SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 328

MODELIRANJE I SINTEZA GLAZBENIH INSTRUMENATA U REALNOM VREMENU KORIŠTENJEM MATLAB-A

Ivan Sokolić

Zagreb, lipanj 2011.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Zahvaljujem se prvenstveno svome mentoru prof. dr. sc. Davoru Petrinoviću na stručnoj pomoći, savjetima i strpljenju, bez kojih bi izrada ovoga rada bila nemoguća. Slijedeća zahvala ide gospođici Nikolini Ivetić na njenoj konstantnoj prisutnosti, podršci, empatičnosti, razumijevanju i motivaciji. Njoj posvećujem ovaj diplomski rad. Također joj se zahvaljujem i na posuđenom Yamaha PSR-520 sintesajzeru koji mi je puno pomogao pri istraživanju. Zahvaljujem se i svojim roditeljima: Ankici i Željku Sokolić na potpori, razumijevanju te što su bili uz mene tijekom cijelog mog akademskog školovanja.

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. Uvod

Zvuk je svuda oko nas. On je sastavni dio našeg svakodnevnog života, i često puta nismo niti svjesni onoga što čujemo. Još od malih nogu učimo povezivati zvukove sa stvarima i događajima, npr. zvuk prolazećeg automobila, cvrkut ptica, govor čovjeka i slično; niti ne razmišljajući o tome kako ih naš mozak percepira i što je to što jedan zvuk razlikuje od svih ostalih. Današnji razvoj tehnologije nam omogućuje iznimno detaljan uvid u procese koji se oko nas zbivaju i u mogućnosti smo umjetnim putem rekreirati jako puno njih. Međutim, još smo daleko od toga da možemo reći da nam je sve poznato i da više nemamo što istraživati i poboljšavati.

Glazba je, po mom skromnom mišljenju, jedno od čovjekovih najvećih i najsavršenijih dostignuća. Fleksibilnost i ekspresivnost su praktički neograničeni pri stvaranju glazbe, a u tome nam pomaže jako široka paleta instrumenata koje smo, tijekom vremena, konstruirali i usavršili. Karakter pojedinog instrumenta određuje način izrade i odabir materijala od kojih je isti napravljen, no najveći doprinos zvuku instrumenta daje upravo glazbenik koji svojim jedinstvenim umijećem i talentom izvlači maksimalno iz njega.

2. Osnove glazbene teorije

Proces kojim se dobiva nota nekog instrumenta je fizikalne prirode, kao na primjer titranje žice klavira ili strujanje zraka kroz cijev flaute, no glazba kao skup nota u određenom vremenu, ima korijene u matematici. Osnovni elementi poput visine tona, trajanja note, ritma, melodije, harmonije, teksture, boje tona, dinamike, te ostalih, određuju strukturu glazbenog djela, te se mogu odrediti prema nekim osnovnim matematičkim pravilima, koja ćemo u daljnjem tekstu uzimati u obzir.

Prije nego li se upustimo u modeliranje pojedinih instrumeata, potrebno je objasniti neke osnove glazbene teorije koja će nam pomoći u shvaćanju metoda modeliranja i njihovoj implementaciji u okviru ovoga rada.

2.1. Elementi glazbenih nota

Note su osnovni elementi od kojih se sastoji neko glazbeno djelo. Svaka nota, ili skup nota, ima određena svojstva koja utječu na ukupan perceptivni doživljaj koji čujemo kao glazbu. Odabir tih svojstava je u načelu proizvoljan, ali postoje određena pravila kojima se poboljšava kvaliteta glazbenog djela, no ovdje nećemo ulaziti dublje u njih. Međutim, razmotriti ćemo par osnovnih svojstava nota, koja će nam poslužiti kao osnova i prvi korak pri modeliranju instrumenata.

2.1.1. Visina tona

Visina tona je glavno slušno svojstvo nota koje omogućuje da ih svrstamo po frekvencijskoj skali, koja se mjeri u Hertz-ima (Hz). Premda su frekvencija i visina tona usko povezane, to nisu ekvivalentni pojmovi. Frekvencija je objektivan pojam dok je visina tona subjektivan. Zvučni valovi, kao takvi, nemaju visinu ali njihove oscilacije se mogu izmjeriti te se na taj način može doći do frekvencije njihova titranja. Međutim, izmjerenu frekvenciju titranja naš mozak može drugačije interpretirati, pogotovo ako je zvuk bogat harmonicima (spektralnim komponentama).

Bez obzira na broj harmonika, od kojih se sastoji zvučni val, ljudsko uho prepoznaje o kojoj se visini tona radi na temelju osnovnog harmonika. Na primjer, ako odsviramo notu A4 (440 Hz) na klaviru i na gitari, u oba slučaja ćemo prepoznati da se radi o istoj noti. No, postoje i slučajevi kada osnovni harmonik može nedostajati iz spektra signala, ali visina tona svejedno može biti prepoznatljiva ako je ostatak spektra jasno koreliran sa osnovnim harmonikom.

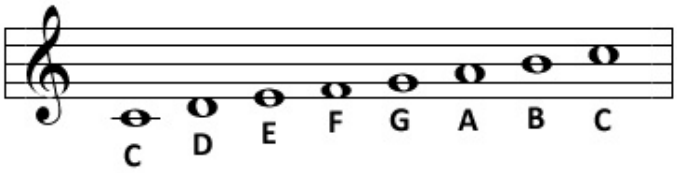
U manjoj mjeri, ali još uvjek važnoj za napomenuti, visina tona je ovisna i o razini zvučnog tlaka. Za niske frekvencije vrijedi da što je ton glasniji (veća razina zvučnog tlaka), to će ton biti percepiran nižim, dok za visoke frekvencije vrijedi obrnuto. Na primjer, ako imamo ton od 200 Hz koji je odsviran jako glasno, naš će ga mozak percepirati za otprilike poluton nižim.

Neki glazbeni instrumenti proizvode zvukove kojima je gotovo nemoguće ili jako teško odrediti osnovnu frekvenciju titranja. To su na primjer bubnjevi i udaraljke koji proizvode zvukove sa izrazito neharmoničkim spektrom. Neharmonički spektri su, za razliku od harmoničkih, sastavljeni od frekvencijskih komponenata koje nisu direktni višekratnici osnovnog harmonika. Zbog toga je teško reći koju visinu tona ima, na primjer, bas bubanj. No, međutim, još uvijek možemo prepoznati da li bubanj zvuči duboko ili visoko. (Wikipedia, 2011b)

2.1.2. Ljestvice i intervali

Da bi znali na koju visinu tona se referenciramo, moramo na neki način označiti pojedine note od kojih svaka predstavlja određenu osnovnu frekvenciju. Prema konvenciji, note se označavaju sa prvih sedam velikih slova abecede (A B C D E F G) i obično se grupiraju u ljestvice, od nižih nota prema višim. Ako želimo koristiti notu koja se nalazi negdje između ovih navedenih, koristimo se sa povisilicama (#) ili snizilicama (b) koje povisuju/snizuju ton za pola stupnja. Međutim, pri korištenju povisilica i snizilica treba biti svjestan jedne činjenice, a to je da ako npr. povisimo notu D za pola stupnja, dobiti ćemo notu D# (čitaj: ”Dis”), no ako notu E snizimo za pola stupnja dobivamo notu Eb (čitaj: ”Es”) koja ima istu visinu tona kao i nota D#, tako da zapravo postaje svejedno koji ćemo naziv iskoristiti. Razmaci između pojedinih nota se u glazbenoj teoriji zovu intervali, a najmanji interval koji se koristi je jedan poluton.

Ljestvice se obično grupiraju u klase, ovisno o intervalima između njenih nota. Najjednostavnija ljestvica, koja se koristi kako u osnovne glazbeno-obrazovne svrhe tako i u praksi, je C–dur ljestvica (Slika 2.1).



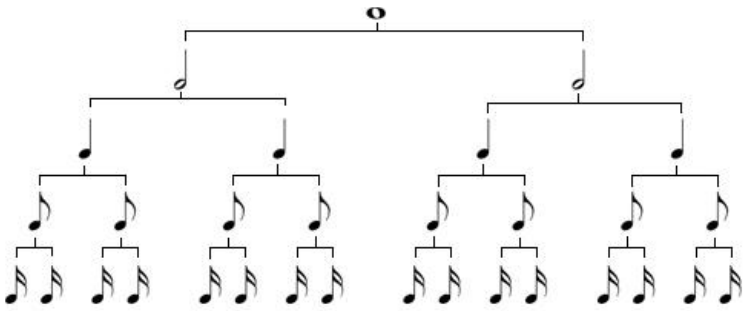
Slika 2.1: C – dur ljestvica

Vidimo da C–dur ljestvica ima osam nota, počevši od note C, s time da je osma nota ponovno nota C ali ovaj puta u višoj oktavi od one na početku niza. To je zbog repetitivnog karaktera nota koje se ponavljaju u nizu od 7 elemenata ali svaki novi niz predstavlja novu oktavu. Ako želimo specificirati da se neka nota nalazi u određenoj oktavi, staviti ćemo broj oktave pored same note (npr. C1, C2, C3,...itd.). No što znači da se nota nalazi u nekoj oktavi? Radi se o intervalima. Tako je na primjer, nota C2 (65.4 Hz) za 7 elemenata ljestvice udaljena od note C1 (32.7 Hz). Primjetimo da je nota C2 dvostruko veće frekvencije od note C1, a to je zbog logaritamskog omjera između susjednih nota, no o tome ćemo detaljnije nešto kasnije, u poglavlju 2.2.

2.1.3. Trajanje note, tempo i ritam

Kako bi glazba imala smisla i kako bi bila ugodna za slušanje, nije dovoljno samo proizvoljno rasporediti note, već im je potrebno odrediti trajanje i ritam. U standardnim notnim zapisima nalaze se informacije o notama koje je potrebno svirati, kao i o njihovom ritmu. Bez tih informacija bilo bi jako teško uskladiti instrumente koji sviraju različite dionice i glazba ne bi zvučala melodiozno već kaotično.

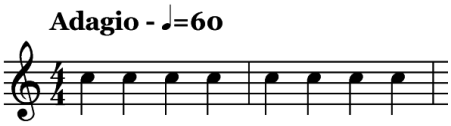
Na Slici 2.2 su prikazane standardne oznake za različita trajanja nota i njihov međusobni hijerarhijski odnos.



Slika 2.2: Oznake za različita trajanja nota

U prvom redu se nalazi oznaka za cijelu notu, u drugom redu se nalaze oznake za polovinke, u trećem redu su četvrtinke, četvrtom osminke, te se u zadnjem redu nalaze šesnaestinke. Možemo i dalje skraćivati note ali je ovo za sada dovoljno, budući da se radi samo o ilustrativnom primjeru. Primjetimo da se svaka nota, određenog trajanja, može podijeliti na dvije note kraćeg trajanja, npr. polovinku možemo zapisati i kao dvije četvrtinke, ili pak četiri osminke...itd.

Kako bi ove navedene oznake imale smisla, mora se na neki način odrediti trajanje neke referentne note prema kojoj onda sve ostale poprimaju svoje vrijednosti. Standardni način na koji se to radi je da se na vrhu notnog zapisa skladbe, prije nego li se počnu navoditi note, označi trajanje jedne četvrtinke u otkucajima po minuti (*engl. Beats Per Minute – BPM*), odnosno tempo cijele pjesme.



Slika 2.3: Oznaka za tempo

Prema Slici 2.3, trajanje jedne četvrtinke je 60 BPM-a (1 otkucaj po sekundi), što znači da će trajanje svih osminki u skladbi biti 120 BPM-a, svih polovinki 30 BPM-a, dok će trajanje cijelih nota biti 15 BPM-a.

Uz oznaku za tempo, potrebno je navesti i ritam koji se obično navodi pokraj ključa notnog crtovlja. Ritam ne mora biti konstantan kroz cijelu pjesmu, može se mijenjati po želji, ali je svaku promjenu potrebno naglasiti. Na Slici 2.3 uočavamo jednu takvu oznaku, koja u ovom slučaju označava tzv. četvero-četvrtinsku mjeru za ritam (**4/4**). To znači da će se u svakom taktu skladbe naći note (i pauze) ekvivalentnog trajanja od 4 četvrtinke. Ovo je najčešće korišteni ritam u popularnoj glazbi, ostale često korištene mjere su tro-četvrtinska (**3/4**), dvo-četvrtinska (**2/4**), dvo-polovinska (**2/2**) te šestero-osminska (**6/8**).

Instrument koji obično drži ritam i zadužen je da ostali instrumenti budu vremenski usklađeni – su bubnjevi. Kao što je već prije napomenuto, bubnjevi imaju inharmonički spektar signala, pa se zbog tog svojstva mogu koristiti uz sve instrumente bez straha da će zvučati atonalno, tj. izvan tonaliteta skladbe.

2.1.4. Melodija i harmonija

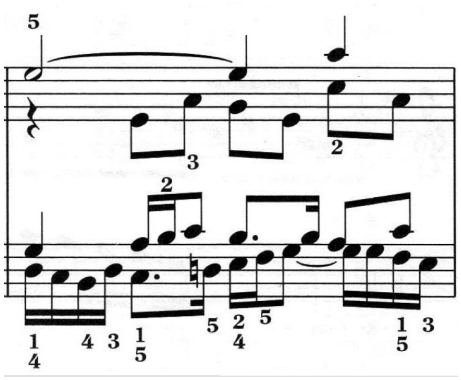
Pod pojmom melodije se podrazumijeva niz nota, određenog trajanja i ritma. Melodija se može sastojati od pojedinačnih nota, no one mogu biti kombinirane i u parove, akorde, može biti repetitivna – ograničenja praktički nema.

No, da bi u glazbenom djelu mogli reći da je nešto melodiozno, moraju se poštovati određena pravila koja je potrebno jako dobro poznavati prije nego ih se počne kršiti. Nećemo ulaziti u dubinu tih pravila, ali ćemo napomenuti jedan važan pojam koji je bitan pri stvaranju glazbe, pogotovo u realnom vremenu, a to je *harmonija*.

Glazbeniku koji svira određeni instrument, moraju u svakom trenutku biti dostupne sve note koje je moguće proizvesti, on sam bira koje note će odsvirati, a koje neće. Mora, također, biti i u mogućnosti istovremeno iskombinirati note u cjeline koje se zovu *akordi*. Akordi su cjeline sačinjene od dvije, tri ili više nota sviranih istovremeno koje su u određenim harmoničkim ili, u nekim slučajevima, inharmoničkim odnosima.

Većini ljudi pojam harmonije je intuitivno vezan uz pjevanje, gdje dva ili više pjevača pjevaju različite note koje zajedno zvuče melodiozno i ugodno za slušanje, tj. harmonično. Da li će note zvučati harmonično ili inharmonično određuje njihov međusobni interval u kojem se nalaze, na glazbeniku/skladatelju je da odabere te intervale i odluči kako će ih upotrijebiti. Dakle, možemo zaključiti da je jako bitno omogućiti glazbeniku/skladatelju potpunu kontrolu nad instrumentom, tako da ga samo njegova vlastita mašta ograničava.

Na slijedećoj slici prikazan je isječak iz skladbe J.S. Bacha ”Fugue br. 17, Ab”, na kojoj možemo vidjeti primjer korištenja melodije i harmonije.

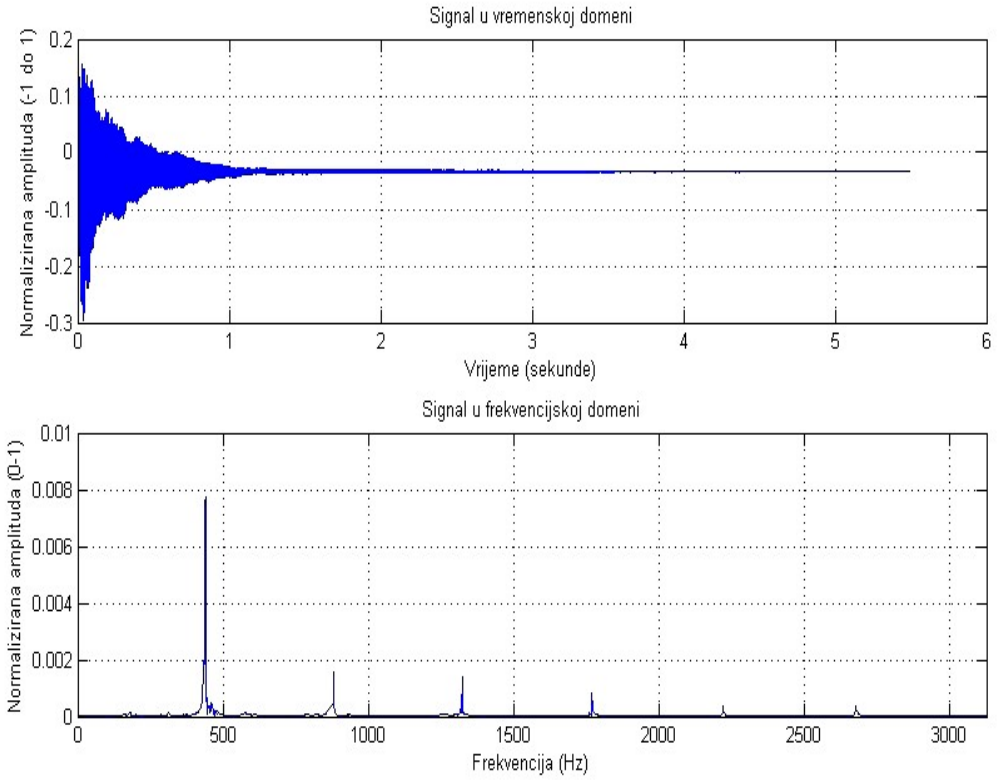


Slika 2.4: Isječak iz skladbe J.S.Bacha : ”Fugue br. 17, Ab”

2.1.5. Boja tona

Ono što razlikuje instrumente jedne od drugih je njihova boja tona. Svaki instrument ima svoj specifičan zvuk, koji je direktna poslijedica načina izrade instrumenta i materijala od kojih je sačinjen. Pojam *boja tona* proizlazi iz činjenice da svaki zvuk u prirodi sadrži u sebi različite frekvencije, odnosno harmonike od kojih je sačinjen. Ti harmonici različitim težinama pridonose ukupnom doživljaju zvuka, baš kao što u slikarstvu kombinacijom različitih boja dobivamo nove boje različitih nijansi.

Fizikalne karakteristike zvuka koje utječu na našu percepciju boje tona jesu spektar signala koji instrument proizvodi i njegova ovojnica. Na slijedećoj slici je prikazan signal note A4 (440 Hz) odsvirane na klaviru u vremenskoj i frekvencijskoj domeni.



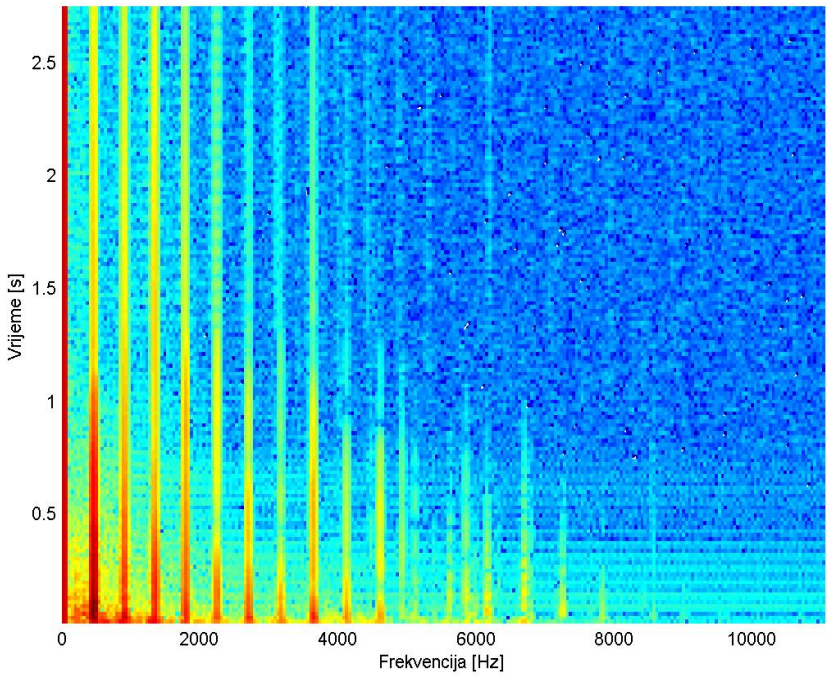
Slika 2.5: Nota A4 klavira u vremenskoj i frekvencijskoj domeni

Slika 2.5 prikazuje promjenu signala s vremenom (gornja slika) i frekvencijom (donja slika) u odnosu na amplitudu signala. Možemo primjetiti da oblik signala ima specifičan izgled, te da se sastoji od niza harmoničnih frekvencija – harmonika. Ti harmonici su glavni razlog zašto nota klavira zvuči na sebi svojstven način, i zbog čega note odsvirane na klaviru možemo odmah prepoznati. U pravilu, harmonici su višekratnici osnovne frekvencije titranja, odnosno prvog harmonika u spektru signala,

koji na slici 2.5 iznosi 440 Hz. No, međutim, male razlike mogu postojati. Tako je, na primjer, za klavir karakteristično da viši harmonici signala nisu čisti višekratnici osnovne frekvencije već su sniženi/povišeni za par Hertz-a.

U vremenskoj domeni, signal prati određenu ovojnicu, odnosno mijenja svoju amplitudu sa vremenom. Ta ovojnica dodaje dinamičnost u glazbenu notu te utječe na ukupnu percepciju zvuka. Zamislimo samo koliko bi monotono i jednolično zvučala nota da kroz cijelo svoje trajanje ima konstantnu amplitudu.

Nešto što je važno napomenuti, a nije vidljivo sa Slike 2.5, je to da se spektar signala glazbenih instrumenata mijenja sa vremenom. To se najbolje može vidjeti ako nacrtamo spektrogram signala, koji prikazuje frekvencijsku ovisnost signala o vremenu (Slika 2.6).



Slika 2.6: Spektrogram note A4 klavira

Sa slike je vidljivo da spektar ima najviše harmonika na početku signala, te da se njihov broj i intenzitet sa vremenom smanjuje. Do ovoga efekta dolazi zbog karakterističnog načina titranja klavirske žice. Naime, kada žica tek počne titrati, ona titra sa velikim brojem modova titranja, do kojih dolazi zbog toga što je žica učvršćena sa oba kraja pa se formiraju stojni valovi na njoj.

Ti modovi titranja s vremenom iščezavaju, a to se u spektru signala očituje kao pad broja harmonika.

Kod sinteze instrumenata, ovojnica i spektar signala će nam poslužiti kao sredstvo kojim ćemo oblikovati zvuk. Promjenom samo ta dva elementa signala možemo dobiti jako velik raspon zvukova, nekih čak koje nikada ne bismo mogli niti zamisliti.

Da bi modeliranjem pokrili cijeli raspon nota koje instrument može proizvesti, moramo najprije pokazati matematičke odnose pojedinih intervala nota, te na koji način se generiraju harmonici signala. Ovu temu obrađujemo u slijedećem odjeljku.

2.2. Matematički odnosi intervala među notama

Poznavanje glazbenih instrumenata, kao i osnove glazbene teorije, nužno je za razumijevanje postupaka modeliranja i sinteze istih. No jednako tako, potrebno je poznavati i matematičke odnose među notama, u cijelom rasponu instrumenta koji se modelira.

Već je ranije napomenuto da se note, osim slovima, označavaju i brojevima koji određuju oktavu u kojoj se nota nalazi. Na taj način, nota je potpuno određena sa samo dva simbola (tri u slučaju da nota ima povisilicu **#** ili snizilicu **b**). Svaka tako određena nota ima svoju pripadnu osnovnu frekvenciju titranja, što je čini jedinstvenom i prepoznatljivom ljudskome uhu.

Najmanji interval između nota koji možemo dobiti je jedan poluton, no taj interval, gledano iz frekvencijske perspektive, nije konstantan. Note i njihove pripadne frekvencije slijede geometrijski niz. Svaki element iz tog niza možemo dobiti ako znamo neku referentnu frekvenciju i broj polutonova koji definira razmak od željene i referentne note. Izraz je dan u slijedećoj jednadžbi (Grondin, 2011):

$$f_{trenutna} = 2^{s/12} \times f_{referentna} \tag{2.1}$$

Kao referentna nota se često koristi A4 nota, koja ima osnovu frekvenciju od 440 Hz. Ako želimo saznati koju frekvenciju ima npr. nota D5, koja se nalazi u slijedećoj oktavi i odmaknuta je od referentne note za s = 5 polutonova, dobit ćemo slijedeće:

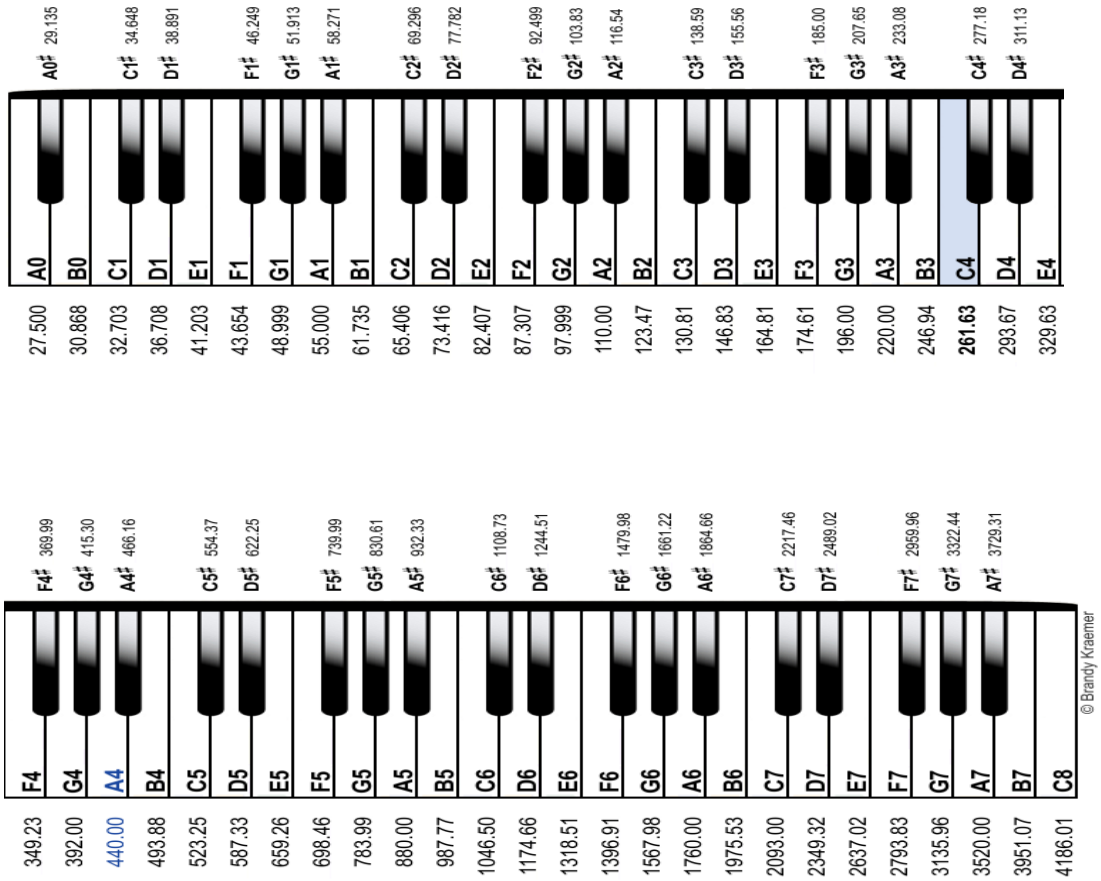
$$f_{trenutna} = 2^{5/12} \times 440,$$

$$f_{trenutna} = 1.33484 \times 440,$$

$$f_{trenutna} = 587.33\ Hz.$$

Želimo li pak izračunati frekvenciju note koja se nalazi niže od referentne note, kao npr. nota E4, koristit ćemo negativan predznak za broj polutonova (s = –5).

Frekvencija note D5 je dakle 587.33 Hz. Sličnim postupkom se mogu dobiti i ostale note. Na slijedećoj slici, prikazan je cijeli raspon nota klavira i njihovih pripadnih frekvencija (Kraemer, 2011).



Slika 2.7: Note klavira i njihove pripadne frekvencije u Hertz-ima

Dakle, svaka nota na tipkovnici klavira odgovara specifičnoj frekvenciji. Na vratu gitare, te note se dobivaju pritiskom žice na određenom pragu vrata, pa se dobivanje nota svodi na skraćivanje žice koja titra. Što je žica kraća, to će proizvesti višu frekvenciju.

$$f \propto \frac{1}{l} \tag{2.2}$$

Jednadžba 2.2 pokazuje da je frekvencija obrnuto proporcionalna duljini žice koja titra. No, frekvencija titranja može ovisiti još o dva parametra, a to su napetost žice te njena gustoća (Wikipedia, 2011b).

$$f \propto \sqrt{T} \tag{2.3}$$

$$f \propto \frac{1}{\sqrt{\rho}} \tag{2.4}$$

Iz jednadžbe 2.3 je vidljivo da je frekvencija proporcionalna drugom korijenu napetosti žice. Manje napeta žica će proizvoditi niže note, nego kada je jače zategnemo. Nasuprot tome, iz izraza 2.4 možemo zaključiti da što je žica deblja (veća gustoća ρ) to će niže frekvencije proizvoditi.

Sada kada znamo osnovne elemente glazbenih nota, te kako se dobivaju različite visine tonova, možemo krenuti u problematiku modeliranja različitih instrumenata. U okviru ovoga rada nije cilj modelirati sve klase instrumenata, već se naglasak stavlja na postupke modeliranja koji se mogu naknadno prilagoditi za veliki broj instrumenata.

3. Modeliranje glazbenih instrumenata

Konstrukcija instrumenata nije lak zadatak. Provela su se stoljeća istraživanja i eksperimentiranja s ciljem kako bi se dobili zanimljivi zvukovi i instrumenti koje je sa relativnom lakoćom moguće kontrolirati. Od klavira i gitare pa sve do bubnjeva i udaraljki, svaki instrument ima specifičan karakter po kojem je prepoznatljiv i koji pridonosi na jedinstven način krajnjem proizvodu – glazbi.

Dok slušamo glazbu, čujemo skup različitih zvukova istovremeno koji su aranžirani prema željama i potrebama skladatelja kako bi se postigao određeni glazbeni ugođaj. Ljudsko uho je sposobno dekodirati svaki zasebni izvor zvuka i prepoznati njegov specifičan karakter, neovisno od ostalih izvora. Svaku notu je moguće pripisati odgovarajućem instrumentu upravo zbog boje zvuka koji dolazi do našeg uha. Baš kao što i svaki čovjek ima drugačiju boju glasa po kojoj je prepoznatljiv, tako i sve što čujemo ima svoju zvučnu boju, koju je moguće dobiti i umjetnim putem ako poznajemo teoriju nastajanja zvuka i način na koji ga percepiramo.

U okviru ovoga rada, obraditi će se tri postupka sinteze instrumenata. Naravno, postoji puno više metoda koje ovdje nećemo obrađivati ali sve one vuku svoje korijene u jednoj od tih metoda ili su kombinacija istih. To su redom: Sinteza frekvencijskom modulacijom (FM sinteza), Sinteza valnim tablicama (Wavetable sinteza), te Fizikalno modeliranje.

3.1. FM sinteza

Jedna od najpopularnijih metoda modeliranja zvuka je sinteza frekvencijskom modulacijom koju je 1973. godine na kalifornijskom sveučilištu Stanford predstavio John M. Chowning (Chowning, 1973).

U tadašnje vrijeme, sintetički signali su zvučali dosta neprirodno te su algoritmi za sintezu bili računski zahtjevni zbog činjenice da tada nije bilo brzih računala sa velikim kapacitetima memorije. Sa ovim ograničenjima bilo je sasvim logično da se krene sa razvojem jednostavnijih algoritama, kojima će ujedno i kvaliteta sinteze dati prirodnije

rezultate. Jedna činjenica, koja je u to vrijeme bila uočena, jest da sintetički signali zvuče neprirodno zbog toga što nije uzeta u obzir njihova spektralna promijenjivost sa vremenom. Kao okosnicu svojeg istraživanja, Chowning je pokušao na neki način tu činjenicu, iskoristiti i implementirati na računalu.

Glavna prekretnica se dogodila kada je svojim sintetičkim signalima mijenjao vibrato efekt. Vibrato je često korištena tehnika pri sviranju instrumenata i pjevanju, kojom se glazbenoj noti u malim intervalima mijenja frekvencija titranja oko osnovne frekvencije, tako da nota dobije na dinamičnosti. Chowning je postupno povećavao brzinu vibrato efekta, sve dok nije došao do toga da se signalu promijenila boja tona.

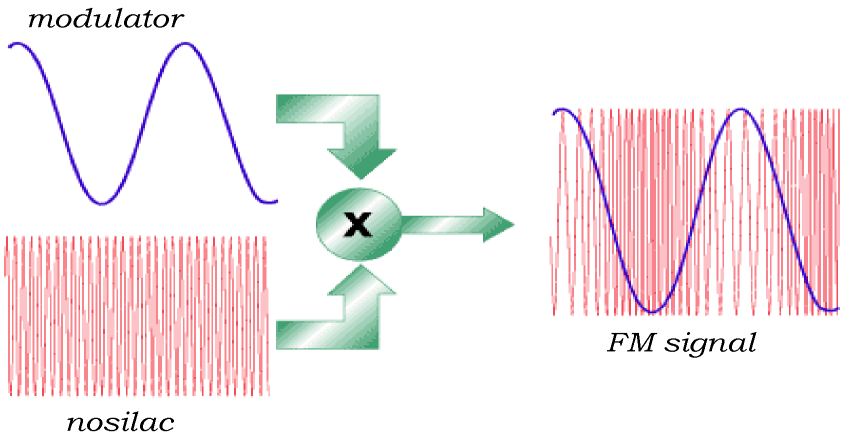
Matematičkom analizom se pokazalo da je učinak na spektar signala isti kao kod FM prijenosa radio–valova, metode do tada korištene jedino u radio–komunikacijama. Dobro poznate jednadžbe FM radio prijenosa dobile su jedan potpuno novi smisao, koji je još i danas jako često korišten u komercijalnim sintesajzerima.

3.1.1. Modulacija frekvencije

U svojoj suštini, metoda frekvencijske modulacije sastoji se od dva signala, nosioca i modulatora. To su dva oscilatora od kojih jedan (modulator) mijenja frekvenciju osciliranja drugome oscilatoru (nosiocu). Proces opisuje slijedeća jednadžba, koja je polazište i osnova FM sinteze (Park, 2010):

$$y(t) = A_{nosilac} \cdot \sin\left(2 \cdot \pi \cdot f_{nosac} \cdot t + A_{modulator} \cdot \sin(2 \cdot \pi \cdot f_{modulator} \cdot t)\right) \quad (3.1)$$

Ilustrativno, ova jednadžba bi izgledala ovako nekako (Irvine, 2011):



Slika 3.1: Frekvencijska modulacija

Jednadžba 3.1 pokazuje kako modulacijski signal utječe na signal nosilac. Izraz možemo i jednostavnije zapisati ako odvojimo dva signala:

$$y(t) = A_{nosilac} \cdot \sin\big(2 \cdot \pi \cdot f_{nosilac} \cdot t + g(t)\big), \tag{3.2}$$

$$g(t) = A_{modulator} \cdot \sin(2 \cdot \pi \cdot f_{modulator} \cdot t) \tag{3.3}$$

Ako dobro promotrimo jednadžbe 3.1 – 3.3, uočiti ćemo da ti izrazi ne predstavljaju direktno modulaciju frekvencije već modulaciju faze. No ove dvije definicije su zapravo ekvivalentne kada se u jednadžbama koristi signal modulator sa samo jednom modulacijskom frekvencijom. Prema strogoj definiciji frekvencijske modulacije, jednadžba za modulator 3.3 će postati (Schottstaedt, 1985):

$$g(t) = A_{modulator} \cdot \int_0^t \sin(2 \cdot \pi \cdot f_{modulator} \cdot t)\,\mathrm{d}t \tag{3.4}$$

Bez obzira na ovu razliku između frekvencijske i fazne modulacije, u daljnjem tekstu ćemo govoriti o frekvencijskoj modulaciji, jer će za naše primjere ova dva pojma biti ekvivalentna.

3.1.2. Spektar FM signala i Besselove funkcije

Ono što FM sintezu čini jedinstvenom, a ponekad i nezamjenjivom, je svakako njen spektar signala koji proizvodi. Najveći uspjeh je doživjela '80-ih godina kada je tvrtka Yamaha predstavila svoj, još i danas korišten, *Yamaha DX 7* sintesajzer (Slika 3.2).

Slika 3.2: Yamaha DX7 sintesajzer

Raspon signala koji je moguće dobiti FM sintezom je zaista impresivan, od stvarnih instrumenata do nekih potpuno novih i jedinstvenih koji su danas postali standard u svim komercijalnim sintesajzerima. Sve te zvukove je moguće dobiti na vrlo elegantan, jednostavan i intuitivan način, uz kontrolu samo nekoliko parametara. Jedan od glavnih parametara koji kontroliraju spektar signala, i koji utječe na boju tona signala je takozvani *indeks modulacije* β.

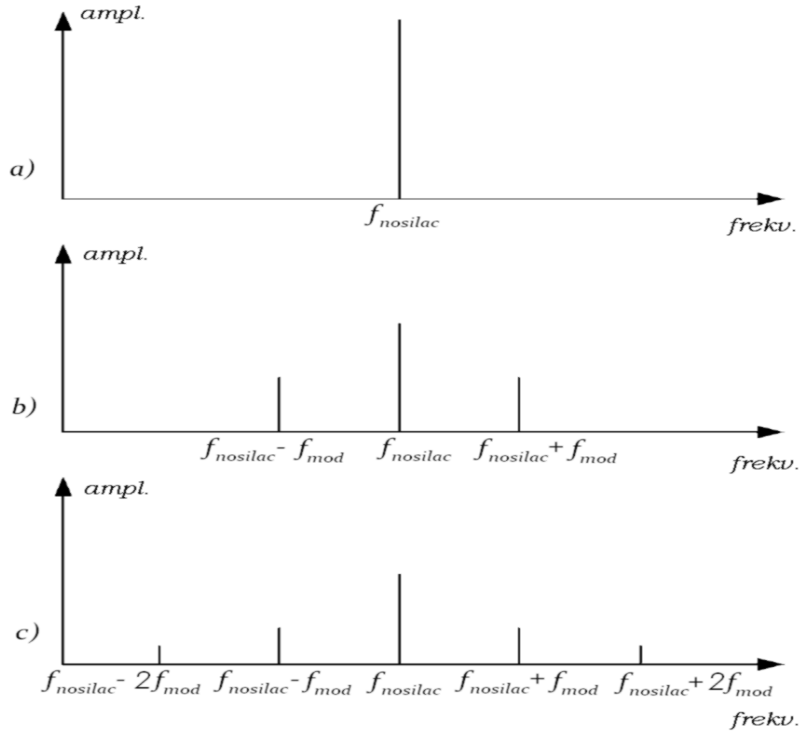
Ako jednadžbu 3.1 zapišemo malo drugačije, sa istaknutim indeksom modulacije, dobiti ćemo slijedeću jednadžbu (Park, 2010):

$$y(t) = A_{nosilac} \cdot \sin\left(\omega_{nosilac} \cdot t + \beta \cdot \sin(\omega_{modulator} \cdot t)\right), \tag{3.5}$$

$$Indeks\, modulacije: \qquad \beta = \frac{A_{mod}}{f_{mod}} \tag{3.6}$$

Iz jednadžbe 3.5 možemo vidjeti da ako je β = 0, modulacije neće biti te ćemo u spektru signala imati samo frekvenciju nosioca.

Ako je $\beta \neq 0$, u spektru signala ćemo oko centralne frekvencije nosioca dobiti harmonike koji su ovisni o modulacijskoj frekvenciji.



Slika 3.3: Spektar FM signala oko centralne frekvencije nosioca

Na slici 3.3 vidimo tri različita slučaja. U slučaju a) nema modulacije ($\beta = 0$) i u spektru signala nisu prisutni harmonici. Za slučaj b) postoji modulacija, ali je relativno mala pa su stvorena samo dva harmonika. Konačno, u slučaju c) vidimo da je stvoreno više harmonika koji su jednoliko odmaknuti od centralne frekvencije signala nosioca.

Općenito, za k–ti harmonik možemo zapisati (Park, 2010):

$$f_{harmonika} = f_{nosilac} \pm k \cdot f_{mod}, \qquad k = 1, 2, 3, ... \tag{3.7}$$

Koliko će harmonika biti prisutno u spektru signala ovisi o indeksu modulacije β. Amplitude tih harmonika određujemo pomoću tzv. Besselovih funkcija, koje ovise o indeksu modulacije.

Jednadžba 3.5 se često zapisuje pomoću Besselovih funkcija (Chowning, 1973):

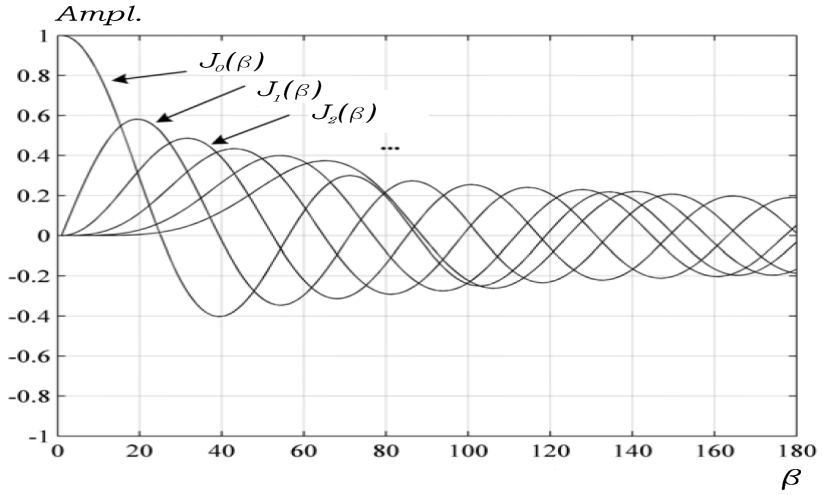
$$y(t) = A_{nosilac} \cdot \sum_{p=-\infty}^{p=\infty} J_p(\beta) \cdot \sin\big((\omega_{nosilac} + p \cdot \omega_{mod})\big) \cdot t, \tag{3.8}$$

$$J_p(\beta) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot (\frac{\beta}{2})^{p+2k}}{k!(p+k)!}, \qquad p = red\,harmonika \tag{3.9}$$

Ako sada jednadžbu 3.8 raspišemo pomoću trigonometrijskih izraza, te uzmemo u obzir samo pozitivne vrijednosti od p ($p \geq 0$), pošto za negativne vrijednosti p imamo identične Besselove funkcije, dobivamo (Chowning, 1973):

$$\begin{aligned} y(t) = A_{nosilac} \cdot \{ & J_0(\beta) \cdot \sin(\omega_{nosilac} t) + \\ & + J_1(\beta) \cdot \big(\sin(\omega_{nosilac} + 1\omega_{mod})t - \sin(\omega_{nosilac} - 1\omega_{mod})t \big) + \\ & + J_2(\beta) \cdot \big(\sin(\omega_{nosilac} + 2\omega_{mod})t + \sin(\omega_{nosilac} - 2\omega_{mod})t \big) + \\ & + J_3(\beta) \cdot \big(\sin(\omega_{nosilac} + 3\omega_{mod})t - \sin(\omega_{nosilac} - 3\omega_{mod})t \big) + \\ & + \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \}. \end{aligned} \tag{3.10}$$

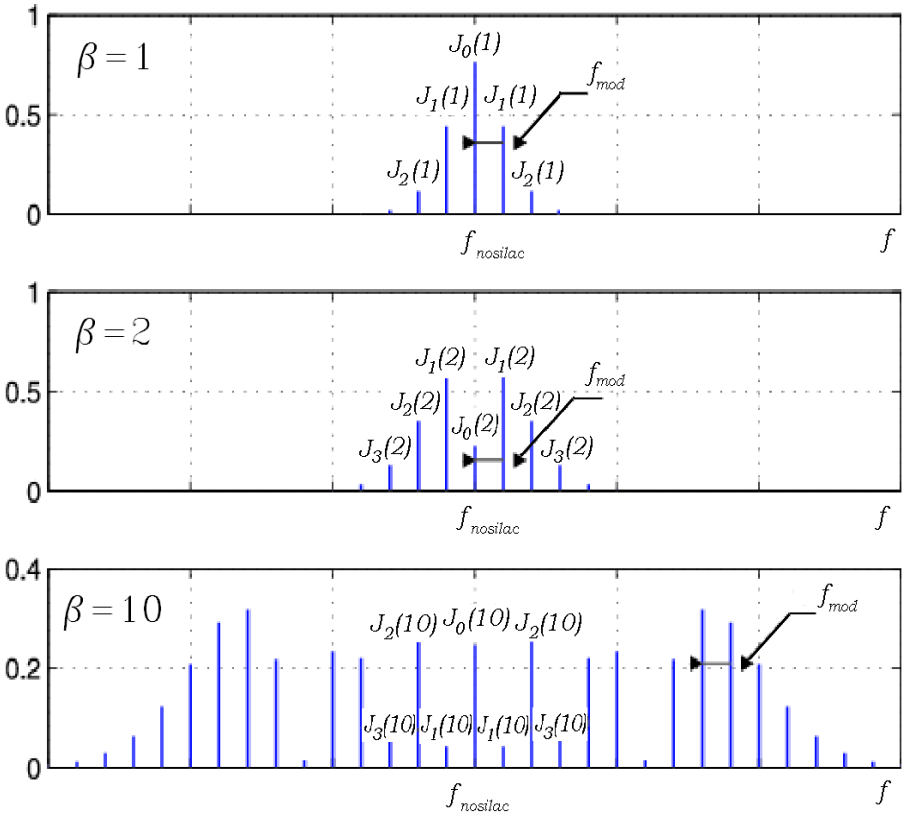
Na slici 3.4 prikazane su Besselove funkcije različitih harmonika, ovisnih o indeksu modulacije β. Slika nam pokazuje da, što je veći indeks modulacije to će amplituda harmonika biti manja, a može biti čak i negativna.



Slika 3.4: Besselove funkcije

Porastom indeksa β, energija signala nosioca preraspodijeljuje se na harmonike po zakonu Besselovih funkcija. Lako je za primjetiti da se spektar signala može kontrolirati samo sa dva parametra: indeksom modulacije β i frekvencijom modulacije f_m.

U frekvencijskoj domeni, promjena indeksa modulacije β i utjecaj Besselovih funkcija, prikazani su na slici 3.5 (FESB, 2008):

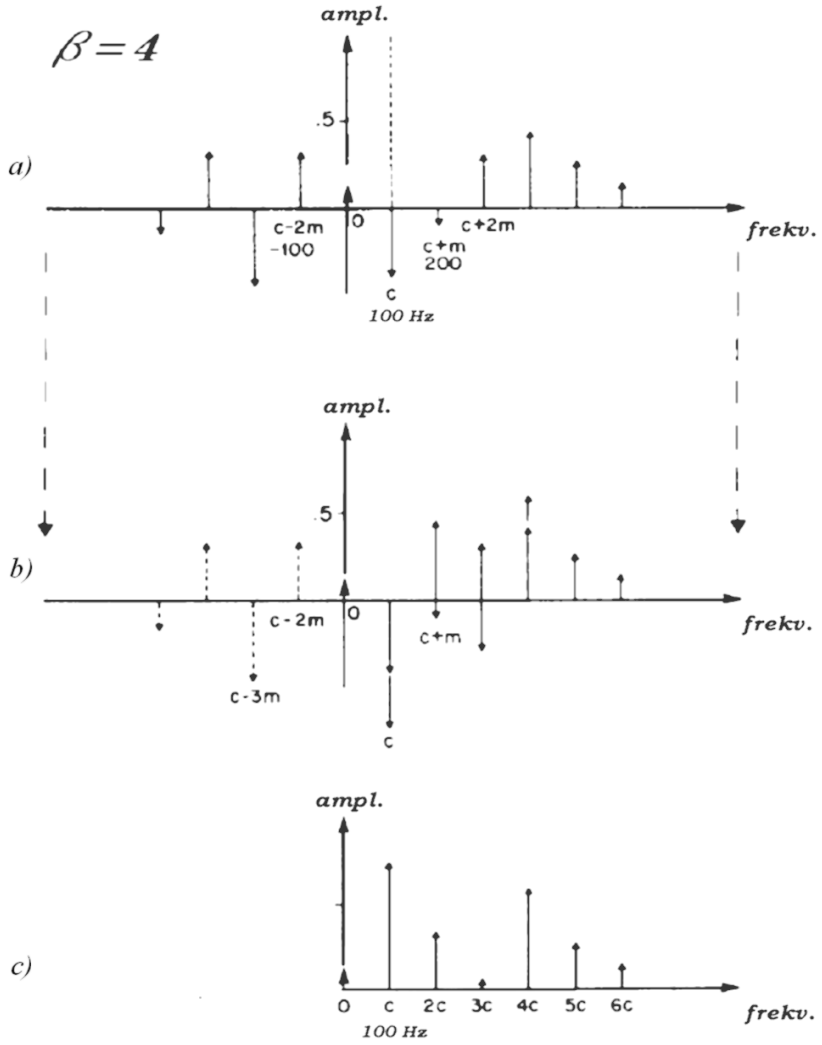


Slika 3.5: Amplitudni spektri FM signala za različite indekse modulacije β

Ovakvo ponašanje spektra FM signala je jako poželjno pri modeliranju glazbenih instrumenata, jer se u njihovim spektrima također može zapaziti opadanje energije kod viših harmonika. Općenito, za glazbene instrumente vrijedi da niži harmonici uvijek imaju veću energiju od viših harmonika.

3.1.3. Omjer frekvencija *C:M*

Posebna kvaliteta FM sinteze krije se u činjenici da postoje omjeri frekvencija nosioca i modulatora *C:M* koji će proizvesti harmonike koji će pasti u negativno područje frekvencija spektra FM signala (Chowning, 1973). Te negativne frekvencije se reflektiraju oko glavne osi od 0 Hz u pozitivnu domenu i ”miješaju” se sa ostalim komponentama spektra (Slika 3.6).



Slika 3.6: Spektar FM signala sa frekvencijama u negativnoj domeni i prebacivanje u pozitivnu domenu

Pri prebacivanju komponenata u pozitivnu domenu potrebno je fazno zakrenuti komponente za $\varphi = 180°$ (Slika 3.6 b)). Tako prebačene komponente se zatim zbrajaju sa odgovarajućim komponentama već prisutnima na pozitivnoj strani, te je konačni spektar signala koji čujemo prikazan slikom 3.6 c)

Zbog ovog efekta postoji praktički neograničen broj harmoničkih i neharmoničkih spektara koji se mogu proizvesti. Ako npr. želimo proizvesti signal određene osnovne frekvencije titranja sa specifičnom bojom tona, postavit ćemo frekvenciju nosioca na tu osnovnu frekvenciju, dok ćemo frekvenciju modulatora podesiti u nekom omjeru u odnosu na nosioc.

No, međutim, to baš i nije tako jednostavno kako se čini na prvi pogled. Osnovna frekvencija titranja generalno odgovara najnižem harmoniku u spektru signala, tako

da bi se svi proizvedeni harmonici trebali nalaziti na višim frekvencijama od osnovne. Ako nepažljivo odaberemo omjer frekvencija *C:M*, moglo bi se dogoditi da nam spektar signala više neće imati osnovnu frekvenciju koju smo htjeli. To se obično dešava za omjere *C:M* kod kojih je frekvencija modulatora manja od dvostruke frekvencije nosioca. Odnosno, vrijediti će: $M \leq 2C$.

Dakle, da bi osnovna frekvencija bila najniža frekvencija u spektru signala, za omjer *C:M* mora vrijediti $M \geq 2C$ ili $C : M = 1 : 1$.

Pri modeliranju instrumenata, najčešći će nam cilj biti proizvesti harmonički spektar signala, što se postiže ako odaberemo *C:M* omjere iz familije gdje je uvijek *C=1*, kao npr. *C:M= 1:1, 1:2, 1:3, 1:4, 1:5, 1:6, 1:7, 1:8, ...* itd. Sve ostale kombinacjie omjera *C:M* će rezultirati neharmoničkim spektrom, što je na primjer korisno pri modeliranju bubnjeva koji imaju izrazito neharmonički spektar signala (Truax, 1977).

3.1.4. ADSR ovojnica signala

Pri konstrukciji umjetnih zvukova nije dovoljno modelirati samo spektar signala, potrebno je modelirati i njegov vremenski promjenjiv karakter. Vidjeli smo ranije u poglavlju 2.1.5 da signal note klavira ima specifičan vremenski oblik. Takav oblik signala opisuje njegova *amplitudna ovojnica*.

Ovojnica određuje dinamiku signala, i velikim je dijelom odgovorna za to kako će naše uho i mozak interpretirati signal. Bitno je naglasiti da pri procesu sinteze, dizajner zvuka može proizvoljno odabrati oblik ovojnice koji će oblikovati njegov signal. Na taj način se mogu dobiti vrlo zanimljivi zvukovi, ograničenja praktički nema osim naše mašte. Međutim, ako se modeliraju realni instrumenti, ovojnice je potrebno konstruirati prema karakterističnom ponašanju dotičnog instrumenta.

Ovojnice se obično konstruiraju u nekoliko faza. Najčešće korištena ovojnica je ***ADSR ovojnica*** koja opisuje četiri faze signala:

1. **A** – *engl.* Attack – vrijeme porasta signala,
2. **D** – *engl.* Decay – vrijeme kratkotrajnog opadanja signala,
3. **S** – *engl.* Sustain – vrijeme zadržavanja signala,
4. **R** – *engl.* Release – vrijeme otpuštanja signala.

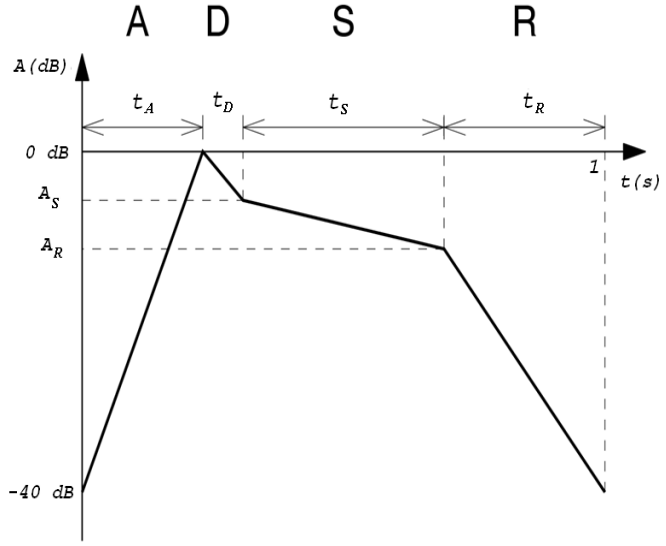
Osim amplitudne ovojnice, u FM sintezi instrumenata koristi se još i *modulacijska ovojnica*. Ovakav tip ovojnice služi za modeliranje spektralne promjenjivosti signala

s vremenom. Kod svih realnih instrumenata, zvuk je bogatiji harmonicima u početnoj fazi razvoja, tj. za vrijeme *attack* faze, te se postepeno kroz ostale faze osiromašuje harmonicima. Postupkom FM sinteze, ovaj učinak se postiže ako primjenimo ovojnicu na modulacijski signal, odnosno tako da odredimo po kojem zakonu će se mijenjati indeks modulacije β.

Upotrebom amplitudne i modulacijske ovojnice postižu se prirodniji zvukovi instrumenata, ali je moguće dobiti i neke potpuno nove i zanimljive zvukove.

Ovojnica limenih puhačkih instrumenata

Na slici 3.7 prikazana je ovojnica koja se koristi pri sintezi limenih puhačkih instrumenata (Chowning, 1973).



Slika 3.7: ADSR ovojnica za model limenih puhačkih instrumenata

Za sintezu limenog puhačkog instrumenta, poput npr. trombona možemo odabrati *C:M* parametre u omjeru *1:1*. Tako ćemo npr. za sintezu note $A4 = 440Hz$ imati frekvencije nosioca i modulatora $f_{nosilac} = f_{modulator} = 440Hz$.

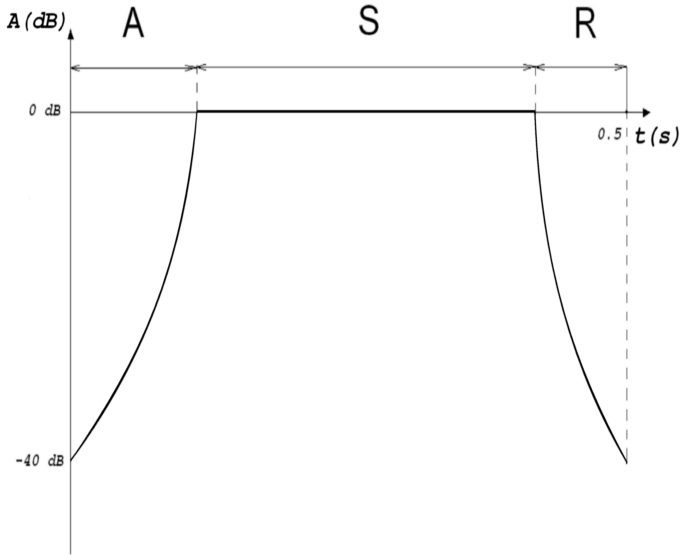
Indeks modulacije β u najjednostavnijem modelu možemo postaviti da se mijenja proporcionalno sa amplitudnom ovojnicom od minimalne do maksimalne vrijednosti npr. $\beta = 0 - 5$.

Zanimljivo je napomenuti, da ako se recimo želi dodati *tremolo efekt* u signal, tj. nisko-frekventna amplitudna modulacija signala, modulacijsku frekvenciju se može uvećati za mali iznos u odnosu na frekvenciju nosioca (npr. $f_{modulator} = 440.5Hz$).

Kao posljedica ove promjene, harmonici koji padaju u negativnu domenu spektra signala, pri preslikavanju u pozitivnu domenu neće točno pasti na svoje pozitivne ekvivalente, već će stvoriti vrlo bliske susjedne harmonike koji će stvoriti čujno i lagano amplitudno titranje izlaznog signala.

Ovojnica drvenih puhačkih instrumenata

Slika 3.8 prikazuje primjer amplitudne ovojnice koja modelira drvene puhačke instrumente (Chowning, 1973):



Slika 3.8: ADSR ovojnica za model drvenih puhačkih instrumenata

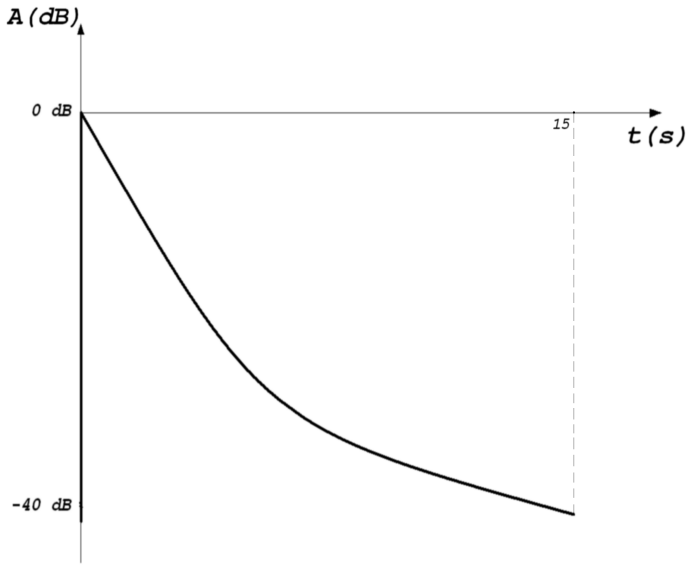
Ako želimo modelirati npr. klarinet, omjer *C:M* ćemo postaviti na npr. $3:2$ tako da se u spektru nađu samo neparni harmonici. Za ovaj slučaj, signal nosioc neće predstavljati osnovnu frekvenciju titranja, jer će se neki stvoreni harmonici nalaziti niže u spektru. Tako ćemo za frekvenciju nosioca od $f_{nosilac} = 900Hz$ i frekvenciju modulatora $f_{modulator} = 600Hz$ imati osnovnu frekvenciju titranja od $300Hz$.

Indeks modulacije možemo staviti da linearno pada od $4 - 2$ tijekom *attack* faze signala te dalje ostaje konstanta, no realniju i prirodniju boju tona ćemo dobiti ako upotrijebimo modulacijsku ovojnicu sličnu onoj na slici 3.8 ali bez *release* segmenta. Umjesto tog segmenta produžuje se *sustain* faza do kraja trajanja note.

Ovojnica udaračkih instrumenata

Općenita ovojnica za veliki broj udaračkih instrumenata je eksponencijalnog oblika, kao na slici 3.9. Dvije činjenice su važne za ovaj tip instrumenata:

1. Spektralne komponente instrumenta su u inharmoničkim odnosima
2. Spektar evoluira kroz vrijeme od kompleksnog do jednostavnog

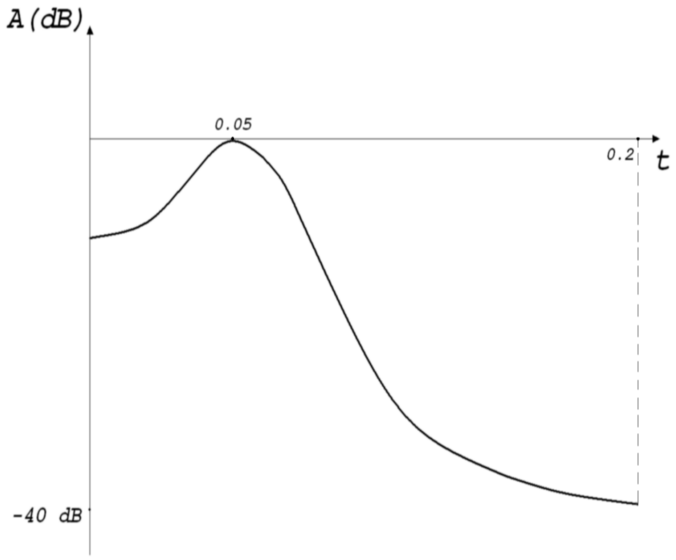


Slika 3.9: ADSR ovojnica za model zvonolikog zvuka

Ako isti oblik ovojnice primjenimo na signal modulator, rezultirajući signal će biti zvonolikog zvuka. Omjer *C:M* treba biti u inharmoničkom odnosu, pa možemo odabrati npr. omjer $1 : 1.4$.

Za zvonolike zvukove, indeks modulacije se mora promijeniti sa relativno velike vrijednosti na malu vrijednost tijekom trajanja signala, no ako želimo postići da se zvuk čuje poput udarca o bubanj, indeks modulacije te ovojnicu sa slike 3.9 moramo malo modificirati.

Kao prvo, udarci o bubanj su vrlo kratkog trajanja tako da se signal treba modulirati u kratkom vremenu, kroz npr. $200\ ms$. Kao drugo, treba malo modificirati oblik ovojnice tako da izgleda kao na slici 3.10:



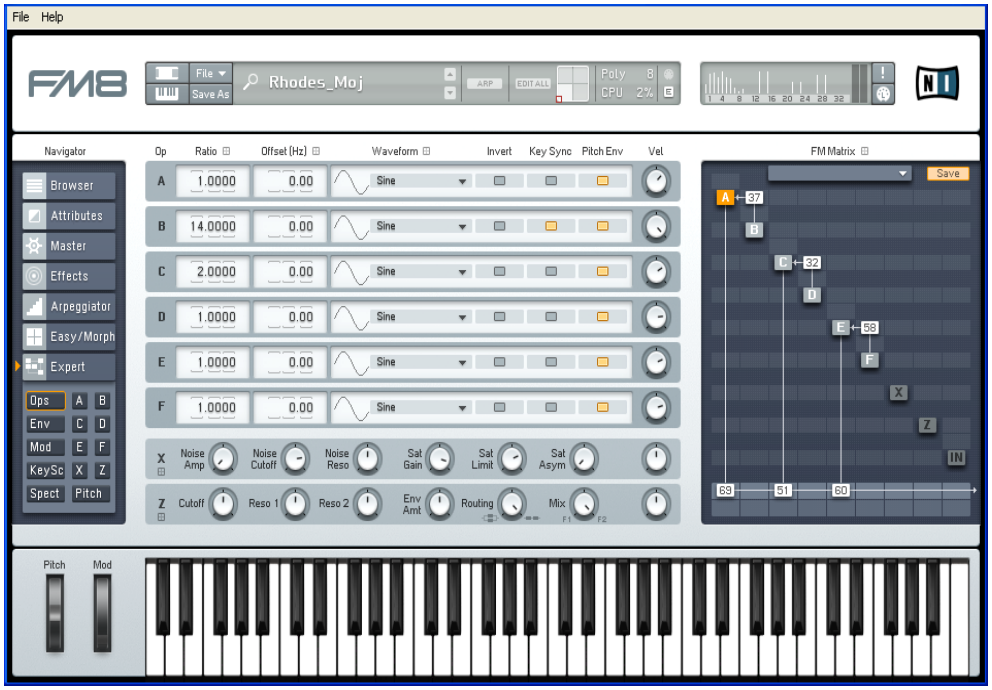
Slika 3.10: ADSR ovojnica za model bubnjeva

Za promjenu indeksa modulacije možemo zadržati oblik ovojnice sa slike 3.9, uz naravno kraće trajanje($200\ ms$), te raspon indeksa od maksimalne do minimalne vrijednosti $2 - 0$.

Iz svih prethodno navedenih primjera možemo vidjeti kako se metoda FM sinteze odlikuje vrlo velikom fleksibilnošću te jako širokom paletom instrumenata, uz kontrolu samo nekoliko parametara. Postoji i mogućnost nadogradnje jednostavnog modela sa slike 3.1 dodavanjem većeg broja nosilaca i modulatora, od kojih na svaki možemo primjeniti zasebne ovojnice te odabrati tip signala (sinusni, kosinusni, pravokutni, pilasti, trokutasti...) (Schottstaedt, 1985).

Vrlo jednostavno je nadograditi i razne efekte koji mijenjaju percepciju signala poput reverberacije, delay efekta, chorus efekta, dodavanje šuma u signal, filtracije itd.

Svoju popularnost i raskošnost FM sinteze najbolje predočuje Yamahin DX7 sintesajzer, koji je zaživio tijekom 80-ih godina 20. stoljeća, no danas također postoje i brojni softverski emulatori ovog popularnog sintesajzera. Jedan takav emulator je i ***"FM8"*** tvrtke *Native Instruments* (Native-Instruments, 2011).



Slika 3.11: Native Instruments FM8 emulator

Sa slike 3.11 možemo vidjeti da sa desne strane postoje mali blokovi označeni slovima A–F koji predstavljaju generatore valnih oblika Svaki od tih generatora može biti nosilac ili modulator, te svaki ima svoju podesivu ovojnicu. Korisniku je na raspolaganju jako veliki broj promjenjivih parametara, od ***C:M*** omjera, audio efekata, polifonije, parametara filtriranja, pa sve do kontrola glavnog miksera.

Posve je očito da je FM sinteza jedan jako moćan alat, koji će sigurno još generacijama biti proučavan i korišten.

3.2. Fizikalno modeliranje

Metoda fizikalnog modeliranja je, poput FM sinteze, jedna od metoda koja nudi veliku fleksibilnost te je u današnje vrijeme jedna od popularnijih metoda. Koncept se temelji na osnovnim fizikalnim zakonima širenja zvuka, otkuda je i dobila svoj naziv. Osnovna ideja polazi od toga da se svaki proces nastajanja zvuka može opisati egzaktnim matematičkim jednadžbama što je dosta složen proces, pa zbog toga ova metoda nije u potpunosti zaživjela sve do 80-ih godina 20. stoljeća.

Osnovna jednadžba koju modeliraju metode fizikalnog modeliranja instrumenata je diferencijalna jednadžba putujućih valova (Pekonen, 2007):

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial^2 y}{\partial x^2} \tag{3.11}$$

Riješenja za ovaj tip jednadžbe predložio je francuski matematičar d'Alembert, koja glase:

$$y(x,t) = y_+(t - \frac{x}{c}), \qquad val\ koji\ se\ giba\ udesno \tag{3.12}$$

$$y(x,t) = y_-(t + \frac{x}{c}), \qquad val\ koji\ se\ giba\ ulijevo \tag{3.13}$$

Generalno riješenje jednadžbe 3.11 je linearna superpozicija gornja dva izraza 3.12 i 3.13.

3.2.1. Karplus–Strong model

Model koji se danas koristi i koji je osnovno polazište za sintezu raznih klasa glazbenih instrumenata predstavila su dvojica znanstvenika Kevin Karplus i Alex Strong svojim radom iz 1983. godine (Karplus i Strong, 1983). Njihov koncept modeliranja trznute žice pomoću samo par komponenata pokrenuo je niz istraživanja i revolucionarizirao cijeli sustav modeliranja fizikalnim procesima.

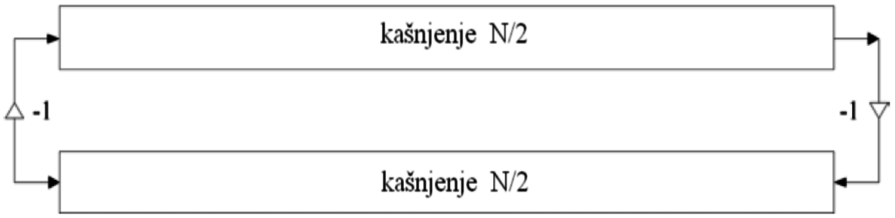
Osnovnu ideju je dobio Alex Strong proučavajući wavetable metodu sintetiziranja. Wavetable sinteza iskorištava jednu osobinu zvučnih signala, a to je njegova *kvazi-periodičnost*. U valnu tablicu se spremaju jedan ili dva perioda snimljenog signala instrumenta te se, za reproduciranje duljih trajanja nota, valna tablica vrti u petlji. Međutim, ovakav postupak će rezultirati sa potpuno periodičnim signalom sa jednoličnim spektrom, dok su spektri realnih instrumenata promjenjivi s vremenom.

Najjednostavnija ideja Karplus–Strong algoritma je da se svaki uzorak perioda signala računa kao srednja vrijednost dvaju uzoraka iz prethodnog perioda prema jednadžbi (Karplus i Strong, 1983):

$$y[n] = \frac{1}{2}(y[n-N] + y[n-N-1]), \qquad (3.14)$$
$$N = period\, valne\, tablice$$

Jednadžba 3.14 se može promatrati kao digitalni filtar, koji na svom izlazu daje signal frekvencije: $\frac{f_s}{(N+\frac{1}{2})}$. Broj *N* se još naziva i *kašnjenje delay linije*.

Fizikalno objašnjenje ove jednadžbe možemo dati pomoću slike 3.12 na kojoj su nacrtane grane putujućih valova, svaka sa kašnjenjem od $N/2$ (Park, 2010):

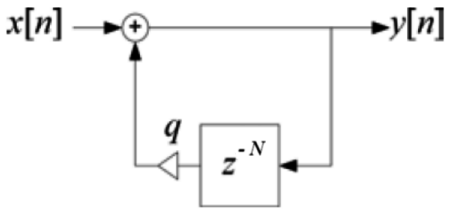


Slika 3.12: Jednostavni model putujućih mehaničkih valova učvršćene žice

Zamislimo, na trenutak, da u ruci držimo dugački komad užeta pričvršćen jednim krajem o stup ili stablo. Uže držimo napetim i periodički ga trzamo što uzrokuje putujuće valove prema drugom kraju užeta. Val koji dođe do učvršćenog kraja se reflektira, fazno zakrene, te se počne vraćati prema izvoru trzaja.

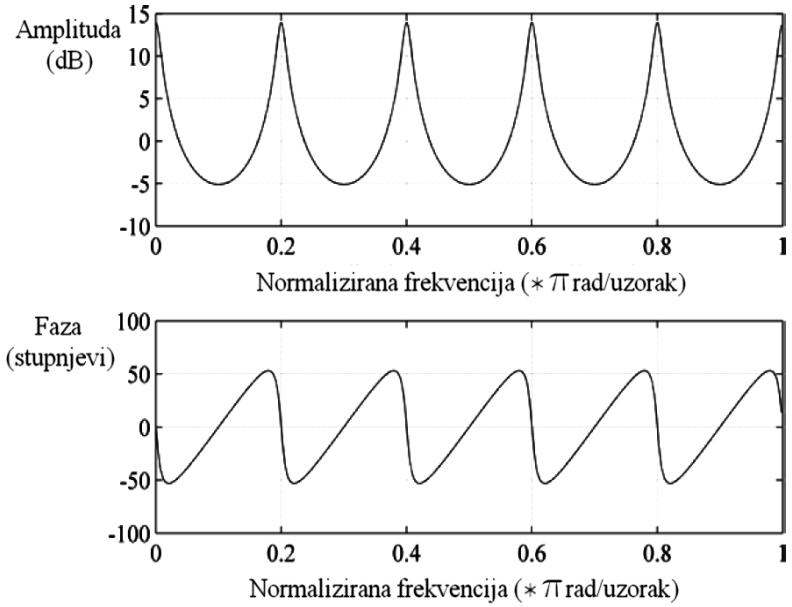
Točno ovaj proces opisuje i gornja slika. Gornja linija kašnjenja predstavlja val koji putuje prema učvršćenom kraju tj. u pozitivnom smjeru, dok donja linija kašnjenja predstavlja negativno putujući val, tj. onaj koj se vraća prema izvoru trzaja. Svaka linija kasni signal za *N/2* uzoraka što znači da val za *N* uzoraka napravi jedan period. Koeficjenti –1 predstavljaju idealnu refleksiju valova na krajevima užeta. U stvarnosti ti koeficjenti nisu idealni, jer valovi postepeno gube svoju energiju reflektirajući se s jednog kraja na drugi.

Osim gubitaka koji se javljaju pri refleksiji, postoje i gubici pri samom širenju valova kroz žicu kao što su npr. unutarnje trenje žice, trenje sa okolnim zrakom...itd. Ove gubitke možemo koncentrirati u jednu nezavisnu varijablu $\boldsymbol{q}$. To smijemo napraviti jer pri dizajniranju digitalnih filtara radimo sa *LTI (eng. Linear Time–Invariant)* sustavima. Ako, pri tome, još i cijelokupno kašnjenje obiju linija koncentriramo u jedan element, dobiti ćemo sustav poput onog na slici 3.13:



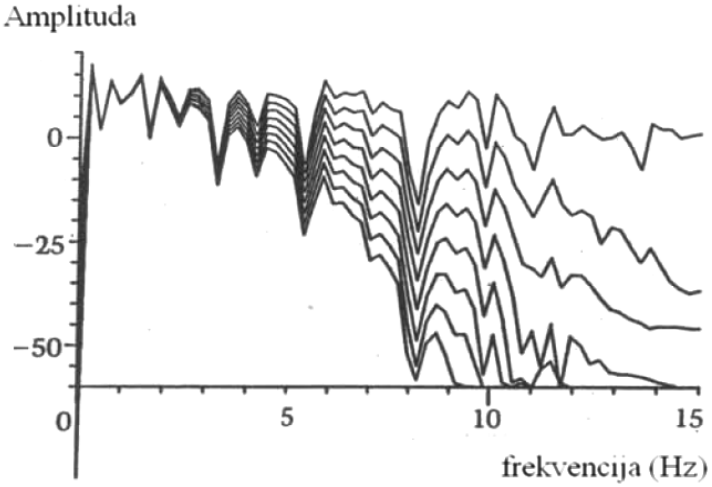
Slika 3.13: Jednostavni digitalni filtar koji modelira širenje valova kroz žicu

Sustav sa slike 3.13 predstavlja tzv. češljasti filtar (*engl. comb–filter*) kod kojeg je za stabilnost sustava potrebno vrijediti $|q| < 1$. Amplitudno–frekvencijska karakteristika češljastog filtra prikazana je slijedećom slikom (Park, 2010):



Slika 3.14: Amplitudno-frekvencijska karakteristika češljastog filtra

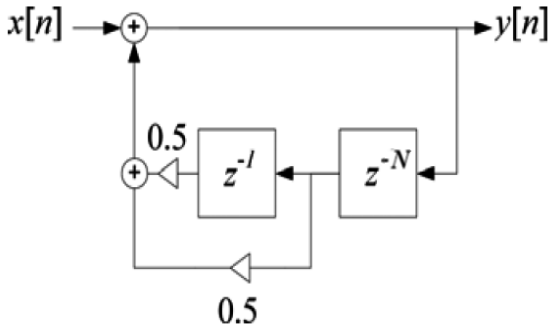
Ako implementiramo model titranja žice sa ovim češljastim filtrom, rezultati baš i neće zvučati prirodno. Razlog tome je što se svi harmonici titranja prigušuju jednoliko faktorom q. Već smo ranije napomenuli da za realističan zvuk instrumenta, viši harmonici moraju biti jače prigušeni i to sa različitim faktorima prigušenja, kao npr. na slici 3.15 (Jaffe i Smith, 1983):



Slika 3.15: Prigušivanje harmonika kod realnih instrumenata

Češljasti filtar nam ne nudi ovo obilježje glazbenih instrumenata, međutim ako gubitke u sustavu predstavimo ranije spomenutom jednadžbom 3.14, dobiti ćemo prigušivanje harmonika slično onome na slici 3.15.

Sustav sada izgleda malo drugačije:



Slika 3.16: Osnovni Karplus-Strong model digitalnog filtra

Jednadžba diferencija koja implementira filtar sa slike 3.16 glasi:

$$y[n] = x[n] + \frac{y[n-N] + y[n-N-1]}{2} \tag{3.15}$$

Gornja jednadžba je jedan običan nisko–propusni filtar koji svakim prolazom kroz petlju povratne veze sve više prigušuje visoke harmonike. Ovim jednostavnim modelom se postižu vrlo uvjerljivi i realistični rezultati.

No pitamo se: ”Kakvim signalom trebamo pobuditi ovaj NP–filtar da bi dobili željeni odziv trznute žice?”. Odgovor je iznenađujuće jednostavan, sve što trebamo

napraviti je na ulaz filtra dovesti signal jako bogat harmonicima, odnosno šum! Šum unosi relativno veliku energiju u signal te se, prolaskom kroz konstruirani NP filtar, ta energija postepeno gubi.

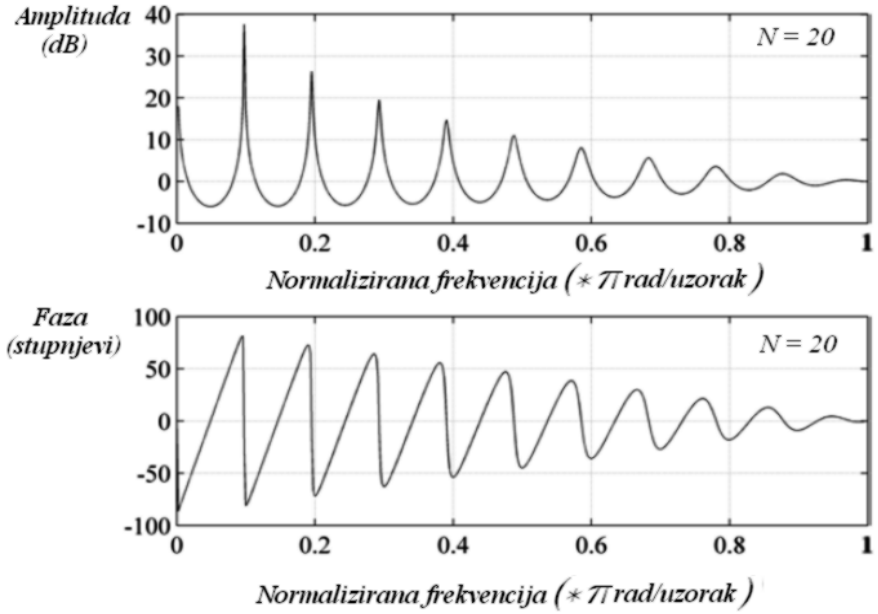
Ako npr. ulazni signal napunimo slučajnim uzorcima, te ga propustimo kroz ovako konstruirani filtar, izlazni signal će uvijek biti sličnog oblika i spektra bez obzira na to kakav je pobudni signal. Zanimljiva posljedica koja proizlazi iz ove činjenice je ta da ako uzastopce pobuđujemo filtar različitim slučajnim uzorcima, svaki odziv će biti za nijansu drugačiji od prethodnog, što dodatno poboljšava prirodnost zvuka koji se modelira.

3.2.2. Amplitudno–frekvencijska karakteristika i stabilnost

Jednadžbu 3.15 možemo zapisati i u malo drugačijem obliku. Ako na nju primjenimo z–transformaciju, dobiti ćemo prijenosnu funkciju digitalnog filtra *H(z)*:

$$H(z) = \frac{1}{1 - 0.5z^{-N} - 0.5z^{-(N+1)}} \tag{3.16}$$

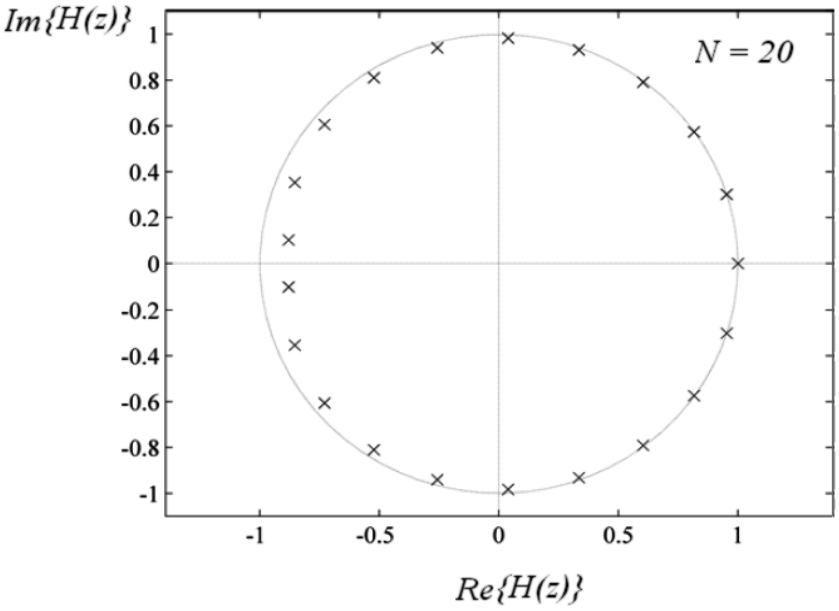
Ako sada, pomoću jednadžbe 3.16 (uz supstituciju $z = e^{j\omega}$), nacrtamo Bodeov dijagram za Karplus–Strong filtar sa slike 3.16, iz amplitudno–frekvencijske karakteristike možemo uočiti postupno prigušavanje harmonika (Park, 2010):



Slika 3.17: Bodeov dijagram za osnovni K–S model

Uočimo da je amplitudno–frekvencijska karakteristika sa slike 3.17 vrlo slična karakteristici češljastog filtra (slika 3.14), no harmonike dodatno prigušuje nisko–propusni filtar kojeg smo dodali u petlju povratne veze.

Nadalje, ako želimo analizirati stabilnost sustava, promotrit ćemo polove u z–ravnini sa slike 3.18. Polove označavaju križići koji se svi nalaze unutar jedinične kružnice, što nam govori da je sustav stabilan. Polovi koji su bliže rubovima jedinične kružnice, na granici su stabilnosti i predstavljaju niske harmonike izlaznog signala koje filtar slabije prigušuje. Viši harmonici su jače prigušeni, pa su zbog toga njihovi polovi više odmaknuti od kružnice te su stabilniji.



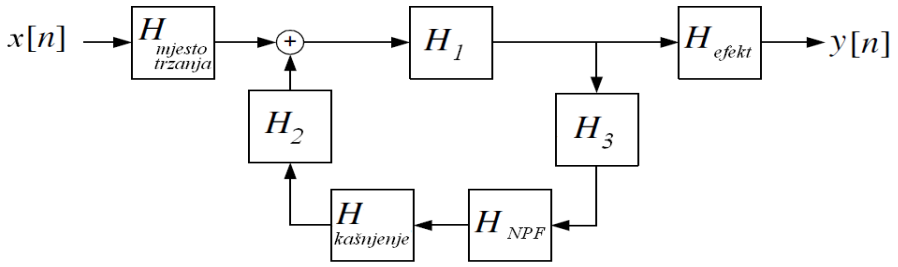
Slika 3.18: K–S filtar u z–ravnini

3.2.3. Proširivanje modela

Do sada smo pokazali kako izgleda jednostavan model Karplus–Strong algoritma trznute žice, no taj model je moguće proširiti sa dodatnim blokovima i primjeniti ih na razne klase instrumenata. Kod dodavanja blokova iskorištavamo svojstva linearnih vremenski–invarijantnih sustava, koja nam dopuštaju da blokove nadodajemo u sustav na koja mjesta želimo te ih međusobno konvoluiramo.

Tako, na primjer, u osnovni K–S model trznute žice možemo nadodati element koji simulira mjesto na kojem je žica trznuta, kojom jačinom je trznuta, možemo u izlaznu granu dodati neki audio efekt itd.

Primjer dodavanja elemenata dan je na slijedećoj slici:



Slika 3.19: Proširivanje osnovnog Karplus–Strong modela

Sa slike 3.19 vrlo lako možemo zaključiti kako je nadogradnja modela iznimno jednostavan proces. Svaki blok modela predstavlja neovisnu funkciju koja se po potrebi može i maknuti, bez utjecaja na funkcionalnost ostalih blokova. Prijenosne funkcije H_1, H_2, H_3 su funkcije koje se stavljaju u granu povratne veze i općenita namjena im je poboljšavanje realističnosti instrumenta te korekcija delay linije.

Kada se izgradi model nekog instrumenta, običaj je na neki način procijeniti sve parametre modela i pojedinih blokova. To se u većini slučajeva radi sa analizom niza akustičkih mjerenja koja se provode u gluhim komorama tako da se dobiju što točnija mjerenja bez utjecaja okolne buke ili karakteristika prostora (veliki odjek npr.).

Karplus–Strong algoritam je pokrenuo pravu revoluciju u svijetu modeliranja instrumenata. Tako je iste godine kada je objavljen Karplus–Strong članak, objavljen i članak koji je poopćio primjenu K–S algoritma, te ga učinio primjenjivim na širok skup glazbenih instrumenata (Jaffe i Smith, 1983).

3.3. Wavetable sinteza

Većina komercijalnih sintesajzera u današnje vrijeme sadrži u sebi nekoliko metoda sinteze instrumenata. Jedna od najćešćih, koju ima praktički svaki malo bolji sintesajzer, je svakako tzv. *Wavetable sinteza* ili *Sinteza valnim tablicama*.

3.3.1. Konstrukcija valnih tablica

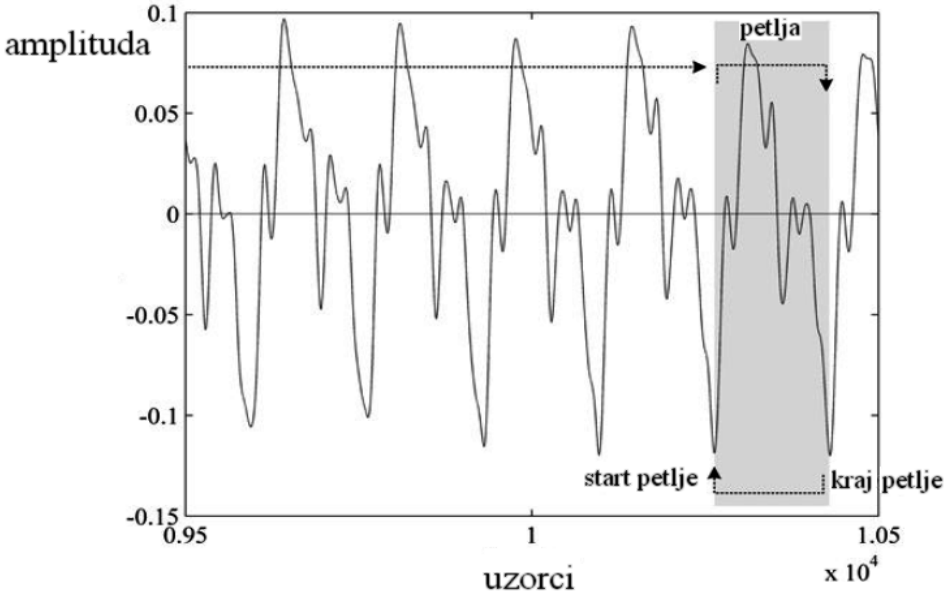
Već smo ranije napomenuli da je metoda valnih tablica potakla ideju za Karplus-Strong algoritam koja je uspjela poboljšati mane wavetable sinteze, no to nikako ne znači da je KS sinteza posve prevladala. U nekim slučajevima, wavetable sinteza je nezamjenjiva jer nudi neke mogućnosti koje ostale metode modeliranja ne mogu ponuditi.

Kao primjer uzmimo bubnjeve za modelirajući instrument. Uza sve današnje moderne tehnike sinteze, prirodni i potpuno realistični model bubnjeva je dosta teško dobiti, pogotovo ako se radi o činelama koje imaju izrazito specifičan inharmonički spektar. Naravno, model je moguće aproksimirati sa zadovoljavajućim rezultatima, no ako želimo realističan odziv udarca o bubanj, pribjeći ćemo tehnici wavetable sinteze.

Tu se radi o zapravo vrlo jednostavnoj ideji, a to je da se odziv instrumenta snimi u memoriju sintesajzera, odnosno u valnu tablicu, te se po potrebi ta tablica iščitava različitim brzinama. U tablicu se obično spremaju vrlo kratki kvazi-periodični segmenti signala iz kojih je potrebno moći sintetizirati zvuk bilo kojeg trajanja.

Broj uzoraka koji ćemo pospremiti u tablicu je naravno proizvoljan i najčešće ovisi o memorijskim ograničenjima uređaja. Kada na raspolaganju imamo ograničenu memoriju, a trebamo imati jako puno valnih tablica, od kojih svaka može predstavljati zaseban instrument, odabrat ćemo veličinu tablice između 1024 i 4096 uzoraka (veličina je potencija broja 2), kako bi mogli pospremiti što više valnih oblika. Da bi sintetizirali određenu notu, sve što moramo napraviti je potražiti odgovarajuće uzorke u tablici te ih na neki način dinamički obraditi pomoću algoritma za obradu valne tablice.

Recimo da u valnu tablicu pospremimo npr. 4096 uzoraka note A4 klavira, imati ćemo jedan vrlo kratak segment koji možemo produljiti tako da ga vrtimo u petlji kao na slici 3.20 (Park, 2010):



Slika 3.20: Konstrukcija signala iz valne tablice

Ako imamo snimljen samo jedan period signala, taj period ćemo vrtiti u petlji od start pozicije pa do krajnje pozicije kao na gornjoj slici. Pri tome moramo pažljivo odabrati pozicije između kojih se petlja vrti kako ne bi došlo do čujnih diskontinuiteta signala.

Još jedan detalj na kojeg moramo paziti jest odabir frekvencije uzorkovanja $f_s(Hz)$ kojom smo otipkali signal valne tablice, te rezolucija kvantizacije uzoraka $n\,(bit)$. Ova dva faktora su jako bitna jer o njima ovisi kvaliteta izlaznog sigala. Prema Nyquistovom kriteriju uzorkovanja, frekvencija f_s ne smije biti manja od dvostruke maksimalne frekvencije signala kojeg uzorkujemo: $f_s \geq 2 \cdot f_{max}$. Ako odaberemo manji f_s od te vrijednosti, postoji mogućnost pojave *aliasinga* tj. može doći do čujnih izobličenja izlaznog signala. Za audio potrebe, gornja granična frekvencija koju ljudsko uho može čuti iznosi $20\,kHz$, pa tako minimalna frekvencija uzorkovanja mora iznositi $40\,kHz$. Kao standard se obično uzima vrijednost $f_s = 44.1\,kHz$, što je i više nego dovoljno da se spriječi pojava aliasinga.

Broj bitova kvantizacije uzoraka valne tablice n nam određuje preciznost valne tablice tj. u kojem rasponu amplituda ćemo dodjeljivati vrijednosti uzorcima. Ako uzmemo npr. $n = 16\,bitova$, raspon amplituda će biti $-2^{16}/2 \leq A \leq 2^{16}/2$ odnosno $-32768 \leq A \leq 32768$. Karakteristične vrijednosti za n su standardizirane i obično su jednake $n = 8, 12, 16, 32$, od kojih se za audio signale najčešće koristi $n = 16\,bitova$ (Thomsen, 1987).

Pošto snimljeni period signala vrtimo u petlji, njegova ovojnica će ostati nepromjenjena kroz vrijeme, tako da na izlazni signal moramo primjeniti i neki tip već ranije spomenutih ADSR ovojnica kako bi se simulirao realističan amplitudni odziv signala (Slika 3.21).

Slika 3.21: Oblikovanje ovojnice

3.3.2. Spektar wavetable sintetiziranog signala

Glavni problem sa kojim se suočavamo pri modeliranju wavetable sintezom je odabir dijela signala snimljenog instrumenta kojeg ćemo koristiti u našoj valnoj tablici. O tom segmentu ovisi izlazni spektar sintetiziranog signala.

Znamo već da realni instrumenti imaju vremenski promjenjiv spektar, no generalno vrijedi da većina instrumenata ima relativno stacionarno ponašanje u svojoj sustain fazi. To je takozvano svojstvo kvazi–periodičnosti signala koje možemo iskoristiti za reprodukciju sintetiziranog signala. No, međutim, dio signala kojeg pospremimo u valnu tablicu iz tog stacionarnog segmenta, ostati će konstantan kada ga provučemo kroz petlju za dobivanje duljih trajanja nota. Time smo dobili periodički signal sa vremenski nepromjenjivim spektrom što se kosi sa temeljnim svojstvom realnih instrumenata, a to je njihova spektralna promjenjivost u vremenu.

Ovaj problem se može riješiti na nekoliko načina. Jedan od njih je da se u valnu tablicu pospremi čitav attack segment signala, i samo mali dio sustain segmenta koji će se do kraja trajanja note vrtiti u petlji. Time dobivamo poprilično zadovoljavajuće rezultate jer se većina spektralne energije realnih zvukova nalazi u njihovoj attack fazi koja je u ovom slučaju potpuno očuvana jer je u cijelosti snimljena u valnu tablicu. Mana ovakvog pristupa je to što sada trebamo više memorije za tablicu, pa zato možemo projektirati samo ograničen broj instrumenata s kojima možemo raspolagati u sintesajzeru.

Za svaki instrument pohranjen u memoriji uređaja, potrebno je biti u mogućnosti sintetizirati cijeli raspon frekvencija tog instrumenta. Ako u valnoj tablici imamo pohranjen samo jedan kratki segment jedne jedine note, kako dolazimo do ostalih nota?

U poglavlju 2.2 smo pokazali da između frekvencija nota postoje određeni matematički odnosi (jednadžba 2.1). Upravo tu činjenicu koristimo za rekonstruiranje svih glazbenih nota iz jednog pospremljenog segmenta valne tablice. Naime, snimljene uzorke možemo jednostavno samo malo brže ili sporije ”pročitati” iz tablice te na taj način dobiti više/niže frekvencije. Ako npr. iz valne tablice očitavamo svaki drugi uzorak, dobiti ćemo duplo višu frekvenciju signala od one pohranjene u tablici odnosno za oktavu višu notu.

Inkrement, koji nam određuje kojim korakom uzimamo uzorke iz tablice određujemo pomoću slijedećeg izraza (Kuo i Lee, 2001):

$$inkr = \frac{N \cdot f_0}{f_s} \tag{3.17}$$

U gornjem izrazu *N* označava broj uzoraka koji je pohranjen u tablici, f_0 je frekvencija koju želimo dobiti pomoću tablice, a f_s je naravno frekvencija uzorkovanja. Pošto inkrement često nije cijelobrojnog tipa već broj sa frakcionalnim dijelom, a uzorci iz tablice su iterirani sa cjelobronim vrijednostima, moramo na neki način doći do vrijednosti uzoraka za frakcionalni inkrement. U ovakvim slučajevima se obično vrijednosti tablice interpoliraju pomoću prvog prethodnog i prvog slijedećeg uzorka tablice, te na taj način povećavamo preciznost potraživanja u valnoj tablici.

4. MIDI

MIDI poruke su standard u svijetu glazbe pomoću kojeg elektronički glazbeni instrumenti i kontroleri komuniciraju međusobno ili sa računalom. Kratica dolazi od engleskih riječi ***M**usical **I**nstrument **D**igital **I**nterface* što u grubom prijevodu znači: *Digitalno Sučelje Glazbenih Instrumenata*.

Ovim porukama se šalju informacije o glazbenim događajima koje neki instrument može proizvesti kao što su npr. broj stisnute note, jačina pritiska note, broj programa instrumenta koji proizvodi notu, te slični. Svaki instrument koji podržava MIDI poruke slijedi istu MIDI 1.0 specifikaciju (Wikipedia, 2011a) pa se na taj način osigurava kompatibilnost svih MIDI uređaja.

4.1. Uvođenje MIDI poruka u okolinu MATLAB–a

Najčešći instrument koji podržava kontrolu i manipulaciju MIDI porukama je standardni sintesajzer sa klavirskim tipkama kod kojega je svaka tipka zadužena za produkciju nezavisnog skupa MIDI poruka.

Općenito vrijedi pravilo da MIDI veza, pomoću odgovarajućih MIDI kablova, predstavlja jednosmjernu komunikaciju između odašiljača (MIDI Out) i prijamnika (MIDI In) različitih uređaja. Streamovi MIDI poruka se na taj način mogu slati iz jednog uređaja prema drugom, i to na 16 različitih kanala kako bi se razdvojili različiti instrumenti koje je moguće istovremeno reproducirati.

Najčešće kategorije poruka koje se mogu slati preko MIDI sučelja su slijedeće:

1. ”NOTE ON” i ’NOTE OFF”
   – označavaju da li je određena nota pritisnuta ili otpuštena

2. ”PITCH–BEND”
   – indikacija da je ton podignut/spušten za ± 2 polutona preciznosti i do 1/8192.

3. ”CONTROL CHANGE”
   – ove poruke se šalju kada muzičar pritisne neki gumb ili zaseban uređaj koji na neki način modificira izlazni zvuk instrumenta

4. ”PROGRAM CHANGE” – generiraju se pri promjeni lokacije sa koje se učitava algoritam za instrument, tj. kada muzičar želi promjeniti instrument na kojem svira odabrat će odgovarajući broj programa koji taj instrument čini dostupnim za uporabu. Podržano je samo 7 bitova za ovaj tip poruka pa je ukupan broj programa rezerviran za instrumente jednak 128. Taj broj se može povećati dodavanjem banki (*engl.* Soundbank).

5. ”AFTERTOUCH”
   – ove poruke podržavaju samo neki sintesajzeri, a označavaju da je na neku tipku mijenjan pritisak za vrijeme dok je stisnuta

6. ”SYSTEM EXCLUSIVE”
   – ove poruke definira proizvođač i često ne spadaju u skupinu korisnih MIDI poruka već su vezane za npr. pražnjenje memorije sekvencera ili promjenu podešenja sintesajzera

Svaka tipka na sintesajzeru odgovara određenoj glazbenoj noti, koja kada se pritisne mora reproducirati zvuk željenog instrumenta i željene frekvencije. Zbog toga je svaka tipka kodirana sa vlastitim MIDI brojem koji služi za prepoznavanje nota. Ako, na primjer, stisnemo notu C4 na sintesajzeru ona će uvijek biti kodirana sa MIDI brojem 60. Cijeli raspon kodiranih nota možemo vidjeti u tablici 4.1.

Pokriveno je ukupno 128 nota od note C-1 sve do note G9, no rijetko koji instrument ima preko 88 tipaka pa efektivni raspon često ide samo od C0 do C8.

Za pretvorbu MIDI broja note u odgovarajuću frekvenciju možemo upotrijebiti slijedeći izraz koji za referentnu notu uzima notu A4 čiji MIDI ekvivalent je broj 69:

$$f = 440 \cdot 2^{(broj-69)/12} \tag{4.1}$$

Tipično se, uz informaciju o trenutno stisnutoj noti, šalje i informacija o tome na kojem kanalu je nota reproducirana, te kojom jačinom je stisnuta. Poruka koja sadrži te tri informacije je formirana u niz od 3 byte-a, od kojih prvi byte[1] odgovara NOTE ON ili NOTE OFF događaju na odgovarajućem kanalu, drugi byte je rezerviran za broj

[1]Prva četiri bita u byte-u (LSb) su broj kanala, a druga četiri bita (MSb) su NOTE ON/NOTE OFF status

note, te je konačno treći byte namjenjen za broj koji predstavlja jačinu kojom je nota pritisnuta.

Raspon brojeva, za prvi byte poruke, u kojem možemo očekivati NOTE ON statuse je 144 do 159 (ako su brojevi prikazani kao predznačni onda je raspon -112 do -97) gdje 144 označava da je nota pritisnuta na 1. kanalu, dok brojka 159 znači da je nota pritisnuta na 16. kanalu. Slično je i sa statusom NOTE OFF kojem je raspon brojeva 128 do 143, gdje opet svaki broj predstavlja zaseban kanal.

Tablica 4.1: Note kodirane MIDI 1.0 standardom

Oktava	Note i pripadni MIDI brojevi											
-	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Za potrebe ovoga rada, MIDI note ćemo u programsku okolinu MATLAB-a uvesti pomoću jednog običnog MIDI-USB kabla te pomoću mogućnosti uvođenja korisničkih Java klasa u MATLAB sučelje.

MATLAB standardno nudi pristup svim ugradbenim Java klasama pomoću kojih se može proširiti funkcionalnost mnogobrojnih problema, no daleko najkorisnije klase su one koje se tiču korisničkog i eventualno grafičkog sučelja. Da bi bili u mogućnosti povezati MATLAB sa uređajem koji će slati MIDI poruke, moramo napraviti dvije stvari: prva je prijamnik koji će čitati dolazeće poruke sa USB porta, preko kojega je spojen MIDI-USB kabel, a druga je pripremiti spremnik (*engl.* buffer) u koji će se te poruke pospremati kako bi bile dostupne za dohvat našem MATLAB programskom kodu. Nakon što se te dvije Java klase implementiraju, potrebno ih je samo dodati u MATLAB-ov *classpath* kako bi bile prepoznatljive kroz cijelo sučelje.

4.2. Prijamnik poruka

Baš kao u svim komunikacijskim uređajima, radio prijamnicima, mobilnim telefonima, televizorima, računalima itd. za kvalitetan prijenos podataka mora postojati veza prijamnik-odašiljač. Upravo je to ono što je potrebno da dva uređaja međusobno kvalitetno razmijenjuju informacije. U našem slučaju, uređaji su računalo i sintesajzer, koje na neki način moramo navesti da razmijenjuju informacije.

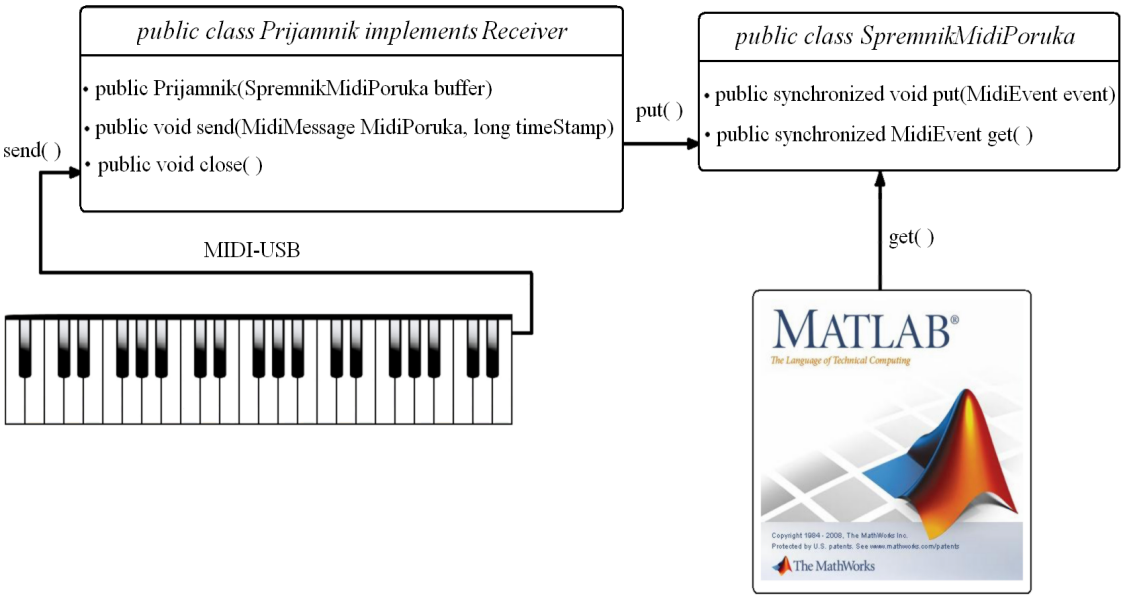
Programski kod ovoga rada je većinom pisan u MATLAB-u, no za potrebe MIDI komunikacije moramo se poslužiti dodatnim alatima koji se vrlo lako mogu ukomponirati u sučelje MATLAB-ove radne okoline. Pošto je MATLAB velikim dijelom napisan u programskom jeziku *Java* (Cadenhead i Lemay, 2007), te nudi pristup ugradbenim Java klasama i paketima, tu činjenicu ćemo iskoristiti u svrhu realizacije prijamnika i spremnika MIDI poruka .

Dakle, kada spojimo sintesajzer sa računalom dobili smo ništa više nego fizičku vezu dvaju uređaja. Da bi mogli detektirati kakve podatke sintesajzer šalje na USB port računala moramo programirati prijamnik koji će se povezati sa ulaznim portom i kontinuirano ”osluškivati” kakve informacije se šalju. Programski jezik Java ima hrpu ugradbenih klasa, paketa i sučelja koji dolaze u sklopu svakog programerskog Java SDK (*engl.* Software Development Kit) okruženja. Za čisti pristup ovim klasama i paketima, MATLAB je dorastao problemu i sintaksa kojom se služi je dosta slična onoj u Javi, no kada je potrebno napisati korisničku klasu, koja npr. implementira neko određeno sučelje, mora se posegnuti za drugim programerskim okruženjem.

Jedno takvo okruženje je *Eclipse SDK* (Eclipse, 2011) programersko sučelje koje je iznimno popularno među Java developerima i programerima, i kojim ćemo se poslužiti u svrhu riješavanja problema komunikacije sa MIDI uređajem.

Nakon što se napiše potrebna Java klasa pomoću Eclipse programa, tu istu klasu je moguće ubaciti u MATLAB tako da je prepoznaje kao i sve ostale ugradbene klase i po potrebi je koristi u programskim realizacijama raznih projekata.

Na slijedećoj slici je prikazana blokovska shema komunikacije između sintesajzera i programske okoline MATLAB-a, koja je implementirana u okviru ovoga rada i kojom se koristimo za potrebe dohvata MIDI poruka:



Slika 4.1: Komunikacija sintesajzera sa MATLAB-om

Dvije su osnovne klase kojima se služimo u svrhu MIDI komunikacije : `Prijamnik.class` i `SpremnikMidiPoruka.class`. Obje klase sadrže karakteristične metode kojima se služe za međusobno razmijenjivanje podataka te za slanje informacija u radnu okolinu.

Klasa `Prijamnik` je napravljena tako da implementira Receiver sučelje i sadrži, pored konstruktor metode, još dvije metode: `send()` i `close()` (Oracle, 2011). Kada MIDI poruka stigne na USB port, zove se `send` metoda koja prima poruku kao argument, te vrijeme kada je poruka stigla u odnosu na vrijeme kada je ulazni port otvoren. U tijelu metode se ova dva ulazna argumenta kombiniraju u instancu jedne posebne klase, a to je `MidiEvent` klasa. Ovo je potrebno radi toga što ćemo, za potrebe ostatka programskog koda pisanog u MATLAB-u, morati biti u mogućnosti dohvatiti vremena NOTE_ON i NOTE_OFF događaja kako bi znali ispravno sintetizirati pojedine note, tj. kada ih trebamo paliti i gasiti. Kreiranje `MidiEvent` instance se može napraviti pomoću slijedećih Java naredbi:

```
//kreiraj MIDI evente iz MIDI poruke i timeStamp-a:
if (status == NOTE_ON || status == NOTE_OFF){
    MidiEvent event = new MidiEvent(MidiPoruka, timeStamp);
    spremnik.put(event);
}
```

Naredbom `spremnik.put(event)` instancu klase `MidiEvent` ubacujemo u

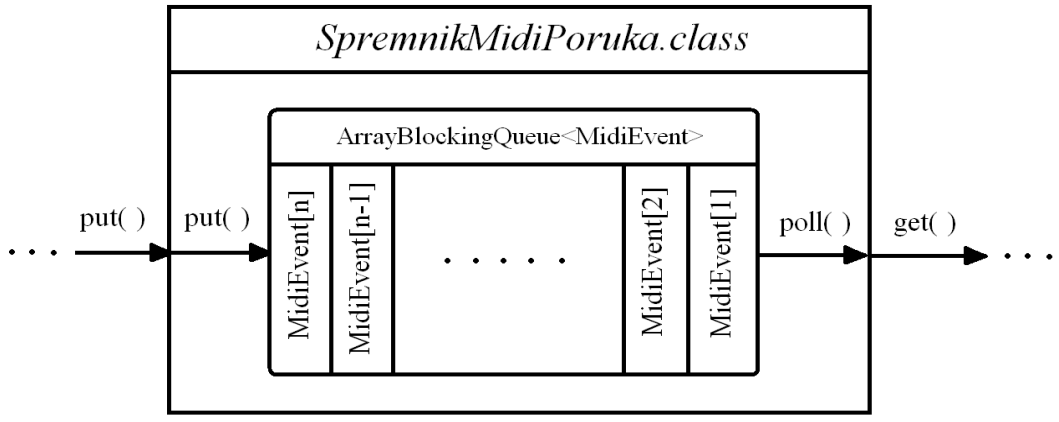
spremnik koji služi kao povezna točka sa našim MATLAB kodom, te iz kojeg dohvaćamo vremenski odštampane poruke za daljnju manipulaciju i sintezu nota instrumenata. Druga metoda koja se nalazi u `Prijamnik` klasi je `close()` metoda koja se najčešće ostavlja da ne radi ništa, ali je moramo uključiti u naš dizajn jer konstruiramo apstraktnu klasu koja ima preddefinirane metode koje moraju biti uključene da bi implementirale `Receiver` sučelje.

Prethodno opisani postupak je u biti osnovna funkcionalnost prijamnika, koja je nužno potrebna da bi klasa ispravno radila, no moguće je dodati neke naredbe koje npr. prate dohvat poruka, radi kontrole ispravnosti koda. Detaljan programski kod `Prijamnik` klase nalazi se u dodatku A.8, a slijedeći korak na kojeg prelazimo je spremnik MIDI poruka iz kojeg ćemo vaditi podatke pomoću glavnog MATLAB programa.

4.3. Spremnik poruka

Spremnik (*engl.* buffer) nam je potreban iz jednog jedinog razloga, a to je da organiziramo, sačuvamo i priredimo informacije, koje dobivamo iz prijamnika, za dohvat i daljnju manipulaciju pomoću glavnog programa.

Implementacija spremnika je vrlo jednostavna. Sve što nam je potrebno jest napraviti red `MidiEvent` objekata koje pospremamo na kraj reda kako stižu iz prijamnika, a vadimo ih sa početka reda. Radi lakšeg razumijevanja, na slijedećoj slici je prikazana shema klase `SpremnikMidiPoruka`:



Slika 4.2: Spremnik `MidiEvent` objekata

Za implementaciju reda objekata poslužiti ćemo se `ArrayBlockingQueue<>`

klasom iz `java.util.concurrent` paketa. Ova klasa omogućuje pospremanje i vađenje objekata reda pomoću dvije metode: `put()` metoda će objekt staviti na kraj reda, dok će `poll()` metoda izvaditi objekt sa početka reda. Na taj način smo u mogućnosti dohvatiti MIDI poruke onim redoslijedom kako su stigle na prijamnik bez straha da će se taj redoslijed pomiješati.

Da bi metode ovoga reda bile dohvatljive prijamniku i MATLAB-ovom glavnom programu, potrebno ih je realizirati unutar internih metoda `PrijamnikMidiPoruka` klase. Dakle, naš spremnik mora sadržavati također dvije metode `put()` i `get()` pomoću kojih komunicira sa ostalim klasama i programima.

Jedan poziv `get()` metode, klase `SpremnikMidiPoruka`, povući će za sobom poziv `poll()` metode `ArrayBlockingQueue<>` klase i na taj način dohvatiti jedan objekt `MidiEvent` klase. Slično se događa i kada pozovemo `put()` metodu klase `SpremnikMidiPoruka`, koja za sobom povlači pozivanje `put()` metode `ArrayBlockingQueue<>` klase. Programski kod koji realizira prethodno opisanu funkcionalnost je slijedeći:

```
public synchronized void put(MidiEvent event){
    try{
        //stavi MIDI event na početak reda:
        spremnik.put(event);

    }catch(InterruptedException e){
        System.out.println(e);
    }
}

public synchronized MidiEvent get(){
    //dohvati MIDI event sa vrha reda:
    noviEvent = spremnik.poll();
    return noviEvent;
}
```

Ove dvije metode moraju imati `synchronized` atribut kako bi se uskladilo naizmjenično spremanje i vađenje objekata iz reda. Potpuni programski kod se nalazi u dodatku A.9 na kraju ovoga rada.

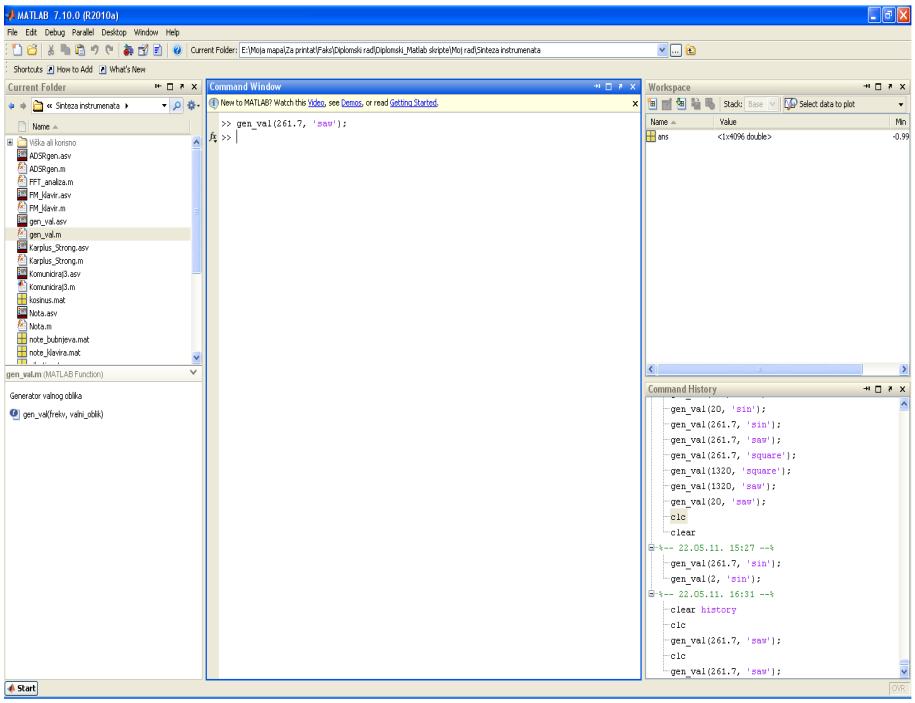
Sada kada imamo implementirane `Prijamnik` i `SpremnikMidiPoruka` Java klase, koje su u potpunosti funkcionalne, moramo ih učiniti dostupnima u MATLAB-

ovoj radnoj okolini. To ćemo učiniti tako da potpuni put do direktorija gdje se nalaze ove dvije klase dodamo u MATLAB-ovu *classpath.txt* datoteku. Ako u radnom prozoru MATLAB-a utipkamo `edit classpath.txt`, otvoriti će nam se prozor sa navedenom datotekom. Na bilo koje slobodno mjesto se upiše put do direktorija gdje se nalaze klase, te se datoteka pospremi i zatvori. Kod idućeg pokretanja MATLAB-a, klase će biti prepoznate i spremne za korištenje, baš kao i sve ostale ugradbene Java klase kojima je moguće pristupiti i iskorištavati ih u razne svrhe. Poprilično iscrpna dokumentacija svih dostupnih Java klasa može se pronaći na internetu (JavaAPI, 2011)

5. Implementacija algoritama za sintezu instrumenata

U prethodnim poglavljima smo diskutirali pretežito o teoretskim aspektima izvedbe pojedinih algoritama. Sada ćemo naglasak staviti na praktična programska riješenja koja ćemo implementirati u programskoj okolini MATLAB-a (Mathworks, 2011).

MATLAB je jedan jako koristan inženjerski i programerski alat koji omogućava inženjerima da naprave razne simulacije, testiranja i proračune u svrhu svojih projekata. Verzija programa koja je korištena u okviru ovoga rada je MATLAB 7.10.0 (R2010a), što je jedna od novijih verzija (najnovija je R2011a, dostupna od travnja 2011. g.).



Slika 5.1: Radni prozor MATLAB programske okoline

5.1. Implementacija frekvencijske modulacije

Za početak ćemo krenuti sa metodom FM sinteze. U poglavlju 3.1 smo se osvrnuli na osnovne elemente koji su potrebni pri dizajnu FM instrumenta, a sada ćemo detaljnije promotriti kako programski napraviti funkcije tih elemenata.

5.1.1. Generator valnog oblika

Osnovna jedinica, koja čini polazište za konstrukciju FM signala jest *generator valnog oblika*. Općenito, pri konstrukciji analognog sintesajzera, dizajner mora na neki način definirati izgradnju osnovnih valnih oblika poput sinusoida ili kosinusoida kojima će dalje manipulirati kako bi dobio željeni odziv. Pri tome se koristi generatorskim jedinicama poput *oscilatora* koji svojim analognim sklopovljem generiraju željene harmoničke titraje.

U svijetu digitalne obrade signala, generatori valnog oblika se mogu implementirati na nekoliko načina, navest ćemo tri najčešća (Lundkvist i Qvarnström, 2009):

1. ***Aproksimacija valnog oblika polinomom***

   – signal se generira pomoću nekog matematičkog polinoma, kao npr. pomoću Taylorovog reda:

   $\sin(\varphi) = \varphi - \frac{1}{3!}\varphi^3 + \frac{1}{5!}\varphi^5 - \frac{1}{7!}\varphi^7 + ...$

   $\cos(\varphi) = 1 - \frac{1}{2!}\varphi^2 + \frac{1}{4!}\varphi^4 - \frac{1}{6!}\varphi^6 + ...$

2. ***Generiranje valnog oblika pomoću rezonatorskog filtra***

   – ako uzmemo filtar sa beskonačnim impulsnim odzivom (IIR filtar), te mu polove namjestimo da se nalaze na jediničnoj kružnici z–ravnine, filtar će se nalaziti na rubu stabilnosti i na svom izlazu će generirati val sinusnog ili kosinusnog oblika

3. ***Potraživanje uzoraka u valnim tablicama***

   – u tablicu od konačnog broja uzoraka pospremimo jednu ili dvije periode valnog oblika, te metodom potraživanja uzoraka valne tablice po potrebi generiramo valni oblik željene frekvencije

Najčešće korištena, a ujedno možda i najbrža metoda generiranja valnog oblika je metoda potraživanja u valnoj tablici, koju ćemo iskoristiti u okviru ovoga rada.

Kreiranje valnih tablica

U MATLAB-u je relativno jednostavno stvoriti valnu tablicu. Sve što je potrebno je uzeti idealni sinusni/kosinusni valni oblik i odsjeći jedan njegov period kojeg spremamo u tablicu. Za naše potrebe je sasvim dovoljno napraviti tablicu veličine $N = 4096\ uzoraka$ otipkanih frekvencijom uzorkovanja $f_s = 44.1\ kHz$. Naredbe su slijedeće:

```
% vektor vremena za tablicu veličine 4096 uzoraka:
t = 0:1/44100:0.0928789;   % 4096/44100 = 0.0928798,

% generiranje sinusa od 10.7666 Hz:
sinus = sin(2*pi*t*10.7666);   % 44100/4096 = 10.7666,

% pospremanje tablice u sinus.mat datoteku:
save 'E:\...direktorij...\sinus.mat' sinus;
% direktorij je proizvoljna datoteka na hard disku (path)
```

Na ovaj način smo u `sinus.mat` datoteku pohranili jedan period sinusnog valnog oblika veličine 4096 uzoraka i frekvencije 10.7666 Hz. Sličan postupak koristimo i za kosinus funkciju, s time da uzorke pospremamo u `kosinus.mat` datoteku.

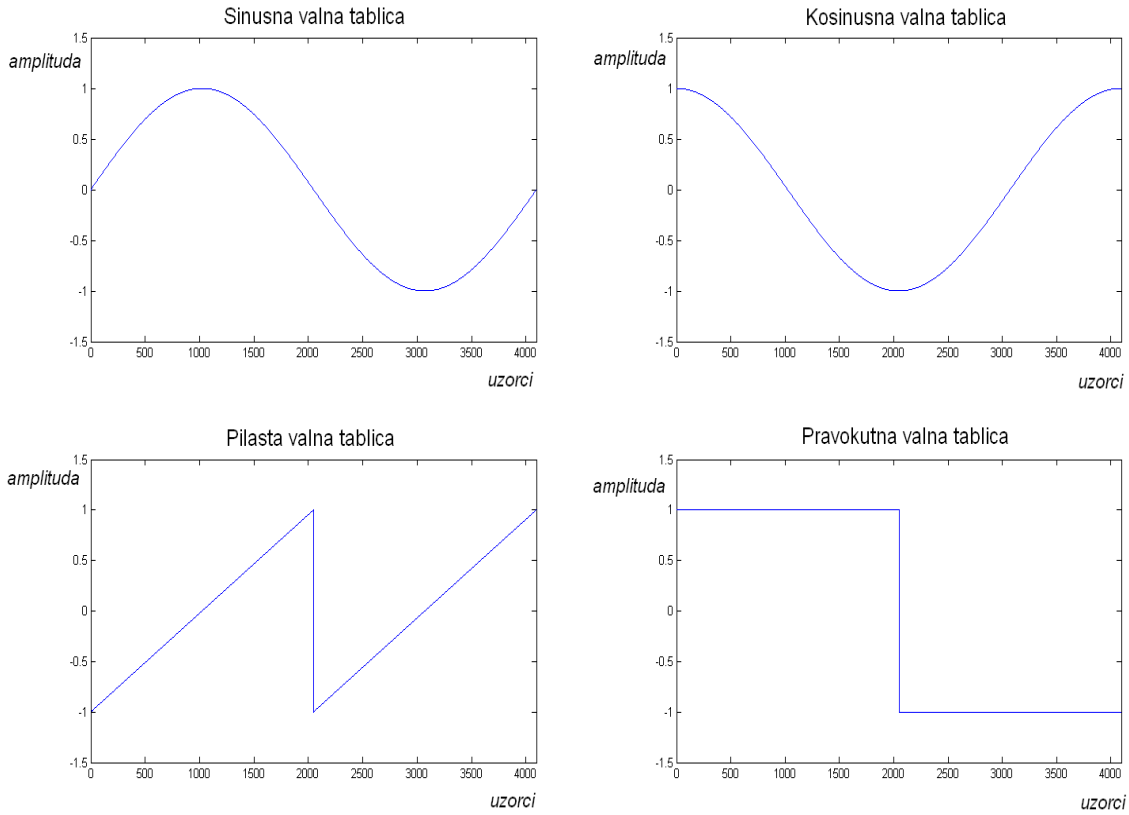
Kao još dva dodatna valna oblika, koji mogu zamijeniti osnovne sinusne/kosinusne funkcije, uzet ćemo pilasti i pravokutni signal, te ih spremiti na ekvivalentan način u `pilasti.mat` te `pravokutni.mat` datoteke, koristeći pritom MATLAB–ove funkcije `sawtooth()` i `square()` (detaljniji opis ovih funkcija može se pronaći u MATLAB–ovom help-u).

Ovime smo napravili četiri valne tablice koje ćemo rabiti za generiranje valnih oblika bilo kojih frekvencija. Pojedina valna tablica se, prije upotrebe, mora učitati u dio programa koji će ju koristiti, a to se radi pomoću `load` naredbe.

Na primjer: `load sinus; load kosinus; ...`

Za potrebe frekvencijske modulacije nećemo koristiti pilasti i pravokutni valni oblik, jer će nam za naš model instrumenta biti dovoljan samo sinusni valni oblik, no ako se promijeni korišteni valni oblik, mogu se dobiti zanimljive izmjene u rezultirajućem izlaznom signalu.

Slijedeća slika prikazuje sva četiri valna oblika koja smo prethodnim naredbama spremili u tablice:



Slika 5.2: Signali valnih tablica

Potraživanje uzoraka u valnoj tablici

U našim tablicama imamo pohranjenih 4096 uzoraka jednog perioda valnog oblika fiksne frekvencije. Ako bi htjeli generirati signal neke druge frekvencije morati ćemo se poslužiti tehnikom cirkularnog potraživanja uzoraka.

Uzmimo, na primjer, sinusnu valu tablicu. Svaki uzorak tablice možemo predstaviti slijedećom jednadžbom (Kuo i Lee, 2001):

$$tablica(n) = \sin\left(\frac{2\pi n}{N}\right), \qquad n = 0, 1, 2, ..., N-1. \tag{5.1}$$

Generiranje valnog oblika se sastoji u tome da potražujemo uzorke u tablici pomoću inkrementa *inkr*, cirkularno se vraćajući na početak tablice kada god prijeđemo broj uzoraka tablice *N*. Inkrement je konstantna vrijednost pomoću koje preskačemo određeni broj uzoraka u tablici te na taj način uzimamo samo uzorke koji su višekratnici inkrementa:

$$inkr = \frac{N \cdot f}{f_s} \tag{5.2}$$

Kao što vidimo iz izraza 5.2, inkrement ovisi o broju uzoraka pohranjenih u valnu tablicu N, željenoj izlaznoj frekvenciji signala f te frekvenciji uzorkovanja f_s. Dakle, da bi generirali sinusni signal broja uzoraka L, upotrijebit ćemo cirkularni pokazivač $indeks$ koji računamo poput:

$$indeks = mod_N(m + l \cdot inkr) \tag{5.3}$$

gdje l predstavlja izlazni uzorak $l = 1, 2, ..., L$, a m je početna faza signala. Pokazivač $indeks$ je, dakle, ostatak dijeljenja izraza $(m + l \cdot inkr)$ sa veličinom tablice N.

Primjećujemo da $indeks$ ne mora nužno biti cijeli broj, već je često puta realni broj sa cijelobrojnim i frakcionalnim dijelom. U tim slučajevima, pokazivač će pokazivati na neku vrijednost između dvaju susjednih uzoraka tablice. Pošto samo možemo dohvatiti cijelobrojne uzorke, jedno riješenje je da se pokazivač $indeks$ zaokruži na prvu cijelobrojnu vrijednost, no ovo riješenje baš i nije najbolje zbog toga što time unosimo pogrešku u naš izlazni signal, koja se često naziva i ***šum potraživanja valne tablice***.

Bolje riješenje je da na neki način pomoću dva susjedna uzorka tablice, između kojih padne pokazivač, izračunamo pravu vrijednost traženog uzorka. To možemo napraviti tako da upotrijebimo *linearnu interpolaciju* tih dvaju uzoraka.

Linearna interpolacija znači da pretpostavljamo da vrijednost našega uzorka leži na ravnoj liniji koja spaja dva susjedna uzorka valne tablice. Ako cijelobrojnu vrijednost pokazivača označimo sa $clbr$, a frakcionalni dio sa $frakc$, do našeg uzorka dolazimo pomoću izraza (Kuo i Lee, 2001):

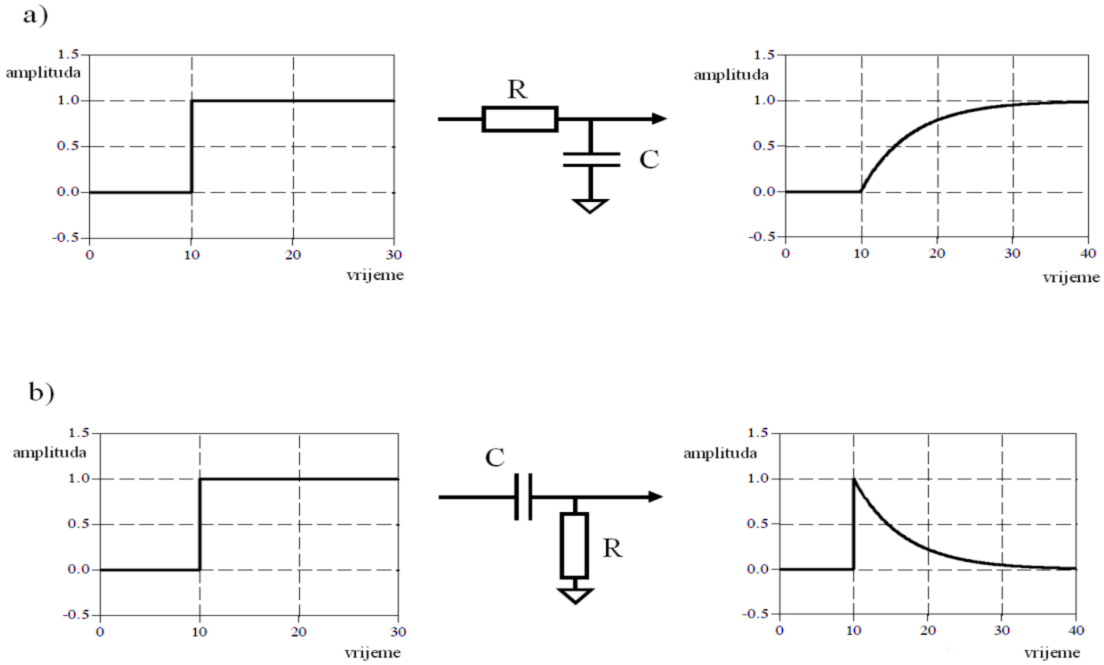
$$izlaz(n) = tablica(clbr) + frakc \cdot [tablica(clbr + 1) - tablica(clbr)] \tag{5.4}$$

gdje $[tablica(clbr + 1) - tablica(clbr)]$ predstavlja nagib ravne linije koja spaja dva susjedna uzorka valne tablice.

5.1.2. Generator ADSR ovojnice

Kod sinteze instrumenata frekvencijskom modulacijom, jako bitan element pri dizajniranju signala je njegova ovojnica. Ovojnica je odgovorna za oblikovanje ne samo amplitude signala, već i njegovog spektra, pa je zbog toga primjenjujemo i na nosilac i na modulator.

Najjednostavniji sklop kojim možemo oblikovati ovojnice jest *digitalni rekurzivni filtar*. To je filtar sa beskonačnim impulsnim odzivom (***IIR***) koji se temelji na jednostavnim analognim sklopovima poput RC ili CR sklopa koji se sastoje od samo jednog para otpornik–kondenzator (Slika 5.3) (Smith, 1997).



Slika 5.3: Analogni rekurzivni filtar: a) Nisko–propusni, b) Visoko–propusni

Ulaz u rekurzivni filtar je step funkcija. Općenita jednadžba koja opisuje digitalni rekurzivni filtar n–tog reda je slijedeća:

$$y[i] = a_0x[i] + a_1x[i-1] + a_2x[i-2] + a_3x[i-3] + ... + a_nx[i-n] \qquad (5.5)$$
$$+ b_1y[i-1] + b_2y[i-2] + b_3y[i-3] + ... + b_ny[i-n]$$

Koeficijenti $a_0, a_1, a_2, a_3, ..., a_n, b_1, b_2, b_3, ..., b_n$ se nazivaju ***rekurzivni koeficijenti*** i određuju kakvog oblika će biti izlaz iz filtra. Pomoću ovih koeficijenata oblikujemo svih četiri stadija ADSR ovojnice.

Za naše potrebe je dovoljno da konstruiramo filtar prvoga reda sa koeficijentima a_0, b_1. Njihove vrijednosti možemo izračunati pomoću slijedećih izraza:

$$a_0 = \beta, \qquad b_1 = 1 - \beta \qquad (5.6)$$
$$\beta = e^{-1/\alpha}, \qquad 0 < \beta < 1 \qquad (5.7)$$

gdje je α broj uzoraka koji određuje trajanje odziva filtra u pojedinom segmentu ADSR ovojnice. Odaberemo li npr. $a_0 = 0.15, b_1 = 0.85$ dobiti ćemo odziv vrlo sličan onome sa slike 5.3 a).

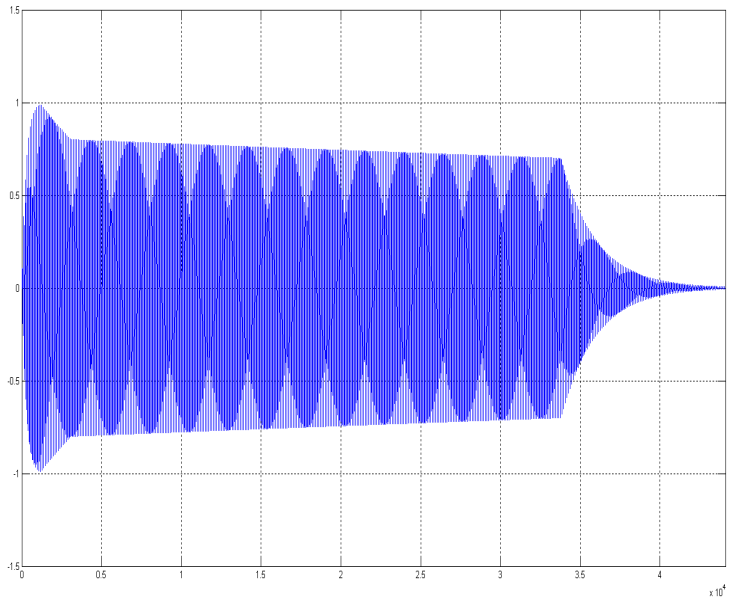
Varijabla β se obično određuje pomoću vremenske konstante filtra prema jednadžbi 5.7. Baš kao što za analogni filtar računamo vremensku konstantu $\tau = R \cdot C$, koja označava broj sekundi koje trebaju filtru da padne na 36.8% početne vrijednosti, isto tako definiramo i vremensku konstantu α digitalnog filtra.

Izraz 5.6 nam može poslužiti pri dizajniranju attack segmenta ADSR ovojnice, no za konstruiranje ostalih segmenata moramo dobiti padajući odziv kao za visoko–propusni rekurzivni filtar sa slike 5.3 b). Koeficjente filtra ćemo u ovom slučaju odabrati ovako:

$$a_0 = (1 + \beta)/2, \qquad a_1 = -(1 + \beta)/2, \qquad b_1 = 1 - \beta \tag{5.8}$$

pri čemu β računamo kao i prije prema izrazu 5.7.

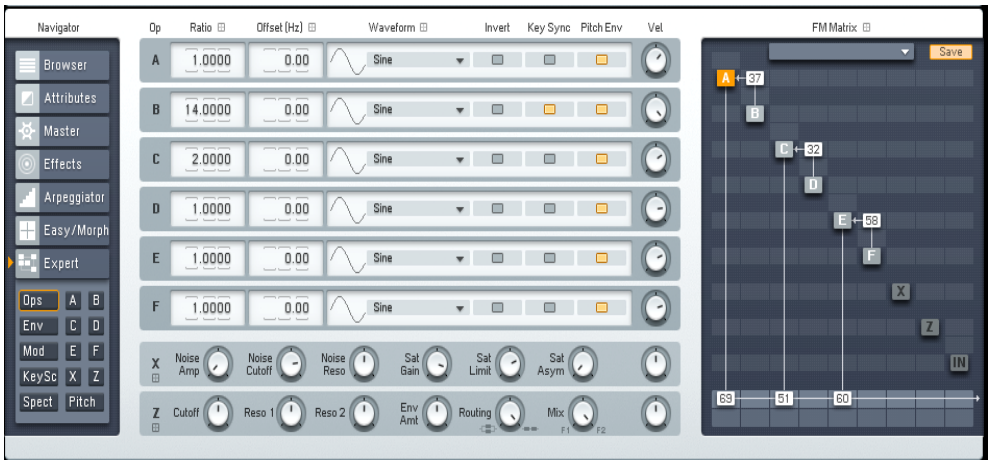
Sada smo u mogućnosti konstruirati sva četiri segmenta ADSR ovojnice. Svaki signal koji provučemo kroz nju biti će modificiran prema odabranim parametrima a_0, a_1, b_1, koje vrlo lako možemo i mijenjati te na taj način namjestiti npr. dulji attack segment, kraći release...itd. Programski kod, napisan u MATLAB-u, koji generira ADSR ovojnicu, nalazi se na kraju ovog rada, u dodatku A.5



Slika 5.4: Primjer generirane ADSR ovojnice u MATLAB-u

5.1.3. Dizajn modela FM instrumenta

Instrument koji ćemo pokušati modelirati pomoću FM sinteze biti će digitalni Rhodes klavir iz Yamahinog DX7 sintesajzera. Kao referentni model poslužit će nam njegova emulacija iz Native Instruments FM8 programa:



Slika 5.5: Emulacija DX7 Rhodes instrumenta

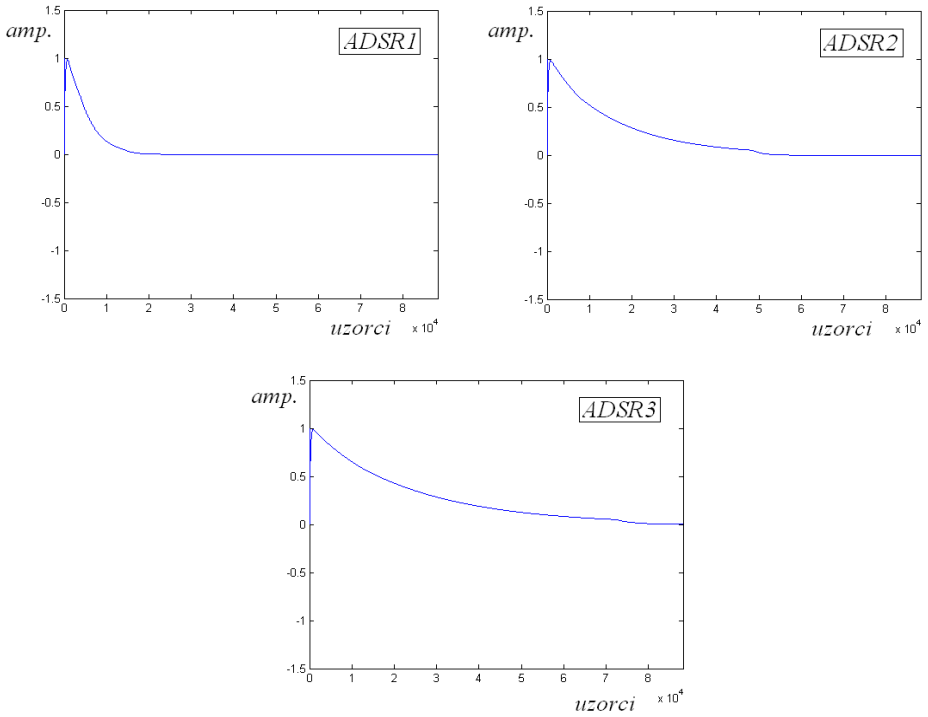
Sa slike 5.5 možemo vidjeti da su za model instrumenta korišteni 6 signala: A, C, E kao nosioci te B, D, F kao modulatori, nakon čega se modulirani nosioci na kraju zbrajaju u jedan signal koji predstavlja rezultat FM sinteze.

Omjeri frekvencija C:M su slijedeći: za prvi par modulator/nosilac omjer je 1:14, za drugi par je 2:1, te za treći 1:1. Svi signali su sinusoidalni bez međusobnog faznog pomaka te je svaki oblikovan svojom ovojnicom.

Ove parametre ćemo primjeniti i u našem programu, uz jednu malu razliku a to je da nećemo koristiti zasebnu ovojnicu za svaki signal, već ćemo dizajnirati 3 glavne ovojnice te ćemo kroz svaku provući po dva signala kako bi uštedjeli na proračunu.

Dakle, prvo što trebamo napraviti je generirati modulatore, jer nam oni služe za moduliranje frekvencije nosilaca, a pri tome se koristimo postupkom pretraživanja valnih tablica opisanim u poglavlju 5.1.1. Nakon toga, potrebno je svaki modulator provući kroz ovojnicu jer ona određuje zakon po kojem se mijenja jačina modulacije nosioca. Ako bi, na primjer imali ravnu ovojnicu modulatora, to bi značilo da će spektar konačnog FM signala biti jednolik kroz cijelo svoje trajanje, što nam ne daje zanimljive rezultate.

Pri dizajnu instrumenta, konstruirane su tri ADSR ovojnice koje se mogu vidjeti na slijedećoj slici:



Slika 5.6: Ovojnice FM nosilaca i modulatora

Ovakvi oblici ovojnica, primjenjeni na modulatore, određuju količinu modulacije kojom moduliraju nosioce. Vidimo da će u attack segmentima modulacija biti najizraženija, dok se u ostalim segmentima postepeno smanjuje. Kada se ovako oblikovani modulatori sada primjene na nosioce, rezultat će biti signal sa vremenski promjenjivim spektrom, što nam je bio i cilj postići.

Kod generiranja nosilaca moramo na neki način uvesti utjecaj modulatora na promjenu frekvencije. Pošto sve signale generiramo potraživanjem uzoraka u valnoj tablici, najlogičnije bi bilo da kod izračuna inkrementa nosioca pridodajemo uzorke modulatora pomnoženog sa indeksom modulacije β. Pri tome možemo napisati slijedeći programski kod:

```
for i = 1:trajanje
    % izračun inkrementa nosioca:
    inkrNos = (L*fc/Fs + beta*modulator(i));

    % indeksi za nosilac
    indeksNos = mod(prethIndeksNos + inkrNos, L);
    prethIndeksNos = indeksNos;
    indNos = floor(indeksNos);
```

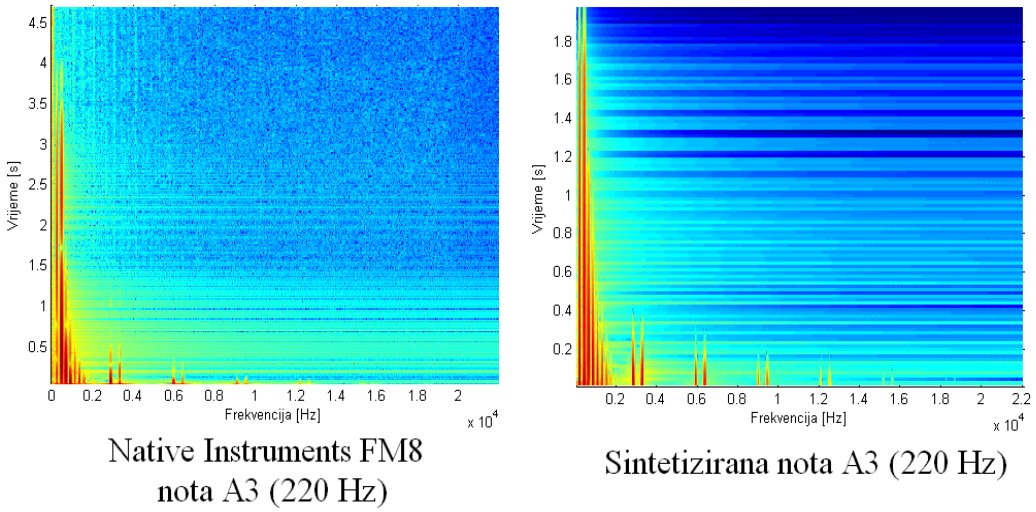
```
        if indNos ~=0
            % linearna interpolacija:
            nos(i) = sinus(indNos) + (indeksNos - indNos)*...
                        (sinus(indNos + 1) - sinus(indNos));

        elseif indNos == 0
            prethIndeksNos = 0;
        end
    end
```

Kao što vidimo, signal se konstruira uzorak po uzorak i svakim prolaskom kroz petlju vrijednost inkrementa se mijenja po zakonu koji određuju modulator i frekvencija nosioca. Na ovaj način se indeks potraživanja u valnoj tablici mijenja u varijabilnim intervalima što uzrokuje modulaciju frekvencije. Naravno, o indeksu modulacije β ovisi jačina modulacije pa ćemo zato odabrati vrijednosti za svaki modulator redom: $\beta_1 = 10, \beta_2 = 2, \beta_3 = 0.75$.

Na kraju, kada su ispravno generirana sva tri nosioca, još ih samo treba provući kroz ovojnice, zbrojiti u konačni FM signal i to je to – sinteza signala je dovršena.

Radi usporedbe, prikazati ćemo spektrograme nasumično odabrane sintetizirane note, te note koja je generirana pomoću Native Instruments FM8 programa. Na slici 5.7 uočavamo kako su spektri signala dosta slični, sa samo jednom razlikom, a to je da naša sintetizirana nota ima kraće trajanje, što je napravljeno iz razloga da sinteza note bude brža. Rezultati su i slušno zadovoljavajući, te je razlika između ovih dvaju signala tek nezamjetno uočljiva.



Slika 5.7: Usporedba sintetizirane note sa notom emulatora FM8

Programski kod, koji implementira sintezu frekvencijskom modulacijom može se pronaći u dodacima A.3-A.5 na kraju ovoga rada.

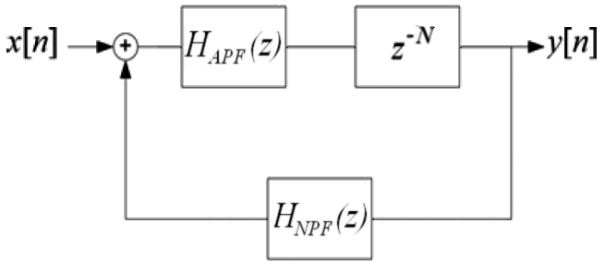
5.2. Implementacija Karplus–Strong modela

U poglavlju 3.2.1 smo vidjeli kako se jednostavno može implementirati model trznute žice prema osnovnim fizikalnim načelima. Programsko riješenje ovog modela je također vrlo jednostavno i daje poprilično dobre rezultate. U MATLAB-u je napravljena funkcija `Karplus-Strong.m` čiji poziv je slijedećeg oblika:

```
function nota = Karplus_Strong(broj_note, NoteOffTime)
```

Funkcija prima MIDI broj note kao argument te vrijeme kada prelazi u *release* fazu, a vraća sintetizirane uzorke u polju `nota`.

Prvo što je potrebno napraviti jest pravilna inicijalizacija parametara. Prisjetimo se da model kojeg pokušavamo izgraditi ima elemente kao na slici 5.8:



Slika 5.8: Implementacija Karplus–Strong modela

Na slici 5.8 vidimo da je dodan jedan novi element, a to je sve-propusni filtar prvoga reda $H_{APF}(z)$. Njegova funkcija je da unese mogućnost frakcionalnog kašnjenja, tj. da linija kašnjenja reda N realizira bilo koju vrijednost kašnjenja . Time se postiže bolja uštimanost instrumenta, odnosno frekvencije koje program proizvodi su točnije.

Dakle, inicijalizacija parametara se sastoji u tome da treba odrediti duljinu linije za kašnjenje *N* pomoću frekvencije koju algoritam treba proizvesti, te je potrebno inicijalizirati pobudni signal, odnosno u našem slučaju šum.

Da bi algoritam brže radio, potrebno je unaprijed odrediti i veličinu polja izlaznog signala kako program ne bi morao tijekom izvršavanja konstantno povećavati polje u koje sprema sintetizirane uzorke. Radi istog razloga i polje linije kašnjenja također treba biti unaprijed alocirano.

Nakon inicijalizacjie parametara potrebno je krenuti u konstrukciju digitalnog nisko-propusnog (NP) filtra, koji se nalazi u petlji povratne veze na slici 5.8, sve-propusnog filtra, te linije kašnjenja. Spomenuti NP filtar prigušuje visoke harmonike svakim prolaskom kroz petlju te na taj način poboljšava realističnost zvuka modeliranog instrumenta. Ako ovaj filtar implementiramo prema jednadžbi 3.16 iz poglavlja 3.2.2, rezultati će biti zadovoljavajući, ali ako primjenimo malo drugačiji filtar možemo dobiti bolji odziv te će nam model zvučati prirodnije i bogatije.

5.2.1. Nisko-propusni filtar za prigušivanje harmonika

Jednu dosta intuitivnu implementaciju nisko-propusnog filtra, koji modelira tijelo gitare, su 1993. godine predstavili M.Karjalainen, V.Välimäki i Z. Jánosy svojim radom na internacionalnoj konferenciji računalne glazbe u Japanu (M.Karjalainen et al., 1993).

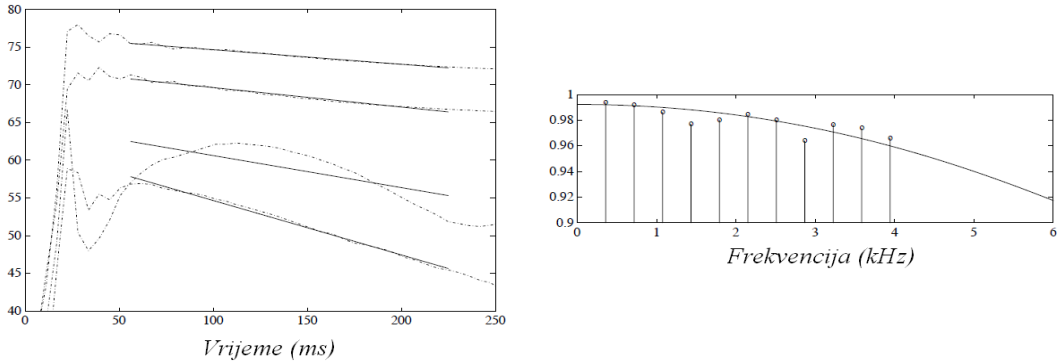
Koraci algoritma, pomoću kojih se konstruira digitalni NP filtar, su slijedeći:

1. Izračunaj kratko-vremensku Fourierovu transformaciju (STFT) iz snimljene note gitare
2. Izmjeri amplitudu svakog harmonika iz svih STFT okvira i formiraj ovojnicu za svaki harmonik
3. Umetni ravni pravac u logaritamskoj skali na svaku ovojnicu harmonika, koji predstavlja nagib pada harmonika
4. Izračunaj pojačanje petlje za svaki nagib pravca
5. Dizajniraj digitalni filtar koji će pratiti prethodno izračunata pojačanja na različitim frekvencijama

Ovim postupkom se aproksimira digitalni IIR filtar prvoga reda koji modelira tijelo gitare te na taj način prirodnije prigušuje harmonike sintetiziranog signala. Na slici 5.9 su prikazani pravci koji prate vremenski pad harmonika s lijeve strane, te aproksimacija amplitudno-frekvencijske karakteristike filtra sa desne strane.

Prijenosna funkcija ovoga filtra je slijedeća (M.Karjalainen et al., 1993):

$$H_{NPF}(z) = \frac{0.9152 + 0.1889z^{-1}}{1 + 0.1127z^{-1}} \tag{5.9}$$



Slika 5.9: Vremenske ovojnice četiri najniža harmonika (slika a)), te amplitudno-frekvencijska karakteristika IIR filtra prvoga reda (slika b)).

5.2.2. Lagrange-ov interpolator

Ranije smo napomenuli kako je za dobro uštimani Karplus-Strong model potrebno dodati filtar koji će omogućiti frakcionalno kašnjenje delay linije. Naime, problem je sličan kao kod potraživanja uzoraka u valnim tablicama, koje smo koristili u implementaciji FM sinteze. Kod računanja duljine linije kašnjenja *N*, gdje *N* računamo kao `N = floor(Fs/f_osn)`, uvijek ćemo dobiti cijelobrojnu vrijednost, međutim to nije precizno zbog toga što takav račun ne daje u potpunosti točne izlazne frekvencije nota.

Jedno od mogućih riješenja je u seriju sa delay linijom umetnuti tzv. *Lagrange-ov interpolator* koji predstavlja jednostavni sve-propusni digitalni filtar prvoga reda. Njegova prijenosna funkcija je slijedeća (Doering, 2008):

$$H_{APF}(z) = \frac{D + z^{-1}}{1 + Dz^{-1}} \tag{5.10}$$

gdje je varijabla *D* frakcionalno kašnjenje koje se računa pomoću izraza 5.11:

$$D = \frac{f_s - f_0(N+1)}{f_0(N-1) - f_s}, \qquad |D| < 1 \tag{5.11}$$

5.2.3. Računanje uzoraka sintetiziranog signala

Sada imamo sve potrebne elemente za izgradnju modela, sve što trebamo napraviti je pronaći prijenosnu funkciju cijelokupnog sustava sa slike 5.8.

Ukupna prijenosna funkcija modela može se napisati pomoću slijedećih izraza:

$$T(z) = \frac{Y(z)}{X(z)} = \frac{H_{APF}(z)z^{-N}}{1 - H_{NPF}(z)H_{APF}(z)z^{-N}} \tag{5.12}$$

$$T(z) = \frac{Dz^{-N} + (1+CD)z^{-(N+1)} + Cz^{-(N+2)}}{1 + (C+D)z^{-1} + CDz^{-2} - ADz^{-N} - (A+BD)z^{-(N+1)} - Bz^{-(N+2)}} \tag{5.13}$$

gdje koeficijenti *A, B, C* predstavljaju redom: $A = 0.9152, B = 0.1889, C = 0.1127$, dok je D već spomenuto frakcionalno kašnjenje iz izraza 5.11.

Kako bi programski mogli izvesti jednadžbu 5.13, potrebno ju je raspisati pomoću jednadžbi diferencija:

$$\begin{aligned} y(n) = D \cdot x\big(n-N\big) + (1+CD) \cdot x\big(n-(N+1)\big) + \\ + C \cdot x\big(n-(N+2)\big) - (C+D) \cdot y\big(n-1\big) - \\ - CD \cdot y\big(n-2\big) + AD \cdot y\big(n-N\big) + \\ + (A+BD) \cdot y\big(n-(N+1)\big) + B \cdot y\big(n-(N+2)\big) \end{aligned} \tag{5.14}$$

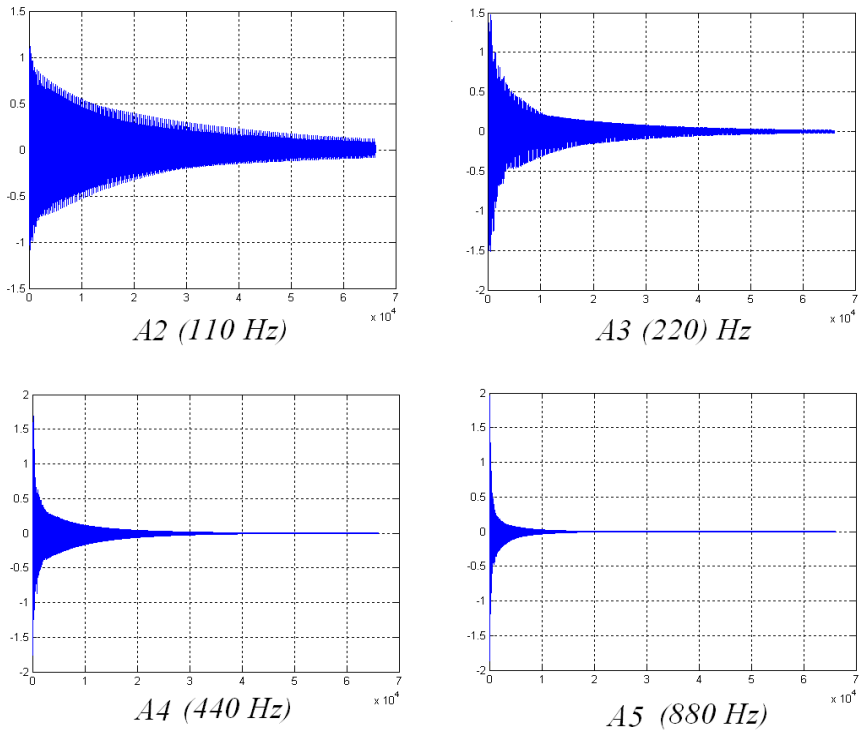
gdje dio izraza 5.14: $D \cdot x\big(n-N\big) + (1+CD) \cdot x\big(n-(N+1)\big) + C \cdot x\big(n-(N+2)\big)$ predstavlja šum.

Pomoću izraza 5.14 signal se konstruira uzorak po uzorak. Nakon što se izračuna jedan uzorak, potrebno je ući u liniju za kašnjenje i pospremiti trenutni uzorak na prvu poziciju polja linije, a sve ostale uzorke pomaknuti za jedno mjesto unutar linije. Proces je potrebno ponavljati sve dok se ne izračunaju svi uzorci izlaznog signala.

Zvuk koji se dobiva prethodno opisanim postupkom zvuči poprilično realistično. Kod svakog pokretanja programa, polje `sluc`, koje se puni slučajnim uzorcima, napunit će se sa drugačijim skupom uzoraka i zbog toga svaki put imamo nešto drugačiji pobudni signal, pa analogno tome i sintetizirani signal zvuči nešto drugačije kod svakog pokretanja programa.

Na slijedećoj slici (5.10) možemo vidjeti par sintetiziranih nota, koje su dobivene pomoću prethodno opisanog algoritma. Možemo primjetiti kako, za više frekvencije, nota dosta brzo utihne, što je direktna posljedica kraće delay linije za visoke frekvencije.

Potpuni programski kod za implementaciju opisanog modela Karplus-Strong algoritma može se pronaći u dodatku A.6



Slika 5.10: Sintetizirane note gitare pomoću Karplus-Strong modela

5.3. Implementacija Wavetable sinteze

Kod programske implementacije frekvencijski moduliranog signala, iz poglavlja 5.1, opisali smo postupak kojim se generiraju razni valni oblici signala nosilaca i modulatora. Snimljene uzorke tih valnih oblika smo iskorištavali za stvaranje signala proizvoljnih frekvencija te na taj način dobivali digitalne oscilatore koje smo iskoristili za konstrukciju FM signala.

Metoda wavetable sinteze iskorištava mogućnost spremanja uzoraka signala u tablice, te se služi raznim postupcima interpolacije za dohvat pojedinih uzoraka. Ako u tablicu pospremimo čitavu snimljenu notu nekog instrumenta, sinteza ostalih nota se svodi samo na pretraživanje tablice i interpolaciju uzoraka.

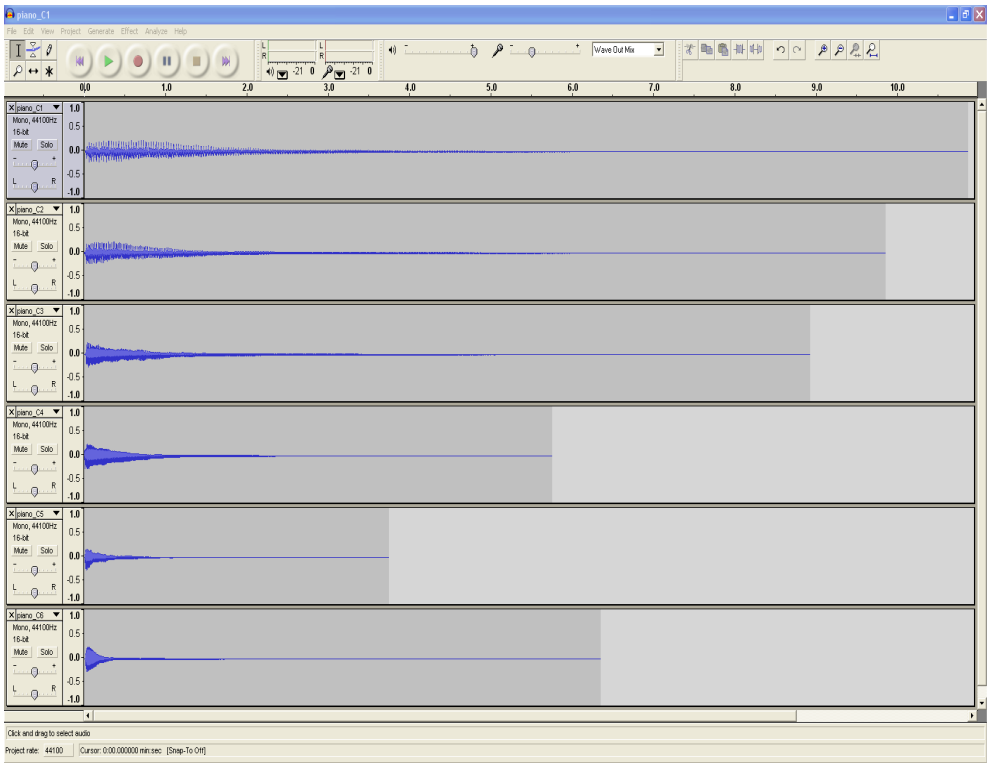
5.3.1. Snimanje uzoraka

U okviru ovoga rada, za snimanje nota instrumenta, poslužiti ćemo se programom ***Audacity (ver. 1.2.6)*** te sintesajzerom *Yamaha PSR-520*. Instrument koji ćemo snimiti u naše valne tablice je ”Grand piano” i to sa odabranom frekvencijom uzorkovanja $f_s = 44.1\ kHz$, te kvantizacijom od $n = 16\ bitova$.

Pošto je klavir dosta kompleksan instrument kojem je spektar signala poprilično vremenski promjenjiv, odabrat ćemo 16 nota koje ćemo snimiti u njihovoj cijelosti tako da imamo pohranjene sve faze signala: attack, decay, sustain i release. Iz tih snimljenih nota ćemo sintetizirati sve ostale te na taj način pokriti raspon od 88 nota, koliko ih se obično nalazi na jednom klaviru.

Audacity je freeware program koji se besplatno može skinuti sa interneta (Audacity, 2011), a namijenjen je snimanju, reprodukciji, te raznoj obradi audio datoteka kao što su *wav, mp3, ogg vorbis, mid* datoteke. Na slici 5.11 prikazano je tipično sučelje Audacity editora. Za snimanje nota, potrebno je spojiti sintesajzer sa portom mikrofona na računalu pomoću 6.3 mm stereo kabla koji na jednom svom kraju ima pretvarač u 3.5 mm stereo priključak kako bi se mogao spojiti na mikrofonski ulaz računala.

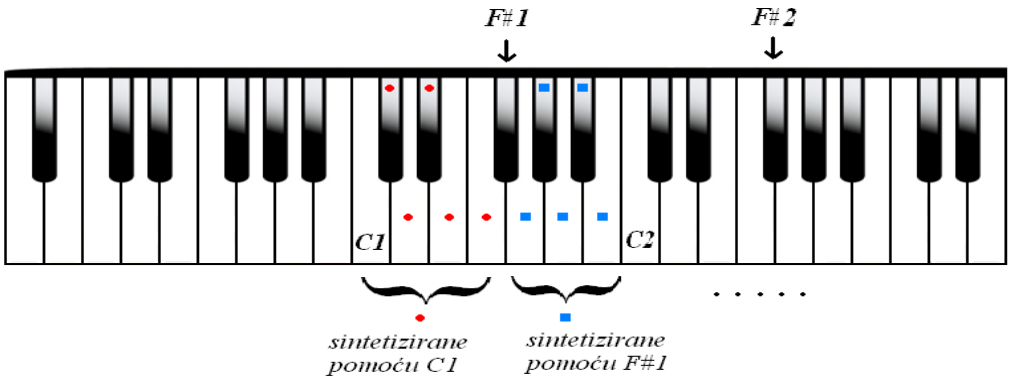
Note klavira koje su snimljene i pospremljene u .wav datoteke na računalu su redom: *C0, F#0, C1, F#1, C2, F#2, C3, F#3, C4, F#4, C5, F#5, C6, F#6, C7, F#7*. Razlog ovakvog odabira je taj da kod sinteze svih preostalih nota, sintetizirana nota zvuči što prirodnije. Naime, karakteristika klavira kao instrumenta, je ta da spektar svake note ima drugačiji vremenski razvoj.



Slika 5.11: Audacity korisničko sučelje

Što su note međusobno udaljenije na klavijaturi, to će im vremenski spektar biti

različitiji, tako da ako želimo sintetizirati notu, moramo se poslužiti nekom od njoj bliskih nota kako sinteza ne bi učinila signal previše umjetnim i sintetičkim, te da bi se zadržala što sličnija spektralna svojstva signala. Iz tog razloga su odabrane note međusobno udaljene 6 polutonova, tako da svaka snimljena nota može "sintetizirati" 5 idućih nota, pri čemu je očuvana realističnost svake note. Ilustracija je dana na slici:



Slika 5.12: Sintetiziranje nota klavira pomoću valnih tablica

Snimljene note su zapisane u stereo obliku u odgovarajuće .wav datoteke, da bi smanjili veličinu tih datoteka možemo pretvoriti stereo signal u mono, te na taj način prepoloviti veličinu datoteka. Naredbe u MATLAB-u kojima se koristimo su slijedeće (primjer za C1 notu):

```
[C1, Fs] = wavread('piano_C1');
C1 = C1(:,1);
C1 = C1';
```

Prva naredba čita datoteku 'piano_C1.wav' i sprema uzorke u polje C1 koje se sastoji od dva stupca (stereo signal), gdje su uzorci signala uzorkovani frekvencijom `Fs`. Druga naredba odbacuje 2 stupac, te na taj način pretvara signal u mono. Treća naredba služi za pretvaranje stupca u redak.

Nakon što identične naredbe primjenimo na ostale snimljene datoteke, sve snimljene note možemo pohraniti u jednu `.mat` datoteku kako bi bile dostupne za dohvat u programu sinteze:

```
% pohranjivanje nota u note_klavira.mat datoteku:
save '...\note_klavira.mat' C* F*
```

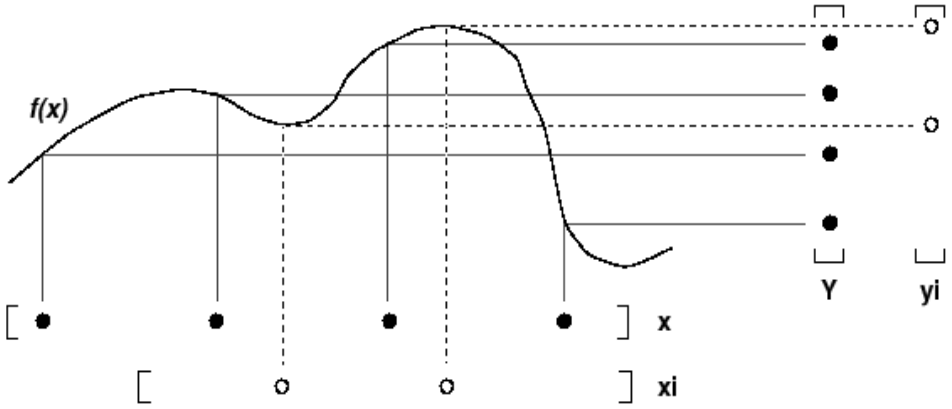
5.3.2. Interpolacija uzoraka

Kako bi dobili ispravnu osnovnu frekvenciju titranja sintetiziranog signala, moramo na neki način interpolirati uzorke signala pomoću uzoraka pohranjenih u valnim tablicama. U MATLAB-u postoji ugrađena funkcija `interp1.m` pomoću koje možemo napraviti interpolaciju. Funkcija je slijedećeg oblika:

```
yi = interp1(x, y, xi, 'method');
```

Funkcija prima argumente *x, y* koji predstavljaju valnu tablicu pomoću koje želimo doći do uzoraka, *xi* su indeksi uzoraka na kojima želimo izračunati vrijednosti *yi* iz tablice, dok *'method'* određuje koju metodu interpolacije želimo (`'nearest'`, `'linear'`, `'spline'`, `'pchip'`, `'cubic'`, `'v5cubic'`). Ako se u pozivu funkcije ne navede metoda, napraviti će se linearna interpolacija uzoraka.

Dakle, imamo na raspolaganju 16 valnih tablica, od kojih je svaka određena sa svojim *x, y* vrijednostima. Ono što želimo jest naći vrijednosti uzoraka *yi* na indeksima *xi* valne tablice, kao što je prikazano na slici 5.13:



Slika 5.13: Interpolacija uzoraka pomoću `interp1.m`

Vrijednost indeksa *xi* ovisi o tome koju notu želimo sintetizirati te se računa pomoću $xi = i \cdot 2^{p/12},\ i = 1, 2, 3, ...$, gdje je *p* broj polutonova udaljenih od note koja je snimljena u tablicu. Dakle, *xi* nije ništa drugo nego skup indeksa na kojima računamo interpolirane vrijednosti iz valne tablice.

Na primjer, ako želimo izračunati interpolirane uzorke note D4, pozvati ćemo valnu tablicu sa snimljenom notom C4. Možemo se koristiti slijedećim naredbama u MATLAB-ovom sučelju:

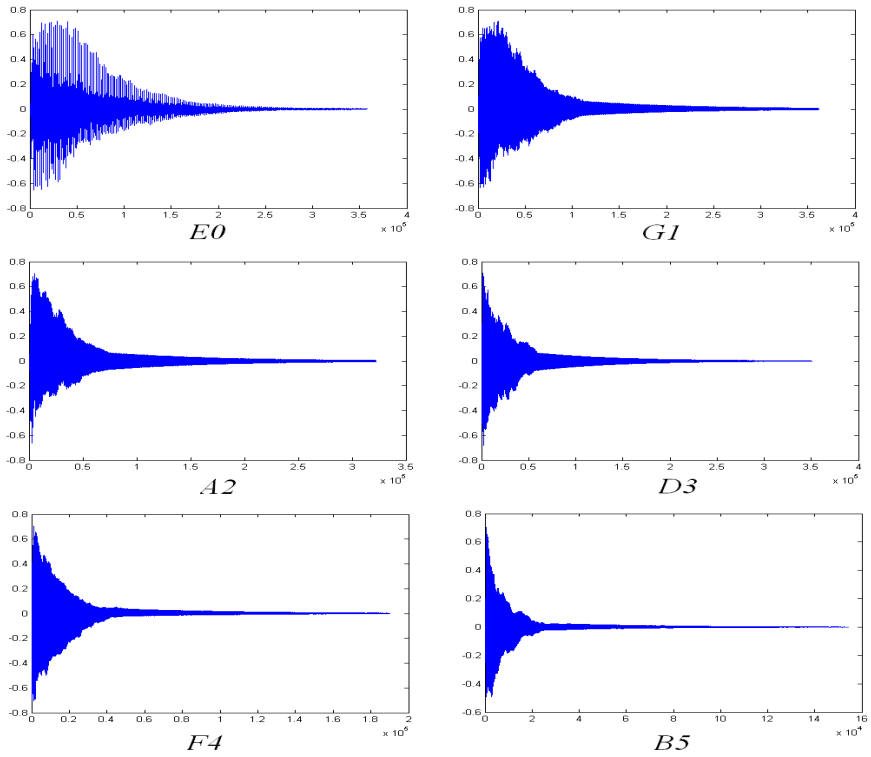
```
nota = C4;
omjer = 2^(2/12);
L = length(nota);
xi = omjer:omjer:L;
snd = interp1(nota, xi);
```

pri čemu se u pozivu funkcije `interp1` navode samo amplitudne vrijednosti polja `note`, te vrijednosti indeksa `xi` na kojima se trebaju interpolirati uzorci.

Raspon nota koje program pokriva je od D#0 (MIDI broj = 15) pa sve do B7 (MIDI broj = 107), što ukupno pokriva 92 note. Funkcija koja izvršava programski kod je slijedećeg oblika:

```
function snd = wavetable(broj_note, NoteOffTime)
```

Ulazni argumenti su MIDI broj note te vrijeme NOTE_OFF događaja (u sekundama), a funkcija vraća sintetizirane uzorke note u polju `snd`. Kao ilustrativni primjer, na slijedećoj slici ćemo prikazati nekoliko nota sintetiziranih ovom funkcijom:



Slika 5.14: Sintetizirane note pomoću wavetable sinteze

Cijelokupni programski kod za funkciju wavetable sinteze se može pronaći u dodatku A.7 ovoga rada.

5.4. Glavni program

Do sada smo uglavnom govorili o realizacijama pojedinih metoda sinteze, koji su njihovi sastavni elementi te o karakteristikama signala koji se dobivaju pomoću tih metoda. Pokazali smo i kako je moguće detektirati MIDI poruke koje instrument šalje, te kako ih je moguće interpretirati.

U glavnom MATLAB programu ćemo sve ove dijelove ukomponirati u jednu cjelinu gdje ćemo sve odsvirane note na sintesajzeru, sintetizirati u jedan izlazni valni oblik. Pošto je MATLAB pretežito simulacijski program koji nema mogućnost paralelnog izvođenja takozvanih *threadova*, kao što je to moguće u Javi ili C/C++ programskim jezicima, vrlo je teško postići real-time odzive na stisnute note sintesajzera. Iz toga razloga, napraviti ćemo offline sintezu gdje prvo vršimo prikupljanje nota, a zatim pokrećemo postupak sinteze, pri čemu vodimo računa o svim vremenima paljenja/gašenja pojedinih nota.

Postupak je ukratko slijedeći:

1. Detektiraj i otvori ulazni MIDI port
2. Poveži Prijamnik klasu sa odašiljačem ulaznog MIDI port-a
3. Dohvati MIDI poruke iz buffera
4. Kreiraj matricu svih odsviranih nota
5. Sintetiziraj note

5.4.1. Otvaranje MIDI IN port-a

Da bi bili u mogućnosti iskoristiti prijamnik koji smo opisali i objasnili u poglavlju 4.2, moramo ga na neki način spojiti sa USB port-om preko kojega se šalju MIDI poruke u računalo. U svrhu ovoga problema, opet posežemo za MATLAB-ovom mogućnošću iskorištavanja Java ugradbenih klasa. Konkretno, iskoristiti ćemo Java `MidiSystem` klasu, koja je sposobna detektirati i manipulirati sa raznim MIDI uređajima koji se mogu spojiti na računalo, ili pak onima koji se nalaze u samom računalu. Pri tome se radi o uređajima poput softverskih sintesajzera, sekvencera, te ulazno/izlaznih MIDI

portova. Svi ti uređaji se mogu međusobno spajati i komunicirati pomoću adekvatnih odašiljača i prijamnika koji su sadržani u svakom MIDI uređaju (Oracle, 2011).

`MidiSystem` klasa je sadržana u Java paketu `javax.sound.midi` koji prikuplja sve važne klase za obradu MIDI datoteka i događaja. Neke od najvažnijih klasa, koje su od velike koristi bile i u svrhu ovoga rada, mogu se pronaći u slijedećoj tablici (JavaAPI, 2011):

Tablica 5.1: Najvažnije klase `javax.sound.midi` paketa

Klasa	***Opis***
`MidiDevice`	Glavno sučelje za sve MIDI uređaje
`Receiver`	Prima `MidiEvent` objekte i radi nešto korisno s njima
`Transmitter`	Šalje `MidiEvent` objekte prema jednom ili više `Receiver`-a
`MidiDevice.Info`	Daje pristup raznim informacijama o MIDI uređaju
`MidiEvent`	Objedinjuje MIDI poruku i vremensku štampu poruke
`MidiMessage`	Osnovna klasa za sve MIDI poruke
`MidiSystem`	Omogućuje pristup raznim MIDI uređajima

Pomoću `MidiSystem` klase dolazimo do popisa svih MIDI uređaja koji su dostupni na računalu, i to pomoću njene `getMidiDeviceInfo()` metode. Ako napišemo slijedeću naredbu u MATLAB-u:

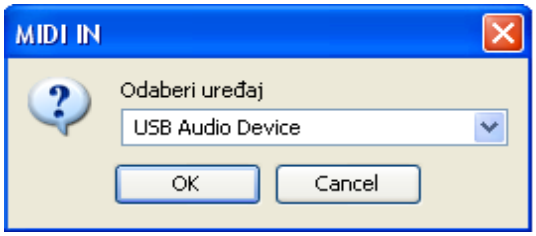
```
infoIn = MidiSystem.getMidiDeviceInfo();
```

u polje `infoIn` će se upisati svi uređaji koje `MidiSystem` klasa pronađe. Ako je u trenutku pokretanja naredbe, na računalo priključen MIDI-USB kabel, klasa će ga prepoznati i uvrstiti u popis dostupnih uređaja.

Nakon što se provjeri dostupnost uređaja, potrebno je preispitati koji od tih uređaja bi mogao biti ulazni MIDI port, jer u polju `infoIn` ta informacija neće biti prikazana. Jedan način, na koji se to može provjeriti je tako da se preispita koji sve uređaji imaju odašiljače, jer ulazni MIDI port mora imati barem jedan odašiljač.

Kada ovako filtriramo sve uređaje, dobivamo suženi krug uređaja, od kojih svaki ima barem jedan odašiljač. U ovom trenutku je najbolja solucija korisniku prikazati te uređaje tako da sam odabere koji od njih će otvoriti, a koji će zanemariti. U tu svrhu je napravljen mali skočni izbornik pomoću kojeg se vrši taj odabir. Primjer izbornika je dan na slici 5.15.

Za konstrukciju ovog jednostavnog izbornika smo se poslužili Java `JOptionPane` klasom iz paketa `javax.swing` koji definira klase potrebne za izgradnju grafičkih



Slika 5.15: Skočni prozor za odabir MIDI IN port-a

korisničkih sučelja (*engl.* GUI - Graphical User Interface). U MATLAB-u nam može poslužiti slijedeća naredba (Cadenhead i Lemay, 2007):

```
odabrani = JOptionPane.showInputDialog([],...
           'Odaberi uređaj', 'MIDI IN',...
           JOptionPane.QUESTION_MESSAGE,...
           [], infoIn, []);
```

Nakon što korisnik odabere uređaj, program automatski prelazi u otvaranje uređaja tako da ga omogući za upotrebu ostatku glavnog programa, gdje se otvaranje vrši pomoću `getMidiDevice()` i `open()` metoda, klase `MidiSystem`.

5.4.2. Komunikacija sa *Prijamnik.class*

Jednom otvoren, ulazni MIDI port je spreman za kontinuirano slanje informacija. Te informacije prikupljamo sa našom `Prijamnik` klasom koju je potrebno spojiti na odašiljač upravo otvorenog ulaznog MIDI port-a. U `MidiSystem` klasi postoje dvije metode za dohvat prijamnika ili odašiljača nekog uređaja, a to su redom: `getTransmitter()` i `getReceiver()` metode. Konkretno u našem slučaju treba povezati odašiljač MIDI ulaznog port-a, te prijamnik koji smo sami konstruirali pomoću `Prijamnik` Java klase.

U MATLAB-u je ovo vrlo jednostavno za izvesti, potrebno je svega par naredbi:

```
spremnik = javaObject('SpremnikMidiPoruka');
odasiljac = ulaz.getTransmitter();
prijamnik = javaObject('Prijamnik', spremnik);
odasiljac.setReceiver(prijamnik);
```

Bitno je za uočiti, da pri stvaranju `prijamnik` Java objekta, ulazni argument `Prijamnik` klase mora biti `spremnik` objekt koji je instanca naše konstruirane klase `SpremnikMidiPoruka`. To je potrebno radi toga da `prijamnik` zna kuda

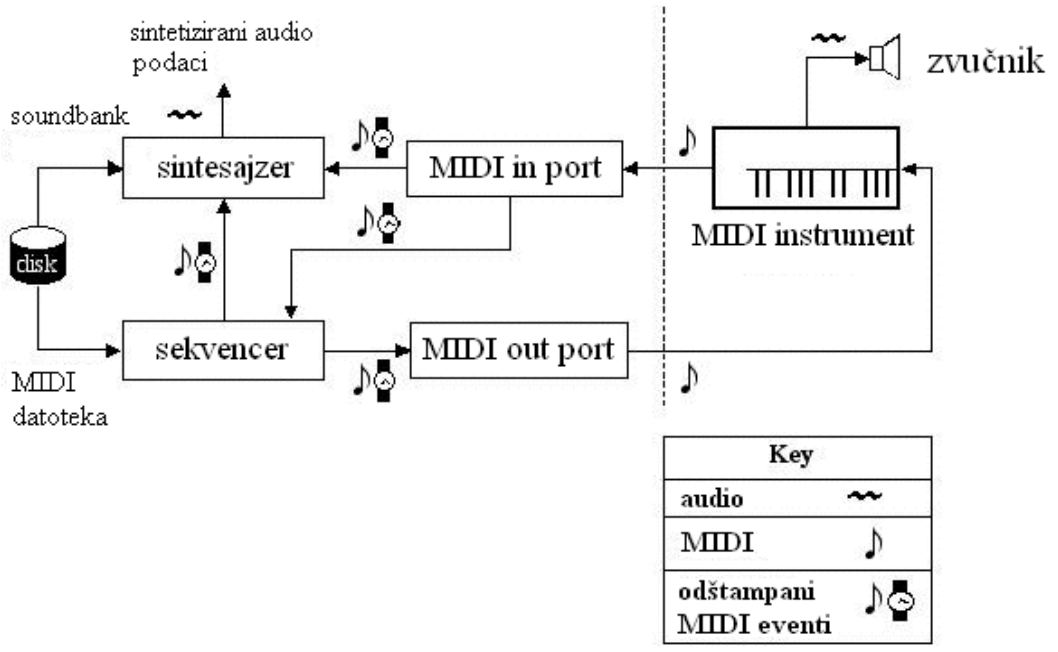
da posprema nadolazeće MIDI evente, kako bi ih se pravilno moglo prirediti za daljnju manipulaciju glavnim programom.

5.4.3. Dohvat MIDI događaja

Ako u ovoj fazi programa počnemo pritiskati tipke sintesajzera, svaka poruka koja stigne na odašiljač ulaznog MIDI port-a biti će proslijeđena u prijamnik te pospremljena u spremnik. Pristigle poruke se spremaju na kraj reda u spremniku onakvim redoslijedom kakvim stižu na prijamnik i iste su precizno vremenski odštampane vremenom izraženim u mikrosekundama koje teče od početka kada se otvorila veza prijamnik-odašiljač.

Ovakvim načinom implementacije programa smo u biti projektirali sekvencer, uređaj kojem je glavna uloga snimanje i pravilno redanje midi evenata kako bi ih se prirediло za kasniju sintetizaciju. Premda se ovdje radi o vrlo primitivnom sekvenceru, funkcionalnost je dovoljna za potrebe ovoga rada, koju je naravno moguće i proširiti, no o tome nešto kasniije.

Na slijedećoj slici (5.16) je u grubo prikazana veza svih dosada spomenutih komponenata koje međusobno komuniciraju u svrhu obrade MIDI podataka (Oracle, 2011).



Slika 5.16: Veza između svih MIDI uređaja

Dakle, komponenta do koje smo trenutno došli je sekvencer. Pretpostavimo na trenutak da nam se spremnik kontinuirano puni MIDI event-ima, što je slijedeće što

moramo napraviti? Sve dokle god pritišćemo tipke na klavijaturi, spremnik će sve više i više biti popunjen event-ima. Ako ne poduzimamo nikakve radnje za izvlačenje evenata, spremnik će nam se prenatrpati i vjerojatno će doći do smrzavanja cijelog programa. Upravo zbog toga moramo kontinuirano izvlačiti event-e i čitati informacije koje sadržavaju. U MATLAB-u pišemo slijedeći skup naredbi:

```
while(~isempty(spremnik))
   midiEvent = spremnik.get();
   if ~isempty(midiEvent)
      j = j + 1;
      vremena(j) = midiEvent.getTick();
      midiPoruka = midiEvent.getMessage();
      poruke{j} = midiPoruka.getMessage();
      if poruke{j}(1) == -97
         ulaz.close();
         break
      end
   end
end
```

Dakle, sve dokle god je spremnik pun, pomoću metode `get()` iz klase `SpremnikMidiPoruka`, dohvaćamo `MidiEvent` objekte. Ako taj objekt nije prazan, iz njega ćemo izvaditi korisne informacije o MIDI događaju i vremenskoj štampi poruke, te ih pospremiti u MATLAB-ovo polje `vremena` i cell polje `poruke`. Pri tome se koristimo metodama `getTick()` i `getMessage()`.

Kada se detektira MIDI event na 16. kanalu, to znači da je došlo do kraja sekvence i da treba zatvoriti komunikaciju prijamnika i odašiljača.

5.4.4. Matrica odsviranih nota

Prije nego što se krene u direktnu sintezu nota, potrebno je malo organizirati prikupljene informacije koje smo izvukli iz `MidiEvent` objekata. Naime, za svaku notu moramo pažljivo odrediti pripadno NOTE_ON i NOTE_OFF vrijeme kako bi vremenski ispravno smjestili notu u cijelokupnom izlaznom signalu. U polju `vremena` imamo pospremljena vremena svih događaja, ali to polje nam ništa ne govori o tome da li je vrijeme vezano za NOTE_ON ili NOTE_OFF događaj. Jedino što nam je poznato iz tog polja je da su vremena pospremljena onim redoslijedom kojim su izvučena iz

spremnika. Međutim, upravo uz pomoć cell polja `poruke` možemo odrediti koje vrijeme pripada kojem događaju zbog toga što je cell polje također punjeno događajima onim redoslijedom kojim su bili punjeni u spremnik.

Prije samog postupka kreiranja matrice odsviranih nota, moramo maknuti neke viška podatke koji nam mogu uzrokovati samo probleme u daljnjem postupku sinteze. Prvo što trebamo učiniti je maknuti podatke o kontrolnoj noti sa 16.-og kanala koja nam je signalizirala kraj sekvence odsviranih nota, a drugo je skalirati sva vremena MIDI evenata prema prvoj primljenoj noti i pretvoriti ih u sekunde.

Skaliranje je potrebno zbog konačnog izlaznog signala, jer na taj način osiguravamo da početak signala bude pravilno određen. Ako bi izostavili skaliranje vremena, poslijedica bi bila ta što bi između početka konačnog izlaznog signala i početka prve sintetizirane note bila vremenska rupa. Ovako osiguravamo da će se prva sintetizirana nota naći na početku vektora izlaznog signala. Naredbe u MATLAB-u kojima ostvarujemo prethodno opisanu funkcionalnost su slijedeće:

```
% makni podatke zadnje poruke, jer je to kontrolna poruka:
vremena = vremena(1:length(vremena)-1);
poruke = poruke(1:length(poruke)-1);

% skaliraj vremena prema prvoj primljenoj noti:
vremena = vremena - (min(vremena)-1000);
vremena = vremena/10^6;  % pretvori vrijeme u sekunde
```

Matrica koju želimo kreirati je opisana slijedećom tablicom. Radi primjera, tablica je popunjena random vrijednostima, čisto radi predodžbe kako bi matrica trebala izgledati:

Tablica 5.2: Matrica odsviranih nota

Kanal	Broj Note	Glasnoća	NoteOnTime	NoteOffTime
-112	72	64	0.1145	0.1267
-112	54	78	0.2573	0.4521
-112	43	78	0.8543	0.8932
-112	102	93	2.4567	3.7833
...	...	...	...	...

Jedna važna stvar koju treba napomenuti, je ta da se pri konstrukciji matrice nota služimo činjenicom da su NOTE_OFF događaji u MIDI porukama prikazani tako da je

glasnoća tih događaja jednaka nuli. To je česta pojava u komercijalnim sintesajzerima, a to je slučaj i za Yamaha SHS-10 sintesajzer kojim se poslužilo pri izradi ovoga rada.

Programski kod u MATLAB-u kojim se kreira matrica nota je slijedeći:

```
for i = 1:length(poruke)
    if poruke{i}(3) ~= 0
        note(j,1) = poruke{i}(1);  % kanal
        note(j,2) = poruke{i}(2);  % broj note
        note(j,3) = poruke{i}(3);  % glasnoća
        note(j,4) = vremena(i);    % noteOn vrijeme
        j = j + 1;
    else
       note(k,5) = vremena(i);    % noteOff vrijeme
       k = k + 1;
    end
end
```

Sada konačno možemo prijeći na sintezu prikupljenih nota.

5.4.5. Sinteza nota

Izlaznom valnom obliku, koji će opisivati naše sintetizirane note, trajanje je određeno preko zadnjeg NOTE_OFF vremena koje se nalazi u matrici nota. Pomoću ove činjenice moramo provesti inicijalizaciju polja izlaznog valnog oblika, u koje pospremamo uzorke sintetiziranih nota, kako nebi dolazilo do repetitivnog alociranja memorije, što bespotrebno usporava izvršavanje programskog koda:

```
t = 0:1/Fs:ceil(max(note(:,5)))+ 0.05;
izlaz = zeros(size(t));
```

pri čemu je konstantna vrijednost od 0.05 sekundi nužna zbog toga da se ostavi mrvicu viška mjesta, da se spriječi naglo rezanje signala.

Sada, za svaku notu iz matrice možemo krenuti sa sintezom. Pri tome se najprije poziva funkcija `Metoda.m` koja na temelju broja kanala note određuje koja metoda sinteze će se pozvati. Odabrano je da se za note prvoga kanala poziva metoda wavetable sinteze, za note drugoga kanala metoda frekvencijske modulacije, te na kraju za treći kanal se poziva Karplus-Strong algoritam.

```
for i = 1:length(poruke)/2  % sintetiziraj notu po notu
    % pozovi program koji odabire metodu sinteze:
    snd = Metoda(note(i,:));
    % podešavanje jačine pritiska note:
    snd = double(note(i,3))/100*snd;
    t2 = round(note(i,4)*Fs)+1:round(note(i,4)*Fs)...
          + length(snd);
    izlaz(t2) = izlaz(t2) + snd;
end
```

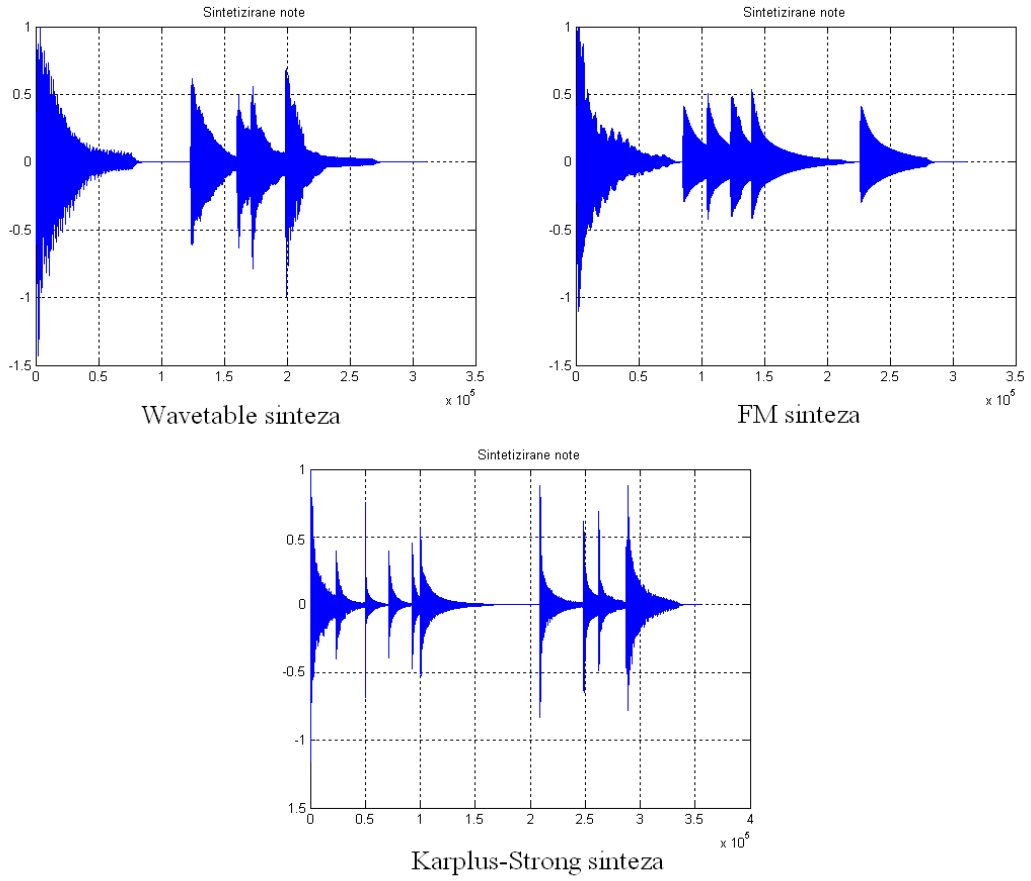
Funkcija `Metoda.m` kao argument prima cijeli redak odgovarajuće note iz matrice, te na temelju tih podataka određuje koju će metodu sinteze pozvati te koje će biti trajanje note. Trajanje pojedine note se određuje razlikom između NoteOnTime i NoteOffTime elemenata matrice, tj. ta razlika određuje kada će nota prijeći u svoju release fazu. Ako bi notu samo utihnuli u tom trenutku, rezultat ne bi zvučao prirodno jer se niti jedan realan instrument ne ponaša na taj način.

Nakon što funkcija `Metoda.m` vrati sintetizirane uzorke, ti uzorci se skaliraju sukladno sa vrijednošću koja je određena elementom *glasnoća* matrice. Efekt je taj da ako je tipka na klavijaturi jače pritisnuta, glasnoća sintetizirane note će biti proporcionalno veća. Konačno, uzorci note se pohranjuju na odgovarajuće mjesto u izlaznom signalu, te se kreće sa sintezom slijedeće note. Postupak se ponavlja sve dok se ne sintetiziraju sve note iz matrice.

Kako ne bi došlo do naglog rezanja signala, do čega je vrlo lako doći pošto se uzorci kontinuirano zbrajaju u izlaznom signalu, normalizacija amplitude je neophodna da bi se postigao neizobličen valni oblik: `izlaz = izlaz/max(izlaz);`

Krajnji rezultat je poprilično zadovoljavajući. Svaka nota se postavlja na pravilno mjesto, trajanje note je također ispravno podešeno, i pri sviranju više nota odjednom ne dolazi do čujnih izobličenja signala. Na idućoj slici (5.17) su, radi ilustracije rezultata, prikazana tri valna oblika, po jedan za svaku metodu sinteze. Note su odsvirane nasumično, sa različitim trajanjima, a na početku svakog valnog oblika je pritisnuto i više nota odjednom kako bi se testirao odziv na akorde.

Potpuni kod prethodno diskutiranog glavnog programa, može se pronaći u dodatku A.1 na kraju ovoga rada.



Slika 5.17: Sintetizirani valni oblici pomoću tri algoritma sinteze

6. Rezultati

6.1. Analiza i diskusija

Na početku istraživanja za ovaj rad, prvotna ideja za implementaciju programa sinteze instrumenata bila je da se implementira odziv instrumenta u realnom-vremenu uz omogućenu polifoniju. To se pokazalo jako teško izvedivim zbog činjenice da programska okolina MATLAB sučelja nije baš prikladna za takve vrste "on-line" problema. Naime, radi se o tome da MATLAB nije *multithreaded* program, koji omogućava paralelno izvršavanje nekoliko segmenata koda (*thread-ova*) istovremeno, kao što to omogućuju npr. Java ili C/C++ jezici, već svoje naredbe izvršava slijedno tj. onim redoslijedom kako su i napisane.

Unatoč tome što je MATLAB većim dijelom pisan u Java okruženju, i što podržava pristup preddefiniranim i korisničkim Java klasama i paketima, te klase služe uglavnom kao proširenja funkcionalnosti MATLAB-ovih naredbi, a pogotovo za ulazno/izlazno korisničko sučelje. Za real-time implementaciju sintesajzera, potpuni prelazak u Java programersku okolinu bi bila prilično dobra ideja, pogotovo zbog toga što Java ima iznimno detaljno opisanu specifikaciju svojih klasa i paketa, pa tako i za pakete `javax.sound.midi` i `javax.sound.sampled` koji služe za riješavanje svih vrsta audio problema. U tim paketima se nalazi i mnoštvo sučelja (*engl. interface*) koja se sastoje od karakterističnih apstraktnih metoda koje je potrebno implementirati kako bi se postigla namjenska funkcionalnost tih sučelja. Tako npr. u paketu `javax.sound.midi` postoji sučelje `Synthesizer` koje u svojoj specifikaciji ima navedeno popis svih metoda koje je potrebno realizirati kako bi sučelje radilo ispravno. U specifikaciji, međutim, nije navedeno kako se te metode trebaju implementirati već to ostaje na mašti, znanju i inovativnosti programera.

6.1.1. Složenost programskog koda

Bez obzira na pojedine ograničenosti MATLAB-a i njegove okoline, implementirani programski kod u off-line načinu rada daje poprilično dobre rezultate sinteze nota instrumenata. Algoritmi su testirani za razne kombinacije nota i različita trajanja te se u svakom slučaju pravilno sintetiziraju, za sve tri metode sinteze. Glavna karakteristika koja razlikuje te metode jednu od druge jest vrijeme koje je potrebno da se sintetiziraju sve note koje su odsvirane i pospremljene u matricu nota. Normalno je za očekivati da što je više nota odsvirano, to će vrijeme sinteze biti duže, no to vrijeme je također ovisno i o složenosti programskog koda.

Ako pažljivo promotrimo sva tri algoritma sinteze i usporedimo ih jedan sa drugim, već na prvi pogled možemo pretpostaviti kojemu od njih će trebati najviše vremena za sintezu nota. Wavetable sinteza je relativno jednostavna metoda koja koristi već unaprijed izračunate i pospremljene uzorke u valnim tablicama, te pomoću interpolacije uzoraka određuje uzorke sintetiziranih nota. Pri tome je interpolacija jedini računski zahtjevniji postupak, pa možemo zaključiti kako će se ovaj algoritam prilično brzo odvijati.

Sinteza Karplus-Strong algoritmom je također među bržim metodama sinteze, kod koje se vrijeme najviše gubi na pomicanje uzoraka linije kašnjenja pri svakoj iteraciji odnosno pri svakom izračunu uzoraka izlaznog signala. No unatoč tome, algoritam je iznimno brz radi činjenice da se izvršava relativno malo računskih operacija koje u pravilu usporavaju programski kod.

Vremenski najzahtjevnija metoda sinteze je svakako FM sinteza, što se odmah može primjetiti iz složenosti algoritma kojim je implementirana. Za svaku FM notu potrebno je generirati 3 modulatora i 3 nosilaca, te je svih tih 6 signala potrebno provući kroz ovojnicu signala, što sve skupa nije kratkotrajan postupak. Ako bi eventualno smanjili broj generiranih signala, složenost bi se smanjila ali pod cijenu kvalitete FM signala. Između ostaloga, cilj sinteze je bio što bolje aproksimirati stvarnu notu DX7 Rhodes klavira, tako da je broj generiranih modulatora i nosilaca FM signala opravdan.

Složenost algoritma Wavetable sinteze

Pri sintetizaciji note metodom wavetable sinteze, glavnina koda je namjenjena određivanju pomoću koje valne tablice, od njih 16 pohranjenih na hard disku, će se izvršiti postupak sinteze. Nakon što se valna tablica odredi, slijedi računski najzahtjevniji (iako dosta brz) dio koda, a to je interpolacija uzoraka. Prikažimo dio koda gdje se ovo nalazi:

```
omjer = 2^(5/12);
L = length(nota);
xi = omjer:omjer:L;
snd = interp1(nota, xi);
```

Interpolacija se izračunava prema izrazu 5.4 spomenutog u ranijim poglavljima, a broj uzoraka koji se interpolira ovisi direktno o veličini valne tablice `L` i varijabli `omjer`. Broj uzoraka izlaznog signala će, dakle, biti `L/omjer`, a broj operacija pomoću kojih se ti uzorci izračunavaju biti će:

$$O_1 \simeq \frac{L}{omjer} \cdot (1\ zbrajanje\ +\ 1\ oduzimanje\ +\ 1\ mnozenje) \simeq \frac{3L}{omjer} \tag{6.1}$$

Budući da su valne tablice veličine čak i po 400 000 uzoraka, broj O_1 doseže otprilike oko 10^6 operacija, samo za interpolaciju uzoraka.

Svakoj sintetiziranoj noti se podešava i trajanje, tako da se za zadano NoteOffTime vrijeme, nota provlači kroz *release* fazu ADSR ovojnice:

```
t_R = (length(snd) - NoteOffTime*Fs);
koeficijenti = [exp(-1/0.19), exp(-1/0.105),...
                exp(-1/0.103), exp(-1/0.13)];
ADSR = ADSRgen(t_R, koeficijenti, 'RELEASE');
snd(NoteOffTime*Fs:length(snd)-1) = ...
            ADSR.*snd(NoteOffTime*Fs:length(snd)-1);
```

Ako izrazimo broj uzoraka koji moramo provući kroz *release* fazu pomoću:

$$M = \frac{L}{omjer} - 44100 \cdot NoteOffTime \tag{6.2}$$

možemo doći do broja operacija O_2 koje su potrebne za izračun uzoraka ovojnice. Naime, pri pozivu `ADSRgen.m` funkcije, broj uzoraka M će se izračunavati prema 'RELEASE' izrazu: `izlaz(i) = izlaz(i-1)- R*izlaz(i-1);`
što znači da je broj operacija:

$$O_2 \simeq M \cdot (1\ oduzimanje\ +\ 1\ mnozenje) \simeq 2M \tag{6.3}$$

Što je NoteOffTime vrijeme veće, to će se manje uzoraka morati množiti sa vrijednostima *release* faze ovojnice i broj operacija O_2 će biti manji.

Sada još samo treba tako izračunate uzorke ADSR ovojnice, pomnožiti sa uzorcima izlaznog signala, što za sobom povlači još 2M množenja:

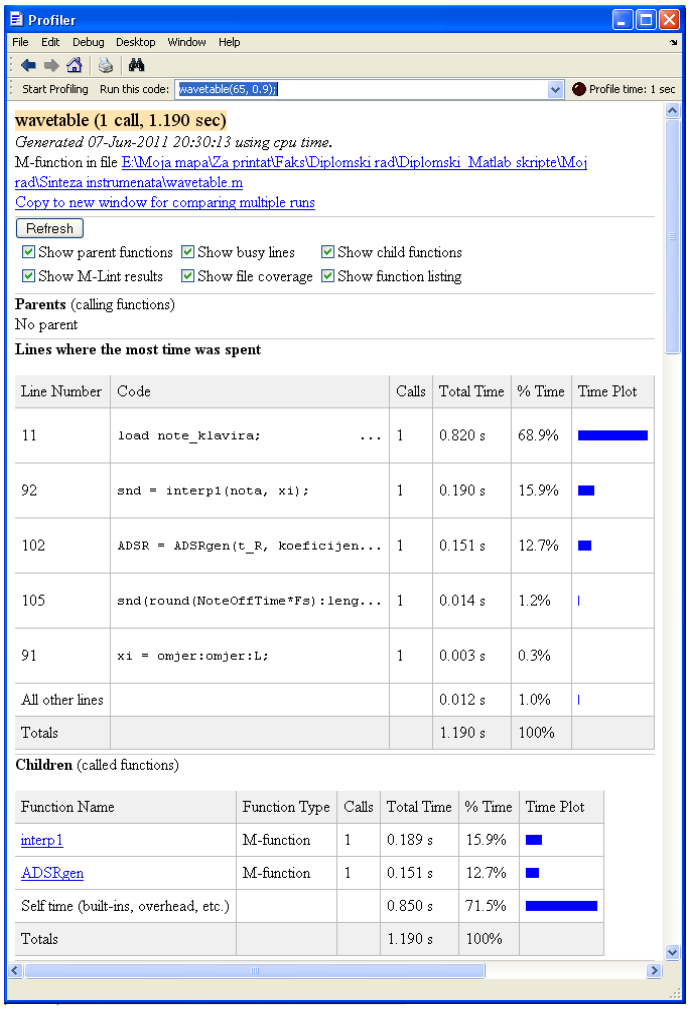
$$O_3 \simeq 2M \tag{6.4}$$

Dakle, imamo ukupan broj operacija:

$$O \simeq O_1 + O_2 + O_3 \simeq \frac{3L}{omjer} + 4M \tag{6.5}$$

Za vrlo kratke note, NoteOffTime vrijeme će biti jako malo pa će i vrijednost M biti približno $M \simeq \frac{L}{omjer}$, a to će rezultirati sa vrijednošću $O \simeq \frac{7L}{omjer}$, što je više nego dupla vrijednost od O_1. No, na sreću, tako kratke note su rijetke pri sviranju prosječnih glazbenih skladbi.

Radi ilustracije, na idućoj slici je prikazan proces izvršavanja koda pri sintetiziranju jedne note wavetable sinteze, koji je dobiven pomoću MATLAB-ovog *Profiler GUI* sučelja:



Slika 6.1: Vrijeme izvršavanja Wavetable sinteze

Osim na već predviđene dijelove koda, prema slici 6.1 možemo vidjeti da se veliki dio vremena (čak 68.9 %) troši na uploadanje valnih tablica, što je vrlo loše. No, međutim, ovo je vrlo lako riješiv problem jer sve što treba napraviti jest prebaciti upload tablica u glavni program, kako bi se očitale samo jedanput, a ne pri svakoj sintezi note kao što je implementirano.

Složenost Karplus-Strong algoritma

Od sve tri implementirane metode sinteze, Karplus-Strong sinteza daje najbrže rezultate. Algoritam počinje sa inicijalizacijom osnovnih parametara, na koje troši svega nekoliko aritmetičkih operacija:

```
f_osn = 440.0*(2.0^((double(broj_note) - 69.0)/12.0));
N = floor(Fs/f_osn);
D = (Fs - f_osn*(N+1))/(f_osn*(N-1)- Fs);
```

Na gornje tri linije koda se troši ukupan broj operacija:

$$O_1 \simeq 5\ oduzimanja\ +\ 3\ dijeljenja\ +\ 3\ mnozenja \simeq 11 \tag{6.6}$$

što je zanemariv broj u usporedbi sa ostatkom koda koji slijedi. Naime, ukupan broj uzoraka Karplus-Strong signala je predodređen da bude $N = 66150$, što određuje kvalitetu signala, ali i ukupan broj operacija nad uzorcima, te broj prolazaka kroz delay liniju. Pri izračunu uzoraka, program prolazi kroz slijedeće naredbe:

```
for i=1:duljina
     nota(i) = sluc(i) - (C+D)*y(1) - C*D*y(2) +...
               A*D*y(N) + (A+B*D)*y(N+1) + B*y(N+2);

     for j=(length(y)-1):-1:1
         y(j+1) = y(j);
     end
     y(1) = nota(i);
end
```

Pri tome se, za izračun izlaznih uzoraka troši O_2 broj aritmetičkih operacija:

$$O_2 \simeq N \cdot (2\ oduzimanja\ +\ 3\ zbrajanja\ +\ 5\ mnozenja) \simeq 661500 \tag{6.7}$$

Delay linija je sastavni dio Karplus-Strong algoritma, i ona predstavlja namjerno unešeno kašnjenje u sustav. Iako se na nju troši najviše vremena kod izvođenja programa, ona ne izvršava nikakve aritmetičke operacije, već samo zakašnjava uzorke za N mjesta, pa je stoga nećemo uzimati u obzir.

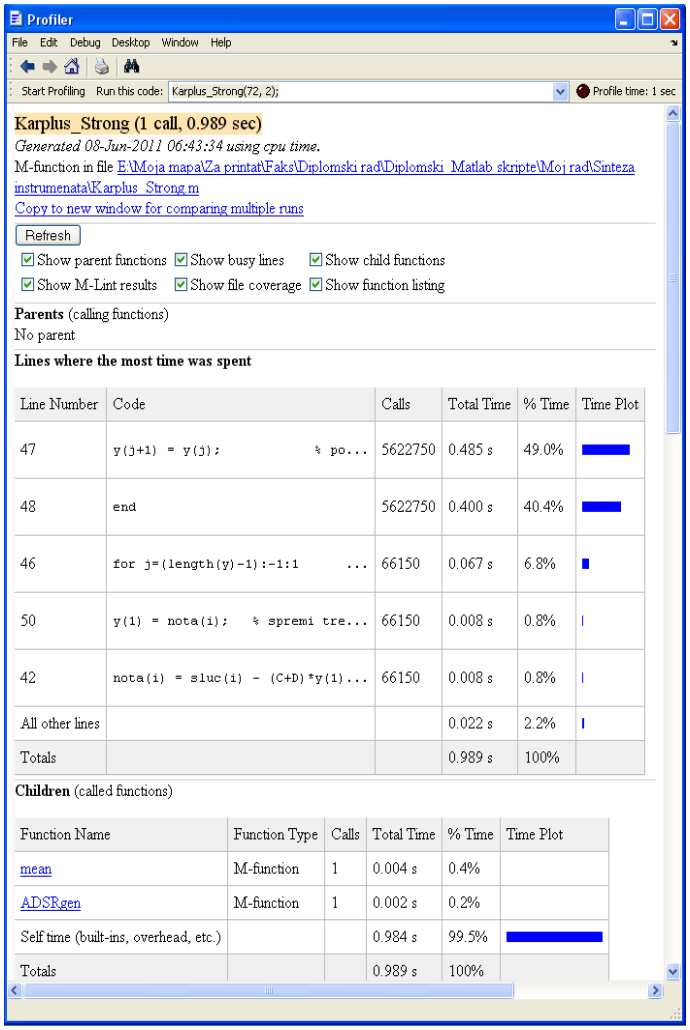
Osim na izračun uzoraka, dio operacija se kao i kod wavetable sinteze, troši na oblikovanje signala ADSR ovojnicom, i to upravo prema izrazima 6.3 i 6.4.

Dakle, ukupna složenost Karplus-Strong algoritma je:

$$O \simeq O_1 + O_2 + 4M \simeq 661511 + 4M \tag{6.8}$$

gdje je broj M određen NoteOffTime vremenom, preko izraza 6.2.

Prikaz redoslijeda izvršavanja koda, te utrošenog vremena na pojedine dijelove, dan je na slici 6.2:

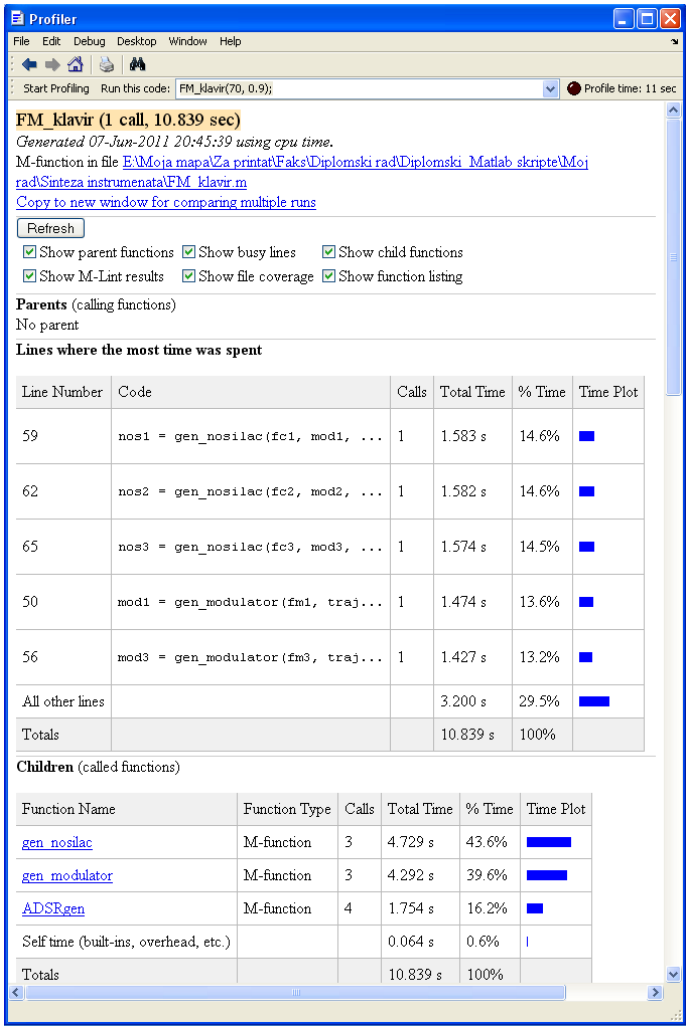


Slika 6.2: Vrijeme izvršavanja Karplus-Strong algoritma

Možemo uočiti kako je izvršavanje koda iznimno brzo, traje svega 0.989 sekundi, od čega 49 % otpada na prolazak uzoraka kroz delay liniju.

Složenost algoritma FM sinteze

Daleko najzahtjevniji od sva tri algoritma sinteze je FM sinteza, što se može vidjeti i iz priloženog dijagrama izvođenja programa, na slici 6.3.



Slika 6.3: Vrijeme izvršavanja algoritma FM sinteze

Vrijeme izvršavanja koda je čak 10.839 sekundi, što znači da bi za sintezu npr. 6 nota trebalo čekati otprilike jednu minutu, što je poprilično puno. Razlog ove složenosti leži u činjenici da je za jednu FM notu potrebno generirati 6 signala, što pri slijednom izvođenju programa konzumira dosta vremena.

Pri pokretanju programa, na prvih nekoliko naredbi, kao što su izračun npr. frekvencije note i trajanja signala, utroši se tek nekoliko aritmetičkih operacija koje slo-

bodno možemo zanemariti. Prvi dio koda koji zahtjeva malo veći izračun je izračun triju ADSR ovojnica:

```
ADSR1 = ADSRgen(trajanje, koeficijenti1);
ADSR2 = ADSRgen(trajanje, koeficijenti2);
ADSR3 = ADSRgen(trajanje, koeficijenti3);
```

Pošto je trajanje FM signala predodređeno da bude 2 sekunde, toliko će trajati i ADSR ovojnica svakog signala. Pozivom funkcije `ADSRgen`, počinje se sa kreiranjem ovojnice i to kroz svih njenih 4 faza, što znači da nam za izračun triju ovojnica treba slijedeći broj operacija:

$$O_1 \simeq \frac{2 \cdot 44100}{4} \cdot (4 \cdot (1\ mnozenje\ +\ 1\ oduzimanje)) \cdot 3 \simeq 529200 \tag{6.9}$$

Nakon ovojnica, potrebno je geneririati modulatore, što se započinje pozivom `gen_modulator.m` funkcije. Najznačajnije naredbe, koje zahtjevaju najviše operacija su iduće:

```
for i = 1:trajanje
    indeks = mod(prethIndeks + inkrMod, L);
    signal(i) = sinus(ind) + (indeks - ind)*...
                (sinus(ind + 1) - sinus(ind));
end
```

Za tri modulatora, gornje naredbe će se izvršiti tri puta, što daje broj aritmetičkih operacija O_2:

$$O_2 \simeq 2 \cdot 44100 \cdot (4\ zbrajanja\ +\ 1\ dijeljenje\ +\ 1\ mnozenje)) \cdot 3 = 1587600 \tag{6.10}$$

Idući na redu su nosioci, čije generiranje na sličan način rezultira brojem aritmetčkih operacija O_3:

$$O_3 \simeq 2 \cdot 44100 \cdot (2\ zbrajanja\ +\ 3\ oduzimanja\ +\ 1\ dijeljenje \tag{6.11}$$
$$+\ 4\ mnozenja) \cdot 3 \simeq 2646000$$

Za regulaciju trajanja note, također imamo broj operacija prema izrazima 6.2, 6.3 i 6.4, a sve zajedno kada se izračuna, dobivamo broj aritmetičkih operacija potrebnih za izvršavanje FM algoritma sinteze:

$$O \simeq O_1 + O_2 + O_3 + 4M \simeq 4762800 + 4M \tag{6.12}$$

6.2. Zaključak

Ovim radom se nastojalo ući u problematiku modeliranja glazbenih instrumenata, te dati uvid u tek manji dio ideja i praktičnih ostvarenja koja se mogu postići tehnikama digitalne obradbe signala. Iako već vrlo napredne, tehnike u ovom području tek danas doživljavaju svoj pravi procvat koji je započeo 80-ih godina prošloga stoljeća. Ako uzmemo u obzir da, pored obrađenih metoda u ovome radu, postoje još deseci jednako efektivnih postupaka sinteze instrumenata, dobiti ćemo dojam o snazi DSP alata sa kojima se susrećemo i u ostalim aspektima našega svakodnevnoga života. Zadatak inženjera i znanstvenika je konstantno "unaprijeđivati napredak" tehnologija i u tome se nediskutabilno uspijeva na dnevnoj razini.

Uvijek ima mjesta za poboljšanja – pa tako i za ovaj rad, koji je izrađen sa relativno skromnim poznavanjem mogućnosti koje nudi svijet digitalne obrade signala. Premda, se kroz izradu rada krenulo u mrvicu drugačijem smjeru zbog ograničenja sredstava kojima se služilo, okosnica istraživanja je ostala nepromjenjena. Svi postupci koji su objašnjeni vrijede općenito i kao takvi se danas iskorištavaju u komercijalne svrhe. U kombinaciji sa hardverskim riješenjima poput DSP procesora, koji su svakodnevno sve brži i sve jači, razvoj softvera je praktički neograničen. Umjesto tri instrumenta, koja su dizajnirana ovim radom, u komercijalnim proizvodima ih se može nalaziti na tisuće. Čak i kod uređaja sa ograničenim kapacitetom memorije, može se poslužiti tehnikama programiranja i kodiranja koje znatno mogu smanjiti memorijske resurse i prepustiti procesoru da radi većinu posla vezano za sintezu signala.

Od opisanih metoda modeliranja, danas se najviše vremena provodi na proučavanju i usavršavanju tehnika fizikalnog modeliranja instrumenata. Izgrađeni su vrlo realistični modeli klavira, gitare, bubnjeva te ostalih vrsta instrumenata među kojima je vrlo teško raspoznati da li se radi o pravom ili "umjetnom" instrumentu, realističnost je stvarno nevjerojatna.

U svakom slučaju, biti će zanimljivo osvrnuti se, za tridesetak godina, unatrag i shvatiti kako su "nekada" postojali fizički instrumenti koji danas samo loše oponašaju moderne digitalne instrumente i uređaje svakodnevnoga života.

LITERATURA

Audacity. The Free, Cross-Platform Sound Editor. http://audacity.sourceforge.net/, Travanj 2011.

Rogers Cadenhead i Laura Lemay. *Teach Yourself Java 6 In 21 Days*. Sams Publishing, Indianapolis, Indiana, 2007.

John M. Chowning. The Synthesis of Complex Audio Spectra by Means of Frequency Modulation. *Journal of the Audio Engineering Society*, 21(7):526–534, 1973.

Ed Doering. Karplus-Strong Plucked String Algorithm with Improved Pitch Accuracy. http://cnx.org/content/m15490/latest/, Ožujak 2008.

Eclipse. Explore the Eclipse Universe. http://www.eclipse.org/, Lipanj 2011.

FESB. Laboratorijske vježbe iz kolegija: "Komunikacjiski Sustavi i Protokoli" - Vježba 2 : Frekvencijska modulacija FM. http://marjan.fesb.hr/~radic/ksip_0708/ksl_0708ch2.html, Svibanj 2008.

Francois Grondin. Guitar Pitch Shifter. http://www.guitarpitchshifter.com/pitchshifting.html, Svibanj 2011.

James Irvine. Transmitting by Radio. http://www.antonine-education.co.uk/New_items/MDM/Sending.htm, Siječanj 2011.

David Jaffe i Julius Smith. Extensions of the Karplus-Strong Plucked-String Algorithm. *Computer Music Journal*, 7(2):56–69, 1983.

JavaAPI. Java 2 Platform Standard Edition 5.0 API Specification. http://download.oracle.com/javase/1,5.0/docs/api/, Travanj 2011.

Kevin Karplus i Alex Strong. Digital Synthesis of Plucked-String and Drum Timbres. *Computer Music Journal*, 7(2):43–55, 1983.

Brandy Kraemer. Scientific Pitch Notation. http://piano.about.com/library/Pitch-Notation/bl_Scientific-Pitch-Notation_2.htm, Svibanj 2011.

Sen M. Kuo i Bob H. Lee. *Real-Time Digital Signal Processing*. John Wiley and Sons Ltd., Chichester, England, 2001.

Andre Lundkvist i Rikard Qvarnström. Project: DSP Signal Generator. http://www.jiisuki.net/reports/s0002e_project_report.pdf, Listopad 2009.

Mathworks. MATLAB - The Language Of Technical Computing. http://www.mathworks.com/products/matlab/?sec=apps, Svibanj 2011.

M.Karjalainen, V.Välimäki, i Z. Jánosy. Towards High-Quality Sound Synthesis of the Guitar and String Instruments. *International Computer Music Conference*, 1993.

Native-Instruments. FM8 - Native Instruments GmbH. http://http://www.native-instruments.com/#/en/products/producer/fm8/, Svibanj 2011.

Oracle. The Java Tutorials. http://download.oracle.com/javase/tutorial/sound/index.html, Lipanj 2011.

Tae H. Park. *Introduction To Digital Signal Processing*. World Scientific Publishing Co. Pte. Ltd., Singapore, 2010.

Jussi Pekonen. Computationally Efficient Music Synthesis - Methods and Sound Design. Magistarski rad, Helsinki University of Technology, Lipanj 2007.

Bill Schottstaedt. An Introduction to FM. https://ccrma.stanford.edu/software/snd/snd/fm.html, Svibanj 1985.

Steven W. Smith. *The Scientist and Engineer's Guide To Digital Signal Processing*. California Technical Publishing, San Diego, 1997.

Dieter Thomsen. *Digitalna Audiotehnika*. Tehnička knjiga, Zagreb, 1987.

Barry Truax. Organizational Techniques for C:M Ratios in Frequency Modulation. *Computer Music Journal*, 1(4):39–45, 1977.

Wikipedia. MIDI 1.0. http://en.wikipedia.org/wiki/MIDI_1.0, Veljača 2011a.

Wikipedia. Pitch (music). http://en.wikipedia.org/wiki/Pitch_(music), Svibanj 2011b.

Dodatak A
Programski kod

A.1. Glavni program

```
% --> Glavni program <--
% Program ostvaruje vezu sa MIDI prijamnikom te detektira note.
% Nakon toga odabire metodu sinteze, ovisno o kanalu,
% te sintetizira sve note koje su odsvirane.

%uključivanje java paketa:
import java.lang.*
import java.io.*
import javax.swing.*
import javax.sound.midi.*

Fs = 44100;   % frekvencija uzorkovanja

% informacije o ulaznim uređajima:
infoIn = MidiSystem.getMidiDeviceInfo();

% filtriraj samo one uređaje koji imaju odašiljače(MIDI IN portovi):
for i = 1:infoIn.length
    temp = MidiSystem.getMidiDevice(infoIn(i));
    if (temp.getMaxTransmitters() == 0)
        infoIn(i) = [];
    end
end
i = 0;  % reset brojača

% izbornik za ulazne uređaje:
odabrani = JOptionPane.showInputDialog([], 'Odaberi uređaj',...
            'MIDI IN', JOptionPane.QUESTION_MESSAGE, [], infoIn, []);
```

```
%otvori ulazni MIDI port:
try
    if (~isempty(odabrani))
      ulaz = MidiSystem.getMidiDevice(odabrani);
      ulaz.open();
    end
catch ME
      id = ME.identifier;
      mess = ME.message;
end

% komuniciranje sa Prijamnik Java klasom
try
    % kreiraj spremnik MIDI poruka:
    spremnik = javaObject('SpremnikMidiPoruka');
    % dohvati odašiljač MIDI IN porta:
    odasiljac = ulaz.getTransmitter();
    % otvori prijamnik:
    prijamnik = javaObject('Prijamnik', spremnik);
    % poveži prijamnik i odašiljač:
    odasiljac.setReceiver(prijamnik);

    j = 0;
    vremena = zeros(1, 500);  % inicijalizacija polja
    poruke = cell(1, 500);    % radi bržeg izvođenja

    % dohvat MIDI eventa iz buffera:
    while(~isempty(spremnik))
       midiEvent = spremnik.get();
       if ~isempty(midiEvent)
          j = j + 1;
          % vrijeme eventa u mikrosekundama:
          vremena(j) = midiEvent.getTick();
          midiPoruka = midiEvent.getMessage();  % dohvati midi poruku
          poruke{j} = midiPoruka.getMessage();  % dohvati byte-ove
          if poruke{j}(1) == -97  % prekini dohvat evenata kad
             ulaz.close();        % stigne nota na 16. kanalu
             break
          end
       end
    end
```

```
% smanjivanje polja, da se izbace nule:
d = find(vremena);  % pronađi elemente različite od nule
vremena = vremena(1:length(d));
poruke = poruke(1:length(vremena));

% makni podatke od zadnje poruke, jer je to kontrolna poruka:
vremena = vremena(1:length(vremena)-1);
poruke = poruke(1:length(poruke)-1);

% skaliraj vremena prema prvoj primljenoj noti:
vremena = vremena - (min(vremena)-1000);
vremena = vremena/10^6;  % pretvori vrijeme u sekunde

% napravi matricu sa svim odsviranim notama:
note = zeros(length(poruke)/2, 5);

j = 1; k = 1;
% kreiraj matricu note:
for i = 1:length(poruke)
    if poruke{i}(3) ~= 0
        note(j,1) = poruke{i}(1);     % kanal
        note(j,2) = poruke{i}(2);     % broj note
        note(j,3) = poruke{i}(3);     % glasnoća
        note(j,4) = vremena(i);       % noteOn vrijeme
        j = j + 1;
    else
        note(k,5) = vremena(i);       % noteOff vrijeme
        k = k + 1;
    end
end


%    ----------> KREIRANA MATRICA NOTA!!! <-------------
%   | Kanal  BrojNote  Glasnoća  NoteOnTime  NoteOffTime |
%   |  ...      ...        ...       ...         ...     |
%   |  ...      ...        ...       ...         ...     |


% napravi vektor vremena za izlazni signal
t = 0:1/Fs:ceil(max(note(:,5)))+ 0.05;
izlaz = zeros(size(t));
```

```
    i = 0;
    for i = 1:length(poruke)/2    % sintetiziraj notu po notu
       % pozovi program koji odabire metodu sinteze:
       snd = Metoda(note(i,:));
       % podešavanje jačine pritiska note:
       snd = double(note(i,3))/100*snd;
       t2 = round(note(i,4)*Fs)+1:round(note(i,4)*Fs)+length(snd);
       izlaz(t2) = izlaz(t2) + snd;
    end

    %normalizacija izlaza
    izlaz = izlaz/max(izlaz);

catch ME
    id = ME.identifier;
    mess = ME.message;
end

soundsc(izlaz, Fs);
plot(izlaz); grid on; title('Sintetizirane note');

% crtanje spektrograma, po potrebi:
%figure(2)
%spectrogram(snd, 1024, 512, [], Fs)
```

A.2. Funkcija za odabir metode sinteze

```
% Funkcija koja određuje algoritam iz midi poruke
% Tri algoritma sinteze --> tri kanala

function snd = Metoda(midiPoruka)

  kanal = midiPoruka(1);       % u intervalu 0-15
  broj_note = midiPoruka(2);
  pocetakNote = midiPoruka(4);
  krajNote = midiPoruka(5);
  NoteOffTime = krajNote - pocetakNote;

  % za kanal 0, noteOn = 144 (-112)
  % za kanal 15, noteOn = 159 (-97)

    switch kanal
        case -112,
            snd = wavetable(broj_note, NoteOffTime);
        case -111,
            snd = FM_klavir(broj_note, NoteOffTime);
        case -110,
            snd = Karplus_Strong(broj_note, NoteOffTime);
    end
```

A.3. Funkcija za Frekvencijsku Modulaciju

```
function nota = FM_klavir(broj_note, NoteOffTime)

% izračun frekvencije titranja:
f_0 = 440.0*(2.0^((double(broj_note) - 69.0)/12.0));

% vremenski parametri
Fs = 44100;
T = 1/Fs;
vrijeme = 2;                 % stvori 2 sekunde signala
trajanje = vrijeme*44100;    % trajanje u uzorcima
t = 0:T:vrijeme;             % vektor vremena
valni_oblik = 'sin';
```

```
% FM parametri (3 nosača i 3 modulatora)
fc1 = f_0;          % centralna frekvencija 1 --> c:m=1:14
fm1 = 14*f_0;       % modulacijska frekvencija 1
beta1 = 10;         % indeks modulacije

fc2 = 2*f_0;        % centralna frekvencija 2 --> c:m=2:1
fm2 = f_0;          % modulacijska frekvencija 2
beta2 = 2;          % indeks modulacije

fc3 = f_0;          % centralna frekvencija 3 --> c:m=1:1
fm3 = f_0;          % modulacijska frekvencija 3
beta3 = 0.75;       % indeks modulacije

%----> napravi tri ovojnice <-----
% najkraća
koeficijenti1 = [exp(-1/0.19), exp(-1/0.115),...
                 exp(-1/0.12), exp(-1/0.13)];
ADSR1 = ADSRgen(trajanje, koeficijenti1);

% srednja
koeficijenti2 = [exp(-1/0.19), exp(-1/0.105),...
                 exp(-1/0.103), exp(-1/0.13)];
ADSR2 = ADSRgen(trajanje, koeficijenti2);

% najduža
koeficijenti3 = [exp(-1/0.19), exp(-1/0.1),...
                 exp(-1/0.099), exp(-1/0.12)];
ADSR3 = ADSRgen(trajanje, koeficijenti3);

% generiraj modulatore i nosioce:
mod1 = gen_modulator(fm1, trajanje, valni_oblik);
mod1 = 40*ADSR1.*mod1;  % signal je normaliziran pa ga treba pojačati

mod2 = gen_modulator(fm2, trajanje, valni_oblik);
mod2 = 40*ADSR2.*mod2;

mod3 = gen_modulator(fm3, trajanje, valni_oblik);
mod3 = 40*ADSR2.*mod3;

nos1 = gen_nosilac(fc1, mod1,beta1, trajanje, valni_oblik);
nos1 = ADSR1.*nos1;

nos2 = gen_nosilac(fc2, mod2, beta2, trajanje, valni_oblik);
```

```
nos2 = ADSR3.*nos2;

nos3 = gen_nosilac(fc3, mod3, beta3, trajanje, valni_oblik);
nos3 = ADSR3.*nos3;

% zbroji nosioce:
nota = nos1 + nos2 + nos3;

% release vremenski segment, za podešavanje trajanja note:
t_R = (length(nota) - NoteOffTime*Fs);

koeficijenti = [exp(-1/0.19), exp(-1/0.105),...
                exp(-1/0.103), exp(-1/0.13)];
ADSR = ADSRgen(t_R, koeficijenti, 'RELEASE');

%kad dođeš do NoteOff vremena, prijeđi u release fazu
nota(NoteOffTime*Fs:length(nota)-1) = ...
            ADSR.*nota(NoteOffTime*Fs:length(nota)-1);

% zadrži samo signal kojem je amplituda
% različita od 0:
nota = nota(nota ~= 0);

%normalizacija amplitude:
max_nota = max(nota);
nota = nota./max_nota;


% ----> Crtanje grafova <----
%sound(nota, Fs);
% crtanje izlaznog signala
%figure(1)
%plot(nota); axis([0 length(nota) -1.5 1.5])

% crtanje spektrograma
%figure(2)
%spectrogram(nota, 1024, 256, [], Fs)
```

A.4. Funkcija za generiranje valnih oblika

A.4.1. Generiranje modulatora

```
function signal = gen_modulator(frekv, trajanje, valni_oblik)

Fs = 44100;

if nargin < 3
    valni_oblik = 'sin';
end

switch valni_oblik
   case 'sin',
      % uploadaj tablicu od 4096 uzoraka za sinus:
      load sinus;

      % inicijalizacija parametara za modulator:
      signal = zeros(1, trajanje);
      prethIndeks = 0;

      L = length(sinus);    % duljina valne tablice

      % izračun inkrementa za modulator:
      inkrMod = L*frekv/Fs;

      try
         for i = 1:trajanje
             % izračun indeksa za modulator:
             indeks = mod(prethIndeks + inkrMod, L);
             prethIndeks = indeks;
             ind = floor(indeks);

             if ind ~= 0
                % linearna interpolacija:
                signal(i) = sinus(ind) + (indeks - ind)*...
                           (sinus(ind + 1) - sinus(ind));

             % za slučaj da ind u jednom trenutku, postane = 0,
             % resetiraj prethIndeks:
             elseif ind == 0
                 prethIndeks = 0;
```

```
            end
        end
    catch ME
        id = ME.identifier;
        mess = ME.message;
    end

case 'cos',
    % uploadaj tablicu od 4096 uzoraka za kosinus:
    load kosinus;

    % inicijalizacija parametara za modulator:
    signal = zeros(1, trajanje);
    prethIndeks = 0;

    L = length(kosinus);     % duljina valne tablice

    % izračun inkrementa za modulator:
    inkrMod = L*frekv/Fs;

    try
        for i = 1:trajanje
            % izračun indeksa za modulator:
            indeks = mod(prethIndeks + inkrMod, L);
            prethIndeks = indeks;
            ind = floor(indeks);

            if ind ~= 0
                % linearna interpolacija:
                signal(i) = kosinus(ind) + (indeks - ind)*...
                            (kosinus(ind + 1) - kosinus(ind));

            % za slučaj da ind u jednom trenutku, postane = 0,
            % resetiraj prethIndeks:
            elseif ind == 0
                prethIndeks = 0;
                signal(i) = 1;
            end
        end
    catch ME
        id = ME.identifier;
        mess = ME.message;
    end
```

```
case 'saw',
   % uploadaj tablicu od 4096 uzoraka za pilasti:
   load pilasti;

   % inicijalizacija parametara za modulator:
   signal = zeros(1, trajanje);
   prethIndeks = 0;

   L = length(pilasti);    % duljina valne tablice

   % izračun inkrementa:
   inkrMod = L*frekv/Fs;

   try
      for i = 1:trajanje
          % izračun indeksa za modulator:
          indeks = mod(prethIndeks + inkrMod, L);
          prethIndeks = indeks;
          ind = floor(indeks);

          if ind ~= 0
             % linearna interpolacija:
             signal(i) = pilasti(ind) + (indeks - ind)*...
                         (pilasti(ind + 1) - pilasti(ind));

          % za slučaj da ind u jednom trenutku, postane = 0,
          % resetiraj prethIndeks:
          elseif ind == 0
              prethIndeks = 0;
          end
      end
  catch ME
      id = ME.identifier;
      mess = ME.message;
  end

case 'square',
   % uploadaj tablicu od 4096 uzoraka za pravokutni:
   load pravokutni;

   % inicijalizacija parametara za modulator:
   signal = zeros(1, trajanje);
```

```
      prethIndeks = 0;

      L = length(pravokutni);    % duljina valne tablice

      % izračun inkrementa za modulator:
      inkrMod = L*frekv/Fs;

      try
         for i = 1:trajanje
             % izračun indeksa za modulator:
             indeks = mod(prethIndeks + inkrMod, L);
             prethIndeks = indeks;
             ind = floor(indeks);

             if ind ~= 0
                % linearna interpolacija:
                signal(i) = pravokutni(ind) + (indeks - ind)*...
                            (pravokutni(ind + 1) - pravokutni(ind));

             % za slučaj da ind u jednom trenutku, postane = 0,
             % resetiraj prethIndeks:
             elseif ind == 0
                 prethIndeks = 0;
             end
         end
     catch ME
         id = ME.identifier;
         mess = ME.message;
     end
end

%sound(signal, Fs);
%figure(1)
%plot(signal); grid on;
```

A.4.2. Generiranje nosioca

```
function nos = gen_nosilac(fc, modulator, beta,  trajanje, valni_oblik)

Fs = 44100;

if nargin < 5
    valni_oblik = 'sin';
end

switch valni_oblik
   case 'sin',
      % uploadaj tablicu od 4096 uzoraka za sinus:
      load sinus;

      % inicijalizacija parametara nosioca:
      nos = zeros(1, trajanje);
      prethIndeksNos = 0;

      % duljina valne tablice:
      L = length(sinus);

      try
         for i = 1:trajanje
             % izračun inkrementa nosioca:
             inkrNos = (L*fc/Fs + beta*modulator(i));

             % indeksi za nosilac
             indeksNos = mod(prethIndeksNos + inkrNos, L);
             prethIndeksNos = indeksNos;
             indNos = floor(indeksNos);

             if indNos ~=0
                % linearna interpolacija:
                nos(i) = sinus(indNos) + (indeksNos - indNos)*...
                           (sinus(indNos + 1) - sinus(indNos));

             elseif indNos == 0
                prethIndeksNos = 0;
             end
         end
     catch ME
         id = ME.identifier;
```

```
      mess = ME.message;
  end

case 'cos',
   % uploadaj tablicu od 4096 uzoraka za kosinus:
   load kosinus;

   % inicijalizacija parametara nosioca:
   nos = zeros(1, trajanje);
   prethIndeksNos = 0;

   % duljina valne tablice:
   L = length(kosinus);

   try
      for i = 1:trajanje
          % izračun inkrementa nosioca:
          inkrNos = (L*fc/Fs + beta*modulator(i));

          % indeksi za nosilac
          indeksNos = mod(prethIndeksNos + inkrNos, L);
          prethIndeksNos = indeksNos;
          indNos = floor(indeksNos);

          if indNos ~=0
             % linearna interpolacija:
             nos(i) = kosinus(indNos) + (indeksNos - indNos)*...
                        (kosinus(indNos + 1) - kosinus(indNos));

          elseif indNos == 0
             prethIndeksNos = 0;
             nos(i) = 1;
          end
      end
  catch ME
      id = ME.identifier;
      mess = ME.message;
  end

case 'saw',
   % uploadaj tablicu od 4096 uzoraka za pilasti:
   load pilasti;
```

```
   % inicijalizacija parametara nosioca:
   nos = zeros(1, trajanje);
   prethIndeksNos = 0;

   % duljina valne tablice:
   L = length(pilasti);

   try
      for i = 1:trajanje
          % izračun inkrementa nosioca:
          inkrNos = (L*fc/Fs + beta*modulator(i));

          % indeksi za nosilac
          indeksNos = mod(prethIndeksNos + inkrNos, L);
          prethIndeksNos = indeksNos;
          indNos = floor(indeksNos);

          if indNos ~=0
             % linearna interpolacija:
             nos(i) = pilasti(indNos) + (indeksNos - indNos)*...
                         (pilasti(indNos + 1) - pilasti(indNos));

          elseif indNos == 0
             prethIndeksNos = 0;
          end
      end
  catch ME
      id = ME.identifier;
      mess = ME.message;
  end

case 'square',
   % uploadaj tablicu od 4096 uzoraka za sinus:
   load pravokutni;

   % inicijalizacija parametara nosioca:
   nos = zeros(1, trajanje);
   prethIndeksNos = 0;

   % duljina valne tablice:
   L = length(pravokutni);
```

```
    try
        for i = 1:trajanje
            % izračun inkrementa nosioca:
            inkrNos = (L*fc/Fs + beta*modulator(i));

            % indeksi za nosilac
            indeksNos = mod(prethIndeksNos + inkrNos, L);
            prethIndeksNos = indeksNos;
            indNos = floor(indeksNos);

            if indNos ~=0
               % linearna interpolacija:
               nos(i) = pravokutni(indNos) + (indeksNos - indNos)*...
                           (pravokutni(indNos + 1) - pravokutni(indNos));

            elseif indNos == 0
               prethIndeksNos = 0;
            end
        end
    catch ME
        id = ME.identifier;
        mess = ME.message;
    end
end

%sound(nos, Fs);
%figure(1)
%plot(nos); grid on; axis([0 trajanje -1.5 1.5]);
```

A.5. Funkcija za generiranje ADSR ovojnice

```
function izlaz = ADSRgen(trajanje, koeficijenti, faza)

    izlaz = zeros(1,trajanje);

    A = koeficijenti(1); D = koeficijenti(2);
    S = koeficijenti(3); R = koeficijenti(4);

    if nargin < 3
        faza = 'ATTACK';
    end
```

```
    for i = 2:trajanje
        switch faza,
           case 'ATTACK',
               izlaz(i) = izlaz(i-1) + A*(1 - izlaz(i-1));
               if izlaz(i) > 0.99
                   faza = 'DECAY';
               end
               continue
           case 'DECAY',
               if izlaz(i) == 0
                  izlaz(1) = 1;
               end
               izlaz(i) = izlaz(i-1) - D*izlaz(i-1);
               if izlaz(i) <= 0.6;
                  faza = 'SUSTAIN';
               end
               continue
           case 'SUSTAIN',
               if izlaz(i) == 0
                  izlaz(1) = 1;
               end
               izlaz(i) = izlaz(i-1) - S*izlaz(i-1);
               if izlaz(i) <= 0.05
                  faza = 'RELEASE';
               end
               continue
           case 'RELEASE',
               if izlaz(i) == 0
                  izlaz(1) = 1;
               end
               izlaz(i) = izlaz(i-1)- R*izlaz(i-1);
               if izlaz(i) < 0.001
                  izlaz(i) = 0;
                  faza = 'OFF';
               end
               continue
           case 'OFF',
               break
        end
    end

% crtanje ovojnice
%plot (izlaz); grid on; axis([0 length(izlaz) -1.5 1.5])
```

A.6. Funkcija za Karplus–Strong algoritam

```
function nota = Karplus_Strong(broj_note, NoteOffTime)

clc
Fs = 44100;
duljina = 66150;

% računanje frekvencije:
f_osn = 440.0*(2.0^((double(broj_note) - 69.0)/12.0));

% kašnjenje delay linije:
N = floor(Fs/f_osn);

% koeficijent APF-a za frakcionalno kašnjenje:
D = (Fs - f_osn*(N+1))/(f_osn*(N-1)- Fs);

% koeficijenti NPF filtra:
A = 0.9152; B = 0.1889;
C = 0.1127;

% slučajni signal u intervalu -1 -> 1 :
sluc = 2*rand(1,N+2);
sluc = sluc - mean(sluc);   % micanje DC komponente

% ako je duljina duža od pobudnog signala,
% ispuni ostatak šuma nulama:
if duljina > length(sluc)
   sluc  = [sluc zeros(1, duljina - length(sluc))];
end

% Inicijalizacija linije kašnjenja:
y  = zeros(1, N+2);

% inicijalizacija izlaznog signala:
nota = zeros(1,duljina);


for i=1:duljina
   % Jednadžba diferencija modela:
   nota(i) = sluc(i) - (C+D)*y(1) - C*D*y(2) +...
             A*D*y(N) + (A+B*D)*y(N+1) + B*y(N+2);
```

```
   %delay linija:
   for j=(length(y)-1):-1:1 % kreni unatrag prema početku
       % pomakni uzorke linije za jedno mjesto unaprijed:
       y(j+1) = y(j);
   end

   % spremi trenutni uzorak na početak linije:
   y(1) = nota(i);
end

% release vremenski segment, za podešavanje trajanja note:
t_R = (length(nota) - NoteOffTime*Fs);

koeficijenti = [exp(-1/0.19), exp(-1/0.105),...
                exp(-1/0.103), exp(-1/0.13)];
ADSR = ADSRgen(t_R, koeficijenti, 'RELEASE');

%kad dođeš do NoteOff vremena, prijeđi u release fazu:
nota(NoteOffTime*Fs:length(nota)-1) = ...
                ADSR.*nota(NoteOffTime*Fs:length(nota)-1);

% zadrži samo signal kojem je amplituda
% različita od 0:
nota = nota(nota ~= 0);

% --> Crtanje signala i zvuk:
%plot(nota); grid on; title('Karplus-Strong algoritam');
%sound(nota, Fs);
```

A.7. Funkcija za Wavetable sintezu

```
function snd = wavetable(broj_note, NoteOffTime)

clc
Fs = 44100;   % frekvencija uzorkovanja

% uploadaj tablice za note klavira:
load note_klavira;

% polje MIDI brojeva nota:
```

```
skupNota = [18, 24, 30, 36, 42, 48, 54, 60, 66,...
            72, 78, 84, 90, 96, 102, 0;
            19, 25, 31, 37, 43, 49, 55, 61, 67,...
            73, 79, 85, 91, 97, 103, 0;
            20, 26, 32, 38, 44, 50, 56, 62, 68,...
            74, 80, 86, 92, 98, 104, 0;
            15, 21, 27, 33, 39, 45, 51, 57, 63,...
            69, 75, 81, 87, 93, 99, 105;
            16, 22, 28, 34, 40, 46, 52, 58, 64,...
            70, 76, 82, 88, 94, 100, 106;
            17, 23, 29, 35, 41, 47, 53, 59, 65,...
            71, 77, 83, 89, 95, 101, 107];

       if broj_note >= 15 && broj_note <= 17
              nota = C0;
       elseif broj_note >= 18 && broj_note <= 23
              nota = Fis0;
       elseif broj_note >= 24 && broj_note <= 29
              nota = C1;
       elseif broj_note >= 30 && broj_note <= 35
              nota = Fis1;
       elseif broj_note >= 36 && broj_note <= 41
              nota = C2;
       elseif broj_note >= 42 && broj_note <= 47
              nota = Fis2;
       elseif broj_note >= 48 && broj_note <= 53
              nota = C3;
       elseif broj_note >= 54 && broj_note <= 59
              nota = Fis3;
       elseif broj_note >= 60 && broj_note <= 65
              nota = C4;
       elseif broj_note >= 66 && broj_note <= 71
              nota = Fis4;
       elseif broj_note >= 72 && broj_note <= 77
              nota = C5;
       elseif broj_note >= 78 && broj_note <= 83
              nota = Fis5;
       elseif broj_note >= 84 && broj_note <= 89
              nota = C6;
       elseif broj_note >= 90 && broj_note <= 95
              nota = Fis6;
       elseif broj_note >= 96 && broj_note <= 101
              nota = C7;
```

```
       elseif broj_note >= 102 && broj_note <= 107
              nota = Fis7;
       end;

% interpolacija uzoraka:
for i = 1:16
    if broj_note == skupNota(1,i),
       snd = nota;  % interpolacija ovdje nije potrebna
       break
    elseif broj_note == skupNota(2,i),
       omjer = 2^(1/12);       % razmak od 1 polutona
       L = length(nota);
       xi = omjer:omjer:L;
       snd = interp1(nota, xi);
       break
    elseif broj_note == skupNota(3,i),
       omjer = 2^(2/12);       % razmak od 2 polutona
       L = length(nota);
       xi = omjer:omjer:L;
       snd = interp1(nota, xi);
       break
    elseif broj_note == skupNota(4,i),
       omjer = 2^(3/12);       % razmak od 3 polutona
       L = length(nota);
       xi = omjer:omjer:L;
       snd = interp1(nota, xi);
       break
    elseif broj_note == skupNota(5,i),
       omjer = 2^(4/12);       % razmak od 4 polutona
       L = length(nota);
       xi = omjer:omjer:L;
       snd = interp1(nota, xi);
       break
    elseif broj_note == skupNota(6,i),
       omjer = 2^(5/12);       % razmak od 5 polutonova
       L = length(nota);
       xi = omjer:omjer:L;
       snd = interp1(nota, xi);
       break
    end
end
```

```
% release vremenski segment, za podešavanje trajanja note:
t_R = (length(nota) - NoteOffTime*Fs);

koeficijenti = [exp(-1/0.19), exp(-1/0.105),...
                exp(-1/0.103), exp(-1/0.13)];
ADSR = ADSRgen(t_R, koeficijenti, 'RELEASE');

%kad dođeš do NoteOff vremena, prijeđi u release fazu
snd(NoteOffTime*Fs:length(snd)-1) = ...
            ADSR.*snd(NoteOffTime*Fs:length(snd)-1);

% zadrži samo signal kojem je amplituda
% različita od 0:
snd = snd(snd ~= 0);

% --> ove naredbe se mogu uključiti za
% --> crtanje signala i zvuk:
%plot(snd)
%soundsc(snd, Fs);
```

A.8. Java klasa za prijamnik MIDI poruka

```
import javax.sound.midi.*;
import static javax.sound.midi.ShortMessage.*;

public class Prijamnik implements Receiver{

   //lokalna varijabla koja pokazuje na spremnik:
   private SpremnikMidiPoruka spremnik;

   //konstruktor metoda:
   public Prijamnik(SpremnikMidiPoruka buffer){
             spremnik = buffer;
   }

   public void send(MidiMessage MidiPoruka, long timeStamp){

        boolean prikazi = true;
        //dohvati MIDI poruku:
        byte[] bajtovi = MidiPoruka.getMessage();
        //maskiranje početnih 4 bita na nulu:
```

```
    int status = bajtovi[0]&0xF0;

    //ispisivanje statusa poruke:
    switch (status){
        case NOTE_ON:
              // podaci o glasnoći note su u trećem bajtu:
              int glasnoca = bajtovi[2];
              if(glasnoca == 0){
                    int kanal = bajtovi[0]&0x0F;
                    bajtovi[0] = (byte)(NOTE_OFF + kanal);
                    System.out.println("NOTE_OFF ");
              }else{
                    System.out.println("NOTE_ON ");
              }
              break;
        case NOTE_OFF:
              System.out.println("NOTE_OFF ");
              break;
        default:
              //nemoj prikazivati active sensing poruke
              //koje uredaj povremeno šalje te ostale poruke:
              prikazi = false;
    }

    //kreiraj MIDI evente iz MIDI poruke i timeStamp-a:
    if (status == NOTE_ON || status == NOTE_OFF){
        MidiEvent event = new MidiEvent(MidiPoruka, timeStamp);
        spremnik.put(event);
    }

    //ispis brojčanih vrijednosti poruka:
    if(prikazi){
        for(int i = 0; i < bajtovi.length; i++){
            int broj = bajtovi[i]&0xFF;
            System.out.println(broj + " ");
        }
    }

    System.out.println();
} // kraj send metode
```

```
    // metoda za zatvaranje prijamnika:
    public void close(){
        // ova metoda ne radi ništa
    }
} // kraj programa
```

A.9. Java klasa za buffer MIDI poruka

```
import javax.sound.midi.*;
import java.util.concurrent.*;

public class SpremnikMidiPoruka {

    // ova klasa radi neprestano, tj. stalno sprema
    // MIDI poruke u spremnik i po potrebi čita iz njega

    //stvori red od 64 MIDI evenata:
    private ArrayBlockingQueue<MidiEvent> spremnik =
                            new ArrayBlockingQueue<MidiEvent>(64);

    private MidiEvent noviEvent;

    public synchronized void put(MidiEvent event){
         try{
               //stavi MIDI event na početak reda:
               spremnik.put(event);

        }catch(InterruptedException e){
                System.out.println(e);
        }
    } // kraj put metode

    public synchronized MidiEvent get(){
        //dohvati MIDI event sa vrha reda:
        noviEvent = spremnik.poll();
        return noviEvent;
    }
}  // kraj programa
```

MODELIRANJE I SINTEZA GLAZBENIH INSTRUMENATA U REALNOM VREMENU KORIŠTENJEM MATLAB-A

Sažetak

Zadatak ovog diplomskog rada bio je obraditi postupke sinteze i modeliranja glazbenih instrumenata u programskoj okolini MATLAB-a. Obrađuju se tri glavne metode sinteze: sinteza frekvencijskom modulacijom (FM sinteza), sinteza valnim tablicama (wavetable sinteza) te fizikalno modeliranje instrumenata pomoću Karplus-Strong algoritma. Opisan je i postupak detektiranja i manipulacije MIDI poruka, koje se šalju u programsku okolinu MATLAB-a pomoću sintesajzera spojenog preko MIDI-USB kabela na računalo. Svi postupci su detaljno analizirani, te je diskutirana i numerička složenost algoritama. Pošto su u radu korištene i MATLAB-ove mogućnosti manipulacije Java klasama i metodama, koje su neophodne za implementaciju primanja MIDI poruka, opisane su sve klase i metode koje su korištene u programskom kodu. U dodacima, na kraju rada, nalazi se i potpuni programski kod.

Ključne riječi: Sinteza instrumenata, FM sinteza, Wavetable sinteza, Fizikalno modeliranje, Karplus-Strong algoritam, MIDI poruke, Sinteza instrumenata u MATLAB-u.

Real-time modelling and synthesis of musical instruments using MATLAB

Abstract

The assignment of this thesis was to discuss and implement different synthesis techniques of musical instruments in MATLAB programming interface. Three major techniques are discussed: Frequency modulation synthesis (FM synthesis), Wavetable synthesis, and Physical modelling using the Karplus-Strong algorithm. Detection and manipulation of MIDI messages is also implemented, which are received via portable musical synthesizer that is connected to the MATLAB interface using a MIDI-USB cable. This process is explained thoroughly along every step of the way. Each algorithm and technique is analyzed in detail, and numerical complexity of the algorithms is also calculated and presented. Since MATLAB has the ability to exploit Java predefined classes and methods, wich are essential if someone wants to implement MIDI message receiving in MATLAB, all the Java classes and methods, that were used in this thesis, are explained. As an appendix, the complete programming code is given.

Keywords: Instrument modelling, FM synthesis, Wavetable synthesis, Physical modelling, Karplus-Strong algorithm, MIDI messages, MATLAB musical instrument modelling.