.org/>

- GNU Octave³² 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 најмногу се употребува во образоването како замена за MATLAB.
- Scilab³³ е слободен софтвер за нумеричка обработка наменет за инженери и научници, во употреба од 1994 година. Scilab во себе вклучува и слободна замена за пакетот Симулинк на MATLAB, наречена Xcos³⁴,
- Python³⁵ е широко распространет, повеќенаменски, интерпретиран и динамичен програмски јазик на високо ниво направен од Гуидо ван Росум³⁶ 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.
- Julia³⁷ 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. Покрај големиот потенцијал на Julia, за сега неговата примена останува ограничена во области во кои е неопходна голема процесирачка моќ, како во комплексни физички симулации.

Audacity

Audacity³⁸, прикажан на Сл. А.1, е слободен софтвер за анализа, едитирање и снимање на дигиталниот звук. Неговиот развој го започнале Доминик Мацони и Роџер Даненберг во 1999 во Универзитетот Карнеги Мелон и е иницијално објавен во 2000 година како верзија 0.8. Од 2011, тој е 11-иот најсимнуван софтвер на Сурсфорц, со 76,5 милиони симнувања. Audacity е добитник на наградата за најдобар проект за мултимедија од заедницата SourceForge во 2007 и 2009 година. Во 2015 г. е преместен на FossHub, каде што е најпопуларниот софтвер со над 114 милиони симнувања.³⁹

³²GNU Octave – Scientific Programming Language <https://www.gnu.org/software/octave/>

³³Scilab – Open source software for numerical computation <https://www.scilab.org/>

³⁴<https://www.scilab.org/software/xcos>

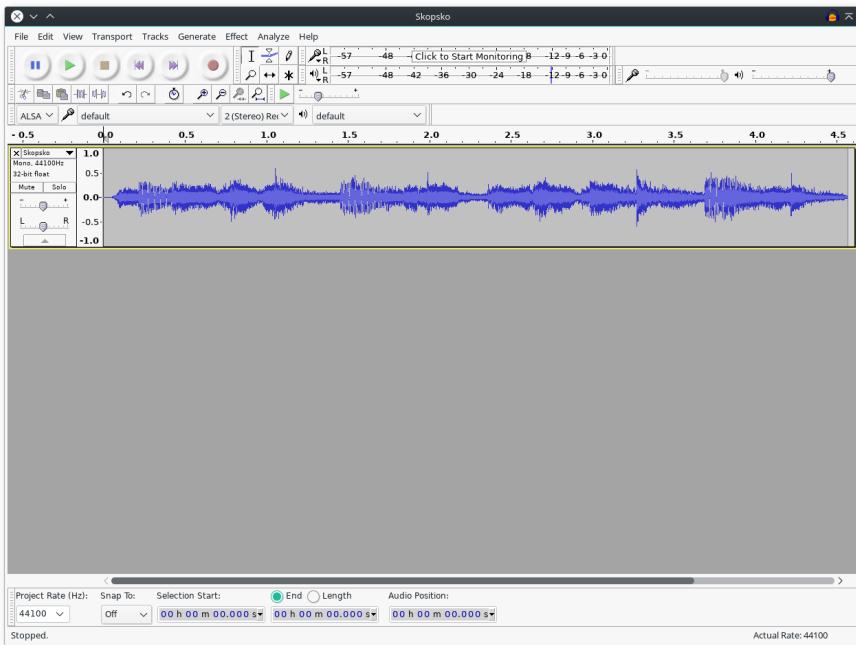
³⁵<https://www.python.org/>

³⁶Wikipedia – Guido van Rossum https://en.wikipedia.org/wiki/Guido_van_Rossum

³⁷<https://julialang.org/>

³⁸<http://www.audacityteam.org/>

³⁹Wikipedia: Audacity (audio editor). [https://en.wikipedia.org/wiki/Audacity_\(audio_editor\)](https://en.wikipedia.org/wiki/Audacity_(audio_editor))



Сл. А.1: Отворен аудиосигнал во главниот прозорец на Audacity.

Освен тоа што поддржува снимање од повеќе извори, Audacity може да се искористи за процесирање на сите типови звук, преку додавање на ефекти како нормализација, отстранување на шум, и прелевање. Тој може да се користи за снимање и миксање на цели албуми, како што е случајот со групата Tune-Yards. Во употреба е и во националниот курс за ICT, ниво 2 на OCR⁴⁰ во Велика Британија.

Главните особини на Audacity вклучуваат:

- Вчитување и снимање на различни типови на аудиоформати, 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 аудиотраки на дигитализирани снимки од касети или грамофонски плочи.
- Повеќеканално миксање.
- Голем број на аудиоэффекти 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.

⁴⁰Oxford, Cambridge and RSA Examinations

- Едитирање на амплитудната анвелопа.
- Намалување на шумот.
- Намалување на вокалите.
- Спектрална анализа со употреба на FFT.
- Поддршка за повеќеканален звук со фреквенција на земање примени 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 поставки.

Тој исто така работи на сите оперативни системи.

Додаток Б

Основи на употреба на GNU/Linux

Иако користењето на Python не е врзано со оперативниот систем GNU/Linux, во овој материјал ќе работиме во GNU/Linux.

Б.1 Инсталирање

Зошто да инсталiram 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 програмски код и вируси,
- **стабилност и перформанси** – познат е по својата стабилност и ефикасност, што е една од главните причини што го прави популарен избор за вгнездени системи, сервери и други критични системи,
- **слобода** – тој е слободен софтвер со сите предности кои идат со тоа.

Уште повеќе, користењето на GNU/Linux ќе ве направи подобри идни инженери поради неговата сеприсутност во индустриската и инженерската и научна пракса:

- **употреба во компаниии 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,

- **пристал 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 за постара верзија од пакетот, или пак да има многу ограничена поддршка, поради тоа што развиваите користат Linux,
- **контрола и приспособување** – кај Linux корисникот има целосна контрола врз оперативниот систем и хардверот на којшто е инсталiran преку едитирање на текстуални датотеки. Дополнително, Linux овозможува висок степен на прилагодување, како на изгледот на десктоп средината, така и на деталите 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 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 кај кои може да се пристапи преку командна линија, како на пример работа со интернет-сервери,
- **распространетост 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, а со дел од нив сигурно ќе се сртнете во вашата професионална кариера.

Всушност вашиот Linux лаптоп е *de facto* сервер на кој можете да се приклучите од друг компјутер, или пак на којшто можете да стартувате веб-апликација до која можат да пристапат корисници преку интернет.

Конечно, софтверот што 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 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 во обсир дека 15% од корисниците на Windows го користат потсистемот за Linux WSL, тогаш излегува дека мнозинството програмери од 55% користат Linux за развој на софтвер.

Начин на инсталације

Доколку веќе немате GNU/Linux, истиот се препорачува да го инсталirate паралелно на постоечкиот оперативен систем во *dual boot* режим. Во најмала рака може да инсталарате 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 инсталације се сведува на дузина од најпопуларните дистрибуции, затоа што популарноста е некој вид гаранција за нивната стабилност и поддршка. Добар преглед на популарноста на различните дистрибуции на Linux, како и повеќе информации за истите може да најдете на вебстраницата *Distrowatch*³. Во моментов таму во 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 инсталације и ажурирање на софтверот и оперативниот систем;
- **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)⁴ на секои две години; некои дистрибуции во периодичните надградби го ажурираат системот на најновите верзии на сите софтвери⁵, со што се заобиколува потребата од големи надградби, таков е на пример Arch и дистрибуциите базирани на него како Manjaro.

Овие карактеристики се заеднички за цели фамилии на дистрибуции, на пример Kubuntu и Ubuntu користат `apt` за инсталације на `.deb` пакети

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/>

⁴Англ. *Long Term Support*.

⁵Таканаречен модел *rolling release*.

како и дистрибуцијата на која се базирани – Debian. Дополнително, сите тие се базираат на поголеми надградби на подолг рок.

Избор на десктоп средина

Важно е да се направи разлика помеѓу дистрибуцијата на GNU/Linux и изгледот на нејзиниот кориснички интерфејс. Интерфејсот на дистрибуцијата се базира на [десктоп средината](#) којашто ја користи. Кај оперативните системи GNU/Linux постојат различни десктоп средини, како GNOME и KDE Plasma, Xfce, LXQt, Cinnamon, Mate, i3wm, Pantheon итн. Дистрибуциите вообичаено вклучуваат и одредено прилагодување на стандардниот изглед на десктоп средината. На пример, Ubuntu дадава панел со икони за стартирање и контрола на програмите од левата страна во GNOME, додека Manjaro дадава докер со икони на долнит раб од десктопот.

GNOME⁶ 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 Plasma⁷ 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 работиме со Ubuntu⁸ 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 Cinnamon⁹ и Pop!_OS со GNOME¹⁰. Тука се и дистрибуции кои се базирани на Ubuntu, а користат различни десктоп средини како на пример Kubuntu со KDE Plasma¹¹ и Xubuntu со Xfce¹². Која било од овие може да се добар прв избор за преминување на Linux.

Напуштајќи го овој вообичаен пристап, една алтернативна препорака е да инсталirate Arch базирана дистрибуција како на пример Manjaro

⁶<https://www.gnome.org/>

⁷<https://kde.org/>

⁸<https://ubuntu.com/>

⁹<https://linuxmint.com/>

¹⁰<https://system76.com/pop>

¹¹<https://kubuntu.org/>

¹²<https://xubuntu.org/>

со KDE Plasma.¹³ Arch базираните дистрибуции имаат подобар систем на репозиторија за инсталирање на пакети, наспроти системот што го користи Ubuntu. Така, во Ubuntu базираните дистрибуции, често за да инсталирате некој софтвер треба да додавате нови репозиториуми со пакети кои треба да ги најдете на интернет.

Од друга страна, во Arch базираните дистрибуции секој софтвер можете да го инсталирате со пребарување на 5-те вградени репозиторија на системот во терминалот, без воопшто да треба да отворате веб-предлестувач. Уште повеќе, во овие системи, оперативниот систем и сите инсталирани софвери 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 најмногу ви одговара.¹⁴

Б.2 Основи поставки

За работа со GNU/Linux ќе го искористиме стандардниот BASH терминал.¹⁵ Вреди да се напомене дека постојат понапредни школки како на пример Z shell или Zsh.¹⁶ Во поново време, некои GNU/Linux дистрибуции доаѓаат токму со Zsh.

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 `bash`. Тоа може да го направите преку GUI¹⁷ алатката за работа со датотеки, или преку терминалот:

```
~ $ mkdir da
```

¹³<https://manjaro.org/download/#Official>

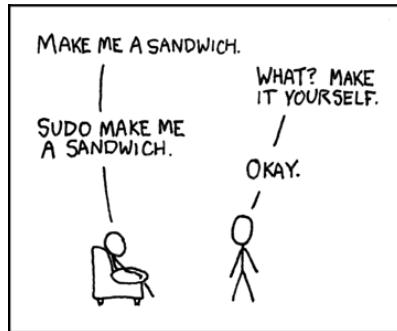
¹⁴Ова неретко се прави, посебно од страна на нови корисници на Linux, и се нарекува *distro hopping*.

¹⁵Bourne-Again Shell (BASH) школката е стандардна за сите дистрибуции на GNU/Linux и MacOS.

Добро напатство за нејзина употреба е: Software Carpentry – The Unix Shell <http://swcarpentry.github.io/shell-novice/>

¹⁶Oh-My-Zsh! A Work of CLI Magic—Tutorial for Ubuntu <https://medium.com/wearetheledger/oh-my-zsh-made-for-cli-lovers-installation-guide-3131ca5491fb>

¹⁷Англ. *Graphical User Interface*.



Сл. Б.1: Примена на sudo во секојдневни ситуации.²⁰

Следно, од Гитлаб-страницата на предметот¹⁸ преземете ја папката со звучни сегменти што ќе ги користиме. 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 SoX¹⁹, 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 аудиофајлови. 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:

```
~/da/ $ play audio/Solzi.wav
```

Solzi.wav:

```
File Size: 345k      Bit Rate: 714k  
Encoding: Signed PCM  
Channels: 1 @ 16-bit  
Samplerate: 44100Hz
```

¹⁸<https://gitlab.com/feeit-freecourseware/digital-audio-processing>

¹⁹SoX - Sound eXchange. <http://sox.sourceforge.net/>

²⁰xkcd: Sandwich <https://xkcd.com/149/>

```
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 историјата²¹,
- 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 еcranот.

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 школката source .bashrc

Структура на директориуми

- коренот на дрвото на сите директориуми во Linux е root /
- cat /etc/fstab , df -h
- корисничка папка / дома ~
- pwd

²¹ zsh нуди понапредно изминување на командите кои започнуваат на буквите кои веќе се напишани.

- 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 школката 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`

Сè 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 ²²
 - 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 поставки 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

²²Името на оваа програма доаѓа од наредбата во најстариот едитор за текст во командната линија ed која гласи g/re/p , а значи „global search for RegEx and print“.

- `echo 600 | sudo tee brightness`

Инсталирање 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 – мокен едитор/развојна средина,
- LazyVim²³ е развојна средина базирана на neovim со вклучени поставки и додатоци,
- emacs – цел оперативен систем за работа во терминал.

Работа со звук

²⁴

- cd electroacoustics/code/audio
- ls
- play viluska.wav – треба да се инсталира sox
- 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 pup²⁵
 - soxi -D *.wav | pup -b 'sum = 0' 'sum += float(x)' -a 'sum'
 - pup --explain -b 'sum = 0' 'sum += float(x)' -a 'sum'
- конверзија на датотеки во mp3

²³<http://www.lazyvim.org/>

²⁴Овој дел е преземен од скриптата за Електроакустика <https://gitlab.com/feeit-freecourseware/electroacoustics>

²⁵<https://github.com/hauntsaninja/pup>

- `sox viluska.wav viluska.mp3` или `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 ffmpeg -i $f -vn -acodec copy '$(basename $f .mp4).aac'; done`
 - или може само `yt-dlp --extract-audio`
 - `tldr yt-dlp`

Работа со процеси

- `xlogo` vs `xlogo &` – треба да се инсталира `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` и `bashtop`
 - за GPU `nvtop`
 - за запишување на диск `iotop`
- Гасење и рестартирање на системот:
 - `poweroff`
 - `shutdown`
 - `reboot`

Корисно

- `zsh`²⁶ - понапредна школка²⁷
- `tmux`²⁸ – 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`²⁹,
- календар: `cal`
- временска прогноза: `curl wttr.in/skopje`

Забава

- `asciiquarium`
- `cmatrix`
- `lolcat` - `ls | lolcat` OR `cmatrix | lolcat`
- `sl`
- `cowsay`, `fortune`, `cowfortune`, `fortune-mod-chuck`
- `cointop`
- `telnet towel.blinkenlights.nl`
- `curl -s http://artscene.textfiles.com/vt100/bambi_godzilla | pv -q -L 9600`

Корисни линкови

- Terminals, shells and SSH³⁰
- MIT Missing Semester³¹

²⁶<https://ohmyz.sh/>

²⁷<https://medium.com/wearetheledger/oh-my-zsh-made-for-cli-lovers-installation-guide-3131ca5491fb>

²⁸<https://github.com/tmux/tmux/wiki>

²⁹<https://github.com/LukeSmithxyz/mutt-wizard>

³⁰<https://j11g.com/2023/01/14/i-dont-understand-terminals-shells-and-ssh/>

³¹<https://youtu.be/Z56Jmr9Z34Q>

Додаток В

Основи на работа со Python

За процесирањето на дигиталните звучни сигнали ќе го користиме програмскиот јазик **Python** и тоа неговата 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 **Pandas**⁴ за работа со податочни множества и табели и за статистички анализи, **Sympy**⁵ за симболичка математика, **SciKit-Learn**⁶ за машинско учење итн.

Како интерфејс за Python ќе ја искористиме интерактивната конзола **IPython**⁷ и научната развојна средина за Python **Spyder**⁸.

Зашто Python?

Python е програмски јазик на високо ниво за општа намена. Во анализите, Python постојано котира како еден од најдобрите и најпопуларни програмски јазици во софтверската индустрија. Така, Python е на

¹<http://www.numpy.org/>

²<http://www.scipy.org/>

³<http://matplotlib.org/>

⁴<http://pandas.pydata.org/>

⁵<http://www.sympy.org/en/index.html>

⁶<http://scikit-learn.org/stable/>

⁷IPython Interactive Computing <http://ipython.org/>

⁸Spyder – The Scientific PYthon Development EnviRonment <https://github.com/spyder-ide/spyder>

прво место на TIOBE индексот на програмски јазици ⁹, а од 2019 г. е на прво место и на годишната ранг-листа на IEEE со максимален број на поени 100/100¹⁰.

Овој раст на популарноста на Python се должи на низа предности¹¹:

- лесен за учење и читање – Python има едноставна и јасна синтакса, што им олеснува на почетниците брзо да го научат јазикот, а на тимовите да можат полесно да одржуваат бази на код,
- 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,
- богат екосистем – Python има голема и активна заедница, што придонесува за големата и постојано растечка колекција на библиотеки и модули од најразновидни области,
- компатибилност – Python е веќе инсталриран 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 апликации кои треба да работат на различни платформи и уреди,
- динамички – Python има динамичко одредување на типот, што значи дека типовите на променливите се одредуваат при извршување, што го олеснува пишувањето и развојот на кодот.

Предности и недостатоци споредено со С и С++

- синтакса – Python има поедноставна и почитлива синтакса, што го олеснува пишувањето и одржувањето на кодот,
- перформанси – С и С++ се компајлирани јазици, што ги прави побрзи од Python, кој е интерпретиран јазик. Тие имаат и поголема контрола врз хардверот и управувањето со меморијата, што ги прави погодни за апликации во кои се критични перформансите,
- развој – развојот на код во Python е генерално побрз и полесен поради неговата поедноставна синтакса и автоматско управување со меморијата,
- 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,

⁹<https://www.tiobe.com/tiobe-index/>

¹⁰<https://spectrum.ieee.org/the-top-programming-languages-2023>

¹¹Овој дел е составен со помош на слободни големи јазични модели (LLM) достапни преку слободниот софтвер Ollama <https://ollama.com/>.

-
- **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 C++ вообичаено се користат за програмирање на системско ниво и развој на 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,¹²
- **цена** – 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 придонесува за поголем избор на библиотеки и алатки, особено во области неповрзани со нумерички пресметки.
- **интероперабилност** – Python може едноставно да се интегрира во други алатки, разновидни софтверски и хардверски платформи и сложени системи,
- **употреба** – Python е јазик за општа употреба и се користи за поширок опсег на задачи, вклучувајќи развој на веб-технологии, научни пресметки, анализа на податоци и вештачка интелигенција.

Python интерпретер

За работа со Python може да ја искористиме стандардната инсталација на Python, која доаѓа со секоја дистрибуција на GNU/Linux. Python интерпретерот, или **толкувачот**, можеме да го повикаме во 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
```

¹²Уште една предност е што Python не бара ; на крајот од секоја линија код!

За излегување од Python конзолата треба да ја притиснеме стандардната кратенка `Ctrl-d` или да напишеме `exit()`.

B.1 Виртуелни средини

Стандардната алатка за инсталирање на пакети во Python е `pip`¹³. Инсталирањето на Python пакети со `pip` директно во оперативниот систем не е препорачливо поради можноста за настанување на нарушување на меѓувисностите на 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 ниво на 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 верзија на Python. 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  
~/da/venv/bin/pip  
(da) ~/da $ deactivate  
~/da $ which pip  
/bin/pip
```

Гледаме дека сме во виртуелната средина според `(da)` пред BASH промптот. Активацијата на виртуелната средина функционира така што патеката до средината се додава на прво место на листата од системски

¹³Python Install Package <https://pypi.org/project/pip/>

патеки \$PATH на кои системот ги бара командите што ги пишуваме во терминалот. Па, наместо системскиот pip, кога ќе ја активираме виртуелната средина школката го наоѓа pip во нашата виртуелна средина.

venv е наједноставниот, но и најограниченот начин за создавање на виртуелни средини во Python. Главен 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

Pipenv¹⁴ е понапреден пакет за менацирање на виртуелни средини во Python. Тој е препорачан од Телото за пакување на Python¹⁵.

Pipenv најпрвин треба да го инсталлираме користејќи го системскиот pip:

```
$ sudo pip install pipenv
```

За создавање на виртуелна средина во нашата папка 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...
creator CPython3Posix(dest=~/.local/share/virtualenvs/da-P-eb7icK, clear=...
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=...
added seed packages: pip==23.0.1, setuptools==67.4.0, wheel==0.38.4
activators BashActivator,CShellActivator,FishActivator,NushellActivator,....

✖ 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 (fedbd2ab7af84c ... )!
Installing dependencies from Pipfile.lock (e4eeef2)...
To activate this project's virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.
```

Со ова pipenv создава виртуелна средина во локална папка во

¹⁴Pipenv – Python Development Workflow for Humans <https://github.com/pypa/pipenv>

¹⁵Python Packaging Authority PyPA <https://packaging.python.org/en/latest/tutorials/managing-dependencies/>

домашната патека на нашиот корисник, во која го копира системскиот Python и pip . Активирањето на виртуелната средина се прави со истата наредба `pipenv shell` , која ја става патеката на виртуелната средина како прва 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, како и претходно, 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 инсталлира со Miniconda¹⁶ минималниот пакет од Anaconda¹⁷ дистрибуцијата на Python пакети за работа со нумерички податоци. `conda` е слободен софтвер зад кого стои компанијата Anaconda Inc., која нуди и други платени сервиси.

Во поново време, развиен е слободен софтвер базиран на `conda` кој се вика `mamba`¹⁸ и нуди поголема брзина на работа и поробусно менаџирање на пакетите во виртуелните средини. Тој може да се инсталлира преку Mambaforge¹⁹.

Разликата помеѓу `conda` и `mamba` со претходните пристапи е тоа што виртуелните средини што ги создаваат не се врзани со папката во која се наоѓаме при нивното креирање, туку тие се инсталираат на ниво на

¹⁶<https://docs.conda.io/en/latest/miniconda.html>

¹⁷<https://www.anaconda.com/>

¹⁸<https://mamba.readthedocs.io>

¹⁹<https://github.com/conda-forge/miniforge#mambaforge>

корисник. Ова ја заобиколува потребата од создавање на нова виртуелна средина за секој нов Python проект кој го работите, односно овозможува да користите една работна средина за произволен број на проекти. Со тоа се олеснува работата на проекти за кои ни требаат горе-долу истите пакети.

Да ја инсталлираме Mambaforge и да создаваме нова виртуелна средина:

```
$ mamba create -n da python
$ mamba activate da
(da) $ which pip
~/mambaforge/envs/da/bin/pip
(da) $ mamba deactivate
```

B.2 Развојни средини

IPython

Поради ограничениот можности на основниот Python интерпретер, вообичаено со Python се работи во интерактивната конзола **IPython** која нуди низа на подобрувања. Неа може да ја инсталлираме 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:

- бојење на кодот според синтаксата на Python,
- пристап до стандардната помош во Python, како на пример описи на модули и функции 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 испрту-

вање, или `%history` за испишување, пребарување или запишување на историјата на извршените наредби; повеќе за овие и други 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,
- автоматско комплетирање на атрибути и методи на објектите со употреба на `. и Tab`,
- 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**²⁰, 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:

```
$ jupyter qtconsole
```

Следниот чекор во развојот на Jupyter проектот и она што му донесе најголем пробив во заедницата се **Jupyter тетратките** што овозможуваат **описно програмирање**²¹ – парадигма воведена во 1984 од Доналд Кнут, во која компјутерскиот код е проследен со описни делови што можат да содржат слики и математички равенки.

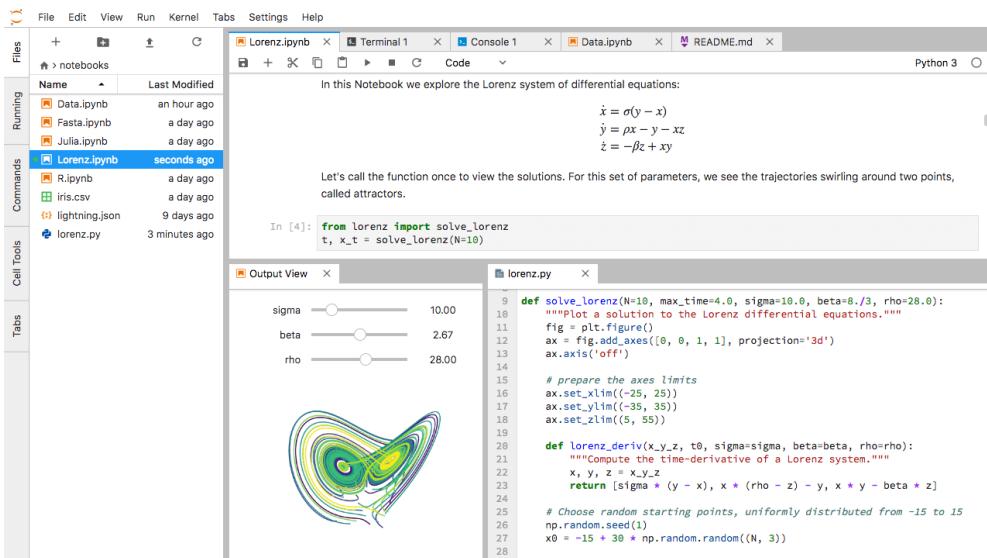
Python тетратките се користат често во научните истражувања како медиум за транспарентно споделување на кодот, што е во основата на истражувањето што може да се репродуцира, а со тоа и потврди. Python тетратките се често медиум и за споделување на туторијали или напатствија за користење на одредени пакети.

Во поново време, Jupyter тетратките се интегрираат во **Jupyter Lab**²²

²⁰<https://github.com/jupyter/qtconsole>

²¹Wikipedia – Literate programming https://en.wikipedia.org/wiki/Literate_programming

²²<https://jupyter.org/>



Сл. B.1: Jupyter Lab развојна средина за работа со IPython тетратки направена како веб-апликација.

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-предстуваач. Jupyter Lab е прикажан на сл. B.1.

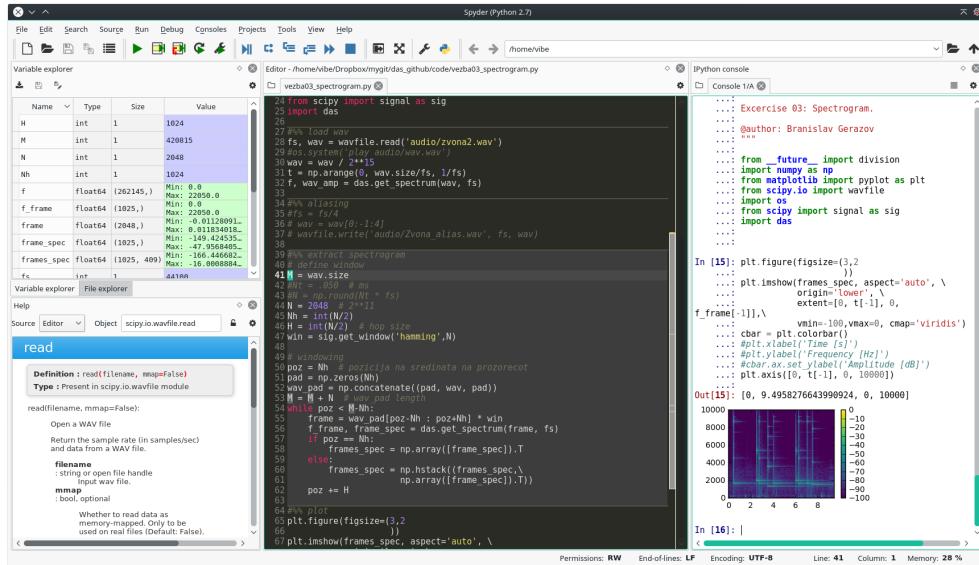
Spyder

Во овој материјал ќе работиме со развојната средина за Python специјализирана за инженерска и научна работа Spyder²³ прикажана на сл. B.2. Spyder во себе вклучува:

- **едитор** – со вклучен предстуваач на функции/класи, јазичен сервер што овозможува анализа на код, автоматско комплетирање, и поврзување на модули и функции, и поддршка за дебагирање,
- **интерактивна конзола** – интегрирана IPython конзола со можност за евалуација на кодот во едиторот на ниво на скрипта, клетка, селекција и линија,
- **документација** – прикажување на документацијата на која било класа или функција повикана во едиторот или конзолата.
- **приказ на променливи** – овозможува приказ и брза анализа на генерираните променливи,
- **предстуваач на датотеки**, и

²³Spyder – The Scientific PYthon Development EnviRonment. <https://www.spyder-ide.org/>

Додаток В. Основи на работа со Python



Сл. В.2: Развојна средина за Python Spyder специјализирана за инженерска и научна работа.

- историја на наредби.

Spyder можеме да го инсталлираме во виртуелната средина што ја создадовме во работната папка `da`:

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

Neovim

Neovim²⁴ е универзален едитор за код и текст во терминал и е модерна верзија на Vim – една од 5-те најпопуларни развојни средини²⁵. Со Neovim може да се пишува код не само во Python, туку и во сите други програмски јазици, како и да се пишуваат текстови, Markdown и LATEX документи, да се едитираат Jupyter тетратки и HTML, да се организираат белешки, настани и задачи со OrgMode²⁶, но и директно да се едитираат датотеки и директориуми.

Neovim има низа предности како што се:

²⁴<https://neovim.io/>

²⁵<https://survey.stackoverflow.co/2022/#section-most-popular-technologies-integrated-development-environment>

²⁶Org Mode – Your life in plain text <https://orgmode.org/>

nvim-orgmode <https://github.com/nvim-orgmode/orgmode>

By Default - Simple, Non-Commercial, Open Source Notes <https://www.youtube.com/watch?v=XRpHIA-2XCE>

- **моќен** – модерен едитор што нуди напредни можности за едитирање на код и текст, како што е интегрираниот **LSP**²⁷,
- **брз** – со време на стартување од 20 ms, висока респонсивност, и мала потрошувачка на RAM, Neovim овозможува брзо пребарување и замена на текстот, како и брзо пребарување на датотеки и директориуми.

Дополнително, Neovim претставува модален едитор, па секое копче од тастатурата претставува и команда за навигација низ текстот односно негово едитирање.

Заедно со скриптирањето, ова го прави Neovim најефикасниот и најбрз едитор за код и текст.

- **проширлив** – овозможува интеграција на додатоци²⁸ за различни програмски јазици и типови документи како и дополнителни можности. Имплементација на додатоците се прави со помош на едноставниот јазик за скриптирање **lua**²⁹,
- **конфигурабилен** – може бескрајно да се конфигурира со дефинирање на сопствени кратенки, команди и функции, како и со употреба и имплементација на додатоци,³⁰,
- **ергономичен** – им овозможува на корисниците да ги извршуваат сите задачи без потреба да користат глушец, а ниту стрелките на тастатурата, со што е одличен за слепо типкање бидејќи прстите воопшто не треба да го напуштат главниот ред на тастатурата.

Ова го прави одличен во комбинација со **tmux** и со автоматското распоредување на прозорците за работа на компјутер само со употреба на тастатурата, што овозможува голема брзина на работа, а дополнително носи и здравствени придобивки,³¹

- **работи во терминал** – овозможува негово користење при пристапување на онлајн сервери, што е и најчестата примена на Vim, или на локални Docker-кonteјнери и
- **пренослив** – може да се користи на различни платформи како што се GNU/Linux, Windows, и MacOS, а поради своите мали мемориски побарувања може да се користи на какви било компјутерски уреди – персонални компјутери, сервери и вгнездени уреди,

²⁷Language Server Protocol

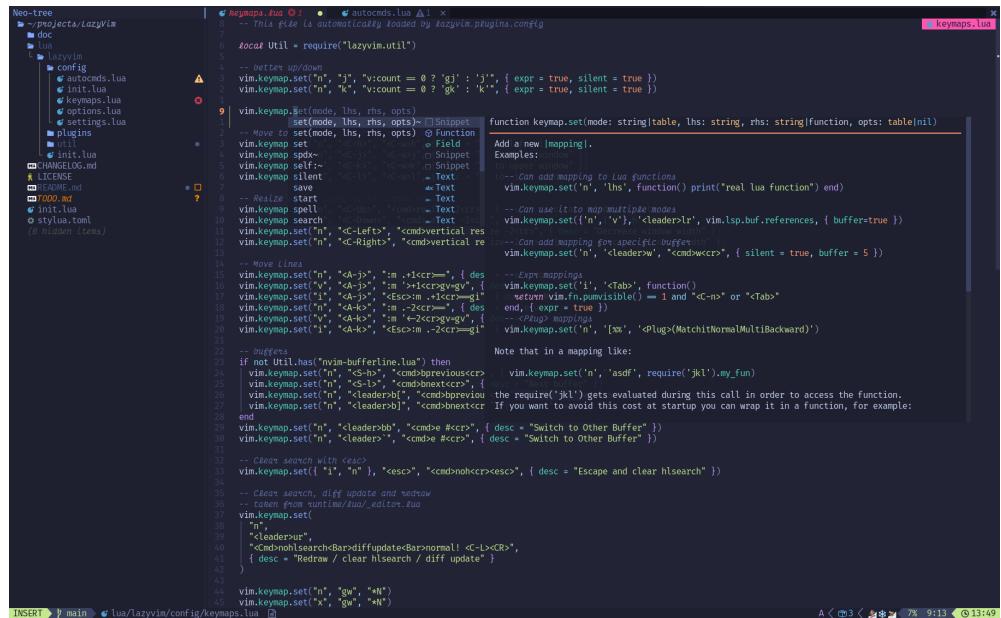
²⁸Англ. *plugins*.

²⁹Learn X in Y minutes, Where X=Lua <https://learnxinyminutes.com/docs/lua/>

³⁰TJ DeVries – Neovim as a Personal Development Environment https://www.youtube.com/watch?v=IK_-COGXfjo

³¹The Primeagen – My Developer Workflow - How I use i3, tmux, and vim <https://www.youtube.com/watch?v=bdumjiHabHQ>

Додаток В. Основи на работа со Python



```
keymaps.lua
-- This file is automatically loaded by lazyvim.plugins.config

local Util = require("lazyvim.util")

-- Better up/down
vim.keymap.set("n", "j", "vcount == 0 ? 'gj' : 'j'", { expr = true, silent = true })
vim.keymap.set("n", "k", "vcount == 0 ? 'gk' : 'k'", { expr = true, silent = true })

-- Move Lines
vim.keymap.set(mode, lhs, rhs, opts)
set(mode, lhs, rhs, opts) -> vim.keymap.set(mode, lhs, rhs, opts)
-- Mode to mode, lhs, rhs, opts -> Function
vim.keymap.set -> Field
-- Snippet
vim.keymap.spcx -> Snippet
vim.keymap.silent -> Text
vim.keymap.noremap -> Text
vim.keymap.set("n", "<C-Left>", "<CmdVerticalRes")
vim.keymap.set("n", "<C-Right>", "<CmdVerticalRe")
vim.keymap.set("n", "<C-Down>", "<CmdBufferDown>")
vim.keymap.set("n", "<C-Up>", "<CmdBufferUp>")

-- Resize start
vim.keymap.spell -> Text
vim.keymap.search -> Text
vim.keymap.set("n", "<C-Left>", "<CmdVerticalRes")
vim.keymap.set("n", "<C-Right>", "<CmdVerticalRe")
vim.keymap.set("n", "<C-Down>", "<CmdBufferDown>")
vim.keymap.set("n", "<C-Up>", "<CmdBufferUp>")

-- Buffer
if not Util.has("nvim-bufferline.lua") then
    vim.keymap.set("n", "<C-j>", "im +icrpgvyy", { desc = "Switch to Next Buffer" })
    vim.keymap.set("n", "<C-k>", "im -icrpgvyy", { desc = "Switch to Previous Buffer" })
    vim.keymap.set("n", "<C-l>", "im +cndbnext<c>", { desc = "Switch to Last Buffer" })
    vim.keymap.set("n", "<C-h>", "im -cndbprev<c>", { desc = "Switch to First Buffer" })
    vim.keymap.set("n", "<leader>bb", "<CmdBb><c>", { desc = "Switch to Other Buffer" })
    vim.keymap.set("n", "<leader>bb", "<CmdBb><c>", { desc = "Switch to Other Buffer" })
end

-- Clear search with <esc>
vim.keymap.set("i", "n", "<esc>, <cmdnoh<c>><esc>", { desc = "Escape and clear hisearch" })
vim.keymap.set("i", "n", "<esc>, <cmdnoh<c>><esc>", { desc = "Escape and clear hisearch" })

-- Clear search, diff update and redraw
-- taken from runtime/lua/_editor.lua
vim.keymap.set(
    "n", "<leader>ur", "<CmdNohSearchBar><diffupdateBar>normal! <C-L>><CR>",
    { desc = "Redraw / Clear hisearch / diff update" }
)

-- Window
vim.keymap.set("n", "gw", "N")
vim.keymap.set("n", "gw", "N")
vim.keymap.set("n", "gw", "N")
vim.keymap.set("n", "gw", "N")

function keymap.set(mode: string|table, lhs: string, rhs: string|function, opts: table|nil)
    Add a new mapping!
    Examples:
        -- Can add mapping to List Functions
        vim.keymap.set('n', 'lh', function() print("real lua function") end)
        -- Can use it to map multiple modes
        vim.keymap.set('n', 'V', {leader>lr, vim.lsp.buf.references, { buffer=true }})
        -- Can add mapping for specific buffers
        vim.keymap.set('n', '<leader>w', {silent = true, buffer = 5})
    Note that in a mapping like:
        -- Expr mappings
        vim.keymap.set('i', '<Tab>', function()
            return vim.fn.tabableable() == 1 and '<C-n>' or '<Tab>'
        end, {expr = true})
        -- <Plug> mappings
        vim.keymap.set('n', '[vn', [vn, '<Plug>(MatchitNormalMultiBackward)'])
end

-- If you want to avoid this cost at startup you can wrap it in a function, for example:
the require('jk1').getValues() gets evaluated during this call in order to access the function.
```

Сл. В.3: Развојна средина LazyVim базирана на Neovim.

Поради низата на предности, Neovim е бр. 1 најсакан едитор од заедницата.³²

За да го испробате, може да го инсталирате LazyVim³³ која е развојна средина со вклучени поставки и додатоци прикажана на сл. В.3.

B.3 Основи на програмирање со Python

Добар вовед во програмскиот јазик Python во рамки на екосистемот за научна и инженерска работа е даден во Скриптата за SciPy (Varoquaux et al., 2024)³⁴ која е резултат на колективен напор на околу 100 соработници и е достапна со слободна лиценца Creative Commons³⁵. 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 Cython³⁶.

Пред да започнеме, мора да нагласиме дека ќе користиме Python 3.³⁷ Кодот ќе го пишуваме во интерактивната конзола IPython. Алтернативно,

³²<https://survey.stackoverflow.co/2022/#section-most-loved-dreaded-and-wanted-integrated-development-environment>

³³<http://www.lazyvim.org/>

³⁴Scipy Lecture Notes <http://scipy-lectures.org/>

³⁵<https://creativecommons.org/>

³⁶<https://cython.org/>

³⁷Python 2 го пројде својот век на траење во јануари 2020 година и повеќе не се одржува.

може да ја искористиме IPython конзолата вградена во Spyder.

```
print("Hello world!")
```

```
Hello world!
```

Пред да започнеме, корисно е да ги видиме максимите што стојат зад дизајнот и философијата за пишување на код во Python. Тие се содржани во „Зен 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`)

Сè во Python е објект. Така, секој тип на променлива поддржува различни методи за манипулација со неа, како и атрибути што ги опишуваат.

Python поддржува динамички типови, со други зборови, интерпретерот го одредува типот на променливата при извршувањето. Типот на променлива исто така може да се менува динамички.

```
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 неколку корисни типови на контејнери, во кои може да се складираат колекции на објекти.

- `str` – низа на знаци или стринг,
- `list` – листите содржат подредена низа на објекти кои можат да се од различен тип,
- `dict` – речници составени од парови „ключ: вредност“ што поддржуваат брзо пребарување,
- `tuple` – n -торки кои се налик на листи, но тие се неменливи, односно нивната содржина не може да се менува по декларирањето,
- `set` – неподредени множества на уникатни објекти.³⁸

³⁸Не може истиот објект да се јави два пати во множеството.

Стрингови

```
b = 'Hello'  
type(b)  
Out: str
```

Стринговите поддржуваат некои математички операции. За нивно декларирање ' и " може подеднакво да се користат.³⁹

```
b + " World!"  
Out: 'Hello World!'  
  
'=' * 60  
Out: '====='
```

Како објекти стринговите вклучуваат методи за манипулација со нив. Можете да ги видите сите методи ако го напишете името на променливата којашто е стринг, па . и притиснете на копчето <Tab> .

```
b.upper()  
Out: 'HELLO'  
  
b.lower()  
Out: 'hello'  
  
b.startswith('he')  
Out: False  
  
b.endswith('lo')  
Out: True  
  
b.replace('e', 'a')  
Out: 'Hallo'
```

Листи

```
colors = ['red', 'blue', 'green', 'black', 'white']  
type(colors)  
Out: list
```

Индексирањето во Python започнува од 0 со која е означен првиот член, а со -1 е означен последниот член.

```
colors[2]  
Out: 'green'  
  
colors[-1]  
Out: 'white'
```

³⁹Наводниците не треба да се стилизирали!

Селектирањето на подлисти⁴⁰ се прави со синтаксата `[start: stop]` и е со отворен интервал од десно, односно ги дава елементите со индекси i за кои $\text{start} \leq i < \text{stop}$. Така, i се движи од `start` до `stop - 1`.

Синтаксата поддржува и поставување на чекор како трет параметар `[start: stop: stride]`.

Секој од трите параметри е опционален и може да се изостави.

```
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 променливата што е листа . и притиснете Tab .

```
colors.append(10)
colors
Out: ['yellow', 'blue', 'green', 'black', 'white', 10]

colors.pop()
Out: 10
```

⁴⁰Англ. *slicing*.

```
colors
Out: ['yellow', 'blue', 'green', 'black', 'white']
```

★ Важно! Како може да создадеме нова листа со иста содржина како `colors`? Да го направиме тоа на следниот начин:

```
boi = colors
boi
Out: ['yellow', 'blue', 'green', 'black', 'white']
```

Ако сега ја променим `boi` ќе добијеме:

```
boi[0] = 'жолта'
boi
Out: ['жолта', 'blue', 'green', 'black', 'white']
```

Но и:

```
colors
Out: ['жолта', 'blue', 'green', 'black', 'white']
```

Што се случува? Зошто е променета содржината на `colors`?

Проблемот доаѓа од таму што преносот на променливите во Python се прави по референца ако тие се изменливи (листи, речници и множества), а не по вредност како кога тие се неменливи (сите останати типови на променливи).

Така, кога ќе извршиме `boi = colors`, нема да биде направена копија од објектот листа на `boi` кој го референцира `colors`, туку новата променлива `boi` ќе биде направена да го референцира истиот тој објект. Овој концепт е контраинтуитивен и може да доведе до грешки во кодот кои се тешки за откривање!

За споредба кај неменливите типови преносот е по вредност:

```
c = 0.5
d = c
d += 1
d
Out: 1.5

c
Out: 0.5
```

За да направиме нова листа со истата содржина може да ја искористиме методата `copy()` или со селектирање на целата листа како подлиста:

```
boi = colors.copy()
boi = colors[:]
```

Речници

```
pins = {'Urosh': 5752, 'Arpad': 5578}
pins['Igor'] = 5915
pins
Out: {'Urosh': 5752, 'Arpad': 5578, 'Igor': 5915}

pins['Arpad']
Out: 5578

pins.keys()
Out: dict_keys(['Urosh', 'Arpad', 'Igor'])

pins.values()
Out: dict_values([5752, 5578, 5915])

pins.items()
Out: dict_items([('Urosh', 5752), ('Arpad', 5578), ('Igor', 5915)])

'Igor' in pins
Out: True
```

Контрола на текот на програмата

Блоковите код во Python се одредени според неговата индентација. Стандардно секое ниво на индентација соодветствува на 4 празни места, кои едиторот автоматски ги вметнува при притискање на `Tab`. На овој начин, се избегнува потребата од загради за ограничување на блоковите, што го намалува бројот на знаци, а истовремено правилната индентација станува задолжителна. Двете придонесуваат за зголемена читливост на кодот.

if/elif/else

```
a = 10

if a == 1:
    print(1)
elif a == 2:
    print(2)
else:
    print('A lot')

Out: A lot
```

`if` подразбира `False` за:

- секој број еднаков на нула (`0`, `0.0`)
- празен контејнер (`list`, `dict`)

- `False`, `None`

а подразбира `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

for name, tel in tels.items():
    print(name, tel)

Out: Urosh 5752
      Arpad 5578
      Igor 5915

vowels = 'aeiou'
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
```

Интересно е што `for` јамката во Python поддржува `else` гранка која се извршува ако `for` јамката не е прекината во извршувањето, односно стигнала до крај.

```
for element in a:
    if element < 0:
        print(element)
        break
else:
    print('No zero found!')

Out: No zero found!
```

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
```

while

```
n = 1
while True:
    print(n)
    n *= n
    if n > 100:
        break
```

```
Out: 1
     2
     4
     8
     16
     32
     64
```

Генерирање на листи во еден ред

Python нуди механизам за генерирање листа од друга листа во еден ред.⁴¹ Оваа алатка е многу моќна, но најдобро е да се користи 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]
```

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
```

⁴¹Англ. *list comprehension*.

```
def disk_area(radius):
    return 3.1415 * radius**2
disk_area(3)
```

Out: 28.273500000000002

Функциите може да се дефинираат со задолжителни позициони аргументи⁴² и опционални аргументи со клучен збор⁴³.

```
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 in K
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 е пожелно да имаат помошни описи наречени `docstring` поместени веднаш под декларацијата. Постојат неколку конвенции за нивно форматирање, една од нив е онаа на пакетот NumPy што ќе ја користиме во материјалов. За да пристапиме до описот на една функција, треба да допишеме `?` по името на функцијата во IPython конзолата.

speed_of_sound?

⁴²Англ. *positional arguments*.

⁴³Англ. *keyword arguments*.

```
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.4 NumPy, SciPy и Matplotlib

NumPy, SciPy и Matplotlib се трите библиотеки што ја сочинуваат основата за користење на Python – јазик за општа намена, за нумеричка обработка на податоци. Во овој дел, накратко ќе покажеме некои од нивните функционалности, а ќе навлеземе подлабоко во останатите глави од материјалов. За да ги инсталлираме овие 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 што овозможуваат брзи нумерички операции.

```
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 *`. На овој начин, го избегнуваме преполнувањето на основниот простор на имиња⁴⁴, а и придонесуваме за зачувување на засебен простор со имиња за секој од импортирани модули. Ова овозможува и одлично автоматско комплетирање на имиња на модули, функции или променливи во Python.

Создавање на NumPy низи

```
a = np.arange(10) # 0 .. n-1 (!)
b = np.arange(1, 9, 2) # почеток, крај, чекор
c = np.linspace(0, 1, 6) # почеток, крај, број на точки
d = np.ones((3, 3)) # (3, 3) е n-торка
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.]]))
```

Повеќедимензионални низи

```
a = np.array([[0, 1, 2], [3, 4, 5]])
a
Out: array([[0, 1, 2],
            [3, 4, 5]])

a.ndim
Out: 2

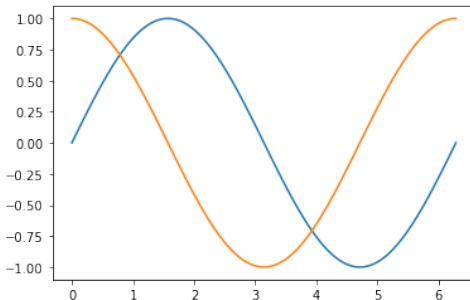
a.shape
Out: (2, 3)

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]
```

⁴⁴Добрата структурираност на просторите на имиња (namespaces) е една од важните одлики на Python, како што е и наведено во [Зен на Python](#).



Сл. В.4: Временски облик на низите добиени со функциите за синус и косинус.

```
arr = np.arange(10)
arr**2
Out: array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

Matplotlib графикони

Matplotlib е флексибилна библиотека што овозможува лесно и брзо исцртување на променливи, 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 и да исцртаме низи генериирани со функциите синус и косинус.

```
from matplotlib import pyplot as plt

x = np.linspace(0, 2 * np.pi, 100)
sin = np.sin(x)
cos = np.cos(x)

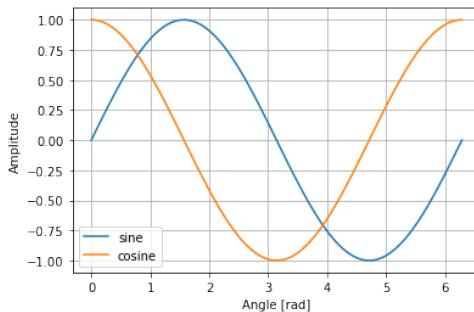
plt.plot(x, sin)
plt.plot(x, cos)

Out: [
```

Графиконот може да го видиме на сл. В.4.

§ Дополнително. Ако IPython не е стартуван со `ipython --matplotlib` за интерактивно плотирање, тогаш за да го прикажеме графиконот треба дополнително да ја извршиме функцијата `plt.show()`.

Можеме да го подобриме графиконот со додавање на информации за оските и мрежа.



Сл. B.5: Временски облик на низите со синус и косинус со дополнителни 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. B.5.

B.5 Создавање на GUI во Python

Креирањето на Графички кориснички интерфејс (GUI)⁴⁵ овозможува полесно експериментирање со параметрите на различните алгоритми за процесирање на аудиосигналите. Исто така, GUI-то овозможува интуитивна демонстрација и употреба на алгоритмите без навлегување во теоријата и имплементацијата, односно кодот, а со тоа и едноставно споделување со пошироката заедница.

Постојат низа на библиотеки за создавање на GUI со Python. Повеќето од нив овозможуваат создавање на GUI-а поддржани на различните платформи: GNU/Linux, Windows и MacOS.

Tk

Основниот алатник за создавање на GUI во Python е TkInter кој претставува Python врапер на алатникот Tk GUI напишан во C.⁴⁶ Тој е вклучен во секоја инсталација на Python и претставува слободен софтвер, па може да се употреби и за создавање на комерцијални апликации.

⁴⁵Англ. *Graphical User Interface*.

⁴⁶Tkinter Python wiki <https://wiki.python.org/moin/TkInter>

wxWidgets

Една алтернатива на Ткинтер е `wxPython`⁴⁷ кој е врапер за повеќеплатформскиот GUI алатник `wxWidgets`⁴⁸, напишан е во C++, во кој се создадени многу апликации вклучувајќи ги Audacity и KiCad, кои ги споменавме во [Додатокот А](#).

GTK

Постојат и помодерни и помоќни платформи за создавање на GUI во Python. Една од нив ја сочинуваат `PyGTK`⁴⁹ и поновиот `PyGObject`⁵⁰ врапер за GUI алатникот `GTK`⁵¹. GTK, напишан во С 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 оперативните системи што го користат системот за прозорци X11 и поновиот Wayland за Linux, тој работи и на други платформи како Windows и Macintosh. GTK е лиценциран под `LGPL`⁵². PyGTK исто така е достапна под `LGPL`.

Уште една голема предност која ја нуди GTK е алатката за брз и лесен развој на GUI-а `Glade`⁵³ прикажана на [Сл. В.6](#). Со нејзина помош може едноставно да се состави едно GUI, без потреба истото да се генерира преку пишување на код. Вака созданото 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 – PAGE⁵⁴, и `wxWidgets` – `wxGlade`⁵⁵, но тие не се толку напредни како Glade.

Qt

Уште една моќна и модерна платформа за развој на GUI-а во Python се базира на алатникот `Qt`. Qt претставува слободен софтвер за создавање на GUI-а и повеќеплатформски апликации не само на трите десктоп опе-

⁴⁷<https://wxpython.org/>

⁴⁸<https://wxwidgets.org/>

⁴⁹Wikipedia: PyGTK <https://en.wikipedia.org/wiki/PyGTK>

⁵⁰PyGObject <https://pygobject.readthedocs.io>

⁵¹The GTK Project <https://gtk.org/>

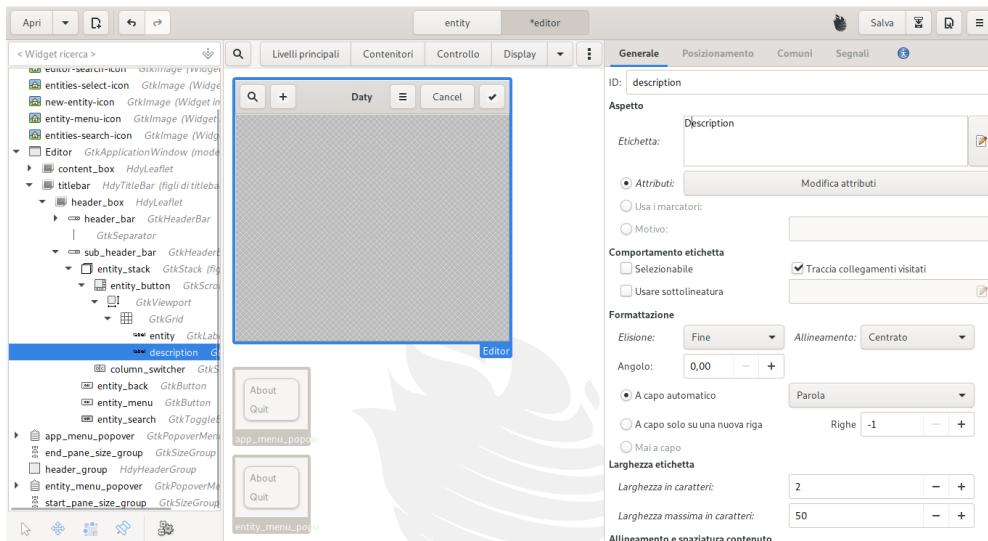
⁵²Wikipedia: GNU Lesser General Public License https://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License

Ова 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.

⁵³Glade - A User Interface Designer <https://glade.gnome.org/>

⁵⁴PAGE – Python Automatic GUI Generator <http://page.sourceforge.net/>

⁵⁵wxGlade – a GUI builder for wxWidgets <http://wxglade.sourceforge.net/>



Сл. В.6: Алатката за дизајн на GTK GUI-а Glade.

ративни системи, тук и на Android и на вгнездените микрокомпјутерски системи.⁵⁶ Qt е достапен под комерцијална лиценца, но и допустливата LGPL, како и GPL лиценцата⁵⁷.

На Qt е базирана една од најнапредните десктоп средини за оперативните системи GNU/Linux – KDE Plasma⁵⁸ 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 PySide.⁵⁹ Тој е достапен под истите лиценци како и алатникот Qt, а е развиен како одговор на PyQt⁶⁰ враперот произведен од друга компанија, кој не вклучува LGPL лиценца. PySide поддржува создавање на апликации за GNU/Linux, Windows и MacOS, а во план е и поддршка за Android апликации.

За дизајн на GUI-а во Qt постои QtCreator⁶¹ прикажан на Сл. В.7. Тој претставува развојна 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 фајл, кој може потоа да се искористи за имплементација на

⁵⁶Wikipedia: Qt [https://en.wikipedia.org/wiki/Qt_\(software\)](https://en.wikipedia.org/wiki/Qt_(software))

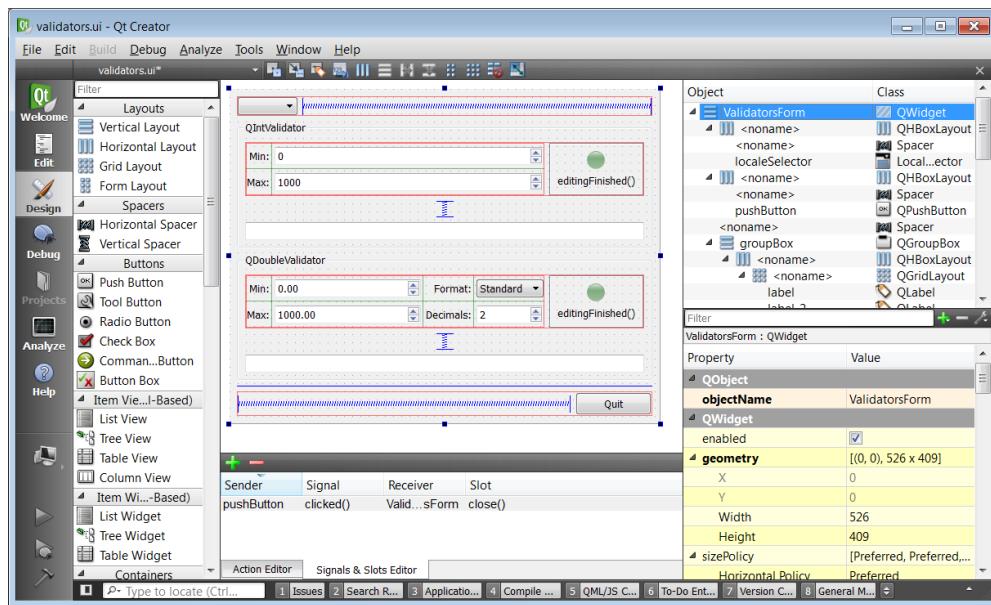
⁵⁷GNU General Public License https://en.wikipedia.org/wiki/GNU_General_Public_License

⁵⁸<https://kde.org/>

⁵⁹Wikipedia: PySide <https://en.wikipedia.org/wiki/PySide>

⁶⁰Wikipedia: PyQt <https://en.wikipedia.org/wiki/PyQt>

⁶¹Wikipedia: QtCreator https://en.wikipedia.org/wiki/Qt_Creator



Сл. В.7: QtCreator за дизајн на GUI во Qt.

апликацијата во различни програмски јазици.

Kivy

Конечно, Kivy⁶² е модерна слободна Python библиотека за развој на мобилни апликации и други софтвери дизајнирани со природни интерфејси, како на пр. тие што работат со допир. Таа е достапна под допустливата MIT лиценца и работи на Android, iOS, GNU/Linux, Windows и Macintosh.

⁶²<https://kivy.org/>

Литература

Момчило Богданов и Софија Богданова. Дигитално процесирање на сигнали. Електротехнички факултет, Скопје, 1997.

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://neuralnetworksanddeeplearning.com>.

Ted Painter and Andreas Spanias. Perceptual coding of digital audio. *Proceedings of the IEEE*, 88(4):451–515, 2000.

Lawrence R. Rabiner and Ronald W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall signal processing series. Prentice-Hall, 1978.

Lawrence R Rabiner and Ronald W Schafer. *Theory and applications of digital speech processing*, volume 64. Pearson Upper Saddle River, NJ, 2011.

Julius O. Smith. *Spectral Audio Signal Processing*. Stanford, 2011. URL <https://ccrma.stanford.edu/~jos/sasp/>.

Andreas Spanias, Ted Painter, and Venkatraman Atti. *Audio signal processing and coding*. John Wiley & Sons, 2006.

Gael Varoquaux, Valentin Haenel, Emmanuelle Gouillart, Zbigniew Jędrzejewski-Szmeik, Ralf Gommers, Fabian Pedregosa, Olav Vahtras, Pierre de Buyl, Gert-Ludwig Ingold, Nicolas P. Rougier, and et al. Scipy Lecture

Литература

Notes: Release 2015.1 beta, 2024. URL <https://lectures.scientific-python.org/>.

Udo Zölzer. *DAFX: Digital Audio Effects*. John Wiley & Sons, Ltd, 2011.

Udo Zölzer. *Digital Audio Signal Processing, 3rd Edition*. John Wiley & Sons, Ltd, 2022.