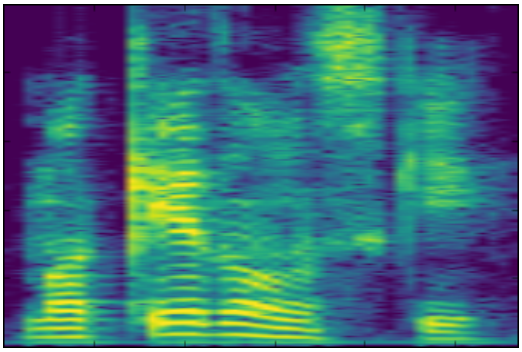


Дигитални аудиосистеми

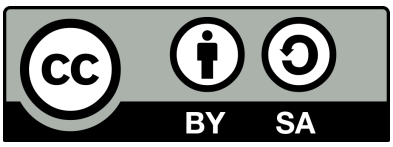
Бранислав Геразов

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

Дигитални аудиосистеми ∼ Предавања и вежби v0.85

Скрипта од предавањата и вежбите по предметот Дигитални аудиосистеми

Институт за електроника
Факултет за електротехника и информациски технологии
Универзитет Св. Кирил и Методиј во Скопје, Македонија



Содржина

Поглавје 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, така преодот на звукот од електричен во дигитален домен уште повеќе го издигнал нивото на поимањето на звукот во техниката и секојдневието. Во извесна смисла станува збор за втора револуција и тоа: во начинот на зачувување на звукот, во неговата обработка и анализа, како и во неговата масовна дистрибуција.

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

1.1 Предности на дигиталниот звук

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

Поширок динамички опсег

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

Зголемена отпорност на шум

Дигиталниот аудио запис со својата еднозначност овозможува отпорност на загадувањето на звучниот сигнал со шум. Додека кај аналогните системи шумот предизвикан од електромагнетните пречки, како и термичкиот шум се акумулираат во звучниот сигнал долж неговото движење низ електронските кола; во дигиталните системи таква акумулација на шум нема. Професионалната дигитална аудио опрема работи со 24–битно кодирање на звукот, што

соодветствува со однос сигнал-шум од 144 dB, а единствениот присутен шум е оној кој ќе влезе во аудио системот пред степенот за дигитализација.

Усовршено и олеснето умножување

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

Уште повеќе, умножувањето на дигиталните аудиозаписи е олеснето поради зголемената брзина со која можат да се направат копиите. Кај аналогното преснимување, поради самата карактеристика на аналогните носачи на звук, речиси секогаш мора да се прави во реално време. Така, за преснимување на грамофонска плоча на аудио касета и во најдобар случај мора да прође онолку време колку што трае самиот звучен запис. Истото важи и кај преснимувањето од аудио касета на касета.[1] Што значи дека за преснимување 60–минутен запис скоро секогаш се потребни 60 минути за преснимување. Од друга страна, за да преснимиме 80-минутно аудио CD на хард дискот на компјутерот ни требаат 5 минути, а за понатамошно умножување на записот во рамките на хард дискот по копија ни се потребни само 15 секунди! [2]

Механизми за корекција на грешка

Погодно е преку примената на ефикасни методи за кодирање и внесување на редунданса да се осигури интегритетот на дигиталниот аудио запис. Повеќето дигитални носачи на звук како на пример CD–то и DAT касетите имаат вградени механизми за корекција на грешка. Ако површината на дискот физички е оштетена, читачот автоматски ја употребува сета останата информација да ги реконструира изгубените податоци.

Зголемена трајност на дигиталниот запис

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

Неограничени можности за обработка и монтирање

Дигиталниот звукот за првпат може визуелно да се претстави и директно да се работи со таа негова визуелизација. Во компјутерските системи звукот е зачуван како временска низа од амплитудни вредности која лесно може да се прикаже на екраните при обработката. Тоа овозможува бескрајна прецизност при сечењето и монтирањето на аудио материјалот. Уште

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

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

повеќе како низа од податоци звукот е достапен за математичка анализа и обработка со помош на софтверски алатки, без потреба од софистициран хардвер.

Поглавје 2

Дигитализација на звукот

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

Првиот чекор во дигитализацијата е дискретизирање во време на овој аналоген електричен сигнал, уште наречено и семплирање[1]. Дискретизацијата во време се прави со периодично земање одбироци од даденият аналоген сигнал во одредени временски интервали. Добиениот временски дискретен сигнал е во вид на диракови импулси, но тој сеуште има континуирано распределени амплитуди. За да се претстави амплитудата на секој од одбироците со конечен број на битови, потребно е заокружување на нивните амплитуди до извесни конкретни вредности од множеството вредности кои можат да се запишат. Ова е вториот чекор во дигитализацијата – дискретизација по амплитуда или квантизација.

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

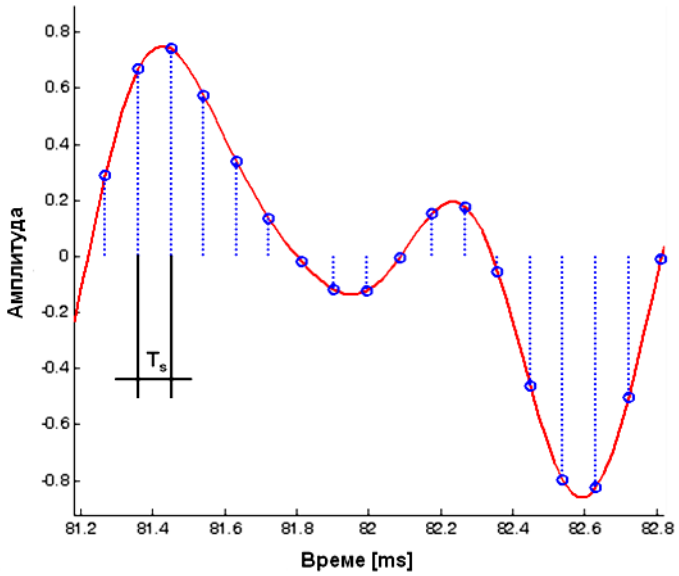
2.1 Земање одбироци или семплирање

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

$$T_s = \frac{1}{f_s}\,. \tag{2.1}$$

Процесот на земање одбироци е прикажан на Сл. 2.1. Ова се прави со помош на соодветни Sample/Hold (SH) електронски кола, а може идеално да се претстави како множење на аналогниот сигнал со низа на диракови импулси. Главната идеа е временската информација за сигналот содржана во земените одбироци 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

[1] Важно е да се напомене дека во музиката под поимот семплирање се подразбира земањето на отсечоци од готови музички траки или звуци од музички инструменти и нивната употреба за креирање на ново музичко дело.



Сл. 2.1: Земање одбироци со фреквенција f_s, односно периода T_s.

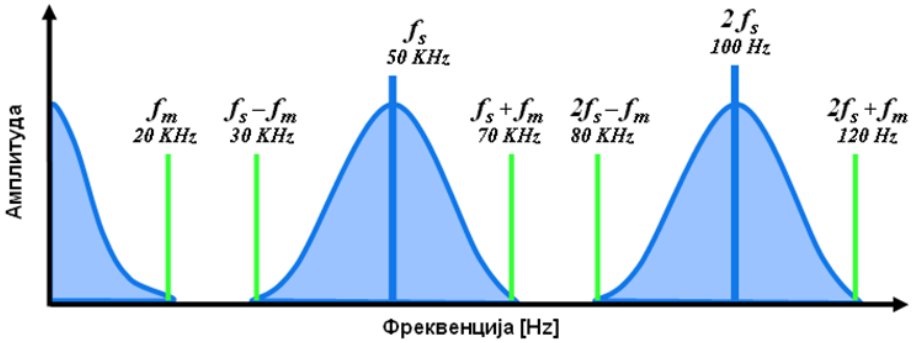
во време да се претстави со конечна низа на вредности? Што станува со информацијата за сигналот помеѓу одбироците? Сепак, математички е докажано дека ова е возможно. Станува збор за познатата теорема за земање одбироци, која за првпат ја открива Владимир Котељников[2] и ја применува во телекомуникациите во 1933та година. До нејзе независно дошле и други научници, и тоа: Хери Најквист[3], Клод Шенон[4], и Едмунд Витакер[5]. Теоремата за земање одбироци гарантира дека за веродостојно зачувување на целосната информација содржана во еден аналоген сигнал со максимална фреквенција на неговиот спектар f_m, доволно е од него да земаме одбироци со фреквенција $f_s \geq 2f_m$. Минималната фреквенција на земање одбироци $f_s = 2f_m$ се нарекува Најквистова фреквенција.

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

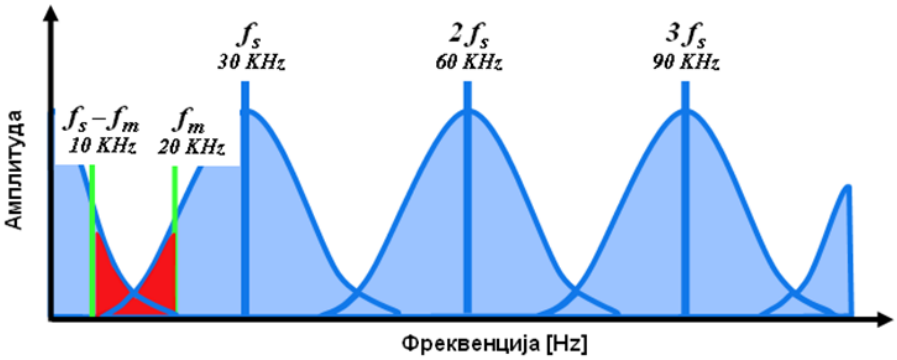
Спектарот на дискретизираниот сигнал е ист со оној на оригиналниот, со тоа што сега постојат копии на овој спектар центрирани на целобројни мултипли од фреквенцијата на земање на одбироци f_s како на Сл. 2.2. На пример, ако звукот кој е ограничен на 20 kHz го дискретизираме со f_s од 50 kHz, тогаш слики од оригиналниот спектар ќе се наоѓаат во интервалите од 30 – 70 kHz, 80 – 120 kHz итн.

Бидејќи овие копии на спектарот на сигналот не смеат да се преклопуваат потребна е фреквенција на земање одбироци поголема од $2f_m$, односно од Најквистовата фреквенција. Од

[2]Wikipedia: Vladimir Kotelnikov https://en.wikipedia.org/wiki/Vladimir_Kotelnikov
[3]Wikipedia: Haryy Nyquist https://en.wikipedia.org/wiki/Harry_Nyquist
[4]Wikipedia: Claude Shannon https://en.wikipedia.org/wiki/Claude_Shannon
[5]Wikipedia: Edmund Taylor Whittaker https://en.wikipedia.org/wiki/E._T._Whittaker



Сл. 2.2: Спектар на дискретизираниот сигнал.



Сл. 2.3: Преклопување на спектрите при ниска фреквенција на земање на одбироци.

друга страна бидејќи сигналите скоро никогаш немаат строго ограничен спектар, мора сигналот да се претфилтрира и да се сведен на фреквенциски опсег со максимална фреквенција f_m. Кога сигналот би имал спектрални компоненти над f_m, или пак кога би го дискретизирале со фреквенција $f_s < f_m$, сликите на оригиналниот спектар би се преклопувале. Преклопувањето на спектрите, прикажано е на Сл. 2.3.[6] Тоа 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 оној пред дискретизацијата.

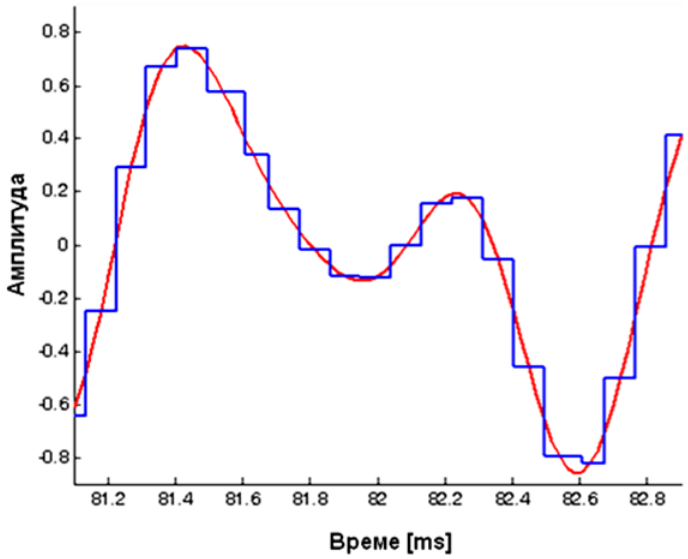
Реконструирањето на аналогниот сигнал од неговите одбироци се врши со употреба на нископропусен (НФ) филтер со гранична фреквенција f_m. Тој во фреквенциски домен има задача да ги отстрани сликите од спектарот на сигналот лоцирани околу мултиплите на фреквенцијата на земање одбироци f_s, по што ќе остане само оригиналниот спектар на сигналот. Во временски домен тоа се сведува на измазнување на дискретниот сигнал, прикажано на Сл. 2.4.

Фреквенции на семплирање во дигиталното аудио

Брзини со кои се врши земањето одбироци во дигиталното аудио се:

- 8.000 Hz – се употребува во телефонската комуникација и е доволна за пренесување на говор со зачувана разбирливост,
- 11.025 Hz и 22.050 Hz – четвртина и половина од фреквенцијата на земање на одбироци во аудио CD стандардот 44.100 Hz, се употребува за зачувување на дигитално аудио со понизок квалитет,

[6] Во англиската литература преклопувањето се нарекува aliasing, од зборот alias што значи лажно име, односно лажно претставување.



Сл. 2.4: Реконструкција на аналогниот од дискретизираниот сигнал со нископропусно филтрирање.

- 32.000 Hz – се употребува во miniDV стандардот за дигитално видео кој го користат дигиталните видео камери, во Digital Audio Tape (DAT) стандардот во модовите за зголемено траење на касетата (long play mode), како и Германското дигитално сателитско радио Digitales Satelitenradio,
- 44.100 Hz – фреквенција на земање одбироци кај аудио CD–то, наследена од ИКМ адаптерите, исто така најчесто се употребува кај MPEG кодираното аудио (како кај VCD, SVCD, MP3),
- 48.000 Hz – дигитално аудио во употреба кај miniDV, дигиталната Телевизија, DVD, DAT, филмови и професионална аудио опрема,
- 96.000 Hz или 192.000 Hz – во употреба кај аудио DVD стандард, за аудиото во новата генерација на DVD–а со син ласер (Blu–ray Disc) и во HD–DVD (High–Definition DVD),
- 2,8224 MHz – се употребува во SACD (Super Audio CD) стандардот развиен од Сони и Филипс, кој употребува 1–битна сигма–делта квантизација. Овој начин на дигитизација не нуди предности над стандардниот кој е во општа примена.

Надсемплирање

Директниот начин на кој може да се изведе земањето одбироци со фреквенција од на пример 44,1 kHz, како кај аудио CD–то, е да се направи токму тоа – со еден AD конвертор да се одмери сигналот во кратки временски интервали со фреквенција од 44,1 kHz. Меѓутоа, ова бара влезниот аналоген НФ филтер во AD конверторот низ кој најпрвин минува сигналот, да е со многу стрма амплитудна карактеристика веднаш над $20^{от}$ kHz. Овој филтер треба во многу краток фреквенциски интервал да обезбеди слабеење од повеќе од 80 dB, колку што изнесува потребниот однос сигнал-шум за Hi–Fi репродукција на звук. Но, бидејќи динамичкиот опсег кој го нуди дигиталниот аудио CD запис е 96 dB, всушност слабеењето на филтерот треба да е уште поголемо. Во практиката, ова подразбира употреба на аналоген филтер од $8^{ми}$ или $10^{ти}$ ред што е скапо и непрепорачливо решение, поради потребата за прецизна усогласеност на составните аналогни компоненти.

Постои уште еден поекономичен начин да се реализира земањето одбироци, а тоа е со употребата на надсемплирање[7]. Надсемплирањето претставува земање на одбироци со фреквенција неколку

[7]Во англиската терминологија тоа е познато како oversampling

пати поголема од Најквистовата. Со него се постигнува олеснување на условите кои влезниот НФ филтер треба да ги задоволи при обликувањето на аналогниот сигнал. Потребното ограничување на спектарот на сигналот, сега може да се изврши во дискретен (дигитален) домен каде нископропусното филтрирање на сигналот е многу поедноставно, а и поефтино. Уште една голема предност на дигиталното филтрирање над аналогното е строгата линеарност на фазната карактеристика која е гарантирана кај ФИР[8] филтрите со конечен импулсен одѕив. Потоа може да се отфрлат вишокот одбироци (одбироци), со што доаѓаме до целната фреквенција на земање одбироци.

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

Пример 1. За да ги илустрираме придобивките од надсемплирањето ќе претпоставиме 4× надсемплирање во однос на аудио–CD стандардот, прикажано на Сл. 2.5. Тогаш фреквенцијата на надсемплирање f_s' ќе биде 4 пати поголема од номиналната (целната) f_s од 44,1 kHz и ќе изнесува:

$$f_s' = 4f_s = 4 \cdot 44{,}1 = 176{,}4\,\text{kHz}. \tag{2.2}$$

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

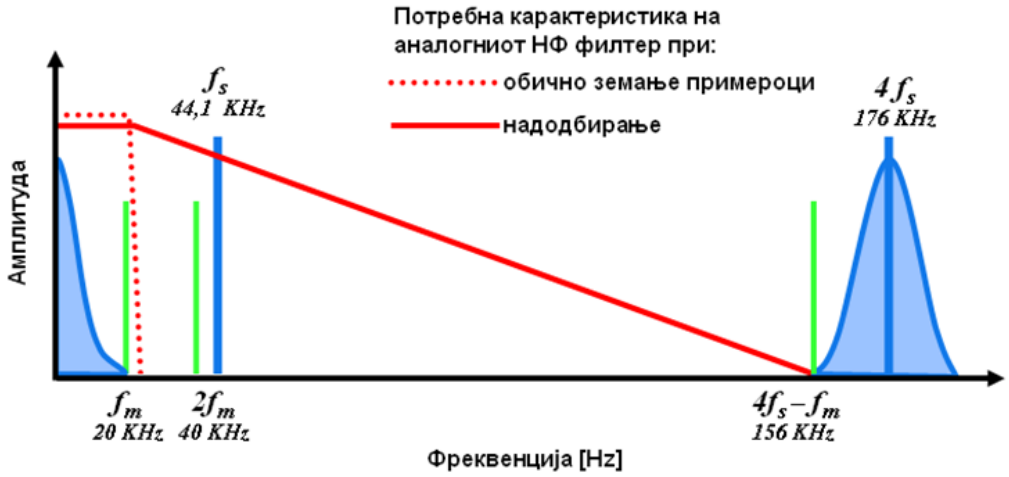
$$f_m' = \frac{f_s'}{2} = \frac{176{,}4}{2} = 88{,}2\,\text{kHz}. \tag{2.3}$$

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 нешто над 20 kHz и да трае сè до 88$^{\text{от}}$ 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 аудио сигнал ни се наоѓа до 20 kHz. Значи, фреквенцијата по која филтерот треба да го постигне бараното слабеење се поместува надесно до $176{,}4 - 20 = 156{,}4$ kHz. Следува дека сега преодниот фреквенциски опсег на филтерот изнесува 120 kHz споредено со 4,1 kHz во случајот кога не се користи надсемплирање. Ова во голема мера ја поедноставува изградбата на аналогниот филтер.

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

Потоа ја намалуваме фреквенцијата на земените одбироци преку процесот наречен скастрување[9]. Ова наједноставно се прави со отфрлање на 3 од секои 4 одбироци со што добиваме верна претстава на аналогниот звучен сигнал со спектар до 22 kHz во дигитален домен. Во стварноста не се отфрлаат по 3 одбирока од секои 4 туку самиот дигитален филтер ја пресметува вредноста

[8] Транслитерација од англиското FIR, што стои за Finite Impulse Response.
[9] Соодветниот англиски термин е downsampling.



Сл. 2.5: Благиот наклон на влезниот НФ филтер овозможен со надодбирањето во споредба со потребната карактеристика на филтерот при обично земање на одбироци.

на еден излезен одбирок на секои 4 влезни со нивно усреднување.

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

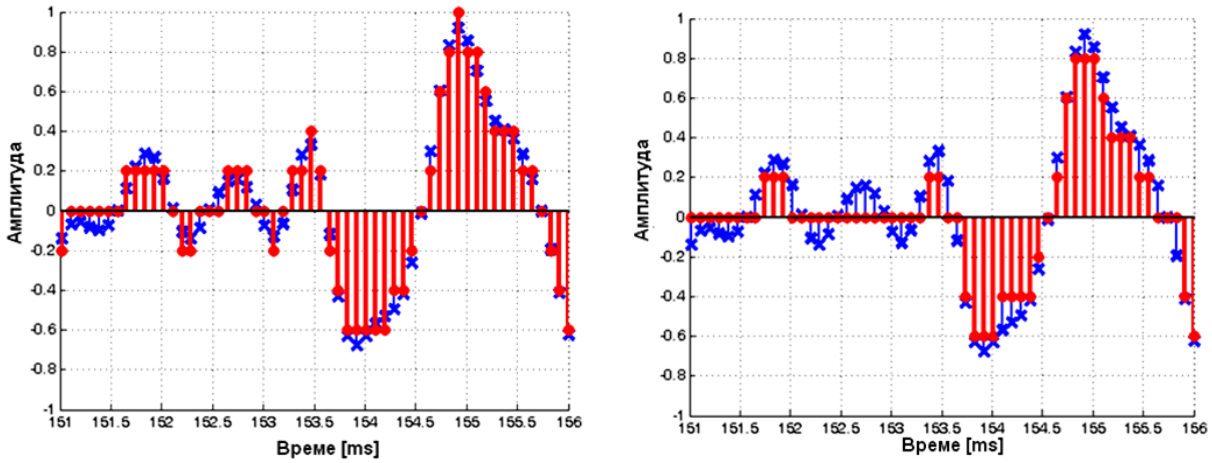
[a] На англиски ова се нарекува up-sampling.

Џитер

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

2.2 Дискретизација по амплитуда или квантизација

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



Сл. 2.6: Квантизација (црвено) на дискретизираниот сигнал (сино) со заокружување (лево) и со отсекување (десно).

уред со кој таа се изведува квантизер[10]. Бројот на битови кој квантизерот може да го генерира се нарекува уште и негова резолуција.

Квантизацијата се одвива на тој начин што квантизерот врши споредба на амплитудата на влезниот сигнал со дискретните нивоа кои тој може внатрешно да ги генерира, наречени квантизациски нивоа. Разликата помеѓу две соседни квантизациски нивоа се нарекува чекор на квантизација и се означува со q. 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[11], до најголемата прецизност која тој може да ја постигне, која е $V_{REF}/2^N$ и се кодира со битот со најмала тежина LSB[12]. Процесот е воден од внатрешната логика на квантизерот. Важно е брзината со која се доаѓа до конечниот коден збор да е поголема од брзината со која доаѓаат одбироците од влезниот сигнал.

Постојат два главни типа на квантизација кои се разликуваат според начинот на кој го ловат влезниот сигнал. Тие се квантизација со заокружување и квантизација со отсекување. Кај првата излезот на квантизерот го дава квантизационото ниво најблиско до амплитудата на влезниот сигнал, додека кај втората тоа е квантизационото ниво веднаш под амплитудата на влезниот сигнал. Типот на квантизација зависи од електронската реализација и механизмот на работа на квантизерот. Разликите помеѓу двата типа се илустрирани на Сл. 2.6.

Квантизација на дигитално аудио

При дигитализација на аудиото се употребуваат следните резолуции на квантизација:

- 1 бит – кај супер аудио-CD стандардот (SACD) кој користи надсемплирање и техники за обликување на шумот на квантизација за постигнување на голем однос сигнал-шум и при 1-битна резолуција.
- 4 бита – ретко се употребува при линеарна импулсно кодна модулација (ИКМ), а во употреба е кај адаптивната квантизација.
- 8 бита – во употреба во телефонските системи за кодирање на говорот при нелинеарна квантизација, стандардна резолуција на старите звучни карти во ерата на флопи-

[10] Влезниот аналоген филтер, земачот на примероци, квантизерот и на крај кодерот се составни делови на еден AD конвертор.
[11] Most Significant Bit.
[12] Least Significant Bit.

дискетите,[13]

- 12 бита – во употреба во 1980$^{\text{тите}}$ во дигиталните уреди за ефекти употребувани во музицирањето, како за електричните гитари.
- 16 бита – дел од аудио-CD стандардот, денес стандардна резолуција за повеќето уреди за дигитално аудио (на пример звучни картички).
- 20 бита – вградена во многу AD конвертори, во новата серија 20-битни ADAT повеќе–канални машини и кај некои звучни картички.
- 24 бита – новиот стандард за висока дефиниција, имплементиран во нови професионални уреди за дигитално аудио, новите DAT машини и во аудио DVD стандардот.
- 32 и 64 бита – не се употребува за снимање на звук, туку за неговата дигитална обработка и спектрална анализа со софтверските алатки; служи за зголемување на точноста во пресметките, особено значајно кога тие би вовеле дисторзија во 16–битно запишан звук.

Кодирање

Под кодирање се подразбира начинот на кој дискретната амплитуда на квантизерот ќе биде запишана во битови и тоа е вградено во самата негова изведба. Наједноставниот начин на кодирање на амплитудите на влезниот сигнал е со нивно директно претворање во бинарни (кодни, дигитални) зборови. Притоа паровите амплитуда-коден збор се линеарно распоредени, со најниската амплитуда запишана со најмалиот дигитален збор (сите 0), а највисоката амплитуда со најголемиот дигитален збор (сите 1). Малку посложена е нелинеарната квантизација, односно кодирање, кај која амплитудите со најголема веројатност на појавување пофино се квантизираат од оние кои поретко се појавуваат. На пример, за крајно високите амплитуди се отстапени помалку кодни зборови отколку за амплитудите поблиску до нулата.

Така, при квантизирање на говорот, посебно во класичните 8-битни телефонски системи, но и денес во врвните синтетизатори на говор како Вејвнет (Oord et al., 2016), вообичаено се употребува нелинеарна квантизација со крива за компресија, наречена и -крива.[14]. -кривата е дефинирана со G.711 стандардот на Меѓународната унија за телекомуникации и е опишана со равенството:

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

каде A во Европа е дефинирано да биде 87,6. На Сл. 2.7 е дадена преносната карактеристика на оваа крива. Како што може да се забележи, по компресијата поголем број на квантни нивоа се на располагање за квантизација на помалите амплитуди на влезниот сигнал, а помал број за поголемите. Ова одговара на статистичката распределба на амплитудите на говорниот сигнал, во кој позастапени се малите амплитуди, па и од таму нужноста за нивна поточна кватнизација.

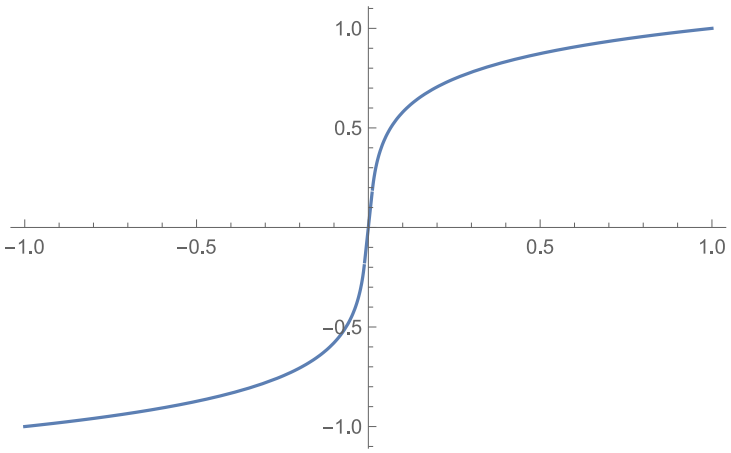
Исто така постојат и методи на квантизација кои ја кодираат разликата помеѓу две последователни амплитуди на влезниот сигнал, така наречена адаптивна квантизација. Тие се употребуваат во апликации каде големината на дигиталниот проток е критичен параметар.

Шум на квантизација

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

1. влезниот сигнал да може да се разгледува како случаен процес и

[13] Денес се јавува во уметничкото движење кое употребува 8 битен звучен израз.
[14] A-law algorithm https://en.wikipedia.org/wiki/A-law_algorithm



Сл. 2.7: Графичка претстава на преносната карактеристика добиена со А кривата за нелинеарна квантизација на говор. [15]

2. тој да има многу поголема амплитудна динамика од чекорот на квантизација q, што е еквивалентно на тоа чекорот на квантизација да е доволно мал,

шумот на квантизација може да го моделираме како бел Гаусов шум, односно тој ќе биде случаен и некорелиран со влезниот сигнал, со средна вредност 0 и максимална вредност $\pm q/2$ во случај на квантизација со заокружување, односно средна вредност $q/2$ и максимална амплитуда q, кога се врши квантизација со отсекување. Во овие услови, моќноста на шумот на квантизација ќе биде рамномерно распределена долж целиот спектар и со константа амплитуда наречена ниво на шумот на квантизација. Во праксата двава услови се исполнети речиси секогаш, особено за сигнали кои имаат мал коефициент на автокорелација, како што се говорот и музиката, како и кога квантизерот е со доволно голема резолуција. Условите не се исполнети само во многу специјални случаи како што се константни сигнали, простопериодични сигнали синхронизирани со фреквенцијата на дискретизација и многу слаби сигнали.

Односот меѓу просечната моќност на аналогниот сигнал и нивото на шумот на квантизација се нарекува однос сигнал-шум (SNR)[16] на дигиталниот сигнал. Неговата вредност може да се добие со помош на равенството:

$$SNR = 10 \log \frac{P_x}{P_N} = 10 \log \frac{\sigma_x^2}{\sigma_q^2}\,, \tag{2.5}$$

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

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

$$\sigma_x^2 = \frac{q^2}{12}\,. \tag{2.6}$$

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

$$\sigma_x^2 = \frac{1}{12 \cdot 2^{2N}}\,, \tag{2.7}$$

па (2.5) станува:

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

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

[16] Signal-to-Noise Ratio.

Табела 2.1: Варијанса на Гаусовиот процес употребен за моделирање на неколку звучни сигнали.

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

При пресметување на равенството (2.8) можеме влезниот сигнал да го моделираме како Гаусов, при што е важно тој да не ја надмине границата на квантизаторот, којашто земавме дека е 1. Ако варијансата на Гаусовиот процес $\sigma_x < 0{,}25$, тогаш веројатноста да се случи пречекорување е занемарлива, т.е. $< 0{,}0001$, што речиси секогаш е исполнето при моделирањето на звукот како Гаусов процес. Неколку вредности на варијансата на Гаусовиот процес добиен со моделирање на различни типови звук се дадени во Табела 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:

$$SNR = 6{,}02n - 1{,}25 \simeq 6n\,[\mathrm{dB}]\,. \tag{2.9}$$

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

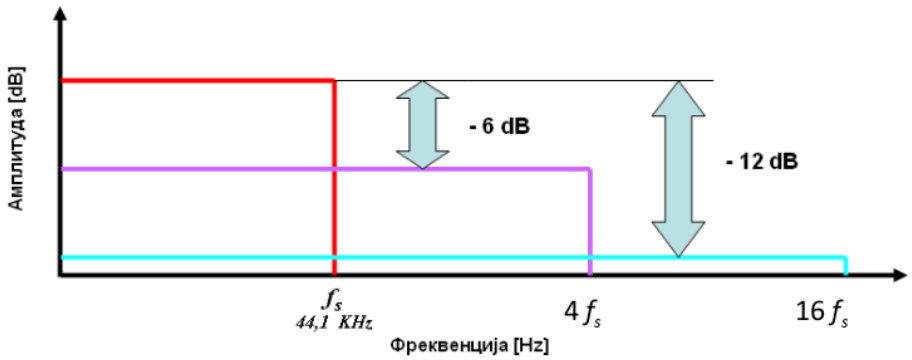
Квантизација при надсемплирање

Ако влезниот аналоген сигнал е случаен процес, па шумот на квантизација е бел и некорелиран, тој ќе биде рамномерно распределен долж целото фреквенциско подрачје кое го зафаќа сигналот. Што се случува ако направиме 4× надсемплирање – односно ако AD конверторот семплира со фреквенција од 4× 44,1 kHz? Во овој случај шумот од квантизација ќе биде распределен на 4 пати поширок фреквенциски опсег, па соодветно и амплитудата на неговата спектрална густина на моќност ќе биде 4 пати помала. Тоа соодветствува на ниво на шум помало за $10 \log {}^1\!/_4 = -6$ dB. Со други зборови 4× надсемплирање има ист ефект врз односот сигнал-шум како и зголемувањето на должината на дигиталниот збор за 1 бит. Слично, 16× надсемплирање соодветствува на зголемена резолуција на AD конверторот од 2 бита, како што е прикажано на Сл. 2.8.[17]

Ова може да се каже и на следниов начин: 16–битен AD конвертор со брзина на земање одбироци од 44,1 kHz нуди ист однос сигнал-шум како 15–битен AD конвертор со 2× надодбирање. Ако одиме понатаму по оваа логика стигнуваме до поедноставени AD конвертори кои со помалку битови по одбирок но со многу поголема фреквенција на одбирање нудат перформанси исти со оние на посложените 16 или 24 битни AD конвертори. Крајната точка во ова дискусија е реализација на квалитетни AD конвертори кои работат со 1 бит по одбирок, како кај супер аудио-CD (SACD) стандардот[18].

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

[18] SACD користи и техники за обликување на шумот дискутирани во Поглавјето за уште поголеми придобивки во SNR.



Сл. 2.8: Намалувањето на нивото на шумот со употреба на надсемплирање.

§ Дополнително. Надодбирањето одбироци нуди подробности и во обратната насока, односно кај DA конверторите. Тука повторно шумот се прераспределува на поширок фреквенциски опсег па можно е отфрлање на дел од битовите по одбирок, а притоа задржување на истиот однос сигнал-шум.

Дитер

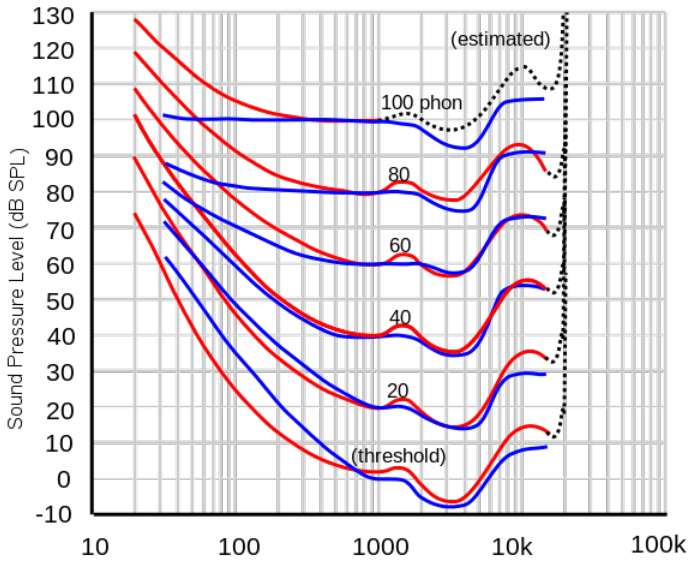
Претходната дискусија за нивото на шум на квантизација важи во случај кога шумот на квантизација е потполно некорелиран со влезниот сигнал. Во праксата, колку повеќе амплитудата на влезниот сигнал се приближува кон амплитудата означена со најмалку значајниот бит, толку повеќе се зголемува корелираноста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

Дитерот[19] е процес со кој може да се елиминира појавата на дисторзија во сигналот под дејство на шумот од квантизација. Тој е воведен од Лоренс Робертс [20], еден од основачите на интернетот, во својата магистерска теза на MIT во 1961. Во него на влезниот сигнал пред дигитализацијата му се додава мало ниво на аналоген шум. Случајниот карактер на шумот внесува случајност и во однесувањето на квантизерот при ниски влезни нивоа на сигналот – ефективно внесувајќи несигурност во неговата работа. На тој начин грешката од квантизација се декорелира од сигналот. Ова доаѓа од таму што сега, излезните нивоа од квантизерот веќе не зависат само од влезниот сигнал туку и од некорелираниот случаен процес внесен како шум.

Концептот зад треперењето може интуитивно да изгледа нелогичен, но принципот на кој функционира е многу едноставен. Треперењето се базира врз одредено специфично однесување на човековото уво. Увото е помалку чувствително на широкопојасен шум со рамномерно ниво, отколку на хармониски изобличувања во спектарот на сигналот кои се јавуваат при корелираност на шумот со сигналот. Бидејќи увото може да открие звук во опсегот на средните фреквенции 10 до 12 dB под нивото на шумот, со треперење може да се постигне 18-битна резолуција кај еден 16-битен запис. Имено, при правилен избор на сигналот со кој се изведува треперењето се овозможува да се запишат сигнали и 110 dB под максималното ниво, иако 16-битната резолуција теориски нуди динамички опсег од 96 dB. Треперењето е во редовна употреба при намалувањето на резолуцијата на еден дигитален звучен запис, како што е редовно случај при мастерирањето. Во тој случај, материјалот снимен во студиото со професионална опрема со 24-битна резолуција, треба да се прилагоди за дистрибуција на носач на звук – CD со 16 бита. Со внесувањето на

[19] Македонскиот термин за дитер би бил „треперење“, а следи од адаптација на англиското „to dither“, чија потекло е од старо-англискиот збор „didderen“ кој значи тресење (како од студ).

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



Сл. 2.9: Криви на еднаква чујност, или изофонски криви, измерени според Флечер и Мунсон (сино) и стандардизирани според ISO стандардот (црвено).[22]

одредено ниво на шум, дитер, пред конверзијата се овозможува зачувување на дел од вишокот информација која ја содржи оригиналниот запис. Ако ова не се направи, тогаш простото отфрлање на информацијата зачувана во 24-битна резолуција со исфлање на долните 8 битови се нарекува скастрување. Ова неминовно ќе доведе до појава на корелиран шум на квантизација како дисторзија во конечниот аудиозапис.

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

Обликување на шумот

Дитерот додава шум со рамна спектрална распределба на моќноста во аналогниот сигнал, но увото не е еднакво осетливо во целиот слушен опсег како што може да се види од изофонските криви прикажани на Сл. 2.9. Изофонските криви ја прикажуваат субјективната јачината на звукот и физичкото ниво на звучниот притисок. Први ги измериле Флечер и Мунсон (1933), па потоа и Робинсон и Дадсон (1956), за сега да бидат стандардизирани со ISO стандардот 226:2003. Тие ја отсликуваат фреквенциски нелинеарната карактеристика на сетилото за слух.[21]

Со техниката на обликување на шумот може да ја менуваме фреквенциска распределба на дитерот, што ни овозможува поголем негов дел да го распределиме надвор од фреквенцискиот опсег во кој увото е најосетливо. Притоа вкупната моќност на шумот останува иста. Со помош на овој пристап може во еден 16-битен сигнал да се постигне однос сигнал-шум на средни фреквенции дури од 150 dB. Обликувањето на шумот е особено згодно при употреба на надсемплирање, каде шумот може да се потисне надвор од слушното подрачје – над $20^{\text{от}}$ kHz. Како и треперењето и обликувањето на шумот се употребува најмногу при конверзијата на дигиталниот сигнал од поголема во помала резолуција.

[21] Повеќе за нив може да чуете во делот Психоакустика во предметот Електроакустика.

[22] "Lindos4" by Lindosland https://commons.wikimedia.org/wiki/File:Lindos4.svg#/media/File:Lindos4.svg

Пример 2. Кај 64× надсемплирање во SACD стандардот, фреквенцијата на земање одбироци изнесува 2,8 MHz, па шумот може да се распредели до 1,4 MHz што овозможува однос сигнал-шум од 100 dB при кодирање со 1 бит, што според (2.9) би требало да овозможи динамички опсег од само 6 dB.

2.3 Формати на дигитално аудио

По земањето на одбироци, 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 дигиталното аудио. Основен формат на дигитално аудио е аудиото кодирано со импулсно кодна модулација (ИКМ)[23]. ИКМ претставува најквалитетниот, 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 ИКМ и линеарно–предиктивното кодирање, ИКМ е најраспространетиот формат за кодирање на високо квалитетно аудио.

ИКМ кодираниот дигитален звучен сигнал на компјутерот се зачувува во вид на датотека, додека пак кај аудио-CD-то тој е запишан во вид на постојан проток, дефиниран според Црвената Книга на аудио-CD стандардот. Разликата е во тоа што на датотеките содржат воведен дел „header“ во кој се зачувани информации за должината на датотеката и параметрите на дигиталното аудио, како фреквенцијата на земање одбироци и резолуцијата на записот, додека кај аудио-CD-то овие информации се излишни. Форматот во кој е зачувано ИКМ аудиото во компјутерските системи зависи од оперативниот систем кој го користи компјутерот. Windows оперативните системи го користат WAV[24] форматот, развиен од страна на Микрософт за Интел базираните компјутерски системи во 1991, врз база на општиот RIFF (Resource Interchange File Format) дефиниран за чување на разни видови на податоци. Компјутерите со Macintosh платформи го употребуваат AIFF[25] форматот на датотеки, развиен од Apple во 1988. Двата паралелни стандарда се базираат на IFF форматот за зачувување на податоци воведен од Electronic Arts во 1985, со клучна разлика во редоследот на запишување на дигиталните податоци. WAV форматот најпрвин ги запишува $8^{\text{те}}$ 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 бајтите од најзначајниот кон најмалку значајниот, познато како „big–endian“.

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

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

[23] Pulse Code Modulation (PCM).
[24] Waveform audio format.
[25] Audio Interchange File Format.

Поглавје 3

Вовед во процесирање на аудиосигналите

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

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

Во духот на философијата зад слободниот софтвер, а следејќи ги светските инженерски и научни трендови, вежбите во предметот Дигитални аудиосистеми во целост ќе бидат изработени со слободен и отворен софтвер, поточно во слободниот програмскиот јазик Питон. За вовед во екосистемот на Питон направен за инженерска и научна работа и за основните системски поставувања погледнете во Додатокот А.

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

3.1 Вчитување на звук во Питон

Вчитувањето на аудио фајлови во Питон се прави со употреба на модулот Сајпај. Да го вчитаме нашиот прв звучен фајл:

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.io import wavfile
fs, wav = wavfile.read('Skopsko_stereo.wav')
```

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

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

```
whos

Variable   Type       Data/Info
------------------------------
fs         int        44100
np         module     <module 'numpy' from '/us<...>ages/numpy/__init__.pyc'>
plt        module     <module 'matplotlib.pyplo<...>s/matplotlib/pyplot.pyc'>
wav        ndarray    811782x2: 1623564 elems, type `int16`, 3247128 bytes (3 Mb)
wavfile    module     <module 'scipy.io.wavfile<...>es/scipy/io/wavfile.pyc'>
```

Гледаме дека `wav` има точно две вектор колони со по 811.782 примероци, што при `fs` од 44,100 kHz е аудио со должина од 18,4 s. Ова може да го пресметаме на следниот начин:

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

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

3.2 Скратување на звукот и префрлање во моно

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

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

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

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

или:

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

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

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

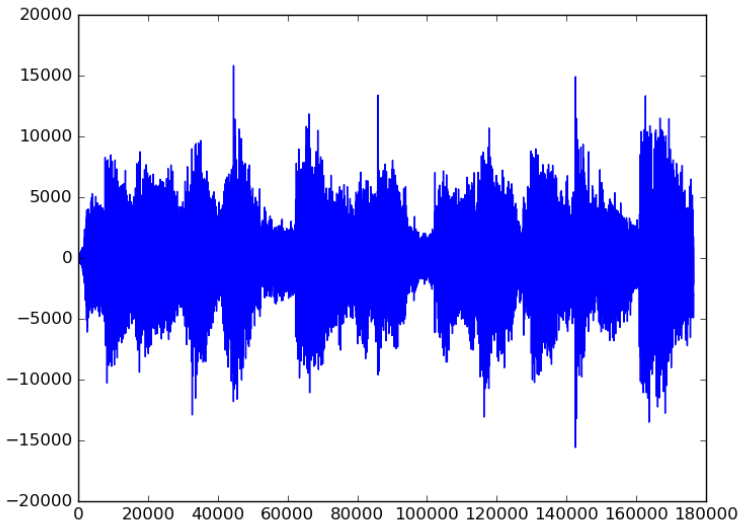
што може да се пресмета и како:

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

односно наједноставно со:

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

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



Сл. 3.1: Приказ на аудиосигналот `wav_mono` како вектор од елементи од типот `int16`.

3.3 Прикажување на аудиосигналот

За да го прикажеме аудиосигналот ќе го искористиме Матплотлиб:

```
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 се движат соодветно од -2^{15} до $2^{15}-1$.

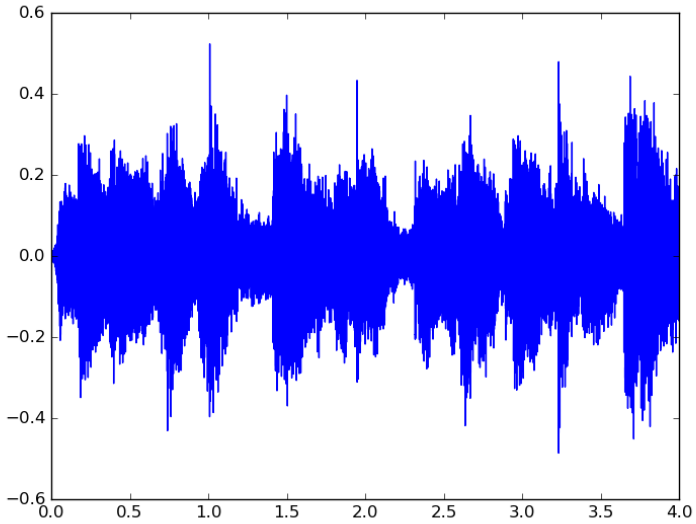
За да ги доведеме примероците во опсег од -1 до 1, сè што треба да направиме е да го поделиме векторот `wav_mono` со 2^{15}:

```
wav_mono /= 2**15
whos

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

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

```
wav_mono = wav_mono / 2**15
t = np.arange(0, wav_mono.size) / fs
%matplotlib
plt.figure()
plt.plot(t, wav)
```

Со што ќе биде исцртан графиконот прикажан на Сл. 3.2.



Сл. 3.2: Приказ на аудиосигналот `wav_mono` како вектор од елементи од типот `float64` .

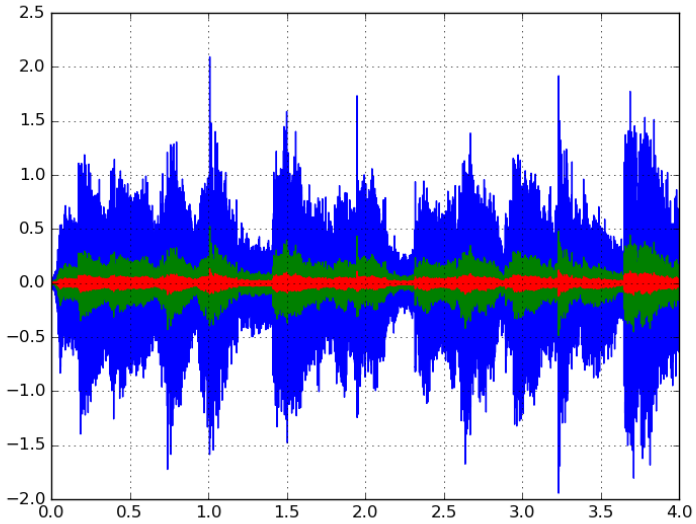
3.4 Преслушување на аудиосигналот

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

```
wavfile.write('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 Питон го запишал аудиофајлот со 64-битна прецизност наместо во оригиналната 16-битна. За ова да го поправиме треба да напишеме:

```
wavfile.write('Skopsko_mono.wav', fs, np.array(wav_mono * 2**15, dtype='int16'))
!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.
```



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

3.5 Менување на амплитудата на аудиосигналот

За понатамошна работа ќе ја искористиме Јупајтер Кјут конзолата, види Додаток А. Во неа треба повторно да ги вчитаме потребните пакети:

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

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

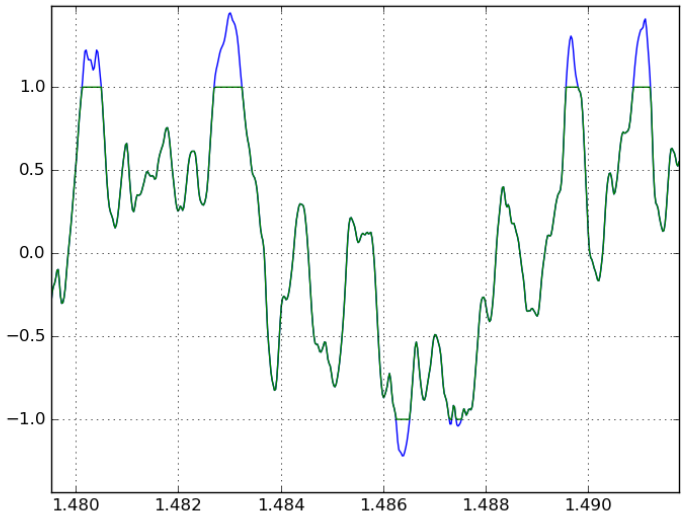
```
wav_glasno = wav_mono * 4
wav_tivko = wav_mono * .25
plt.figure()
plt.plot(t, wav_glasno)
plt.plot(t, wav_mono)
plt.plot(t, wav_tivko)
plt.grid()
```

Добиениот графикон е даден на Сл. 3.3. Може да видиме дека засилениот сигнал `wav_glasno` има амплитуда над дозволениот опсег од -1 до 1. Ова значи дека при снимањето на сигналот во wav фајл вредностите надвор од опсегот нема да бидат запишани, туку ќе бидат пресечени. Ова може да го чуеме со `play` наредбата.

```
wavfile.write('wav_glasno.wav', fs, np.array(wav_glasno, dtype='int16'))
!play wav_glasno.wav
```

Оваа појава на пресекување на амплитудите на звучниот сигнал се нарекува дисторзија и во некои типови на музика се користи како пожелен аудиоефект. За да ја прикажеме дисторзијата графички може да напишеме:

```
wav_dist = wav_glasno.copy()
wav_dist[wav_dist > 1] = 1
```



Сл. 3.4: Приказ на дисторзија во аудиосигналот `wav_glasno`.

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

Копирањето во првата линија е нужно за да го задржиме оригиналниот вектор `wav_glasno`. Со приближување може да го добиеме приказот на Сл. 3.4.

3.6 Нормализација на аудиосигналот

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

$$L = 20 \log \frac{A_{max}}{1} \text{ [dBFS]}, \tag{3.1}$$

каде како референтно е земено максималното ниво во дигитален домен 1. Ова дигитално ниво на звукот се изразува во dBFS, што е кратенка од dB од полна скала[1]. Од (3.1) следува дека за максималната дозволена амплитуда аудиосигналот има ниво од 0 dBFS. Во праксата вообичаено звучните сигнали се нормализираат на амплитуда помала од 1, за при нивното понатамошно процесирање или пренос да не дојде до дисторзија.[2]

Да напишеме 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 може да ја употребиме за нормализација на звучните сигнали:

[1] На англиски *Full Scale*.
[2] Оваа амплитудна маргина која што се остава до дозволениот максимум на англиски се нарекува headroom.

```
wav_mono_0dBFS = normalize(wav_mono, 0)
wav_mono_3dBFS = normalize(wav_mono, -3)
wav_mono.max()
wav_mono_0dBFS.max()
wav_mono_3dBFS.max()
```

3.7 Генерирање на звук во Питон

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

```
sound = np.sin(2*np.pi*200*t)
plt.plot(sound)
wavfile.write('sin.wav', fs, np.array(sound * 2**15, dtype='int16'))
!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 Плума со наредбата:

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

Во скриптата внесете го следниот код:

```
import numpy as np
from scipy.io import wavfile
import sys
import os

# користејќи го модулот sys можеме на нашата скрипта да и пренесеме
# влезни параметри преку командната линија
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('sin.wav', fs, np.array(sound * 2**15, dtype='int16'))
os.system('play sin.wav')
```

и повикајте ја од ИПитон[3]:

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

§ Дополнително. Користејќи ја скриптата `make_sound.py` тестирајте ги границите на вашиот слух!

[3]За да го смените јазичето искористете ја комбинацијата `ctrl-pgup/pgdwn`.

Поглавје 4

Филтри

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

4.1 Основи на дигиталните филтри

Дигиталните филтри претставуваат линеарни и временски инваријантни (ЛВИ), односно линеарни и инваријантни на поместување дискретни системи[1] Овие две особини изискуваат ако за даден влезен сигнал $x(n)$ системот го дава излезниот сигнал $y(n) = H\{x(n)\}$, тогаш мора да важи (Богданов, 1997):

$$H\{\sum_{m=1}^{M} a_m x_m(n)\} = \sum_{m=1}^{M} a_m y_m(n)\,, \quad \forall a_m \quad \text{– линеарност и} \tag{4.1}$$

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

Секој ЛВИ систем е во потполност опишан од неговиот импулсен одзив $h(n)$ односно неговата преносна функција во z-домен $H(z)$, или пак конечно од неговата фреквенциска преносна функција во Фуриеов домен $H(\omega)$. Како и за останатите ЛВИ 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):

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

$$Y(z) = H(z)X(z)\,, \tag{4.4}$$

$$Y(\omega) = H(\omega)X(\omega)\,. \tag{4.5}$$

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

$$h(n) \equiv 0\,, \quad n < 0\,. \tag{4.6}$$

[1] Linear time-invariant (LTI), односно linear shift-invariant (LSI) системи.

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

$$\sum_{n=-\infty}^{\infty} |h(n)| < \infty \,. \tag{4.7}$$

Типови филтри според должината на импулсниот одзив

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

- конечен импулсен одзив (ФИР[2]) и
- бесконечен импулсен одзив (ИИР[3]).

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

Сите ЛВИ системи кои се практично применливи како дигитални филтри можат да се опишат со диференцната равенка:

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

Според ова равенство секој одбирок 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. ИИР филтрите секогаш се реализираат со повратна врска, од каде произлегуваат и проблемите со нивната стабилност. ФИР филтрите можат да бидат реализирани и со повратна врска, но најчесто се без неа. Преносната функција на системот опишан од диференцната равенка можеме да ја добиеме од (4.8):

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

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

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

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

$$q = p = N \,, \tag{4.12}$$

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

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

Типови филтри според амплитудната карактеристика

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

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

[2]Од англиското Finite impulse response.
[3]Од англиското Infinite impulse response.

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

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

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

- нископропусни (НП),
- високопропусни (ВП),
- пропусни на опсег (ПО),
- непропусни на опсег (НО) и
- сепропусни (СП).[4]

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

$$\tau(\omega) = -\frac{\mathrm{d}\phi(\omega)}{\mathrm{d}\omega} = \mathrm{const}\,. \tag{4.16}$$

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

Типови ФИР филтри со линеарна фазна карактеристика

Строго линеарната фазна карактеристика не може да биде постигнато со ИИР или со аналогните филтри (Богданов, 1997), туку само со ФИР филтри чив импулсен одѕив е симетричен во однос на средниот примерок $h(\frac{N-1}{2})$:

$$h(n) = h(N-1-n)\,, \tag{4.17}$$

или пак кога е антисимтеричен во однос на него:

$$h(n) = -h(N-1-n)\,. \tag{4.18}$$

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

Постојат четири типови на ФИР филтри со линеарна фазна карактеристика:

Тип I – Симетричен импулсен одѕив, N непарен

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

$$h(n) = h(0)\delta(n) + h(1)\delta(n-1) + \cdots + h(\tfrac{N-1}{2})\delta(n-\tfrac{N-1}{2}) + \cdots \tag{4.19}$$

$$\cdots + h(1)\delta(n-(N-2)) + h(0)\delta(n-(N-1))\,, \tag{4.20}$$

па за преносната функција имаме:

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

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

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

[4] Овој тип на филтри наоѓа примена кај системите за синтеза на дигитално ехо и реверберација.

каде:

$$a(n) = \begin{cases} h(\frac{N-1}{2}), & n = 0 \\ 2\,h(-n + \frac{N-1}{2}) & n = 1,2,\ldots,\frac{N-3}{2} \end{cases} . \tag{4.24}$$

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

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

Тип II – Симетричен импулсен одзив, N парен

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

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

каде:

$$b(n) = 2\,h(-n + \tfrac{N}{2})\,, \quad n = 1,2,\ldots,\tfrac{N}{2}\,. \tag{4.27}$$

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

Тип III – Антисиметричен импулсен одзив, N непарен

Фреквенциската карактеристика на типот III е:

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

каде:

$$c(n) = 2\,h(-n + \tfrac{N-1}{2})\,, \quad n = 1,2,\ldots,\tfrac{N-1}{2}\,. \tag{4.29}$$

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

Тип IV – Антисиметричен импулсен одзив, N парен

Фреквенциската карактеристика на овој филтер е:

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

каде:

$$d(n) = 2\,h(-n + \tfrac{N}{2})\,, \quad n = 1,2,\ldots,\tfrac{N}{2}\,. \tag{4.31}$$

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

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

- метода на прозорци,

- дизајн заснован на DFT,
- оптимален дизајн на еднаквобранести филтри.

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

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

4.2 Дизајн на ФИР филтер

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

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

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

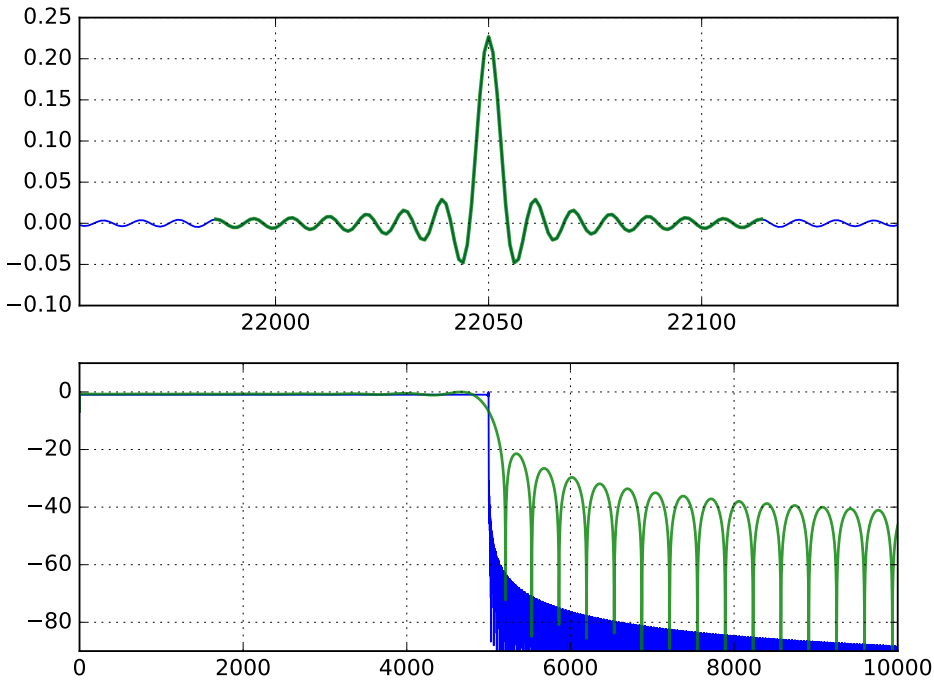
$$h_l(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_l(\omega)\, e^{jn\omega} d\omega = \frac{1}{2\pi}\int_{-\omega_l}^{\omega_l} e^{jn\omega} d\omega = \frac{\sin(n\omega_l)}{\pi n}\,. \tag{4.33}$$

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

За да ја илустрираме оваа метода ќе напишеме код кој идеалниот филтер ќе го конструира во Фуриеов домен во `n_fft = fs` точки, неговиот импулсен одзив ќе го пресмета со употреба на ИДФТ, за од него да задржиме еден дел со употреба на правоаголен прозорец со должина `n`. Поради тоа што импулсниот одзив го пресметуваме од идеалната фреквенциска карактеристика, имплементираниот алгоритам за дизајн на ФИР филтер всушност претставува комбинација од методите за дизајн базирани на ДФТ и примената на прозорци.[5]

```
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 pi до 2pi
h_l = fp.ifft(h_spec_l, n_fft)
h_l = fp.fftshift(h_l)  # го правиме системот каузален
```

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



Сл. 4.1: Импулсни одзиви и фреквенциски карактеристики на конструираниот идеален нископропусен филтер и оној добиен со методата на прозорци.

```
# %% 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 = das.get_spectrum(h_l, fs)
plt.plot(f, h_spec_l)
f, h_spec_rect = das.get_spectrum(h_rect_long, fs)
h_spec_rect = h_spec_rect - np.max(h_spec_rect)  # нормализација
plt.plot(f, h_spec_rect, lw=1.5, alpha=.8)
plt.grid()
```

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

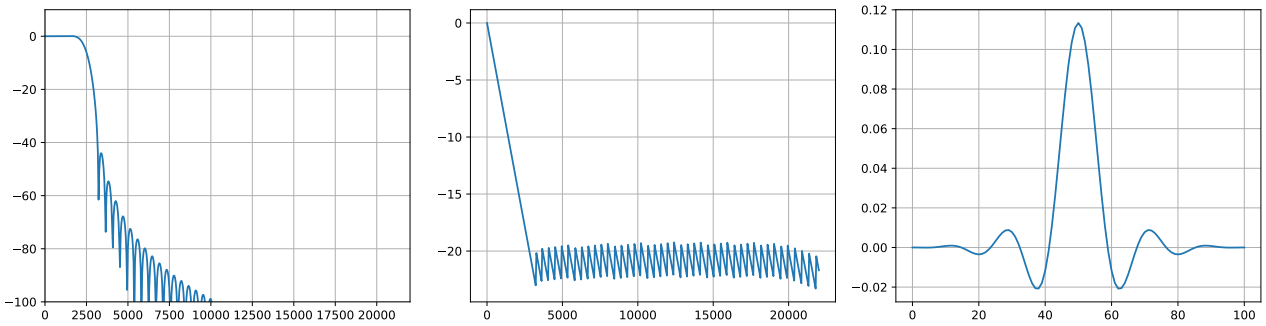
§ Дополнително. На Сл. 4.1 може да се види дека ни идеалниот импулсен одзив ја нема оригинално конструираната идеална фреквенциска карактеристика. Ова е поради тоа што вистински идеалниот импулсен одзив има бесконечно многу примероци, а овој кој ние го конструиравме има `fs`. Всушност и тој самиот како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

4.3 Филтрирање на аудиосигнал со ФИР филтер

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

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

# %% 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

# %% исцртај спектрограм
__ = das.get_spectrogram(fs, wav_orig)

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

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

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

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



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

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

✓ Задача за час. Проверете како се менуваат карактеристиките на филтерот со промена на:

а) граничната фреквенција,
б) редот на филтерот, и
в) типот на прозорецот.

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

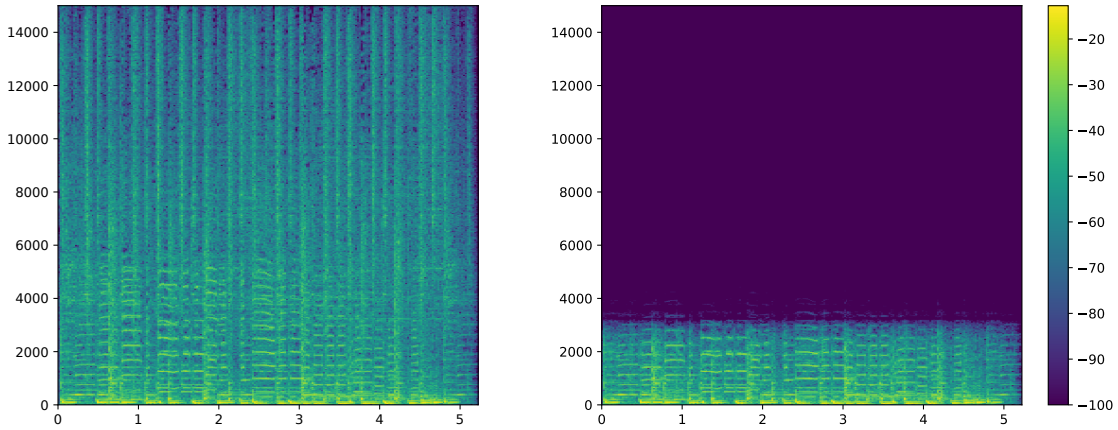
```
# %% исфилтрирај го аудиосигналот и прикажи го спектрограмот на излезниот сигнал
wav_filt = sig.lfilter(b, 1, wav_orig)
__ = das.get_spectrogram(fs, wav_filt)

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

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

4.4 Дизајн на ИИР филтри

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

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



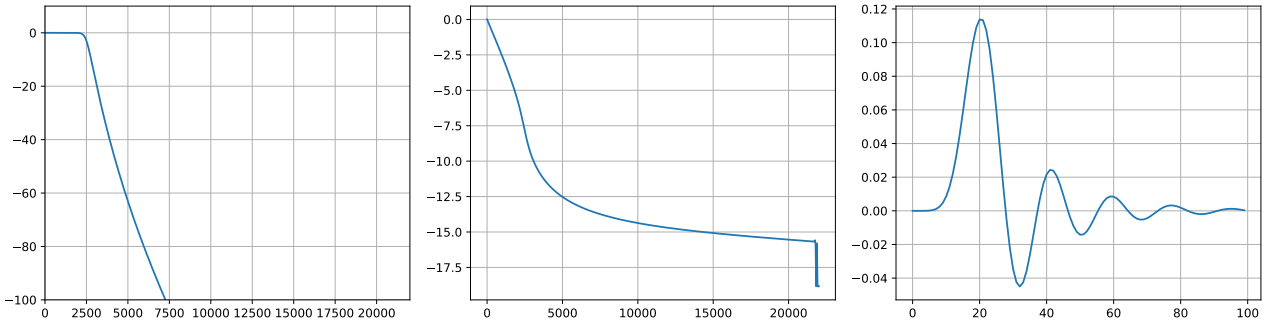
Сл. 4.3: Спектрограм на аудиосигналот `Mara` пред (лево) и по филтрирањето со ФИР филтерот (десно).

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

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

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

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

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

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



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

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

4.5 Употреба на ИИР филтри за еквализација

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

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

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

Во аналогната техника бројот на филтри е ограничен со цената на уредот, додека во дигиталната техника со процесирачката моќ. Затоа добрите аналогни аудио системи најчесто вклучуваат едноставна бас и требл[6] еквализација, додека дигитални аудио плеери вообичаено подржуваат 6 и повеќе-канална еквализација. Притоа дигиталните филтри се најчесто од ИИР тип од 4 ред,

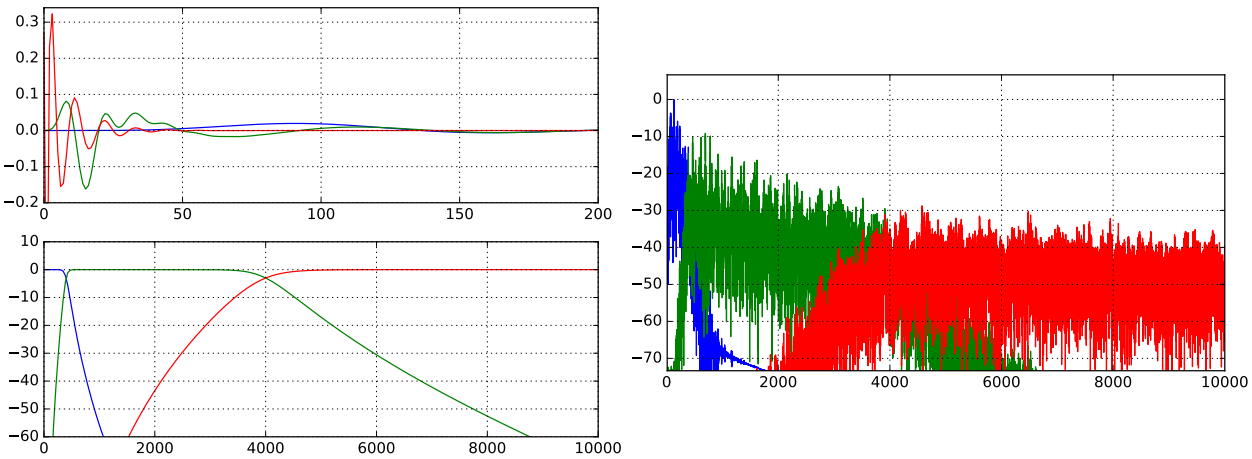
[6] Анг. bass и treble.

за намалување на комплексноста. Притоа, треба да се обрне особено внимание на нелинеарната фазна карактеристика на ИИР филтрите.

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

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

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

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



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

```
# %% засилување
g_bass = -20  # vo dB
g_mid = 0  # vo dB
g_treble = 20  # vo 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 = das.get_spectrum(wav, fs)
f, wav_bass_spec = das.get_spectrum(wav_bass, fs)
f, wav_mid_spec = das.get_spectrum(wav_mid, fs)
f, wav_treble_spec = das.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 одзиви, фреквенциските карактеристики на трите дизајнирани филтри и добиените подопсези од аудиосигналот се прикажани на Сл. 4.5.

4.6 ИИР филтри непропусни на фреквенција – ноч филтри

Една од примените на дигиталните филтри за која неприкосновени се ИИР филтрите е потиснувањето на одредена фреквенција во аудиосигналите. Најчесто потребата за ваков тип на процесирање се јавува кога треба да се потисне хармоничниот шум од градската мрежа на 50 Hz, познат како брум, кој влегол во аудиосигналот пред неговата дигитализација. За таа цел потребно е филтерот непропусник да има што потесен опсег и да биде со што поголемо слабеење,

односно голем Q фактор. Овие типови на филтри се нарекуваат ноч филтри[7].

[7]На англиски *notch* филтер

Поглавје 5

Дигитални аудиоефекти

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

5.1 Дигитални аудиоефекти базирани на процесирање во временски домен

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

Амплитудни аудиоефекти

Најпознати амплитудни аудиоефекти се:

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

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

[1] Wikipedia – Sound effect https://en.wikipedia.org/wiki/Sound_effect
[2] Анг. *fade-in* и *fade-out*.
[3] Анг. *cross-fade*.

Основи на процесирање со задоцнување

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

$$y[n] = x[n] + b_D x[n - D]\,. \tag{5.1}$$

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

$$h[n] = 1 + b_D \delta[n - D]\,, \tag{5.2}$$

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

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. Поради овој облик овие системи се нарекуваат уште и чешлести филтри. Ваквата фреквенциска карактеристика е неповолна за обработка на аудиосигналите поради спектралните изобличувања кои ќе бидат внесени во сигналот.

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

$$y[n] = x[n] + b_{D_0} x[n - D_0] + b_{D_1} x[n - D_1] + \ldots + b_{D_{M-1}} x[n - D_{M-1}]\,, \tag{5.4}$$

$$h[n] = 1 + b_{D_0} \delta[n - D_0] + b_{D_1} \delta[n - D_1] + \ldots + b_{D_{M-1}} \delta[n - D_{M-1}]\,, \tag{5.5}$$

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

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

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

$$y[n] = x[n] - a_D y[n - D] \tag{5.7}$$

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

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

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

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

$$y[n] = b_D x[n] + x[n-D] - b_D y[n-D]\,, \tag{5.9}$$

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

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

Дигитално ехо

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

Најпрвин да ги увеземе потребните модули и пакети.

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

Сега да ги имплементираме овие три типови на системи.

```
fs = 22050

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

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

# бескрајно 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 одзиви на овие четири системи.

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

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

# ИИР
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))

# АР
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))
```

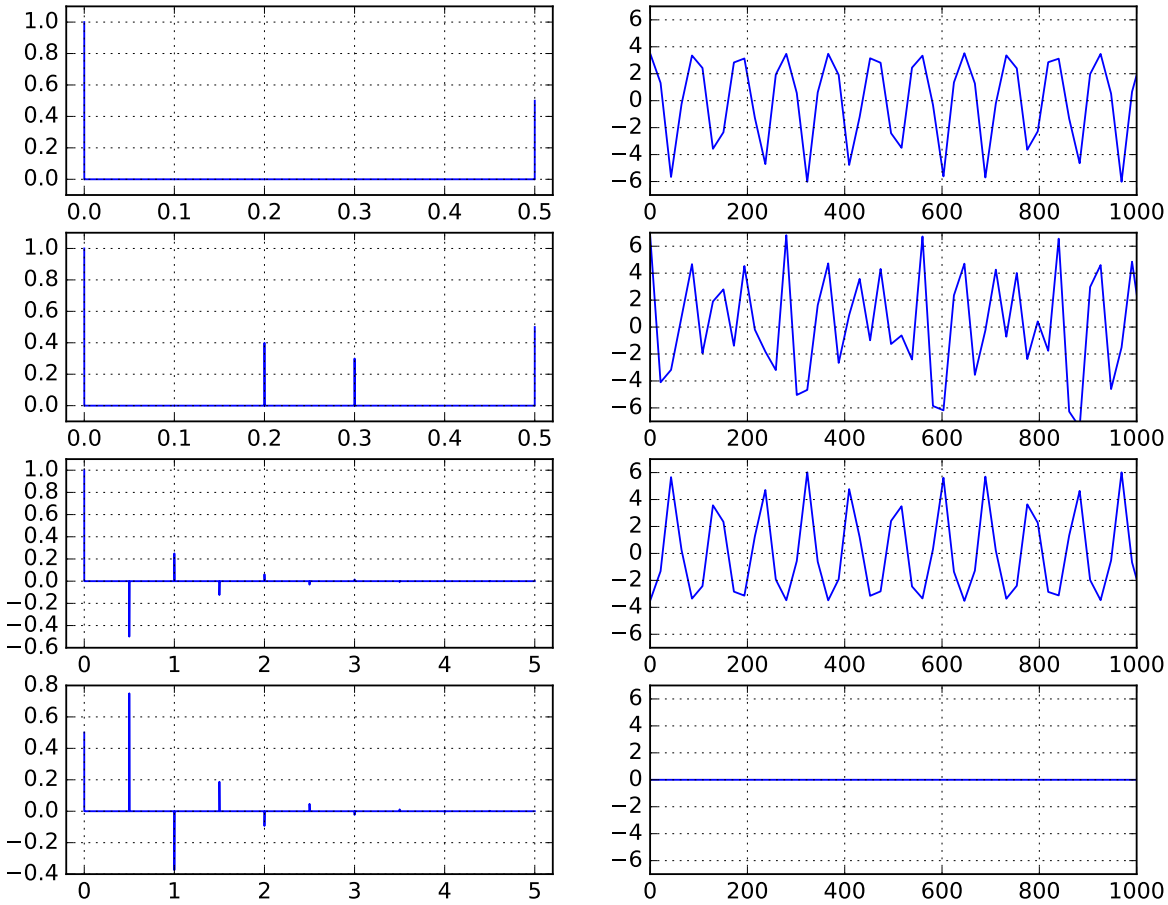
Пресметаните импулсни и фреквенциски одзиви на четирите системи за генерирање на ехо се претставени на Сл. 5.1. Може да се види дека првите три системи навистина имаат чешлеста амплитудна фреквенциска карактеристика, додека филтерот сепропусник има рамна карактеристика со амплитуда од 0 dB.

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

```
#%% филтрирање
fs, wav = wavfile.read('audio/Pato_22K.wav')
wav_eho = sig.lfilter(b_fir, [1], wav)
wav_eho = sig.lfilter(b_fir_mul, [1], wav)
wav_eho_iir = sig.lfilter([1], a_iir, wav)
wav_eho_ap = sig.lfilter(b_ap, a_ap, wav)

#%% преслушување
os.system('play audio/Pato_22K.wav')
wavfile.write('audio/Pato_eho_fir.wav', fs, np.array(wav_eho, dtype='int16'))
os.system('play audio/Pato_eho_fir.wav')
wavfile.write('audio/Pato_eho_fir_mul.wav', fs, np.array(wav_eho, dtype='int16'))
os.system('play audio/Pato_eho_fir_mul.wav')
wavfile.write('audio/Pato_eho_iir.wav', fs, np.array(wav_eho, dtype='int16'))
os.system('play audio/Pato_eho_iir.wav')
wavfile.write('audio/Pato_eho_ap.wav', fs, np.array(wav_eho, dtype='int16'))
os.system('play audio/Pato_eho_ap.wav')
```

Дигитална реверберација

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



Сл. 5.1: Импулсни и фреквенциски одѕиви на четирите системи за генерирање на ехо (од горе надолу): ФИР филтер – единечно ехо, ФИР филтер – повеќекратно ехо, ИИР филтер – бесконечно ехо и филтер сепропусник.

Други дигитални ефекти базирани на доцнење

Други аудиоефекти базирани на употреба на доцнење или сепропусни филтри се: хор, фленџ и фејзер ефектите. Ефектот хор[4] е ефект во кој на аудиосигналот му се додаваат една или повеќе негови верзии добиени со променливо но мало задоцнување. На тој начин се постигнува впечаток дека наместо еден музички извор, во аудиосигналот постојат повеќе извори кои се релативно добро усогласени, но на моменти приметно разгодени.

Фленџ[5] ефектот се добива преку примена на ФИР систем за еднократно ехо кое има променливо доцнење. Аудитивно овој ефект е сличен на фејзерот опишан во поглавјето , кој пак може да се реализира со каскада од променливи сепропусни филтри. Разликата меѓу двата ефекта е во тоа што фленџот се базира на чешлестиот облик на фреквенциската карактеристика на ФИР филтерот, па генерира максимуми и минимуми во излезниот сигнал кои се во хармониски сооднос, односно се мултипли од основниот максимум. Од друга страна, ова не е случај кај фејзерот.

5.2 Дигитални аудиоефекти базирани на процесирање во фреквенциски домен

[4]Wikipedia: Chorus effect. https://en.wikipedia.org/wiki/Chorus_effect
[5]Wikipedia: Flanging. https://en.wikipedia.org/wiki/Flanging

Дигитални аудиоефекти базирани на филтри

Освен еквализаторите, кои исто така можат да се сметаат за дигитални аудиоефекти, најпознатите дигитални аудиоефекти базирани на филтри се ва-ва и фејзерот. Ва-ва ефектот[6] е првенствено направен за изведба на музика на електрична гитара, иако за првпат го пронашле трубачите и тромбонистите во 1920^те. Тој се изведува со помош на филтер пропусник на опсег чија централна фреквенција се менува со време а под контрола на свирачот преку потенциометар поставен во педала за дозирање. Аудиосигналот кој филтерот го дава на излез се меша со оригиналниот сигнал за да се добие крајниот ефект. Во електронската музика педалата може да биде заменета од нискофреквенциски осцилатор (НФО).

Фејзерот[7] исто така претставува гитарски ефект добиен со употреба на каскада на notch филтри чии што фреквенции на потиснување периодично се менуваат. На овој начин тие вршат селективно слабеење на делови од спектарот на сигналот кое може да се види како траг во спектрограмот. Повторно менувањето на фреквенцијата на овие филтри може да биде направена од свирачот преку педала со потенциометар или со употреба на НФО. Дополнително фејзерот може да се реализира и со сепропусни филтри дискутирани во поглавјето .

[6] Wikipedia: Wah-wah (music). https://en.wikipedia.org/wiki/Wah-wah_%28music%29
[7] Wikipedia: Phaser (effect). https://en.wikipedia.org/wiki/Phaser_%28effect%29

Поглавје 6

Компресија на аудиосигналите

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

- методи без загуби[1] и
- методи со загуби.[2]

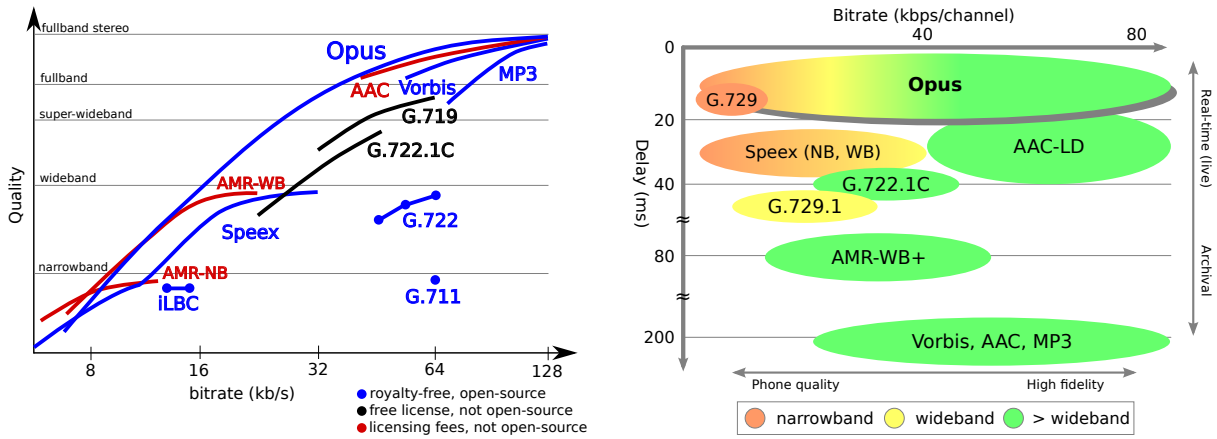
Методите за компресија без загуби не исфрлаат никакви информации од оригиналното аудио. Овие методи не се многу различни од општите методи за компресија. Сепак тие успеваат да ја намалат големината на звучните записи само на 50 – 60% од онаа на оригиналната датотека. Ова се должи на комплексноста на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

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

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

Најдобриот алгоритам за компресија на аудио со загуби во моментов е слободниот Опус[4] формат. Тој не само што постигнува подобар субјективен квалитет, оценет преку двоен слеп тест, од сите останати формати за сите излезни битски брзини, туку и постигнува најдобри перформанси во

[1] Анг. *lossless*.
[2] Анг. *lossy*.
[3] Free Lossless Audio Codec (FLAC) https://xiph.org/flac/
[4] Wikipedia – Opus (audio format) https://en.wikipedia.org/wiki/Opus_(audio_format)



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

однос на латентноста со стандардно доцнење од 26,5 ms кое овозможува комуникација во реално време, споредено со 200 ms на мп3 стандардот. Со жртвување на излезниот битски проток ова време може да се намали до 5 ms. Исто така, Опус нуди оптимизација за имплементација во вгнездените компјутерски системи. Конечно, Опус е слободен стандард и сите патенти кои го покриваат алгоритамот се бесплатни. Перформансите на Опус споредени со останатите алгоритми за компресија со загуби се дадени на Сл. 6.1.

Опус се базира и го заменува слободниот формат за компресија на музички сигнали ворбис (OGG)[5]. Тој исто така сосема го заменува слободниот Спикс[6] формат, наменет за компресија на говор. Компресијата на музичките сигнали во Опус се базира на слободниот ЦЕЛТ[7] алгоритам, кој пак е базиран на модифицираната дискретна косинусна трансформација (МДКТ). Компресијата на говорните сигнали пак, се базира на слободниот формат за компресија на говорни сигнали Силк[8].

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

6.1 MPEG-1 ниво III

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

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

- да објавува технички резултати и извештаи поврзани со компресија на аудио и видео,

[5]Vorbis audio compression. https://xiph.org/vorbis/
[6]Wikipedia – Speex https://en.wikipedia.org/wiki/Speex
[7]Wikipedia – Constrained Energy Lapped Transform (CELT) https://en.wikipedia.org/wiki/CELT
[8]Wikipedia – SILK https://en.wikipedia.org/wiki/SILK

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

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

- MPEG-1,
- MPEG-2 BC (Backwards-Compatible),
- MPEG-2 NBC/AAC (Non-Backward Compatible/Advanced Audio Coding),
- MPEG-4 и
- MPEG-7,
- MPEG-21.

Од нив последните два не се стандарди за компресија. MPEG-7 дефинира дескриптори на аудио/видео содржините, со чија помош може да се опишат за побрз пристап до нив во бази на податоци. MPEG-21 дефинира мултимедијална рамка и нуди менаџирање 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 аудио стандардот: MPEG-1, MPEG-2, MPEG-4 итн. Нивоата означуваат фамилии на алгоритми за кодирање внатре во MPEG фазата. Нивоа се дефнирани 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 аудио стандард, објавен од MPEG групата во 1992, по четиригодишна напорна работа на усогласено истражување на светските експерти од областа на аудио компресијата. Неговата намена била да се овозможи стерео-CD-квалитет. MPEG-1 подржува стерео аудио CD квалитет на 192 kbit/s. Тоа го достигнува со примена на флексибилна техника за хибридно компресија на аудиото која се базира на сплет од неколку методи и тоа:

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

MPEG-1 аудио кодекот работи со 16-битен ИКМ влез со fs од 32, 44,1 и 48 kHz. Тој нуди различни модови на работа за моно, стерео, двојно моно и заедничко (joint) стерео. Протоците на компресираното аудио се дефинирани во опсег 32 – 192 kbit/s за моно и 64 – 384 kbit/s за стерео.

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

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

Целта на психоакустичката анализа е определување на праговите на приметлива дисторзија JND (Just Noticeable Distortion) за различните подопсези. Тие ја одредуваат максималната грешка на квантизација која може да си ја дозволи кодерот при распределбата на расположливиот број на битови помеѓу подопсезите. При динамичкото доделување на битови, приоритет (највеќе битови) ќе добијат подопсезите со најниски JND. Нивната вредност зависи од два параметри прикажани на Сл. 6.2:

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

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

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

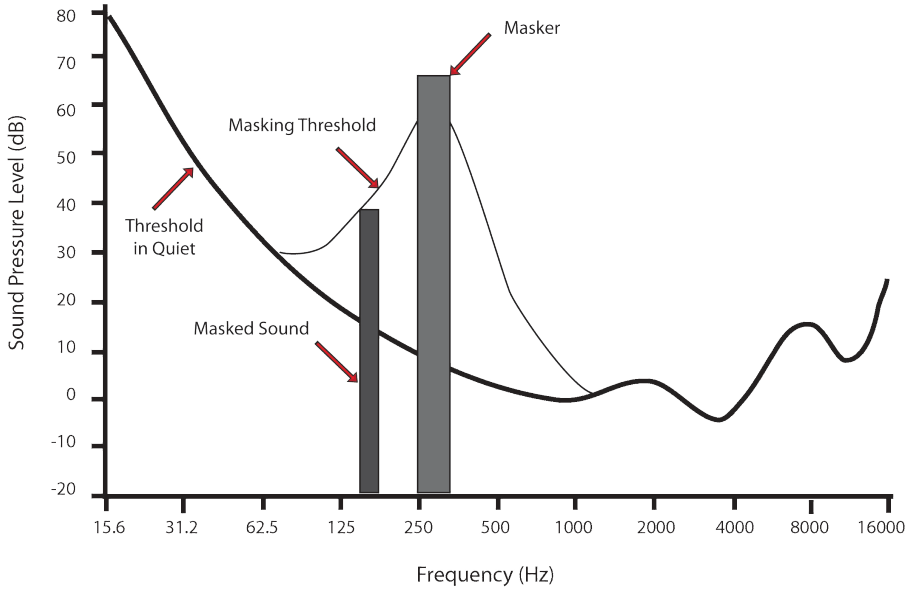
На крајот започнува итеративна процедура на доделување на битови која ги користи пресметаните JND прагови за секој подопсег за одбирање на оптимален кватизер за тој подопсег (од дадено почетно множество на квантизери). Изборот на квантизерите се прави така што двата критериуми – зададениот краен проток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

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

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

[9] Wikipedia: Psychoacoustics – Masking effects: https://en.wikipedia.org/wiki/Psychoacoustics#Masking_effects

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



Сл. 6.2: Прагот на чујност и појавата на фреквенциско маскирање на кои се базира принципот на работа на психоакустичката аудиокомпресија.[11]

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

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

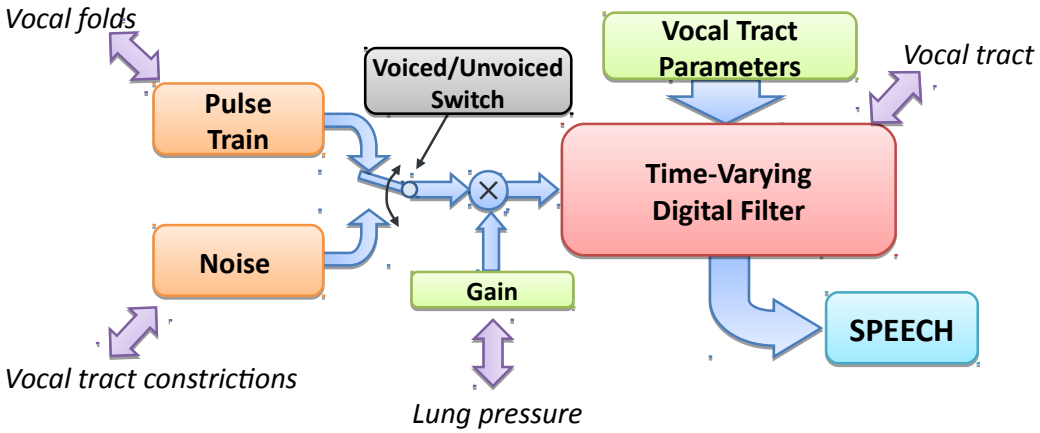
На крајот, нивото III користи софистицирано доделување на битови и стратегии на квантизација кои се базираат на:

- неуниформна квантизација,
- анализа преку синтеза и
- ентрописко кодирање.

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

6.2 Компресија на говор

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



Сл. 6.3: Моделот извор-филтер на принципот на создавање на говорот.

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

LPC се базира на моделот извор-филтер на создавање на говорот прикажан на Сл. 6.3. Во овој модел изворот го претставуваат неколку модули кои ги моделираат побудните механизми на човековиот говорен апарат и тоа:

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

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

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

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

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

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

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

- звучност – дали се работи за звучен или безвучен глас,

- засилување – енергијата на сигналот во рамката,
- коефициенти на филтерот – кои соодветствуваат на преносната карактеристика на вокалниот тракт, и
- висина – периода на основниот хармоник.

Најосновната имплементација на LPC кодерот нуди одлична разбирливост на кодираниот говор. Но, кавлитетот на излезниот говор не е на високо ниво и има каркатеристичен роботски призвук. Federal Standard (FS) 1015 кодекот од 1982 г. е првиот кодер базиран на LPC. Протокот кој тој го остварува е 2,4 kbit/s, што во споредба со вообичаениот проток на дигиталниот говор во телефонијата од 64 kbit/s[12], претставува компресија од 26 пати. Бидејќи филтерот кој го моделира вокалниот тракт е со ред 10, кодекот се нарекува и LPC-10. Ако не го знаете моделот, параметрите кои се испраќаат, немаат никакво значење. Поради тоа, FS 1015 кодекот е направен за американската војска, за потоа да го присвои и НАТО.

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

[12] 8 kHz · 8 bit = 64 kbit/s

Поглавје 7

Линеарна предикција

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

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

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

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

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

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

Преносната функција на овој систем во z-домен е:

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

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

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

можеме да видиме дека ако е задоволен условот:

$$\alpha_k = a_k \,, \quad \text{за } k = 1,2,...p \,, \tag{7.6}$$

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

$$e[n] = Gx[n] \,, \tag{7.7}$$

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

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

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

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

7.1 Анализа и синтеза на глас со линеарна предикција

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 с'ржта на нејзината примена за компресија на говорните сигнали. За целите на оваа анализа направете снимка од сопствениот глас како ги изговарате петте самогласки во македонсиот јазик. Тоа ќе го направиме со помош на Аудасити.[1]

Моделирање на гласот /а/

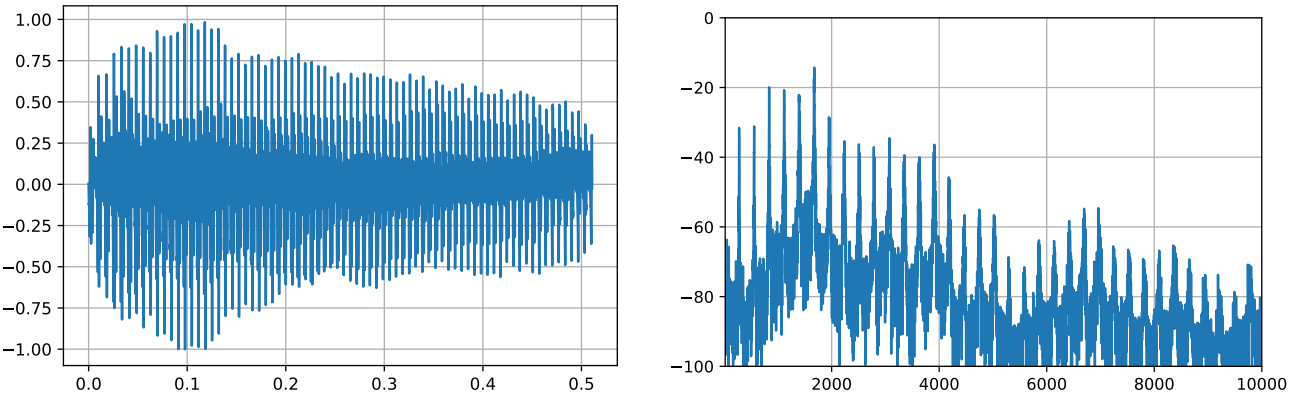
Во нашата анализа ќе ја искористиме имплементацијата на линеарната предикција, поточно алгоритамот на Дурбин за линеарна предикција која е содржана во модулот `scikit.talkbox`[2]. Овој модул содржи некои основни функционалности за обработка на говор или за аудиосигналите поопшто и може едноставно да се инсталира користејќи го `pip`:

```
$ sudo pip install scikits.talkbox
```

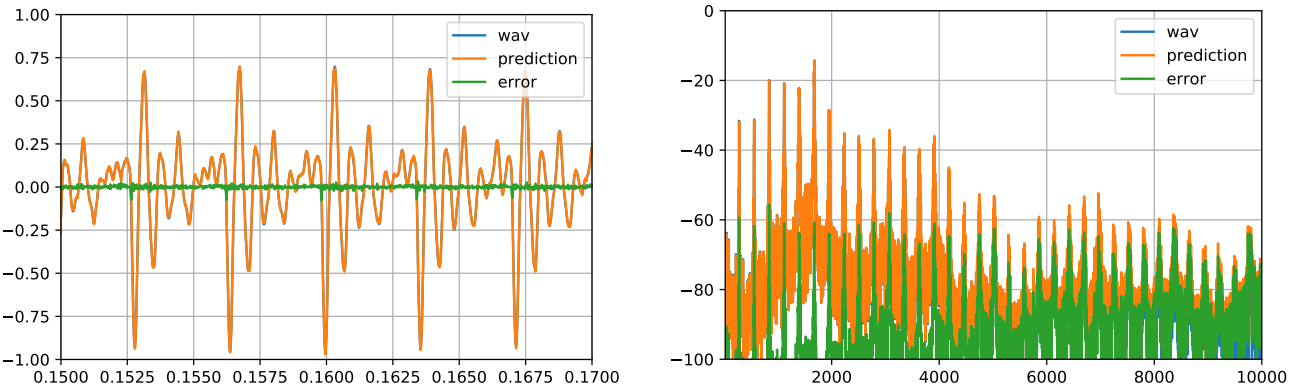
Во кодов што следи ќе го вчитаме аудио записот со самогласката /а/ и ќе го прикажеме неговиот временски и спектрален облик, Сл. 7.1.

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

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

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

# %% исцртај спектрален домен
f, wav_spec = das.get_spectrum(fs, wav)
```

[1] За Аудасити види повеќе во Поглавјето .
[2] scikits.talkbox http://www.scikits.appspot.com/talkbox



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



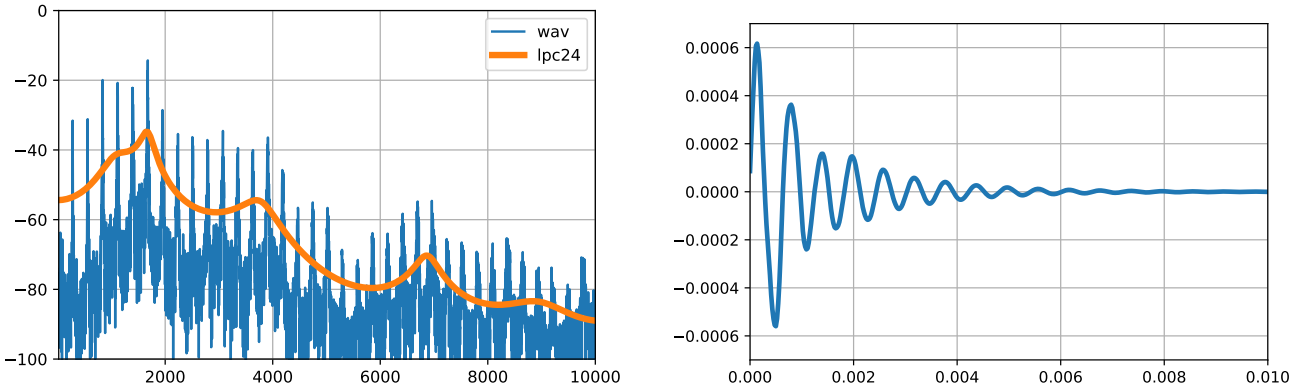
Сл. 7.2: Оригиналниот аудиосигнал, неговата предикција и сигналот на грешка во временски (лево) и во спектрален (десно) домен.

```
plt.figure()
plt.plot(f, wav_spec)
plt.xscale('log')
plt.grid('on')
plt.axis([0, 2e4, -100, 0])
```

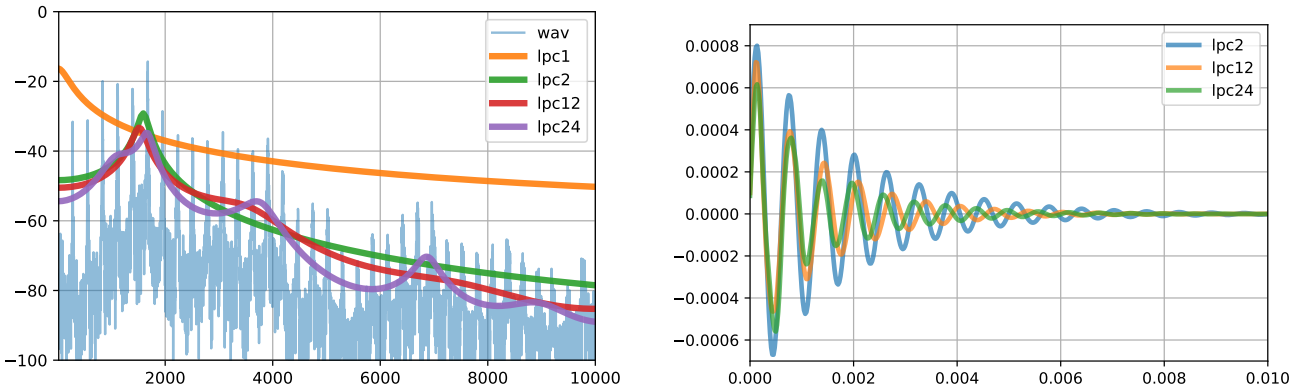
Во спектралниот облик на сигналот може да се видат хармониците кои се должат на периодичноста на сигналот, како и анвелопата на спектарот која има изразени врвови. Овие врвови одговараат на резонантните фреквенции на вокалниот тракт при изговор на гласот /a/. Кога се работи за самогласки ти се нарекуваат форманти. Сега ќе ги најдеме коефициентите на филтерот кои треба да ја опишат анвелопата на сигналот со употреба на функцијата `lpc`.

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

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



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



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

енергијата на сите негови хармоници е речиси иста. Може да се каже дека сигналот на грешка претставува спектрално „обелена“ верзија на оригиналниот сигнал.[3]

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

```
G = err
w, H_filt = sig.freqz(G, a_filt)
f_H = w / np.pi * fs/2
H_filt = 20*np.log10(np.abs(H_filt))
plt.figure()
plt.plot(f, wav_spec)
plt.plot(f_H, H_filt, lw=4)
```

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

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

На Сл. 7.4 е дадена споредба на фреквенциските карактеристики и импулсните одзиви на филтри за линеарна предикција со различен ред. Може да се види дека филтер од прв ред го доловува

[3]На англиски процесот на „белење“ на еден сигнал се нарекува „whitening“.

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 го зголемуваме бројот на коефициенти во филтерот така сè подобро тие ја претставуваат спектралната анвелопа на аудиосигналот. Едно непишано правило е дека при моделирање на говор бројот на коефициенти на филтерот треба да е еднаков на бројот на форманти, т.е. резонантни фреквенции, плус 2 пола за нулите. Вообичаено се зема дека во секој kHz од спектарот на сигналот има по еден формант, па од таму оптималниот ред е широчината на сигналот во kHz плус 2. Во нашиот случај за f_s од 44,1 kHz тоа значи дека оптималниот ред на филтерот би бил 22.

Поглавје 8

Примена на машинското учење на аудиосигналите

Машинското учење, а особено длабокото учење, е област која започнува да доминира во голем број на инженерски и научни области. Во дигиталните аудиосистеми, машинското учење не само што донело решенија за проблемите за кои немало решение или за кои решението било прекомплексно и прескапо, на пр. системи базирани на експертски правила, тоа во некои области понудило и перформанси блиску до човечките.

8.1 Основни парадигми во машинското учење

Машинското учење е подобласт од областа вештачка интелигенција. Тоа опфаќа модели чии параметри се адаптираат на множество на податоци низ процес наречен на тренирање, односно учење (Bishop, 2006; Müller et al., 2016). На највисоко ниво разликуваме три парадигми на машинско учење, според кои ги делиме моделите од оваа област на три категории:

- учење со надзор[1],
- учење без надзор[2], и
- насочено учење[3].

Учење со надзор

Кај учењето со надзор моделите се тренираат со познати целни излезни секвенци, уште наречени основна вистина[4]. Според типот на податоци во излезните секвенци разликуваме два генерални типови на модели и тоа модели за:

- класификација – ако излезните секвенци се дискретни па може да се каже дека одговараат на затворено множество на класи или категории, и
- регресија – ако излезните секвенци се континуирани.

Овие два тима на модели вообичаено ја имаат истата структура; клучната разлика е во типот на излезот кој го даваат.

Примери за употреба на модели за класификација во дигиталните аудиосистеми се алгоритмите за: препознавање на говор, препознавање на говорник, препознавање на јазик, препознавање на музички инструмент, и препознавање на аудиосцена. Од друга страна, примери за модели

[1] Анг. *supervised learning.*
[2] Анг. *unsupervised learning.*
[3] Анг. *reinforcement learning.*
[4] Анг. *ground truth.*

за регресија се алгоритмите за: синтеза на интонација, спектар и временски аудиосигнал во системите за синтеза на говор, итн.

Учење без надзор

Алгоритмите за учење без надзор се од посебно значење денес, поради широката достапност на големи множества на податоци за кои нема ознаки, односно целни секвенци. Ова е така поради тоа што процесот на анотација вообичаено се базира на рачно внесување на ознаки од луѓе експерти, што целиот процес го прави временски захтевен и скап. Најпознатите претставници на алгоритмите за учење без надзор се алгоритмите за:

- кластерирање – кои вршат групирање на влезните податоци во одреден број на класи кој може да биде внесен од корисникот или одреден автоматски. Најпознат алгоритам за оваа намена е алгоритамот на К-средни вредности[5] кој го дели множеството податоци во K кластери опишани со средната вредност на примероците во секој кластер, наречени и центроиди, и
- намалување на димензионалноста – служат за визуелизација на распределбата на едно множество на повеќедимензионални податоци во две или три димензии преку нивна трансформација. Анализата на принципиелни (главни) компоненти (ПЦА)[6] е еден пример за егзактна математичка метода за оваа намена, која е и реверзибилна. Тука спаѓаат стохастични алгоритми базирани на ПЦА, а најпознат алгоритам за оваа намена е алгоритамот за т-дистрибуираното стохастичко врамување на соседи (т-СНЕ)[7].

Насочено учење

Во оваа категорија спаѓаат алгоритмите кои вообичаено се поистоветуваат со областа вештачка интелигенција. Тие имаат крајна цел, но таа не е доволна за директно тренирање на моделот. Тука спаѓаат алгоритмите за играње на компјутерски игри кои скоро ги надиграа најдобрите човечки играчи[8], но и алгоритмите за контрола на агенти во виртуелни симулации кои треба да научат некоја задача, на пример како оптимално да го придвижат човечкото тело за да стигнат од точка А до точка Б[9].

Крајната цел на овие алгоритми е да победат, но таа е предалеку во иднината за директно да може да се искористи во учењето. За надминување на овој проблем, се прават стратегии за наградување на алгоритамот долж одвивањето на играта, со цел учењето да се насочи во правата насока. На пример, влезот на алгоритамот може да биде моменталната состојба на полето за игра, а тој треба на излез да го даде следниот потег кој би го довел во подобра позиција, односно со поголеми шанси да победи. Овие алгоритми вообичаено треба да балансираат помеѓу истражување и искористување, или експлорација и експлоатација. Тренирањето на моделите со насочено учење се одвива преку методата на залудни обиди, односно алгоритамот се пушта да игра самиот против себе милиони пати се додека не научи да победува.

8.2 Вообичаени чекори во примената на машинското учење

Постојат низа од чекори кои вообичаено се остваруваат во употребата на алгоритмите за машинско учење при решавањето на даден проблем:

- анализа на влезните податоци / сигнали / слики – во оваа фаза се анализира типот на влезните податоци, се одредува дали постојат некомплетни податоци, се прави

[5] Анг. *K-means.*
[6] Анг. *Principal Component Analysis (PCA).*
[7] Анг. *t-distributed Stochastic Neighbour Embedding (t-SNE).*
[8] Google DeepMind: Ground-breaking AlphaGo masters the game of Go https://www.youtube.com/watch?v=SUbqykXVx0A
[9] Google's DeepMind AI Just Taught Itself To Walk https://www.youtube.com/watch?v=gn4nRCC9TwQ

визуелизација на нивната распределба, и статистиката на нивните карактеристики,

- поделба на множества за тренирање, валидација и тестирање – дел од податоците се одвојува за конечна оценка на истренираниот модел, додека дел од остатокот се одвојува за континуирана валидација во фазата на тренинг,
- претпроцесирање и екстракција на карактеристики – во оваа фаза се прави скалирање или нормализација на влезните податоци, притоа треба да се води особена сметка на поделбата на податоците на множества за тренирање/валидација/тестирање. Во случај на употреба на влезни податоци со голем број на корелирани димензии, како и во случај кога се работи со аудиосигнали, вообичаено во оваа фаза се врши екстракција на најзначајните карактеристики[10]. За ова може да се применат методи како ПЦА, пресметување на спектар, спектрограми, итн. На пример, за препознавањето на говор најпознатите карактеристики се мел-Фреквенциските кепстрални коефициенти (МФЦЦ),[11]
- тренирање на моделот – оптимизација на параметрите на моделот со употреба на множеството за тренирање и валидација, види Поглавје 8.3,
- конечна евалуација на моделот – ова се прави со одвоеното множество за тестирање со цел да се оцени како работи истренираниот модел за множество од невидени влезни податоци.

8.3 Тренирање на моделите за машинско учење

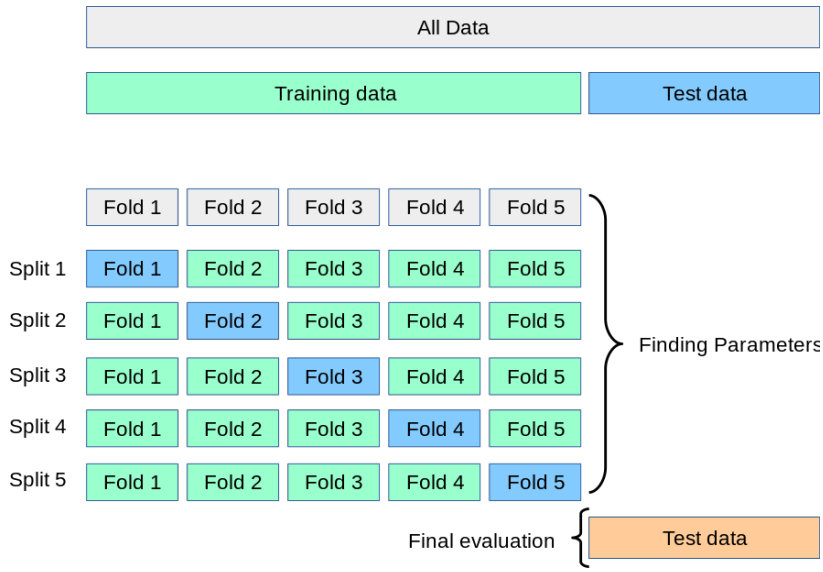
Тренирањето на моделот, односно нагодувањето на неговите параметри се врши со помош на множеството на тренирање. Овој процес се изведува итеративно. Овде ќе претпоставиме дека се работи за алгоритам за учење со надзор, за кој се познати точните вредности кои сакаме да ги добиеме на неговиот излез. Во секоја итерација на учењето, се пресметува грешката која моделот ја прави врз база на познатите излези и неговите параметри се адаптираат во насока на намалување на оваа грешка. Чекорот со кој се врши адаптацијата на параметрите се нарекува и чекор на учење[12].

Тренирањето вообичаено завршува кога грешката на моделот не се подобрува за одредено ниво на толеранција последователно во избран број на епохи или ако е постигнат максималниот дозволен број на итерации. Ако тренирањето запре поради постигнување на максималниот број на итерации а не дојде до заситување на опаѓањето на грешката на моделот, тогаш велиме дека настанало подтренирање[13], односно дека алгоритамот има потенцијал да се дотренира за подобрување на неговите перформанси.

Проблемот со овој начин на тренинг 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 невидени влезни податоци. Оваа појава се нарекува надтренирање[14] и претставува посебен проблем во тренирањето на моделите за машинско учење. Постојат три главни начини тоа да се избегне:

- ограничување на моќта на моделот – поедноставни модели не можат апсолутно да опишат покомплексни влезни податоци и сигнали,
- регуларизација – низа на методи кои ја ограничуваат способноста на моделите да опишат комплексни податоци без намалување на нивната моќ. Степенот на регуларизација се нагодува преку коефициентот на регуларизација,

[10] Анг. *feature extraction.*
[11] Анг. *mel-frequency cepstral coefficients (MFCC).*
[12] Анг. *learning rate.*
[13] Анг. *underfitting.*
[14] Анг. *overfitting.*



Сл. 8.1: K-кратна меѓувалидација за $K = 5$.[18]

- рано запирање[15] – постапка која ја евалуира грешката на моделот на множеството за валидација секоја итерација и го запира тренирањето кога таа ќе започне да расте.

Освен за рано запирање, множеството за валидација служи и за избор и оптимизација на глобалните параметри на моделот и процесот на учење, како на пример: архитектурата на моделот, т.е. бројот на неговите параметри, чекорот на учење, коефициентот на регуларизација итн. Овие глобални, суштински параметри се нарекуваат хиперпараметри.

Изборот на хиперпараметри се прави врз основа на оптимизирање на перформансите на моделот на множеството за валидација. Притоа, за да се избегне влијанието на изборот на подмножество од влезните податоци на овој процес, што е од посебно значење кога располагаме со мало количество на влезни податоци, вообичаено се прави оптимизација на хиперпараметрите во јамка за меѓувалидација[16]. Тоа се изведува на следниот начин, се дели влезното множество на податоци на множества за тренирање и тестирање. Множеството на тренирање потоа се дели на K подмножества од кои секое се зема за множество за валидација во една итерација на јамката за меѓувалидација, а останатите се употребуваат за тренирање, како што е прикажано на Сл. 8.1. Ова се нарекува K-кратна меѓувалидација[17].

Постојат низа на проблеми кои можат да настанат при употреба на овој едноставен пристап за меѓувалидација. Секој од нив решава со адаптација на изборот на примероци во подмножествата за меѓувалидација:[19]

- ако датасетот е подреден по класите кои треба да се препознаат тогаш може да се случи некоја од класите да ја има само во тест множеството; поради ова вообичаено се употребува случајно мешање[20] на датасетот пред неговото тренирање,
- ако датасетот има групи, на пример субјекти, тогаш може да се случи истиот субјект 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 групи[21]

[15] Анг. *early stopping*.
[16] Анг. *cross-validation*.
[17] Анг. *K-fold cross-validation*.
[18] Превземено од сајтот scikit-learn – 3.1. Cross-validation: evaluating estimator performance `https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation`
[19] scikit-learn – 3.1. Cross-validation: evaluating estimator performance `https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation`
[20] Анг. *shuffle*.
[21] Анг. *group k-fold cross-validation*.

на датасетот пред неговото тренирање,

- 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 оценката; ова се надминува со стратификација на класите по подмножествата, т.н. K-кратна меѓувалидација со стратификација[22], и
- во проблеми со временски серии во кои треба да се предвиди иднината врз база на претходните податоци, едноставната поделба на подмножества може да доведе алгоритамот да искористи информации од иднината за да направи подобра предикција; во тој случај, при оформување на подмножествата треба да се внимава на временската последователност на податоците.

Овие различни пристапи во меѓувалидацијата, како и нивната комбинација, се илустрирани на Сл. 8.2.

По завршување на меѓувалидацијата, моделот со избраните хиперпараметри се тренира повторно врз целото множество на влезни податоци, без множеството за тестирање. Вака трениранот модел се евалуира на тест множеството. И овде, за избегнување на влијанието на изборот на тест множество врз оценката на моделот, што е од посебно значење кога имаме мало множество на податоци, може да се направи меѓуоценка на моделот преку јамка на вгнездена меѓувалидација. Во овој пристап, целото множество на податоци најпрвин се дели на N подмножества од кои секое се користи како тест множество во недворешната јамка, а остатокот се дели на M подмножества во стандардна јамка за меѓувалидација.

8.4 Невронски мрежи

Еден од најпознатите модели за машинско учење, којшто лежи во основата на длабокот учење се вештачките невронски мрежи (НМ). Основната градбена единка на невронските мрежи се софтверските неврони чиј начин на работа е инспириран од физиологијата на невроните опишана во Поглавјето **??**.

Вештачки неврон

Секој неврон има K влезови на кои се носи влезниот вектор $\mathbf{x} = [x_0, x_1, \dots x_{K-1}]$. Влезовите на невронот се скалирани со тежински коефициенти, или тежини[24] w_k, кои можат да бидат и негативни. Активацијата на невронот a се добива преку сумирање на скалираните влезови. Конечно, излезот на невронот y се одредува преку излезната нелинеарност $f(a)$. Притоа, вообичаено се додава и коефициент на поместување[25] кој одговара на прагот на излезната нелинеарност.

$$y = f(a) = f(\sum_{k=0}^{K-1} w_k x_k + b) = f(\mathbf{w}\mathbf{x}^T + b) \tag{8.1}$$

Тука, $\mathbf{w} = [w_0, w_1, \dots w_{K-1}]$ е векторот на тежини на овој неврон. Ако на влез се донесат низа на N примероци од влезните податоци $\mathbf{x}_n$, (8.1) преминува во:

$$y_n = f(a_n) = f(\mathbf{w}\mathbf{x}_n^T + b) \quad \text{за } n = 0, 1, 2, \dots N-1\,, \tag{8.2}$$

$$\mathbf{y} = f(\mathbf{a}) = f(\mathbf{w}\mathbf{X}^T + b)\,, \tag{8.3}$$

каде со $\mathbf{X}$ е означена матрицата по чии редици се поставени влезните примероци $\mathbf{x}_n$.

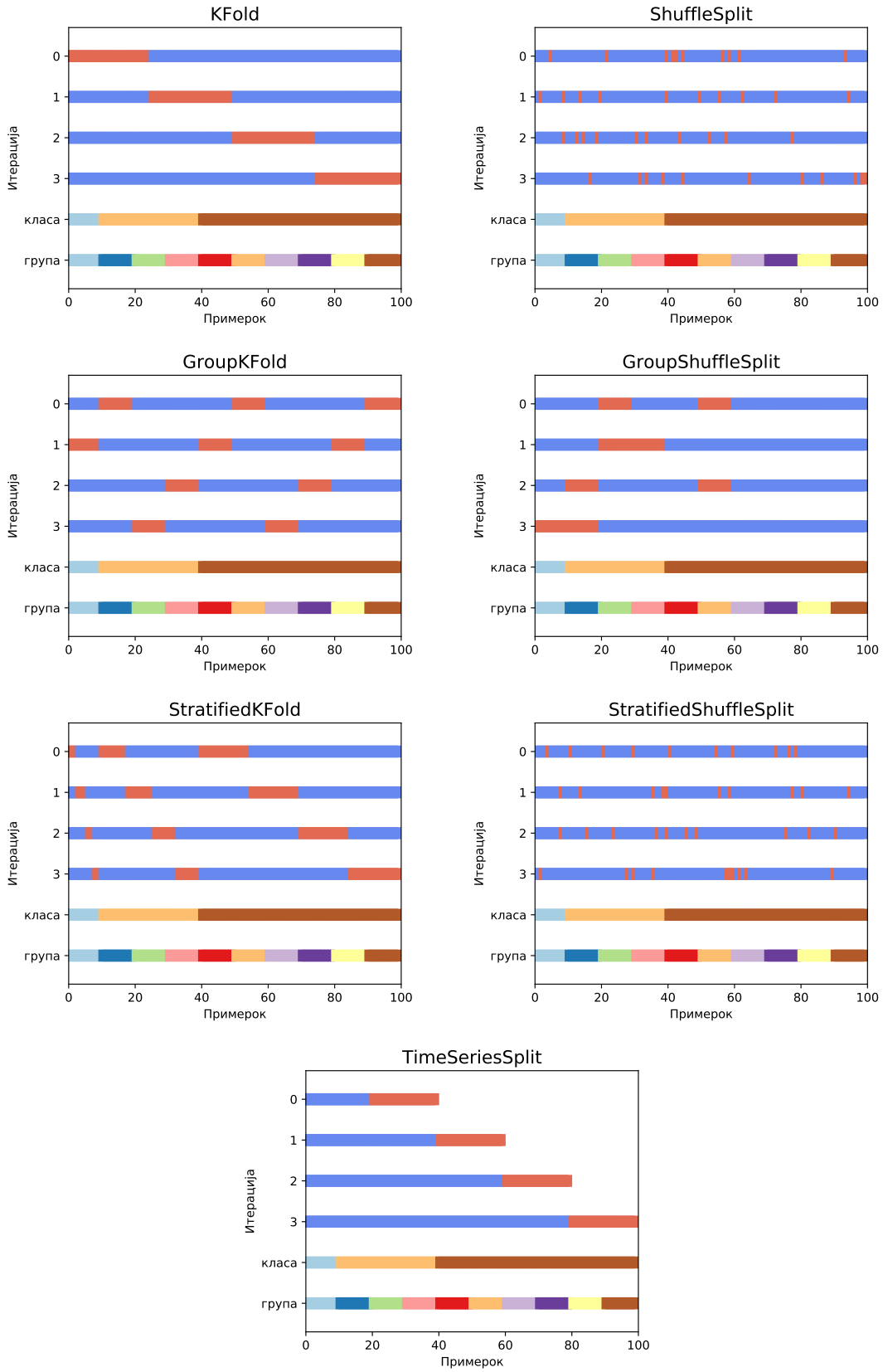
[22] Анг. *stratified k-fold cross-validation.*

[23] Генерирано со адаптиран код од оној на сајтот scikit-learn –Visualizing cross-validation behavior in scikit-learn `https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html`

[24] Анг. *weights.*

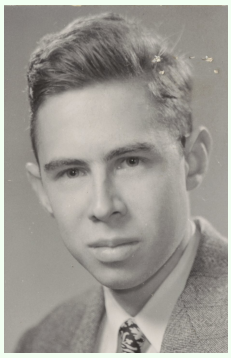
[25] Анг. *bias.*

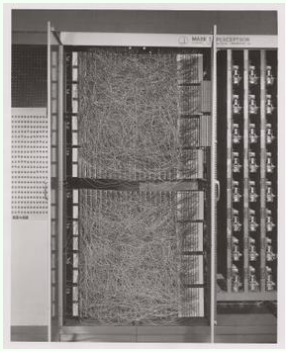
[26] Wikipedia – Perceptron `https://en.wikipedia.org/wiki/Perceptron`



Сл. 8.2: Напредни методи за K-кратна меѓувалидација за $K = 4$. Прикажани се множеството за тренирање (сино) и множеството за тренирање (црвено) за секоја итерација од јамката за меѓувалидација.[23]

[23] Од анонимен извор - http://www.peoples.ru/science/psihology/frank_rosenblatt/, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=64998425





§ Дополнително. Првата имплементација на вештачки неврон, наречена перцептрон била изработена од американскиот психолог Франк Розенблат[26]во 1958 во Лабораторијата за аеронаутика во Корнел, САД. Тој бил најпрвин изведен во софтвер на ИБМ 704, а потоа и во хардвер. Хардверската изведба била направена за препознавање на слики регистрирани со 400 фотоќелии, тежините на влезовите биле реализирани со потенциометри, а нивното учење со вртење на потенциометрите со електрични мотори

Моделите за регресија базирани на употреба на еден неврон се нарекуваат линеарна регресија, додека оние за класификација се нарекуваат логистичка регресија.

Плитки невронски мрежи

Во наједноставниот случај, невронските мрежи имаат еден скриен слој и еден излезен слој на неврони, како што е прикажано на Сл. 8.3. Ваквите модели се нарекуваат плитки невронски мрежи. Секој неврон од скриениот слој е поврзан со секој од коефициентите на влезниот вектор $\mathbf{x}$. Секој неврон од излезниот слој пак е поврзан со секој неврон од скриениот слој. Поради ова поврзување, овој вид на слоеви се нарекуваат и целосно поврзани односно густи[27]. Ова едноставна архитектура сепак им овозможува на плитките невронски мрежи да моделираат било која нелинеарна функција, па тие се уште познати и како универзални апроксиматори.

Излезот на една плитка невронска мрежа за даден влезен вектор на податоци $\mathbf{x}$ може да ја пресметаме преку:

$$\mathbf{y}_h = f_h(\mathbf{a}_h) = f_h(\mathbf{W}_h\mathbf{x}^T + \mathbf{b}_h)\,, \tag{8.4}$$

$$\mathbf{y} = f_o(\mathbf{a}_o) = f_o(\mathbf{W}_o\mathbf{y}_h + \mathbf{b}_o) = f_o(\mathbf{W}_o \cdot f_h(\mathbf{W}_h\mathbf{x}^T + \mathbf{b}_h) + \mathbf{b}_o)\,, \tag{8.5}$$

каде со $_h$ се означени параметрите и излезите добиени од скриениот слој, а со $_o$ оние од излезниот слој. Овојпат, бидејќи во секој слој може да имаме повеќе неврони, нивните тежини се распоредени долж редиците на матриците за тежини $\mathbf{W}$ а нивните коефициенти на поместување во векторите колони $\mathbf{b}$.

На тој начин, излезот на мрежата се добива со процесирање на влезните податоци слој по слој се додека не се дојде до излезниот слој на мрежата.[28]. Овој процес се нарекува и пропагација нанапред.[29]

Повторно, ако на влез се донесат низа на N примероци од влезните податоци $\mathbf{x}_n$ добиваме:

$$\mathbf{Y}_h = f_h(\mathbf{A}_h) = f_h(\mathbf{W}_h\mathbf{X}^T + \mathbf{b}_h)\,, \tag{8.6}$$

$$\mathbf{Y} = f_o(\mathbf{A}_o) = f_o(\mathbf{W}_o\mathbf{Y}_h + \mathbf{b}_o) = f_o(\mathbf{W}_o \cdot f_h(\mathbf{W}_h\mathbf{X}^T + \mathbf{b}_h) + \mathbf{b}_o)\,, \tag{8.7}$$

Во рамките на една невронска мрежа, излезните нелинеарности на невроните во скриените и излезните слоеви вообичаено се разликуваат. Типичен избор за нелинеарности во скриениот слој се:

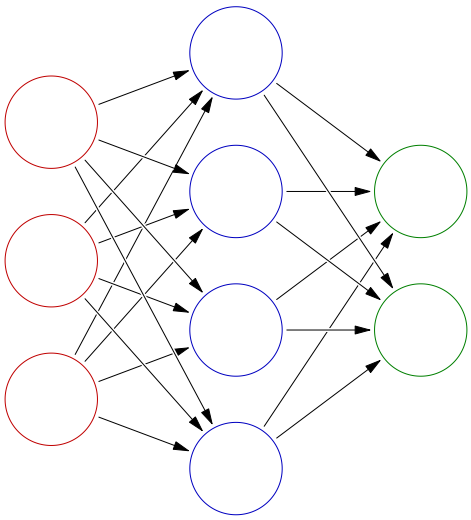
- сигмоида $\sigma(a) = \frac{1}{1+e^{-a}}$ – дава излез во опсег 0 – 1,

Од Source (WP:NFCC#4), Fair use, https://en.wikipedia.org/w/index.php?curid=47541432

[27]Анг. *fully connected* и *dense*.

[28]Длабоките невронски мрежи имаат повеќе скриени слоеви, види понатаму во Поглавјето

[29]Анг. *forward pass* или *feed forward*.



Сл. 8.3: Плитка невронска мрежа со еден скриен слој составен од четири неврони (сино), и еден излезен слој составен од 2 неврони (зелено). Влезните податоци вообичаено се цртаат како влезен слој (црвено).[30]

- тангенс хиперболикум $\tanh(a) = \frac{e^{2a}-1}{e^{2a}+1}$ – дава излез во опсег од -1 до 1, и
- полубранов насочувач $ReLU(a) = \begin{cases} a & \text{ако } a > 0 \\ 0 & \text{поинаку} \end{cases}$ – предноста на употреба на полубрановиот насочувач е поедноставната пресметка на излезот на невроните, подобрата пропагација на градиентот во процесот на тренирање, како и реткоста на активација на невроните – при случајна иницијализација на тежините половина од невроните ќе дадат 0. Уште една мотивација за употреба на оваа нелинеарност е биолошката аналогија која се должи на нејзината асиметричност.

За невроните во излезниот слој вообичаено се користат:

- сигмоида – за класификација,
- софтмакс $f(a_j) = \frac{e^{a_j}}{\sum_{j=0}^{J-1} e^{a_j}}$ – за класификација со повеќе излезни класи J, каде a_j е активацијата на невронот кој соодветствува на класата j; софтмакс функцијата го нормализира збирот на излезот на сите излезни неврони, па може да се каже дека ни дава апроксимација на веројатноста на секоја од класите $f(a_j) \approx P(y = j \mid \mathbf{a})$,
- линеарна $f(a) = a$ – кај моделите за регресија.

Длабоко учење

Со додавање на повеќе скриени слоеви во невронската мрежа се добиваат длабоки невронски мрежи (ДНН)[31] Длабокото учење е подобласт во машинското учење која ги опфаќа моделите базирани на повеќеслојни вештачки невронски мрежи (Goodfellow et al., 2016). Иако плитките невронски мрежи претставуваат универзални апроксиматори, тоа важи само кога моќта на мрежата, одредена од бројот на неврони во скриениот слој, е доволно голема. Се покажува дека додавање на неврони во скриениот слој, односно додавање на широчина на мрежата не е толку ефикасно како додавање на длабочина на мрежата.

Невронските мрежи кои постигнуваат денес надчовечки перформанси имаат и до 1000 скриени слоеви (He et al., 2016). Поради комплексноста на тренирањето на милионите параметри на длабоките невронски мрежи, тие доживуваат процут дури на почетокот од XXI век, по

[30] Модифицирано 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

[31] Анг. *Deep Neural Networks (DNNs).*

општата достапност на графичките процесори кои овозможуваат паралелна имплементација на алгоритмите за тренинг.

Постојат три основни подвидови на длабоките невронски мрежи:

- обични длабоки невронски мрежи – овие се нарекуваат и повеќенивовски перцептрони (МЛП)[32] во чест на првиот перцептрон,
- конволуциски невронски мрежи (КНМ) – 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 (РНМ) – базирани на повратна врска на невроните од скриените слоеви што им дава еден вид на меморија за претходните примероци, се употребуваат за процесирање на 1Д сигнали,
- рекурентни конволуциски невронски мрежи (РКНМ) – комбинација од РНМ која го употребува излезот од КНМ, се употребуваат за процесирање на 4Д сигнали, пред сè видео.

Тренирање на невронските мрежи

Проблемот на тренирање на невронските мрежи се сведува на промена на тежините и коефициентите на поместување на секој од невроните во насока на намалување на грешката која ја прави мрежата (Nielsen, 2015). Еден начин тоа да се направи е со примена на алгоритамот спуштање по градиентот (ГД)[33], кој е итеративен алгоритам за пронаоѓање на минимумот на дадена функција. Конкретно, за да дојдеме до минимумот на функцијата $f(x)$ од моментната позиција x_i, треба да направиме чекор во насока спротивна на градиентот за таа позиција:

$$x^{i+1} = x^i - \frac{\mathrm{d}f}{\mathrm{d}x} \cdot \Delta x \,, \tag{8.8}$$

каде со Δx е означен големината на чекорот кој го земаме.

За примена на ГД при тренирањето на невронските мрежи дефинираме функција на грешка $\mathcal{L}(y, \tilde{y}) = \mathcal{L}(y, g(\boldsymbol{\theta}, \mathbf{x}))$ која зависи од точниот, т.е. целниот излез y и излезот добиен од мрежата $\tilde{y} = g(\boldsymbol{\theta}, \mathbf{x})$, кој пак зависи од параметрите на невронската мрежа $\boldsymbol{\theta}$ и влезниот вектор $\mathbf{x}$. Притоа, $\boldsymbol{\theta} = [\theta_0, \theta_1, \dots \theta_{L-1}] = [\mathbf{W}_0, \mathbf{b}_0, \mathbf{W}_1, \mathbf{b}_1, \dots \mathbf{W}_{L-1}, \mathbf{b}_{L-1}]$ каде L е бројот на слоеви на мрежата. Тогаш имаме:

$$\theta_l^{i+1} = \theta_l^i - \frac{\partial \mathcal{L}}{\partial \theta_l} \cdot \eta \quad \text{за } \theta_l \in \boldsymbol{\theta} \,, \tag{8.9}$$

каде $\frac{\partial \mathcal{L}}{\partial \theta_l}$ е парцијалниот извод на функцијата на грешка во однос на параметарот θ_l, а η е чекорот на учење. Притоа, за пресметување на парцијалниот извод се користи правилото за пресметување на извод на сложена функција[34], односно:

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \frac{\partial \mathcal{L}}{\partial \tilde{y}} \cdot \frac{\partial \tilde{y}}{\partial y_{L-2}} \cdots \frac{\partial y_{l+1}}{\partial \theta_{l+1}} \cdot \frac{\partial y_l}{\partial \theta_l} \,, \tag{8.10}$$

Каде y_l е излезот на l-от слој на мрежата. На пример, за модел составен од еден неврон, градиентот во однос на тежината w_0, ќе биде:

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{\partial \mathcal{L}}{\partial \tilde{y}} \cdot \frac{\partial \tilde{y}}{\partial w_0} \,. \tag{8.11}$$

Користејќи го правилото за извод на сложена функција, пресметување на градиентот за апдејтирање на сите параметри $\boldsymbol{\theta}$ на невронската мрежа започнува со пресметка на градиентите

[32] Анг. *multi layer perceptron.*
[33] Анг. *gradient descent (GD).*
[34] Анг. *chain-rule.*

за излезниот слој, па за последниот скриен слој и оди наназад до почетокот на мрежата. Овој процес се нарекува пропагација наназад.[35]

За ГД да може да се употреби за тренирање на невронски мрежи, мора функцијата на грешка, како и сите излезни нелинеарности на невроните во мрежата да бидат диференцијабилни. Инаку не би можел да се пресмета градиентот за секој од параметрите. Ова е причината за употреба на сигмоидата наместо пресекување на активацијата со остар праг, како што било направено во оригиналниот перцептрон, кај кого важело:

$$y = f(a) = \begin{cases} 1 & \text{ако } a > 0{,}5 \\ 0 & \text{поинаку} \end{cases} . \tag{8.12}$$

Денес најчесто се употребуваат следните функции за грешка:

- средна квадратна грешка $MSE = \frac{1}{N}\sum_{n=0}^{N-1}(y - \tilde{y})^2$ – основна функција на грешка за регресија и бинарна класификација,
- меѓу-ентропија $CE = -\frac{1}{N}\sum_{n=0}^{N-1} y \ln \tilde{y} + (1-y)\ln(1-\tilde{y})$ – кај модели за класификација со излезна нелинеарност сигмоида; нејзиниот извод има подобри карактеристики,
- логаритамска веројатност[36] $LL = -\ln \tilde{y}$ – кај моделите со софтмакс функција на излез.

Во рамките на функцијата на грешка се вклучува и дел за регуларизација. Најчесто тоа е регуларизација на $L2$ нормата на параметрите на моделот $\boldsymbol{\theta}$. На пример, при употреба на средната квадратна грешка, би имале:

$$L(y, \tilde{y}, \boldsymbol{\theta}) = L(y, \tilde{y}) + \lambda \sum_{l=0}^{L-1} (\mathbf{W}_l^T \mathbf{W}_l + \mathbf{b}_l^T \mathbf{b}_l) , \tag{8.13}$$

каде 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.

При тренирањето на невронски мрежи, поради тоа што вообичаено се работи со големи множества на влезни податоци, станува неисплатливо градиентот да се пресметува за сите влезни примероци. Другиот екстрем е адаптацијата на параметрите да се прави со градиентот пресметан за секој од примероците земен по случаен избор. Оваа варијанта на алгоритамот се нарекува стохастично спуштање по градиентот (СГД)[37]. Изминувањето на целото тренинг множество се нарекува и епоха. Случајниот избор на влезните примероци технички се изведува преку случајно мешање на датасетот пред секоја епоха, по што следи секвенцијално земање на примероците.

Сепак, пресметката на градиентот по примерок не дава добра естимацијата на вистинскиот градиент на функцијата на грешка. Поради тоа, најчесто се употребува компромисно решение во кое се зема подмножество, или купче[38] примероци од множеството за тренирање по случаен избор, и се врши адаптација на параметрите за пресметаниот градиент.

Оваа верзија на ГД алгоритамот се нарекува спуштање по градиентот со купчиња (МБГД)[39] Технички постои разлика помеѓу МБГД и БГД – кај МБГД се врши апдејтирање на параметрите за секое купче, додека кај БГД 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 МБГД алгоритамот но под името СГД.

При употреба на СГД, чекорот на учење е еден од најважните параметри во тренирањето на невронските мрежи. Ако е преголем оптимизацијата може да го натфрли минимумот на

[35] Анг. *backpropagation*.
[36] Анг. *log-likelihood*.
[37] Анг. *stochastic gradient descent (SGD)*.
[38] Анг. *batch* или *mini-batch*.
[39] Анг. *mini batch gradient descent (MBGD)*.

функцијата на грешка, додека пак, ако е премал, на алгоритамот ќе му бидат потребни многу итерации за да заврши тренирањето. Затоа вообичаено се употребуваат стратегии на промена на чекорот на учење[40]. Една едноставна, а често употребувана, стратегија е чекорот да се намалува во тек на тренирањето. Постојат и понапредни алгоритми, каков што е Адам (Kingma and Ba, 2014), кои в'предвид ги земаат првиот и вториот момент, односно брзината и забрзувањето, на промената на градиентот во однос на претходните итерации за адаптација на чекорот на учење.

Еден голем проблем со употребата на СГД алгоритамот е можноста алгоритамот да заглави во локален минимум, по цена на промашување на глобалниот минимум. Овој ризик е поголем кај помалите НМ, додека кај длабоките НМ, веројатностите за постоење на локален минимум се минорни. Кај нив проблемот се сведува на заглавување на СГД во рамни региони на функцијата на грешка. Двата проблеми можат да се решат преку стратегии за промена на чекорот на учење, кои се базираат на негова реиницијализација на голема вредност по одреден број на итерации.

[40] Анг. *learning rate scheduling.*

8.5 Препознавање на звучен извор

Во овој дел ќе примениме невронска мрежа за препознавање на звучен извор[41] врз база на аудиосигналот кој тој го генерира. За таа цел ќе искористиме дел од аудиофајловите кои ги користевме досега, а дел ќе позајмиме од предметот Електроакустика на ФЕИТ.[42]

```
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)
```

Притоа, ако аудиосигналот е подолг од 5 s истиот ќе го скратиме за да не доминира во множеството на податоци.

Анализа на влезните податоци

За запознавање со аудиосигналите кои се дел од датасетот кој го креиравме може да го искористите знаењето кое досега го стекнавте на предметов. За поголема компактност, тука директно ќе прејдеме на поделбата на датасетот на подмножества.

Поделба на податоците на множества за тренирање и тестирање

Во оваа демонстрација на процесот на примена на машинското учење ќе направиме само едноставна поделба на податоците на множества за тренирање и тестирање. Ќе одвоиме 50% од секој од аудиосигналите за креирање на множеството за тест.

```
test_coef = .5  # 50%
x_trains = []
x_tests = []
for x in xs:
    x_len = x.size
    test_len = int(x_len * test_coef)
    x_train = x[:-test_len]
    x_test = x[-test_len:]
    x_trains.append(x_train)
    x_tests.append(x_test)
```

[41] Анг. *sound source recognition (SSR)*.
[42] https://github.com/FEEIT-FreeCourseWare/Electroacoustics

Екстракција на карактеристики

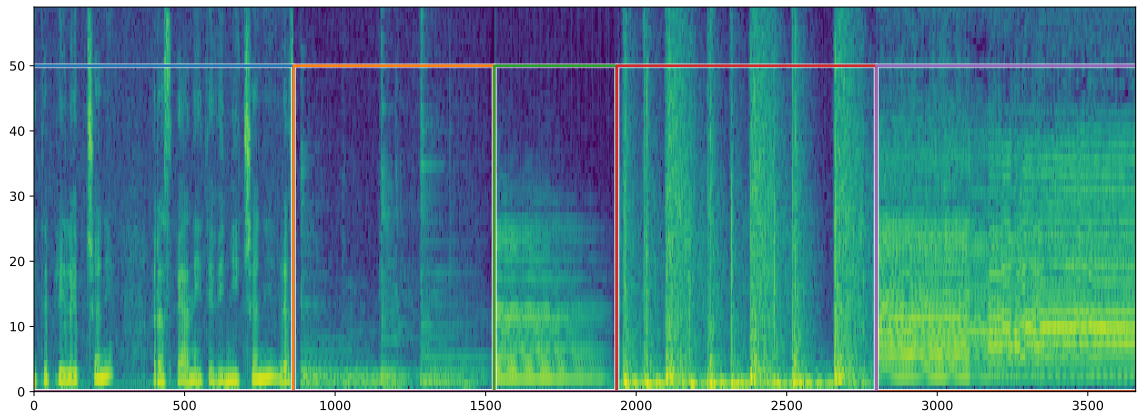
Вообичаена пракса е кога се работи за препознавање на звучните извори, или на пример за препознавање на говор[43] или говорник[44], да се употребат спектрални карактеристики за опис на аудиосигналите. Тука, ќе го искористиме амплитудниот спектар како наједноставна репрезентација на спектралната содржина, екстрахиран со методата на прозорци. Притоа ќе употребиме краток прозорец од 256 примероци, што соодветствува на 5,8 ms за фреквенција на семплирање 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. Со тоа ќе обезбедиме мрежата да ја научи спектралната анвелопа, односно бојата на тонот, наместо неговата висина.

```
import das

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 = das.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 = das.get_spectrogram(
            fs, x_test_sig, n_win=256, plot=False)
    x_test_feats = spectrogram.T
    mask_spectrogram = np.all(x_test_feats < -80, axis=1)
    x_test_feats = x_test_feats[~mask_spectrogram, :60]
    n_samples = x_test_feats.shape[0]
    y_test_targets = np.zeros([n_samples, n_labels])
    y_test_targets[:, i] = 1

    if x_train is None:
        x_train = x_train_feats
        x_test = x_test_feats
        y_train = y_train_targets
        y_test = y_test_targets
    else:
        x_train = np.concatenate((x_train, x_train_feats), axis=0)
        x_test = np.concatenate((x_test, x_test_feats), axis=0)
        y_train = np.concatenate((y_train, y_train_targets), axis=0)
        y_test = np.concatenate((y_test, y_test_targets), axis=0)
```

Притоа ги задржавме само најниските 60 бинови од спектрограмот кои одговараат на фреквенции до 10.3 kHz, и правиме негово транспонирање за примероците да бидат редици а карактеристиките колони во матриците на влезни податоци.

За визуелизација на спектрограмите за различните класи и потврда дека тие можат да се искористат за нивно разликување ќе ги прикажеме на графикон преклопени со типот на

[43] Анг. *automatic speech recognition (ASR).*
[44] Анг. *automatic speaker recognition.*



Сл. 8.4: Карактеристики екстрахирани од множеството за тестирање со задржување на првите 80 коефициенти, или до фреквенција од 13.8 kHz, на спектарот за рамки од 256 примероци, или околу 5,8 ms од сигналот. Врз нив се исцртани петте класи: глас, гитара, виолина, тапан, и зурла.

инструментот. Резултатот може да се види за тест множеството на Сл. 8.4 Може да видиме дека постојат јасни разлики во спектрите на сигналите за различните звучни извори, па може да очекуваме дека ќе добиеме добра точност на предикција на нашиот класификатор.

```
x_axis = np.arange(x_train.shape[0])
y_axis = np.arange(x_train.shape[1])
plt.figure()
plt.imshow(x_train.T, aspect='auto', origin='lower',
           extent=[0, x_axis[-1], 0, y_axis[-1]], vmin=-100,)
plt.plot(y_train*50, lw=2)

x_axis = np.arange(x_test.shape[0])
y_axis = np.arange(x_test.shape[1])
plt.figure()
plt.imshow(x_test.T, aspect='auto', origin='lower',
           extent=[0, x_axis[-1], 0, y_axis[-1]], vmin=-100,)
plt.plot(y_test*50, lw=2)
```

Тренирање на моделот

За оваа анализа ќе ја искористиме имплементацијата на невронска мрежа за класификација во класата `MLPClassifier` во модулот `neural_network` на пакетот за машинско учење Сајкит-лерн.[45]

Пред тренирањето на моделот може да направиме случајно мешање на примероците во множеството за тренирање и тестирање. Ова мешање на примероците од различните класи, ќе придонесе кон подобрување на апроксимацијата на градиентот во СГД алгоритамот. За тоа ќе го искористиме `random` модулот на Нумпај кој содржи генератор на псевдо-случајни броеви[46]. Тоа е машина со конечен број на состојби кој генерира секвенца на броеви чија статистика е како на случајна низа на броеви, иако начинот на кој се генерираат е детерминистички. Така, со поставување на состојбата на генераторот со наредбата `np.random.seed`, при секое извршување на кодот ќе ги добиеме истите резултати. Во конкретниот случај тоа ќе биде истото мешање на датасетот. Ова е вообичаена пракса во областа машинско учење во која случајноста игра голема улога, на пр. при иницијализација на параметрите на моделот, со што се обезбедува репродуцибилност на резултатите.

[45] sciKit-Learn http://scikit-learn.org/stable/

[46] Анг. *pseudo-random number generator (PRNG).*

```
# измешај ги податоците
np.random.seed(42)
train_ind_shuffle = np.random.permutation(x_train.shape[0])
x_train = x_train[train_ind_shuffle]
y_train = y_train[train_ind_shuffle]
test_ind_shuffle = np.random.permutation(x_test.shape[0])
x_test = x_test[test_ind_shuffle]
y_test = y_test[test_ind_shuffle]
```

Уште подобро при тренирањето на невронските мрежи е податоците да се измешаат на почетокот од секоја епоха. Тоа може да се овозможи во `MLPClassifier` со аргументот `shuffle`. Тој и низата други параметри ни овозможува да ги поставиме сите хиперпараметри за тренирање на нашиот класификатор.

```
from sklearn import neural_network
from sklearn import metrics

mlp = neural_network.MLPClassifier(
        hidden_layer_sizes=(100, 20, 10),  # број на неврони во скриените слоеви
        activation='relu',  # нелинеарност на невроните во скриените слоеви
        learning_rate_init=0.01,  # чекор на учење
        alpha=1e-4,  # коефициент на L2 регуларизација
        solver='adam',  # алгоритам за тренирање
        shuffle=True,  # мешање на податоците во секоја епоха
        random_state=42,  # иницијализација на ПРНГ
        max_iter=3000,  # максимален број на итерации
        tol=1e-9,  # толеранција за заситување на тренирањето
        n_iter_no_change=40,  # запирање на тренирањето заради заситување
        early_stopping=False,  # рано запирање со валидациски сет
        verbose=1)  # побогат испис при тренирањето
mlp.fit(x_train, y_train)  # тренирање на моделот
```

```
Iteration 1, loss = 9.14906290
Iteration 2, loss = 2.60331358
Iteration 3, loss = 2.40171081
Iteration 4, loss = 2.32905906
Iteration 5, loss = 2.22766382
Iteration 6, loss = 2.12148965
Iteration 7, loss = 2.04304694
Iteration 8, loss = 1.97768373
Iteration 9, loss = 1.88609207
...
Iteration 491, loss = 0.29752739
Training loss did not improve more than tol for 40 consecutive epochs. Stopping.
```

Може да видиме дека тренирањето на моделот завршува за 491 итерација при постигната грешка од 0,2975. Во Сајкитлерн во невронските мрежи за класификација на повеќе класи се употребува софтмакс како излезен слој, а меѓу-ентропијата како функција на грешка.

Евалуација на моделот

За да направиме предвидување врз база на множеството за тестирање може да се послужиме со `mlp.predict_proba` функцијата која ќе ни го даде излезот на невронската мрежа, односно `mlp.predict` функцијата која ќе ни ја даде предвидената излезна класа како бинарен вектор користејќи праг од 0,5.

За евалуација на мрежата, дополнително е имплементирана функцијата `mlp.score` која ќе ја искористиме за да видиме која е точноста на предвидувањата на нашата мрежа.

```
accuracy = mlp.score(x_test, y_test)
print(accuracy)
```

```
0.885792349726776
```

Може да видиме дека нашиот модел постигнува 88,58% точност што е далеку над точноста на случаен избор од 20%! Дополнителни подобрувања може да остваримe со земање на подолги аудиосигнали од нашите звучни извори, употреба на понапредни карактеристики, како и оптимизација на хиперпараметрите на невронската мрежа.

За дополнително да уочиме каде најмногу греши невронската мрежа може да ја пресметаме матрицата на грешка.[47] За да ја пресметаме, ќе направиме предвидување на класите со тест множеството.

```
y_pred_prob = mlp.predict_proba(x_test)
cm = metrics.confusion_matrix(
        y_test.argmax(axis=1),
        y_pred_prob.argmax(axis=1))
print(cm)
```

```
[[812  26   2  22   0]
 [ 78 514   3  71   1]
 [ 19   4 384   0   0]
 [ 19  72   0 769   2]
 [  2   2  13   0 845]]
```

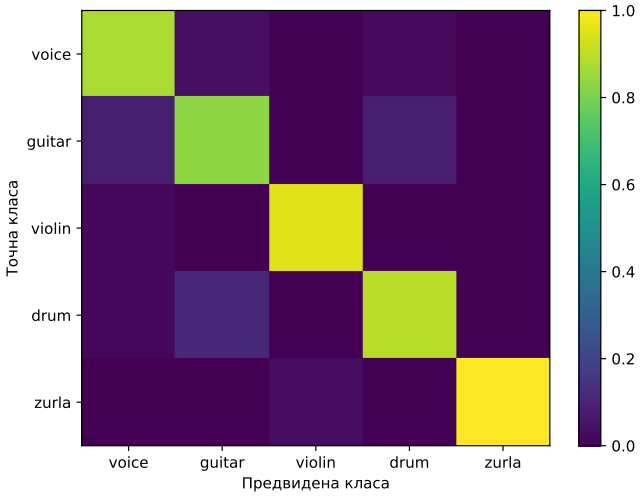
Редиците претставуваат точните класи, а колоните класите предвидени од невронската мрежа. Истата можеме да ја нормализираме за да ја добиеме процентуалната грешка која ја прави мрежата по предвидена класа. Исто така, за појасен приказ ќе ја прикажеме матрицата на грешка преку топлинска мапа[48] на Сл. 8.5

```
cm = cm / cm.sum(axis=0)
print(cm)

fig, ax = plt.subplots()
im = ax.imshow(cm, aspect='auto', interpolation='nearest', vmax=1, vmin=0)
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=y_labels,
       yticklabels=y_labels,
       ylabel='Точна класа',
       xlabel='Предвидена класа')
fig.colorbar(im)
```

```
[[0.87311828 0.0420712  0.00497512 0.02552204 0.        ]
 [0.08387097 0.83171521 0.00746269 0.08236659 0.00117925]
 [0.02043011 0.00647249 0.95522388 0.         0.        ]
 [0.02043011 0.11650485 0.         0.89211137 0.00235849]
 [0.00215054 0.00323625 0.03233831 0.         0.99646226]]
```

Може да се види дека најчесто грешки невронската мрежа прави при одлучување помеѓу класите глас и гитара, како и гитара и тапан, кои имаат сличности во спектрите во дел од примероците во тест множеството прикажани на Сл. 8.4.

[47] Анг. *confusion matrix*.
[48] Анг. *heat map*.



Сл. 8.5: Матрица на грешка за истренираната невронска мрежа.

✓ Задача за дома.

1. Дополни го кодот за екстракција на карактеристики со банка на 23 филтри која ќе ја пресмета средната енергија во секој фреквенциски подопсег. Ова ќе овозможи примена на подолги прозорци за пресметка на спектрограмите што ќе ни даде временско усреднување на сигналите. Одреди колку ќе се подобрат перформансите на невронската мрежа.
2. Искористи ја функцијата за меѓувалидација со мешање `ShuffleSplit` имплементирана во Сајкитлерн и оптимизирај го изборот на хиперпараметри на невронската мрежа. Која е најдобрата точност што може да се постигне?

Додаток А

Слободен и отворен софтвер за инженерска и научна работа

Еден од најпрочуените софтверски пакети за нумеричка обработка е програмскиот пакет Матлаб[1]. Матлаб, 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 1 милион корисници инженери, научници и економисти.

Сепак Матлаб, како комерцијален софтвер носи и низа од недостатоци, пред сè високата цена која го става вон дофат на студентите, истражувачите, малите компании, како и на научно истражувачките и образовните установи во поголем дел од светот. Други недостатоци на Матлаб се ограничената преносливост на кодот, како и неговата затвореност.

А.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 Сталман[3] во 1983 г. со креирањето на ГНУ оперативниот систем, а подоцна со воспоставување на Фондацијата за слободен софтвер[4] во 1985 г., како и поширокото движење за отвореност[5], а тоа е заедништво во создавањето и напредувањето на технологијата и човештвото.

А.2 Четири слободи

Слободниот софтвер е дефиниран со четирите слободи:[6]

- Слобода 0. Слобода да ја користите програмата за било која намена.

[1] MATLAB®Matrix Laboratory, The MathWorks, Inc., Natick, Massachusetts, United States. http://www.mathworks.com/products/matlab/

[2] Wikipedia – Free software movement https://en.wikipedia.org/wiki/Free_software_movement

[3] Wikipedia – Richard Stallman https://en.wikipedia.org/wiki/Richard_Stallman

Предавање на Ричард Сталман за философијата на движењето за слободен софтвер – Richard Stallman – Free software, free society, TEDxGeneva 2014 https://www.youtube.com/watch?v=Ag1AKIl_2GM

[4] Wikipedia – Free Software Foundation https://en.wikipedia.org/wiki/Free_Software_Foundation

[5] Wikipedia – 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_lLNtd0

[6] Превземено од вебстраната на организацијата Слободен софтвер Македонија https://slobodensoftver.org.mk/shto

Додавањето рестрикции за користење на слободен софтвер, како што се временските рестрикции („Пробен период од 30 дена“, „Лиценцата истекува на 1 јануари 2005“), рестрикции на целта („Дозволена е употреба за истражувачки и некомерцијални цели“) или рестрикции на географската област („Мора да се користи во земјата А“), ја прават програмата неслободна.

- Слобода 1. Слобода да проучите како работи програмата и како истата да ја адаптирате на сопствените потреби.

  Додавањето легални или практични рестрикции на разбирањето или менувањето на програмата, како што се задолжително купување на специјални лиценци, потпишување на спогодба за неоткривање (Non-Disclosure-Agreement) или правењето изворниот код да биде недостапен, исто така ја прават програмата неслободна. Без слободата да се менува програмата, луѓето ќе останат на милост на единствен снабдувач.

- Слобода 2. Слобода да редистрибуирате копии за да му помогнете на вашиот сосед.

  Софтверот може да се копира/дистрибуира скоро без никакви трошоци. Ако не смеете да му дадете некоја програма на некој човек кому таа му треба, тоа ја прави програмата неслободна. Се разбира, доколку сакате, за ваквите активности можете да наплатите.

- Слобода 3 Слобода да ја подобрувате програмата и да ги издадете вашите подобрувања во јавноста, од што корист ќе има целата заедница.

  Сите луѓе не се подеднакво добри програмери. Некои луѓе пак воопшто не знаат да програмираат. Оваа слобода им дозволува на оние луѓе кои немаат време или знаење да решат некој проблем индиректно да пристапат до слободата за менување на програмата. Се разбира, доколку сакате, за ваквите активности можете да наплатите.

Доколку софтверот не ги исполнува сите горни услови, тогаш тој не е слободен софтвер.

A.3 Предности на слободниот софтвер

Од практичен аспект, отворениот софтвер има низа предности над затворениот софтвер и тоа:

- достапноста – поради основната премиса на давање на изворниот код, со цел да се овозможи неговиот развој од заедницата, отворениот софтвер е *de facto* и бесплатен софтвер. Така, повеќето производители на слободниот софтвер живеат од донации, но и од продавање поддршка за нивниот производ.

- безбедноста – поради достапноста на изворниот код, не постои начин производителот на софтверот да прави нешто скриено од вас, а секој спорен дел од кодот е подложен на промена од заедницата. Кај затворениот софтвер тоа не е случај.[7,8]

- слободата од производителот – како корисници на отворениот софтвер, вие не сте затворени во екосистемот на производителот.[9] Истиот тој софтвер може да биде превземен од друга заедница на програмери и да продолжи неговото одржување и развој во друга насока.

[7] Во Windows 10 производителот го задржува правото да ги чува вашите приватни податоци како што вели во изјавата за приватност: “Finally, we will access, disclose and preserve personal data, including your content (such as the content of your emails, other private communications or files in private folders), when we have a good faith belief that doing so is necessary ...”
Истите механизми се додадени во претходните верзии на Windows преку автоматските надградби.
Zach Epstein, Windows 10 is spying on almost everything you do – here’s how to opt out, Jul 31, 2015, http://bgr.com/2015/07/31/windows-10-upgrade-spying-how-to-opt-out/
Ashley Allen, How to Stop Windows 7 and 8 From Spying on You http://www.eteknix.com/stop-windows-7-8-spying/

[8] Епл и Самсунг ги забавија телефоните на корисниците преку нивното редовно ажурирање https://www.cnet.com/news/apple-and-samsung-fined-for-slowing-down-phones-with-updates/

[9] Don Reisinger – Steve Jobs wanted to ’further lock customers’ into Apple’s ’ecosystem’ https://www.cnet.com/news/steve-jobs-wanted-to-further-lock-customers-into-apples-ecosystem/

- подобар квалитет – при воспоставување на критична големина на заедницата околу еден отворен софтвер, развојот не може да се спореди со ресурсите кои ги поседува било која корпорација во светот. Така, развојот на Линукс јадрото[10], кое е во основата оперативниот систем ГНУ/Линукс познат и само како Линукс[11] и повеќе од 600-те ГНУ/Линукс дистрибуции[12], првично напишано од Линус Торвалдс[13], денес претставува најголемиот здружен проект во историјата на човештвото со околу 6000 активни развивачи, над 20 милиони редови на код, и со проценета развојна вредност од над 2 милијарди евра.[14]

Сите овие придобивки заедно придонесуваат за широка распространетост на слободниот софтвер денес. Така, ГНУ/Линукс и ФриБСД[15] оперативните системи се во употреба во 98,27 % од серверите на интернет (споредено со 1,73 % со Виндоус), 79,3 % од паметните телефони (Андроид оперативниот систем)[16], и 99 % од суперкомпјутерите[17]. Сепак, неговиот пробив во персоналните компјутери засега е незначителен – 2,1 % (наспроти 87 % на Виндоус и 9,7 % на МекОС).

А.4 Одржливост

Постојат различни начини на кои се реализира финансиската поддршка на слободниот софтвер и покрај бесплатноста и тоа:

- финансиска поддршка од компании – зад многу пакети слободен софтвер стојат компании од чиј интерес е неговиот развојот, Најдобар пример за тоа е можеби самото Линукс јадро на кое работат инженери од многу компании од целиот свет, а најголемиот придонес го има компанијата Интел. Тука се и низа на ГНУ/Линукс дистрибуции меѓу кои Убунту[18], Федора[19], и ОпенСусе[20], како и пакетите за длабоко учење Тензорфлоу[21] и Пајторч[22], исто така развивани од компании,

- финансиска поддршка од јавно финансирање и грантови – голем број на слободни софтвери се плод на работата на инженери и научници финансирани од државите низ светот или од приватни фондации. Таков е на пример КиКАД софтверот за електронски дизајн и изработка на печатени плочи развиван во ЦЕРН[23], софтверот за обработка на аудио Аудасити започнат во Универзитот Карнеги Мелон[24], или пак пакетот за машинско учење Сајкитлрн започнат во Инриа[25],

- бизнис модел базиран на поддршка – најголемата компанија која денес работи исклучиво со слободен софтвер е Ред Хет чиј ГНУ/Линукс оперативен систем е еден од најзастапените на интернет серверите.[26] Ред Хет заработува преку продажба на поддршка за овој оперативен систем и во моментов е проценета на вредност од 38 милијарди УСД,

- финансиска поддршка од донации – многу слободни софтвери егзистираат благодарејќи на донации направени од нивните корисници. Тука спаѓаат најголем број од ГНУ/Линукс

[10] Wikipedia – Linux kernel https://en.wikipedia.org/wiki/Linux_kernel
[11] Wikipedia – Linux https://en.wikipedia.org/wiki/Linux
[12] Wikipedia – List of Linux distributions https://en.wikipedia.org/wiki/List_of_Linux_distributions
[13] Wikipedia – Linus Torvalds https://en.wikipedia.org/wiki/Linus_Torvalds
[14] Добар документарен филм за раѓањето и развојот на ГНУ/Линукс оперативниот систем е Revolution OS - 2001 https://www.youtube.com/watch?v=Eluzi70O-P4
[15] FreeBSD https://www.freebsd.org/
[16] Оваа бројка е речиси 91 % ако се има в'предвид дека и iOS е базиран на Линукс јадрото.
[17] Linux is Running on Almost All of the Top 500 Supercomputers https://itsfoss.com/linux-supercomputers-2017/
[18] Ubuntu https://www.ubuntu.com/
[19] Fedora https://getfedora.org/
[20] OpenSUSE https://www.opensuse.org/
[21] TensorFlow – An end-to-end open source machine learning platform https://www.tensorflow.org/
[22] PyTorch – from research to production https://pytorch.org/
[23] KiCad EDA – A Cross Platform and Open Source Electronics Design Automation Suite http://kicad-pcb.org/
[24] Audacity – Free, open source, cross-platform audio software https://www.audacityteam.org/
[25] scikit-learn – Machine Learning in Python https://scikit-learn.org/stable/index.html
[26] Red Hat – The world's leading provider of open source solutions https://www.redhat.com

дистрибуциите како на пример Манџаро[27] или Минт[28], а исто така СпајдерSpyder – The Scientific Python Development Environment https://manjaro.org/ развојната средина за Питон која ќе ја користиме во предметов,

- ентузијазам – мотивот нешто да се создаде или подобри и да се сподели со целиот свет понекогаш е доволен мотив за развој на слободниот софтвер. Постојат низа пакети со заедници на развивачи кои немаат финансиски придобивки од нивната работа на проектот, но сепак продолжуваат да работат на него водени од сопствените убедувања и стремеж кон повисоки вредности.

А.5 Слободен софтвер за инженерска и научна работа

Постојат низа на слободни софтвери кои можат да бидат искористени за обработка на нумерички податоци.

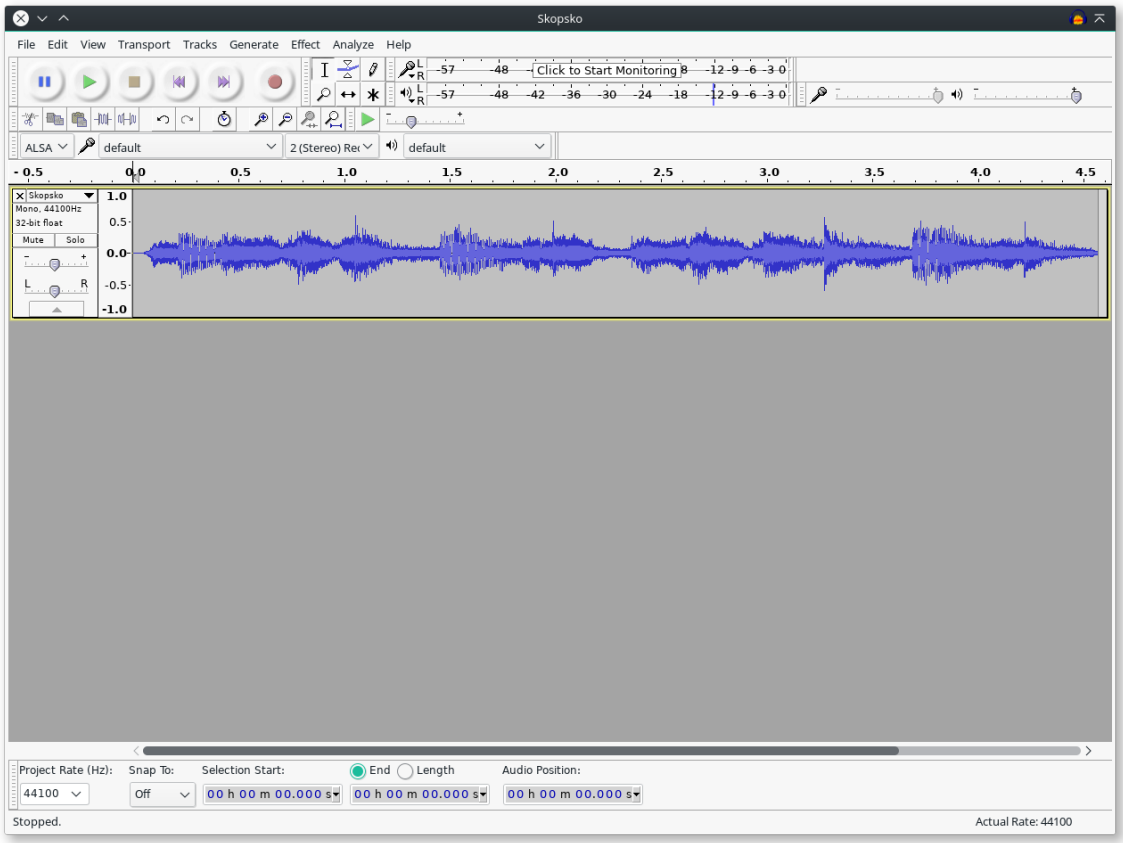
- ГНУ Октејв[29] има синтакса направена да биде во голема мера компатибилна со онаа на Матлаб. Во Октејв се реализирани голем број на пакети кои можат да се искористат за обработка на најразлични типови на сигнали. Проблемот со Октејв е во неговата мала брзина на извршување, поради што најмногу се употребува во образованието како замена за Матлаб.
- Сајлаб[30] е слободен софтвер за нумеричка обработка наменет за инженери и научници, во употреба од 1994 г. Сајлаб во себе вклучува и слободна замена за Симулинк пакетот на Матлаб, наречена Икскос[31].
- Питон[32] е широко распространет, повеќенаменски, интерпретиран и динамичен програмски јазик на високо ниво направен од Гуидо ван Росум[33] 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 Питон идеална основа за слободната работа и соработка на научната и образовната заедница ширум светот.
- Џулиа[34] е јазик за нумеричко процесирање со компајлирање направен на МИТ, кој иако има синтакса на високо ниво како онаа на Матлаб, работи речиси еднакво брзо со код напишан во С. И покрај големиот потенцијал на Џулија, за сега неговата примена останува ограничена во области во кои е неопходна голема процесирачка моќ.

Аудасити

Аудасити[35] е уште еден слободен софтвер кој ќе ни биде од корист во анализата, едитирањето и снимањето на дигитално аудио, прикажан на Сл. А.1. Неговиот развој го започнале Доминик Мацони и Роџер Даненберг во 1999 во Универзитетот Карнеги Мелон и е иницијално објавен во 2000 како верзија 0,8. Од 2011, тој е 11-от најсимнуван софтвер на Сорсфорџ, со 76,5 милиони симнувања. Аудасити е добитник на наградата за најдобар проект за мултимедија од заедницата Сорсфорџ во 2007 и 2009. Во 2015 е преместен на Фосуб каде за 4 месеци постигнува 10 милиони симнувања.[36]

Освен тоа што поддржува снимање од повеќе извори, Аудасити може да се искористи за процесирање на сите типови на аудио, преку додавање на ефекти како нормализација,

[27] Manjaro – Professional Linux at its best https://manjaro.org/
[28] Linux Mint – From freedom came elegance https://linuxmint.com/
[29] GNU Octave – Scientific Programming Language https://www.gnu.org/software/octave/
[30] Scilab – Open source software for numerical computation https://www.scilab.org/
[31] Xcos https://www.scilab.org/software/xcos
[32] Python https://www.python.org/
[33] Wikipedia – Guido van Rossum https://en.wikipedia.org/wiki/Guido_van_Rossum
[34] The Julia Programming Language https://julialang.org/
[35] Audacity. http://www.audacityteam.org/
[36] Wikipedia: Audacity (audio editor). https://en.wikipedia.org/wiki/Audacity_(audio_editor)



Сл. А.1: Отворен аудио фајл во главниот прозорец на Аудасити.

поткастрување, и прелевање. Тој може да се користи за снимање и миксање на цели албуми, како што е случајот со групата Tune-Yards. Тој е во употреба и во националниот курс за ICT ниво 2 на OCR[37] во Велика Британија. Главните особини на Аудасити вклучуваат:

- Вчитување и снимање на различни типови на аудио формати, како WAV, AIFF, MP3, Ogg Vorbis, FLAC, WMA, AAC, AMR и AC3.
- Снимање и репродукција на звук.
- Едитирање со неограничен број на undo.
- Автоматска поделба на аудио траки на дигитализирани снимки од касети или грамофонски плочи.
- Повеќеканално миксање.
- Голем број на аудио ефекти и плагини. Додатни ефекти можат да се напишат во Nyquist кој е диалект на Lisp, а поддржани се плагини направени во отворениот LV2 стандард, како и VST плагини.
- Едитирање на амплитудната анвелопа.
- Намалување на шумот.
- Намалување на вокалите.
- Спектрална анализа со употреба на FFT.
- Подршка на повеќеканално дигитално аудио со фреквенција на семплирање до 96 kHz и резолуција до 32 bit.

[37] Oxford, Cambridge and RSA Examinations

- Прецизно нагодување на брзината на аудиото без промена во фреквенцијата на звукот.
- Нагодување на висината на тонот без промена на брзината.
- Можности за модерно повеќеканално едитирање.
- Работа на повеќе платформи.
- Приказ на ефектите базирани на LADSPA, VST (32-bit) и Audio Unit (OS X) во реално време.
- Зачувување и вчитување на кориснички предпоставувања.

Тој исто така работи на сите оперативни системи.

Додаток А

Питон за процесирање на аудиосигналите

За процесирањето на дигиталните аудиосигнали ќе биде искористен програмскиот јазик Питон и тоа неговата понова верзија 3, заедно со библиотеките:

- Нумпај – за работа со вектори и матрици,[1]
- Сајпај – за дигитално процесирање на сигнали,[2]
- Матплотлиб – за визуелизација.[3]

Освен овие постојат мноштво библиотеки за Питон кои се користат во научните истражувања како на пример Пандас[4] за статистички анализи, Симпај[5] за симболичка математика, Сајкитлрн[6] за машинско учење итн.

Како интерфејс кон Питон ќе ја користиме интерактивната конзола ИПитон[7] и научната развојна средина за Питон Спајдер[8].

А.1 Основи поставки во ГНУ/Линукс

Иако користењето на Питон не е врзано со ГНУ/Линукс оперативниот систем, вежбите во овој предмет ќе се базираат на работа под ГНУ/Линукс. Доколку веќе немате ГНУ/Линукс, истиот се препорачува да го инсталирате паралелно на постоечкиот оперативен систем. Во најмала рака може да инсталирате ГНУ/Линукс во виртуелна машина, но ова може да ги ограничи постоечките ресурси за процесирање на сигналите.[9] Во Лабораторијата за дигитално процесирање на сигнали ќе работиме со Убунту Мате[10] кој е базиран на Убунту, а ја користи десктоп средината Мате чиј изглед е базиран на Гном 2, а е имплементиран во Гном 3.[11]

[1] NumPy http://www.numpy.org/
[2] SciPy http://www.scipy.org/
[3] MatPlotLib http://matplotlib.org/
[4] Pandas http://pandas.pydata.org/
[5] SymPy http://www.sympy.org/en/index.html
[6] SciKit-Learn http://scikit-learn.org/stable/
[7] IPython Interactive Computing http://ipython.org/
[8] Spyder – The Scientific PYthon Development EnviRonment https://github.com/spyder-ide/spyder
[9] Добар преглед на популарноста на различните Линукс дистрибуции, како и повеќе информации за истите може да најдете на вебстраната *Distrowatch*. http://distrowatch.com/
[10] Ubuntu MATE. https://ubuntu-mate.org/
[11] Кај ГНУ/Линукс оперативните системи можат да се користат различни десктоп средини како GNOME и MATE, но уште и KDE Plasma, Xfce, LXQt, Cinnamon, Pantheon, итн.

За работа со ГНУ/Линукс можеме да го искористиме стандардниот БАШ терминал.[12] Вообичаена кратенка за отворање на нов терминал е `ctrl-alt-t`, или ако дистрибуцијата доаѓа со терминал на спуштање копчето `F12`.

За почеток треба во вашиот основен фолдер[13] да отворите нова папка со името на предметот. Тоа може да го направите преку фајл експлорерот, или преку терминалот:

```
~ $ mkdir das
```

Следно, од Гитхаб страната на предметот Дигитални аудиосистеми[14] превземете го фолдерот со звучни сегменти кои ќе ги користиме во вежбиве. Тоа можете да го направите на следниот начин:

```
~ $ cd das
~/das/ $ git clone https://github.com/FEEIT-FreeCourseWare/Digital-Audio-Systems.git
~/das/ $ cp -r Digital-Audio-Systems/code/audio .
```

За да може да ги слушнеме овие аудиозаписи треба да го инсталираме SoX[15] кој претставува моќна алатка за конверзија на аудиофајлови од еден формат во друг, но може да се искористи и за додавање на различни аудиоефекти, како и снимање и преслушување на аудиофајлови. За инсталирање и надградба на софтверот и самиот оперативен систем во Линукс е одговорен менаџерот на пакети. Кај дистрибуциите од фамилијата на Убунту како менаџер се користи `apt-get` или на повисоко ниво `apt`. Поради безбедносни причини во Линукс при секое менување на инсталираниот софтвер и системските фајлови мора да се повикаме на администраторски привилегии преку наредбата sudo[16]:

```
~/das/ $ sudo apt install sox
```

По што може да преслушаме некој од аудиофајловите:

```
~/das/ $ play audio/Solzi.wav

Solzi.wav:

File Size: 345k      Bit Rate: 714k
Encoding: Signed PCM
Channels: 1 @ 16-bit
Samplerate: 44100Hz
Replaygain: off
Duration: 00:00:03.87

In:52.8% 00:00:02.04 [00:00:01.83] Out:90.1k [  -===|===-  ]        Clip:0
```

A.2 Основи на работата со Питон

Питон интерпретер

За работа со Питон може да ја искористиме стандардната инсталацијата на Питон која доаѓа со секоја ГНУ/Линукс дистрибуција. Питон интерпретерот можеме да го повикаме во стандардниот

[12] Bourne-Again Shell (BASH) средината е стандардна за сите дистрибуции на ГНУ/Линукс и МекОС. Добро напатствие за нејзина употреба претставува: Software Carpentry – The Unix Shell http://swcarpentry.github.io/shell-novice/

Постојат и понапредни шел средини како на пример Z shell – скратено Zsh. Еве еден туторијал за нејзина инсталација: Oh-My-Zsh! A Work of CLI Magic—Tutorial for Ubuntu https://medium.com/wearetheledger/oh-my-zsh-made-for-cli-lovers-installation-guide-3131ca5491fb

[13] Во ГНУ/Линукс основен фолдер на секој корисник е `/home/user_name/`, а кратенка за него е `~`.

[14] https://github.com/FEEIT-FreeCourseWare/Digital-Audio-Systems

[15] SoX - Sound eXchange. http://sox.sourceforge.net/

[16] Кратенка од *super user do*.

БАШ терминал со:

```
$ python
```

```
Python 3.7.2 (default, Jan 10 2019, 23:51:51)
[GCC 8.2.1 20181127] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
```

За излегување од Питон конзолата треба да ја притиснеме стандардната кратенка `ctrl-d` или да напишеме `exit()`.

ИПитон

Поради ограничените можности на основниот Питон интерпретер, вообичаено со Питон се работи во интерактивната конзола ИПитон која нуди низа на подобрувања. Нејзе може да ја инсталираме 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
```

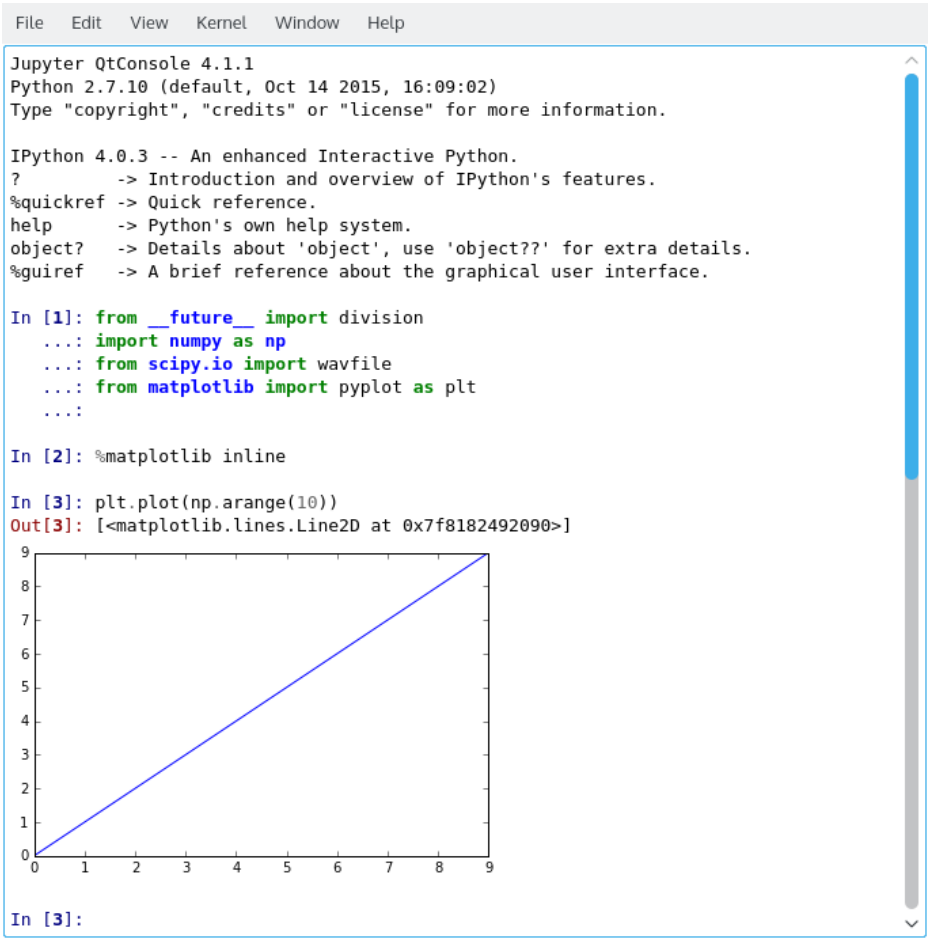
Некои од главните придобивки кои ги носи ИПитон се:

- пристап до стандардната помош во Питон, како на пример докстрингови на објекти и напатствието за Питон, преку наредбата `help`,
- низа од специјални наредби, наречени и „магии“, како на пример `%timeit` за мерење на времето потребно за извршување на една наредба, `%matplotlib` за овозможување на интерактивно исцртување, или `%history` за испишување, пребарување или запишување на историјата на извршените наредби; повеќе за овие наредби може да се види со наредбата `%magic`,
- информации за секој објект преку употреба на `?` наредбата,
- автоматско комплетирање на имињата на објектите и променливите од локалниот простор на имиња, како и имиња од локалниот фолдер, со употреба на `Tab` копчето,
- пребарување на претходно внесени наредби со стрелките и внесување на првите букви од саканата наредба, а со `ctrl-r` и со пребарување на целата содржина на претходните наредби,
- извршување на шел наредби со помош на `!`.

За да ги видите сите можности кои ги нуди ИПитон напишете `?` или `%quickref` во интерактивната конзола.

Дополнително, ИПитон е основата зад Јупајтер Кјутконзолата[17] прикажана на Сл. А.1, која е реализирана во Кјут технологијата и овозможува плотирање во самата конзола кое

[17] Jupyter QtConsole https://github.com/jupyter/qtconsole



Сл. А.1: Јупајтер Кјут конзолата нуди напредна интерактивност.

може да се активира со наредбата `%matplotlib inline`. Таа се стартува во терминалот со наредбата:

```
$ jupyter qtconsole
```

Виртуелни средини за Питон

Инсталирањето на Питон пакети, како што направивме со ИПитон козолата, директно во оперативниот систем не е препорачливо. За избегнување на судир помеѓу системската инсталација на Питон на ГНУ/Линукс оперативниот систем, како и за изолирање на екосистемот од инсталирани модули на секој засебен проект, правилно е да направиме Питон виртуелна средина за процесирање на дигиталните аудиосигнали. Постојат низа пакети за создавање и раководење со виртуелните средини во Питон, од кои `pipenv` [18] е оној кој е препорачан од Телото за пакување на Питон[19].

За инсталирање на `pipenv` треба да го инсталираме користејќи го системскиот `pip` [20] Поради можноста на употреба на Питон 2 како стандарден во инсталираната ГНУ/Линукс дистрибуција, најдобро е да напишеме:

```
$ sudo pip3 install pipenv
```

А доколку системот го нема `pip3`, истиот може да се инсталира како:

[18] Pipenv – Python Development Workflow for Humans https://github.com/pypa/pipenv
[19] Python Packaging Authority PyPA https://www.pypa.io/en/latest/
[20] Python Install Package (pip) https://pypi.org/project/pip/

```
$ sudo apt install python3-pip
```

За креирање на виртуелна средина во фолдерот за овој предмет ќе напишеме:

```
$ mkdir das
$ pipenv --python 3
```

```
Creating a virtualenv for this project…
Pipfile: /tmp/das/Pipfile
Using /usr/bin/python3 (3.7.2) to create virtualenv…
Creating virtual environment...Using base prefix '/usr'
New python executable in ~/.local/share/virtualenvs/das-aWLbLjVf/bin/python3
Also creating executable in ~/.local/share/virtualenvs/das-aWLbLjVf/bin/python
Installing setuptools, pip, wheel...
done.
Running virtualenv with interpreter /usr/bin/python3

Successfully created virtual environment!
Virtualenv location: ~/.local/share/virtualenvs/das-aWLbLjVf
Creating a Pipfile for this project…
```

Со ова `pipenv` креира нова виртуелна средина во локален фолдер во нашиот кориснички фолдер, во неа го копира системскиот Питон 3 и `pip`. За активирање на виртуелната средина може да ја искористиме наредбата `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 Питон интерпретерот или да инсталираме пакет со `pip`, тоа ќе се случува внатре во виртуелната средина:

```
$ which pip
/usr/bin/pip

$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/bin ...

$ pipenv shell
Launching subshell in virtual environment…
 . ~/.local/share/virtualenvs/das-aWLbLjVf/bin/activate

(das) $ echo $PATH
~/.local/share/virtualenvs/das-aWLbLjVf/bin:/usr/local/sbin:/usr/local/bin:...

(das) $ which pip
~/.local/share/virtualenvs/das-aWLbLjVf/bin/pip
```

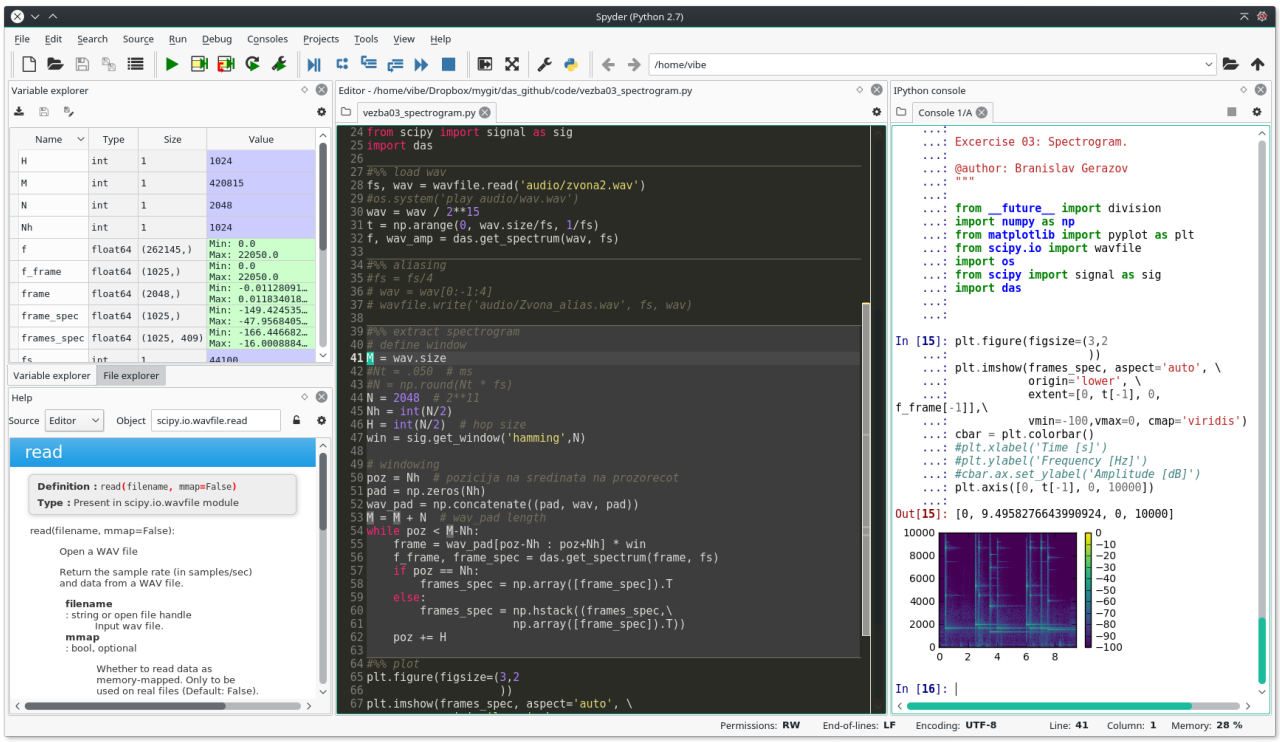
Гледаме дека `pipenv` ни дозначува дека сме во виртуелната средина со `(das)` пред БАШ промптот. За да излеземе од неа повторно може да ја употребиме кратенката `ctrl-d` или да напишеме `exit`.

Спајдер

Наместо да работиме со ИПитон конзолата во терминалот, во предметот дигитални аудиосистеми ќе ја употребиме развојната средина за Питон специјализирана за инженерска и научна работа Спајдер[21] прикажана на Сл. А.2. Спајдер во себе вклучува:

- Едитор – со вклучен прелистувач на функции/класи, можности за анализа на код, автоматско завршување на код, и вчитување на дефиниции.

[21] Spyder - The Scientific PYthon Development EnviRonment. https://github.com/spyder-ide/spyder



Сл. А.2: Спајдер развојната средина за Питон специјализирана за инженерска и научна работа.

- Интерактивна конзола – интегрирани Питон и иПитон конзоли со работни простори и подршка за дебагирање и поддршка за Матплотлиб, овозможуваат инстантна евалуација на кодот напишан во едиторот.
- Документација – покажување на документацијата на било која класа или функција повикана во едиторот или конзолата.
- Приказ на променливи – овозможува брза анализа на променливите генерирани со некој код.
- Приказ на фајлови и фолдери.
- Историја на наредби.

Спајдер можеме да го инсталираме во виртуелната средина која ја креиравме во работниот фолдер `das`:

```
(das) $ pip install spyder
```

A.3 Основи на Нумпај и Матплотлиб

За да ги инсталираме потребните модули во новата виртуелна средина ќе напишеме:

```
(das) $ pip install numpy matplotlib scipy
```

Сега може да ги повикаме во ИПитон:

```
In [1]: import numpy as np
In [2]: from matplotlib import pyplot as plt
In [3]: x = np.linspace(0, 2*np.pi, 100)
In [4]: y = np.sin(x)
In [5]: %matplotlib
```

```
Using matplotlib backend: Qt5Agg

In [6]: plt.plot(x, y)
Out[6]: [<matplotlib.lines.Line2D at 0x7fcbb448ffd0>]
```

Забележете дека го импортираме Нумпај со `import numpy as np` наместо со `from numpy import *`. Ова е препорачана практика за избегнување на оптеретување на постоечките функции во основниот простор на имиња[22], како и за зачувување на засебен простор со имиња за секој од импортираните модули. Ова овозможува и одлично автоматско надополнување на започнатото име на модул, функција или променлива во Питон.[23] Со наредбата `%matplotlib` се овозможува интерактивното плотирање во ИПитон конзолата.

Добар вовед во програмскиот јазик Питон во рамки на екосистемот за научна работа е даден во Скриптата за Сајпај (Varoquaux et al., 2015)[24] која е достапна под слободна лиценца[25]. Оваа книга претставува отворен проект и во неа, благодарејќи на многуте автори и придонесувачи, се поместени основите за работа не само со Питон, Нумпај, Матплотлиб и Сајпај, туку и Пандас, Симпај, Сајкитимиц, Сајкитлрн, па дури и Цајтон[26].

За запознавање со основите на програмскиот јазик Питон можат да послужат мноштво на материјали, каков што е официјалниот туторијал за Питон на вебстраната на Питон[27] и Викикнигата Програмирање во Питон[28], обете во употребува на МИТ.

[22] Добрата структурираност на просторите на имиња (namespaces) е една од важните одлики на Питон како што е и наведено во стиховите на Зенот на Питон кој може да го прочитате ако напишете `import this`.

[23] Автоматското надополнување се активира со притискање на `Tab`.

[24] Scipy Lecture Notes http://scipy-lectures.org/

[25] Creative Commons https://creativecommons.org/

[26] Cython https://cython.org/

[27] The Python Tutorial. https://docs.python.org/3/tutorial/index.html

[28] Python Programming https://en.wikibooks.org/wiki/Python_Programming

Литература

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006. URL https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.

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.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Ted Painter and Andreas Spanias. Perceptual coding of digital audio. *Proceedings of the IEEE*, 88 (4):451–515, 2000. URL http://www.cns.nyu.edu/~david/courses/perceptionGrad/Readings/PainterSpanias-ProcIEEE2000.pdf.

Lawrence R. Rabiner and Ronald W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall signal processing series. Prentice-Hall, 1978. ISBN 9780132136037.

Andreas Spanias, Ted Painter, and Venkatraman Atti. *Audio signal processing and coding*. John Wiley & Sons, 2006.

Gael Varoquaux, Valentin Haenel, Emmanuelle Gouillart, Zbigniew Jędrzejewski-Szmek, Ralf Gommers, Fabian Pedregosa, Olav Vahtras, Pierre de Buyl, Gert-Ludwig Ingold, Nicolas P. Rougier, and et al. scipy-lecture-notes: Release 2015.1 beta, 2015. URL http://www.scipy-lectures.org/.

Софија Богданов, Момчило и Богданова. *Дигитално процесирање на сигнали*. Електротехнички факултет, Скопје, 1997. ISBN 9989-630-15-1.