LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

SKAŅAS IERAKSTU SASPIEŠANAS ALGORITMS LAIKA TELPĀ AR ZUDUMIEM

MAĢISTRA DARBS

Autors: **Juris Evertovskis**

Stud. apl. je08011

Darba vadītājs: LU asoc. prof. Dr. Sc. Comp. Juris Vīksna

RĪGA 2015

Anotācija

Šajā darbā tiek piedāvāta metode skaņas saspiešanai ar zudumiem, nepārejot uz frekvenču telpu. Balstoties uz pieņēmumu, ka svarīgākā informācija par signālu ir tā ekstrēmos, tiek formulēts algoritms, kurš skaņas ierakstu pārveido saspiestā veidā, saglabājot informāciju tikai par tā lokālajiem ekstrēmiem un atspiežot signāls tiek rekonstruēts ar splainu interpolācijas metodēm. Noskaidrots, ka šāda signāla saglabāšana un rekonstruēšana patiešām ir iespējama un algoritms saspiež datnes kompaktāk nekā citi algoritmi, kas netransformē signālu frekvenču telpā. Rekonstruētās skaņas kvalitāte tika eksperimentāli novērtēta kā piemērota vairumam mūzikas un balss ierakstu.

Atslēgas vārdi: signāli, skaņa, saspiešana ar zudumiem, laika telpa

Abstract

A lossy audio compression method that does not involve transformations to frequency domain is presented in this work *Lossy audio compression algorithm in time domain*. Based on assumption that the main information about signal is found in its local extrema, we derive an algorithm that compresses audio recordings by saving data only about the extrema. Upon decompression the signal is restored using spline interpolation. It was found that it is really possible to save and reconstruct a signal using such method and the compression rate is better than for any other non-frequency-domain audio compression algorithm. The quality of reconstructed audio was experimentally found to be adequate for majority of voice and music recordings.

Keywords: signals, audio, lossy compression, time domain

Autoreferāts

Šī darba pamatā ir autora ideja par metodi, kas būtu izmantojama skaņas saspiešanā. Piedāvātā metode ir zudumradoša, bet atšķirībā no jau eksistējošām metodēm tā darbojas ar signālu laika telpā nevis frekvenču telpā. Ar literatūras avotu palīdzību tika izpētīti esošie skaņas saspiešanas algoritmi un tajos izmantotās metodes, kas ļāva saprast ar kādām metodēm apvienojama autora piedāvātā ideja, lai iegūtu skaņas saspiešanas algoritmu. Darbā tika izstrādāts gan algoritms un tā matemātiskais aparāts, gan arī šī algoritma realizācija programmas veidolā. Izmantojot izstrādātās programmas, algoritms tika arī pārbaudīts, novērtējot gan tā ātrdarbību un saspiešanas spēju, gan arī dzirdamās skaņas kvalitāti un tās atbilstību dažādiem pielietojumiem.

Saturs

Izmantotie apzīmējumi **3**

Ievads **5**

Skaņas saspiešana 5
Pētījuma tēma 6
Darba mērķis un uzdevumi 6
Darba struktūra 6

1. Esošo skaņas apstrādes un saspiešanas metožu apskats **7**

1.1. Digitālie skaņas ieraksti 7
1.2. Transformācijas frekvenču telpā 8
1.2.1. Furjē integrālā transformācija 9
1.2.2. Diskrētā kosinusu transformācija 9
1.2.3. Spektrogrammas 10
1.3. Bezzudumu saspiešanas paņēmieni 10
1.3.1. Hafmena kods 10
1.3.2. Lineārā prognozēšana 11
1.3.3. Kanālu korelācijas 12
1.3.4. Golomba kods 13
1.4. Metodes saspiešanai ar zudumiem 15
1.4.1. Amplitūdu izšķirtspējas samazināšana 15
1.4.2. Spektrālās izšķirtspējas samazināšana 16
1.4.3. Kanālu korelācija 17
1.5. Populārākie algoritmi 18
1.5.1. *FLAC* 18
1.5.2. *MP3* 18
1.5.3. *Vorbis* 20
1.6. Algoritmu kvalitātes novērtējums 20

2. Teorētiskā daļa **22**

2.1. Piedavātā jaunā metode 22
2.2. Saspiešanas algoritms—jaunās metodes apvienojums ar jau pazīstamām idejām 22
2.3. Signāla rekonstrukcija 25
2.3.1. Rekonstrukcija ar lineāru interpolāciju 26
2.3.2. Rekonstrukcija ar polinomiālu splainu interpolāciju 27
2.3.3. Rekonstrukcija ar trigonometrisku splainu interpolāciju 30

3. Praktiskā daļa **35**
3.1. Algoritma pamatidejas realizācija . . . 35
3.2. Pilna saspiešanas algoritma realizācija . . . 35
3.2.1. Saspiestās datnes formāts . . . 35
3.2.2. Programmas realizācija . . . 37

4. Eksperimenti un rezultāti **39**
4.1. Spektrogrammas . . . 39
4.2. Veiktspējas eksperimenti . . . 40
4.2.1. Saspiešanas pakāpe . . . 40
4.2.2. Ātrdarbība . . . 43
4.3. Kvalitātes eksperimenti . . . 45
4.3.1. Eksperimenta apraksts . . . 45
4.3.2. Eksperimenta rezultāti . . . 46

Novērtējums un secinājumi **49**
Saspiešanas spēja . . . 49
Ātrdarbība . . . 50
Skaņas kvalitāte . . . 50
Darbā paveiktais un galvenie secinājumi . . . 51
Iespējamie tālāko pētījumu virzieni . . . 51

A Algoritma realizācija ar *Wolfram Mathematica* **53**
A1. Saspiešana . . . 53
A2. Atspiešana . . . 53

B Algoritma realizācija ar *C++* **55**
B1. Saspiešanas programma *tdc-comp* . . . 55
B2. Atspiešanas programma *tdc-decomp* . . . 58

C Dažādu saspiešanas algoritmu rekonstruēto signālu attēli un spektrogrammas **64**
C1. Signālu attēli . . . 65
C2. Signālu spektrogrammas . . . 67

D Algoritma realizācijas ātrdarbības pārbaude **70**

E Rekonstruētā signāla kvalitātes testi **74**
E1. Paraugu apraksts . . . 74
E2. Eksperimenta detaļas . . . 76
E3. Eksperimenta rezultāti . . . 77

Izmantotā literatūra un avoti **79**

Izmantotie apzīmējumi

AAC

Uzlabota skaņas kodēšana (angliski—*Advanced Audio Coding*). Algoritms (standarts) skaņas saspiešanai ar zudumiem. Tiek izmantota datnēs ar paplašinājumu *.mp4*, *.m4a* un citās. [1, 2] [1]

C++

Objektorientēta programmēšanas valoda.

CRC

Cikliskā redundances pārbaude (angliski—*cyclic redundancy check*). Algoritms kontrolsummas aprēķinam, kas ļauj pārbaudīt vai datos nav parādījušās kļūdas. [3]

CSV

Komatatdalīto vērtību fails (angliski—*comma-separated values*). Datņu formāts, kur tabulārus datus saglabā teksta veidā, tabulas rindas atdalot atsevišķās teksta rindās un tabulas kolonnas atdalot ar kādu simbolu (bieži komatu, semikolu, tabulēšanas rakstzīmi).

DCT

Diskrētā kosinusu transformācija (angliski—*discrete cosine transform*). Diskretizēts Furjē transformācijas paveids, ko izmanto skaņas signālu apstrādē.

FLAC

Bezmaksas bezzudumu skaņas kodeks (angliski—*Free Lossless Audio Codec*). Skaņas saspiešanas algoritms, aprakstīts 1.5. nodaļā.

HTML5

HTML—hiperteksta iezīmēšanas valodas (angliski—*HyperText Markup Language*) piektā versija. [4]

Hz, kHz

Hercs, kilohercs. Frekvences mērvienība. $1Hz = 1s^{-1}$; $1kHz = 1000Hz$.

[1] Apzīmējumu saraksta dažiem jēdzienu skaidrojumiem pievienotas norādes uz avotiem, kur iegūstama plašāka informācija. Tie ir gan oficiālie standarti, gan enciklopēdiska rakstura avoti, kas bieži ir lasītājam pieejami vieglāk.

JPEG

Attēlu saspiešanas metode. [5]

LPCM

Lineārā impulskodu modulācijas sistēma (angliski—linear pulse-code modulation). Metode analogo signālu digitalizācijai, aprakstīta 1.1. nodaļā.

MIDI

Mūzikas instrumentu ciparsaskarne (angliski—Musical Instrument Digital Interface). Standarts mūzikas instrumentu un citu ierīču saziņai. Šajā darbā izmantotas *MIDI* standartam atbilstošas datnes par mūzikas fragmentu, kuras iespējams atskaņot, izmantojot operētājsistēmas *MS Windows* komplektācijā esošu programmatūru.

MP3

Skaņas saspiešanas algoritms, aprakstīts 1.5. nodaļā.

OGG

Multimediju konteiners—palīgdatu standarts datnēm ar paplašinājumu *.ogg*. Parasti skaņa *.ogg* datnēs ir saspiesta, izmantojot *Vorbis* standartu, tāpēc bieži (arī šajā darbā) *Vorbis*, *OGG* un *Ogg Vorbis* tiek izmantoti kā sinonīmi.

TDC

Šajā darbā piedāvātā algoritma realizācija, kas izstrādāta šī darba ietvaros.

Vorbis

Skaņas saspiešanas algoritms, aprakstīts 1.5. nodaļā. Darbā reizēm saukts arī par *OGG*, jo saspiestais signāls tiek saglabāts *OGG* standarta datnē ar *.ogg* paplašinājumu.

WAVE

Skaņas datņu standarts, parasti izmanto nesaspiestas skaņas saglabāšanai.

Wolfram Mathematica

Matemātiskā datorprogramma, kas izmantota šajā darbā.

ZIP

Bezzudumu datņu formāts. [6]

Ievads

Skaņas saspiešana

Digitālo datņu saspiešana tiek pētīta un attīstīta jau daudzas desmitgades. Aplūkojot datus kā neatkarīgu simbolu virkni, Deivids Hafmens jau 1952. gadā atrada veidu, kā pierakstīt šos datus binārā formā maksimāli īsi (un to pierādīja) [7]. Tagad šis pieraksts pazīstams kā Hafmena kods. Tomēr reālos datos parasti ir korelācijas—iespējamo simbolu sadalījums nav neatkarīgs no iepriekšējiem simboliem. Izmantojot šādas atkarības, ir iespējams datnes saglabāt vēl taupīgāk.

Šajā darbā tiek pētīta skaņas saspiešana un piedāvāts jauns risinājums. Aplūkosim skaņas ierakstu standartu, kas tiek izmantots kompaktdiskos [8]. Tiek saglabāti divi signāli (*labais* un *kreisais*), katrs signāls ir amplitūdu virkne ar 44100 vērtībām ik sekundi un amplitūda ir 16 bitos pierakstīts skaitlis. Tātad viena sekunde skaņas tiek fiksēta 176400 baitos, bet stunda mūzikas aizņem 635 040 000 baitus. Mūsdienās par pašsaprotamu tiek uzskatīta iespēja klausīties mūziku, kas saglabāta pārnēsājamos atskaņotājos, viedtālruņos un planšetdatoros, un šo ierīču atmiņas parasti ir mērāmas dažos gigabaitos vai dažos desmitos gigabaitu. Lai dažas stundas mūzikas neaizņemtu visu ierīces atmiņu, ir nepieciešama skaņas saspiešana.

Klasiskās datņu saspiešanas metodes parasti ļauj mūzikas ierakstus saspiest aptuveni par desmitdaļu. Izmantojot metodes, kas ņem vērā skaņas ierakstiem raksturīgās korelācijas, parasti izdodas samazināt izmantotās atmiņas apjomu par trešdaļu līdz pusi no sākotnējā apjoma. Būtiskāku saspiešanu izdodas panākt, izmantojot saspiešanu ar zudumiem—metodes, kas daļu informācijas nesaglabā, pieņemot, ka tā klausītājam nebūs nepieciešama. Ar šādām metodēm izdodas skaņas ierakstu datnes saspiest aptuveni desmitkārt.

Algoritmi skaņas saspiešanai ar zudumiem ir sarežģīti—informācijas atmešana tiek veikta frekvenču telpā, signālu iepriekš transformējot ar kādu no Furjē saimes transformācijām. Tas prasa vienlaicīgi apstrādāt lielu (parasti mērāmu simtos un tūkstošos) datu kopu.

Pētījuma tēma

Šajā darbā tiek izmantota autora izvirzīta hipotēze, ka būtiskākā informācija skaņas signālā ir šī signāla ekstrēma punkti (maksimumi un minimumi). Tiek piedāvāta ideja, kā nepārejot frekvenču telpā samazināt saglabājamās informācijas apjomu—jāsaglaba tikai signāla ekstrēmi. Darba ietvaros tiks pārbaudīts, pie kādiem rezultātiem var nokļūt, uzskatot izvirzīto hipotēzi par patiesu. Tiks realizēta šī metode un pētītas tās iespējas, priekšrocības un trūkumi.

Darba mērķis un uzdevumi

Darba mērķis: izpētīt un novērtēt autora piedāvāto skaņas ierakstu saspiešanas metodi—tajos saglabāt informāciju tikai par signālu ekstrēma punktiem.

Šī mērķa sasniegšanai izvirzīti šādi uzdevumi:

1. Veikt literatūras analīzi, izpētot jau pazīstamās metodes skaņas saspiešanā un aplūkojot konkrētus algoritmus.
2. Izstrādāt algoritmu skaņas ierakstu saspiešanai un atspiešanai, izmantojot piedāvāto ideju.
3. Realizēt izstrādāto algoritmu ar datorprogrammu un veikt eksperimentus algoritma novērtēšanai.

Darba struktūra

Darba pamatdaļa sākas ar esošo skaņas saspiešanas metožu apskatu, kurā izskaidrotas idejas, kas tiek izmantotas eksistējošos skaņas saspiešanas algoritmos, kā arī aprakstīti daži no pazīstamākajiem skaņas saspiešanas algoritmiem. Darba teorētiskajā daļā tiek noformulēta piedāvātā metode un tās kombinācija ar jau pazīstamām metodēm, veidojot skaņas saspiešanas algoritmu. Teorētiskajā daļā tiks definēti arī rezultātu analīzes kritēriji. Praktiskajā daļā aprakstīta algoritma realizācija—izveidotās programmas un to pielietošana, bet eksperimentālie rezultāti un to iegūšana iztirzāti atsevišķi rezultātu sadaļā.

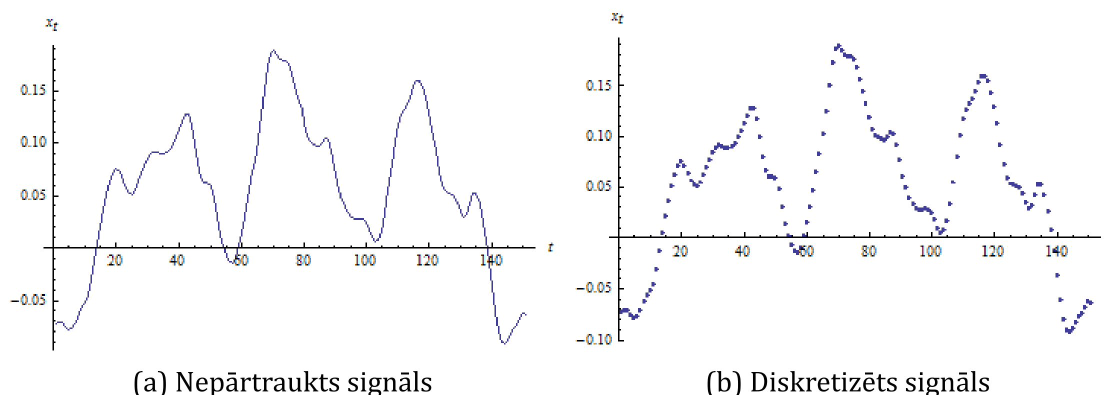
1. Esošo skaņas apstrādes un saspiešanas metožu apskats

1.1. Digitālie skaņas ieraksti

Fizikā par skaņu sauc mehāniskās svārstības, kas izplatās elastīgā vidē. Šī darba ietvaros mēs aplūkosim tikai cilvēkam dzirdamo skaņu—viļņus gaisā, kuru svārstību frekvence ir no 20 Hz līdz 20 kHz [9]. Fizikālie lielumi, kuri svārstās, ir gaisa molekulu koordinātes novirze no līdzsvara stāvokļa un gaisa spiediens. Ar atbilstošu sensoru palīdzību šīs svārstības (parasti spiediena svārstības) tiek pārveidotas elektriskās svārstībās un ierakstītas.

Par signālu mēs sauksim no laika atkarīgu funkciju $x(t)$ (skat. 1.1.a att.), kuras vērtības ir spiediens, spriegums vai cits nomērītais un piefiksētais lielums. Pētot patvaļīgus signālus, neinteresējamies par funkcijas $x(t)$ vērtību fizikālo izcelsmi un fizikālā lieluma nosaukumu katrā no situācijām. Tāpēc funkcijas $x(t)$ vērtību literatūrā mēdz dēvēt par momentāno amplitūdu, bet šajā darbā mēs to sauksim par *amplitūdu*.

Digitālos ierakstos signāli ir diskretizēti—amplitūdas tiek saglabātas ar noteiktu datu tipu, kas ierobežo amplitūdas iespējamo vērtību kopu līdz galīgai un amplitūdas tiek fik-



(a) Nepārtraukts signāls (b) Diskretizēts signāls

1.1. Att.: Signāls—amplitūdas atkarība no laika

sētas pēc noteiktiem (parasti nemainīgiem) laika intervāliem (skat. 1.1.b att.). Gadījumos, kad aplūkosim diskretizētu signālu, amplitūdu laika momentā (solī) t pierakstīsim kā x_t. Ja laika intervāli starp amplitūdām ir nemainīgi, varam tos raksturot ar *biežumu*—cik reizes sekundē fiksēta amplitūdas vērtība. Šo lielumu angliski sauc par *rate* vai *sampling rate* (tulkojot—temps, ātrums), bet latviski parasti izmanto nosaukumu iztveršanas frekvence, ko šajā darbā mēģināsim nelietot, rezervējot vārdu *frekvence* skaņas augstuma apzīmēšanai, kas parādās arī signālu spektrālajos attēlos. Digitālo skaņas signālu ierakstos parasti amplitūdu iespējamās vērtības ir vienmērīgos intervālos (nevis lielāka vai mazāka izšķirtspēja atkarībā no vērtības). Signāla pierakstu šādā veidā mēdz apzīmēt ar saīsinājumu *LPCM (linear pulse-code-modulation* [10].

Nesaspiestas skaņas ierakstu datnes visbiežāk sastopamas *WAVE* formātā (papalašinājums *.wav*). Šīs datnes sākas ar informāciju par signālu skaitu, laika intervāliem signālos un dažiem papilddatiem, kam seko LPCM signāli. Jāpiebilst, ka *WAVE* datnes var saturēt arī saspiestu signālu, taču praksē šis formāts tiek izmantots gandrīz tikai nesaspiestu signālu glabāšanai.

1.2. Transformācijas frekvenču telpā

Iepazīsimies ar dažiem jēdzieniem un matemātiskām operācijām—aplūkosim transformācijas, kas funkcijas attēlo frekvenču telpā. Tieši šajā reprezentācijā signāla frekvence un spektrs no abstraktiem, intuitīviem jēdzieniem kļūst par noteiktiem skaitliskiem lielumiem, kurus varam mērīt, salīdzināt un mainīt.

Šeit aprakstītā teorija ir pamatā vairumam pazīstamo zudumradošo saspiešanas metožu. Tiesa, šī nodaļa nepretendē uz pilnvērtīgu matemātiskā aparāta izklāstu, bet gan uz tā ieskicēšanu tādā līmenī, lai varētu izskaidrot dažādu esošu saspiešanas algoritmu darbības idejas, kā arī izmantot šeit ieviestos jēdzienus (frekvence, spektrs, spektrogramma) darba rezultātu analīzē.

1.2.1. Furjē integrālā transformācija

Funkcijas, kas atkarīgas no laika, iespējams attēlot tā sauktajā frekvenču telpā, izmantojot Furjē transformāciju [11]:

$$\hat{x}(\nu) = \int\limits_{-\infty}^{\infty} x(t)e^{-2\pi i\nu t}\mathrm{d}x \tag{1.2.1}$$

Mainīgo ν saucam par frekvenci, bet funkciju $\hat{x}(\nu)$ saucam par funkcijas $x(t)$ attēlu frekvenču telpā (arī par Furjē attēlu). Reizēm $\hat{x}(\nu)$ mēdz dēvēt par spektru (biežāk optikā, bet arī akustikā).

Aplūkojot tieši skaņas signālu, amplitūda $\hat{x}(\nu)$ raksturo to, cik daudz frekvence ν ir saklausāma signālā. Dažādas signālā saklausāmās frekvences mēdz dēvēt arī par harmonikām. Frekvenču telpu izmanto daudzos plaši pazīstamos saspiešanas algoritmos ar zudumiem, piemēram, *MP3*, *AAC*, attēlu saspiešanas algoritmā *JPEG* u.c. Tāpat Furjē attēls noder signālu analīzē, novērtējot manipulāciju ietekmi uz signālu.

1.2.2. Diskrētā kosinusu transformācija

Bieži praksē izmanto ne tikai Furjē transformāciju, bet arī citas līdzīgas transformācijas, kas attēlo funkciju frekvenču telpā. Lai attēls būtu reāls, $e^{-2\pi i\nu t}$ vietā izmanto sinusu vai kosinusu. Turklāt ņemsim vērā, ka strādājam ar diskrētām nevis nepārtrauktām funkcijām, tātad faktiski integrēšanas vietā jāsummē.

Signālu apstrādē parasti izmanto diskrēto kosinusu transformāciju, jo ar to iegūtaja attēlā galvenā informācija par signālu parasti ir mazākā frekvenču skaitā nekā sinusu transformācijas attēlā [12].

Šai transformācijai ir vairākas formas, no kurām populārākā ir *DCT-II*, kuru bieži pazīst arī kā vienkārši *DCT*. Šo transformāciju lietosim arī šajā pētījumā un tā ir šāda:

$$\hat{x}_\nu = \frac{1}{\sqrt{T}} \sum_{t=0}^{T-1} x_t \cos\left(\frac{\pi}{T}\left(t + \frac{1}{2}\right)\nu\right), \tag{1.2.2}$$

kur $\nu \in \{0, \ldots, T-1\}$ un T ir x_t punktu skaits.

Lai iegūtu sākotnējo funkciju, attēlam jāpielieto tā sauktā *DCT-III* jeb vienkārši apgriez-tā *DCT*:

$$x_t = \frac{1}{\sqrt{T}} \left(\hat{x}_\nu + 2 \sum_{\nu=1}^{T-1} \hat{x}_\nu \cos \left(\frac{\pi}{T} \nu \left(t + \frac{1}{2} \right) \right) \right), \tag{1.2.3}$$

kur $t \in \{0, \ldots, T-1\}$ un T ir $\hat{x}_\nu$ punktu skaits.

1.2.3. Spektrogrammas

Parasti skaņas signālā dzirdamās frekvences mainās atkarībā no laika, tāpēc arī spektrālo attēlu vēlamies redzēt atšķirīgu dažādiem mirkļiem nevis vienu visam signālam. Šādu attēlu sauc par spektrogrammu [13].

Lai iegūtu šādu attēlu, signāls jāsadala fragmentos, kas (salīdzinājumā ar visa signāla ilgumu) uzskatāmi par momentāniem un jāveic diskrētā Furjē transformācija katram fragmentam atsevišķi. Fragmentus mēdz sadalīt arī tā, ka tie daļēji pārklājas.

Bieži spektrogrammu veidošanai izmanto arī algoritmu, ko sauc par ātro Furjē transformāciju. Tomēr šajā darbā tiek izmantota parastā Furjē transformācija (protams, diskrētām vērtībām) un spektrogrammā tiek attēlotas amplitūdu absolūtās vērtības.

1.3. Bezzudumu saspiešanas paņēmieni

Šajā nodaļā aplūkosim bezzudumu informācijas saspiešanas metodes, kuras pēc iespējas tiek apvienotas gan ar zudumu, gan bezzudumu saspiešanas idejām, lai iegūtu algoritmus ar iespējami labu saspiešanas spēju.

1.3.1. Hafmena kods

Hafmena kods [7] ir izmantojams jebkuras simbolu virknes pierakstam. Diemžēl, skaņas datņu saspiešana ar šo metodi nav pārāk pietiekami efektīva—ar Hafmena kodu realizējošām programmām, piemēram, *ZIP* saspiešanu, var pārliecināties, ka *WAVE* datņu apjoms tiek samazināts tikai par aptuveni desmito daļu, bieži pat mazāk.

Autors atļaujas pieņemt, ka lasītājam Hafmena kods ir pazīstams, taču lasītājiem, kas ar algoritmiem nav tik tuvu pazīstami, ieskicēsim Hafmena koda pamatidejas. Dati ir simbolu virkne, kur simboli nāk no noteiktas iespējamo simbolu kopas (alfabēta) un katram

simbolam ir noteikts (vai izmērāms) parādīšanās biežums (jeb varbūtība). Hafmena kods biežākos simbolus kodē īsākās bitu virknēs nekā tos, kas parādās retāk.

Piemēram, ja dota simbolu virkne ar burtiem A, B un C, kur A parādās ievērojami biežāk nekā parējie simboli, tad pierakstīsim simbolus ar šādiem binārajiem vārdiem: $A = 0$, $B = 10$, $C = 11$.

Ņemot vērā Hafmena koda pamatideju un rezultātus, saspiežot skaņas datnes ar Hafmena kodu realizējošām programmām, varam secināt, ka dažādu simbolu (vērtību) parādīšanās signāla ir gandrīz vienlīdz varbūtīga. Tomēr aplūkojot reālu signālu (skat. 1.1.b att.) redzam, ka situācija ir mazliet niansētāka—pat, ja pie nejauša i amplitūdas x_i var būt jebkāda ar līdzvērtīgu varbūtību (par to liecina eksperimenti ar Hafmena kodu), tad lokāli atkarība un korelācija starp x_i un tam netāliem punktiem x_{i+k} ir acīmredzama (k—nosacīti neliels pozitīvs vai negatīvs vesels skaitlis).

1.3.2. Lineārā prognozēšana

Par autoregresīvu procesu $AR(p)$ sauc tādu procesu, kurā vērtība x_i ir lineāri atkarīga no iepriekšējām vērtībām[14]:

$$x_i = a_0 + \sum_{k=1}^{p} a_k x_{i-k} + \epsilon_i, \tag{1.3.1}$$

kur a_k—autoregresīva modeļa parametri un ϵ_i—stohastiskā komponente jeb troksnis.

Signālu analīzē parasti izmanto vienkāršāku modeļa specgadījumu, kur $a_0 = 0$.

Lineārā prognozēšana (*linear prediction*) [15] ir metode, kurā izvirzām hipotēzi, ka signāls ir autoregresīvs process un tāpēc varam izvirzīt vērtības x_i tuvinātu novērtējumu $\hat{x}_i$, ja zināmas iepriekšējas:

$$\hat{x}_i = \sum_{k=1}^{p} a_k x_{i-k} \tag{1.3.2}$$

Par novērtējuma kļūdu saucam $\epsilon_i = \hat{x}_i - x_i$.

Šī pieeja ļauj mums savādāk pierakstīt signālu—ir iespējams rekonstruēt signālu, ja ir zināmas pirmās x_i vērtības ($i \in \{1, \ldots p\}$) un visas kļūdu vērtības ϵ_i, kur $i > p$. Ja signāls patiešām izrādās autoregresīvs process, tad nelielas ϵ_i vērtības būs sastopamas biežāk

nekā lielas, tātad šo vērtību virkni būs iespējams saglabāt efektīvāk, izmantojot Hafmena kodu.

Lai atrastu modeļa parametrus a_k, jārisina vienādojumu sistēma

$$\sum_{i=1}^{p} a_k R_{j-k} = -R_j, \tag{1.3.3}$$

kur $j \in \{1, \ldots p\}$ un R_j ir autokorelācijas koeficienti

$$R_j = M[x_i x_{i-j}], \tag{1.3.4}$$

kur $M[\xi]$—empīriskā matemātiskā cerība [15]. Parametru skaitu p gan jāizraugas iepriekš vai jāatrod eksperimentālā ceļā.

1.3.3. Kanālu korelācijas

Skaņas ierakstu datnēs parasti ir nevis viens signāls (t.s. *mono* ieraksts), bet gan vairāki. Signālus, kas laikā tiek atskaņoti vienlaicīgi, mēdz saukt par kanāliem. Korelācijas starp kanāliem tiek izmantotas, lai samazinātu saglabājamās informācijas apjomu [16].

Idejas aprakstīsim, aplūkojot divu kanālu gadījumu—*stereo* ierakstu, kurā ir labais kanāls l_i un kreisais kanāls k_i. No šiem kanāliem veidojam divus agregātkanālus, kurus saglabājam sākotnējo vietā:

$$v_i = \frac{l_i + k_i}{2} \qquad m_i = \frac{l_i - k_i}{2} \tag{1.3.5}$$

Signālu v_i saucam par vidējo kanālu, bet m_i—par malējo. Ja līdzība starp signāliem patiesi ir bijusi, kanāla v_i vērtības būs proporcionālas sākotnējiem kanāliem, bet kanāla m_i vērtības būs tuvināti konstantas, bieži nelielas, kas ļaus šim kanālam efektīvāk pielietot Hafmena idejas.

Sākotnējos kanālus iespējams rekonstruēt šādi:

$$l_i = v_i + m_i \qquad k_i = v_i - m_i \tag{1.3.6}$$

1.3.4. Golomba kods

Uzmanīgs lasītājs būs pamanījis, ka 1.3.2. un 1.3.3. apakšnodaļās iegūtās vērtību rindas piedāvāts saspiest nevis tieši ar Hafmena kodu, bet izmantots vispārīgāks apzīmējums *Hafmena idejas*.

Iepriekš aprakstīto manipulāciju rezultātā tiek iegūtas rindas, kurās mazās vērtības parādās ievērojami biežāk nekā lielās, tātad sadalījums ir tuvs ģeometriskajam sadalījumam [17]. Lai izveidotu Hafmena kodu tabulas, vajadzētu izveidot statistiku par visu vērtību parādīšanās biežumiem, turklāt katrai datnei atsevišķi (vai tad nebūtu izšķērdīgi izmantot vienu tabulu visiem, ja kādā datnē neparādās pat puse no vērtībām, kam piešķirti koda vārdi?). Parasti skaņas saspiešana izmanto citu pieeju—Golomba kodu[18].

Aplūkosim nenegatīvu vērtību virkni, kur vērtības ir ģeometriski sadalītas:

$$P(x_i = k) = (1-p)^k p \tag{1.3.7}$$

Golomba koda ideja—novilkt robežu starp mazajām vērtībām, kas parādās bieži, un lielajām vērtībām. Izraugāmies šo robežu—parametru (veselu skaitli) M. Izmantojot parametru un kodējamo vērtību x_i, aprēķinām divus skaitļus:

$$q_i = \left\lfloor \frac{x_i}{M} \right\rfloor \qquad r_i = x_i - q_i M \tag{1.3.8}$$

Skaitlis q_i parasti būs mazs, visbiežāk—nulle, tāpēc to saglabāsim unārā pierakstā un galā pievienosim ’0’ (ja unārajam pierakstam izmantojām vieniniekus). Ieviesīsim skaitli b:

$$b = \lceil \log_2 M \rceil \tag{1.3.9}$$

Ja $r_i < 2^b - M$, atvēlam $b-1$ bitus, lai saglabātu r_i binārajā kodā. Ja nē, tad izmantosim b bitus un tajos saglabāsim skaitli $r_i + 2^b - M$.

Piemērs. Izvēlēsimies $M = 12$ un atradīsim koda vārdus skaitļiem $x_1 = 3$, $x_2 = 10$, $x_3 = 20$.

$q_1 = \left\lfloor \frac{3}{12} \right\rfloor = 0$; $q_2 = \left\lfloor \frac{10}{12} \right\rfloor = 0$; $q_3 = \left\lfloor \frac{20}{12} \right\rfloor = 1$

Tātad koda vārdu pirmās daļas būs: ’0’, ’0’ un ’10’.

$r_1 = 3 - 0 \cdot 12 = 3$; $r_2 = 10 - 0 \cdot 12 = 10$; $r_1 = 20 - 1 \cdot 12 = 8$

$b = \lceil \log_2 12 \rceil = 4$ un $2^b - M = 4$

$r_1 < 4$, tāpēc saglabāsim r_1 ar $b - 1 = 3$ bitiem: '011' un x_1 atbilstošais koda vārds ir '0011'.

$r_2 \geq 4$, tāpēc saglabāsim $r_3 + 2^b - M = 14$ ar $b = 4$ bitiem: '1110' un x_2 atbilstošais koda vārds ir '01110'.

$r_3 \geq 4$, tāpēc saglabāsim $r_3 + 2^b - M = 12$ ar $b = 4$ bitiem: '1100' un x_3 atbilstošais koda vārds ir '10110'.

Piemērs. Dots $M = 12$. Atšifrēsim pirmo skaitli bitu virknē '1100101101..'

Skaitām vieniniekus līdz pirmajai nullei—tie ir divi, tātad $q = 2$.

Nākamais simbols ir '0'. Zinām, ka pie $M = 12$ atlikumus, kas ir ≥ 4 pieraksta formā $4 + r \geq 8$, izmantojot četrus bitus, tātad pirmais bits būtu 1. Secinām, ka šajā gadījumā ir *mazais* atlikums un atšifrējam trīs bitu virkni '010': $r = 2$.

Tātad $x = qM + r = 26$.

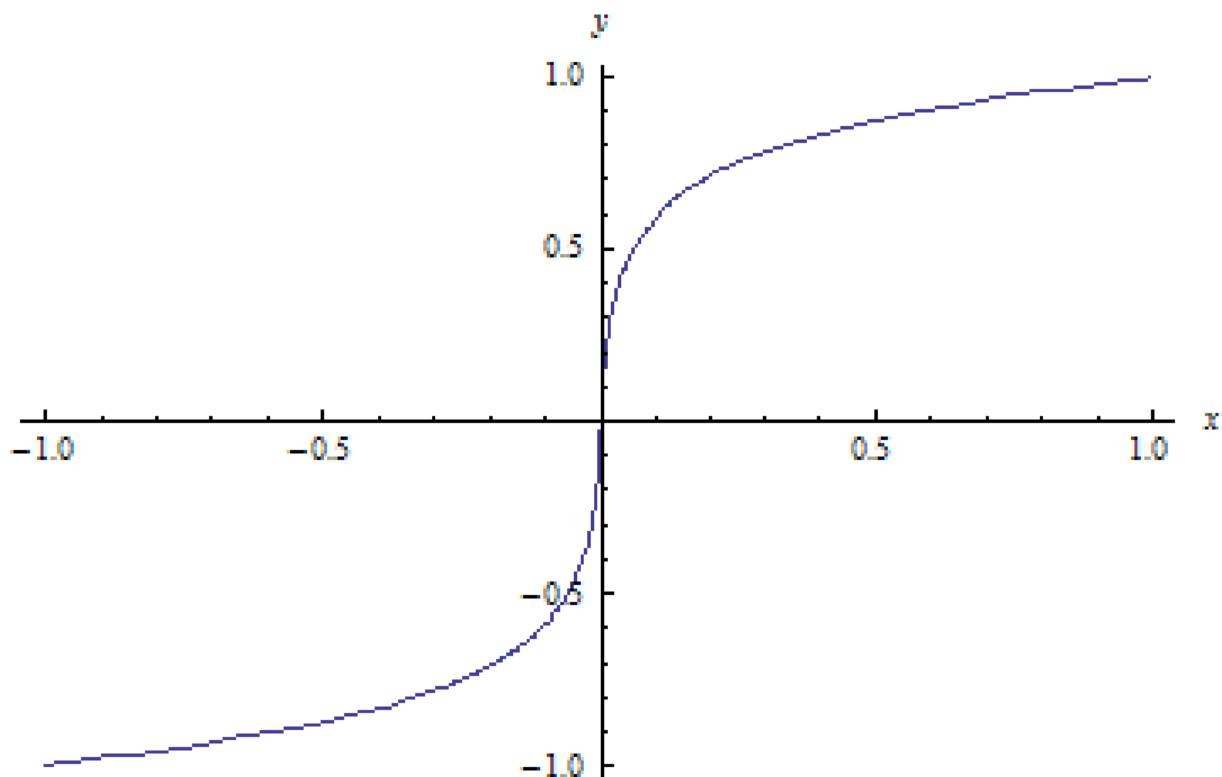
Kā izraudzīties parametru M? Ja vērtību sadalījums atbilst ģeometriskajam un ir zināms tā parametrs p, tad M jāapmierina šādu nevienādību:

$$(1-p)^M + (1-p)^{M+1} \leq 1 < (1-p)^{M-1} + (1-p)^M, \tag{1.3.10}$$

tad Golomba kods ir optimāls un ekvivalents Hafmena kodam. Bet, ja par sadalījumu nav tik daudz informācijas, parametru M jāmeklē eksperimentālā ceļā.

Kaut arī Golomba kods paredzēts nenegatīvām vērtībām, viegli varam to koriģēt veselo skaitļu kodēšanai, ja pievienojam zīmes bitu, piemēram, pēc '0', kas noslēdz unāro daļu.

Praksē bieži izmanto arī Golomba-Raisa kodu—Roberts Raiss, izstrādājot adaptīvu kodu [19], izmantoja Golomba kodu, bet izvēlējas tikai tādas M vērtības, kas ir 2 pakāpe. Tādējādi tiek iegūta vienkāršāka realizācija—ikviens atlikums r_i tiek bez nobīdes saglabāts binārā kodā, izmantojot $b = \log_2 M$ bitus. Arī dalīšana un atlikuma atrašana ar dalītāju $M = 2^b$ binārajā aritmētikā ir vienkāršāka nekā ar patvaļīgu M.



1.2. Att.: μ-likuma transformācija

1.4. Metodes saspiešanai ar zudumiem

1.4.1. Amplitūdu izšķirtspējas samazināšana

Telekomunikācijās tiek izmantoti A-likuma un μ-likuma algoritmi. Abi standarti ir līdzigi (viens tiek izmantots Eiropā, otrs—Amerikā un Japānā). Aplūkosim μ-likuma algoritmu.

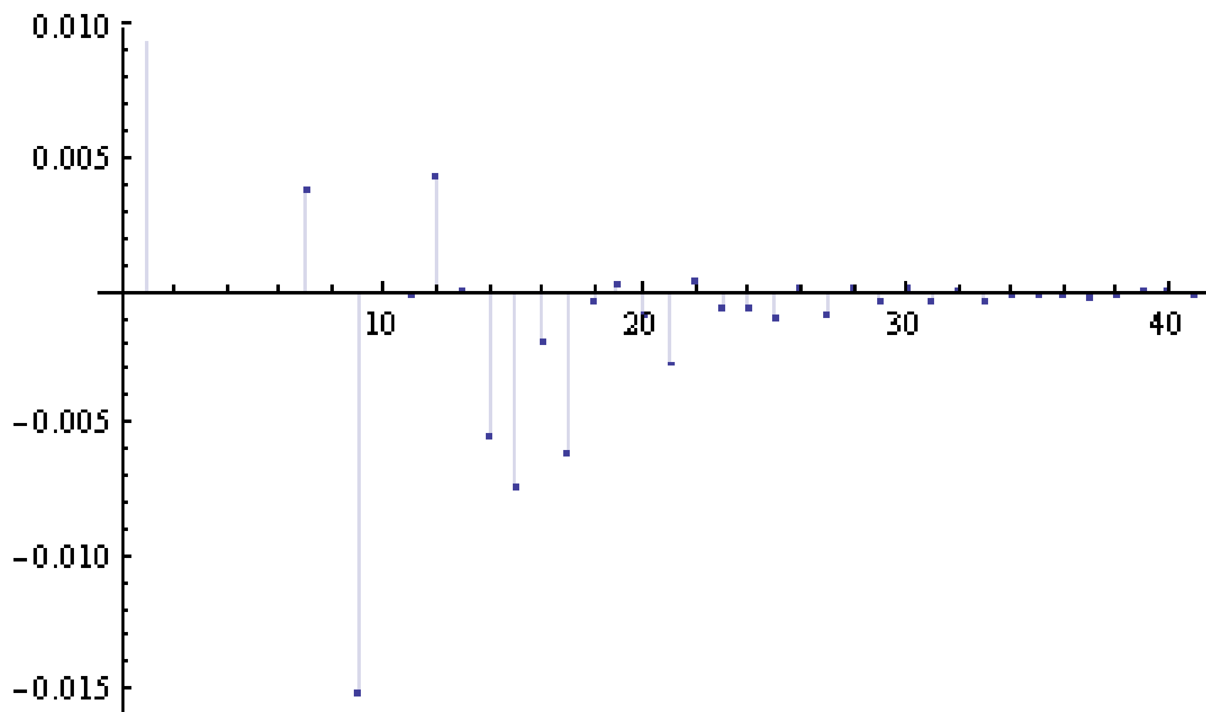
Pēc μ likuma vērtību $-1 \leq x \leq 1$ transformē šādi:

$$y = (sgn)(x) \frac{\log(1 + \mu|x|)}{\log(1 + \mu)}, \tag{1.4.1}$$

kur $\mu = 255$.

Analogā gadījumā to var interpretēt kā amplitūdu mērogošanu—lielākas amplitūdu vērtības tiek padarītas savstarpēji tuvākas, mazākās amplitūdas izšķirtspēja uzlabojas (skat. 1.2. att.). Tas telekomunikācijās ir lietderīgi, jo palīdz izšķirt klusākos signālus no trokšņa, tā vietā ietaupot izšķirtspēju lielākos skaļumos.

Bet aplūkojot lineāri diskretizēta signāla pārveidošanu diskretizācijā pēc μ-likuma [20] iegūstam saspiešanu ar izšķirtspējas samazināšanu.



1.3. Att.: Signāla spektrālais attēls

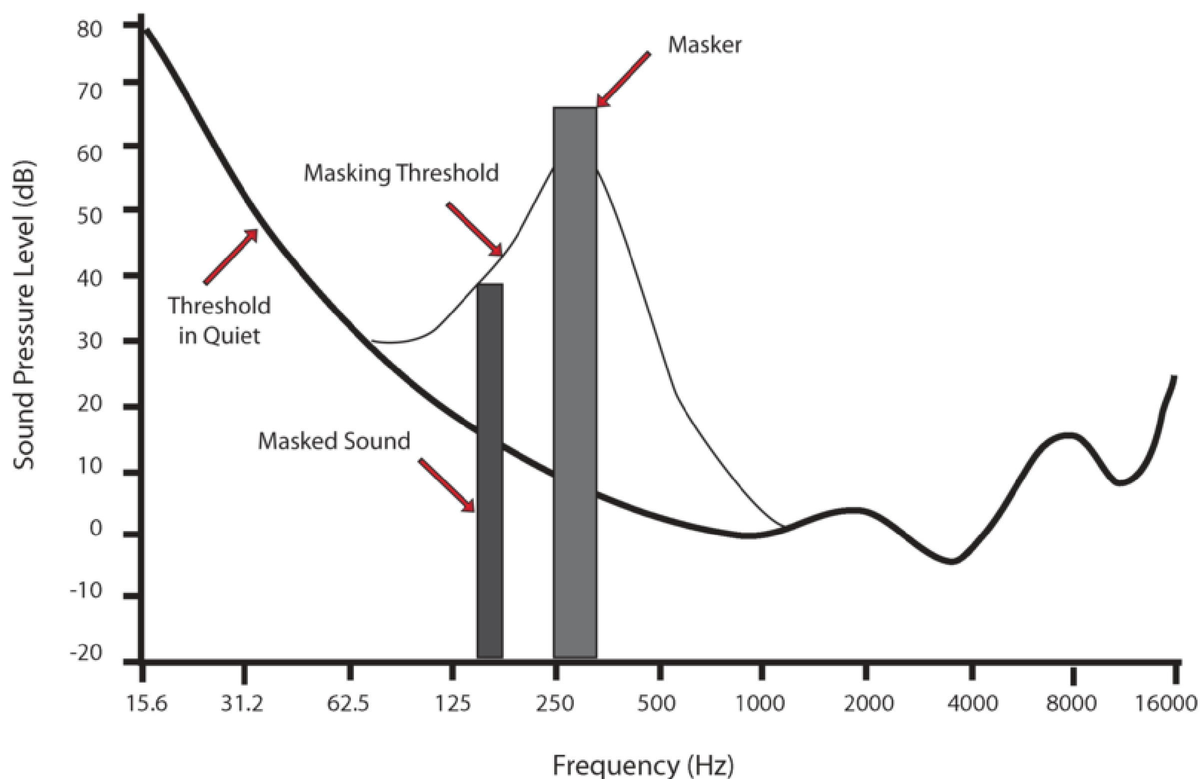
Sadalām iespējamo amplitūdu kopu apagabalos: 0-31, 31-95, 95-223, 223-479, ...4063-8159. Katru apgabalu sadalām 16 vienlielos intervālos (t.i. lielākajā apgabalā katrā intervālā ietilpst 256 vērtības, priekšpēdēja—128 utt.). Par katru amplitūdu saglabājam tikai informāciju, kurā apgabalā un kurā intervālā tā ietilpa. Tādējādi samazinām izšķirtspēju, it īpaši ietaupot lielo amplitūdu apgabalos. Samazinot izšķirtspēju ar μ-likuma algoritmu, tiek pāriets no 14 bitu amplitūdām uz 8 bitu amplitūdām, tādējādi samazinot nepieciešamās atmiņas apjomu līdz $\frac{8}{14}$ no sākotnējā.

Praktiski šādus algoritmus izmanto tikai balss pārraidei telekomunikācijā, bet mūzikas ierakstu saspiešanai tos nemēdz lietot. Tomēr tie ir interesanti ar to, ka datu apjomu samazina, nepārejot uz frekvenču telpu. Visi pārējie literatūras izpētē atrastie algoritmi un paņēmieni skaņas saspiešanai ar zudumiem, transformē signālu frekvenču telpā.

1.4.2. Spektrālās izšķirtspējas samazināšana

Intuitīvi varam aplūkot kāda signāla spektrālo attēlu (skat. 1.3. att.) un novērot, ka dažas frekvences ir krietni izteiktākas par citām. Varbūt mēs varam mazāk izteiktās frekvences uzskatīt par nebūtiskām un nesaglabāt?

Eksperimentu rezultātā ir izdevies aprakstīt, kādas frekvences nebūs dzirdamas citu frekvenču klātbūtne [21]. Šo efektu dēvē par frekvences maskēšanu. Tomēr sakarības nav



1.4. Att.: Maskēšanu ilustrējošs attēls. Treknā līnija—dzirdamības slieksnis (decibelos) klusumā atkarība no skaņas frekvences. Gaiši pelēkais stabiņš—skaļa skaņa frekvencē ap 300Hz. Plānā līnija—dzirdamības slieksnis skaļās skaņas klātesamībā. Tumši pelēkais stabiņš—klusāka skaņa, kas ir nedzirdama (maskēta) skaļākās skaņas klātbūtnē. [22]

ne tuvu lineāras, to apraksti ir sarežģīti un tos dēvē par psihoakustiskiem modeļiem (skat. 1.4. att.). Ņemot tos vērā, iespējams tiešām nesaglabāt *nedzirdamās* frekvences.

Maskēšana iespējama arī skaņām, kas nav vienlaicīgas, bet arī tad, ja tās seko īsi viena pēc otras. Skaļa skaņa var maskēt skaņas, kas seko sekundes desmitdaļu (un mazāk) pēc tās, kā arī skaņas, kas bijušas dažas sekundes simtdaļas pirms maskējošās skaņas [23].

1.4.3. Kanālu korelācija

Korelācijas starp paralēliem signāliem mēdz izmantot arī metodēs ar zudumiem.

Telpisko dzirdi cilvēkiem veido divi efekti: laika starpība starp abiem (labās un kreisās auss) signāliem un amplitūdu starpība starp tiem. Turklāt jutība uz laika starpību ir novērojama tikai zemās frekvencēs (līdz 2kHz) [24].

Ņemot vērā šos efektus, mēdz atsevišķus kanālus saglabāt tikai zemākajās frekvencēs, bet augstākās sapludina tos vienā kanālā, papildus saglabājot nelielus raksturlielumus (*virzienu*) telpiskuma rekonstrukcijai augstākās frekvencēs [25].

1.5. Populārākie algoritmi

1.5.1. *FLAC*

FLAC (*Free Lossless Audio Codec*) ir visplašāk izmantotais bezzudumu saspiešanas algoritms. Šis algoritms nav patentēts un tā implementācijas kods ir atvērts, turklāt ir pieejama arī algoritma dokumentācija [26].

Vispirms algoritms sadala signālu blokos. Pēc noklusējuma katrā blokā ir 4096 punkti, bet bloka izmērs ir maināms parametrs. Katrā blokā tiek meklēts datus aprakstošs modelis (iespējama gan lineārā prognozēšana, gan polinomiāls modelis). Novirzes no modeļa tiek kodētas ar Raisa kodu (Golomba-Raisa kodu ar adaptīvu—no datiem lokāli mainīgu—parametru M). *FLAC* var strādāt arī režīmā, kad novirzes pirms kodēšanas sadala komponentēs.

Saglabājot datnē, pirms katra bloka ir informācija (t.s. galvene) par šī bloka saglabāšanu. Tiek sarēķināta arī 8 bitu *CRC* kontrolsumma galvenei un 16 bitu *CRC* kontrolsumma visam kadram (blokam kopā ar papilddatiem).

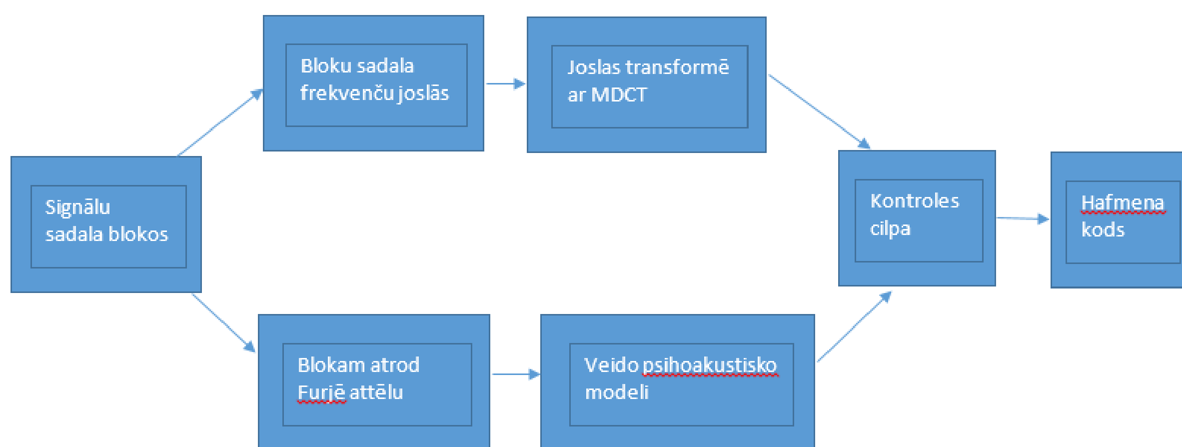
FLAC ņem vērā arī korelācijas starp kanāliem, turklāt ir iespējams (norādāms kā parametrs) tāds darbības režīms, ka katrā blokā tiek izvērtēts, vai izdevīgāk ir saglabāt labo un kreiso, vai vidējo un malējo kanālu.

1.5.2. *MP3*

MPEG-1 Audio Layer III un *MPEG-2 Audio Layer III*, plašāk pazīstami ar saīsinājumu *MP3* ir vispopulārākais skaņas saspiešanas algoritms. Tas ir patentēts standarts [27], kura dokumentācija nav pieejama bez maksas. Tomēr ir bezmaksas atvērtā koda bibliotēka *LAME* [28], kurā realizēts *MP3* standarts, un tieši ar *LAME* bibliotēku tiek sagatavota liela daļa no sastopamajām datnēm ar *.mp3* paplašinājumu.

Šis algoritms (skat. 1.5. att.) vispirms sadala signālu blokos pa 576 vērtībām (*MPEG-1* standarta kadri satur 1152 laika vienības, *MPEG-2* kadri ir 576 vienības kadri, tāpēc mazliet atšķiras, kā tiek saglabāti *MP3* bloki, bet pašā skaņas saspiešanā atšķirību nav).

Individuālu bloku ievada 32 frekvenču filtriem (sauktiem par *filter bank*), kas sadala signālu 32 vienlīdz platās frekvenču joslās. *MPEG Audio Layer I* un *Layer II* (*.mp2*) algoritmos šīm joslām piekārto svarus un samazina izšķirtspēju, ņemot vērā standartā doto psihoakustisko modeli.



1.5. Att.: *MP3* algoritma galveno soļu shematisks attēlojums

MP3 algoritmā šīs joslas ir starprezultāts, ko transformē frekvenču telpā ar modificēto diskrēto kosinusu transformāciju. Tā ir specifiska transformācija, kas paredzēta, lai transformētu apgabalus, kas pārklājas viens ar otru (līdz ar to nav strikti izteiktas robežas starp apgabaliem un katrs attēls nepieder pilnīgi atsevišķam apgabalam).

Paralēli sākotnējam blokam tiek veikta Furjē transformācija un konstruēts psihoakustiskais modelis, nosakot, kas tiks maskēts, kā arī pieļaujamo trokšņu un kropļojumu līmeni pie dažādām frekvencēm.

Psihoakustisko modeli un joslas dod kā ievaddatus kontroles cilpai. Vispirms joslām ir piekārtoti svari atbilstoši modelim, kas nosaka, ar cik bitiem tiks kodēts šīs joslas spektrālais attēls. Tas tiek nokodēts ar atvēleto bitu skaitu un kontroles cilpa tad analizē faktisko kropļojumu līmeni katrā joslā un salīdzina ar pieļaujamo. Ja tas kādā joslā ir virs pieļaujamā, šai joslai palielina svaru un atkārto procedūru (tātad tagad šīs joslas kodēšanai tiks izmantoti mazāk biti, bet citām joslām, kuru svari netika palielināti, atliks mazāk bitu). Šīs iterācijas tiek atkārtotas, līdz visās frekvenču joslās trokšņu līmenis ir gana zems vai ir atklāts, ka tas nav iespējams. Gadījumā, ja tas nav iespējams, *MP3* standarts liek lietot pēdējo iterāciju, savukārt *LAME* ar papildfunkciju meklē labāko iterāciju.

Iegūto rezultātu pēc tam saspiež, izmantojot Hafmena kodu. Tiesa, Koks netiek konstruēts—tiek izmantotas jau gatavas koda tabulas. Visbeidzot, nokodētajam kadram var pievienot divu baitu *CRC* kontrolsummu (bet standarts ļauj to arī nedarīt).

MP3 kodēšanā izmantoto sadalīšanu 32 joslās var uzskatīt par atavismu no *.mp2* un senākiem standartiem—jaunāki algoritmi, piemēram, *MPEG-4* iekļautais *AAC* paļaujas tikai uz modificēto diskrēto kosinusu transformāciju.

1.5.3. *Vorbis*

Atvērtā koda programmatūra, kas tēmēta uz spējīgākiem lietotājiem, parasti apiet *MP3* patentu, liekot lietotājam pašam lejupielādēt un pievienot programmai *LAME* bibliotēku. Komerciāli produkti tā parasti nedara, tāpēc ir jāpērk patents vai jāizvēlas brīvi pieejami standarti, no kuriem visbiežāk izmantotais algoritms skaņas saspiešanai ar zudumiem ir *Vorbis* [25]. To izmanto daudzās datorspēlēs un sākotnēji tas bija iekļauts arī *HTML5* specifikācijā kā ieteicamais formāts.

Arī šis algoritms izmanto modificēto diskrēto kosinusu transformāciju, bet iegūtos spektrālos attēlus tas aplūko, izmantojot modeli "troksnis (bāze) + novirzes". Būtiska atšķirība no *MP3* ir tā, ka šim algoritmam nav nedz noteiktu kadru, nedz konstanta bitu blīvuma uz laika vienību. Tā vietā šim algoritmam parasti norāda vēlamo kvalitāti (cik lielai jābūt līdzībai ar oriģinālo signālu) un algoritms izvērtē, kuras informācijas vienības drīkst atmest.

1.6. Algoritmu kvalitātes novērtējums

Svarīgākie kritēriji, pēc kuriem salīdzināt dažādus skaņas saspiešanas algoritmus, ir saspiešanas pakāpe (cik daudz atmiņas izdodas ietaupīt), algoritma ātrdarbība un skaņas kvalitāte (algoritmiem ar zudumiem). Skaņas kvalitāte ir tikai šim algoritmu veidam raksturīga metrika, tāpēc ir vērts to aplūkot.

Visbiežāk skaņas kvalitātes novērtēšanai izmanto *ABX* testu [29]. Tajā pārbauda, vai klausītāji spēj atšķirt paraugus A un B vienu no otra.

Tiek izvirzīta hipotēze $H_0 = A$ nav atšķirams no B.

Testa subjekti (klausītāji) noklausās paraugus A (oriģinālo signālu) un B (saspiesto signālu). Pēc tam klausītājam jānoklausās paraugs X (kas var būt A vai B) un jāpasaka, vai tas ir vairāk līdzīgs signālam A vai B.

Pēc tam testa rezultātus analizē ar statistiskām metodēm un vai nu apgāž vai neapgāž H_0.

Ir arī citas metodes, kas ļauj novērtēt skaņas kvalitāti kvantitatīvi, piemēram *MUSHRA* [30], taču šādi testi prasa, lai subjekti būtu pieredzējuši, profesionāli klausītāji—tajos pēc paraugu atpazīšanas tiek novērtēta arī to kvalitāte, piešķirot par to, piemēram, 0-100 punktus.

Var piebilst, ka psihoakustiskie modeļi parasti tiek veidoti, eksperimentos izmantojot ABX un tam līdzīgus testus. Piemēram, signāls A ir viena skaņa, bet signāls B—tā pati skaņa un vēl viena (klusāka skaņa). Ja klausītāji nespēj atšķirt šos signālus, tātad klusākā skaņa tiek maskēta.

2. Teorētiskā daļa

2.1. Piedavātā jaunā metode

Šajā darbā tika izvirzīta hipotēze: *svarīgākā informācija par skaņas signālu ir atrodama tā lokālo ekstrēmu (minimumu un maksimumu) punktos* (skat. 2.1. att.). Balstoties uz šo hipotēzi, tiek piedāvāta metode—signāla vietā saglabāt tikai informāciju par signāla ekstrēma punktiem—to pozīcijas laikā t_{ex} un to amplitūdu vērtības x_{ex}, bet pārējo signālu pēc vajadzības rekonstruēt ar interpolāciju. Intuitīvi šo metodi var saprast kā signāla saglabāšanu ar neregulāriem laika intervāliem, ņemot vērā pašu signālu—ja signāla oscilācijas ir straujākas, šajā apgabalā ir vairāk ekstrēma punktu, līdz ar to tiek saglabāts lielāks vērtību skaits, bet tur, kur signāla oscilācijas ir retākas, to var saglabāt, izmantojot mazāku atmiņas apjomu.

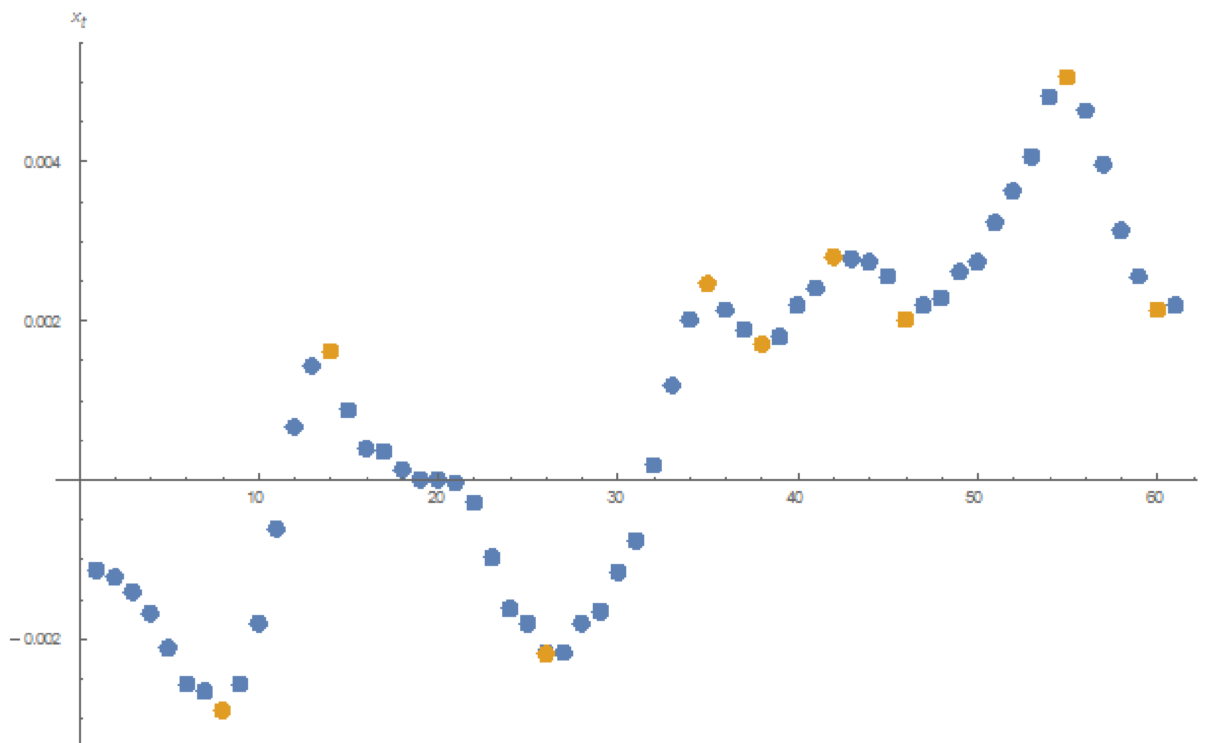
Lai no saglabātās informācijas atjaunotu signālu, ar interpolācijas palīdzību tiek ievietoti punkti starp saglabātajiem signāla ekstrēmiem. Sīkāk izmantotās rekonstrukcijas metodes iztirzāts 2.3. nodaļā.

Vēl viena ideja, ko izmantosim darbā - ņemot vērā, ka sākotnējie laika intervāli ir regulāri izvietoti, skaidrs, ka faktiski nav nepieciešams saglabāt ekstrēmu pozīcijas, bet pietiek saglabāt attālumus (veselos laika soļos) starp šīm pozīcijām.

2.2. Saspiešanas algoritms—jaunās metodes apvienojums ar jau pazīstamām idejām

Lai piedāvāto metodi pārbaudītu, jāizstrādā konkrēts algoritms, kas to izmanto.

Signāla ekstrēmus meklēsim, izejot cauri visiem signāla punktiem. Ja ekstrēms ir plato (t.i. nav viens lokālais ekstrēms, bet ekstrēms sastāv no vairākiem punktiem ar vienādu amplitūdu), tad par ekstrēma punktu fiksēsim to, pēc kura plato beidzas. Tātad, ja aplūko-



2.1. Att.: Signāls un tā lokālie ekstrēmi (iekrāsoti oranžā krāsā)

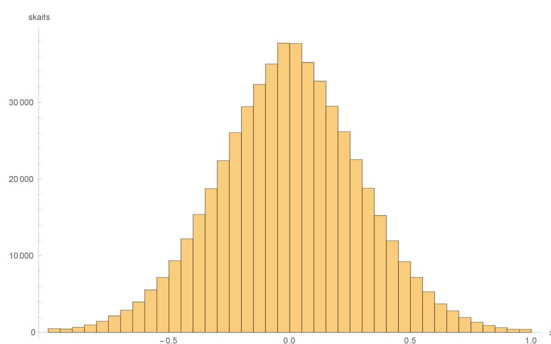
jam secīgas signāla amplitūdas, kas ir pieaugošā secībā, tad kā ekstrēma punktu fiksēsim to, pēc kura vērtības sāk dilt (jo tikai pie šī punkta kļūst skaidrs, ka šis stacionārais apgabals ir bijis ekstrēms).

Vienmēr saglabāsim pirmo un pēdējo amplitūdu. Ekstrēma punktiem saglabāsim nenegatīvu skaitli t—cik laika soļus pēc iepriekšējā saglabātā punkta ir šis ekstrēms. Šo vērtību kodēsim ar Golomba-Raisa kodu. Koda parametrs M tiks meklēts empīriskā ceļā.

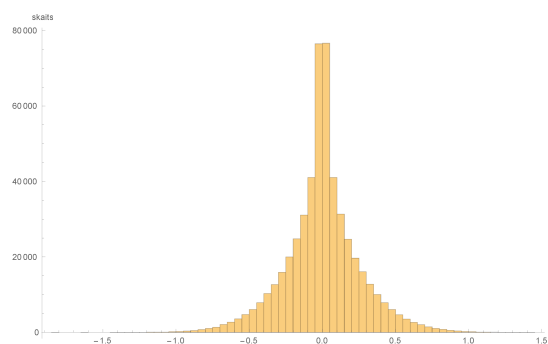
Pēc personīgiem novērojumiem darba autors izvirza hipotēzi: *sekojošu ekstrēmu amplitūdas parasti ir tuvas*. Ja tā ir patiesa, tad ekstrēmu virkni ir izdevīgi aprakstīt ar šādu autoregresīvu modeli:

$$x_i = x_{i-1} + \epsilon_i \tag{2.2.1}$$

Tas mums ļauj saglabāt tikai novirzes ϵ_i, kas, ja hipotēze ir patiesa, parasti ir mazas un saglabājamas efektīvāk. Patiesi, ja aplūko reāla mūzikas ieraksta amplitūdu (skat. 2.2.a att.) un noviržu (skat. 2.2.b att.) histogrammas, redzam, ka amplitūdu sadalījums ir tuvs normālajam sadalījumam, bet noviržu sadalījums ir līdzīgs divpusējam eksponenciālajam sadalījumam—tātad sadalījumam ar mazāku entropiju, kas ir vieglāk saspiežams.



(a) Ekstrēmālo amplitūdu sadalījums



(b) Amplitūdu starpību sadalījums

2.2. Att.: Amplitūdu ekstrēmu un to starpību sadalījums *Totenkopf* dziesmas *Burn The Desert* kreisajā kanāla

Ja papildus ņemam vērā, ka saglabājamie punkti ir secīgi lokālie ekstrēmi, tad skaidrs, ka starpības starp tiem ir ar mainīgu zīmi (pēc maksimuma vienmēr sekos minimums, kam būs mazāka amplitūda nekā nupat bijušajam maksimumam un otrādi), tad varam ieviest šādu modeli:

$$x_i = x_{i-1} + (-1)^i \epsilon_i, \tag{2.2.2}$$

kas nozīmē, ka nesaglabāsim arī ϵ_i zīmi. Saglabājamās ϵ_i vērtības saglabāsim, izmantojot Golomba-Raisa kodu, kura parametru M atradīsim empīriskā ceļā.

Tātad konspektīvi algoritms ir šāds:

1. Izvēlamies parametrus M_t un M_ϵ, kas tiks izmantoti laika intervālu un amplitūdu ekstrēmu noviržu Golomba-Raisa kodēšanai.

2. Saglabājam palīginformāciju par signālu (skat. 3.2.1. nodaļu) un pirmo amplitūdas vērtību.

3. Aplūkojam sekojošās amplitūdas (un skaitām aplūkotās), līdz atrodam kādu, kas ir mazāka vai lielāka par iepriekšējo. Ja tā ir mazāka, pārejam pie piektā punkta.

4. Pārlūkojam sekojošas amplitūdas (un skaitām aplūkotās), līdz atrodam kādu, kas ir mazāka par iepriekšējo. Ar Golomba-Raisa kodu saglabājam pārlūkoto vērtību skaitītāja rādījumu un priekšpēdējās aplūkotās amplitūdas un iepriekšējās saglabātās amplitūdas starpības absolūto vērtību. Iestādām pārlūkoto amplitūdu skaitītāju uz vērtību ’0’.

5. Pārlūkojam sekojošas amplitūdas (un skaitām aplūkotās), līdz atrodam kādu, kas ir lielāka par iepriekšējo. Ar Golomba-Raisa kodu saglabājam pārlūkoto vērtību skaitītāja rādījumu un priekšpēdējās aplūkotās amplitūdas un iepriekšējās saglabātās amplitūdas starpības absolūto vērtību. Iestādām pārlūkoto amplitūdu skaitītāju uz vērtību '0'.

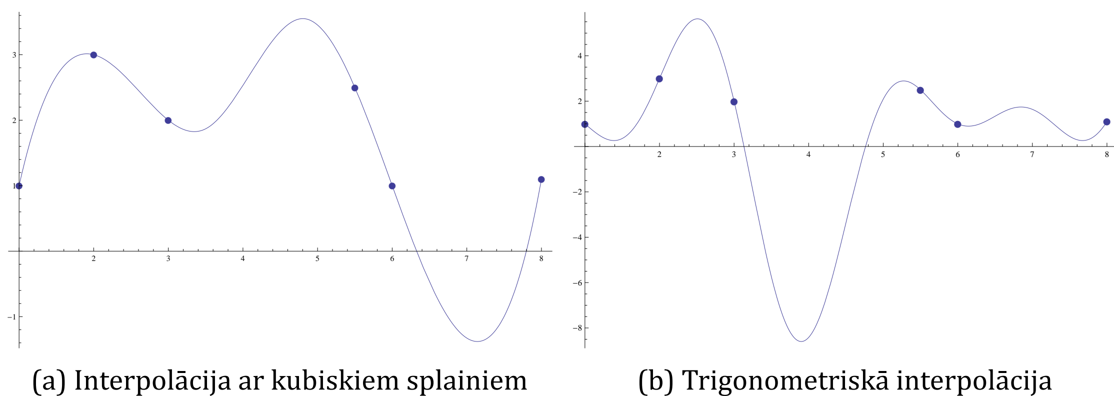
6. Ja kādā no iepriekšējiem soļiem sasniegtas signāla beigas, ar Golomba-Raisa kodu saglabājam pārlūkoto vērtību skaitītāju un iepriekšējās saglabātās amplitūdas un pēdējās amplitūdas starpības absolūto vērtību un beidzam darbu.

2.3. Signāla rekonstrukcija

Signāla atspiešanas jeb rekonstrukcijas procedūru tik sīki neaprakstīsim, jo vairums soļu ir saprotami, ja saprotams saspiešanas process. Atsevišķu uzmanību jāpievērš tikai trūkstošo vērtību rekonstrukcijai, kam tiks veltīta šī nodaļa.

Amplitūdu no ekstrēmu saraksta, līdz kurai signālu jau esam rekonstruējuši, apzīmēsim ar x_i, bet nākamo ekstrēmu—ar x_{i+n}, kur n—attālums laika soļos starp šīm amplitūdām.

Izmantojot klasiskas interpolācijas metodes, piemēram, interpolāciju ar kubiskiem splainiem un trigonometrisko interpolāciju, novērojama rekonstruētā signāla neatbilstība sagaidāmajam (skat. 2.3. att.).



(a) Interpolācija ar kubiskiem splainiem (b) Trigonometriskā interpolācija

2.3. Att.: Interpolācija starp signāla ekstrēmiem ar klasiskām interpolācijas metodēm.

Trigonometriskā interpolācija ir populāra skaņas signālu rekonstrukcijai un signāla vērtību biežuma (*sampling rate*) palielināšanai (*upsampling*) [31]. Trigonometriskā interpolācija tiek izmantota, jo tā neizmaina signālu spektrālo attēlu—faktiski trigonometriskā interpolācija ir signāla spektrālais (Furjē) izvirzījums. Tomēr, praktiski izmēģinot trigono-

metrisko interpolāciju, novērotas nevēlamas signāla svārstības (skat. 2.3.b att.). Ja zināms, ka visi dotie punkti ir ekstrēmi, tad skaidrs, ka interpolācija intervālā $(3; 5.5)$ ievērojami atšķiras no līknes, kādai jābūt starp blakus esošiem ekstrēmiem.

Lai novērstu interpolācijas *svārstīšanos* jeb Runges fenomenu [32], kas raksturīgs arī polinomiālajai interpolācijai (kurā meklē polinomu, kura vērtības zināmajos punktos sakrīt ar dotajām), parasti izmanto splainu interpolāciju. Splainu interpolācija starp katriem diviem zināmajiem punktiem funkcijas vērtības apraksta ar atsevišķu polinomu, kurus izvēlās tā, lai zināmajos punktos interpolējošās funkcijas pirmais un otrais atvasinājums būtu nepārtraukts. Izmēģinot šo interpolācijas metodi, novērotas ievērojami mazākas svārstības (skat. 2.3.a att.), tomēr joprojām redzams—ja zinām, ka dotie punkti ir signāla ekstrēmi, tad skaidrs, ka interpolējošā funkcija neatbilst patiesajai.

2.3.1. Rekonstrukcija ar lineāru interpolāciju

Vienkāršākā metode, ar kādu varam rekonstruēt signālu, ir lineāra interpolācija—blakus esošu punktu savienošana ar taisnēm. Ar šādu metodi arī tiek garantēts, ka iegūtās funkcijas ekstrēmi patiešām būs zināmajos ekstrēmu punktos.

Taisne, kas iet caur diviem punktiem (x_1, y_1) un (x_2, y_2), aprakstāma ar šādu vienādojumu [33]:

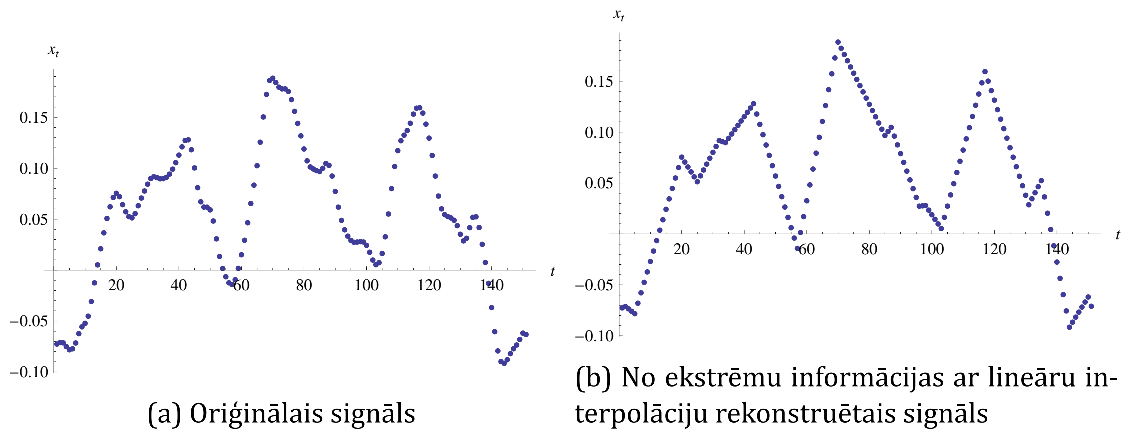
$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \tag{2.3.1}$$

Diskretizēsim interpolāciju amplitūdām, kas dotas vienmērīgos laika intervālos. Ja ar x_i un x_{i+n} apzīmējam divus blakus esošus dotos punktus, bet n ir laika soļu skaits starp šīm amplitūdām, tad amplitūdu x_k, kur $i < k < i + n$ varam tuvināti rekonstruēt ar šādas formulas palīdzību:

$$x_k = x_i + \frac{x_{i+n} - x_i}{n}(k - i) \tag{2.3.2}$$

Redzams, ka iegūtā interpolācija nav gluda (skat.2.4.a un 2.4.b att.)—lai arī tajā saglabāta informācija, ka dotie punkti ir ekstrēmi, tomēr atvasinājums mainās lēcienveidā.

Reāla skaņas signāla amplitūda atspoguļo kāda svārstoša punkta koordinātes, piemēram, membrānas pozīciju. Tātad signāla atvasinājums ir fizikāla objekta ātrums, kurš nevar mainīties lēcienveidā, jo lēcienveida ātruma izmaiņa nozīmētu bezgalīgu paātrinājumu, ta-



(a) Oriģinālais signāls

(b) No ekstrēmu informācijas ar lineāru interpolāciju rekonstruētais signāls

2.4. Att.: Signāla fragments no *Totenkopf* dziesmas *Burn The Desert* kreisā kanāla

ču bezgalīgam paātrinājumam saskaņā ar Ņūtona II likumu [34] nepieciešams bezgalīgs spēks.

2.3.2. Rekonstrukcija ar polinomiālu splainu interpolāciju

Ja izmantojam visu saglabāto informāciju—gan punktus, gan to, ka tie ir ekstrēma punkti, tad katrā no saglabātajiem punktiem mums īstenībā ir zināmi divi parametri—signāla un tā atvasinājuma vērtība. Mēģināsim konstruēt līknes, kas ņem vērā visu šo informāciju. Starp katriem diviem punktiem novilksim vienu polinomiālu līkni līdzīgi splainu interpolācijai un iepriekš izmantotajai lineārajai interpolācijai. Literatūrā polinomiālu splainu interpolāciju, kam uzdotas arī atvasinājumu vērtības, sauc par interpolāciju ar Ermita splainiem [35].

Interpolējošās funkcijas posmam $[x_1, x_2]$ doti četri robežnosacījumi:

$$y(x_1) = y_1; \tag{2.3.3a}$$

$$y'(x_1) = 0; \tag{2.3.3b}$$

$$y(x_2) = y_2; \tag{2.3.3c}$$

$$y'(x_2) = 0; \tag{2.3.3d}$$

Lai apmierinātu četrus robežnosacījumus, nepieciešams polinoms ar četrām brīvības pakāpēm, tāpēc izvēlēsimies trešās kārtas polinomu:

$$P(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \tag{2.3.4}$$

Šī polinoma atvasinājums:

$$P'(x) = 3a_3x^2 + 2a_2x + a_1 \tag{2.3.5}$$

Polinoms (2.3.5) ir kvadrātisks, tātad tam ir divas saknes. Robežnosacījumi (2.3.3b) un (2.3.3d) tieši nosaka divus punktus, kuros (2.3.5) vērtībai jābūt 0. Tātad jāatrod tādi koeficienti (a_3, a_2, a_1), lai polinoma (2.3.5) saknes sakristu ar šiem punktiem. Pārfrāzējot teiksim, ka vienādojuma

$$3a_3x^2 + 2a_2x + a_1 = 0 \tag{2.3.6}$$

saknēm jābūt x_1 un x_2. Zināms, ka kvadrātvienādojuma saknes izsakāmas ar šādu formulu:

$$x_{1,2} = \frac{-a_2 \pm \sqrt{a_2^2 - 3a_1a_3}}{3a_3} \tag{2.3.7}$$

Ievērojot, ka (2.3.7) zemsaknes izteiksme nevar būt negatīva un saucējs nevar būt 0 (citādi nebūs divu reālu sakņu), varam pārrakstīt (2.3.7) šādā formā:

$$3a_3x_{1,2}^2 + 2a_2x_{1,2} + a_1 = 0 \tag{2.3.8}$$

Tātad nosacījumi koeficientu (a_3, a_2, a_1, a_0) atrašanai ir šādi:

$$y_1 = a_3x_1^3 + a_2x_1^2 + a_1x_1 + a_0 \tag{2.3.9a}$$

$$0 = 3a_3x_1^2 + 2a_2x_1 + a_1 \tag{2.3.9b}$$

$$y_2 = a_3x_2^3 + a_2x_2^2 + a_1x_2 + a_0 \tag{2.3.9c}$$

$$0 = 3a_3x_2^2 + 2a_2x_2 + a_1 \tag{2.3.9d}$$

Vienādojumu sistēmas (2.3.9) atrisinājums ir šāds:

$$a_3 = \frac{2(y_2 - y_1)}{(x_1 - x_2)^3} \tag{2.3.10a}$$

$$a_2 = \frac{3(x_1 + x_2)(y_1 - y_2)}{(x_1 - x_2)^3} \tag{2.3.10b}$$

$$a_1 = \frac{6x_1x_2(y_2 - y_1)}{(x_1 - x_2)^3} \tag{2.3.10c}$$

$$a_0 = \frac{(3x_1 - x_2)x_2^2y_1 + (x_1 - 3x_2)x_1^2y_2}{(x_1 - x_2)^3} \tag{2.3.10d}$$

Veiksim koordinātu transformāciju, pārējot uz šādiem mainīgajiem—attālumu kopš pirmā punkta $\hat{x} = x - x_1$ un starpību no pirmās vērtības $\hat{y} = y - y_1$. Šajās koordinātes dotais punkts $(\hat{x}_1, \hat{y}_1) = (0, 0)$.

Tad koeficientu izteiksmes (2.3.10a) izsakāmas šādi:

$$a_3 = -2\frac{\hat{y}_2}{\hat{x}_2^3} \tag{2.3.11a}$$

$$a_2 = 3\frac{\hat{y}_2}{\hat{x}_2^2} \tag{2.3.11b}$$

$$a_1 = 0 \tag{2.3.11c}$$

$$a_0 = 0 \tag{2.3.11d}$$

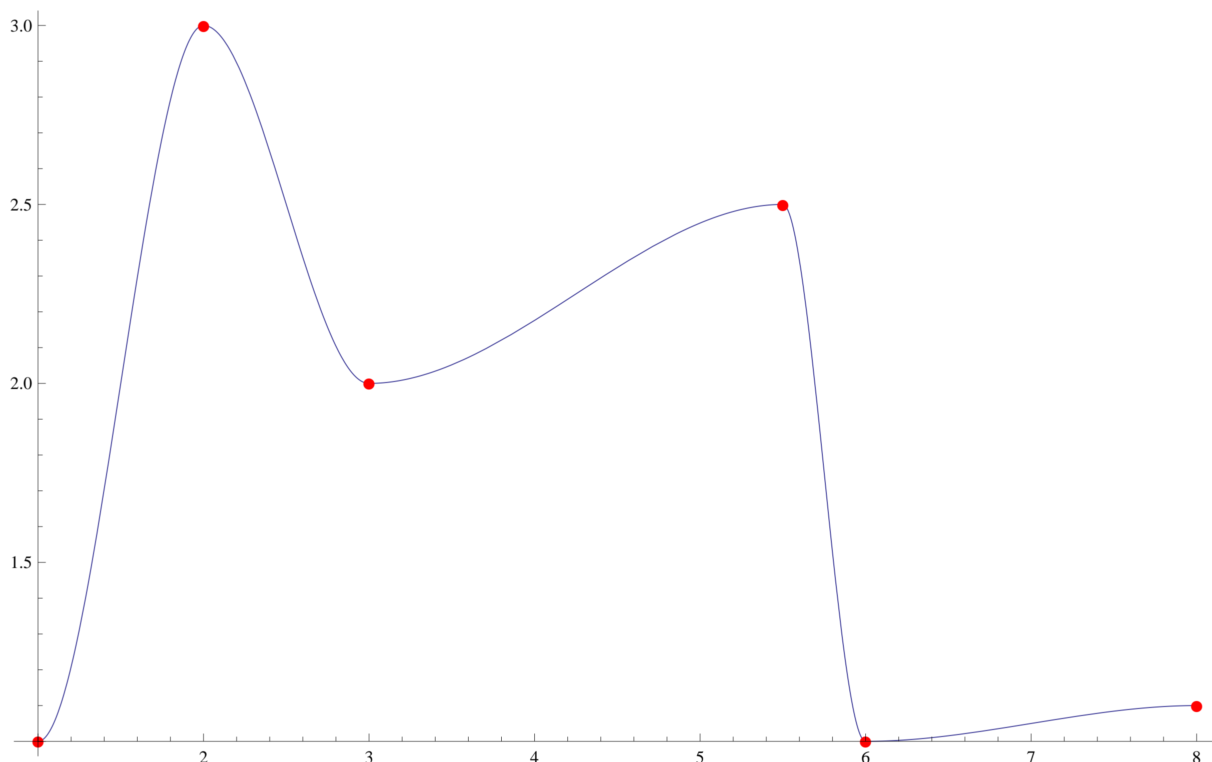
Turpmāk strādāsim šajās koordinātēs un jumtiņus nerakstīsim. Izmantojot koeficientus formā (2.3.11a), interpolējošā funkcija uzrakstāma šādā formā:

$$y = -2\frac{y_2}{x_2^3}x^3 + 3\frac{y_2}{x_2^2}x^2 = \frac{y_2}{x_2^3}(-2x^3 + 3x_2x^2) \tag{2.3.12}$$

Vizuāli šī interpolācija apskatāma 2.5. attēlā.

Diskretizēsim šo interpolāciju amplitūdām, kas dotas vienmērīgos laika intervālos. Tāpat kā iepriekš ar x_i un x_{i+n} apzīmējam divus blakus esošus dotos punktus, bet n ir laika soļu skaits starp šīm amplitūdām. Amplitūdu x_k, kur $i < k < i + n$ aprēķinām ar šādu izteiksmi:

$$x_k = x_i + \frac{x_{i+n} - x_i}{n^3}(-2k^3 + 3nk^2) \tag{2.3.13}$$



2.5. Att.: Polinomiālā splainu interpolācija ar nosacījumu $y' = 0$ dotajos punktos.

2.3.3. Rekonstrukcija ar trigonometrisku splainu interpolāciju

Līdzīgi varam punktus savienot arī ar trigonometrisku funkciju fragmentiem. Izvēlēsimies kosinusu (kosinuss no sinusa atšķiras ar konstantu nobīdi, tāpēc iegūtas formulas būtu analogas).

Kā jau 2.3.2. nodaļā aplūkojām, lai apmierinātu četrus robežnosacījumus (2.3.3) nepieciešamas četras brīvības pakāpes. Tāpēc interpolējošo funkciju meklēsim šādā formā:

$$f(x) = a\cos(\omega x + \phi) + c \tag{2.3.14}$$

Funkcijas (2.3.14) atvasinājums:

$$f'(x) = -a\omega\sin(\omega x + \phi) \tag{2.3.15}$$

Šai funkcijai ir bezgalīgi daudz sakņu, taču mūsu vajadzībām jāizvēlas divas blakus esošas saknes (kas atbilst diviem blakus esošiem (2.3.14) ekstrēmiem). Zinot sinusa īpašības, attālums starp secīgām saknēm x_1 un x_2 atrodams no šādas sakarības sinusa argumentam:

$$(\omega x_2 + \phi) - (\omega x_1 + \phi) = \pi \tag{2.3.16}$$

Tas mums ļauj atrast, ka

$$\omega = \frac{\pi}{x_2 - x_1} \tag{2.3.17}$$

Izmantojot robežnosacījumu $f'(x_1) = 0$ un izvēloties punktā x_1 kosinusa maksimumu (ω atradām tādu, ka minimums tad būs pie x_2), iegūstam

$$\omega x_1 + \phi = 2\pi N, \quad N \in \mathbb{Z} \tag{2.3.18}$$

No kā varam izteikt

$$\phi = \frac{\pi x_1}{x_1 - x_2} + 2\pi N, \quad N \in \mathbb{Z} \tag{2.3.19}$$

Zinot, ka kosinusa minimālā un maksimālā vērtība atšķiras par 2, varam atrast, ka

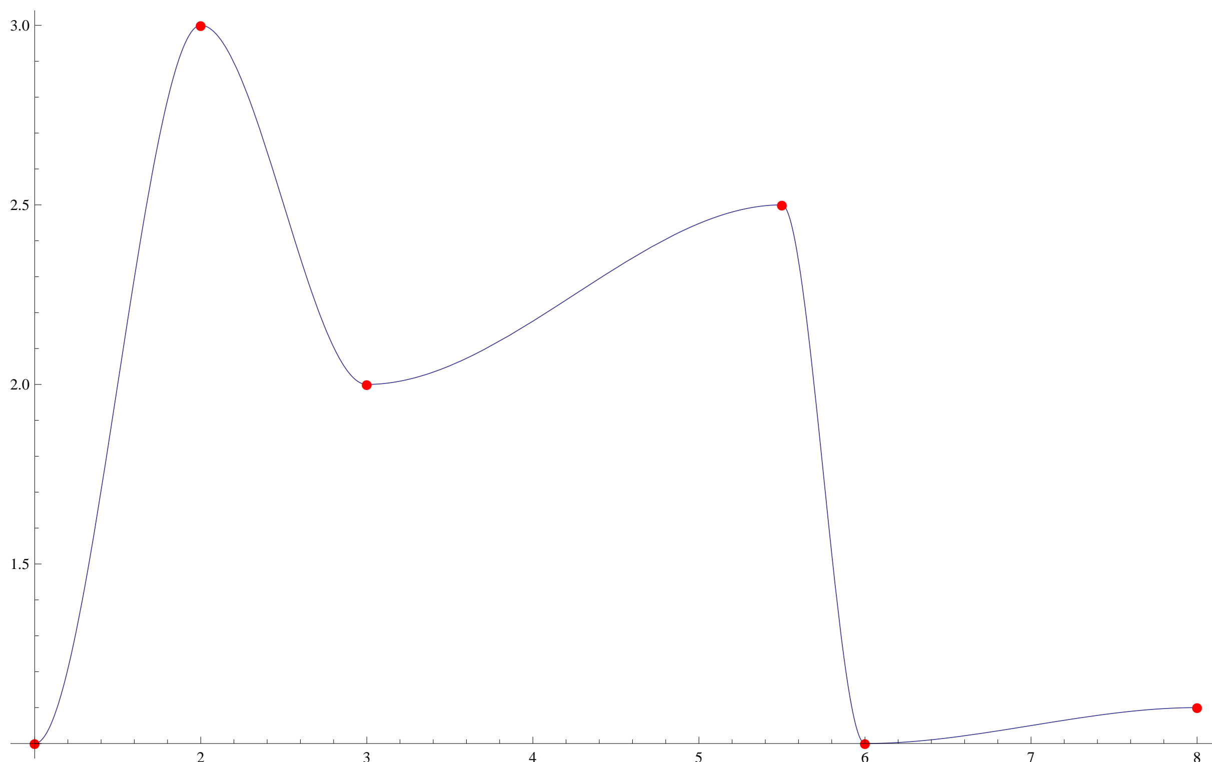
$$a = \frac{y_1 - y_2}{2} \tag{2.3.20}$$

Visbeidzot no nosacījuma $f(x_1) = y_1$ iegūstam:

$$c = \frac{y_1 + y_2}{2} \tag{2.3.21}$$

Izvēloties $N = 0$ parametra ϕ izteiksmē, iegūtās parametru vērtības ir šādas:

$$a = \frac{y_1 - y_2}{2} \tag{2.3.22a}$$
$$c = \frac{y_1 + y_2}{2} \tag{2.3.22b}$$
$$\omega = \frac{\pi}{x_2 - x_1} \tag{2.3.22c}$$
$$\phi = \frac{\pi x_1}{x_1 - x_2} \tag{2.3.22d}$$



2.6. Att.: Trigonometriskā splainu interpolācija ar nosacījumu $y' = 0$ dotajos punktos.

Izdarīsim koordinātu transformāciju kā 2.3.2. nodaļā: $\hat{x} = x - x_1$ un $\hat{y} = y - y_1$. Tad $(\hat{x}_1, \hat{y}_1) = (0, 0)$ un parametru vērtības vienkāršojas:

$$a = -\frac{\hat{y}_2}{2} \tag{2.3.23a}$$

$$c = \frac{\hat{y}_2}{2} \tag{2.3.23b}$$

$$\omega = \frac{\pi}{\hat{x}_2} \tag{2.3.23c}$$

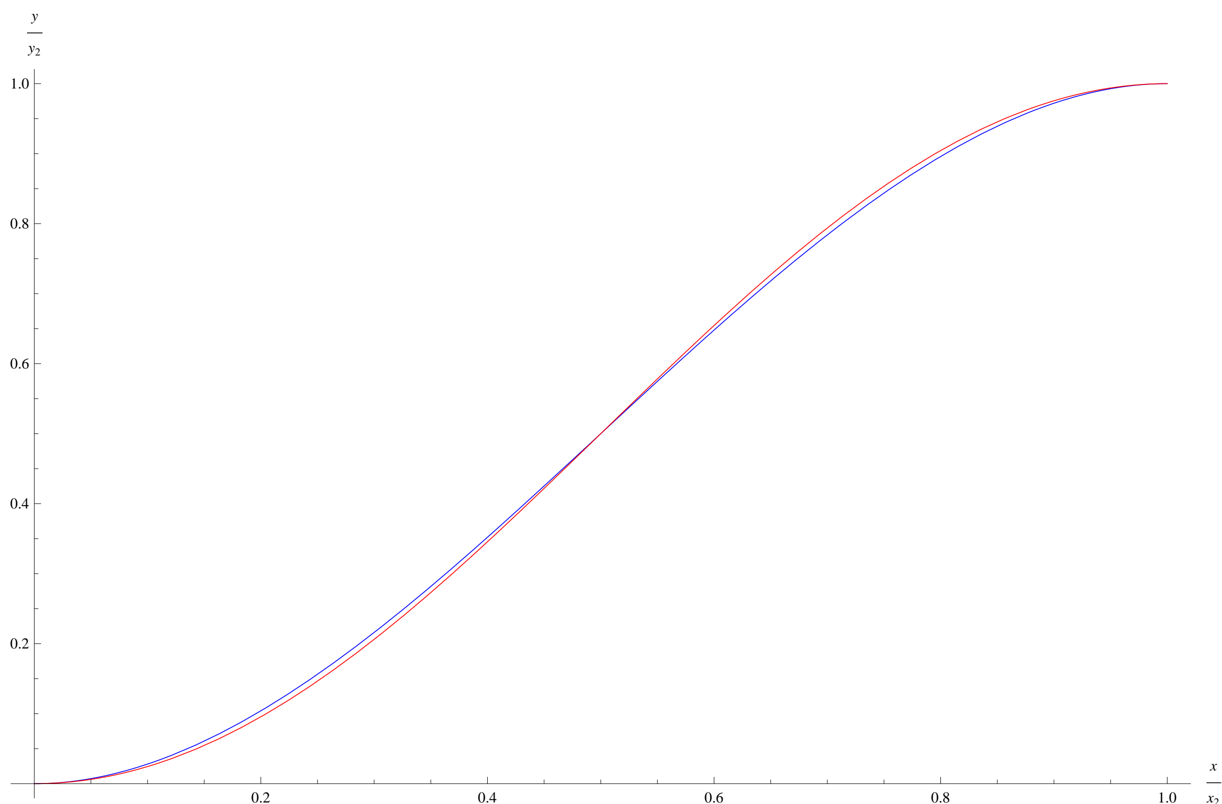
$$\phi = 0 \tag{2.3.23d}$$

Turpmāk lietosim jaunās koordinātes un jumtiņus tām vairs nerakstīsim. Interpolācija šajās koordinātēs:

$$y = \frac{y_2}{2}(1 - \cos\frac{\pi x}{x_2}) \tag{2.3.24}$$

Vizuāli šī interpolācija apskatāma 2.6. attēlā.

Uzmanīgs lasītājs, iespējams, pamanīs, ka attēls 2.6. līdzinās 2.5. attēlam. Patiešām tā arī ir—abu interpolāciju salīdzinājums redzams 2.7. attēlā.

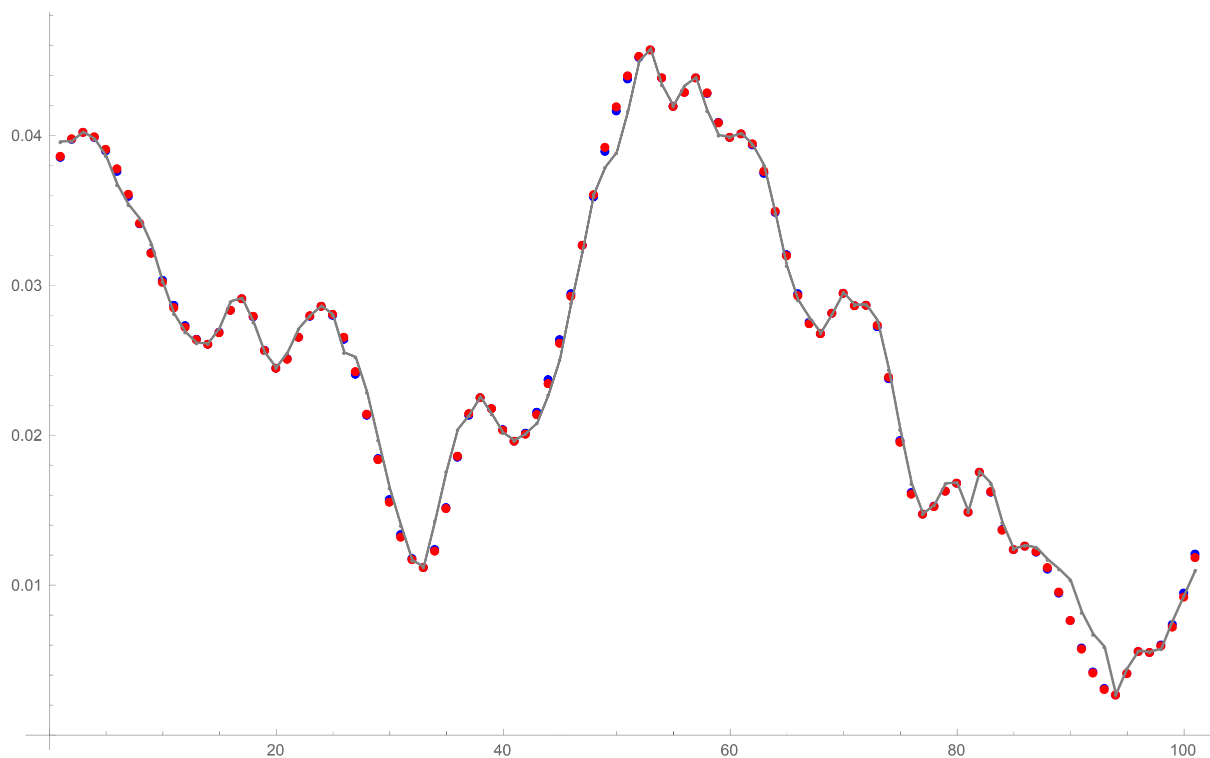


2.7. Att.: Trigonometriskās (sarkanā līkne) un polinomiālās (zilā līkne) splainu interpolācijas salīdzinājums.

Diskretizējot šo funkciju amplitūdām, kas dotas vienmērīgos laika intervālos, ar x_i un x_{i+n} apzīmēsim divus blakus esošus dotos punktus, bet n ir laika soļu skaits starp šīm amplitūdām. Amplitūdu x_k, kur $i < k < i + n$ aprēķinām ar šādu izteiksmi:

$$x_k = x_i + \frac{x_{i+n} - x_i}{2}(1 - \cos\frac{\pi k}{n}) \tag{2.3.25}$$

Reāla signāla rekonstrukcija ar trigonometrisku un polinomiālu splainu interpolācijām aplūkojama 2.8. attēlā.



2.8. Att.: Signāls (pelēks) un tā rekonstrukcija ar polinomiāliem (zilā krāsā) un trigonometriskiem (sarkanā krāsā) splainiem.

3. Praktiskā daļa

3.1. Algoritma pamatidejas realizācija

Pirmajiem eksperimentiem tika izveidotas divas programmas *Wolfram Mathematica* vidē—viena, kas *WAVE* formāta datni ar diviem kanāliem *saspiež*, bet otra to *atspiež*. Šī vide tika izvēlēta, jo tajā izpildāmais kods ir īss, viegli manipulējams un arī ar starprezultātiem iespējamas dažādas operācijas. Vairums rezultātu un attēlu, kas redzami šajā darbā, iegūti tieši apstrādājot šo programmu rezultātus un starprezultātus. Šo programmu kods ir pievienots A pielikumā.

Šīs realizācijas mērķis bija novērtēt manipulāciju ietekmi uz signālu un veikt pirmos izmēģinājumus ar piedāvāto metodi, tāpēc atsevišķas teorētiskajā daļā aprakstītās nianses, piemēram, Golomba-Raisa kods, tajā nav realizētas.

3.2. Pilna saspiešanas algoritma realizācija

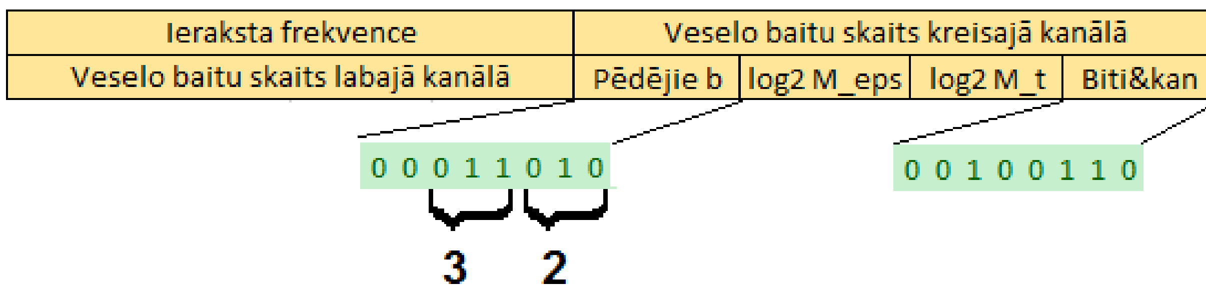
3.2.1. Saspiestās datnes formāts

Vispirms jāvienojas par datnes formātu, kādā tiks saglabāti dati jeb par protokolu.

Atgādināsim, ka signāls ar noteiktu frekvenci (amplitūdu biežumu) tiek aizstāts ar to pīķu virkni un attālumiem starp šiem pīķiem. Turklāt, šīs vērtības tiek saglabātas ar Golomba-Raisa kodu, tātad vērtības aizņem dažādu bitu skaitu.

Ja kāds signāls, piemēram, kreisais kanāls, mūsu pierakstā aizņem N bitus, tad par šī signāla veselo baitu skaitu saucam skatili $q = \lceil \frac{N}{8} \rceil$, bet $r = N - 8q$ saucam par bitu skaitu pēdējā baitā.

Datne sākas ar 16 baitu garu palīgdatu bloku (galveni), kurā ir šādi dati norādītajā secībā (skat. arī 3.1. att.):



3.1. Att.: Galvenes paraugs. Atšifrējums: kreisā kanāla pēdējā baitā—3 biti, labā—2. Kreisā kanāla pirmais pīķis negatīvs, tad signāls aug. Labā kanāla pirmais pīķis pozitīvs, pēc tam signāls dilst.

- Ieraksta frekvence (sampling rate)—4 baiti;
- Veselo baitu skaits kreisajā kanālā—4 baiti;
- Veselo baitu skaits labajā kanālā—4 baiti;
- Neizmantoti—2 biti;
- Bitu skaits pēdējā kreisā kanāla baitā—3 biti;
- Bitu skaits pēdējā labā kanāla baitā—3 biti;
- Golomba-Raisa koda parametra M_ε binārais logaritms—1 baits;
- Golomba-Raisa koda parametra M_t binārais logaritms—1 baits;
- Neizmantoti—2 biti;
- Pirmās amplitūdas zīme kreisajā kanālā—1 bits (1, ja negatīva);
- Sākotnējā signāla izmaiņa kreisajā kanālā—1 bits (1, ja vispirms signāla vērtības dilst);
- Pirmās amplitūdas zīme labajā kanālā—1 bits (1, ja negatīva);
- Sākotnējā signāla izmaiņa labajā kanālā—1 bits (1, ja vispirms signāla vērtības dilst);
- Amplitūdu izšķirtspēja—1 bits (0, ja oriģinālās amplitūdas bijušas 16 biti, 1, ja 8 biti);
- Ieraksta kanālu skaits—1 bits (0, ja 2 kanāli; 1, ja 1 kanāls).

Pēc galvenes datnē saglabāsim vispirms kreiso, pēc tam—labo kanālu (ja tāds ir). Kanāla saglabāšanā kā pirmo vērtību liekam kanāla pirmo amplitūdu, kam seko attālums līdz pirmajam pīķim. Pirmā pīķa vietā saglabājam pirmā pīķa un pirmās amplitūdas starpības absolūto vērtību. Tālāk pamīšus saglabājam attālumus līdz nākamajam pīķim un pīķu vērtību starpību absolūtās vērtības.

Ja kreisais kanāls neaizņem veselus baitus, tad pēdējā baitā neizmantotos bitus atstāj neizmantotus—labo kanālu sāk rakstīt jaunā baitā.

Datnei izvēlamies paplašinājumu *.tdc* un konkrētības labad sauksim šo algoritma realizāciju par *TDC* (no angļu *time domain compression*—laika telpas saspiešana).

3.2.2. Programmas realizācija

Tika izveidota *C++* programmatūra, lai novērtētu piedāvātā algoritma saspiešanas pakāpi, ātrdarbību un veiktu citas pārbaudes, kuras *Wolfram Mathematica* vidē ir sarežģītas vai ilgas.

Tika izstrādātas divas programmas, kas veic attiecīgi saspiešanas un atspiešanas procesu. Programmu kods dots B pielikumā. Tāpat programmu kods un arī izpildāmas programmu versijas (*.exe* datnes) pieejamas arī maģistra darba digitālajā pielikumā [36].

Izstradātās programmas darbam ar *WAVE* datnēm izmanto bibliotēku *Aquila* [37], izstrādes gaitā nācas arī modificēt *Aquila* datņu saglabāšanas funkcionalitāti tā, lai būtu iespējams saglabāt arī *WAVE* datnes ar diviem kanāliem.

Programmas izsaucamas no komandrindas, norādot apstrādājamo datni un izvaddatnes nosaukumu. Pēc datnes apstrādes programma beidz darbu.

Datnes *avots.wav* saspiešana datnē *saspiests.tdc* veicama ar šādu komandu:

```
tdc-comp avots.wav saspiests.tdc
```

Iespējams arī norādīt parametrus Golomba-Raisa kodam:

```
tdc-comp avots.wav saspiests.tdc -m 4096 4
```

Šajā gadījumā $M_\varepsilon = 4096$, $M_t = 4$. Ja vērtības netiek norādītas, tiek izmantotas noklusētās vērtības, kas (kopā ar īsu instrukciju angļu valodā) noskaidrojamas izsaucot programmu bez argumentiem:

```
tdc-comp
```

Datnes *saspiests.tdc* atspiešana izsaucama ar šādu komandu:

```
tdc-decomp saspiests.tdc atspiests.wav
```

Papildus iespējams norādīt rekonstrukcijas metodi, kas var būt lineāra, polinomiāla vai trigonometriska interpolācija, ko apzīmē ar atslēgvārdiem attiecīgi *linear*, *poly*, *trig*, piemēram:

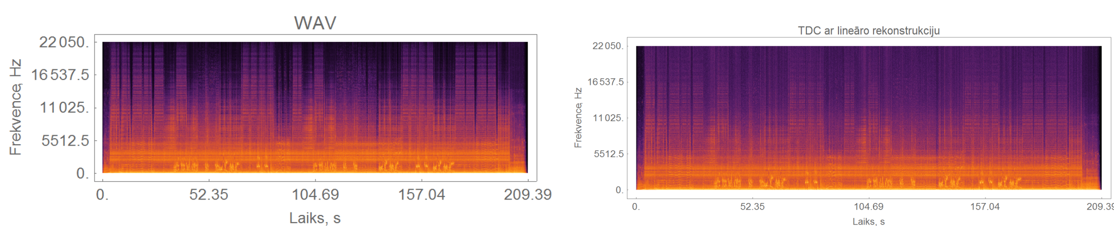
```
tdc-decomp saspiests.tdc atspiests.wav poly
```

4. Eksperimenti un rezultāti

4.1. Spektrogrammas

Lai pētītu algoritma ietekmi uz signāla spektrālo sadalījumu, ar izstrādātajām programmām *tdc-comp* un *tdc-decomp* tika saspiests un atspiests mūzikas ieraksts *Totenkopf - Burn the Desert* (oriģinālā—*WAVE* formāta datne ar 16 bitu amplitūdu izšķirtspēju un 44100 Hz temporālo izšķirtspēju).

Iegūtais rezultāts analizēts gan klausoties, gan pētot signālu spektrālos attēlus. Novērots, ka, rekonstruējot signālu ar lineāro interpolāciju, dzirdams augstas frekvences troksnis. Šāds pats defekts redzams arī aplūkojot spektrālos attēlus (skat. 4.1. att.).



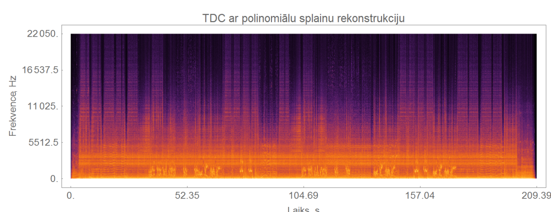
(a) Oriģinālā signāla spektrogramma

(b) Lineāri rekonstruētā signāla spektrogramma

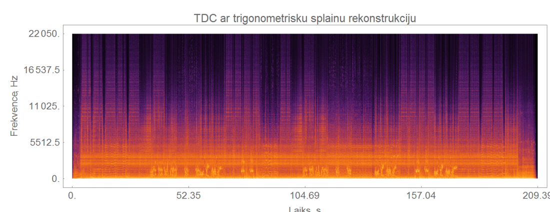
4.1. Att.: Oriģinālā signāla un ar lineāro interpolāciju rekonstruētā signāla spektrogrammu salīdzinājums *Totenkopf* dziesmai *Burn the Desert*.

Savukārt, aplūkojot ar polinomiālo interpolāciju (skat. 4.2.a att.) un ar trigonometrisko interpolāciju (skat. 4.2.b att.) rekonstruēto signālu spektrogrammas, ievērojamas atšķirības no oriģināla (skat. 4.1.a att.) nav novērojamas. Arī klausoties šos signālus saklausāmās atšķirības ir ļoti nelielas. Turklāt arī savā starpā ar abām interpolācijas metodēm iegūtie rezultāti ir ļoti līdzīgi.

Šīs sadaļas spektrālie attēli lielākā izmērā, kā arī citu saspiešanas algoritmu doto signālu un to spektrogrammu attēli aplūkojami C pielikumā. Citu algoritmu (*AAC*, *MP3* un *Ogg*



(a) Ar polinomiālo interpolāciju rekonstruētā signāla spektrogramma



(b) Ar trigonometrisko interpolāciju rekonstruētā signāla spektrogramma

4.2. Att.: Ar polinomiālo un trigonometrisko interpolāciju rekonstruētā signāla spektrogrammu salīdzinājums *Totenkopf* dziesmai *Burn the Desert*.

Vorbis) signālu un spektrālie attēli konstruēti, lai būtu redzams, kā šajā darbā piedāvātā algoritma ietekme uz signālu un tā spektrālo attēlu atšķiras no citu algoritmu ietekmes.

Pielikumā redzams, ka šajā darbā piedāvātā pieeja (*TDC*) atšķirība no citiem zudumu algoritmiem neatmet daļu spektra (pārējie atmet augstās frekvences). Tomēr tā vietā parādās trokšņi, kas spektrālajā attēlā izskatās kā *migla*. C pielikumā redzams arī kā signālu rekonstruē dažādas saspiešanas metodes. Novērojams, ka citas metodes signālu rekonstruē mazāk līdzīgu oriģinālam (īpaši *MP3* gadījumā)—ja oriģinālajā signālā ir pārliekuma vai ekstrēma punkts, tad arī rekonstruētajā tāds šajā laika momentā ir atrodams, taču absolūtās vērtības netiek rekonstruētas pārāk precīzi (dažādiem algoritmiem—dažāda precizitāte).

4.2. Veiktspējas eksperimenti

Algoritma veiktspējas pārbaudei tika veikti eksperimenti ar divpadsmit dažāda žanra mūzikas ierakstiem (skat. 4.1. tabulu). Tālāko rezultātu pārskatāmības labad šīs dziesmas apzīmētas ar saīsinātiem apzīmējumiem. Visu izmantoto ierakstu oriģinālie avoti ir *WAVE* vai *FLAC* datnes—skaņas ieraksti bez iepriekš radītiem zudumiem un saspiešanas defektiem.

4.2.1. Saspiešanas pakāpe

Golomba-Raisa kodēšanas parametru vērtības optimālai saspiešanas pakāpei tika meklētas iteratīvi, atkārtoti izpildot programmu *tdc-comp* pie dažādām parametru vērtībām (skat. 4.3. att.). Tika atrastas maksimālās saspiešanas pakāpes un noskaidrotas optimālās Golomba-Raisa kodēšanas parametru vērtības katrai no 12 dziesmām (skat. 4.2. tab.). Tika

4.1. Tabula: Veiktspējas eksperimentos izmantotās dziesmas

Dziesma	Apz.	Žanrs	Ilgums	Apjoms, kB
Cave In - Anchor	CIA	Alt. rock	3:14	33562
Jamiroquai - Seven Days in Sunny June	JSD	Funk	3:57	40977
Jasper Byrne - Miami	JBM	Ambient	4:00	41345
Kenny Loggins - Danger Zone	KDZ	Hard rock	3:36	37376
Lucy Rose - Watch Over	LRW	Folk rock	3:31	36445
Madeon - Finale	MDF	House	3:26	35598
Mark Ronson - Uptown Funk	MRU	Disco	4:29	46398
Metallica - Hit the Lights	MHL	Thrash metal	4:17	44346
OceanLab - Breaking Ties	OBT	Trance	5:14	54216
Solar Fields - Introduction	SFI	Ambient	5:34	57609
The American Dollar - Age of Wonder	ADA	Post-rock	4:52	50470
Thomas Datt - Phoenix Burn	TDP	Trance	5:45	59537

Mt \ Mx	1024	2048	4096	8192
1		8825	8634	
2	9352	8474	8290	8459
4		8521	8334	
8		8862		

4.3. Att.: Optimālo parametru meklēšana *Foo Fighters* dziesmas *Arlandria* saspiešanai—saspiestās datnes izmērs kilobaitos pie dažādām Golomba-Raisa kodu parametru vērtībām. Ar *Mx* un *Mt* saprotami attiecīgi M_ε un M_t.

novērots, ka visbiežāk optimālās parametru vērtības ir $M_\varepsilon = 2048$ un $M_t = 2$, tāpēc šādas vērtības tiek izmantotas kā noklusētās programmā *tdc-comp*.

4.2. tabulā ar S_0 apzīmēts *WAVE* datnes sākotnējais apjoms kilobaitos, ar S_{so} apzīmēts saspiestās datnes apjoms kilobaitos pie optimālajām Golomba-Raisa kodēšanas parametru vērtībām un $C_o = \frac{S_{so}}{S_0}$ ir saspiešanas pakāpe pie šīm parametru vērtībām. Ar M_ε un M_t apzīmētas attiecīgo parametru optimālās vērtības. S_{sn} ir saspiestās datnes apjoms kilobaitos pie noklusētajām Golomba-Raisa parametru vērtībām un $C_n = \frac{S_{sn}}{S_0}$ ir saspiešanas pakāpe pie šīm vērtībām.

Salīdzinot skaņas saspiešanas algoritmus mēdz aplūkot arī izmantoto bitu biežumu (angliski—*bitrate*)—daudzumu laika vienībā. Jāņem gan vērā, ka *MP3* algoritmam bitu

4.2. Tabula: Maksimālās saspiešanas pakāpes, optimālās parametru vērtības un saspiešanas pakāpe pie optimālajām parametru vērtībām.

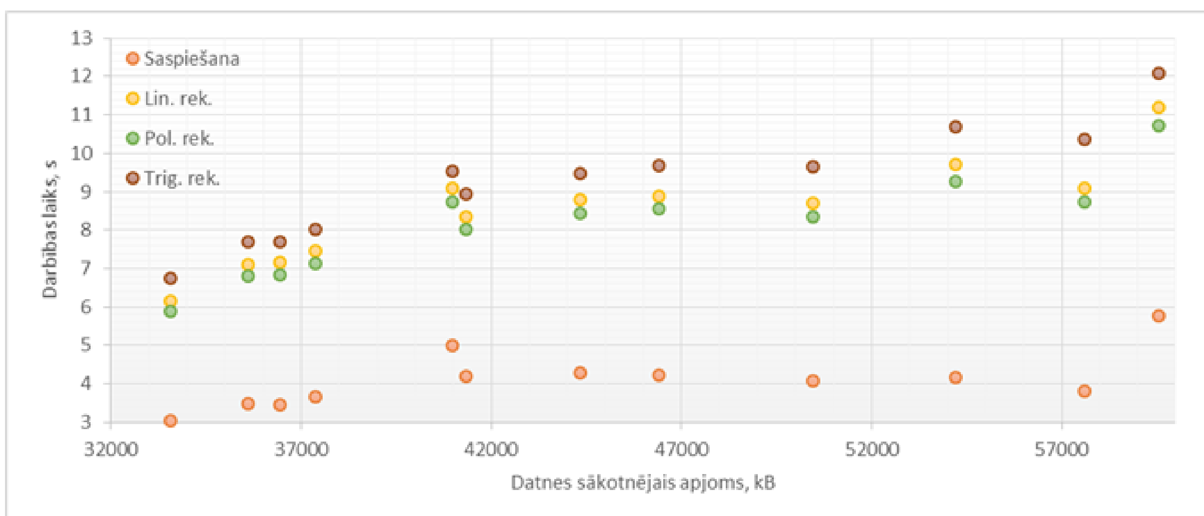
Dziesma	S_0	S_{so}	C_o	M_ε	M_t	S_{sn}	C_n
CIA	33562	9547	28%	4096	2	10064	30%
JSD	40977	16497	40%	2048	1	16580	40%
JBM	41345	13835	33%	2048	2	13835	33%
KDZ	37376	12014	32%	2048	2	12014	32%
LRW	36445	11277	31%	2048	1	11277	31%
MDF	35598	10952	31%	8192	2	11662	33%
MRU	46398	13705	30%	2048	2	13705	30%
MHL	44346	14021	32%	4096	2	14114	32%
OBT	54216	13281	24%	2048	2	13281	24%
SFI	57609	12043	21%	2048	2	12041	21%
ADA	50470	13111	26%	2048	2	13111	26%
TDP	59537	18984	32%	2048	2	18984	32%

4.3. Tabula: Saspiešanas pakāpes C un bitu biežumi b dažādiem algoritmiem.

Dziesma	C_{TDC}	$b_{TDC}, kb/s$	C_{FLAC}	$b_{FLAC}, kb/s$	C_{Vorbis}	$b_{Vorbis}, kb/s$
CIA	30%	394	74%	1028	7,7%	107
JSD	40%	557	73%	1004	8,1%	112
JBM	33%	461	66%	905	8,1%	111
KDZ	32%	445	69%	956	7,6%	105
LRW	31%	428	63%	866	7,8%	107
MDF	33%	425	75%	1037	8,2%	113
MRU	30%	408	67%	921	7,7%	106
MHL	32%	436	70%	964	7,8%	108
OBT	24%	338	61%	836	8,1%	112
SFI	21%	288	64%	886	8,5%	118
ADA	26%	359	65%	901	8,0%	110
TDP	32%	440	62%	850	8,0%	111

biežums ir parametrs. Arī *Vorbis* algoritmam, kuram bitu biežums nosacīti ir patvaļīgi mainīgs, ir parametrs *kvalitāte* un dažādām kvalitātes vērtībām atbilst noteikti nominālie bitu biežumi, tātad arī šajā gadījumā to var uzskatīt par izvēlamu parametru. *TDC* un *FLAC*, savukārt, bitu biežums nekādi nav maināms pēc lietotāja izvēles un ir atkarīgs tikai no apstrādājamā signāla īpašībām.

Ar dažādiem algoritmiem iegūtās saspiešanas pakāpes un bitu biežumi parādīti 4.3. tabulā. Ar *MP3* iegūtais bitu biežums bija 125,1-125,6 kb/s un kompresijas pakāpe 9,071%-9,074%, tātad šīs vērtības mainījās mazāk nekā par procentu un tāpēc netiek attēlotas tabulā. Novērojams, ka ar *TDC* saglabājamo datu apjoms ir 2-3 reizes mazāks nekā *FLAC* algoritmam, bet ar *Vorbis* iegūtas 2,5-5 reizes mazākas datnes nekā ar *TDC*.



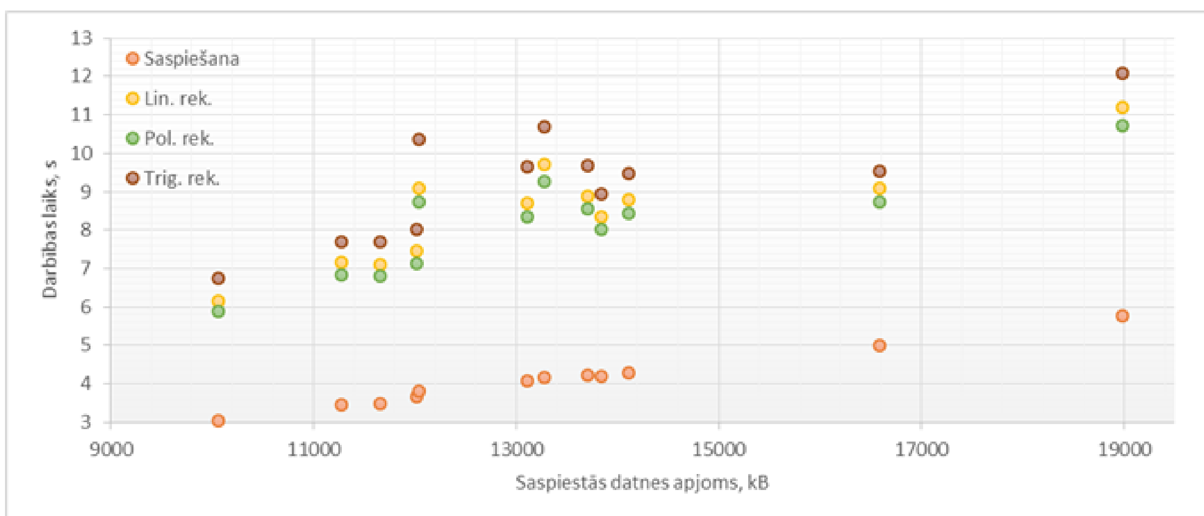
4.4. Att.: Algoritma realizācijas *TDC* darbības ilgums dažādām datnēm.

4.2.2. Ātrdarbība

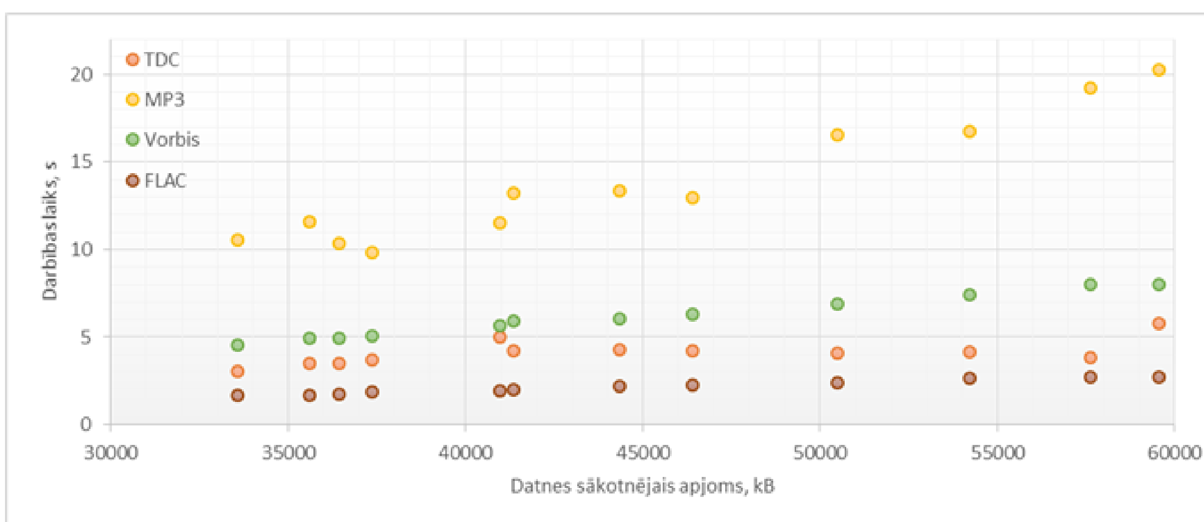
Algoritma realizācija *TDC* ātrdarbība tika mērīta, izmantojot *Windows* komandrindas iespējas fiksēt programmas darba sākuma un beigu laiku. Vispirms, atkārtojot mērījumus vienam ierakstam nemainīgos apstākļos, tika noskaidrots, ka mērījumu neprecizitāte nepārsniedz 1%. Pēc tam 12 pētāmie ieraksti tika saspiesti un atspiesti ar visām realizētajām rekonstrukcijas metodēm. Detalizētu ātrdarbības pārbaužu aprakstu un rezultātu tabulas var atrast D pielikumā.

Aplūkojot ātrdarbības testa rezultātus (skat. 4.4. att.) redzam, ka saspiešanas programma aplūkotajos gadījumos darbojas 3-6 sekundes, savukārt signāla rekonstruēšana notiek 6-12 sekundes. Novērojama korelācija, ka apjomīgākas datnes tiek apstrādātas ilgāk, taču redzams, ka apstrādi ietekmē arī citi faktori—atsevišķos gadījumos lielākas datnes ir apstrādātas ātrāk nekā mazākas. Interesanti, ka monotonāka atkarība apstrādes laikiem ir nevis no sākotnējās, bet saspiestās datnes apjoma (skat. 4.5. att.). Saspiešanas programmas darbības ilgums pat izrādās kļūdas robežās (1%) monotoni augošs atkarībā no saspiestās datnes apjoma. Tiesa, nelielais eksperimentālo datu apjoms neļauj saukt šo tendenci par striktu sakarību—iespējams, ka arī citas signāla īpašības var ietekmēt saspiešanas programmas darbības ilgumu.

Izmantojot programmu *SoX*[38], tika noskaidrots arī *MP3*, *Vorbis* un *FLAC* algoritmu darbības laiks uz pētāmajiem ierakstiem.



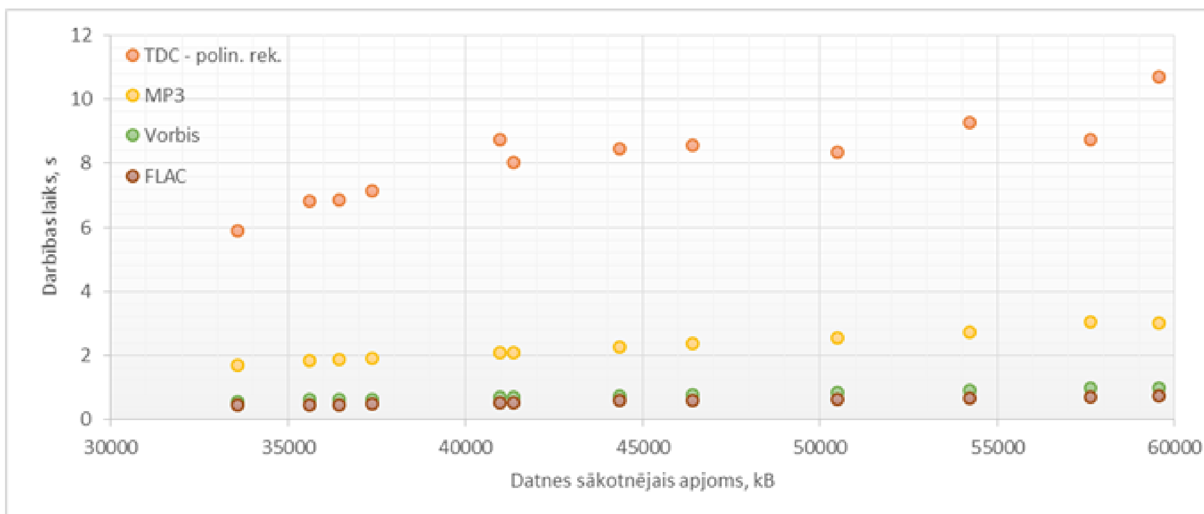
4.5. Att.: Algoritma realizācijas *TDC* darbības ilgums dažādām datnēm.



4.6. Att.: Saspiešanas ilgums dažādiem algoritmiem.

Novērots, ka saspiešana (skat. 4.6. att.) visātrāk veicama ar bezzudumu algoritmu *FLAC*, par (vidēji) nepilnām divām sekundēm lēnāka ir *TDC* darbība, no kā līdzīgā apmērā atpaliek *Vorbis*. Ievērojami lēnāka ir *MP3* darbība.

Novērtējot atspiešanas procesa ilgumu (skat. 4.7. att.) ar dažādiem algoritmiem, novērots, ka *TDC* darbojas ievērojami lēnāk par pārējiem—*MP3* darbības laiks ir 3-4 reizes īsāks. Savukārt *Vorbis* un *FLAC* signālu rekonstruē 3-4 reizes ātrāk nekā *MP3*.



4.7. Att.: Atspiešanas ilgums dažādiem algoritmiem.

4.3. Kvalitātes eksperimenti

4.3.1. Eksperimenta apraksts

Lai novērtētu rekonstruētās skaņas kvalitāti, tika veikts klausīšanās eksperiments. Ņemot vērā kvalitātes novērtēšanas paņēmienus, kas parasti tiek izmantoti (skat. 1.6. nodaļu), tika izveidota vienkāršota eksperimentālā procedūra, kas būtu realizējama ar brīvprātīgiem dalībniekiem bez īpašas sagatavotības, tai pat laikā iegūstot izmantojamus datus.

Eksperimentā tika izmantoti 15 skaņas ieraksti (6 dziesmu fragmenti, 3 balss ieraksti un 6 vienkāršas skaņas—pīkstieni, viena mūzikas instrumenta skaņa) ar ilgumu no 1 līdz 9 sekundēm. Pilnīgāku paraugu aprakstu (un citu papildinformāciju par šo eksperimentu) var aplūkot E pielikumā. Izmantotie paraugi ir atrodami maģistra darba digitālajā pielikumā [36].

Līdzīgi literatūrā aprakstītajām metodēm [29], eksperimenta dalībnieki (subjekti) noklausījās oriģinālo signālu (apzīmēts ar A) un rekonstruēto signālu (apzīmēts ar B). Tika dots arī trešais signāls X, kurš bija A vai B kopija. Subjekta uzdevums—noteikt, vai X ir A vai B kopija. Pēc tam subjektam jānovērtē šādas rekonstrukcijas pielietojamība šim paraugam ar atzīmi no 1 līdz 10, kuru nozīme izskaidrota E pielikumā.

Eksperiments tika veikts neklātienē, subjektiem bija iespēja klausīties paraugus tik reizes, cik vēlas. Eksperimenta paraugu apjoms un skaits tika izvēlēts tā, lai taupītu subjektu laiku. Saskaņā ar subjektu atsauksmēm, eksperimenta veikšana aizņēma 14-60 minūtes.

4.4. Tabula: Eksperimenta dalībnieku vērtējumi par paraugu kvalitāti.

Paraugs	Kļūdu procents	Kvalitātes vērtējums	Kvalitāte 1-6	Kvalitāte 7-9	Kvalitāte 10
song1	20%	7,0	6	6	3
song2	20%	9,1	1	7	7
song3	13%	8,0	3	8	4
song4	20%	8,1	3	7	5
song5	40%	8,8	0	8	7
song6	0%	4,5	15	0	0
sound1	27%	7,3	5	8	2
sound2	0%	4,9	13	2	0
sound3	0%	6,4	10	3	2
sound4	7%	5,1	12	2	1
sound5	0%	5,7	10	4	1
sound6	7%	4,5	14	1	0
voice1	7%	7,2	4	10	1
voice2	20%	7,9	4	7	4
voice3	47%	8,4	2	8	5

4.3.2. Eksperimenta rezultāti

Informāciju par eksperimenta dalībniekiem un eksperimentā iegūtos datus var atrast E pielikumā. Katram no paraugiem tika noskaidrots, cik procentos gadījumu subjekti nepareizi noteica, vai X ir A vai B kopija. Rupji novērtējot var uzskatīt—gadījumi, kad A un B nebija atšķirami ir divreiz vairāk par kļūdu gadījumiem, jo neatšķiramības gadījumā ir *50%* iespēja uzminēt pareizi.

Eksperimenta rezultātu kopsavilkums redzams E6. tabulā. Tajā attēlots, cik procentos gadījumu paraugs tika atpazīts nepareizi; kāds ir vidējais aritmētiskais no subjektu dotā novērtējuma (10 ballu skalā) par šī parauga kvalitāti; cik dalībnieku novērtējuši rekonstruētā parauga kvalitāti ar 10 ballēm; cik dalībnieku novērtējuši parauga kvalitāti ar 7-9 ballēm; cik dalībnieku novērtējuši parauga kvalitāti ar 1-6 ballēm. Dalībnieki tika instruēti, ka 10 balles nozīmē—rekonstrukciju no oriģināla atšķirt nevar, 7-9 balles atbilst nenozīmīgām atšķirībām, bet pielietojumam pietiekamai kvalitātei, zemāki vērtējumi—sliktākai kvalitātei, kuras lietošana šādiem paraugiem nav pieņemama.

Autors vēlas norādīt, ka ne visi subjekti pieturējās rekomendētajiem kritērijiem un atsevišķos gadījumos signāls tika atpazīts pēc "čerkstoņa, kad beidzas" un citiem kritērijiem, kas tika pamanīti ar grūtībām, taču ielikts vērtējums 6 vai 7. Citos gadījumos piezīmēs norādīts, ka nevarēja signālus atšķirt, taču ielikts vērtējums 8 vai 9. Tomēr nekādus vērtējumus autors neizmainīja, tie tika analizēti tādi, kādus subjekti iesniedza.

Redzams, ka paraugus *song6*, *sound2*, *sound3*, *sound4*, *sound5*, *sound6*, *voice1* pareizi identificējuši visi eksperimenta dalībnieki (vai visi izņemot vienu—šādus gadījumus uzskatīsim par kļūdu atbildot). Šajos gadījumos rekonstruētais signāls ir atšķirams no oriģinālā.

Turpretim septiņiem paraugiem novērots, ka 20% un vairāk dalībnieku savās atbildēs kļūdījās. Kā iepriekš minēts, tas varētu liecināt par to, ka aptuveni 40% un vairāk dalībnieku nevarēja atšķirt šo paraugu rekonstrukciju no oriģināla. Par neatšķiramību pareizo atbilžu gadījumā reizēm liecināja arī dalībnieku komentāri piezīmēm atvēlētajā vietā.

Ar īpaši lieliem kļūdu procentiem izceļas paraugi *song5* un *voice3*—tuvu 50%. Tātad var pieņemt, ka pārliecinošs vairākums dalībnieku ir minējuši, atbildot par šiem paraugiem. Vairums subjektu arī norādīja, ka šo paraugu rekonstrukcijas nebija atšķiramas no oriģināla. Arī paraugam *song2* aptuveni puse subjektu norādīja par neatšķiramību. Darba autoram neatškirami šķita paraugi *song2* un *voice3*, savukārt paraugs *sound1* ar vienu atskaņošanas ierīci (kvalitatīvām austiņām) likās neatšķirams, bet ar citu (portatīvā datora skaļruni) rekonstrukcijas atšķirības no oriģināla bija ļoti viegli pamanāmas.

Jāpiebilst, ka daži subjekti (konkrēti—četri) piezīmēs norādīja, ka itin visus paraugus varēja atšķirt, ko pierādīja ar pareizu atbilžu virkni. Šie dalībnieki arī viszemāk novērtēja skaņas kvalitāti, vidējos vērtējumus (vidējais aritmētiskais no visu paraugu vērtējumiem) liekot šādus: 5,07; 5,07; 5,67; 6,20. Vidējais vērtējums pa visiem dalībniekiem un paraugiem ir 6,87. Tas liecina, ka izmantojot gana kvalitatīvu aparatūru, atšķirības kļūst pamanāmas. Trīs no šiem dalībniekiem izmantoja kvalitatīvas austiņas un ceturtais savu atskaņošanas aparatūru aprakstīja šādi:

1. E-MU 0404 USB skaņas karte ar toroidālo barošanas bloku, kas nobaro ar 18v opampus, kuri ielikti labāki kā oriģinālie + audio grade kondensatori.
2. Sony TA-N80ES MKIII jaudas pastiprinātājs ar uzlabotām komponentēm (pārāk daudz, lai visas uzskaitītu).
3. Visaton B200 pilna spektra high-end skaļruņi ar open baffle kasti.
4. Pasīvais filtrs, kas ceļ atpakaļ zemo frekvenču fāzi.
5. High-end kabeļi (USB, RCA un skaļruņu).

Aplūkojot dziesmu fragmentu rezultātus, kā katastrofālu izņēmumu varam redzēt *song6*, kuru atpazinuši visi eksperimenta dalībnieki un kvalitāti novērtējuši vidēji ar 4,5 ballēm,

kas, saskaņā ar rekomendētajiem kritērijiem (skat. E pielik.), atbilst absolūti nepietiekamai kvalitātei, kas ir uz saprotamības robežas. Neviens šo kvalitāti neatzina par piemērotu mūzikai.

Turpretim pārējiem dziesmu fragmentos vidējie vērtējumi ir 7,0–9,1 balle, kas pēc rekomendētajiem kritērijiem atbilst kvalitātei, kura ir pieņemama šādu ierakstu saspiešanai. Īpaši pozitīvi novērtēti paraugi *song2* un *song5*, no kuriem katru 7 subjekti atzina par neatšķiramiem un visi (*song2* gadījumā—tomēr viens ne) atzina kvalitāti par atbilstošu pielietojumam.

Vienkāršo skaņu atpazīšanā subjektu panākumi bija lielāki. Daļai subjektu neizdevās pareizi atpazīt *sound1*, pēc kļūdu procenta (27%) var novērtēt, ka aptuveni pusei subjektu šis paraugs likās neatšķirams. Arī vidējais kvalitātes vērtējums tam ir pieņemamā robežās—7,3. Ar pārējo vienkāršās skaņas paraugu atšķiršanu gandrīz nevienam grūtības neradās un attiecīgi arī atzīmes vairumā gadījumu norāda, ka rekonstrukcijas kvalitāte ir bijusi neapmierinoša.

Balss ierakstu atpazīšanā 7 dalībnieki kļūdaini atšķīra *voice3* versijas. Interesanti, ka tikai 4 no šiem dalībniekiem piešķīra kvalitātes novērtējumu 10. Lai arī pārējos paraugos kļūdu procents bija mazāks, katru no paraugiem par gana kvalitatīvu atzina vairāk nekā divas trešdaļas eksperimenta dalībnieku.

Novērtējums un secinājumi

Vispirms aplūkosim metodes priekšrocības, kas izriet no tās pamatprincipiem. Lai realizētu saspiešanu ar šo metodi, ir nepieciešams vienlaicīgi strādāt tikai ar dažām pēdējām vērtībām (ar četrām—tekošo un iepriekšējo signāla amplitūdu, iepriekšējā ekstrēma amplitūdu un attālumu laika soļos kopš iepriekšējā ekstrēma). Tas ir ievērojami atšķirīgi no citiem zudumradošajiem algoritmiem, kas izmanto frekvenču telpu un vienlaikus strādā ar simtiem un tūkstošiem vērtību. Turklāt piedāvātajā metodē veicamie aprēķini ir vienkāršāki nekā vairumā algoritmu saspiešanai ar zudumiem. Šīs īpašības dod iespēju algoritmu izmantot tekoša signāla (*straumes*) ātrai un vienkāršai saspiešanai pirms nosūtīšanas. Tiesa, patlaban algoritms ir realizēts tikai darbam ar statiskām datnēm. Arī atspiešanas procesā nepieciešams rīkoties tikai ar atsevišķām vērtībām, tātad algoritmam nepieciešams ļoti neliels atmiņas apjoms.

Saspiešanas spēja

Novērtējot kvantitatīvos rezultātus, redzam, ka piedāvātais algoritms pēc saspiešanas pakāpes krietni pārspēj jebkuru eksistējošo bezzudumu saspiešanas paņēmienu, taču atpaliek no algoritmiem, kas saspiež ar zudumiem (izņemot μ-likuma un A-likuma algoritmus). Var apgalvot, ka pēc saspiešanas pakāpes šis algoritms ir visspēcīgākais no tiem, kas netransformē signālu frekvenču telpā.

Par pretēju piemēru gan varētu nosaukt tiešu amplitūdu biežuma (*sampling rate*) samazināšanu, kas ir veicama neierobežoti, taču proporcionāli samazina signāla spektrālo diapazonu—maksimālā frekvence, kāda var būt saglabāta signālā ir vienāda ar pusi no amplitūdu biežuma (to sauc par Nīkvista frekvenci [39]). Ja spektrālā diapazona samazināšana ir pieļaujama, tad to var veikt jau neapstrādātajam signālam un darbā piedāvāto metodi pielietot tad, kad parasta amplitūdu biežuma sadalīšana tālāk vairs nav pieļaujama. Tātad šīs metodes nevar uzskatīt par konkurējošām.

Golomba-Raisa parametru izvēle dažu divnieka pakāpju (1024, 2048, 4096, 8192) robežās būtiski neietekmē saspiešanas pakāpi (skat. 4.3. att.). No tā varam secināt, ka Golomba-Raisa kodēšana ir pietiekama—nebūs iegūstams ievērojams uzlabojums, realizējot Golomba kodēšanu, kur parametrs var būt jebkurš naturāls skaitlis. Jānorāda, ka tas tiešām ir ti-

kai dažu vērtību robežās, bet pie lielākām novirzēm (piemēram, optimālā $M_\varepsilon = 2048$ vietā izmantojot 64) kodēšanas rezultātā notiek nevis saspiešana, bet pat izplešana.

Ātrdarbība

Mērot izstrādātās algoritma realizācijas *TDC* ātrdarbību, noskaidrots, ka saspiešanas ātrumā *TDC* atpaliek no *FLAC*, taču strādā ātrāk nekā zudumradošie algoritmi *MP3* un *Vorbis*.

Novērtējot atspiešanas programmas darbību, tā izrādījās ilgāka nekā pārējiem pētītajiem algoritmiem. *TDC* darbības ilgums bija 6-12 sekundes, atspiežot 3-5 minūtes garas dziesmas. Tas nozīmē, ka atspiešanas ātrums būtu pietiekams reālā laika atskaņošanai, tomēr salīdzinājumā ar citiem algoritmiem tas ir uzskatāms par neapmierinošu.

Autors uzskata, ka algoritma realizācijas ātrdarbība varētu būt ievērojami labāka, ja realizācija būtu optimizēta līdzīgā līmenī kā pārējiem algoritmiem—*TDC* datņu nolasīšanas un ierakstīšanas darbs notiek aptuveni tik pat ilgi (autora novērojums), cik visa *FLAC* darbība. Tāpēc autors uzskata, ka saspiešanas algoritms būtu realizējams programmā, kas strādā *FLAC* ātrumos (mazāk nekā 10% robežās) un atspiešanas algoritma realizācijai būtu jāstrādā lēnāk par *FLAC*, taču ātrāk nekā *MP3* un *Vorbis* atspiešanas procedūrām. Šie pieņēmumi balstīti uz katrā no algoritmiem veicamo aprēķinu apjomu.

Skaņas kvalitāte

Noteikts, ka iegūstamā skaņas kvalitāte vairumam mūzikas ierakstu ir piemērota, atsevišķos gadījumos—tuva ideālai. Tomēr jāatzīmē, ka ir arī ieraksti, kuros ar izmantoto metodi rodas nepieņemami skaņas kvalitātes kropļojumi.

Saskaņā ar vairuma eksperimenta dalībnieku viedokli, arī balss ierakstu kvalitātei šāda saspiešanas kvalitāte ir pieņemama. Šeit gan jāpiebilst, ka tika izmantoti tikai trīs balss ieraksta paraugi, kas neļauj apgalvot, ka patiešām visiem balss ierakstiem kvalitāte būs pietiekama.

Vienkāršiem skaņas ierakstiem—monohromatiskām skaņām un atsevišķām viena instrumenta notīm—ieraksta rekonstrukcijas kvalitāte izrādījās nepietiekama. Izrādās, ka šādos skaņas ierakstos rekonstrukcijas radītie defekti ir ļoti viegli pamanāmi un traucējoši.

Tika novērots, ka eksperimenta subjektiem, kas izmantoja kvalitatīvāku aparatūru, reproducētās skaņas kvalitāte likās zemāka un bija vieglāk atšķirt oriģinālu no rekonstrukcijas. Tiesa, tas ir raksturīgs ikvienam skaņas saspiešanas algoritmam ar zudumiem.

Novērots arī, ka atsevišķu paraugu skaņas kvalitāte būtiski atšķīrās pie dažādām atskaņojošajām iekārtām neatkarīgi no kvalitātes, t.i. dažreiz arī uz kādām no nekvalitatīvākajām iekārtām atšķirības un defekti izcēlās vairāk nekā uz citām. Tomēr netika novērotas tiešas likumsakarības, kas izskaidrotu šo parādību.

Darbā paveiktais un galvenie secinājumi

Darbā izdevās izpildīt visus izvirzītos darba uzdevumus un darba mērķis uzskatāms par sasniegtu—ir izstrādāta metode, kas balstās uz izvirzīto ideju un tā ir pārbaudīta. Sākumā izvirzītā hipotēze par šādas metodes lietderību atzīstama par patiesu—ir pielietojumi, kuros šāda skaņas saspiešanas metode dod pielietojumam nepieciešamo kvalitāti.

Secināts arī, ka ne visos gadījumos metode ir pielietojama—bija paraugi, kuros kvalitāti par nepietiekamu atzina ikviens no eksperimenta dalībniekiem.

Noskaidrots, ka balss ierakstiem šāda metode ir pielietojama, arī vairumam (tiesa, ne katram) mūzikas ierakstu šāda kvalitāte izrādījās pietiekama saskaņā ar eksperimenta dalībnieku viedokli.

Lai arī algoritma realizācijas ātrdarbība ir pietiekama skaņas ierakstu atskaņošanai reālā laikā, autors cerēja izveidot realizāciju, kas gan saspiešanas, gan rekonstruēšanas procesā pārspētu visus zudumradošos skaņas saspiešanas algoritmus. Tas izdevās tikai saspiešanas ziņā.

Kā vēl vienu no uzdevumiem, ko autors cerēja veikt pilnīgāk, jāpiezīmē subjektīvo klausīšanās testu apjomu, kur paraugu un subjektu skaits nebija pietiekams, lai veiktu nopietnu statistisko analīzi.

Iespējamie tālāko pētījumu virzieni

Darba izstrādes gaitā iezīmējās šādi tālāko pētījumu un attīstības virzieni:

- Jāveic subjektīvās kvalitātes testi kontrolētos apstākļos, pārbaudot rekonstruētā signāla atšķiramību no oriģinālā, izmantojot dažādu atskaņošanas aparatūru. Nepieciešams arī par kārtu lielāks subjektu un paraugu skaits.

- Jāizstrādā programma vai programmu komponente (bibliotēka), kas ļautu atskaņot ar šo metodi saspiestās datnes iepriekš nerekonstruējot tās *WAVE* datnē, jāuzlabo arī ātrdarbība salīdzinājumā ar realizāciju *TDC*.

- Jāpēta iespējas uzlabot rekonstruētās skaņas kvalitāti, piemēram, izmantojot frekvenču filtrus.

- Jāpēta iespējas uzlabot saspiešanas pakāpi, izmantojot metodes no citiem algoritmiem, piemēram, kanālu korelācijas un citas likumsakarības, kā, piemēram, *FLAC* algoritmā.

A Algoritma realizācija ar *Wolfram Mathematica*

A1. Saspiešana

Šajā pielikuma sadaļā dots *Wolfram Mathematica* kods, kas veic *WAVE* formāta datnes *input.wav* saspiešanu *CSV* formāta datnē *compressed.comp*.

```
SetDirectory@NotebookDirectory[]; input = Import[input.wav];
rate = input[[1, 2]]; chan1 = input[[1, 1, 1]];
chan2 = input[[1, 1, 2]];
peaks1 = Pick[chan1[[2;;−2]], Differences[Sign[Differences[chan1]]], elem_/;elem===−2||elem===2];
peaks2 = Pick[chan2[[2;;−2]], Differences[Sign[Differences[chan2]]], elem_/;elem===−2||elem===2];
positions1 = Flatten[1+Position[Differences[Sign[Differences[chan1]]], elem_/;elem===−2||elem===2]];
positions2 = Flatten[1+Position[Differences[Sign[Differences[chan2]]], elem_/;elem===−2||elem===2]];
gaps1 = Flatten[Append[{positions1[[1]] − 1}, Differences[positions1]]];
gaps2 = Flatten[Append[{positions2[[1]] − 1}, Differences[positions2]]];
compressed = {rate, peaks1, peaks2, gaps1, gaps2};
Export[compressed.comp, compressed, CSV];
```

A2. Atspiešana

Šajā pielikuma sadaļā dots *Wolfram Mathematica* kods, kas veic *CSV* formāta datnes *compressed.csv* atspiešanu (rekonstruēšanu) *WAVE* formāta datnē *decompressed.wav*. Signāla amplitūdas starp saglabātajām vērtībām tiek rekonstruētas, izmantojot lineāru interpolāciju.

```
SetDirectory@NotebookDirectory[];
input = Import[compressed.comp, CSV];
rate = input[[1, 1]];
peaks1 = input[[2]];
peaks2 = input[[3]];
```

```
gaps1 = input[[4]];
gaps2 = input[[5]];
reChan1 = Table[0, {Total[gaps1] + 1}];
reChan2 = Table[0, {Total[gaps2] + 1}];
n = 1;
prevVal = reChan1[[1]];
For[i = 1, i ≤ Length[peaks1], i++,
val = peaks1[[i]];
diff = val − prevVal;
gap = gaps1[[i]];
For[j = 1, j ≤ gap, j++,
reChan1[[n + 1]] = reChan1[[n]] + diff/gap;
n++; ];
prevVal = val; ]
n = 1;
prevVal = reChan2[[1]];
For[i = 1, i ≤ Length[peaks2], i++,
val = peaks2[[i]];
diff = val − prevVal;
gap = gaps2[[i]];
For[j = 1, j ≤ gap, j++,
reChan2[[n + 1]] = reChan2[[n]] + diff/gap;
n++; ];
prevVal = val; ]
decompressed = Sound[SampledSoundList[{reChan1, reChan2}, rate]];
Export[decompressed.wav, decompressed]
```

Lai veiktu atspiešanu ar citu interpolācijas metodi, rindiņas:

```
reChan1[[n + 1]] = reChan1[[n]] + diff/gap;
reChan2[[n + 1]] = reChan2[[n]] + diff/gap;
```

jāaizstāj ar citu interpolācijas metodi.

Piemēri polinomiālai un trigonometriskai interpolācijai (nosauktajā secībā:

$\text{reChan1}[[n+1]] = \text{diff}\left(3\text{gap}j^2 - 2j^3\right)/\text{gap}^3 + \text{prevVal};$

$\text{reChan1}[[n+1]] = \frac{1}{2}\text{diff}\left(1 - \cos\left(\frac{\pi j}{\text{gap}}\right)\right) + \text{prevVal};$

B Algoritma realizācija ar *C++*

Programmas dotas saīsinātā versijā—bez kļūdu pārbaudēm un komentāriem.

B1. Saspiešanas programma *tdc-comp*

main.cpp

```
#include <iostream>
#include <fstream>
#include <vector>

#include "aquila\source\WaveFile.h"

#include "io.h"
#include "encoding.h"

using namespace std;

int main(int argc, char *argv[])
{
  int mPeak, mGap;

  readArg(argc, argv, mPeak, mGap);

  char *inFile = argv[1];
  char *outFile = argv[2];

  Aquila::WaveFile wavLEFT(inFile);  //Ielasa failu (kreiso kanalu) mainigaja wavLEFT;
  Aquila::WaveFile wavRIGHT(inFile, Aquila::RIGHT);  //Ielasa failu (labo kanalu) mainigaja wavRIGHT;

  vector <int> gapsLEFT;  //Spraugu un piku masivi abiem kanaliem
  vector <int> peaksLEFT;
  vector <int> gapsRIGHT;
  vector <int> peaksRIGHT;
  int firstSignRIGHT, firstChangeRIGHT, firstSignLEFT, firstChangeLEFT;


  compress(wavLEFT, peaksLEFT, gapsLEFT, firstSignLEFT, firstChangeLEFT);  //wavLEFT pikus ieraksta masiva peaksLEFT un spraugas - gapsLEFT

  if(wavLEFT.getChannelsNum()>1)
    compress(wavRIGHT, peaksRIGHT, gapsRIGHT, firstSignRIGHT, firstChangeRIGHT);

  vector <bool> bitsLEFT; //masivi kodetajam vertibam
  vector <bool> bitsRIGHT;

  encode(peaksLEFT, gapsLEFT, bitsLEFT, mPeak, mGap); //Masivu peaksLEFT un gapsLEFT elementus saliek pamishus un kode ar Golomba-Raisa kodu

  if(wavLEFT.getChannelsNum()>1)
    encode(peaksRIGHT, gapsRIGHT, bitsRIGHT, mPeak, mGap);

  ofstream out;  //izvadfails
  out.open(outFile, ios::binary);

  uint32_t sampleRate = wavLEFT.getSampleFrequency();
  writeHeader(out, sampleRate, bitsLEFT.size(), bitsRIGHT.size(), mPeak, mGap, wavLEFT.getBitsPerSample(),
                       wavLEFT.getChannelsNum(), firstSignRIGHT, firstChangeRIGHT, firstSignLEFT, firstChangeLEFT);

  writeChan(out, bitsLEFT);

  if(wavLEFT.getChannelsNum()>1)
    writeChan(out, bitsRIGHT);

  out.close();
  return 0;
}
```

io.h

```
#ifndef IO_H_INCLUDED
#define IO_H_INCLUDED

void help(char *argv[]);
int checkFile(char *file, char *argv[]);
int readArg(int argc, char *argv[], int& mPeak, int& mGap);
int writeHeader(std::ofstream& out, uint32_t sampleRate, std::vector<bool>::size_type bitCountLEFT,
        std::vector<bool>::size_type bitCountRIGHT, int mPeak, int mGap, unsigned short bitRate, unsigned short channels,
              int& firstSignRIGHT, int& firstChangeRIGHT, int& firstSignLEFT, int& firstChangeLEFT);
int writeChan(std::ofstream& out, std::vector<bool>& bits);

#endif
```

encoding.h

```
#ifndef ENCODING_H_INCLUDED
#define ENCODING_H_INCLUDED

int compress(Aquila::WaveFile wav, std::vector <int>& peaks, std::vector <int>& gaps, int& firstSign, int& firstChange);
int encode(const std::vector <int>& peaks, const std::vector <int>& gaps, std::vector <bool>& bits, int mPeak, int mGap);
void golomb(std::vector <bool>& bits, int num, int m, int b);

#endif
```

io.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
#include <cstring>
#include <vector>

#define MPEAK_DEFAULT 4096   //jabut 2 pak - Golomba-Raisa koda parametra nokluseta vertiba piku kodesanai
#define MGAP_DEFAULT 2   //jabut 2 pak - Golomba-Raisa koda parametra nokluseta vertiba spraugu kodesanai

using namespace std;

void help(char *argv[])
{
  cout << "Usage: " << argv[0] << " <Input File> <Output File> [-m MPeak MGap]" << endl;
}

int readArg(int argc, char *argv[], int& mPeak, int& mGap)
{
  if(argc == 6)  // Ja ir 6 argumenti, ielasam Golomba-Raisa kodesanas parametrus
  {
    istringstream m1(argv[4]);
    if (!(m1 >> mPeak))
    {
      help(argv);
      return 1;
    }

    if(log2(mPeak) != floor(log2(mPeak)))
    {
      help(argv);
      return 1;
    }

    istringstream m2(argv[5]);
    if (!(m2 >> mGap))
    {
      help(argv);
      return 1;
    }

    if(log2(mGap) != floor(log2(mGap)))
    {
      help(argv);
      return 1;
    }

    return 0;
  }

  if(argc == 3)  // Ja ir 3 argumenti, izmantojam noklusetas vertibas.
  {
    mPeak = MPEAK_DEFAULT; //jabut 2 pak - Golomba-Raisa koda parametra vertiba piku kodesanai
    mGap = MGAP_DEFAULT; //jabut 2 pak - Golomba-Raisa koda parametra vertiba spraugu kodesanai

    return 0;
  }

  help(argv);
  return 1;
}

int writeHeader(ofstream& out, uint32_t sampleRate, vector<bool>::size_type bitCountLEFT,
 vector<bool>::size_type bitCountRIGHT, int mPeak, int mGap, unsigned short bitRate, unsigned short
  channels, int& firstSignRIGHT, int& firstChangeRIGHT, int& firstSignLEFT, int& firstChangeLEFT)
{
  uint32_t bytesLEFT = bitCountLEFT / 8; //Veselo baitu skaits katra kanala
  uint32_t bytesRIGHT = bitCountRIGHT / 8;

  int lastLEFT = bitCountLEFT % 8;  //Papildus bitu skaits pedeja baita
  int lastRIGHT = bitCountRIGHT % 8;
```

```
  uint8_t lastBits = lastLEFT*8 + lastRIGHT;  //Ierakstam abus skaitlus (kam max=7) viena baita

  uint8_t logMPeak = ceil(log(mPeak) / log(2));
  uint8_t logMGap = ceil(log(mGap) / log(2));

  uint8_t bitAndChan = 0;  //Pedejais bits 0—>2 kanali, 1—>1 kanals; priekspedejais bits 1—>8bit, 0—>16bit
  if(8 == bitRate)
    bitAndChan = 2;
  if(1 == channels)
    bitAndChan += 1;

  if(—1 == firstSignLEFT) //3. bits uzstadits, ja pirmais pikis kreisaja kanala negativs
    bitAndChan += 32;
  if(—1 == firstChangeLEFT)  //4. bits uzstadits, ja pec pirma pika kreisaja kanala dilst
    bitAndChan += 16;
  if(—1 == firstSignRIGHT) //5. bits uzstadits, ja pirmais pikis labaja kanala negativs
    bitAndChan += 8;
  if(—1 == firstChangeRIGHT)  //6. bits uzstadits, ja pec pirma pika labaja kanala dilst
    bitAndChan += 4;

  out.write((char*)&sampleRate,4);
  out.write((char*)&bytesLEFT,4);
  out.write((char*)&bytesRIGHT,4);
  out.write((char*)&lastBits,1);
  out.write((char*)&logMPeak,1);
  out.write((char*)&logMGap,1);
  out.write((char*)&bitAndChan,1);

  return 0;
}

int writeChan(ofstream& out, vector<bool>& bits)
{
  for (auto it = bits.begin(); it != bits.end();)
  {
      uint8_t b = 0;
      for (int i = 0; i < 8; ++i)
      {
          if(it != bits.end())
          {
            b |= (*it & 1) << (7 — i);  //ierakstam pa 1 bitam ar OR jeb |
            ++it;
          }
          else
          {
            b |= (0 & 1) << (7 — i);
          }

      }
      out.write((char*)&b, 1);
  }

  if(!out)
  {
    cerr << "Writing data to file failed." << endl;
    return 1;
  }

  return 0;
}
```

encoding.cpp

```
#include <vector>
#include <cmath>

#include "aquila\source\WaveFile.h"

#include "encoding.h"

using namespace std;

int compress(Aquila::WaveFile wav, vector <int>& peaks, vector <int>& gaps, int& firstSign, int& firstChange)
{
  auto it = wav.begin();  //Iterators pa signalu
  int gap = 0;  //Skaititajs spraugam starp pikiem
  int prevVal = *it;  //Pedeja apskatita vertiba — vajadziga, lai redzams, vai signals aug vai dilst
  int lastPeak = *it;  //Pedejais pikis — vajadzigs, lai redzamas starpibas starp pikiem

  if(*it < 0)
    firstSign = —1;
  else
    firstSign = 1;

  peaks.push_back(abs(*it));  //Ieraksta pirmo vertibu pikju masiva
  it++;

  while((peaks[0] == *it) && (it != wav.end()))  //Mekle, kur beidzas stacionara signala vertiba
  {
    gap++;
    it++;
  }

  firstChange = —1;

  if((*it > prevVal) && (it != wav.end()))  //Ja signala vertibas sakuma ir augosas, apstradajam atseviski vispirms
  {
    firstChange = 1;
```

```
    prevVal = *it;
    it++;
    while((*it >= prevVal) && (it != wav.end()))  //Kamer nedilst (turpina augt)
    {
      prevVal = *it;
      gap++;
      it++;
    }
    peaks.push_back(prevVal - lastPeak); // Ieraksta starpibu starp piki un sakuma vertibu
    lastPeak = prevVal;
    gaps.push_back(gap);  //Ieraksta spraugu starp sakuma vertibu un pirmo piki
    gap = 0;
  }

  while(it != wav.end())  //Apstradajam visu atlikuso signalu.
  {                       //Vispirms dils, jo, ja vispirms aug, tad tas jau ir apstradats
    while((*it <= prevVal) && (it != wav.end()))  //Kamer neaug (turpina dilt)
    {
      prevVal = *it;
      gap++;
      it++;
    }
    peaks.push_back(lastPeak - prevVal);  //Starpiba starp pedejo un iepr piki, sis bija zemaks (jo signals dila)
    lastPeak = prevVal;
    gaps.push_back(gap-1); //Atnem viens, jo sprauga ir par 1 mazaka neka solu skaits no viena pika uz nakamo
    gap = 0;

    if(it == wav.end())  //Ja pienakusas beigas, jaiziet no cikla, lai otra puse neieraksta pedejo vertibu velreiz
      break;

    while((*it >= prevVal) && (it != wav.end()))  //Kamer nedilst (turpina augt)
    {
      prevVal = *it;
      gap++;
      it++;
    }
    peaks.push_back(prevVal - lastPeak);  //Starpiba starp pedejo un iepr piki, sis bija augstaks (jo signals auga)
    lastPeak = prevVal;
    gaps.push_back(gap-1); //Atnem viens, jo sprauga ir par 1 mazaka neka solu skaits no viena pika uz nakamo
    gap = 0;
  }

  return 0;
}

int encode(const vector <int>& peaks, const vector <int>& gaps, vector <bool>& bits, int mPeak, int mGap)
{
  int bPeak = log2(mPeak);  //Parametri Golomba-Raisa kodam
  int bGap = log2(mGap);  //Tiek aprekinati vienreiz un atklata veida doti Golomba-Raisa funkcijai

  auto pit = peaks.begin();
  auto git = gaps.begin();

  golomb(bits, *pit, mPeak, bPeak);  //Kode pirmo piki un ieraksta masiva bits

  pit++;

  while(pit != peaks.end())
  {
    golomb(bits, *git, mGap, bGap);
    golomb(bits, *pit, mPeak, bPeak);
    git++;
    pit++;
  }

  return 0;
}

void golomb(vector <bool>& bits, int num, int m, int b)
{
  int q,r;

  q = num/m;  //Kvocientu pec dalisanas ar m saglaba unari
  for(int i=0; i<q; i++)
    bits.push_back(1);
  bits.push_back(0);

  r = num - (q*m);  //Atlikumu saglaba binari
  for(int i=0; i<b; i++)
    bits.push_back(1 & (r >> (b-1-i)));
}
```

B2. Atspiešanas programma *tdc-decomp*

main.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <numeric>

#include "aquila\source\WaveFile.h"

#include "io.h"
#include "decoding.h"
```

```
using namespace std;

int main(int argc, char *argv[])
{
  char *inFile = NULL;
  char *outFile = NULL;
  string method;
  readArg(argc, argv, inFile, outFile, method);

  uint32_t sampleRate;
  uint8_t bitRate, channels;
  uint32_t bitCountLEFT, bitCountRIGHT; //bitu skaits katra kanala
  int mPeak, mGap; //parametri Golomba—Raisa kodam
  int firstSignRIGHT, firstChangeRIGHT, firstSignLEFT, firstChangeLEFT;

  ifstream in;
  in.open(inFile, ios::binary);

  readHeader(in, sampleRate, bitCountLEFT, bitCountRIGHT, mPeak, mGap, bitRate, channels,
        firstSignRIGHT, firstChangeRIGHT, firstSignLEFT, firstChangeLEFT);

  vector <bool> bitsLEFT; //masivi kodetajam vertibam
  vector <bool> bitsRIGHT;

  readChan(in, bitsLEFT, bitCountLEFT);

  if(channels==2)
    readChan(in, bitsRIGHT, bitCountRIGHT);

  in.close();

  vector <int> gapsLEFT;  //Spraugu un piku masivi abiem kanaliem
  vector <int> peaksLEFT;
  vector <int> gapsRIGHT;
  vector <int> peaksRIGHT;

  decode(peaksLEFT, gapsLEFT, bitsLEFT, mPeak, mGap);

  if(channels==2)
    decode(peaksRIGHT, gapsRIGHT, bitsRIGHT, mPeak, mGap);

  vector <int> chanLEFT;
  vector <int> chanRIGHT;

  decompress(peaksLEFT, gapsLEFT, chanLEFT, method, firstSignLEFT, firstChangeLEFT);

  if(channels==2)
    decompress(peaksRIGHT, gapsRIGHT, chanRIGHT, method, firstSignRIGHT, firstChangeRIGHT);

  ofstream out;  //izvadfails
  out.open(outFile, ios::binary);

  writeHeader(out, channels, sampleRate, bitRate, chanLEFT.size());

  writeData(out, channels, bitRate, chanLEFT, chanRIGHT);

  out.close();
  return 0;
}
```

io.h

```
#ifndef IO_H_INCLUDED
#define IO_H_INCLUDED

void help(char *argv[]);
int checkFile(char *file, char *argv[]);
int readArg(int argc, char *argv[], char *& inFile, char *& outFile, std::string& method);
int readHeader(std::ifstream& in, uint32_t& sampleRate, uint32_t& bitCountLEFT, uint32_t& bitCountRIGHT, int& MPeak, int& MGap,
        uint8_t& bitRate, uint8_t& channels, int& firstSignRIGHT, int& firstChangeRIGHT, int& firstSignLEFT, int& firstChangeLEFT);
int readChan(std::ifstream& in, std::vector<bool>& bits, uint32_t& bitCount);
int writeHeader(std::ofstream& out, uint8_t& channels, uint32_t& sampleRate, uint8_t& bitRate, std::vector<int>::size_type chanSize);
int writeData(std::ofstream& out, uint8_t& channels, uint8_t& bitRate, std::vector<int>& dataLEFT, std::vector<int>& dataRIGHT);
int write8bitMono(std::ofstream& out, std::vector<int>& data);
int write16bitMono(std::ofstream& out, std::vector<int>& data);
int write8bitStereo(std::ofstream& out, std::vector<int>& dataLEFT, std::vector<int>& dataRIGHT);
int write16bitStereo(std::ofstream& out, std::vector<int>& dataLEFT, std::vector<int>& dataRIGHT);

#endif
```

decoding.h

```
#ifndef DECODING_H_INCLUDED
#define DECODING_H_INCLUDED

int decode(std::vector<int>& peaks, std::vector<int>& gaps, std::vector<bool>& bits, int mPeak, int mGap);
int invGolomb(std::vector<bool>& bits, std::vector<bool>::iterator& bit, int m, int b);
int decompress(std::vector<int>& peaks, std::vector<int>& gaps, std::vector<int>& chan, std::string& method,
                  int& firstSign, int& firstChange);
int linearGapInterpolation(int i, int gap, int prevVal, int nextVal);
int polyGapInterpolation(int i, int gap, int prevVal, int nextVal);
int trigGapInterpolation(int i, int gap, int prevVal, int nextVal);
double pi();

#endif
```

io.cpp

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <string>
#include <cstring>

#include "aquila\source\WaveFile.h"

#include "io.h"

using namespace std;

extern const string def_interpolation = "linear";  //nokluseta interpolacijas metode

void help(char *argv[])
{
  cout << "Usage: " << argv[0] << " <Input File> <Output File> [Interpolation method]" << endl;
}

int readArg(int argc, char *argv[], char *& inFile, char *& outFile, string& method)
{
  if(argc < 3)
  {
    help(argv);
    return 1;
  }

  inFile = argv[1];
  outFile = argv[2];

  method = def_interpolation;

  if(argc == 4)
    method = argv[3];
  else
    method = def_interpolation;

  if(argc > 4)
  {
    help(argv);
    return 1;
  }

  return 0;
}

int readHeader(std::ifstream& in, uint32_t& sampleRate, uint32_t& bitCountLEFT, uint32_t& bitCountRIGHT, int& MPeak, int& MGap,
        uint8_t& bitRate, uint8_t& channels, int& firstSignRIGHT, int& firstChangeRIGHT, int& firstSignLEFT, int& firstChangeLEFT)
{
  in.read((char*)&sampleRate, 4);

  uint32_t bytesLEFT, bytesRIGHT;
    in.read((char*)&bytesLEFT, 4);
    in.read((char*)&bytesRIGHT, 4);
  uint8_t lastBits;
    in.read((char*)&lastBits, 1);
    uint8_t lastLEFT = lastBits / 8;  //3-6 bits ir interesejosais skaitlis
    uint8_t lastRIGHT = lastBits % 8;  //pedejie 3 biti
  bitCountLEFT = bytesLEFT * 8 + lastLEFT;
  bitCountRIGHT = bytesRIGHT * 8 + lastRIGHT;

  uint8_t logMPeak;
  in.read((char*)&logMPeak, 1);
  MPeak = pow(2, logMPeak);

  uint8_t logMGap;
  in.read((char*)&logMGap, 1);
  MGap = pow(2, logMGap);

  uint8_t bitAndChan;
  in.read((char*)&bitAndChan, 1);
  if(bitAndChan%2) //ja pedejais bits ir 1
    channels = 1;
  else
    channels = 2;

  if((bitAndChan/2)%2) //ja priekspedejais bits ir 1
    bitRate = 8;
  else
    bitRate = 16;

  if(bitAndChan&32)  //ja 3. bits uzstadits
    firstSignLEFT = -1;
  else
    firstSignLEFT = 1;

  if(bitAndChan&16)  //ja 4. bits uzstadits
    firstChangeLEFT = -1;
  else
    firstChangeLEFT = 1;

  if(bitAndChan&8)  //ja 5. bits uzstadits
    firstSignRIGHT = -1;
  else
    firstSignRIGHT = 1;

  if(bitAndChan&4)  //ja 6. bits uzstadits
```

```
    firstChangeRIGHT = -1;
  else
    firstChangeRIGHT = 1;

  return 0;
}

int readChan(std::ifstream& in, vector<bool>& bits, uint32_t& bitCount)
{
  bool cont = true;  //vai turpinat nolasisanu
  while(cont )//&& !in.eof())
  {
    uint8_t b;
    in.read((char*)&b, 1);
    for(int i = 0; (i<8) && (bits.size()<bitCount); i++)
    {
      bool bit = b & (1 << (7-i)); //liekam masku, lai dabutu pa 1 bitam
      bits.push_back(bit);
    }
  }

  if(bits.size() < bitCount)   //Ja fails bijis par isu
  {
    cout << "Bits read: " << bits.size() << endl;
    cout << "Bits expected: " << bitCount << endl;
    cerr << "EOF reached, not all data read. File corrupted?" << endl;
    return 1;
  }

  return 0;
}

int writeHeader(ofstream& out, uint8_t& channels, uint32_t& sampleRate, uint8_t& bitRate, vector<int>::size_type chanSize)
{
  //Implementation based on WaveFileHandler::createHeader function from library Aquila
  Aquila::WaveHeader header;

  uint32_t bytesPerSec = channels*bitRate*sampleRate/8;
  uint32_t waveSize = channels*bitRate*chanSize/8;

  strncpy(header.RIFF, "RIFF", 4);
  header.DataLength = waveSize + sizeof(Aquila::WaveHeader) - 8;
  strncpy(header.WAVE, "WAVE", 4);
  strncpy(header.fmt_, "fmt ", 4);
  header.SubBlockLength = 16;
  header.formatTag = 1;
  header.Channels = channels;
  header.SampFreq = sampleRate;
  header.BytesPerSec = bytesPerSec;
  header.BytesPerSamp = channels*bitRate/8;
  header.BitsPerSamp = bitRate;
  strncpy(header.data, "data", 4);
  header.WaveSize = waveSize;

  out.write((const char*)(&header), sizeof(Aquila::WaveHeader));

  return 0;
}

int writeData(ofstream& out, uint8_t& channels, uint8_t& bitRate, vector<int>& dataLEFT, vector<int>& dataRIGHT)
{
  if(1 == channels)
  {
    if(8 == bitRate)
    {
      write8bitMono(out, dataLEFT);
      return 0;
    }
    else if(16 == bitRate)
    {
      write16bitMono(out, dataLEFT);
      return 0;
    }
  }

  if(2 == channels)
  {
    if(8 == bitRate)
    {
      write8bitStereo(out, dataLEFT, dataRIGHT);
      return 0;
    }
    else if(16 == bitRate)
    {
      write16bitStereo(out, dataLEFT, dataRIGHT);
      return 0;
    }
  }

  return 1;
}

int write8bitMono(ofstream& out, vector<int>& data)
{
  for(auto it = data.begin(); it != data.end(); it++)
  {
    uint8_t sample = static_cast<uint8_t>(*it);
    out.write((char*)&sample,1);
  }
  return 0;
```

```
}

int write16bitMono(ofstream& out, vector<int>& data)
{
  for(auto it = data.begin(); it != data.end(); it++)
  {
    int16_t sample = static_cast<int16_t>(*it);
    out.write((char*)&sample,2);
  }
  return 0;
}

int write8bitStereo(ofstream& out, vector<int>& dataLEFT, vector<int>& dataRIGHT)
{
  auto lit = dataLEFT.begin();
  auto rit = dataRIGHT.begin();

  for(; lit != dataLEFT.end() && rit != dataRIGHT.end(); lit++, rit++)
  {
    uint8_t sampleLEFT = static_cast<uint8_t>(*lit);
    uint8_t sampleRIGHT = static_cast<uint8_t>(*rit);
    out.write((char*)&sampleLEFT,1);
    out.write((char*)&sampleRIGHT,1);
  }
  return 0;
}

int write16bitStereo(ofstream& out, vector<int>& dataLEFT, vector<int>& dataRIGHT)
{
  auto lit = dataLEFT.begin();
  auto rit = dataRIGHT.begin();

  for(; lit != dataLEFT.end() && rit != dataRIGHT.end(); lit++, rit++)
  {
    int16_t sampleLEFT = static_cast<int16_t>(*lit);
    int16_t sampleRIGHT = static_cast<int16_t>(*rit);
    out.write((char*)&sampleLEFT,2);
    out.write((char*)&sampleRIGHT,2);
  }
  return 0;
}
```

decoding.cpp

```
#include <vector>
#include <cmath>
#include <string>

#include "decoding.h"

using namespace std;

int decode(vector<int>& peaks, vector<int>& gaps, vector<bool>& bits, int mPeak, int mGap)
{
  int bPeak = log2(mPeak);  //Parametri Golomba-Raisa kodam
  int bGap = log2(mGap);  //Tiek aprekinati vienreiz un atklata veida doti Golomba-Raisa funkcijai

  auto bit = bits.begin();

  peaks.push_back(invGolomb(bits, bit, mPeak, bPeak));

  while(bit != bits.end())
  {
    gaps.push_back(invGolomb(bits, bit, mGap, bGap));
    peaks.push_back(invGolomb(bits, bit, mPeak, bPeak));
  }

  return 0;
}

int invGolomb(vector<bool>& bits, vector<bool>::iterator& bit, int m, int b)
{
  int q = 0; //kvocients
  int r = 0; //atlikums

  while(0 != *bit)  //Saskaita unaro dalu, lai iegutu kvocientu
  {
    q++;
    bit++;
  }
  bit++;  //Pec unaras dalas seko 0

  for(int i=0; i<b; i++)  //Noskaidro atlikumu no binaras dalas
  {
    r *= 2;
    r += *bit;
    bit++;

  }

  return q*m+r;
}

int decompress(vector<int>& peaks, vector<int>& gaps, vector<int>& chan, string& method, int& firstSign, int& firstChange)
{
  auto pit = peaks.begin();
  auto git = gaps.begin();
```

```
  *pit *= firstSign;

  int prevVal = *pit;
  pit++;

  if(-1 == firstChange)  //ja vispirms dilst, tad to apstradajam atseviski
  {
    *pit = prevVal - *pit;
    prevVal = *pit;
    pit++;
  }

  while(pit != peaks.end())  //No starpibam rekonstruejam absolutas vertibas
  {
    *pit += prevVal;
    prevVal = *pit;
    pit++;

    if(pit == peaks.end())
      break;

    *pit = prevVal - *pit;
    prevVal = *pit;
    pit++;
  }

  pit = peaks.begin();
  chan.push_back(*pit);
  prevVal = *pit;
  pit++;

  int (*interpolate)(int i, int gap, int prevVal, int nextVal);

  if(method == "linear")
    interpolate = &linearGapInterpolation;

  if(method == "poly")
    interpolate = &polyGapInterpolation;

  if(method == "trig")
    interpolate = &trigGapInterpolation;

  while(pit != peaks.end())  //izsauc interpolacijas funkciju un rekonstrue signalu
  {
    for(int i=0; i<*git; i++)
      chan.push_back(interpolate(i, *git, prevVal, *pit));

    chan.push_back(*pit);
    prevVal = *pit;

    pit++;
    git++;

  }

  return 0;
}

int linearGapInterpolation(int i, int gap, int prevVal, int nextVal)
{
  int x = i + 1;
  int d = gap + 1;
  return prevVal + (nextVal-prevVal)*x/d;
}

int polyGapInterpolation(int i, int gap, int prevVal, int nextVal)
{
  int x = i + 1;
  int d = gap + 1;
  return prevVal + ((nextVal-prevVal) * x*x * (3*d - 2*x) /(d*d*d));
}

int trigGapInterpolation(int i, int gap, int prevVal, int nextVal)
{
  int x = i + 1;
  int d = gap + 1;
  return prevVal + ((nextVal-prevVal) * (1 - cos(pi()*x/d) ) / 2);
}

double pi()
{
  return 3.141592653589793;
}
```

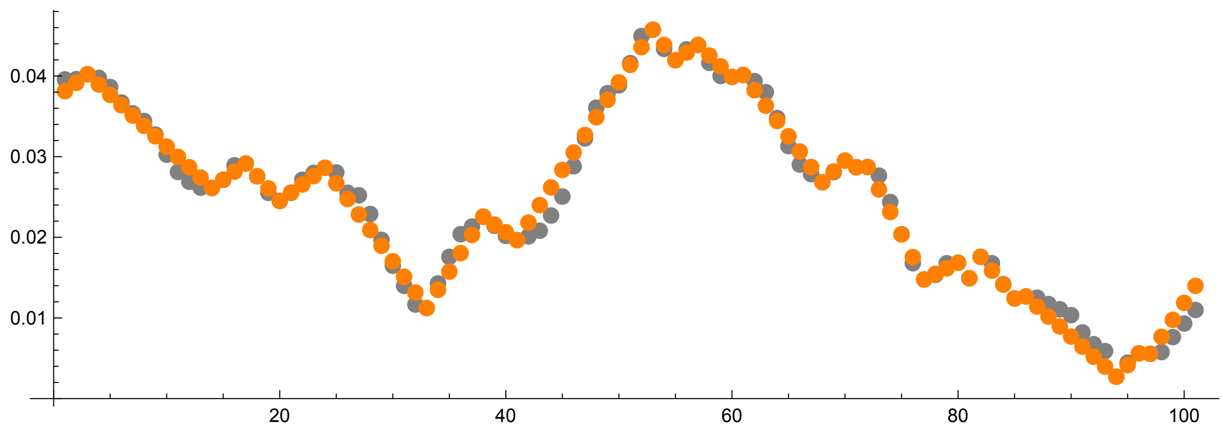
C Dažādu saspiešanas algoritmu rekonstruēto signālu attēli un spektrogrammas

Salīdzināšanas un izglītošanās nolūkā *Totenkopf* dziesmai *Burn the Desert* tika pielietoti dažādi saspiešanas algoritmi (iekavās saspiestās datnes apjoms):

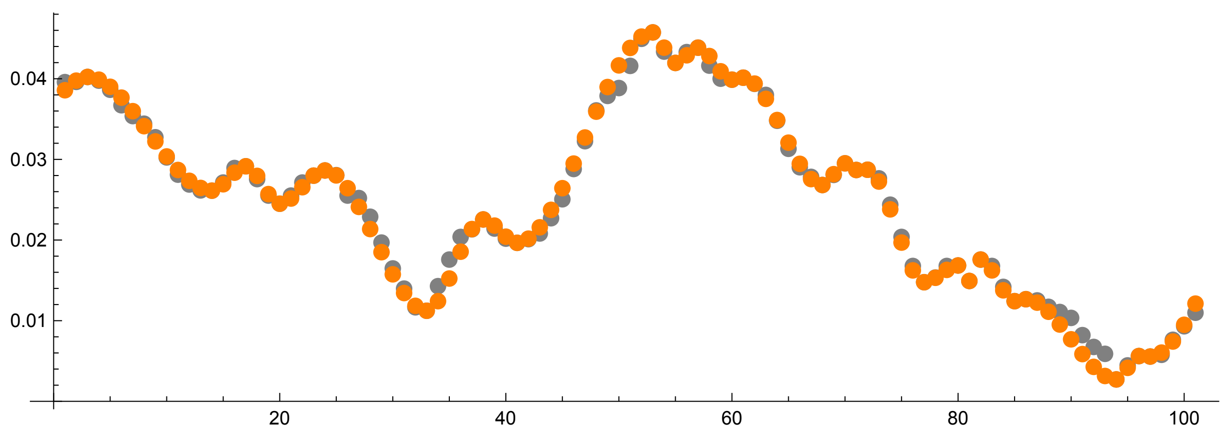
- *TDC*, signālu rekonstruējot ar lineāru interpolāciju (10177 kB)
- *TDC*, signālu rekonstruējot ar polinomiālu interpolāciju (10177 kB)
- *TDC*, signālu rekonstruējot ar trigonometrisku interpolāciju (10177 kB)
- *AAC* ar datu blīvumu 128 kilobiti sekundē (3274 kB)
- *MP3* ar datu blīvumu 140 kilobiti sekundē (3658 kB)
- *Vorbis* (jeb *Ogg Vorbis*, tālāk arī *Ogg*) ar datu blīvumu 121 kilobits sekundē (3164 kB)

Tika iegūtas šo algoritmu reproducēta signāla spektrogrammas un attēloti arī nelieli šī signāla fragmenti.

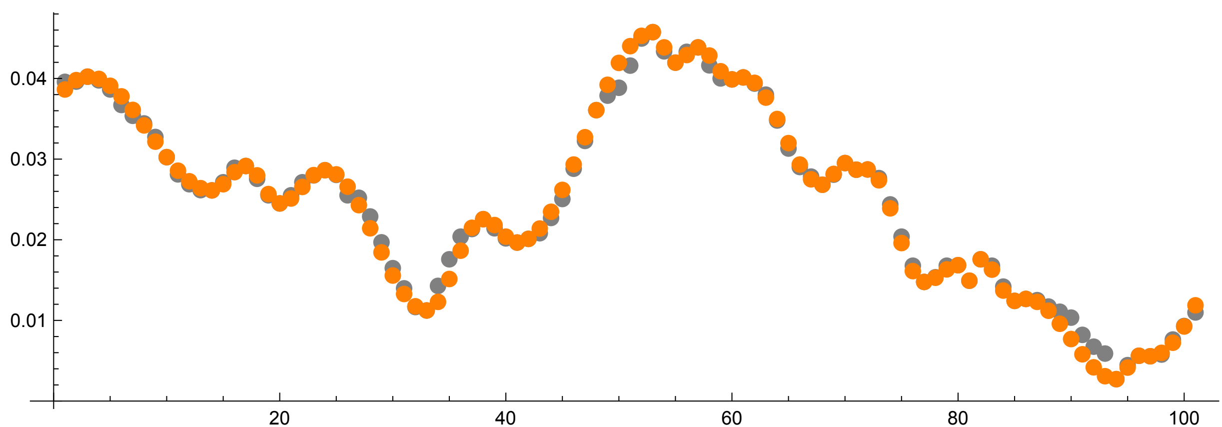
C1. Signālu attēli



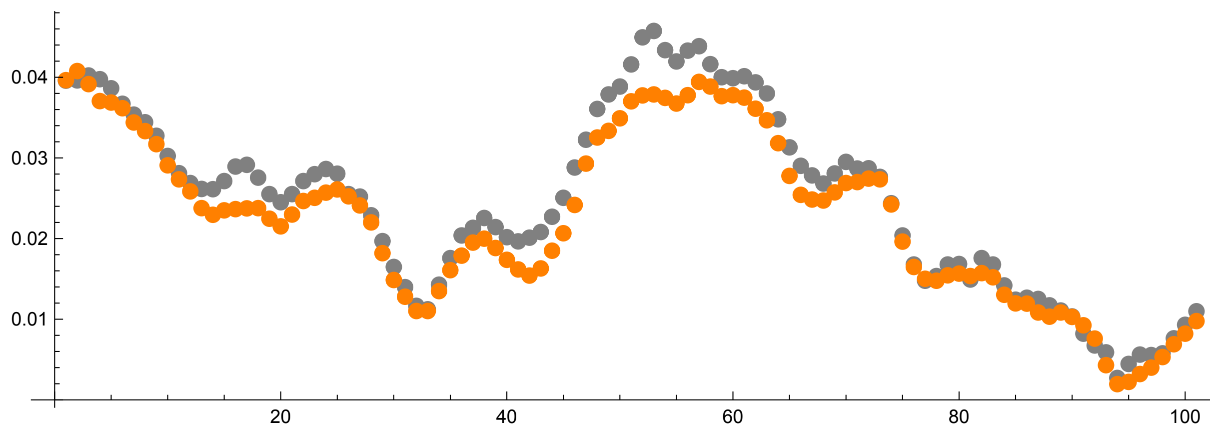
C1. Att.: Oriģinālā signāla (pelēks) un ar lineāru interpolāciju rekonstruēta signāla (oranžs) salīdzinājums.



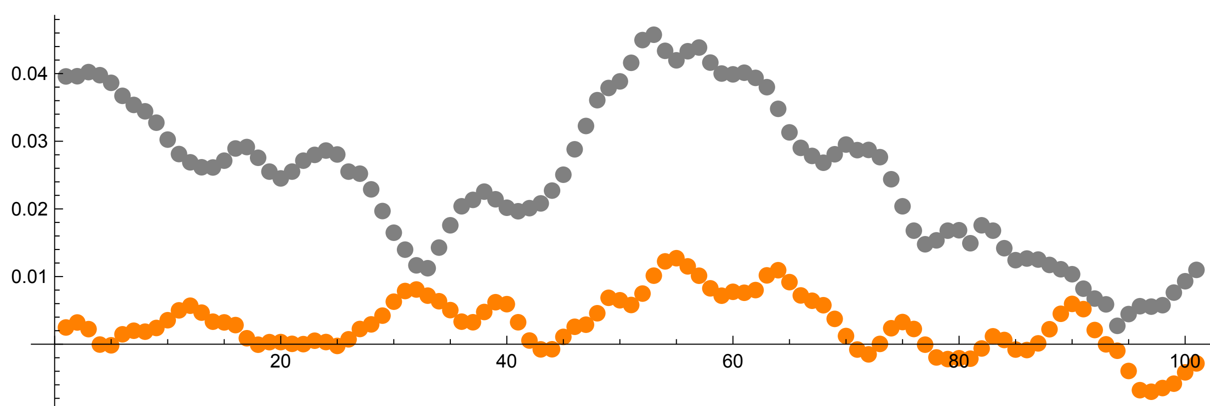
C2. Att.: Oriģinālā signāla (pelēks) un ar polinomiālu interpolāciju rekonstruēta signāla (oranžs) salīdzinājums.



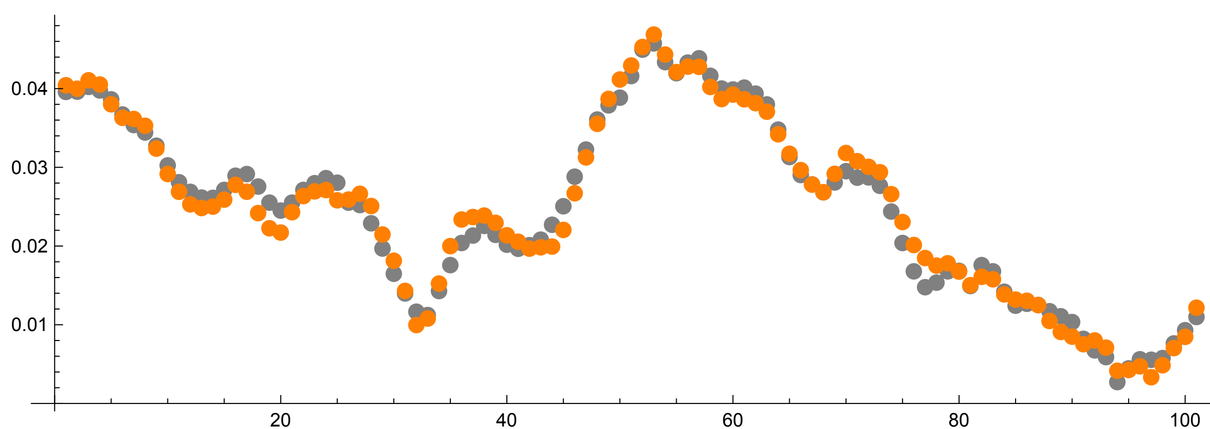
C3. Att.: Oriģinālā signāla (pelēks) un ar trigonometrisku interpolāciju rekonstruēta signāla (oranžs) salīdzinājums.



C4. Att.: Oriģinālā signāla (pelēks) un ar *AAC* algoritmu reproducētā signāla (oranžs) salīdzinājums.



C5. Att.: Oriģinālā signāla (pelēks) un ar *MP3* algoritmu reproducētā signāla (oranžs) salīdzinājums.

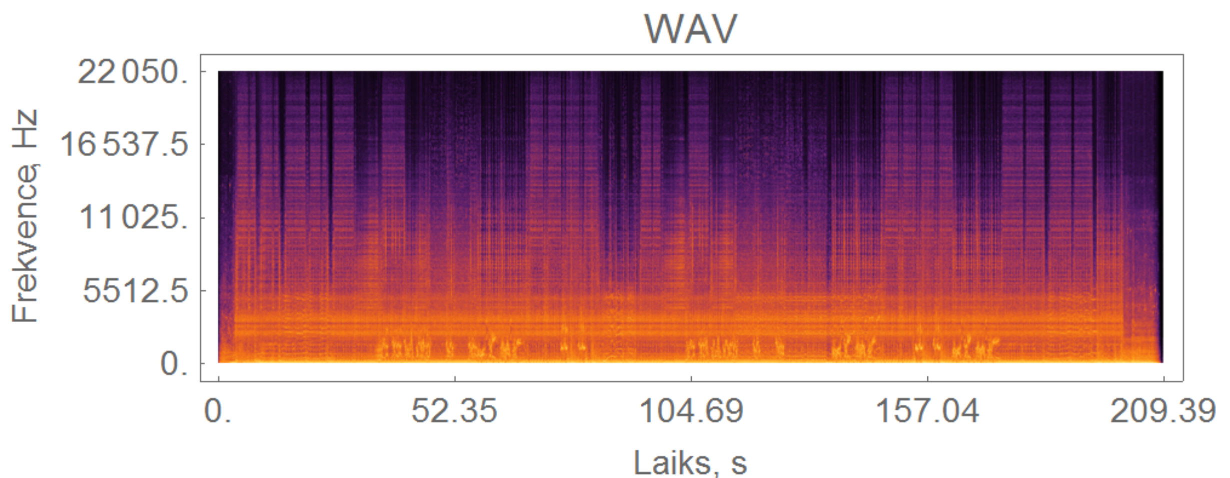


C6. Att.: Oriģinālā signāla (pelēks) un ar *Ogg Vorbis* algoritmu reproducētā signāla (oranžs) salīdzinājums.

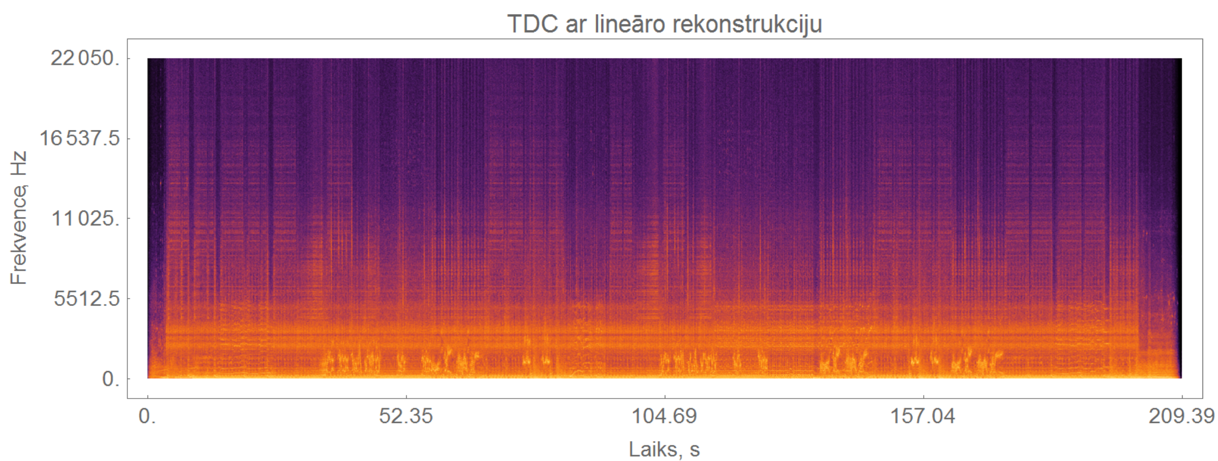
C2. Signālu spektrogrammas

Signālu spektrogrammas ir attēli, kuros redzamas frekvenču intensitātes atkarībā no laika. Piemēram, ja C8. attēlā laika momentā ap $50s$ attēla augšdaļa (rajonā ap $20000Hz$) ir melna, bet momentā ap $150s$ augšdaļa ir violeta, tas interpretējams tā, ka momenta ap $50s$ augstās frekvences signāla nav, taču pie $150s$ tādas ir saklausāmas. Visos laika momentos intensīvākas ir zemās frekvences—to parāda gaišāka krāsa. Krāsu shēmu, kas izmantota attēlos, var aplūkot C7. attēlā. Tumšākās krāsas (kreisā puse) atbilst mazai frekvences amplitūdai (sākot no 0), gaišākās krāsas (tuvāk labajai pusei) atbilst lielākai frekvences amplitūdai (līdz pat maksimālajai amplitūdai konkrētajā spektrogrammā—skala tiek relatīvi pieskaņota katras spektrogrammas diapazonam). Jāpiebilst, ka enerģijas daudzums un tās plūsma (jauda) katrā no frekvencēm ir proporcionāla attēlotās frekvences amplitūdas kvadrātam.

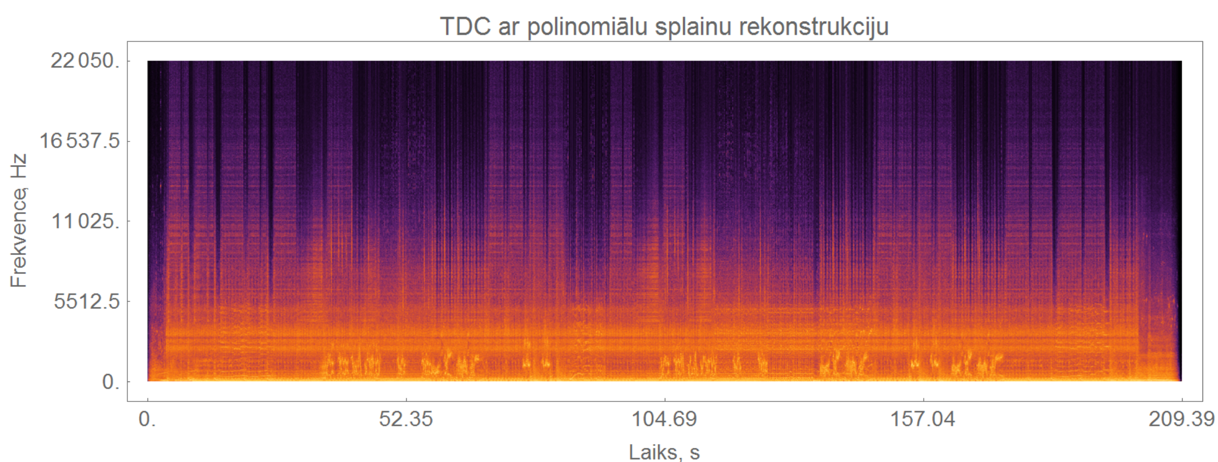
C7. Att.: Spektrogrammās izmantotā krāsu skala.



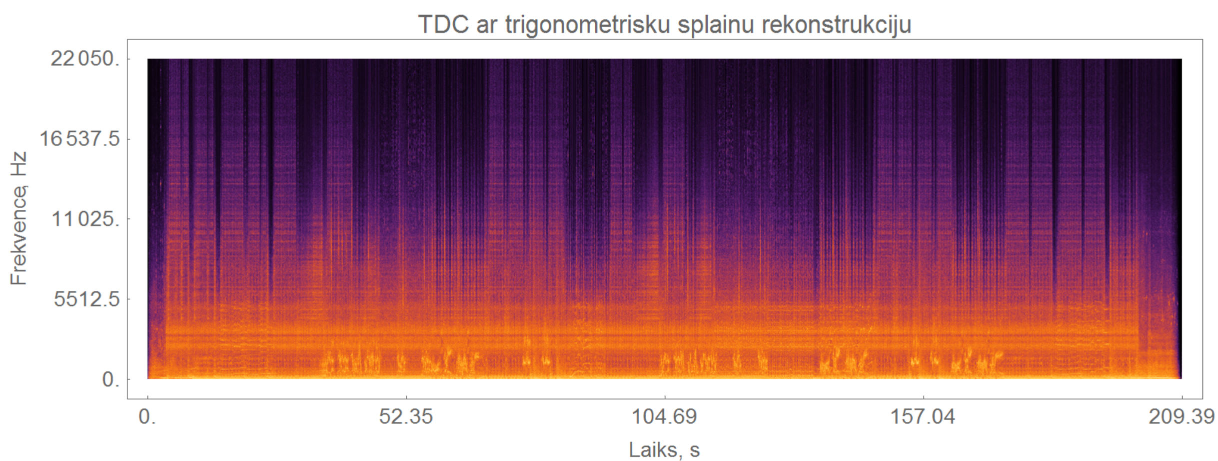
C8. Att.: Oriģinālā signāla spektrogramma.



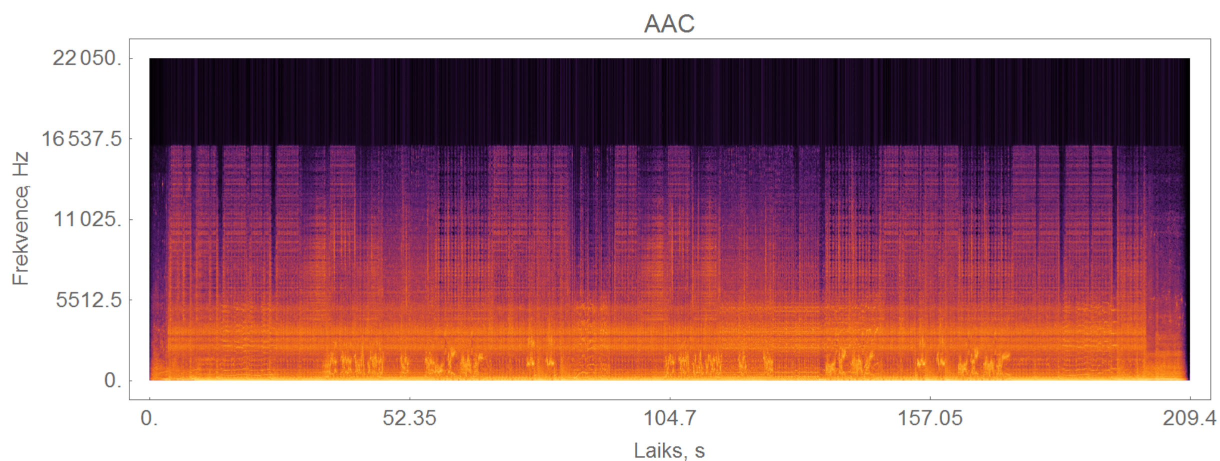
C9. Att.: Ar lineāru interpolāciju rekonstruēta signāla spektrogramma.



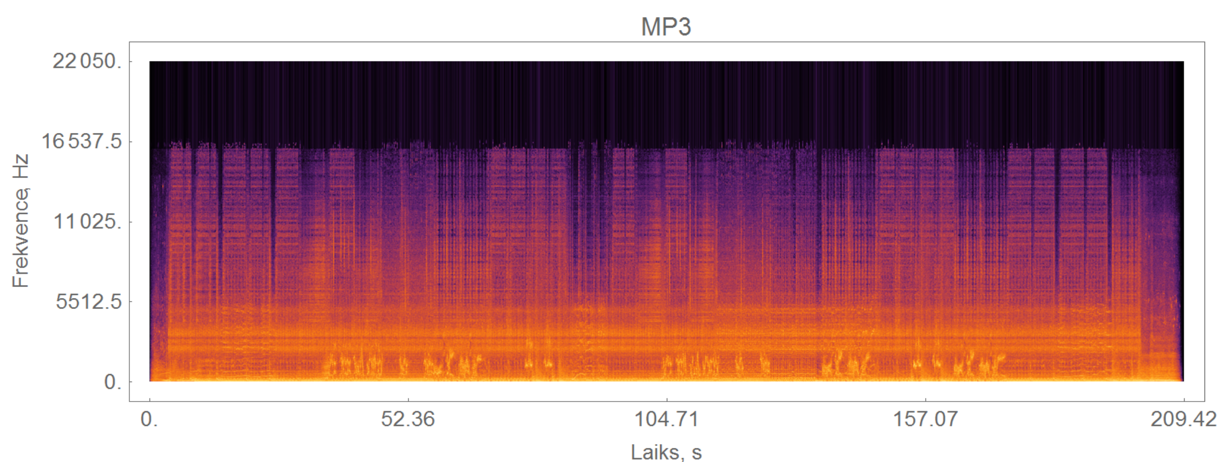
C10. Att.: Ar polinomiālu interpolāciju rekonstruēta signāla spektrogramma.



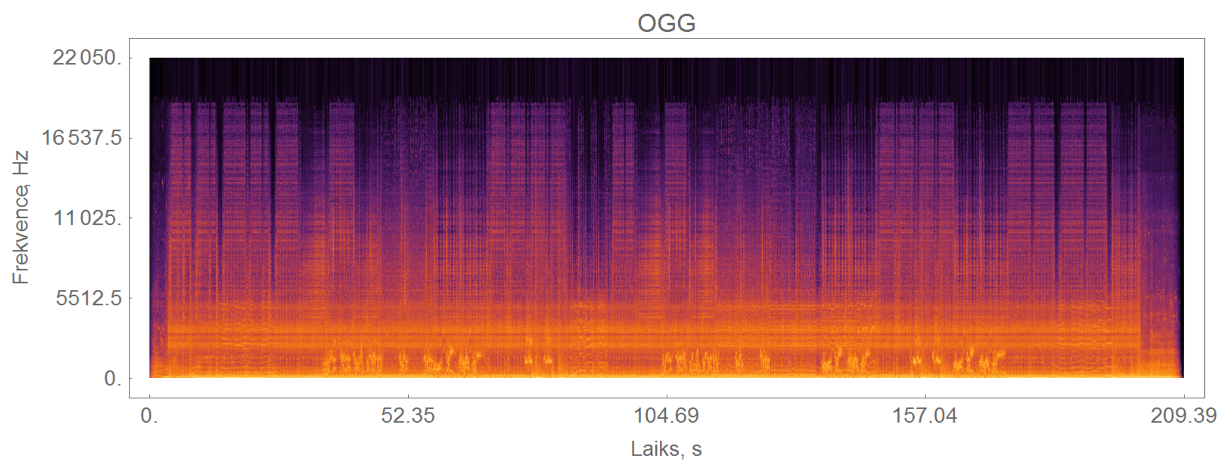
C11. Att.: Ar trigonometrisku interpolāciju rekonstruēta signāla spektrogramma.



C12. Att.: Ar *AAC* algoritmu reproducēta signāla spektrogramma.



C13. Att.: Ar *MP3* algoritmu reproducēta signāla spektrogramma.



C14. Att.: Ar *OGG* algoritmu reproducēta signāla spektrogramma.

D Algoritma realizācijas ātrdarbības pārbaude

Ātrdarbības eksperimenti tika veikti ar datoru, kam ir šādi parametri: *Intel Core i5-2500 @ 3.30GHz* procesors, 8 GB operatīvā atmiņa, *Samsung SSD 850 EVO* cietais disks.

Lai noskaidrotu *TDC* darbības ilgumu, tika izmantots šāds *Windows* komandrindas skripts (*.bat* datne):

```
@echo OFF

echo Compression start and end:
echo %time%
tdc-comp tkpf.wav speed.tdc
echo %time%

echo Linear decompression start and end:
echo %time%
tdc-decomp speed.tdc rec.wav
echo %time%

echo Polynomial decompression start and end:
echo %time%
tdc-decomp speed.tdc rec.wav poly
echo %time%

echo Trigonometrical decompression start and end:
echo %time%
tdc-decomp speed.tdc rec.wav trig
echo %time%
```

Skripts izsauc *TDC*, kas saspiež datni *in.wav* un pēc tam rekonstruē to ar trim dažādām interpolācijas metodēm. Izpildes gaitā komandrindā tiek izvadīts katra procesa sākuma un beigu laiks, no kuriem iespējams aprēķināt procesa ilgumu.

Vispirms skripts tika desmit reizes izpildīts dziesmai *Cave In - Anchor*, lai novērtētu izpildes laiku fluktuācijas. No iegūtajiem rezultātiem (skat. D1. tab.) tika atrastas laiku minimālās un maksimālās vērtības, diapazons, kādā tās svārstās un aprēķināts relatīvais svārstību diapazons—svārstību diapazona un laika vidējās vērtības attiecība. Secināts, ka izpildes laiki svārstās mazāk nekā par 1%. Arī statistiskas metodes to apliecina—vērtību

D1. Tabula: Dziesmas *Cave In - Anchor* saspiešanas laiks T_S un atspiešanas laiks ar lineāro (T_L), polinomiālo (T_P) un trigonometrisko (T_T) rekonstrukciju, izmantojot *TDC*.

Mēģinājuma nr.	T_S, s	T_L, s	T_P, s	T_T, s
1	3,07	6,15	5,90	6,76
2	3,05	6,16	5,89	6,77
3	3,05	6,16	5,90	6,75
4	3,05	6,16	5,90	6,76
5	3,04	6,15	5,91	6,76
6	3,05	6,17	5,91	6,76
7	3,05	6,16	5,90	6,76
8	3,06	6,17	5,89	6,76
9	3,04	6,16	5,89	6,75
10	3,04	6,16	5,90	6,76
minimālais	3,04	6,15	5,89	6,75
maksimālais	3,07	6,17	5,91	6,77
svārst. diapazons	0,03	0,02	0,02	0,02
vidējais	3,05	6,16	5,90	6,76
relat. svārst.	0,98%	0,32%	0,34%	0,30%
3σ	0,028	0,020	0,022	0,017

standartnovirzes (σ) ir tik nelielas, ka 3σ visos gadījumos ir mazāk nekā 1% no vērtības. Pieņemot, ka mērījumu vērtības ir gadījuma lielumi, kas pakļaujas normālajam sadalījumam, pēc 3σ likuma varam pieņemt, ka 99,7% gadījumu mērījuma vērtība neatšķirsies no šī lieluma vidējās (*patiesās*) vērtības. Tāpēc turpmākos mērījumus veiksim, izdarot vienu mērījumu un uzskatot, ka tas ar 1% precizitāti atbilst īstajam izpildes laikam.

Aizdomas radīja novērojums, ka rekonstrukcija ar polinomiālo interpolāciju ir ātrāka nekā rekonstrukcija ar lineāro interpolāciju. Tika mainīta rekonstrukciju secība un rekonstrukcijas izpildītas atsevišķi, lai pārbaudītu, vai nav tā, ka pēc pirmās programmas izpildes apstrādājamā datne vairs nav atkārtoti jānolasa no cietā diska. Tika noskaidrots, ka šie mērījumi nav kļūdaini—patiešām polinomiālā rekonstrukcija strādā ātrāk nekā lineārā.

Pēc precizitātes noskaidrošanas skripts tika izpildīts visiem 12 pētāmajiem ierakstiem un iegūti rezultāti, kas aplūkojami D2. tabulā.

Pēc tam tika izmantota programma *SoX*[38], lai šos pašu ierakstus saspiestu arī ar *MP3*, *FLAC* un *Ogg Vorbis* algoritmiem. Tika izmantotas noklusētās parametru vērtības:

- *MP3* tika pielietots ar 128 kb/s bitu biežumu un kvalitāti 5;
- *Ogg Vorbis* tika pielietots ar kvalitāti 3 (atbilst aptuveni 112 kb/s);
- *FLAC* tika pielietots ar kompresijas līmeni -8.

Mērījumu veikšanai izmantots iepriekšējam līdzīgs komandrindas skripts, kurā *TDC* programmu vietā darbojas *SoX* programma:

D2. Tabula: Saspiešanas laiks T_S un atspiešanas laiks ar lineāro (T_L), polinomiālo (T_P) un trigonometrisko (T_T) rekonstrukciju, izmantojot *TDC*.

Dziesma	T_S, s	T_L, s	T_P, s	T_T, s
CIA	3,05	6,16	5,90	6,76
JSD	4,99	9,08	8,74	9,54
JBM	4,20	8,36	8,02	8,94
KDZ	3,67	7,46	7,14	8,03
LRW	3,47	7,15	6,85	7,69
MDF	3,48	7,10	6,81	7,70
MRU	4,22	8,89	8,56	9,68
MHL	4,30	8,80	8,45	9,49
OBT	4,18	9,70	9,27	10,69
SFI	3,82	9,09	8,74	10,36
ADA	4,08	8,72	8,35	9,65
TDP	5,76	11,20	10,71	12,08

```
@echo OFF

echo Vorbis compression start and end:
echo %time%
sox in.wav ogg.ogg
echo %time%

echo Vorbis decompression start and end:
echo %time%
sox ogg.ogg tempv.wav
echo %time%

echo FLAC compression start and end:
echo %time%
sox in.wav flac.flac
echo %time%

echo FLAC decompression start and end:
echo %time%
sox flac.flac tempf.wav
echo %time%

echo MP3 compression start and end:
echo %time%
sox in.wav mp3.mp3
echo %time%

echo MP3 decompression start and end:
echo %time%
sox mp3.mp3 temp3.wav
echo %time%
```

D3. Tabula: Saspiešanas un atspiešanas laiks (sekundēs) ar dažādiem skaņas saspiešanas algoritmiem.

Dziesma	$T_S(MP3)$	$T_R(MP3)$	$T_S(Vorbis)$	$T_R(Vorbis)$	$T_S(FLAC)$	$T_R(FLAC)$
CIA	10,56	1,71	4,56	0,55	1,65	0,44
JSD	11,54	2,09	5,62	0,70	1,94	0,52
JBM	13,25	2,07	5,94	0,71	2,01	0,51
KDZ	9,85	1,90	5,05	0,62	1,88	0,49
LRW	10,34	1,86	4,91	0,61	1,76	0,46
MDF	11,57	1,83	4,95	0,61	1,70	0,46
MRU	12,94	2,36	6,29	0,76	2,24	0,59
MHL	13,37	2,25	6,02	0,73	2,21	0,58
OBT	16,74	2,71	7,44	0,91	2,63	0,67
SFI	19,20	3,03	8,01	0,99	2,70	0,71
ADA	16,54	2,54	6,89	0,84	2,38	0,64
TDP	20,26	3,01	7,98	0,99	2,71	0,73

Iegūtie dati attēloti D3. tabulā. Ar $T_S(x)$ apzīmēts algoritma *x* saspiešanas procesa ilgums sekundēs, bet ar $T_R(x)$ apzīmēts algoritma *x* signāla rekonstruēšanas procesa ilgums sekundēs.

E Rekonstruētā signāla kvalitātes testi

E1. Paraugu apraksts

Tika sagatavoti 15 atšķirīgi skaņas paraugi. Sagatavotie skaņas paraugi atrodami maģistra darba digitālajā pielikumā [36].

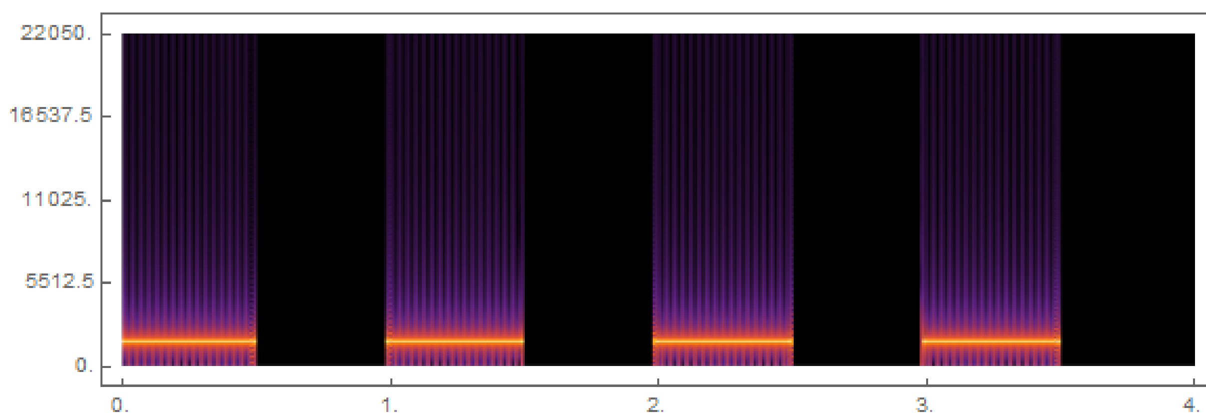
Paraugi tika apzīmēti ar neitrāliem nosaukumiem (piemēram, *song1*), kas izmantoti to datņu nosaukumus. Piemēram, ja parauga apzīmējums ir *sound1*, tad oriģinālais ieraksts atradās datnē *sound1A.wav*, rekonstruētais—*sound1B.wav* un nezināmais—*sound1X.wav*. Visām trim datnēm bija identisks apjoms, *A* un *B* paraugu datnes nebija atšķiramas arī pēc laika zīmoga—tika nodrošināts, lai nav nejauši pamanāmas atšķirības starp *A* un *B* datnēm.

Tiesa, datnes bija iespējams atšķirt, aplūkojot to saturu (tekstuālā reprezentācijā) vai saspiežot tās *zip* vai līdzīgā datnē (saspiesto versiju izmēri atšķiras). Ņemot vērā arī to, ka eksperiments notika neklātienē, tas nozīmē, ka nācās paļauties uz subjektu godīgumu. Jāpiebilst, ka atsevišķi eksperimenta dalībnieki ir izmantojuši vairākas atskaņošanas ierīces, lai atšķirtu paraugus gadījumā, kad ar vienu no atskaņošanas ierīcēm atšķirību nevarēja saklausīt.

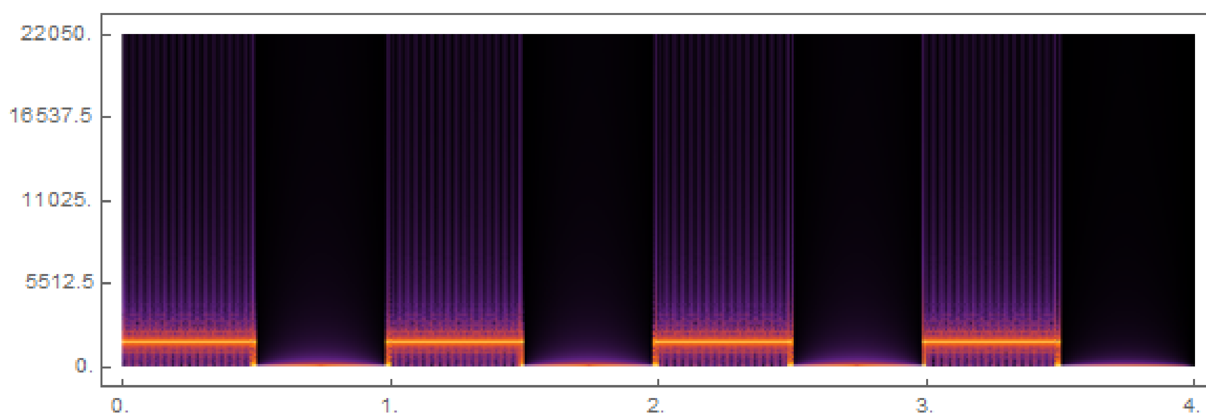
Paraugu rekonstrukcijā tika izmantota interpolācija ar trigonometriskiem splainiem. Ja signāla attēlos būtiskas atšķirības starp interpolācijām nav pamanāmas (tas noprotams jau no 2.7. att., skatīt arī C pielikumu), tad analizējot vienkāršu signālu spektrogrammas (skat. E1. att.) tika pamanīts, ka, rekonstruējot signālu ar trigonometriskiem splainiem (skat. E2. att.) defektu ir mazāk nekā rekonstrukcijā ar polinomiāliem splainiem (skat. E3. att.). Tiesa, rekonstrukcijā ar trigonometriskiem splainiem parādās nenulles spektrs tur, kur iepriekš bijis klusums. Ņemot vērā, ka vairumā paraugu klusuma periodu nav, tika nolemts visu paraugu rekonstrukcijai izmantot trigonometriskos splainus.

Seši no paraugiem bija jau iepriekšējos eksperimentos izmantoto dziesmu fragmenti (skat. E1. tab.). Paraugi tika izvēlēti tā, lai būtu dažāda skaļuma un piesātinājuma mūzika—sākot ar mierīgām viena instrumenta partijām ar un bez vokāla un beidzot ar piesātinātiem roka un elektroniskās mūzikas fragmentiem.

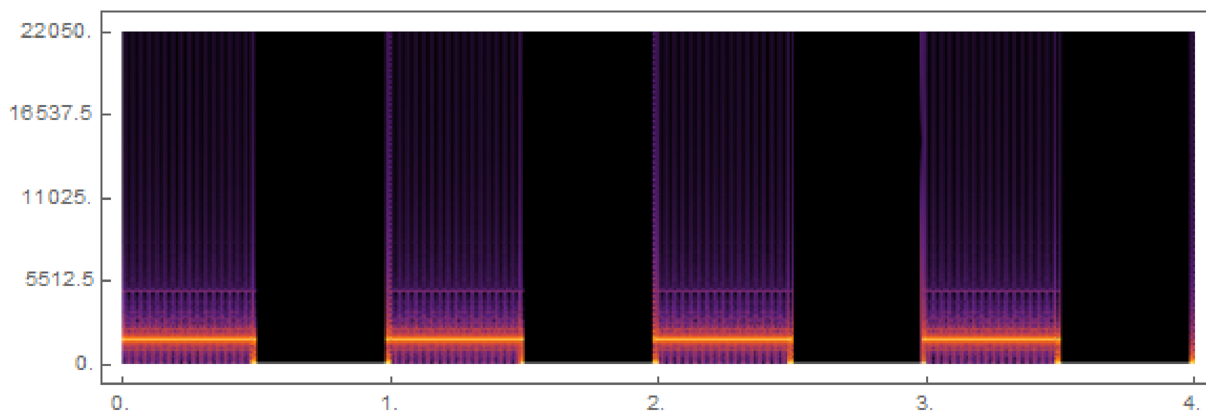
Seši no paraugiem bija vienkāršas skaņas (skat. E2. tab.) . Trīs no tiem bija viens un tas pats sešu nošu motīvs dažādās oktāvās, izpildīts klavieru skaņās ar datora *MIDI* atskaņotāju. Divi paraugi sastāvēja katrs no 4 pussekundi gariem pīkstieniem ar pussekundi ilgam pauzēm. Paraugi atšķīrās ar pīkstienu signāla frekvenci. Pīkstieni tika radīti ar *Wolfram*



E1. Att.: Parauga *sound2* (četri 1600 Hz pīkstieni) spektrogramma.



E2. Att.: Parauga *sound2* rekonstrukcijas ar trigonometriskiem splainiem spektrogramma.



E3. Att.: Parauga *sound2* rekonstrukcijas ar polinomiāliem splainiem spektrogramma.

E1. Tabula: Eksperimentā izmantotie dziesmu fragmenti.

Dziesma	Apzīmējums	Ilgums
Cave In - Anchor	song1	0:05
Jamiroquai - Seven Days in Sunny June	song2	0:08
Kenny Loggins - Danger Zone	song3	0:05
Madeon - Finale	song4	0:09
Metallica - Hit the Lights	song5	0:07
OceanLab - Breaking Ties	song6	0:05

E2. Tabula: Eksperimentā izmantotās vienkāršās skaņas.

Paraugs	Apzīmējums	Ilgums
160 Hz pīkstiens	sound1	0:04
1600 Hz pīkstiens	sound2	0:04
Klavieru motīvs 2. oktāvā	sound3	0:03
Klavieru motīvs 2. oktāvā	sound4	0:03
Klavieru motīvs 2. oktāvā	sound5	0:03
Piecas alta notis	sound6	0:12

E3. Tabula: Eksperimentā izmantotie balss ieraksti.

Paraugs	Apzīmējums	Ilgums
Apollo 13 misijas radiosarunas	voice1	0:02
Izsauciens *Ah, come on now baby*	voice2	0:01
Lasījums no Stīvena Hokinga grāmatas	voice3	0:09

Mathematica palīdzību, diskretizējot sinusoīdu. Vēl viens paraugs—alta skaņas—tika paņemts no *Wolfram Mathematica* skaņas paraugu bibliotēkas.

Trīs no paraugiem bija balss ieraksti (skat. E3. tab.). Radiosarunu ieraksts no *Apollo 13* misijas tika paņemts no *Wolfram Mathematica* skaņas paraugu bibliotēkas. Izsauciens *Ah, come on now baby* ņemts no balss ierakstu bāzes internetā [40] un lasījumu no Stīvena Hokinga grāmatas *Diženais plāns* maģistra darba autors ierunāja pats.

E2. Eksperimenta detaļas

Šajā pielikuma nodaļā precizētas dažas detaļas, kas nav izskaidrotas darba pamatdaļā.

Eksperimenta dalībniekiem bija jānorāda šāda papildinformācija:

- Dzimums;
- Vecums;
- Atskaņošanas aparatūra, kas izmantota, veicot eksperimentu.

Tika dotas šādas rekomendācijas vērtējumu skalai:

- 10—ideāla kvalitāte, nav pamanāma atšķirība no oriģināla;
- 7-9—piemērota kvalitāte, ir pamanāmas atšķirības, taču šādam signālam tās netraucē;
- 4-6—saprotama, bet nepiemērota kvalitāte, ir lielas atšķirības no oriģināla un tās traucē šādam signālam;

E4. Tabula: Eksperimenta dalībnieki.

Numurs	Vecums	Dzimums	Izmantotā aparatūra
1	24	Vīrietis	Philips Headphones SHL3000
2	25	Sieviete	Portatīvā skaļrunis
3	25	Vīrietis	Austiņas
4	49	Vīrietis	Beats Solo HD Headphones
5	24	Sieviete	Portatīvā datora skaļrunis
6	26	Sieviete	Skaļrunis
7	24	Vīrietis	Philips SHP 2500
8	22	Sieviete	Philips (SHE3590) tamponaustiņas
9	30	Vīrietis	Razer Electra austiņas
10	25	Sieviete	Radiotehnikas 2 S-90F un 2 S-30 skaļruņi
11	49	Vīrietis	Arcam pastiprinātajs, JMLab skaļrunīši
12	29	Sieviete	Portatīvā datora skaļrunis (Lenovo)
13	35	Vīrietis	Austiņas Panasonic (ausīs liekamās)
14	25	Vīrietis	c2i4 sonY MDR-RF865R bezvadu austiņas
15	31	Vīrietis	Visaton B200 pilna spektra skaļruņi

- 1-3—nesaprotams signāls, kvalitāte ir pilnīgi nepiemērota šādam signālam.

Papildus subjektiem atbilžu lapā bija paredzēta vieta komentāru rakstīšanai par katru no paraugiem.

E3. Eksperimenta rezultāti

Eksperimentu veica 15 subjekti. Šī darba autors nebija neviens no eksperimenta subjektiem. Informācija par subjektiem aplūkojama E4. tabulā. Atsevišķos gadījumos darba autors ir saīsinājis izmantotās aparatūras aprakstu un veicis korekcijas ortogrāfijā. Citā veidā subjektu izmantotie aparatūras apzīmējumi nav izmainīti.

Subjektu dotās atbildes paraugu atpazīšanā redzamas E5. tabulā. Ja atbilde bijusi pareiza (t.i. paraugs X tiešām bija vienāds ar subjekta norādīto), tā iekrāsota zaļā krāsā.

Subjektu novērtējumi par rekonstruēto paraugu kvalitāti aplūkojami E6. tabulā.

E5. Tabula: Eksperimenta dalībnieku atbildes paraugu atpazīšanas jautājumā.

Numurs	song1	song2	song3	song4	song5	song6	sound1	sound2	sound3	sound4	sound5	sound6	voice1	voice2	voice3
1	A	A	B	B	A	A	B	A	B	B	B	B	B	A	A
2	B	B	A	B	B	B	A	A	B	B	B	A	A	A	B
3	B	B	A	B	A	A	A	B	B	B	A	B	A	A	B
4	A	B	A	A	B	A	B	B	B	A	A	B	B	B	B
5	A	B	A	B	A	B	A	A	B	A	B	B	B	A	B
6	B	B	B	B	A	A	A	A	B	A	B	A	A	B	B
7	B	A	B	B	B	A	B	A	B	B	B	B	A	B	B
8	A	A	B	B	B	A	B	B	A	A	A	A	B	A	A
9	A	B	B	B	B	B	B	A	B	A	A	B	A	B	A
10	B	B	B	B	A	B	A	A	B	A	A	B	B	A	A
11	A	A	B	A	A	B	B	B	A	B	B	A	A	B	B
12	B	A	A	B	A	B	A	A	A	B	B	A	B	B	B
13	B	A	A	A	A	A	A	B	B	B	B	A	B	A	A
14	A	B	A	A	A	B	A	B	B	A	B	A	B	A	A
15	A	A	B	B	B	A	B	A	B	A	B	B	B	A	A

E6. Tabula: Eksperimenta dalībnieku vērtējumi par paraugu kvalitāti.

Numurs	song1	song2	song3	song4	song5	song6	sound1	sound2	sound3	sound4	sound5	sound6	voice1	voice2	voice3
1	7	10	8	8	8	5	5	6	9	5	7	4	8	8	10
2	8	10	9	9	9	6	6	6	9	8	7	4	9	10	8
3	10	10	10	10	10	5	8	6	5	4	4	6	6	7	10
4	6	10	7	10	7	4	7	4	5	4	5	4	7	6	10
5	9	10	9	9	10	3	9	4	8	6	7	4	5	10	5
6	10	8	10	10	10	6	6	4	6	4	4	6	8	8	9
7	4	8	4	7	8	4	7	4	4	4	6	4	7	7	7
8	5	9	10	10	10	4	9	7	6	4	4	5	10	10	8
9	6	9	7	6	7	4	8	4	5	5	5	7	7	6	7
10	10	10	9	10	10	6	10	7	10	8	9	5	9	9	10
11	9	9	9	9	9	4	7	4	5	4	5	4	7	9	9
12	8	10	10	7	10	6	4	2	10	10	10	4	4	10	8
13	2	9	6	6	7	2	8	5	4	2	2	2	6	6	9
14	7	9	8	7	10	4	10	4	6	5	6	5	8	7	10
15	4	6	4	4	7	4	6	6	4	4	4	4	7	6	6

Izmantotā literatūra un avoti

[1] ISO/IEC 13818-7:2003 Information technology – Generic coding of moving pictures and associated audio information – Part 7: Advanced Audio Coding (AAC).

[2] Wikipedia. Advanced Audio Coding — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Advanced_Audio_Coding, 2015. [Tiešsaitē; 24.05.2015.].

[3] Wikipedia. Cyclic redundancy check — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Cyclic_redundancy_check, 2015. [Tiešsaitē; 24.05.2015.].

[4] W3C. HTML5. http://www.w3.org/TR/html5/, 2015. [Tiešsaitē; 24.05.2015.].

[5] Wikipedia. JPEG — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Cyclic_redundancy_check, 2015. [Tiešsaitē; 24.05.2015.].

[6] Wikipedia. Zip (file format) — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Zip_(file_format), 2015. [Tiešsaitē; 24.05.2015.].

[7] D. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[8] K.C. Pohlmann. *The Compact Disc: A Handbook of Theory and Use*. Computer Music and Digital Audio Series. A-R Editions, 1989.

[9] E. Šilters u.c. *Fizikas rokasgrāmata*. Zvaigzne, 1985.

[10] Peter Noll N.S. Jayant. *Digital Coding of Waveforms*. Prentice Hall, 1984.

[11] George Arfken. *Mathematical Methods for Physicists*. Academic Press, 1985.

[12] N. Ahmed, T. Natarajan, and K.R. Rao. Discrete Cosine Transform. *Computers, IEEE Transactions on*, C-23(1):90–93, Jan 1974.

[13] Wikipedia. Spectrogram — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Spectrogram, 2015. Tiešsaitē 20.05.2015].

[14] Terence C. Mills. *Time Series Techniques for Economists*. Cambridge University Press, 1991.

[15] Garry A. Einicke. *Smoothing, Filtering and Prediction - Estimating The Past, Present and Future*. InTech, 2012.

[16] *From Joint Stereo to Spatial Audio Coding - Recent Progress and Standardization*, 2004.

[17] Jānis Smotrovs. *Varbūtību teorija un matemātiskā statistika, 1. daļa*. Zvaigzne ABC, 2004.

[18] S.W. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, 12(3):399–401, 1966.

[19] R. Rice and J. Plaunt. Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data. *Communication Technology, IEEE Transactions on*, 19(6):889–897, December 1971.

[20] Recommendation G.711. 1993.

[21] S.A. Gelfand. *Hearing - An Introduction to Psychological and Physiological Acoustics*. Marcel Dekker, 2004.

[22] Wikipedia. Psychoacoustics — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Psychoacoustics, 2015. [Tiešsaitē; 27.01.2015.].

[23] B.C.J. Moore. *Cochlear Hearing Loss*. Whurr Publishers Ltd, 1998.

[24] J. Blauert. *Spatial Hearing: The PsychoPhysics of Human Sound Localization*. MIT Press, 1997.

[25] Xiph.Org Foundation. Ogg Vorbis Documentation. http://xiph.org/vorbis/doc/stereo.html. [Tiešsaitē; 28.10.2015.].

[26] Xiph.Org Foundation. FLAC - Free Lossless Audio Codec. [Tiešsaitē; 28.10.2015.].

[27] ISO/IEC 13818-3:1998 Information technology – Generic coding of moving pictures and associated audio information — Part 3: Audio.

[28] LAME MP3 Encoder. http://lame.sourceforge.net/. [Tiešsaitē; 28.10.2015.].

[29] W. A. Munson and Mark B. Gardner. Standardizing Auditory Tests. *The Journal of the Acoustical Society of America*, 22(5):675–675, 1950.

[30] Recommendation BS.1534-2. 2014.

[31] L. Tan. *Fundamentals of Analog and Digital Signal Processing*. AuthorHouse, 2008.

[32] Carl Runge. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Zeitschrift für Mathematik und Physik*, 46:224–243, 1901.

[33] Biruta Siliņa Kārlis Šteiners. *Augstākā matemātika, II daļa*. Zvaigzne ABC, 1998.

[34] L.D. Landau and E.M. Lifshitz. *Mechanics*. Butterworth Heinemann. Butterworth-Heinemann, 1976.

[35] Erwin Kreyszig. *Advanced Engineering Mathematics: Maple Computer Guide*. John Wiley & Sons, Inc., New York, NY, USA, 8th edition, 2000.

[36] Juris Evertovskis. Maģistra darba digitālais pielikums. http://totenkopf.lv/scomp, 2015. [Tiešsaitē; 24.05.2015.].

[37] Zbigniew Siciarz. Aquila 3.0 Open source DSP library for C++. http://aquila-dsp.org/. [Tiešsaitē; 10.05.2015.].

[38] SoX - Sound eXchange. http://sox.sourceforge.net/. [Tiešsaitē; 20.05.2015.].

[39] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition, 2002.

[40] Vocal Downloads. http://www.vocaldownloads.com/. [Tiešsaitē; 22.05.2015.].

Maģistra darbs “Skaņas ierakstu saspiešanas algoritms laika telpā ar zudumiem” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Juris Evertovskis

______________________ ______________________
(paraksts) (datums)

Rekomendēju darbu aizstāvēšanai.
Vadītājs: LU asoc. prof. Dr. Sc. Comp. Juris Vīksna

______________________ ______________________
(paraksts) (datums)

Recenzents: LU docents Dr. Sc. Comp. Kārlis Freivalds

______________________ ______________________
(paraksts) (datums)

Darbs iesniegts ______________________
(datums)

____
(darbu pieņēma)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

______________________ prot. Nr. __________, vērtējums__________
(datums)

Komisijas sekretārs/-e: ______________________ ______________________
(Vārds, Uzvārds) (paraksts)