Helilooming kasutades kahemõõtmelisi reaal- ja kompleksarvulisi kujutisi

Magistritöö

Üliõpilane: Liisi Raudväli
Üliõpilaskood: 231939LAFM
Juhendaja: Dmitri Kartofelev, PhD, Küberneetika instituut, teadur
Õppekava: rakendusfüüsika ja andmeteadus

Tallinn 2025

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Liisi Raudväli

Töö vastab bakalaureusetööle/magistritööle esitatavatele nõuetele.
Juhendaja: Dmitri Kartofelev, PhD

Sisukord

Sissejuhatus

Matemaatika ja muusika on omavahel alati olnud tihedalt seotud. Kuni 17. sajandini käsitleti muusikat lausa kui ühte reaalteadustest. Muusikateooria arendamisse on oma panuse andnud mitmed ajaloost tuntud matemaatikud ja füüsikud – Pythagoras (u. 550—500 eKr) arendas välja algelised heliredelid ja taevaste sfääride harmooniate teooriad, Johannes Kepler (1571–630) kirjeldas planeetide kaugustega vastavuses olevaid muusikalisi harmooniaid, Leonhard Euler (1707–1783) uuris muusikaliste intervallide kooskõla meeldivuse astmeid (Fauvel et al., 2003).

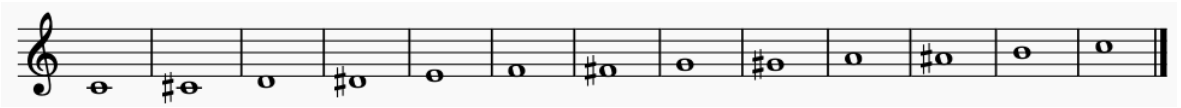
Samal ajal kui matemaatikud on olnud võlutud muusikas leiduvates matemaatilistest seaduspärasustest, on mitmed heliloojad pöördunud just vastupidi matemaatika ja erinevate algoritmide poole, et kasutada neid oma heliloomingus. Heaks näiteks saab siinkohal tuua Johann Sebastian Bachi (1685–1750) fuugad – ühte muusikalist teemat töödeldakse mitut moodi: esitused erinevates helistikes, helivältuste pikendamine või lühendamine, teema peegeldamine horisontaal- või vertikaaltelje suhtes jne. Lisaks võib mainida ka Arnold Schönbergi (1874—1951) kaksteisttoontehnikat ehk dodekafooniat – kaheteistkümne võrdtempereeritud häälestuses heli võrdset kasutamist (Fauvel et al., 2003).

Muusika loomiseks kasutatavateks algoritmideks sobivad ka mitmed mittelineaarsed kujutised, millest mõned on fraktalid. Kuna neid iteratiivseid valemeid on võimalik kasutada tasandil erinevate mustrite loomiseks, siis miks mitte kasutada neid ka muusikaliste mustrite ehk meloodiate loomiseks ning tekitada fraktaalset muusikat.

Antud magistritöö on edasiarendus autori bakalaureusetööst Logistiline kujutis meloodia generaatorina, kus kasutati meloodiate loomiseks ühemõõtmelist reaalarvulist kujutist – logistilist kujutist. Kujutise iteraadid määrasid meloodia nootide helikõrgused ning lõplik muusikateos loodi nootide kestuste määramise ja saate lisamisega autori poolt (Raudväli, 2022). Käesolevas töös kasutatakse kahemõõtmelisi reaal- ja kompleksarvulisi kujutisi, et määrata mitte ainult meloodia helikõrgus kujutiste iteraatide abil, vaid kasutada neid terviklike muusikateoste loomiseks.

Fraktalid ja muusika

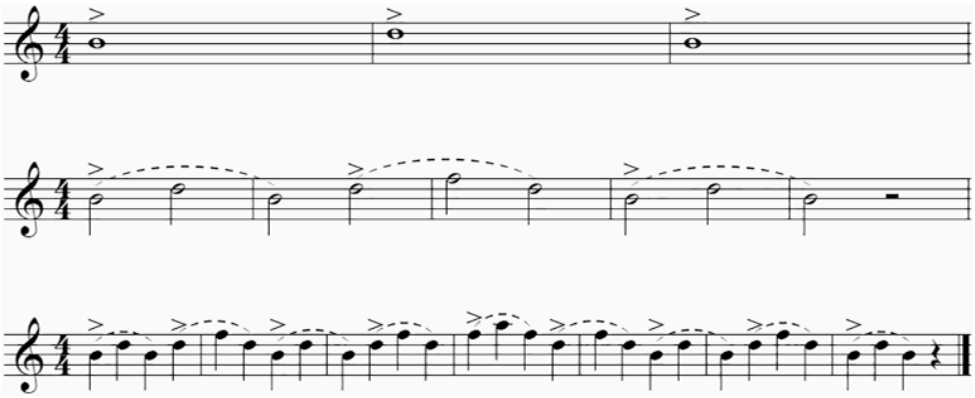
Võrdtempereeritud häälestus ehk mingi muusikalise intervalli võrdseteks osadeks jagamine, mis on levinud just lääne klassikalises muusikas alates 18. sajandist, ei ole lineaarne, vaid logaritmiline. See tähendab, et heliredelis (Joonis 1) kahe järjestikuse helikõrguse vahe on nende helikõrguste sageduste suhe, mis kromaatilises heliredelis on $2^{1/12} \approx 1.059$ (Fauvel et al., 2003).



Joonis 1. Võrdtempereeritud häälestuse kromaatiline heliredel alates C-st ühe oktavi ulatuses

Matemaatilisi seaduspärasusi leidub muusikas veelgi. Helide Fourier spekter muusikas allub 1/*f* astmeseadusele, viidates muusika fraktaalsetele omadustele (Voss et al., 1975). Erinevatele muusikažanritele on arvutatud välja lausa nende žanrite fraktaalsed dimensioonid (Bigerelle et al., 2000). Muusika helikõrguste vaheldumine on enesesarnane ning see tasakaal ennustatavuse ja juhuslikkuse vahel on üks põhjustest, miks muusika kuulamine on inimesele nauditav (Levitin et al., 2012).

1/*f* spekter on iseloomulik ka kaosele – ebaregulaarsele liikumisele, mis on omane teatud tüüpi mittelineaarsetele süsteemidele ning tekib hoolimata determineeritud mudelitest. Kaootilisi protsesse iseloomustavad ka tundlikkus algtingimuste suhtes, ühe või mitme kontrollparameetri muutmisel ilmnevad korrapäraste liikumiste bifurkatsioonid, kaootiliste intervallide esinemine korrapärases liikumises ning atraktori fraktaalne mikrostruktuur faasiruumis (Engelbrecht & Uus, 1993). See mittelineaarsete kujutiste väga mitmekülgne käitumine lubab meil neid kasutada heliloomingus. Fraktalid võivad tekitada väga lihtsaid meloodiaid, näiteks võime konstrueerida midagi Kochi kõvera baasil iteratiivselt (Joonis 2), kuid on võimalik tekitada ka keerukamaid muusikateoseid, mis ongi käesoleva töö eesmärk.



Joonis 2. Kochi kõvera baasil visuaalselt konstrueeritud meloodiad; iga partii vastab ühele iteratsioonile (McDonough & Herczynski, 2023)

Kahemõõtmelised kujutised

Definitsiooni järgi on kujutis *matemaatiline reegel, mis teisendab antud n-mõõtmelise hulga teatud alamhulga punktid teise alamhulga punktideks*. Kujutisi kasutatakse dünaamiliste protsesside modelleerimiseks, eriti kasulikud on need kaootilise käitumise kirjeldamiseks diskreetsetes dünaamilistes süsteemides. Näiteks kasutatakse logistilist kujutist, mis on üks lihtsamaid kujutisi, putukate populatsiooni dünaamika modelleerimiseks (Engelbrecht & Uus, 1993). Järgnevalt tutvume lähemalt töös kasutatud kujutistega: Hénoni kujutis, Mandelbroti hulk, *Burning Ship* ehk Põleva laeva fraktal.

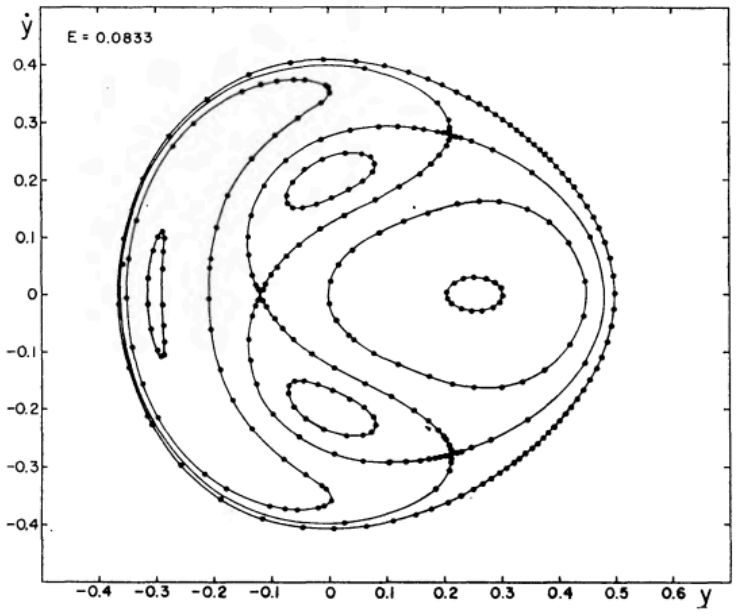
Kahemõõtmeline reaalarvuline kujutis: Hénoni kujutis

Hénoni kujutis on lihtsustatud Poincaré lõike mudel kuulsast Lorenzi süsteemist (vt. seos (2)), ning on antud järgneva valemiga:

$$\begin{cases} x_{n+1} = 1 - ax_n^2 + y_n, \\ y_{n+1} = bx_n \end{cases} \tag{1}$$

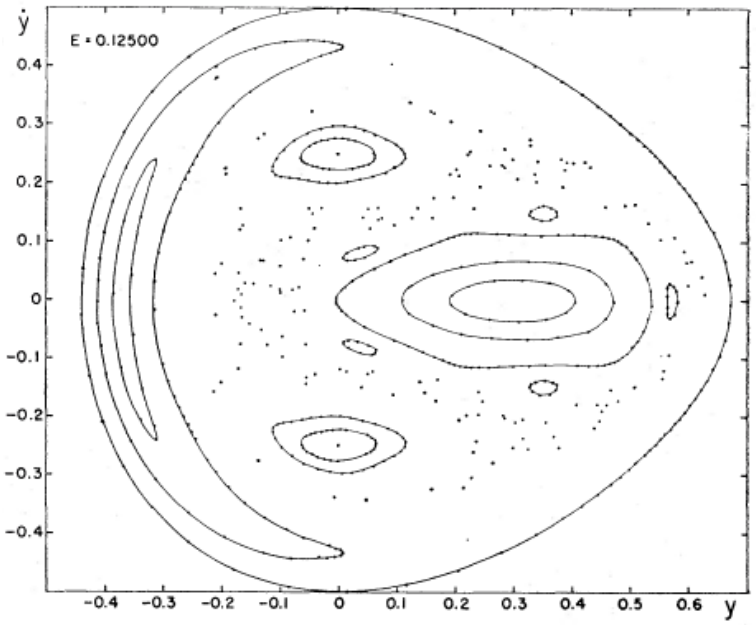
kus x_n ja y_n on põhimuutujad ning a ja b on süsteemi parameetrid (Hénon, 1976).

Michel Hénon (1931–2013) oli prantsuse matemaatik ja astronoom. 1960-ndatel aastatel uuris ja simuleeris ta tähtede orbiite kaugetes galaktikates. Ajaskaalas 200 miljonit aastat ei moodusta orbiidid mitte täiuslikke ellipseid, vaid on pigem kolmemõõtmelise iseloomuga. Selliseid ruumilisi orbiite on keeruline ette kujutada ning visualiseerida, mistõttu Hénon võttis kasutusele järgneva tehnika: galaktika ühte otsa tuleb asetada vertikaaltasand, nii et kõik orbiidid seda läbiksid. Siis tuleb märkida punkt tasandil, kust orbiit seda läbib. Tekkinud lõikepunktid tekitavad kujundeid, mis kirjeldavadki tähe orbiiti – idee, mis on sarnane Poincaré lõikele (Joonis 3).

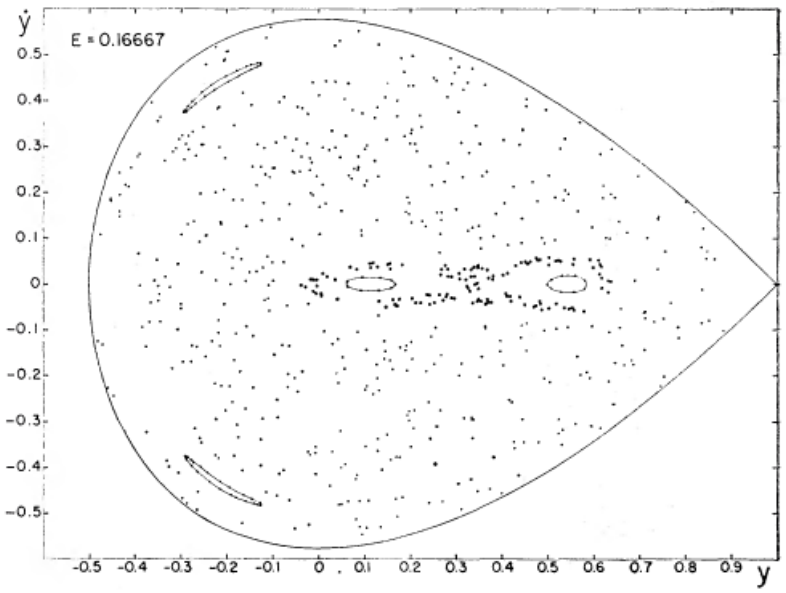


Joonis 3. Tähe orbiidi lõikumine tasandiga (y, $\dot{y}$); süsteemi energia E = 0.0833 (Hénon & Heiles, 1964)

Need orbiidid tundusid alguses kui mitte regulaarsed, siis vähemalt ennustatavad. Kuid kui Hénon ja tema õpilane Carl Heiles (1939–) hakkasid tõstma oma abstraktsete süsteemide energiataset, juhtus midagi huvitavat: tekkisid aina keerulisemad ja keerulisemad struktuurid (Joonis 4) kuni järsku muutusid orbiidid nii ebastabiilseks, et punktid moodustusid justkui juhuslikult mõne üksiku korrapärase piirkonnaga (Joonis 5). Sellest järeldasid Hénon ja Heiles, et niimoodi konstrueeritud pilte suurendades on võimalik leida ebakorrapärastest piirkondades uusi korrapärased piirkondi, nii lõpmatuseni välja. Kuid matemaatiline tõestus antud probleemile osutus liiga keeruliseks ja Hénon liikus edasi teiste probleemide suunas (Hénon & Heiles, 1964)(Gleick, 1987).



Joonis 4. Tähe orbiidi lõikumine tasandiga; süsteemi energia *E* = 0.12500 (Hénon & Heiles, 1964)

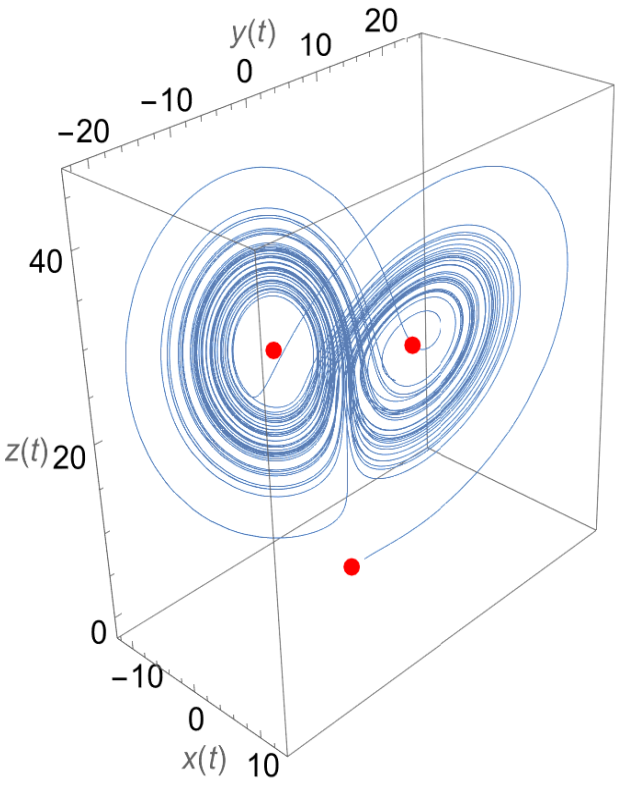


Joonis 5. Tähe orbiidi lõikumine tasandiga; süsteemi energia *E* = 0.16667 (Hénon & Heiles, 1964)

Avastuse juurde tuli Hénon tagasi alles 14 aastat hiljem, kui tutvus Edward Lorenzi (1917–2008) tööga. Edward Lorenz oli ameerika matemaatik ja meteoroloog ning kaoseteooria pioneer. Ta tuletas üllatavalt lihtsa kolmemõõtmelise süsteemi, mis modelleeris atmosfääri konvektsiooni (Lorenz, 1963):

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = rx - y - xz, \\ \dot{z} = xy - bz \end{cases} \tag{2}$$

kus σ, r, $b > 0$ on süsteemi parameetrid. See esmapilgul lihtne deterministlik süsteem sisaldab väga korrapäratut käitumist. Nimelt laias valikus parameetrite σ, r ja b korral on saadavad lahendid ebaregulaarsed, mis ei kordu kunagi täpselt kuid samas jäävad mingisse kindlasse piiratud faasiruumi piirkonda. Joonistades süsteemi trajektoori faasiruumis välja kolmes dimensioonis, oli tulemuseks keerukas trajektoor, mida tunneme Lorenzi atraktori nime all (Joonis 6)(Strogatz, 2015).

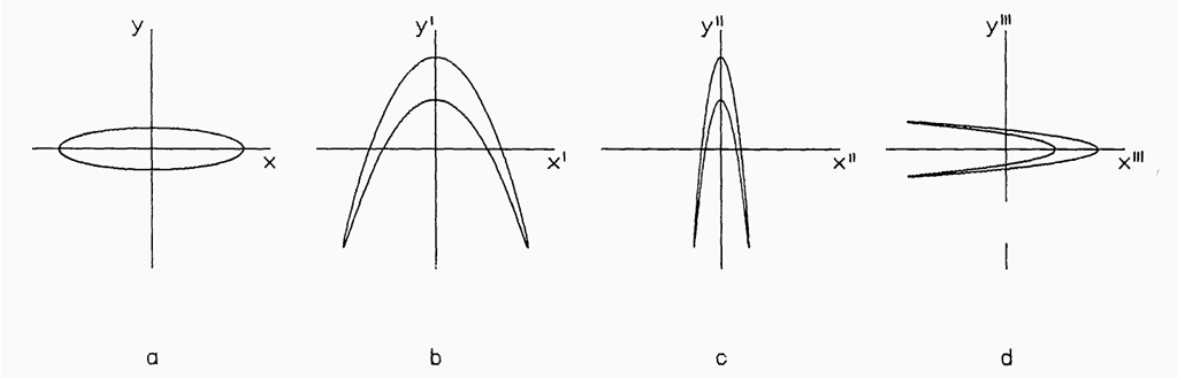


Joonis 6. Lorenzi atraktor ja tema püsipunktid (märgitud punasega); $\sigma = 3$, $r = 26.5$ ja $b = 1$

Lorenzi atraktor on oma iseloomult nn veider atraktor. Atraktorid on punktide hulgad või alamhulgad faasiruumis, kuhu meelevaldselt valitud algtingimusega lahendi trajektoor suundub pärast üleminekuperioodi möödumist. Veidrad atraktorid on atraktorite alamliik – algtingimuste suhtes tundlikud trajektooride hulgad faasiruumis. Veidrad atraktorid tekivad just kaootilise liikumise puhul, mida Lorenz ja Hénon kirjeldasid (Lepik & Engelbrecht, 1999).

Olles inspireeritud Lorenzi tööst, pakkus Hénon välja seletuse, kuidas süsteemi muutused loovad veidra atraktori ning hakkas uurima seda, kuidas atraktori muutkond loob fraktaalse

mikrostruktuuri – miski, mis tekitas nii Lorenzile kui ka teistele füüsikutele raskusi. Erinevalt Lorenzist keskendus Hénon geomeetriale – diferentsiaalvõrrandite, mis on ajas pidevad, asemel kasutas ta diferentsvõrrandeid ehk kujutisi, mis on ajas diskreetsed (Gleick, 1987). Ta oletas, et faasiruumi tuleb venitada, voltida ja painutada nagu tainast. Selleks kasutas ta järgmist teisenduste jada (Joonis 7).



Joonis 7. Ovaalse piirkonna venitamine, voltimine ja algsesse orientatsiooni paigutamine (Hénon, 1976)

Olgu meil mingi ovaalne piirkond mis on *x*-telje suunas piklik (Joonis 7a). Rakendame sellele teisenduse *T'*:

$$T' : x' = x,\ y' = 1 + y - ax^2 \tag{3}$$

Tulemuseks saame paraboolikujuliseks volditud ovaali (Joonis 7b). Surume *x*-telje suunas seda regiooni uuesti kokku teisendusega T'':

$$T'' : x'' = bx',\ y'' = y' \tag{4}$$

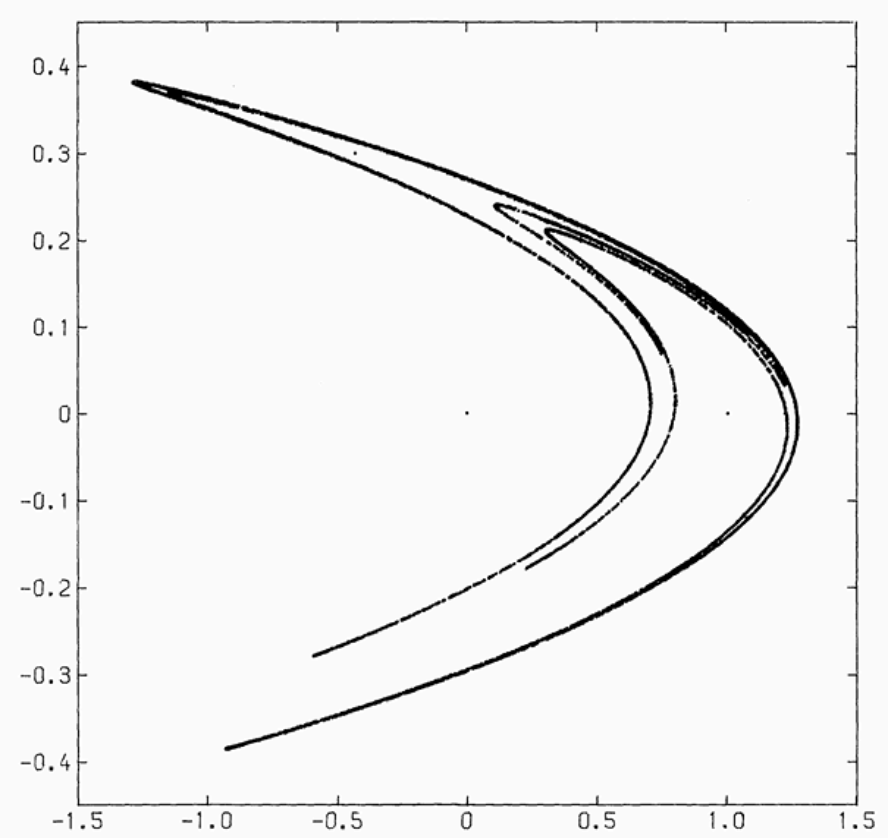
kus $0 < b < 1$. Tulemuseks saame kokkusurutud paraboolid (Joonis 7c). Lõpuks jõuame tagasi *x*-telje suunas oleva kujutise juurde (Joonis 5d) rakendades teisendust *T'''*:

$$T''' : x''' = y'',\ y''' = x'' \tag{5}$$

Hénoni kujutis (Joonis 8) on defineeritud liitteisenduse $T = T'''T''T'$ järgi (Hénon, 1976):

$$T : x_{i+1} = y_i + 1 - ax^2,\ y_{i+1} = bx_i \tag{6}$$

kus lõppolek $x''' \rightarrow x_{i+1}$, algseisund $y \rightarrow y_i$, lõppolek $y''' \rightarrow y_{i+1}$, algseisund $x \rightarrow x_i$.



Joonis 8. Hénoni atraktor; 10 000 iteratsiooni algtingimustega $x_0 = 0$, $y_0 = 0$, $a = 1.4$ ja $b = 0.3$ (Hénon, 1976)

Kahemõõtmeline kompleksarvuline kujutis: Mandelbroti hulk

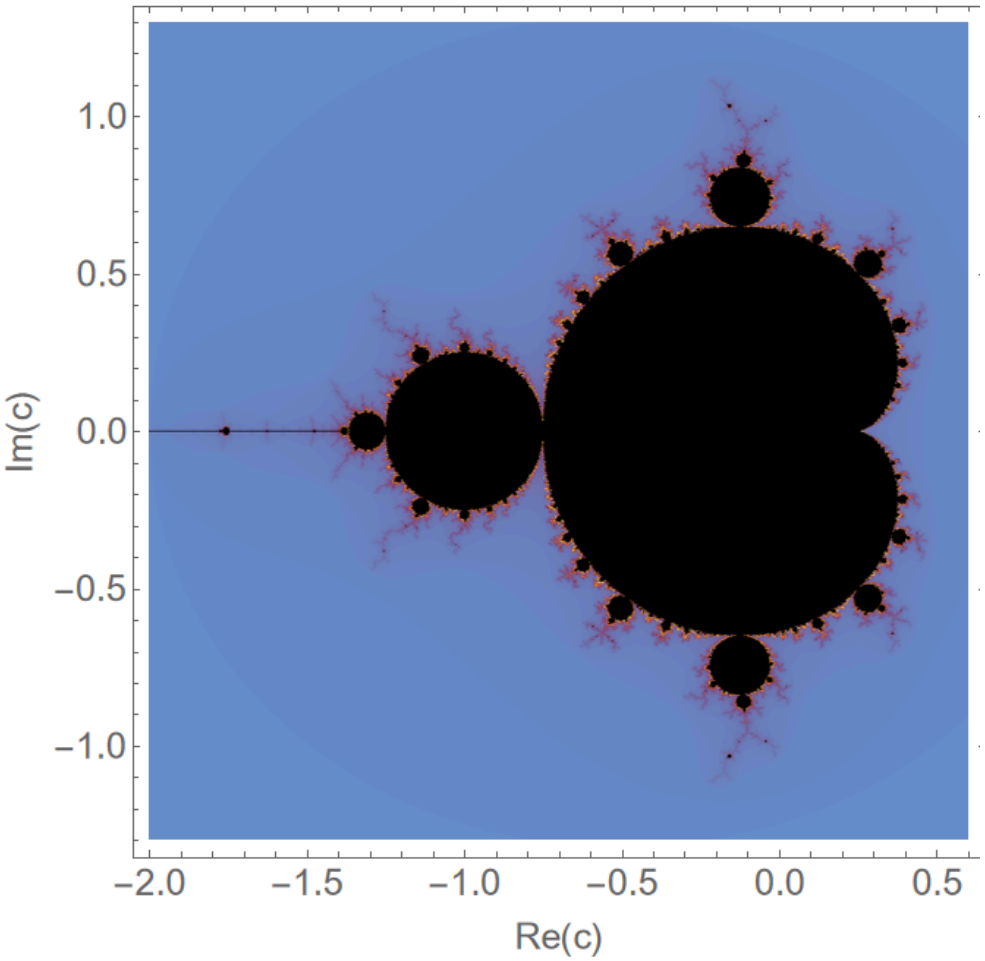
Mandelbroti hulk *M* on üks tuntumaid fraktaleid, kui mitte kõige tuntum. See on defineeritud kui:

$$\begin{cases} z_{n+1} = z_n^2 + c \\ z_0 = 0 \\ c \in M \Leftrightarrow \limsup_{n\to\infty} |z_n| \leq 2 \end{cases}, \quad (7)$$

kus *z*, c on kompleksarvud ja *n* on positiivne täisarv. Mandelbroti hulk moodustab komplekstasandil fraktaalse kujundi (Joonis 9): mustad punktid on arvu *c* väärtused, mille korral iteratsioonid koonduvad kas püsipunktiks või periood-p punktiks ning värvilised punktid on arvu *c* väärtused, mille korral kaob itereeritud arvujada lõpmatusse. Värviskaala näitab lõpmatusse kadumise kiirust (*escape time*). Kaootiline režiim asub kujutise serva lähedal (Kartofelev, 2024)(Engelbrecht & Uus, 1993).

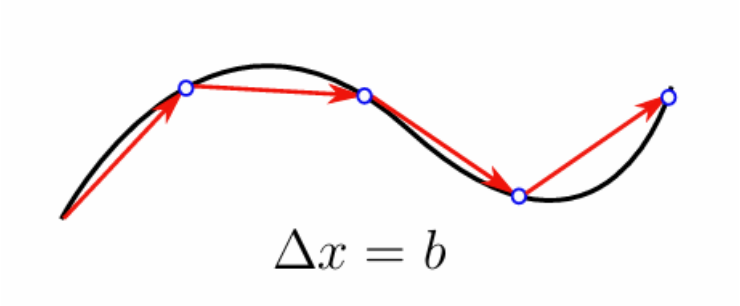
Benoit Mandelbrot (1924–2010) oli poola-prantsuse matemaatik. Ta oli üks esimesi, kes ei rahuldunud Eukleidese geomeetriaga maailma kirjeldamiseks ning otsis midagi uut. „Pilved pole kerad", armastas Mandelbrot öelda. Universum meie ümber ei ole lineaarne, täiuslikke ringe ja sirgjooni kohtab harva. Pigem ümbritseb meid kare, segiaetud keerukus (Gleick, 1987). Üheks ehedaks näiteks sellest on rannajoone paradoks – lühema sammuga mõõtmised ei täpsusta rannajoone pikkust, vaid hoopis pikendavad seda, teoreetiliselt lõpmatuseni. Osakest tervikust

võib käsitleda kui uut tervikut – statistiline enesesarnasus, nagu Mandelbrot seda nimetas (Mandelbrot, 1967).



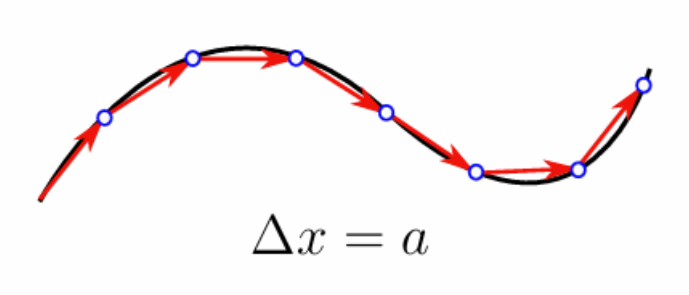
Joonis 9. Mandelbroti hulk

Olgu meil mingi kõver, mille pikkust soovime mõõta. Meil on mõõtmiseks joonlaud ehk mõõtsamm pikkusega *b*. Seda joonlauda kasutades saame mingi lähendatud kõvera pikkuse L_1 (Joonis 10).



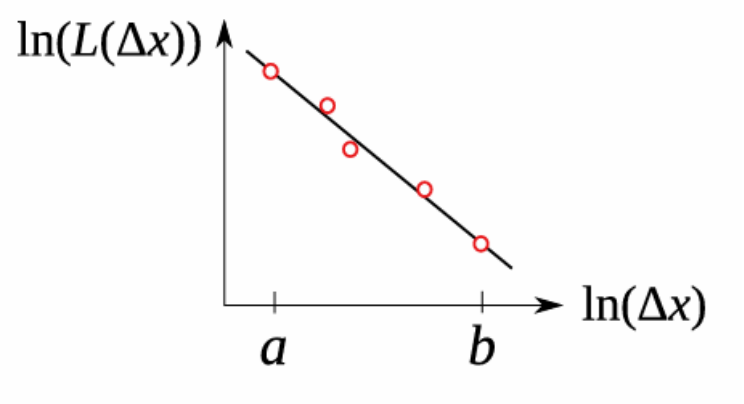
Joonis 10. Kõvera mõõtmine joonlauaga pikkusega *b*; Δ*x* on mõõtmise resolutsioon (Kartofelev, 2024)

Kasutame nüüd sama kõvera pikkuse mõõtmiseks joonlauda pikkusega *a*, kusjuures $a < b$. Saame uue lähendatud kõvera pikkuse L_2 (Joonis 11).



Joonis 11. Kõvera mõõtmine joonlauaga pikkusega *a*; Δ*x* on mõõtmise resolutsioon (Kartofelev, 2024)

Oleks see kõver täiuslikult sile, koonduksid mõõtmistulemused *L* mõõtmise resolutsiooni *Δx* vähendades mingiks täpseks väärtuseks. Kuid enesesarnase, abstraktse fraktaalse kõvera korral seda koondumist ei toimu ning *L* pikeneb lõpmatuseni. Lõpmata väikeste rannajoone osade mõõtmiste liitmisel saaksime justkui lõpmatult pika rannajoone. Kasutades mõõtmistulemustel logaritmilist skaalat, saame huvitava seaduspärasuse (Joonis 12):



Joonis 12. Rannajoone pikkuse ja mõõtühiku logaritmiline sõltuvus ehk astmeseadus *L*(Δ*x*) ja Δ*x* vahel (Kartofelev, 2024)

Mõõtmistulemused asetsevad sirgel tõusuga *D*:

$$D = -\frac{\ln(L(\Delta x))}{\ln(\Delta x)} = \frac{\ln(L(\Delta x))}{\ln(1/\Delta x)} \tag{8}$$

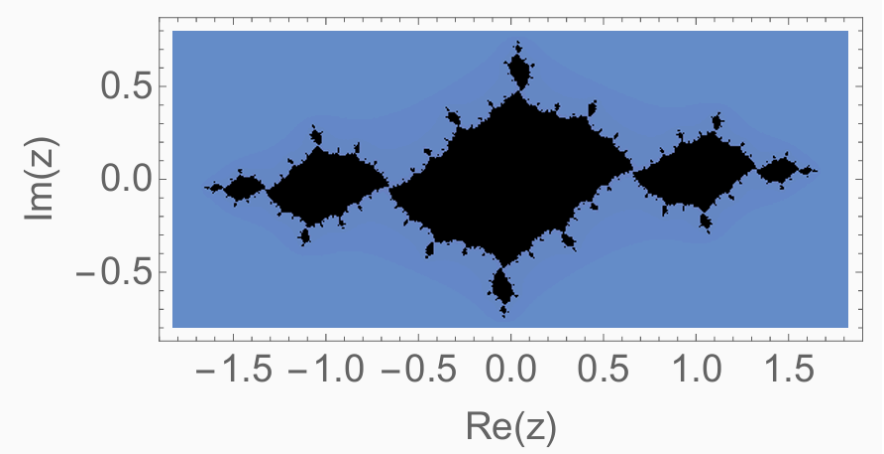
Seega oleme saanud fraktaalseid kõveraid ehk fraktaleid iseloomustava suuruse fraktaalse dimensiooni *D* (Strogatz, 2015). Erinevus Eukleidese ruumis kasutatava topoloogilise dimensiooni D_T ja fraktaalse dimensiooni *D* vahel on see, et topoloogiline dimensioon on alati täisarv (joone dimensioon D_T = 1, pinna dimensioon D_T = 2), kuid fraktaalne dimensioon on üldjuhul murdarv (Suurbritannia rannajoone *D* = 1.3)(Engelbrecht & Uus, 1993).

Erinevaid fraktaalseid kõveraid uurides jõudis Benoit Mandelbrot lõpuks kujutiseni, mida tunneme Mandelbroti hulga nime all. Täpsemalt üritas ta tekitada mingit üldistust Fatou ja Julia hulkadest (Joonis 13), mille avastasid prantsuse matemaatikud Gaston Julia (1893–1978) ja Pierre Fatou (1878–1929) I maailmasõja aegu. Fatou hulk F_c on defineeritud kui:

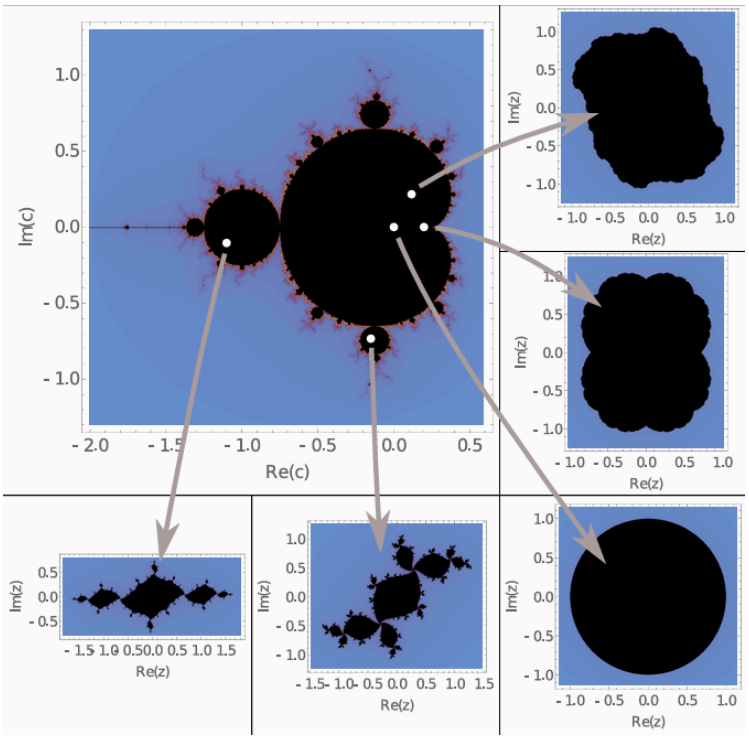
$$\begin{cases} z_{n+1} = z_n^2 + c \\ c = \text{const} = |c| \leq 2 \\ z_0 \in F_c \Leftrightarrow \limsup_{n\to\infty} |z_n| \leq 0.5 + \sqrt{0.25 - |c|} \end{cases}, \quad (9)$$

kus *z*, c on kompleksarvud ja *n* on positiivne täisarv ning Julia hulk J_c on defineeritud kui mittetühja Fatou hulga kompaktne väline piir.

Mandelbrot avastas, et ta suudab komplekstasandil tekitada kujutise, mis oleks justkui nende hulkade kataloog – kujutise, mis saigi tuntuks Mandelbroti hulgana (Joonis 14)(Kartofelev, 2024)(Strogatz, 2015)(Gleick, 1987).



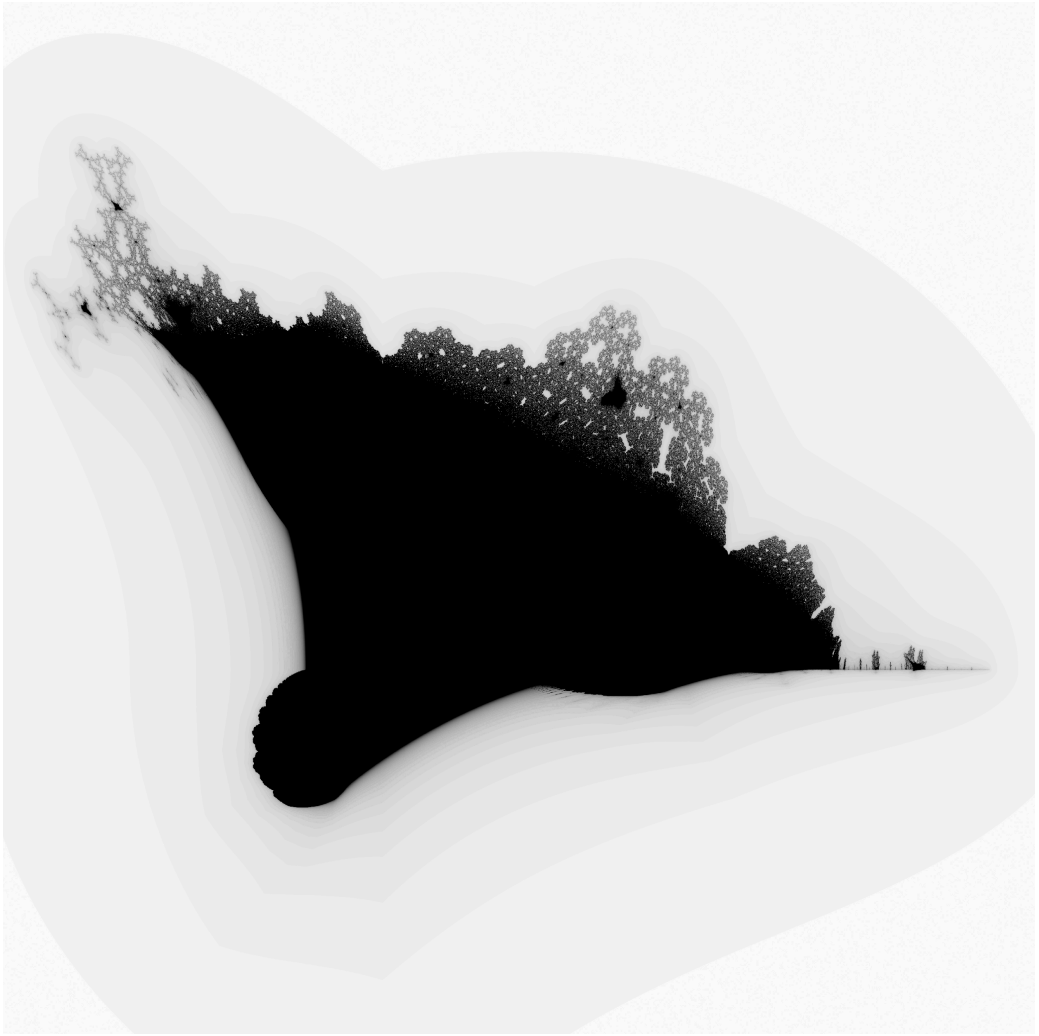
Joonis 13. Fatou hulk ehk täidetud Julia hulk; *c* = – 1.1 – 0.1i (Kartofelev, 2024)



Joonis 14. Mandelbroti hulk kui Fatou hulkade kataloog (Kartofelev, 2024)

Burning Ship fraktal

Mandelbroti hulga analoog on *Burning Ship* („Põlev laev”) fraktal ehk *BS* hulk, mis on oma nime saanud selle järgi, et kujutab põlevat laeva (Joonis 15).

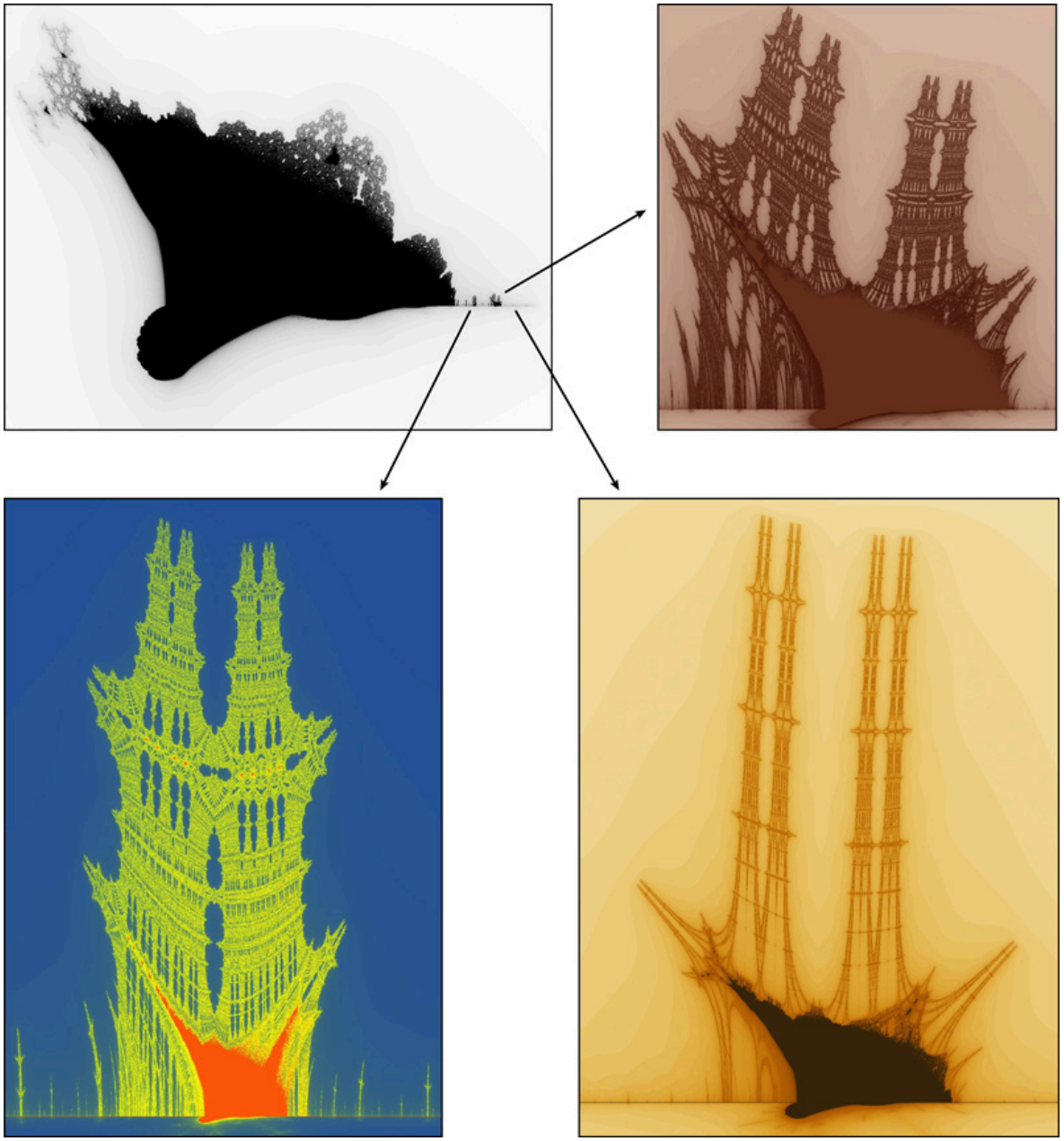


Joonis 15. *Burning Ship* fraktal (Bourke, 1993)

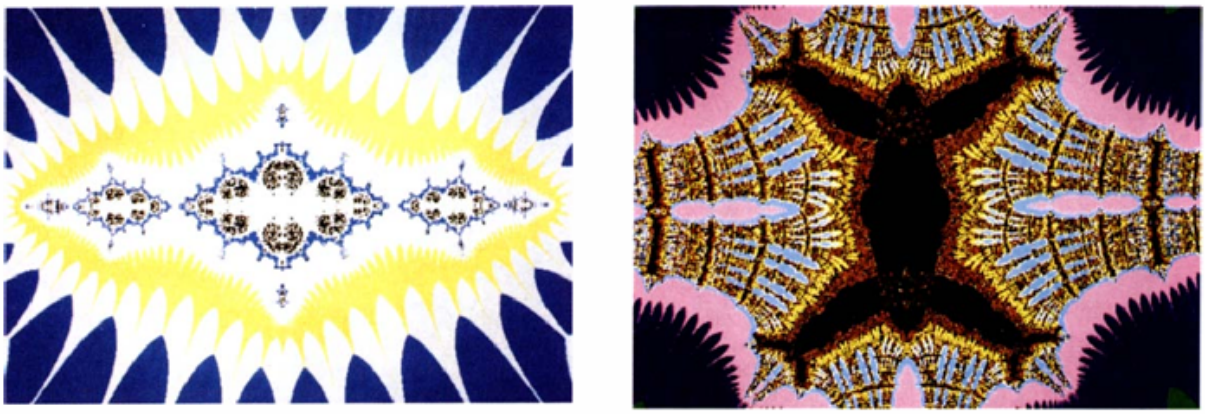
Fraktali genereerisid esimesena Michael Michelitsch ja Otto E. Rössler (1940–) Tübingeni Ülikoolist Saksamaal aastal 1992. Nad asendasid Mandelbroti hulgas kompleksarvu reaal- ja imaginaarosa väärtused nende absoluutväärtustega, saades uue kujutise *Burning Ship*, mis on defineeritud kui:

$$\begin{cases} z_{n+1} = (|\mathrm{Re}\,\{z_n\}| + \mathrm{i}\,|\mathrm{Im}\,\{z_n\}|)^2 + c \\ z_0 = 0 \\ c \in BS \Leftrightarrow \limsup\limits_{n\to\infty} |z_n| \leq 2 \end{cases} \tag{10}$$

Burning Ship fraktal on ilus näide fraktalite poolt loodavatest huvitavatest mustritest (Joonis 16)(Joonis 17). Kaootiline piirkond on *BS* puhul ebamäärasem kui *M* puhul ning kaos on seal segunenud perioodiliste kujutistega (Michelitsch & Rössler, 1992).



Joonis 16. *Burning Ship* fraktal ja tema sabas leiduvad laevade kujutised (Bourke, 1993)



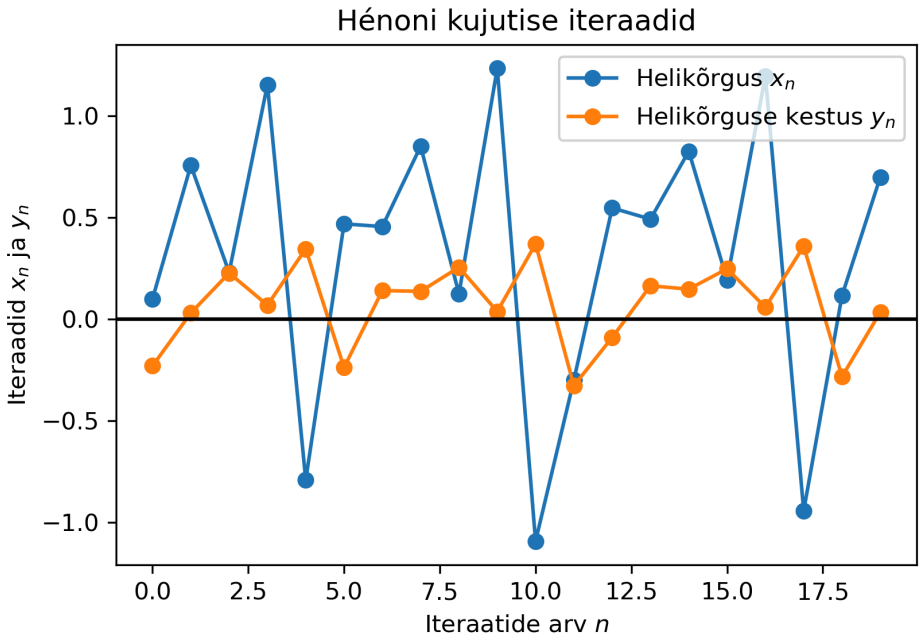
Joonis 17. *Burning Ship* fraktalis leiduvad erinevad kujutised (Michelitsch & Rössler, 1992)

Muusika loomine

Heliloomingu jaoks kirjutasime kaks programmi, üks reaalarvuliste kujutiste jaoks (Lisa 1) ja üks kompleksarvuliste kujutiste jaoks (Lisa 2), mis etteantud parameetrite järgi väljastavad muusikateosed. Programmid on kirjutatud Pythoni programmeerimiskeeles ning muusika on väljastatud MIDI protokollis. MIDI (*Musical Instrument Digital Interface*) protokoll osutus valituks, kuna see võimaldab avada teose nii helifailina multimeediaprogrammis (nt. *Windows Media Player*) kui ka noodikirjana notatsiooniprogrammis (nt. *MuseScore 3*). Programmides kasutatakse Pythoni NumPy teeki matemaatiliste rakenduste jaoks ning MIDIUtil teeki muusikaliste rakenduste jaoks. Alljärgnevalt seletan üksikasjalikumalt lahti programmide tööpõhimõtted.

Programm reaalarvuliste kujutiste jaoks

Reaalarvuliste kujutiste, antud töös Hénoni kujutise korral kasutatakse iteraati x_{n+1} meloodia helikõrguse genereerimiseks ning iteraati y_{n+1} selle helikõrguse kestuse määramiseks (Joonis 18). Saate valisime meelevaldselt ning see on lihtne helistikust sõltuv kolme järjestikuse helikõrguse kordus, saate esimeseks noodiks on helistiku esimene noot.



Joonis 18. Hénoni kujutise iteraadid; $a = 1.4$, $b = 0.3$, $x_0 = 0.1$, $y_0 = -0.23$

Alustuseks tuleb sisestada muusikalised parameetrid heliteose jaoks: heliteose pikkus, taktimõõt, instrument, tempo ja valjusus:

```
# muusika sisend
n = 20  # nootide arv
tml = 3  # taktimoodu lugeja
tmn = 2  # taktimoodu nimetaja ruudus
mitmehaalne = True  # ajas ulekattumised lubatud
channel = 0  # instrumendi kanal
```

```
program = 0  # instrument, vt wiki, 0 klaver
time = 0  # in beats, alguse indeks
time_s = 0  # in beats, saate esimene aeg
tempo = 170  # in BPM
volume = 127  # 0-127, MIDI standard, vol=0 -> paus (rest)
```

Samuti tuleb sisestada matemaatilised parameetrid, mis kirjeldavad kasutatavat kujutist, siinkohal Hénoni kujutist. Kuid on võimalik kasutatada ka teisi kahemõõtmelisi reaalarvulisi kujutisi. Hénoni parameetrid a = 1.4 ja b = 0.3 on klassikalised väärtused, kus antud kujutis on kaootiline ning algväärtused x_0 ja y_0 on valitud meelevaldselt.

```
# kujutise definitsioon ja initsialiseerimine
a = 1.4  # Henoni parameeter
b = 0.3  # Henoni parameeter
x0 = 0.1  # algtingimus kujutise jaoks
y0 = -0.23  # algtingimus kujutise jaoks
x_func = lambda x, y: 1 - a * x**2 + y  # meloodia noodid
y_func = lambda x, y: b * x  # meloodia nootide kestused
```

Viimasena tuleb sisestada saade, mis on salvestatud Pythoni `list`i, ehk määrata milliseid helikõrguseid etteantud helistikust kasutatakse saates. Sisestada tuleb MIDI noodi indeks helistikus:

```
# saate list
saade_kordus_lst = [0, 2, 4]
```

Pärast algtingimuste sisestamist läheme edasi funktsioonide juurde. Programmi on kirjutatud kuus erinevat funktsiooni:

- `saade_index_list(lst, total_n)`,
- `my_map(x0, y0, n, x_func, y_func)`,
- `norm_map_pitch(lst, helistik)`,
- `norm_map_dur(lst, dur)`,
- `midisaator(obj, track, pitch, duration, kattuvad, time, tempo=tempo, tml=tml, tmn=tmn, volume=volume)`,
- `Melody(n, Key_name, mitmehaalne, time, time_s)`.

Esimene funktsioon `saade_index_lst` tekitab meile saate, korrates etteantud noote (sisend `lst`) vastavalt heliteose pikkusele (sisend `total_n`). Vastavalt muusikateooria reeglitele määratakse saate viimaseks noodiks helistiku esimene noot. Funktsioon väljastab muutuja `result`, mis sisaldab nimekirja nootide indeksitest helistikus.

```
# saate definitsioon, kordab etteantud listi
def saade_index_lst(lst, total_n): # kirjutatud python listi jaoks
    repeats = total_n // len(lst)
    if repeats < 1:
        result = lst[:total_n]
    else:
        result = repeats * list(lst)  # np.array käitub teistmoodi
        if len(result) < total_n:
```

```
            missing = total_n - len(result)
            result.extend(lst[:missing])
    result[-1] = 0  # viimane noot == helistiku esimene noot
    return result
```

Teine funktsioon `my_map` tekitab meile reaalarvulise kujutise iteraadid. Funktsiooni sisendiks on kujutise algtingimused `x0` ja `y0`, iteraatide arv `n` ja kujutise võrrandid `x_func` ja `y_func`. Funktsioon väljastab kujutise iteraadid massiivides `x` ja `y`.

```
# kujutise iteraadid
def my_map(x0, y0, n, x_func, y_func):
    x = np.ones(n)*x0  # helikorgus
    y = np.ones(n)*y0  # noodi kestus
    for i in range(n - 1):
        if i == 0:
            x[i + 1] = x_func(x0, y0)  # helikorgus
            y[i + 1] = y_func(x0, y0)  # noodi kestus
        else:
            x[i + 1] = x_func(x[i], y[i])  # helikorgus
            y[i + 1] = y_func(x[i], y[i])  # noodi kestus
    return x, y
```

Kolmas funktsioon `norm_map_pitch` normeerib kujutise iteraadid x_{n+1}, et seada need vastavusse valitud helistiku helikõrgustega, tekitades meloodia esimese komponendi. Funktsioon võtab sisendiks eelneva funktsiooni `my_map` väljundi `x` ning sõnaraamatust valitud helistiku `helistik`. Väljundiks on muutuja `result`, mis sisaldab nimekirja nootide indeksitest helistikus.

```
# MELOODIA normeerimine helistiku helikorgustele
def norm_map_pitch(lst, helistik):
    min_r, max_r = 0, len(helistik) - 1
    minv, maxv = np.min(lst), np.max(lst)
    res = np.round(min_r + (lst - minv) / (maxv - minv) * (max_r
    min_r))
    result = [helistik[int(i)] for i in res[:-1]]
    result.append(helistik[0])
    return result
```

Neljas funktsioon `norm_map_dur` normeerib kujutise iteraadid y_{n+1}, et seada need vastavusse noodi kestustega, tekitades meloodia teise komponendi. Funktsioon võtab sisendiks eelneva funktsiooni `my_map` väljundi `y` ning lubatud nootide kestuste nimekirja `dur`. Väljundiks on muutuja `dur`, mis sisaldab nimekirja indeksitest nootide kestuste nimekirjast.

```
# aja normeerimine, sees antud ajakestuste jaoks
def norm_map_dur(lst, dur):
    min_r, max_r = 0, len(dur) - 1
    minv, maxv = np.min(lst), np.max(lst)
    res = np.round(min_r + (lst - minv) / (maxv - minv) * (max_r
    min_r))
    return [dur[int(i)] for i in res]
```

Viies funktsioon `midisaator` tekitab MIDIfaili objekti *„track"* ehk partii, kuhu kirjutatakse kujutise iteraatide poolt tekitatud meloodia. Sisendiks on programmi alguses defineeritud

muusikalised parameetrid ja muutujad `pitch` ja `duration`, mis on eelnevalt väljastatud normeeritud iteraatide nimekirjad funktsioonide `norm_map_pitch` ja `norm_map_dur` poolt. Väljundiks on heliteose partii.

```
# tekitab MIDI track'i, muudab MIDIFile objekti
def midisaator(obj, track, pitch, program, duration, kattuvad, time,
tempo=tempo, tml=tml, tmn=tmn, volume=volume):
    obj.addTempo(track, time, tempo)
    obj.addTimeSignature(track, time, tml, tmn, 0,
    notes_per_quarter=8)
    # instrumendi muutmiseks, 0-127, MIDI standard
    obj.addProgramChange(track, channel, time, program)

    if track == 0:
        for i in range(n):
            obj.addNote(track, channel, pitch[i], time,
            duration[i], volume)
            if kattuvad:  # noodid voivad kattuda ajas.
                time += 1
           else:
                time += duration[i]
    else:
        for i in range(n):
            obj.addNote(track, channel, pitch[i], time, duration,
            volume)
           time += 1
```

Kuues ja viimane funktsioon `Melody` võtab kokku kõik eelnevalt tehtu ning loob heliteose. Funktsioon võtab sisendiks nootide arvu `n` (heliteose kestus), soovitava helistiku `Key_name`, mitmehäälsust lubava *boolean* parameetri `mitmehaalne` ning partiide algused ajas `time` ja `time_s`. Funktsioonis on defineeritud muutuja `ajastik` võimalike noodi kestustega löökides ning sõnastikud `helistikud` ja `helistikud_s`, kus on kirjas MIDI noodi indeksid vastava helistiku jaoks. Meloodia partii ja saate partii jaoks on kasutuses erinevad sõnastikud, et saade kõlaks oktav madalamalt. Kasutades eelnevalt defineeritud viis funktsiooni, luuakse meloodia ja saate partiid ning pannakse neist kokku terviklik heliteos.

```
def Melody(n, Key_name, mitmehaalne, time, time_s):
    ajastik = [0.5, 1.0, 1.5, 2.0, 2.5]
    helistikud = {
        "c-duur": [60, 62, 64, 65, 67, 69, 71, 72],
        "d-duur": [62, 64, 66, 67, 69, 71, 73, 74],
        "e-duur": [64, 66, 68, 69, 71, 73, 75, 76],
        "f-duur": [65, 67, 69, 70, 72, 74, 76, 77],
        "g-duur": [55, 57, 59, 60, 62, 64, 66, 67],
        "a-duur": [57, 59, 61, 62, 64, 66, 68, 69],
        "b-duur": [59, 61, 63, 64, 66, 68, 70, 71],
        "c-moll": [60, 62, 63, 65, 67, 68, 70, 72],
        "d-moll": [62, 64, 65, 67, 69, 70, 72, 74],
        "e-moll": [64, 66, 67, 69, 71, 72, 74, 76],
        "f-moll": [65, 67, 68, 70, 72, 73, 75, 77],
        "g-moll": [55, 57, 58, 60, 62, 63, 65, 67],
        "a-moll": [57, 59, 60, 62, 64, 65, 67, 69],
```

```
        "b-moll": [59, 61, 62, 64, 66, 67, 69, 71],
    }  # midi helistikud, meloodia jaoks


helistikud_s = {
        "c-duur": [48, 50, 52, 53, 55, 57, 59, 60],
        "d-duur": [50, 52, 54, 55, 57, 59, 61, 62],
        "e-duur": [52, 54, 56, 57, 59, 61, 63, 64],
        "f-duur": [53, 55, 57, 58, 60, 62, 64, 65],
        "g-duur": [43, 45, 47, 48, 50, 52, 54, 55],
        "a-duur": [45, 47, 49, 50, 52, 54, 56, 57],
        "b-duur": [47, 49, 51, 52, 54, 56, 58, 59],
        "c-moll": [48, 50, 51, 53, 55, 56, 58, 60],
        "d-moll": [50, 52, 53, 55, 57, 58, 60, 62],
        "e-moll": [52, 54, 55, 57, 59, 60, 62, 64],
        "f-moll": [53, 55, 56, 58, 60, 61, 63, 65],
        "g-moll": [43, 45, 46, 48, 50, 51, 53, 55],
        "a-moll": [45, 47, 48, 50, 52, 53, 55, 57],
        "b-moll": [47, 49, 50, 52, 54, 55, 57, 59],
    }  # meloodia helistikule vastav saate helistik, oktav madalam

    # meloodia helikorgused ja kestused, track
    helistik = helistikud[Key_name]
    x, y = my_map(x0, y0, n, x_func, y_func)
    pitch_meloodia = norm_map_pitch(x, helistik
    duration = norm_map_dur(y, ajastik)

    # saate helikorgused, track
    helistik_s = helistikud_s[Key_name]
    bassline = saade_index_lst(saade_kordus_lst, n)
    pitch_saade = [helistik_s[bassline[i]] for i in range(n)]
    MyMIDI = MIDIFile(2)

     midisaator(MyMIDI, 0, pitch_meloodia, program, duration,
    mitmehaalne, time)
    midisaator(MyMIDI, 1, pitch_saade, program, 1, mitmehaalne,
    time_s)

    rn = np.random.randint(10000, 99999)

with open(f"Henon_x{x0}_y{y0}_h_{Key_name}_{rn}.mid", "wb") as f:
   MyMIDI.writeFile(f)
```

Programm kompleksarvuliste kujutiste jaoks

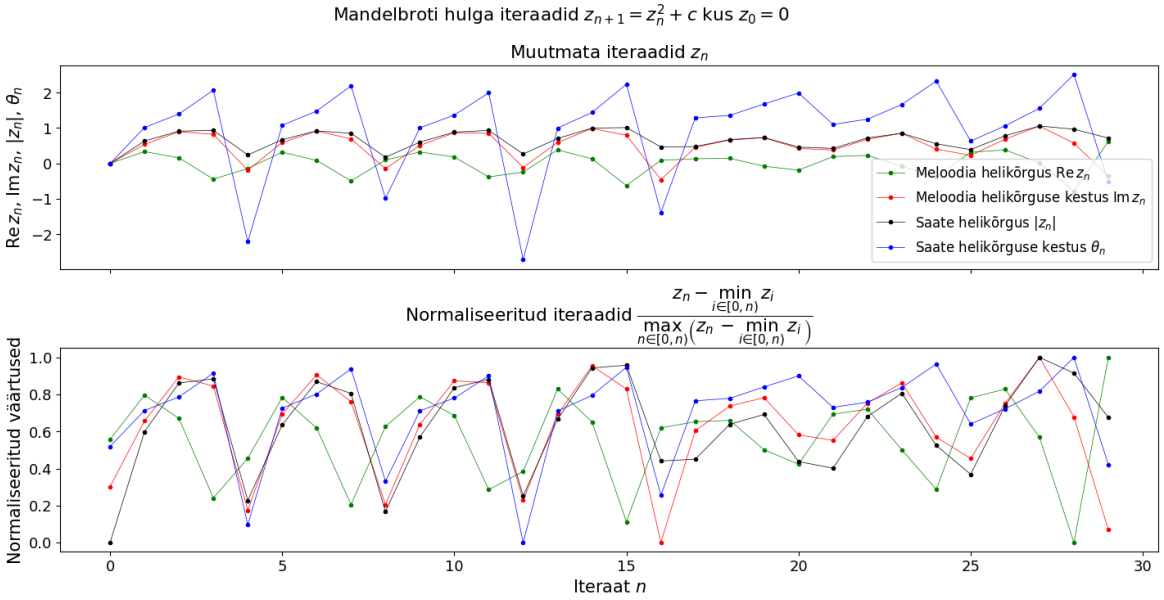
Programm kompleksarvuliste kujutiste jaoks töötab sarnaselt reaalarvuliste kujutiste programmiga. Algselt etteantavad muusikalised parameetrid ning funktsioonid `norm_map_pitch` ja `norm_map_dur` on samad. Kuid kuna meil on tegemist kompleksarvudega, toimub matemaatiliste parameetrite määramine teistmoodi. Lisaks erinevalt reaalarvuliste kujutiste programmist tekitatakse nii meloodia kui ka saate partii kujutise iteraatidest.

Matemaatiliste parameetrite jaoks valime algväärtuseks z_0 = 0 ning fikseerime polünoomi *c* kaootilises regioonis:

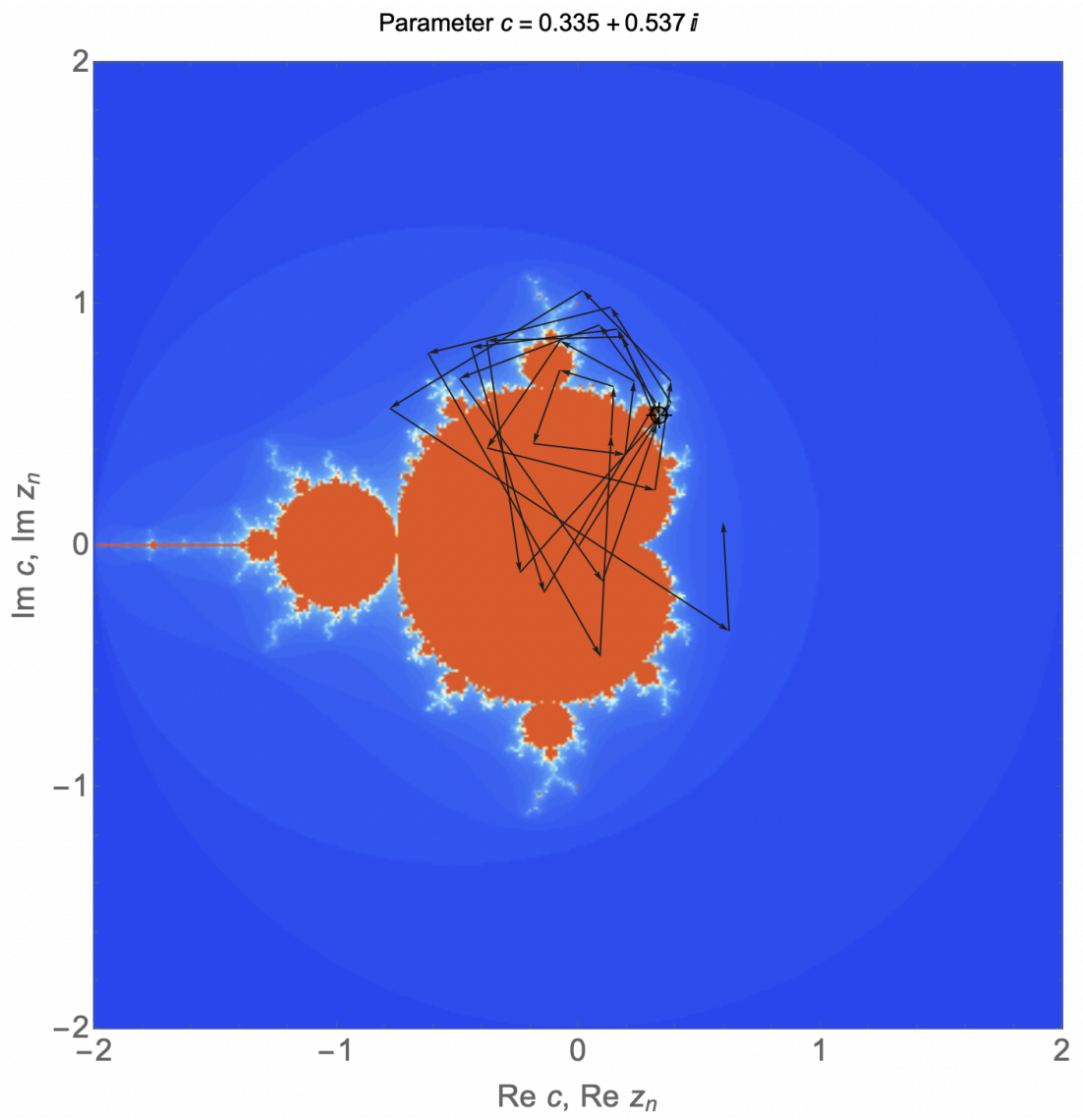
```
c = 0.335 + 0.537j
```

Itereerime läbi kompleksarvulise kujutise (Mandelbroti hulk)(Joonis 19)(Joonis 20):

```
z = np.zeros(n, dtype=complex)  # nullid, sisaldab z[0] = (0+0j).
esc_i = n

for i in range(n - 1):
    z[i + 1] = z[i] ** 2 + c  # Mandelbrot
    if np.abs(z[i + 1]) > 2:
        esc_i = i + 1
        print(
            f"""Premature escape!
            Displayed iterates = {esc_i}.
            Calculated iterates = {esc_i + 1}.
            (start count. from 1.)"""
        )
        break
```



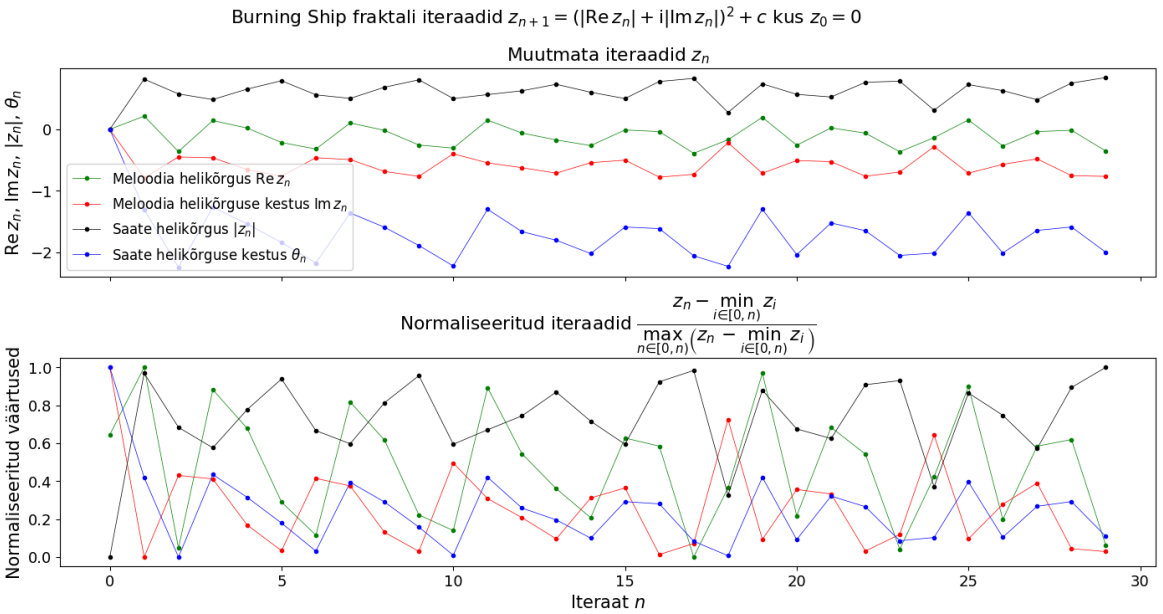
Joonis 19. Mandelbroti hulga iteraadid, kus *c* = 0.335 + 0.537i



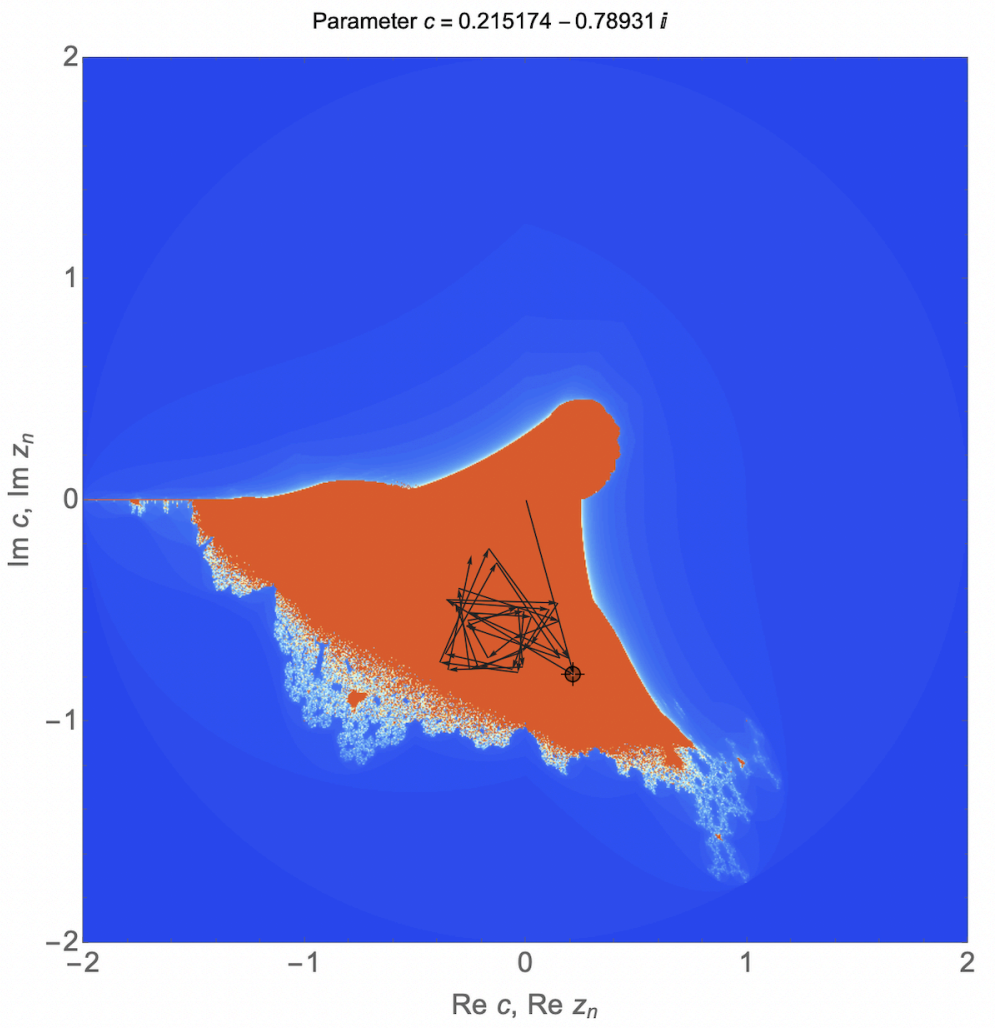
Joonis 20. Mandelbroti hulga iteraadid komplekstasandil, kus *c* = 0.335 + 0.537i

Kasutasin töös kompleksarvuliseks kujutiseks ka *Burning Ship* fraktalit, sellisel juhul näeb iteratsioon välja selline (Joonis 21)(Joonis 22):

```
z = np.zeros(n, dtype=complex)  # nullid, sisaldab z[0] = (0+0j).
esc_i = n

for i in range(n - 1):
    z[i + 1] = (np.abs(z[i])) ** 2 + c  #Burning Ship
    if np.abs(z[i + 1]) > 2:
        esc_i = i + 1
        print(
            f"""Premature escape!
            Displayed iterates = {esc_i}.
            Calculated iterates = {esc_i + 1}.
            (start count. from 1.)"""
        )
        break
```



Joonis 21. *Burning Ship* fraktali iteraadid, kus c = 0.215174 - 0.78931i



Joonis 22. *Burning Ship* fraktali iteraadid komplekstasandil, kus c = 0.215174 - 0.78931i

Saadud iteraatide massiivi `z` kasutame meloodia ja saate partiide nootide defineerimiseks. z_n reaalosa on meloodia noodi helikõrgus ning imaginaarosa meloodia noodi kestus. z_n moodul on saate noodi helikõrgus ning faasinurk saate noodi kestus.

```
# massiivide defineerimine z abil
Zr = z[:esc_i].real  # reaalosa
Zi = z[:esc_i].imag  # imaginaarosa
Zm = np.abs(z[:esc_i])  # moodul
Za = np.angle(z[:esc_i])  # faasinurk
```

Funktsioon `midisaator` on praktiliselt sama mis reaalarvulise kujutise puhul, kuid kuna saade on ka nüüd defineeritud läbi kujutise iteraatide, on muutuja `duration` asemel muutuja `duration[i]` (erinevus toodud välja punasega):

```
if track == 0:
    for i in range(n):
        obj.addNote(track, channel, pitch[i], time, duration[i],
        volume)
        if kattuvad:  # noodid voivad kattuda ajas
            time += 1
        else:
            time += duration[i]
    else:
        for i in range(n):
            obj.addNote(track, channel, pitch[i], time,
            duration[i], volume)
                time += 1
```

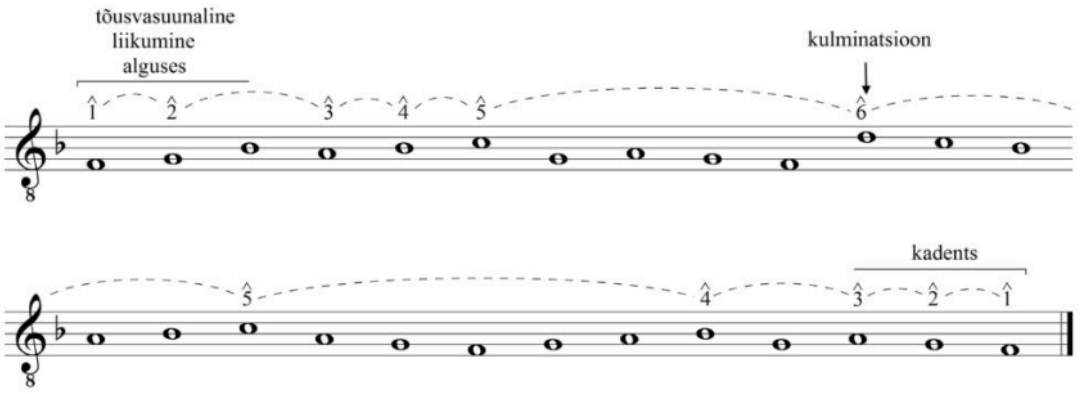
Ka viimase funktsiooni Melody muudatused on seotud sellega, et saade on defineeritud läbi kujutise iteraatide, samuti pole eraldi funktsiooni iteraatide tekitamiseks (erinevused toodud välja punasega):

```
# meloodia helikorgused ja kestused, track
helistik = helistikud[Key_name]
pitch_meloodia = norm_map_pitch(Zr, helistik)
duration = norm_map_dur(Zi, ajastik)

# saate helikorgused, track
helistik_s = helistikud_s[Key_name]
pitch_saade = norm_map_pitch(Zm, helistik_s)
duration_s = norm_map_dur(Za, ajastik)
MyMIDI = MIDIFile(2)

# meloodia
midisaator(MyMIDI, 0, pitch_meloodia, program, duration, mitmehaalne, time)
# saade
midisaator(MyMIDI,  1,  pitch_saade,  program,  duration_s,  mitmehaalne,
time_s)
rn = np.random.randint(10000, 99999)
with open(f"Mandelbrot_c{c}_h_{Key_name}_{rn}.mid", "wb") as f:
        MyMIDI.writeFile(f)
```

Muusika näited ja järeldused

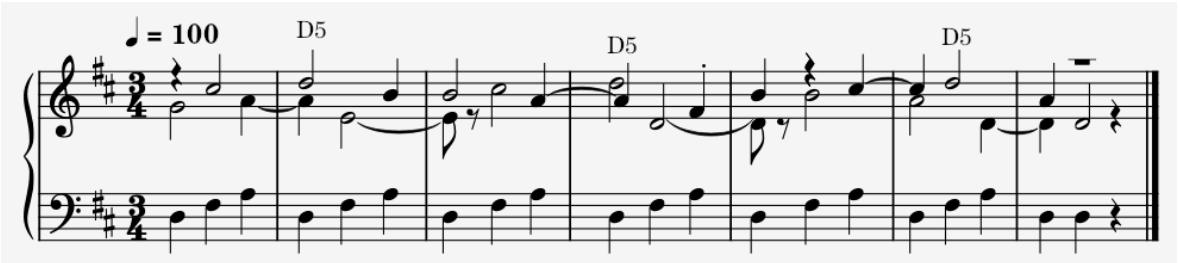
Muusika kui termin ei ole üheselt defineeritud. Üldiselt käsitletakse muusikat kui mingit korrastatud helijärgnevust. Heli tekib ja levib helilainetena ehk õhurõhu väikeste muutustena. Ebaregulaarseid muutuseid tajutakse üldjuhul kui müra, regulaarseid muutuseid aga mingi kindla helikõrgusena. Muusikalise tähenduse saab heli taustsüsteemis, kus see on seotud teiste helidega mingil viisil, mis tekitab kuulajale esteetilise elamuse. Natuke täpsemalt saame me defineerida meloodia, mis on muusikateaduse professor Kerri Kotta järgi *üksteisele järgnevate helide korrastatud ja terviklik jada* (Kotta, 2019).

Fraktali alusel genereeritud meloodiate analüüsiks on varasemalt rakendatud kontrapunkti tehnikat (Raudväli, 2022). Kontrapunkti tehnika põhineb meloodia elementaarkujule ehk *cantus firmus*'ele (Joonis 23) seatud järgnevatest põhimõtetest, mida tuleb komponeerimisel järgida: *cantus firmus* peab moodustuma valdavalt sujuvast astmelisest liikumisest, mis ei tohiks ületada deetsimit ehk 10-astmelist intervalli; *cantus firmus*'el peaks olema selge kaaretaoline kuju, mis algab tõusvasuunalise liikumisega; meloodia kulminatsioon ehk kõige kõrgem heli kõlab reeglina vaid üks kord ning on ette valmistatud järk-järgult ning ei ole meloodia lõpuosas, et oleks aega liikuda lõpukadentsile (Kotta, 2019).



Joonis 23. Cantus firmus ja tema erinevad komponendid (Kotta, 2019)

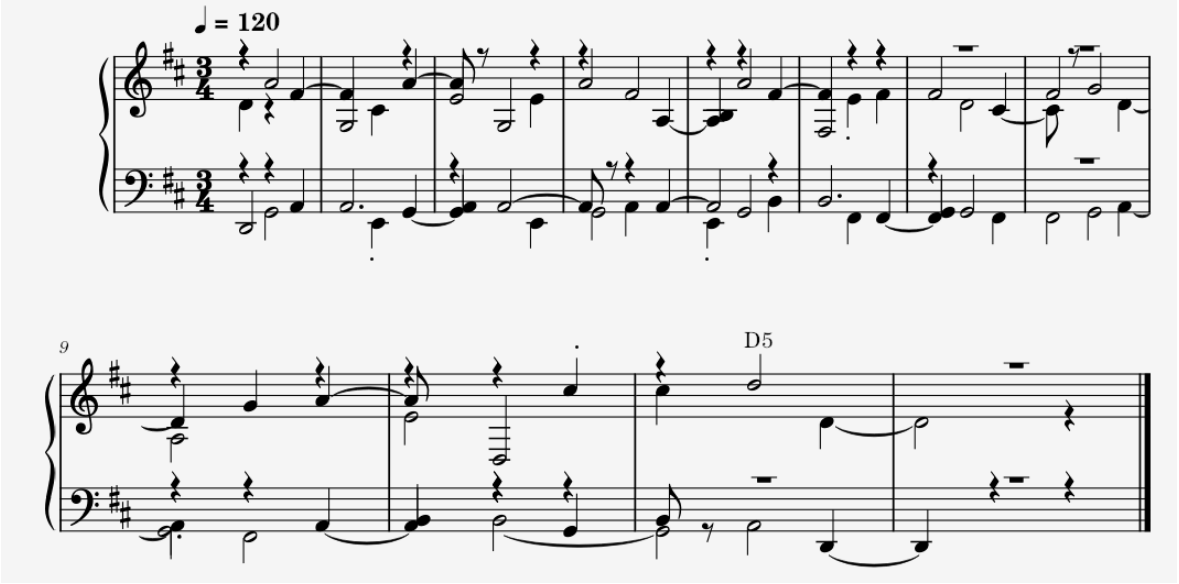
Rakendame kontrapunkti tehnikat kujutiste alusel loodud heliteostele. Alustame Hénoni kujutise alusel loodud teosest (Joonis 24.):



Joonis 24. Hénoni kujutise alusel loodud heliteos; a = 1.4, b = 0.3, x_0 = 0.1, y_0 = – 0.23

Tõepoolest on siin nähtav meloodia kaaretaoline kuju, mis algab liikumisega tõusvas joones ning kõige suurem esinev intervall on 7-astmeline. Kuid kulminatsioon D5 ei kõla mitte ainult ühe korra, vaid kolm korda, ning viimane neist kordadest on üsna meloodia lõpu lähedal.

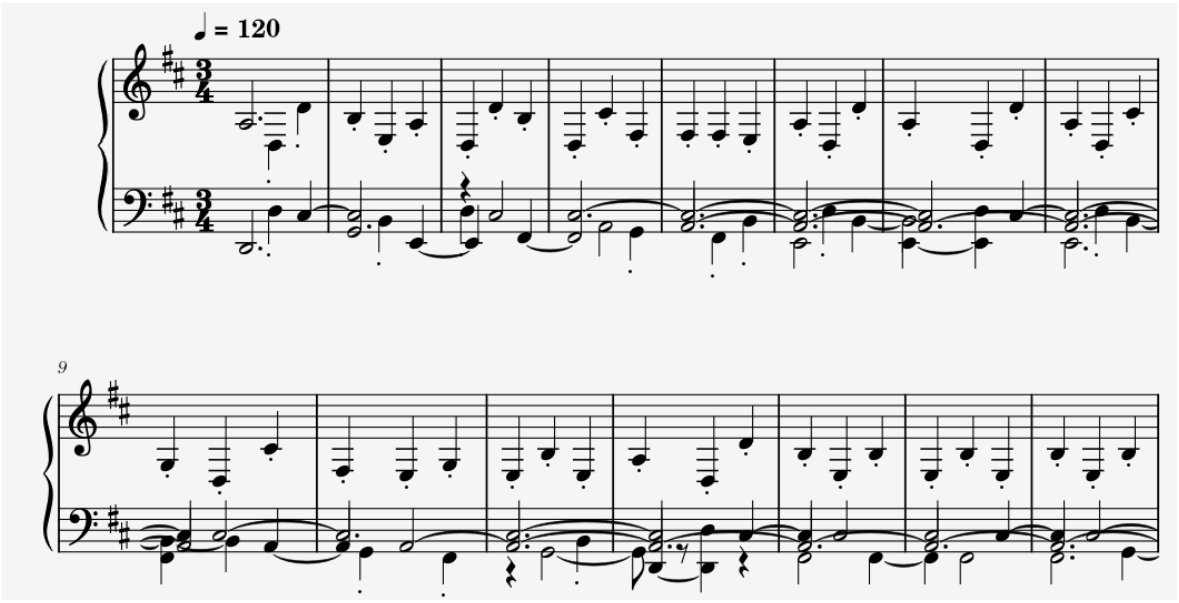
Järgmisena vaatame Mandelbroti hulga alusel loodud teost (Joonis 25):



Joonis 25. Mandelbroti hulga alusel loodud heliteos; *c* = 0.335 + 0.537i

Meloodia alustab siin samuti tõusvas joones liikumist, kuid varieeruvus on suurem ja ei ole nii sujuv, kõige suurem intervall on lausa 13-astmeline. Kulminatsioon D5 see-eest esineb ainult üks kord, siiski jälle üsna meloodia lõpu lähedal.

Lõpetuseks vaatame ka ühte *Burning Ship* fraktali alusel loodud heliteost (Joonis 26):



Joonis 26. *Burning Ship* fraktali alusel loodud heliteos; *c* = 0.215174 - 0.78931i

Siin on kontrapunkti tehnikaga vähe ühist, kuid see-eest kõlab tekkinud teos minu hinnangul kõige huvitavamalt. Kontrapunkti tehnika ei ole ainus komponeerimise tehnika ning kindlasti oleks

võimalik muusikateooria vaatepunktist teha sügavam analüüs antud heliteostele, kuid see läheb antud töö ulatusest välja. Need kolm näidet kolme erineva kujutise põhjal loodud heliteostest siiski illustreerivad kenasti, et fraktaalne muusika ei ole üksluine ja erinevaid teoseid on võimalik luua lõputu hulk, seades piiriks vaid helilooja kujutlusvõime. Algoritmilise muusika maailmas võib tõusuteel olla küll tehisintellekti poolt tekitatud helilooming, kuid fraktaalse muusika võlu ei saa eirata, sest aastakümneid tuntud lihtsad valemid suudavad meid ikka ja jälle millegi uuega üllatada.

Tänuavaldused

Soovin tänada oma magistritöö juhendajat Dmitri Kartofelevit eduka koostöö jätkumise eest, mis viis antud töö valmimiseni. Eriti soovin teda tänada selgete seletuste eest kõikides mittelineaarse dünaamikaga seotud küsimustes ning koodi koostamisse antud panuse eest.

Soovin tänada ka oma „vaimse tervise tiimi" Saku Tervisekeskusest, kuna antud magistritöö kirjutamine sattus minu elus keerulisele perioodile. Aitäh Tiia Raudväli, Triinu Tänavsuu ja Tiina Saks teraapiasessioonide, kasulike nõuannete ja hoolivate sõnade eest.

Lõpetuseks soovin tänada oma kursakaaslasi, kellega koos on võiduka lõpuni tuldud. Otto, Katriin, Heleri, Maike Mona, Artur ja Riido – tänan teid jagatud materjalide, raamatukogus veedetud tundide ja meeldiva seltskonna eest. Ilma teieta oleks need kaks magistrantuuris veedetud aastat palju keerulisemad olnud.

Kasutatud kirjanduse loetelu

Fauvel, J., Flood, R., & Wilson, R. J. (2003). *Music and mathematics: From Pythagoras to Fractals*. Oxford University Press, USA.

Raudväli, L. (2022). *Logistiline kujutis meloodia generaatorina* [bakalaureusetöö, Tallinna Tehnikaülikool].

McDonough, J., & Herczyński, A. (2023). Fractal patterns in music. *Chaos, Solitons & Fractals,* **170,** 113315. https://doi.org/10.1016/j.chaos.2023.113315

Voss, R.F., Clarke, J. (1975). '1/*f* noise' in music and speech. *Nature, 258,* 317-318

Bigerelle, M., Iost, A. (2000). Fractal dimension and classification of music. *Chaos, Solitons & Fractals,* **11**(14)*,* 2179-2192. https://doi.org/10.1016/S0960-0779(99)00137-X

Levitin, D. J., Chordia, P. & Menon, V. (2012). Musical rhythm spectra from Bach to Joplin obey a 1/*f* power law. *Proceedings of the National Academy of Sciences,* **109**(10), 3716-3720. https://doi.org/10.1073/pnas.111382810

Engelbrecht, J. & Uus, A. (1993). *Mittelineaarne dünaamika ja kaos.* Tartu Ülikooli Kirjastus.

Lepik, Ü. & Engelbrecht, J. (1999). *Kaoseraamat.* Teaduste Akadeemia Kirjastus.

Hénon, M. (1976). A two-dimensional mapping with a strange attractor. *Communications in Mathematical Physics,* **50**(1), 69–77. https://doi.org/10.1007/bf01608556

Henon, M., & Heiles, C. (1964). The applicability of the third integral of motion: Some numerical experiments. *The Astronomical Journal,* **69**, 73. https://doi.org/10.1086/109234

Gleick, J. (1987). *Chaos: Making a New Science.* Random House.

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences,* **20**(2), 130–141. https://doi.org/10.1175/1520-0469(1963)020

Strogatz, S. H. (2015). *Nonlinear dynamics and chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. CRC Press.

Mandelbrot, B. (1967). How long is the coast of Britain? Statistical Self-Similarity and Fractional Dimension. *Science,* **156**(3775), 636–638. https://doi.org/10.1126/science.156.3775.636

Kartofelev, D. (2024). *Nonlinear Dynamics: Lecture Notes #14* [PDF] https://www.tud.ttu.ee/web/dmitri.kartofelev/YFX1560.html

Burning Ship Fractal. (n.d.). https://paulbourke.net/fractals/burnship/ Kasutatud 21.05.2025

Michelitsch, M., & Rössler, O. E. (1992). The "burning ship" and its quasi-Julia sets. *Computers & Graphics,* **16**(4), 435–438. https://doi.org/10.1016/0097-8493(92)90032-q

Kotta, K. (2019). *Muusikateooria.* Eesti Muusika- ja Teatriakadeemia. http://mt.ema.edu.ee/

Annotatsioon

Maailm meie ümber on täis fraktaleid. Neid keerulisi kujutisi on võimalik lõputult genereerida lihtsatest valemitest, mis ei paku huvi ainult matemaatikutele, vaid ka heliloojatele. Fraktaalset geomeetriat on täheldatud ka muusikas, kirjandusest on teada, et inimheliloomingu statistiline analüüs viitab erinevate astmeseaduste olemasolule. See omakorda annab potentsiaali vastupidiselt fraktalite kasutamisele muusika genereerimiseks. Käesolev magistritöö on edasiarendus autori bakalaureusetööst *Logistiline kujutis meloodia generaatorina* ning selle eesmärgiks on luua terviklikke muusikateoseid, kasutades kahemõõtmelisi reaal- ja kompleksarvulisi kujutisi. Heliteoste loomiseks kirjutati programmid, kus kujutise iteraatidele seati vastavusse helikõrgused ja helikõrguste kestused, luues terviklikud meloodiad, millest moodustuvad lõpuks heliteosed. Kolme erineva kujutise – Hénoni kujutise, Mandelbroti hulga ja *Burning Ship* fraktali kasutamisel oli tulemuseks kolm üksteisest märgatavalt erinevat ja ainulaadset lühikest heliteost.

Abstract

The world around us is full of fractals. Those complicated shapes can be endlessly generated from simple equations that not only interest mathematicians, but also music composers. Fractal geometry has also been observed in music, namely that it is known from the literature that statistical analysis of human musical compositions indicates the existence of various power laws. This in turn provides the potential to reverse the use of fractals to generate music. This master's thesis is a further development of the author's bachelor's thesis *Logistic Map as Melody Generator* and the goal of it is to create complete musical works using two-dimensional real and complex maps. Programs were created to generate musical compositions, where the iterates of the map were assigned to pitch and duration, creating complete melodies that make up a musical composition. Using three different maps – Hénon map, Mandelbrot set and Burning Ship fractal, the results were three noticeably distinct and unique short musical compositions.

Lisad

Lisa 1. Programm reaalarvuliste kujutiste jaoks

```
import numpy as np
from midiutil import MIDIFile

# muusika sisend
n = 20  # nootide arv
tml = 3  # taktimoodu lugeja
tmn = 2  # taktimoodu nimetaja ruudus
mitmehaalne = True  # ajas ulekattumised lubatud
channel = 0  # instrumendi kanal
program = 0  # instrument, vt wiki, 0 klaver
time = 0  # in beats, alguse indeks
time_s = 0  # in beats, saate esimene aeg
tempo = 170  # in BPM
volume = 127  # 0-127, MIDI standard, vol=0 -> paus (rest)
# kujutise definitsioon ja initsialiseerimine
a = 1.4  # Henoni parameeter
b = 0.3  # Henoni parameeter
x0 = 0.1  # algtingimus kujutise jaoks
y0 = -0.23  # algtingimus kujutise jaoks
x_func = lambda x, y: 1 - a * x**2 + y  # meloodia noodid
y_func = lambda x, y: b * x  # meloodia nootide kestused

# saate list
saade_kordus_lst = [0, 2, 4]  # midi noodi indeks helistikus

# saate definitsioon, kordab etteantud listi
def saade_index_lst(lst, total_n):  # kirjutatud python listi jaoks
    repeats = total_n // len(lst)
    if repeats < 1:
        result = lst[:total_n]
    else:
        result = repeats * list(lst)  # np.array käitub teistmoodi
        if len(result) < total_n:
            missing = total_n - len(result)
            result.extend(lst[:missing])
    result[-1] = 0  # viimane noot == helistiku esimene noot
    return result

# kujutise iteraadid
def my_map(x0, y0, n, x_func, y_func):
    x = np.ones(n)*x0  # helikorgus
    y = np.ones(n)*y0  # noodi kestus
    for i in range(n - 1):
        if i == 0:
            x[i + 1] = x_func(x0, y0)  # helikorgus
            y[i + 1] = y_func(x0, y0)  # noodi kestus
        else:
            x[i + 1] = x_func(x[i], y[i])  # helikorgus
            y[i + 1] = y_func(x[i], y[i])  # noodi kestus
```

```
    return x, y

# MELOODIA normeerimine helistiku helikorgustele
def norm_map_pitch(lst, helistik):  # tootab ka kui lst on python list
    min_r, max_r = 0, len(helistik) - 1
    minv, maxv = np.min(lst), np.max(lst)
    res = np.round(min_r+(lst-minv) / (maxv-minv) * (max_r-min_r))
    result = [helistik[int(i)] for i in res[:-1]]
    result.append(helistik[0])
    return result

# aja normeerimine, sees antud ajakestuste jaoks
def norm_map_dur(lst, dur):  # midiutil: dur ei saa olla 0; tootab ka kui
lst on python list
    min_r, max_r = 0, len(dur) - 1  # kokku kestust
    minv, maxv = np.min(lst), np.max(lst)
    res = np.round(min_r+(lst-minv) / (maxv-minv) * (max_r-min_r))
    return [dur[int(i)] for i in res]

# tekitab MIDI track'i, muudab MIDIFile objekti
def midisaator(
    obj,
    track,
    pitch,
    program,
    duration,
    kattuvad,
    time,
    tempo=tempo,
    tml=tml,
    tmn=tmn,
    volume=volume
):
    obj.addTempo(track, time, tempo)
    obj.addTimeSignature(track, time, tml, tmn, 0, notes_per_quarter=8)

    # instrumendi muutmiseks, 0-127, MIDI standard
    obj.addProgramChange(track, channel, time, program)

    if track == 0:
        for i in range(n):
          obj.addNote(track, channel, pitch[i], time, duration[i],
          volume)
          if kattuvad:  # noodid voivad kattuda ajas
              time += 1
          else:
              time += duration[i]
    else:
        for i in range(n):
          obj.addNote(track, channel, pitch[i], time, duration, volume)
          time += 1

def Melody(n, Key_name, mitmehaalne, time, time_s):
```

```
      ajastik = [0.5, 1.0, 1.5, 2.0, 2.5]  # voimalikud nootide kestused
meloodias
    helistikud = {
        "c-duur": [60, 62, 64, 65, 67, 69, 71, 72],
        "d-duur": [62, 64, 66, 67, 69, 71, 73, 74],
        "e-duur": [64, 66, 68, 69, 71, 73, 75, 76],
        "f-duur": [65, 67, 69, 70, 72, 74, 76, 77],
        "g-duur": [55, 57, 59, 60, 62, 64, 66, 67],
        "a-duur": [57, 59, 61, 62, 64, 66, 68, 69],
        "b-duur": [59, 61, 63, 64, 66, 68, 70, 71],
        "c-moll": [60, 62, 63, 65, 67, 68, 70, 72],
        "d-moll": [62, 64, 65, 67, 69, 70, 72, 74],
        "e-moll": [64, 66, 67, 69, 71, 72, 74, 76],
        "f-moll": [65, 67, 68, 70, 72, 73, 75, 77],
        "g-moll": [55, 57, 58, 60, 62, 63, 65, 67],
        "a-moll": [57, 59, 60, 62, 64, 65, 67, 69],
        "b-moll": [59, 61, 62, 64, 66, 67, 69, 71],
    }  # midi helistikud, meloodia jaoks

    helistikud_s = {
        "c-duur": [48, 50, 52, 53, 55, 57, 59, 60],
        "d-duur": [50, 52, 54, 55, 57, 59, 61, 62],
        "e-duur": [52, 54, 56, 57, 59, 61, 63, 64],
        "f-duur": [53, 55, 57, 58, 60, 62, 64, 65],
        "g-duur": [43, 45, 47, 48, 50, 52, 54, 55],
        "a-duur": [45, 47, 49, 50, 52, 54, 56, 57],
        "b-duur": [47, 49, 51, 52, 54, 56, 58, 59],
        "c-moll": [48, 50, 51, 53, 55, 56, 58, 60],
        "d-moll": [50, 52, 53, 55, 57, 58, 60, 62],
        "e-moll": [52, 54, 55, 57, 59, 60, 62, 64],
        "f-moll": [53, 55, 56, 58, 60, 61, 63, 65],
        "g-moll": [43, 45, 46, 48, 50, 51, 53, 55],
        "a-moll": [45, 47, 48, 50, 52, 53, 55, 57],
        "b-moll": [47, 49, 50, 52, 54, 55, 57, 59],
    }  # meloodia helistikule vastav saate helistik, oktav madalam

    # meloodia helikorgused ja kestused, track
    helistik = helistikud[Key_name]  # list midi noodi numbritega
    x, y = my_map(x0, y0, n, x_func, y_func)  # toored kujutise iteraadid
    pitch_meloodia = norm_map_pitch(x, helistik)  # x normaliseeritud
    # noodi pikkused, y normaliseeritud
    duration = norm_map_dur(y, ajastik)

    # saate helikorgused, track
    helistik_s = helistikud_s[Key_name]  # list midi noodi numbritega
    #saade, list indeks in list
    bassline = saade_index_lst(saade_kordus_lst, n)
    # midi noodi number
    pitch_saade = [helistik_s[bassline[i]] for i in range(n)]
    MyMIDI = MIDIFile(2)

    # meloodia
    midisaator(MyMIDI, 0, pitch_meloodia, program, duration, mitmehaalne,
```

```
    time)
    # saade
    midisaator(MyMIDI, 1, pitch_saade, program, 1, mitmehaalne, time_s)

    # suvaline 5kohaline number faili nimeks
    rn = np.random.randint(10000, 99999)

    with open(f"Henon_x{x0}_y{y0}_h_{Key_name}_{rn}.mid", "wb") as f:
        MyMIDI.writeFile(f)


if __name__ == "__main__":
    Melody(n, "d-duur", mitmehaalne, time, time_s)
```

Lisa 2. Programm kompleksarvuliste kujutiste jaoks

```
import numpy as np
from midiutil import MIDIFile

# muusika sisend
n = 100  # nootide arv
tml = 3  # taktimoodu lugeja
tmn = 2  # taktimoodu nimetaja ruudus
mitmehaalne = True  # ajas ulekattumised lubatud
channel = 0  # instrumendi kanal
program = 8  # instrument, vt wiki, 0 klaver
time = 0  # in beats, alguse indeks
time_s = 0  # in beats, saate esimene aeg
tempo = 120  # in BPM
volume = 127  # 0-127, MIDI standard, vol=0 -> paus (rest)

# algvaartused ja polunoomi parandamine
c = -1.85 - 0.1j  # vali c kaootilises regioonis: z_{n+1} = z_{n}^2 + c

#####################################################
# Mandelbrot voi Burning Ship
z = np.zeros(n, dtype=complex)  # nullid, sisaldab z[0] = (0+0j).
esc_i = n

for i in range(n - 1):
    z[i + 1] = (np.abs(z[i])) ** 2 + c
    if np.abs(z[i + 1]) > 2:
        esc_i = i + 1
        print(
            f"""Premature escape!
            Displayed iterates = {esc_i}.
            Calculated iterates = {esc_i + 1}.
            (start count. from 1.)"""
        )
        break

# massiivide defineerimine z abil
Zr = z[:esc_i].real  # reaalosa; meloodia helikorgus
Zi = z[:esc_i].imag  # imaginaarosa; meloodia kestus
Zm = np.abs(z[:esc_i])  # moodul, absvaartus; saate helikorgus
Za = np.angle(z[:esc_i])  # faasinurk vahemik (-pi, pi]; saate kestus

# MELOODIA normeerimine helistiku helikõrgustele
def norm_map_pitch(lst, helistik):  # tootab ka kui lst on python list
    min_r, max_r = 0, len(helistik) - 1
    minv, maxv = np.min(lst), np.max(lst)
    res = np.round(min_r+(lst-minv) / (max-minv) * (max_r-min_r))
    result = [helistik[int(i)] for i in res[:-1]]
    return np.append(result, helistik[0])
```

```
# aja normeerimine, sees antud ajakestuste jaoks
def norm_map_dur(lst, dur):
    # midiutil: dur ei saa olla 0; tootab ka kui lst on python list
    min_r, max_r = 0, len(dur) - 1  # kokku kestust
    minv, maxv = np.min(lst), np.max(lst)
    res = np.round(min_r+(lst-minv) / (maxv-minv) * (max_r-min_r))
    return [dur[int(i)] for i in res]

# tekitab MIDI track'i, muudab MIDIFile objekti
def midisaator(
    obj,
    track,
    pitch,
    program,
    duration,
    kattuvad,
    time,
    tempo=tempo,
    tml=tml,
    tmn=tmn,
    volume=volume
):
    obj.addTempo(track, time, tempo)
    obj.addTimeSignature(track, time, tml, tmn, 0, notes_per_quarter=8)
    # instrumendi muutmiseks, 0-127, MIDI standard
    obj.addProgramChange(track, channel, time, program)

    if track == 0:
        for i in range(n):
            obj.addNote(track, channel, pitch[i], time, duration[i],
            volume)
            if kattuvad:  # noodid voivad kattuda ajas
                time += 1
            else:
                time += duration[i]
    else:
        for i in range(n):
            obj.addNote(track, channel, pitch[i], time, duration[i],
            volume)
            time += 1

def Melody(n, Key_name, mitmehaalne, time, time_s):
    ajastik = [0.5, 1.0, 1.5, 2.0, 2.5]  # voimalikud nootide kestused
meloodias
    helistikud = {
        "c-duur": [60, 62, 64, 65, 67, 69, 71, 72],
        "d-duur": [62, 64, 66, 67, 69, 71, 73, 74],
        "e-duur": [64, 66, 68, 69, 71, 73, 75, 76],
        "f-duur": [65, 67, 69, 70, 72, 74, 76, 77],
        "g-duur": [55, 57, 59, 60, 62, 64, 66, 67],
        "a-duur": [57, 59, 61, 62, 64, 66, 68, 69],
        "b-duur": [59, 61, 63, 64, 66, 68, 70, 71],
        "c-moll": [60, 62, 63, 65, 67, 68, 70, 72],
```

```
        "d-moll": [62, 64, 65, 67, 69, 70, 72, 74],
        "e-moll": [64, 66, 67, 69, 71, 72, 74, 76],
        "f-moll": [65, 67, 68, 70, 72, 73, 75, 77],
        "g-moll": [55, 57, 58, 60, 62, 63, 65, 67],
        "a-moll": [57, 59, 60, 62, 64, 65, 67, 69],
        "b-moll": [59, 61, 62, 64, 66, 67, 69, 71],
    }  # midi helistikud, meloodia jaoks

    helistikud_s = {
        "c-duur": [48, 50, 52, 53, 55, 57, 59, 60],
        "d-duur": [50, 52, 54, 55, 57, 59, 61, 62],
        "e-duur": [52, 54, 56, 57, 59, 61, 63, 64],
        "f-duur": [53, 55, 57, 58, 60, 62, 64, 65],
        "g-duur": [43, 45, 47, 48, 50, 52, 54, 55],
        "a-duur": [45, 47, 49, 50, 52, 54, 56, 57],
        "b-duur": [47, 49, 51, 52, 54, 56, 58, 59],
        "c-moll": [48, 50, 51, 53, 55, 56, 58, 60],
        "d-moll": [50, 52, 53, 55, 57, 58, 60, 62],
        "e-moll": [52, 54, 55, 57, 59, 60, 62, 64],
        "f-moll": [53, 55, 56, 58, 60, 61, 63, 65],
        "g-moll": [43, 45, 46, 48, 50, 51, 53, 55],
        "a-moll": [45, 47, 48, 50, 52, 53, 55, 57],
        "b-moll": [47, 49, 50, 52, 54, 55, 57, 59],
    }  # meloodia helistikule vastav saate helistik, oktav madalam

    # meloodia helikorgused ja kestused, track
    helistik = helistikud[Key_name]  # list midi noodi numbritega
    pitch_meloodia = norm_map_pitch(Zr, helistik)  # Zr normaliseeritud
    # noodi pikkused, Zi normaliseeritud
    duration = norm_map_dur(Zi, ajastik)

    # saate helikorgused, track
    helistik_s = helistikud_s[Key_name]  # list midi noodi numbritega
    pitch_saade = norm_map_pitch(Zm, helistik_s)  # Zm normaliseeritud
    # noodi pikkused, Za normaliseeritud
    duration_s = norm_map_dur(Za, ajastik)
    MyMIDI = MIDIFile(2)

    # meloodia
    midisaator(MyMIDI, 0, pitch_meloodia, program, duration, mitmehaalne,
    time)
    # saade
    midisaator(MyMIDI, 1, pitch_saade, program, duration_s, mitmehaalne,
    time_s)

    # 5 digit random number for fname
    rn = np.random.randint(10000, 99999)

    with open(f"Burning_Ship_c{c}_h_{Key_name}_{rn}.mid", "wb") as f:
        MyMIDI.writeFile(f)

if __name__ == "__main__":
    Melody(n, "d-duur", mitmehaalne, time, time_s)
```

Lisa 3. Lihtlitsents

Lisa
rektori 07.04.2020 käskkirjale nr 1-8/17

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina Liisi Raudväli

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Helilooming kasutades kahemõõtmelisi reaal- ja kompleksarvulisi kujutisi", mille juhendaja on Dmitri Kartofelev, PhD,

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

21.05.2025

[1] *Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. jq 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.*