Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Kostyál Domonkos Marcell

TÖBBSÁVOS DINAMIKASZABÁLYOZÓ EFFEKT MEGVALÓSÍTÁSA VST KÖRNYEZETBEN

KONZULENS

Dr. Rucz Péter

BUDAPEST, 2021

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Kostyál Domonkos Marcell**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2021. 12. 11.

...…….
Kostyál Domonkos Marcell

Összefoglaló

A mai világban egyre jobban elvárják a szoftverektől, hogy egy gombnyomásra mindent megoldjanak. Nincs ez máshogy az audió eszközöknél sem. A dinamikaszabályozó pluginek egy továbbgondolt megvalósítása a többsávos implementációjuk, amiknek fő célja, hogy csak a megadott frekvenciatartományt módosítsák. Szakdolgozatom a dinamikaszabályozók témakörével, azon belül a többsávos jelfeldolgozással foglalkozik.

Elsőként a dinamikaszabályozók alapvető működését és paraméterezését mutatom be. Ezt követően ismertetem a típusait és a különböző megvalósítási lehetőségekre is kitérek. A továbbiakban az optimális szűrőelrendezést vizsgálom, 24 sáv szétválasztására MATLAB segítségével, amik keretei közt az alapvető szűrőmodellek előnyeit és hátrányait is bemutatom. Linkwitz-Riley szűrőkarakterisztikáját használtam, mely két azonos paraméterezésű Butterworth-szűrő kaszkádba kötésével valósítható meg. A szoftverem végleges megvalósítását VST plugin szoftverként kiviteleztem, ezért a VST SDK fejlesztői környezetét, és az ebben való processzálás megvalósítását is bemutatom. A kellően intuitív grafikus felület is fontos számomra, emiatt FFT algoritmus felhasználásával létrehoztam egy valós idejű spektrumszámító és ábrázoló szoftverkomponenst, mely a felhasználó számára segítséget nyújt a megfelelő analízisre. Ezután a megvalósított plugint tesztelem, és összevetem a korábban lefektetett elvárásokkal. Végül megvizsgálom néhány hasonló szoftver működési elvét és eltéréseit az általam megvalósított eszközhöz képest.

Abstract

In today's world, programs are increasingly expected to do everything at the touch of a button, and audio effects are no different. A more sophisticated implementation of dynamics control plugins is their multiband implementation, whose main purpose is to modify only the given frequency range. My thesis studies the topic of dynamics control plug-ins, in particular with multiband signal processing.

First, the basic operation and parameterization of dynamics controllers is presented. Then, effect types and different implementation options are described. Next, I will investigate the optimal filter design for 24-band separation using MATLAB, in which the advantages and disadvantages of the basic filter models will be presented. I have used Linkwitz-Riley's filter design, which can be implemented by cascading two Butterworth filters with the same parameterization. The final implementation of my software is a VST plugin, so I will also present the development environment of the VST SDK and the implementation of processing in it. A sufficiently intuitive graphical interface is also important to me, so I created a spectrum using an FFT algorithm to help the user to analyse it properly. I will then test the implemented plugin and compare it to the previously laid out expectations. Finally, I examine the working principles and differences of some similar software, comparing them to the tool I implemented.

1 Bevezetés

A hangtechnika szakterületein mindig is a stúdiók alapfelszereltségéhez tartoztak a dinamikaszabályozók. Az 1930-as években először rádiók és tévék kezdték ezeket az eszközöket használni, a tisztább és egyenletesebb kivezérlés érdekében. Velük párhuzamosan a zenészek is albumfelvétel közben a mikrofon és akkoriban használatos szalagos felvevő eszköz közé helyezték ezeket a hardver effekteket, hogy így könnyedén szabályozni tudják a hangszerek dinamikáját. Az 1980-as években kezdtek megjelenni a digitális dinamikaszabályozók, melyek többségének a fő célja a korábbi analóg eszközök szimulálása, hangzásvilágának másolása volt. Az egyik legnépszerűbb analóg kompresszor effekt a *Universal Audio* által gyártott *1176* -os elnevezésű eszköz [1], melynek emulációját a mai napig optimalizálják.

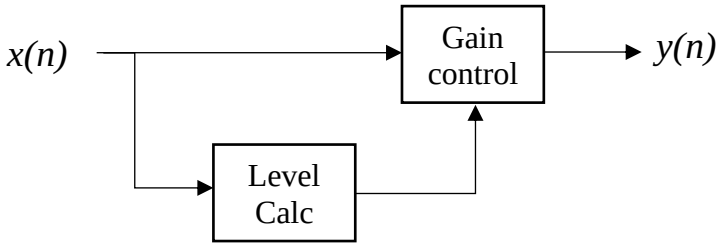
A szakdolgozatom témájaként ettől a gondolkodásmódtól elrugaszkodva egy újabb oldalról szeretném megvizsgálni ezt a kérdéskört, és egy modernebb megoldást találni a dinamika szabályozáshoz, ezzel egy újabb színt víve az ágazatba. Lehet, van a piacon már néhány közel hasonló megközelítés, de kíváncsian álltam neki a problémának és volt egy konkrét célom: Egy olyan frekvenciafüggő nemlinearitás létrehozása, melynek a sávjainak a száma 24, így egyfajta dinamikus szűrőeffektként is lehet rá tekinteni. Emellett fontos, hogy a szoftver megfelelő grafikus felületet is biztosítson a jel analizáláshoz.

A szoftver megtervezéséhez MATLAB szoftverkörnyezetet használtam és a megvalósításhoz a Steinberg német zenei szoftvereket fejlesztő cég tulajdonában levő, VST (Virtual Studio Technology) néven elnevezett szoftverfejlesztői csomagban dolgoztam, mely az iparágban széleskörűen elismert és az audió effektek és szintetizátorok alapvető szabványa. A grafikus felület létrehozásához a JUCE keretrendszerét használtam.

A következő fejezetekben először bemutatom a dinamikaszabályozó effektek alapvető működését és típusait, ezután a többsávos rendszer kiépítését részletezem, melyet a VST plugin megvalósítása követ, végül pedig tesztelem és összehasonlítom hasonló funkcionalitású szoftverekkel a megvalósított szoftvert.

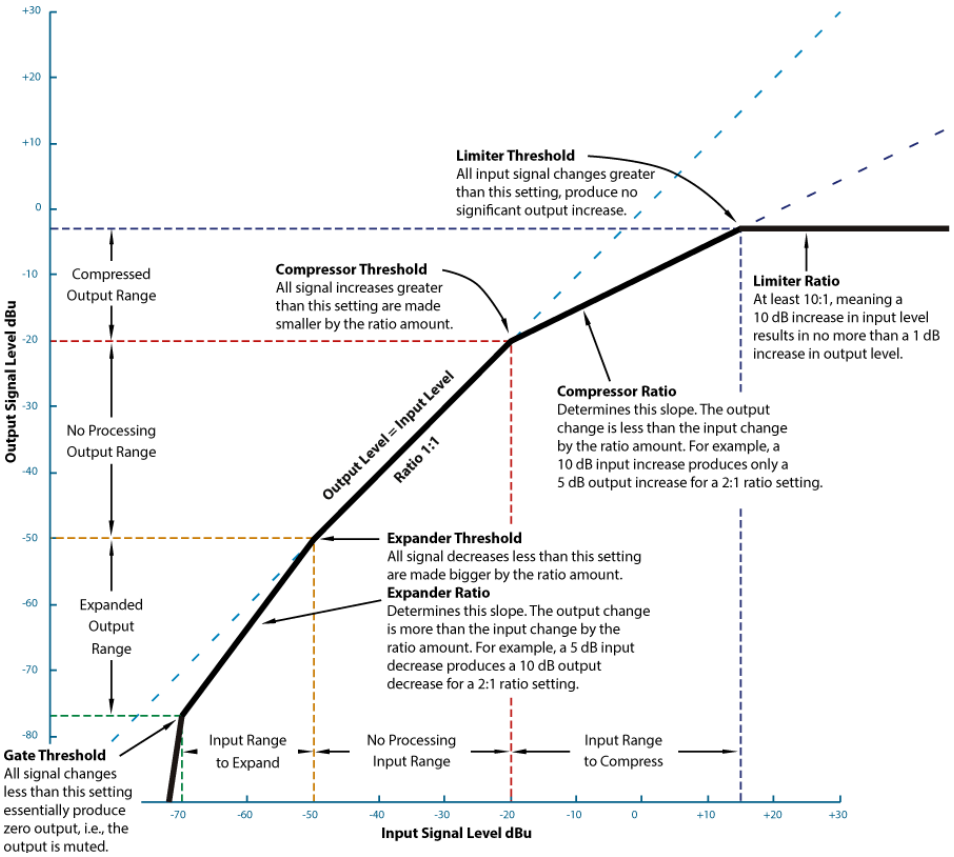
2 Dinamikaszabályozó effektek

Elsőként általánosan a dinamikaszabályozókat mutatom be. Mint a nevükben is benne van és a 2.1. ábra is mutatja, ezek az eszközök a jel amplitúdóját modulálják (a dinamika tartományát módosítják) a jelszint függvényében. A processzálásának módját különböző paraméterek segítségével lehet befolyásolni az adott cél eléréséhez.



2.1. ábra. Egy általános dinamikaszabályozó blokkdiagramja

Az ilyen típusú effektek működését szemléletesen egy jelleggörbe grafikonon lehet vizualizálni, ahol a vízszintes tengely a bemenő, a függőleges tengely pedig a kimenő jel amplitúdóját reprezentálja. A 2.2. ábra a főbb dinamikaszabályozó típusokat mutatja be.



2.2. ábra. Általános jelleggörbe grafikon [1]

2.1 Jelszint mérése

Mivel ezek a szoftverek a jel amplitúdója alapján végzik a modulációt emiatt a legelső feladat ennek meghatározása. Erre több megoldás is létezik, melyek között a különbség a későbbi funkcionalitás. Két fő típusra bonthatóak ezek az eszközök: a hangerősség mérésére és a pillanatnyi jelszint mérésére. A kettő között a mintavétel mennyisége a legfőbb különbség. A hangerősség mérésére sokkal nagyobb időablakot használnak, így leginkább abban segít a meghatározása, hogy egy átfogó képet tudjunk alkotni az adott jelről (erre használatosak a *LuFs* és *Leq (m)* szabványok). Ezek konkrét működése nem témája a szakdolgozatomnak, így nem foglalkozom részletesebben velük.

A jelszint mérésekor viszont a pillanatnyi erősségre vagyunk kíváncsiak, és ez az, amit használni tudunk a dinamikaszabályozók megtervezésekor. Ennek meghatározására kétféle módszert szoktak használni: A csúcs (*peak*) és a négyzetes átlagolás (*RMS*) mérést. A csúcs mérés leginkább limitáláskor hasznos, mivel egy pontos értéket ad az aktuális jel állapotáról. Ennek hátránya, hogy használatakor több torzítást vihet a rendszerbe, mivel értéke nagyon gyorsan változhat. Egy változata a *True peak* (valós csúcs) mérés, mely annyiban különbözik a szokványos csúcs méréstől, hogy az analóg jel csúcspontjait veszi figyelembe, nem pedig a digitális pontok maximumát méri.

Az *RMS* (Root Mean Square) ezzel ellentétben egy kicsit lassabb, általában néhány 100 ms körüli az időállandója. Ezt úgy éri el, hogy a jelet először négyzetre emeli, majd egy *IIR* (végtelen impulzus válaszú) elsőfokú aluláteresztő szűrővel megszűri, melynek a vágási frekvenciája határozza meg, milyen gyorsan kövesse a jelet, így audió jelek esetén érdemes 10 Hz környékire állítani. A 10 Hz környéki választást az indokolja, hogy így az emberi hallás sávjába eső állandó amplitúdójú jelek esetén nem fog a szűrő kimenete ingadozni a legkisebb frekvenciák esetén sem. Végül pedig gyököt von és így kapjuk meg a kívánt amplitúdót, ahogy ezt a 2.3. ábra is mutatja. A megvalósított szoftverben ezt a megoldást használtam.

2.3. ábra. RMS számítás blokkdiagramja

2.2 Dinamikaszabályozók paraméterei

Annak ellenére, hogy az amplitúdómodulációt több oldalról is meg lehet közelíteni, a paraméterezésüket általában ugyanazokkal a változókkal lehet definiálni. Ezek a paraméterek a moduláció két állapota közötti kapcsolatot (a szabályozott és szabályozatlan) és a moduláció mértékét írják le:

- **Threshold** – az egyik legalapvetőbb érték, mely azt határozza meg, hogy mi az a jelszint, ami felett (vagy alatt) a szabályozás bekövetkezzen. A *threshold* szint átlépésekor kapcsolja az adott moduláció ki, illetve be.
- **Ratio** – csak a szabályozott állapottal van kapcsolatban. Ez egy arányszám, mely megadja, hogy milyen arányban változtasson a dinamikaszabályozó az eredeti jel amplitúdóján. Nagyon fontos, hogy ez a paraméter csak a *thresholdon* túli dB értéket vizsgálva módosítja az aránynak megfelelően a jel szintjét.
- **Attack time** – azaz felfutási idő megadja, hogy mi az az idő, amibe kerülnie kell, hogy lekövesse a kívánt osztást. A paraméter csak az osztás növekedését szabályozza. Ez általában néhány ms-os nagyságrendű érték.
- **Release time** – azaz elengedési vagy lefutási idő, melynek funkciója megegyezik a felfutási idővel, annyi különbséggel, hogy a paraméter csak az osztás csökkenését szabályozza. Ez általában néhány 10, akár 100 ms-os nagyságrendű érték. A két paraméter azért fontos, hogy az eredeti jelalakot ne érhesse nagyobb torzítás az esetleges ugrások miatt.

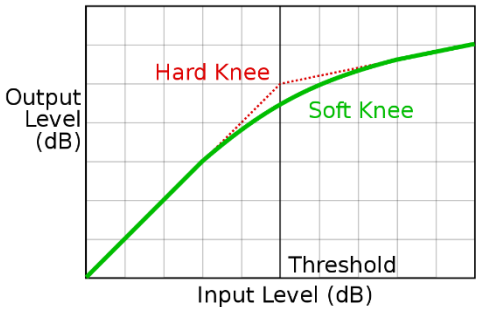
Mind a felfutási, mind a lefutási időt érdemes egyfajta mozgóátlaggal megvalósítani, ahol a moduláció mértékének a változását állítjuk vele:

$$Gain = (1 - Coeff) * Gain_{Régi} + Coeff * Gain_{Új}$$

$$Coeff = \frac{1}{T * SampleRate}$$

ahol T az attack vagy release time másodpercben megadva és a $SampleRate$ a mintavételi frekvencia *Hz*-ben.

- **Look-ahead** – legfőképpen limitereknél fontos ez a paraméter, mivel szoros kapcsolatban van a felfutási idővel. Azt az időtartamot adja meg, hogy a szabályozó *threshold* komparálása mennyivel „nézzen előre” a jelben. Tehát ha „látja”, hogy később szabályozni kell, akkor ennyivel hamarabb elkezd átváltani a szabályozott állapotba a rendszer. Így például egy kompresszornál az amplitúdó „tüskék”, melyeket a felfutási idő okoz, eliminálhatóak. Könnyen belátható, hogy ez a paraméter a limitereknél kap nagyobb jelentőéget.
- **Knee** – ezt a paramétert nem mindig implementálják. A motivációja, hogy az emberi fülnek természetesebbnek hangzó modulációt érhetünk el ennek növelésével. Ez legfőképp akkor fontos, ha olyan jellel kell dolgozni, aminek a jelszintje *threshold* közeli. Szemléletesen a jelleggörbével lehet működési elvét ábrázolni. A 2.4. ábra egy kompresszor két különböző *knee* beállítása látható. Alapvetően a ratio értékét manipulálja úgy, hogy ha növeljük az értékét, akkor a szabályozatlan és szabályozott állapot között egy folyamatos átmenetet képez.



2.4. ábra. Knee jelleggörbe diagramja [3]

2.3 Dinamikaszabályozás típusai

A következő pontokban a jellegzetesebb dinamikaszabályozásra alkalmas effekteket részletezem.

2.3.1 Kompresszor

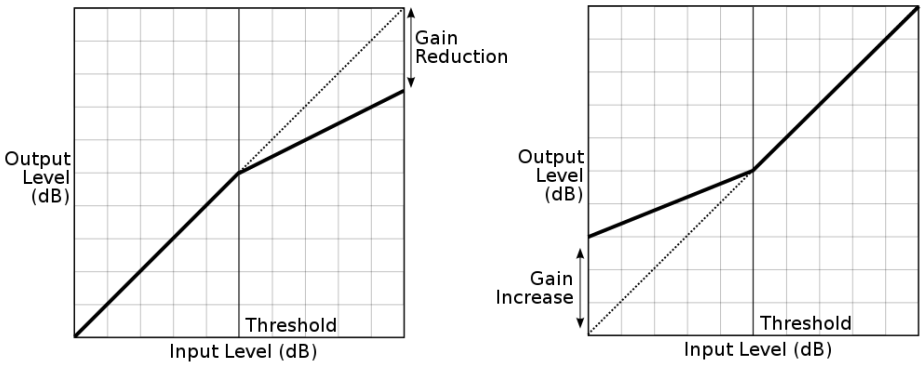
A kompresszor az iparág legelterjedtebb dinamikaszabályozó típusa. Fő feladata a dinamikatartomány csökkentése. Ezt úgy teszi, hogyha a jel *threshold* fölé kerül, akkor azt a *ratio* arányában csökkenti. A csökkentés arányát úgy lehet meghatározni, ha például a *ratio = 4:1*-hez akkor az azt jelenti, hogy a jel túllőtt részét negyedére kell csökkenteni. Fontos, hogy itt a jel erősségét mind dB-ben határozzuk meg és ezzel is számolunk. Ebből következik, hogy minél nagyobb az arány, annál nagyobb az osztás, viszont sohasem éri el a limitálást. Ennek matematikai megvalósításához, a jelleggörbe segítség lehet, viszont végül a dB-ben mért csökkentés mértékét kell meghatároznunk, melyet az alábbi egyenlettel tudunk kiszámolni:

$$Gain\ Reduction = \begin{cases} (1 - Ratio) * \Delta X\,, & Threshold < X_{jel} \\ 0\,, & Threshold \geq X_{jel} \end{cases}$$

$$ahol: \Delta X = \left(Threshold - X_{jel}\right)$$

És az X_{jel} az *RMS* méréssel kiszámolt aktuális értéke a jelnek dB-ben.

Kétféle kompresszor létezik, a halkító (*downwards*) és hangosító (*upwards*) kompresszor. Mindkét típus feladata alapvetően ugyanaz, a dinamikatartomány csökkentése. Eddig a halkító kompresszorokat részleteztem, de a hangosító is nagyon hasonló hozzá: a *threshold* alatti amplitúdót a megfelelő arányban erősíti. Az egyenlet ugyanaz, a különbség annyi, hogy a ΔX pozitív lesz emiatt, és ez logikus, mivel erősítenünk kell ekkor a jelet (2.5. ábra).



2.5. ábra. Halkító (baloldali) és hangosító (jobboldali) kompresszor jelleggörbéinek összehasonlítása [3]

A hangosító kompresszor esetében be kell vezetnünk egy maximális erősítést, mivel, ha a bemenő jel nullához tart, akkor a dB értékek miatt $X_{jel} \to -\infty$, ami azt eredményezné, hogy $\Delta X \to \infty$, és így a képletből látható, hogy a $Gain\ Reduction \to \infty$, ami végül hatalmas túllövéshez és emiatt torzításhoz vezet. Ezt én úgy oldottam meg, hogy az erősítés sohasem lehet 90 dB-nél több, ezzel elkerülve a változó elszállását.

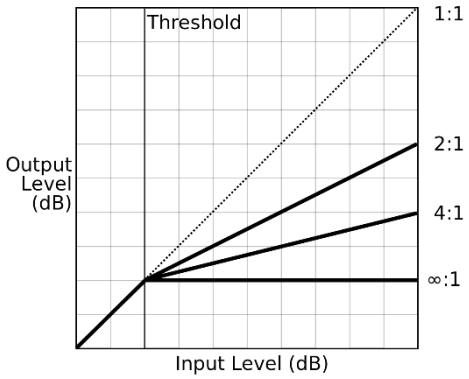
2.3.2 Limiter

Sokszor szokás egy magasabb *ratiora* beállított kompresszort limiternek nevezni, mivel a limiterek fő feladata, hogy egy bizonyos szint fölé ne engedje a jel szintjét. Általában ezt az effektet szokták utoljára alkalmazni hangsávokon, hogy maximalizálják a kivezérlést. Egy minőségi limiter megvalósítása kifejezetten bonyolult, mivel a jel minőségének minimális torzulására kell törekedni amellett, hogy teljes kontrol alatt maradjon a jel amplitúdója.

A problémának két általánosabb megoldási lehetősége van: Az első, hogy a felfutási időt a lehető legkisebbre állítjuk (0-nem lehet az átlagolás miatt), ezzel egy nagyon gyors kompresszort létrehozva. A másik lehetőség, hogy beiktatjuk a *look-ahead* változót, amivel az adott pontra már teljes kompressziót tudunk alkalmazni. Az első megoldással az a probléma, hogy egy minimális túllövést mindenképpen elszenved a jel, és összességében csak egy erős és gyors kompresszor fog megvalósulni, így ez a módszer nem annyira használatos. A második módszerrel az a probléma, hogy késleltetés meg fog jelenni a jelben, mivel néhány ms-mal előre kell vizsgálni a hangsávot, hogy hamarabb elkezdődhessen a limitálás, ha szükséges.

A korábbi képlet annyival változik, hogy az arány helyett teljes redukciót alkalmazunk tehát $Ratio \to \infty : 1$ (2.6. ábra) ami törtként kifejezve belátható, hogy $Ratio \to 0$, így:

$$Gain\ Reduction = \begin{cases} \Delta X\,, & Threshold < X_{jel} \\ 0\,, & Threshold \geq X_{jel} \end{cases} \quad ahol: \Delta X = \left(Threshold - X_{jel}\right)$$



2.6. ábra. Különböző ratioval paraméterezett kompresszorok és limiter (∞:1) [3]

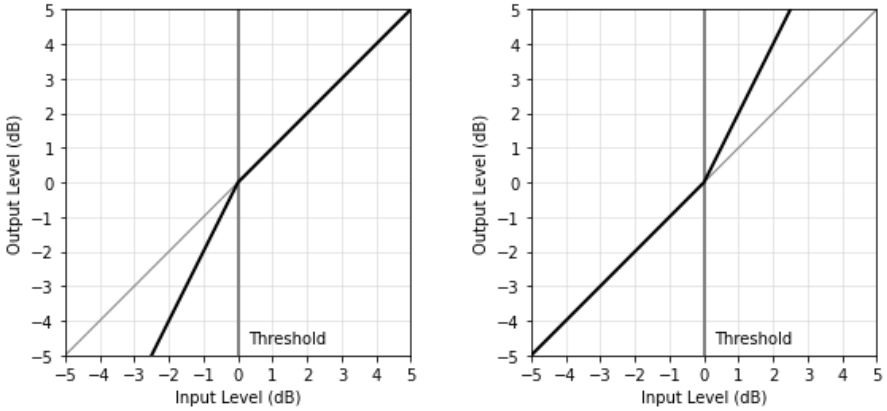
2.3.3 Expander

A kompresszor ellentéte az Expander. Feladata a dinamikatartomány növelése. Hasonlóan a kompresszorhoz ezt kétféleképpen teheti meg: vagy a halkítva, vagy hangosítva (2.3.3. ábra). A hangosítást jellemzően nem használják, mivel azzal nagyon könnyű túllőni a jel maximumán, ezzel erős torzítást víve a rendszerbe. Ez legfőképp azért jöhet létre, mivel a digitális audió jelfeldolgozásban a jel értékei [-1, +1] közötti tartományon mozoghatnak, és bármilyen érték, ami kívül esik ebből a tartományból a legközelebbi szélsőértékre lesz kerekítve. Emellett könnyen beláthatjuk, hogy ebben az esetben, ha a jel *threshold* fölé kerül, akkor tovább erősíti az amplitúdót, figyelmen kívül hagyva a jel korlátosságát.

A képlet az expander amplitúdókompenzálásához nagyon hasonló a kompresszoréhoz, annyi különbséggel, hogy az előjelet meg kell cseréljük és a relációjeleket is, ugyanis ahol eddig csökkentettünk, növelnünk kell:

$$Gain\ Reduction = \begin{cases} 0\,, & Threshold < X_{jel} \\ (1 - Ratio) * \Delta X\,, & Threshold \geq X_{jel} \end{cases}$$

$$ahol: \Delta X = \left(X_{jel} - Threshold\right)$$



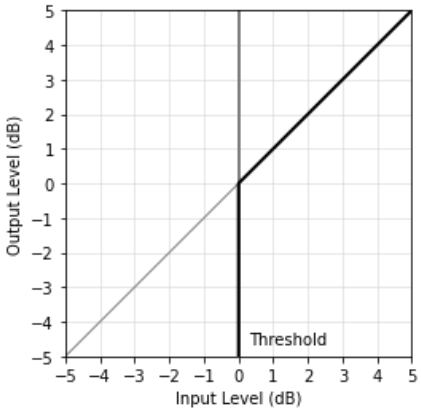
2.3.3. ábra. Halkító (baloldali) és hangosító (jobboldali) expander jelleggörbéinek összehasonlítása

2.3.4 Zajzár

A zajzár azaz *gate* hasonló kapcsolatban van az expanderrel, mint a limiter a kompresszorral. Különbségük, hogy ebben az esetben az expander *ratio* paraméterét állítjuk kellően nagyra a kívánt effekt eléréséért, ahogy azt a 2.7. ábra is mutatja. Mint a magyar elnevezésében is benne van, legfőképpen arra használatos, hogy a háttérzajt kiszűrjük a jelünkből. Fontos, hogy mivel a jelszintet vizsgálva működik, így arra nem használható, hogy egy például folytonos háttérzúgást teljesen kivegyen egy jelből.

Az effekt pontos vezérléséhez még bevezettek két paramétert ,melyek alapvetően nem kötelezőek, de a megfelelő cél eléréséért sokszor implementálják ebben az effektben. Az egyik ilyen vezérlő a mélység (*depth*), mely azt adja meg, hogy hány dB-lel csökkentsük a jelszintet, ha *threshold* alatti a jel erőssége. Ez amiatt fontos, mert mint a hangosító kompresszornál, itt is egy olyan probléma lép fel, hogy végtelen halkítást kellene végrehajtani a jelen, ha az értéke 0 (tehát nincs kimenet), így a *depth* bevezetésével egy maximális és egyben adott halkítást viszünk véghez. A másik paraméter a tartás (*hold*), mely azt mondja meg, hogy mi az a minimális idő, amíg a zajzár effektnek nyitva kell lennie. Ezzel a nagyon rövid kis kiállásokat lehet eliminálni. A képlet annyiban egyszerűbb a többihez képest, hogy valójában egy eltolást kell megvalósítanunk, ahol, ha a *threshold* alá kerül a jel, akkor a *depth* mennyiségével csökkentse az amplitúdót. Ennek értéke 64 dB körüli szokott lenni, így én is ezt az értéket adtam meg a szoftverem megvalósítása során.

$$Gain\ Reduction = \begin{cases} 0\,, & Threshold < X_{jel} \\ -64\,, & Threshold \geq X_{jel} \end{cases}$$



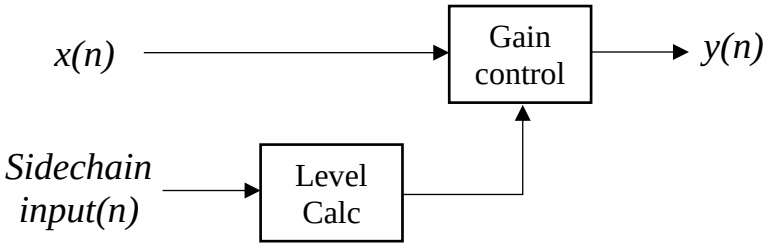
2.7. ábra. Zajzár ideális jelleggörbéje

2.3.5 Egyéb alkalmazások

- **Sidechain**

  Az elektronikus zene világában mára szinte alapvető technikaként használatos a sidechain kompresszálás. Az alapgondolat, hogy az analizált jel és a processzált jel két különböző hangsáv, ahogy azt a 2.8. ábra is mutatja. Így például, ha az egyik hangszert ki szeretnénk emelni egy másik hangszerhez képest, akkor ezzel lehetséges, hogy visszahalkítsuk a másikat amíg az egyik szól.

  Jellegzetes probléma a basszus és lábdob közötti kapcsolat, mivel mindkettő ugyanabban a frekvenciatartományban szól, így könnyen zavarossá válhat a zene, ha egyszerre vannak jelen a zenében. Erre a problémára tökéletes a sidechain kompresszió, ugyanis, ha a basszuson elvégezzük a processzálást a lábdob vezérlésével, akkor megoldható, hogy ne szóljanak soha egyszerre, így tiszta maradhat a mély frekvenciatartomány.



2.8. ábra. Sidechain processzálás blokkdiagramja

- **De-esser**

  Fő felhasználási területe az emberi hangok tisztítása. A motiváció mögötte, hogy az ének szellőssége kellemes hangzást nyújt a fülünknek, melyet a magas frekvenciatartományok kiemelésével érhetünk el. Viszont ezzel együtt előjön az a probléma, hogy a kimondott ’sz’ és ’z’ betűk nagyon harsányak lesznek a kiemelés után, így ezeket egy speciális kompresszorral vissza kell venni, mely csak a magas frekvenciákon veszi vissza a kiugrásokat. Erre egy olyan kompresszort alkalmaznak, ami szűrők segítségével csak a problémás frekvenciákat fedi le és a thresholdját dinamikusan változtatja az alsó frekvenciák jelszintjének függvényében. Belátható, hogy az általam megvalósított többsávos kompresszor effekt erre a feladatra is használható.

3 Szűrőbank megtervezése MATLAB környezetben

3.1 Elvárások

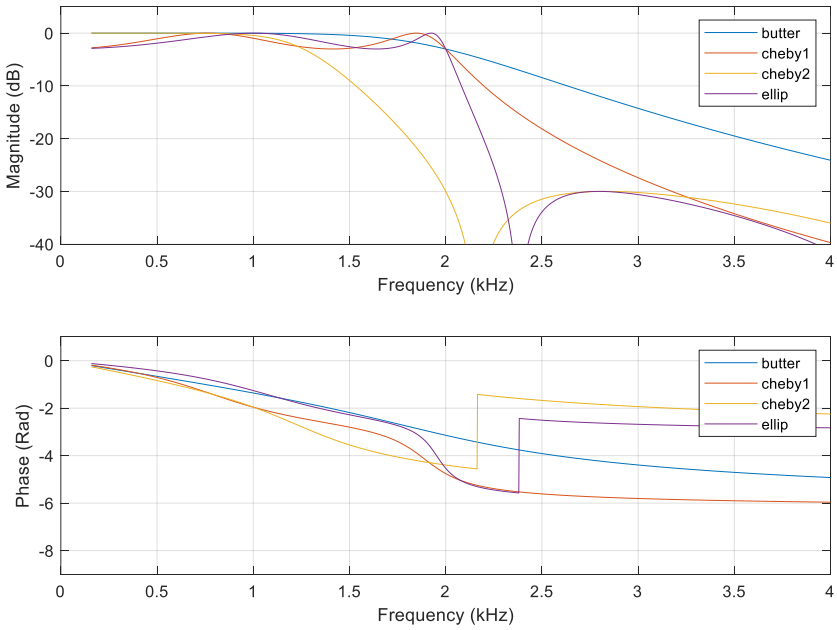
A több sáv külön feldolgozásához egyik kézenfekvő megoldás, hogyha ezeket a sávokat szűrők segítségével szétválasztjuk és ezután végezzük el külön a sávokon a modulációt. Ezt követően pedig újra összeadjuk a sávokat, ezzel megkapva a módosult kimeneti jelet.

Könnyen belátható, hogy emiatt a szoftver egyik kritikus pontja a megfelelő szűrőelrendezés megtervezése volt. A cél az volt, hogy minél kevésbé torzuljon a jelalak, és lehetőleg a fázistolás minél jobban közelítse a lineárisat. Kiindulási alapként megvizsgáltam a különböző szűrőkarakterikákat és utánajártam, hogy általában milyen modellt használnak a hangszórók tervezésekor frekvenciaszétválasztásra.

3.2 Különböző szűrőkarakterisztikák vizsgálata

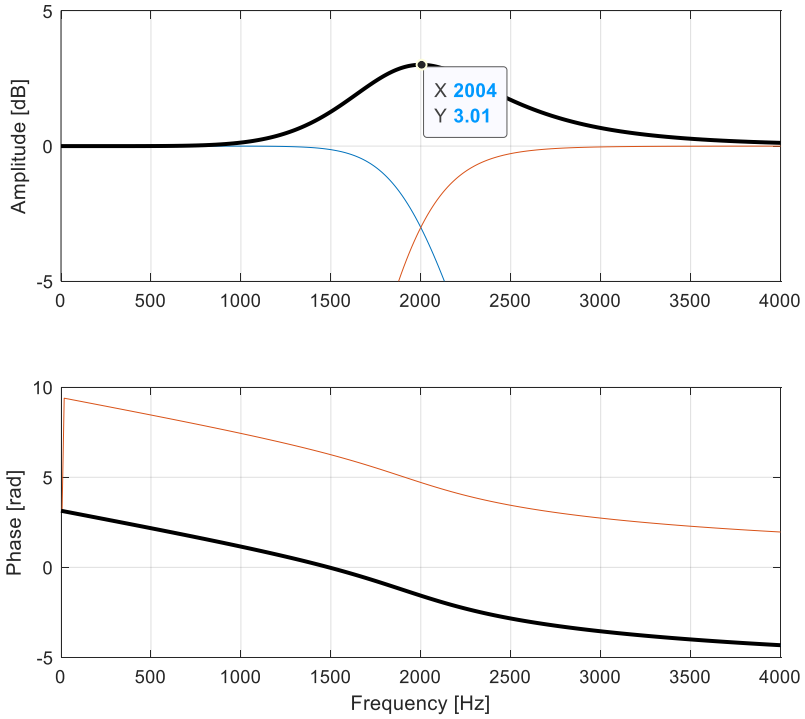
Négy általános szűrőelrendezés vizsgálatával kezdtem a tervezést: *Butterworth, Chebyshev 1, Chebyshev 2* és *Elliptikus* szűrőkkel. Ahogy azt a 3.2. ábra mutatja, mindegyiknek megvan az előnye és hátránya is. Ezeket mérlegelve kellett kiválasztani a megfelelő szűrőkarakterisztikát.

- A **Butterworth** szűrő amplitúdómenete az áteresztő tartományban a legegyenletesebb és a fázistolása is közel lineáris, viszont a vágási tartományban nem vág kellően meredeken.
- A **Chebyshev 1** a vágási tartományban sokkal meredekebben viselkedik, mint a *Butterworth* szűrő, viszont az áteresztőtartományban az amplitúdómenete nem egyenes.
- A **Chebyshev 2** hasonló a *Chebyshev 1* szűrőhöz, annyi különbséggel, hogy a vágási tartományban nem tökéletes a szűrő és az áteresztő tartományban közelíti az ideálist.
- Az **Elliptikus** szűrő a két *Chebyshev* kombinációja, kellően meredek, de mind áteresztő, mind vágási tartományában az amplitúdómenete mozog, emellett a fázistolása sem lineáris.



3.2. ábra. Butterworth, Chebyshev 1, Chebyshev 2 és Elliptikus negyedfokú aluláteresztő szűrők Bode-diagramja, f_c = 2000 Hz vágási frekvenciával.

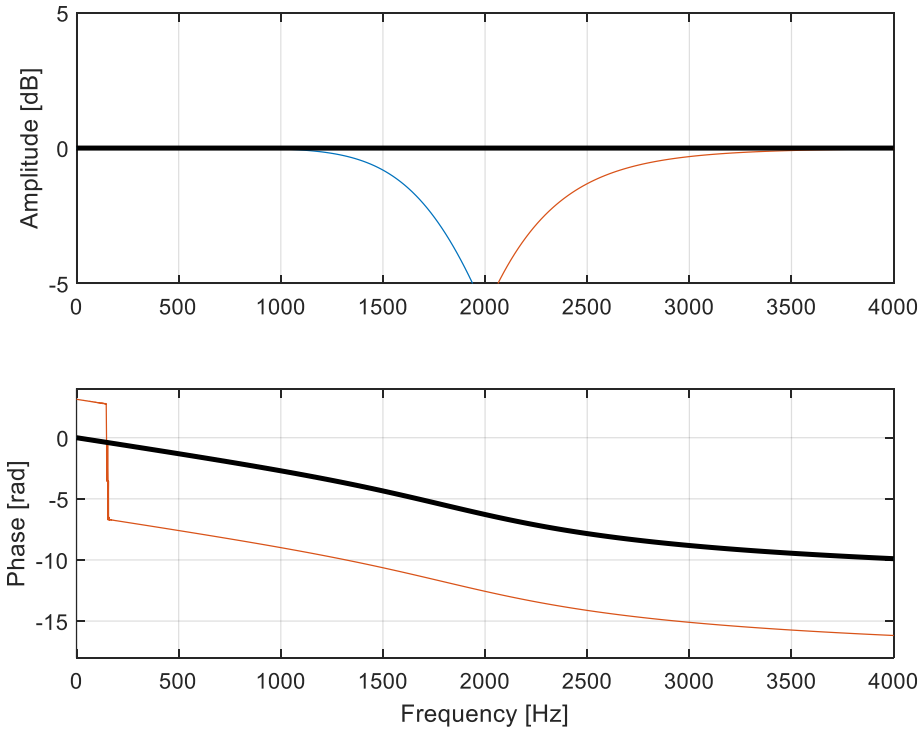
Ezek közül a Butterworth szűrő tűnt a legmegfelelőbb megoldásnak, így megvizsgáltam, hogy hogyan viselkedik, ha egy negyedfokú aluláteresztő és



3.1. ábra. Butterworth negyedfokú alul és felül áteresztő szűrők és összegüknek Bode-diagramja f_c = 2000 Hz vágási frekvenciával

felüláteresztő szűrőt összeadok ugyanazzal a vágási frekvenciával. A 3.1. ábra mutatja, hogy ebben az esetben egy 3 dB erősítés létre fog jönni a vágási frekvenciánál, így tovább kellett vizsgálódni, ugyanis ez az elvárásoknak nem volt megfelelő.

A hangszórók frekvenciaváltójának vizsgálatával folytattam a kutatásomat, mivel ezen esetekben kifejezetten fontos a fázis linearitásának megőrzése és az amplitúdómenet egyenletessége. Mint kiderült, a legtöbb esetben *Linkwitz-Riley* [4] szűrőket használnak erre a problémára. A szűrő megvalósítását illetően nem több, mint két azonosan paraméterezett *Butterworth*-szűrő sorosan kapcsolva egymás után. Így a fázismenet megmarad és a korábbi -3 dB-es vágási frekvencia -6 dB-re fog csökkenni, ezzel az alul- és felülátereszto szűrők összeadásakor megszűnik a kiemelés ahogy ezt a 3.3. ábra is mutatja.

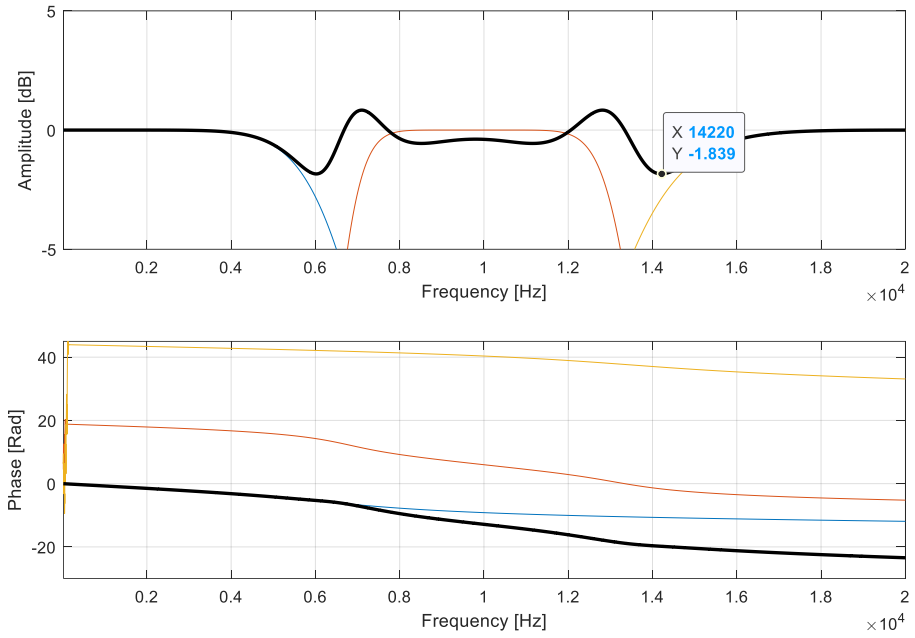


3.3. ábra. Linkwitz-Riley negyedfokú alul- és felülátereszto szűrők és összegük Bode-diagramja fc = 2000 Hz vágási frekvencián

3.3 Tervezés

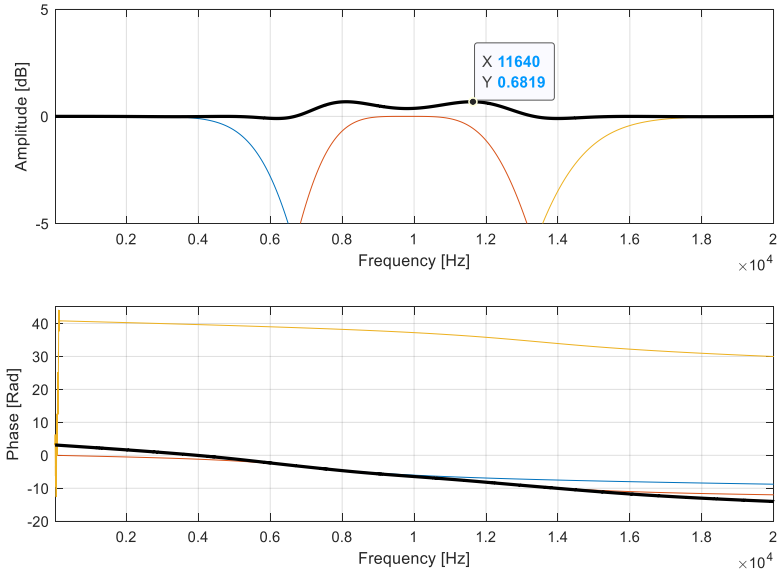
A következő feladatom a több sávra szétosztás volt, amihez újabb szűrőket kellett beillesztenem a rendszerbe úgy, hogy a teljes hallható frekvenciatartományt egyenletesen osszák szét, ügyelve az elvárások teljesülésére. Erre az alul és felülateresztő szűrők közé sáváteresztőszűrők beillesztését találtam legalkalmasabbnak.

három sávval kezdtem a tesztelést negyedfokú szűrőkön. Ahogy azt a 3.4. ábra is mutatja 1.8 dB-es kilengés volt ebben az esetben a maximális, és mivel feltételezhetjük, hogy ez több sáv esetén nőni fog, így tovább finomítottam az elrendezésen.



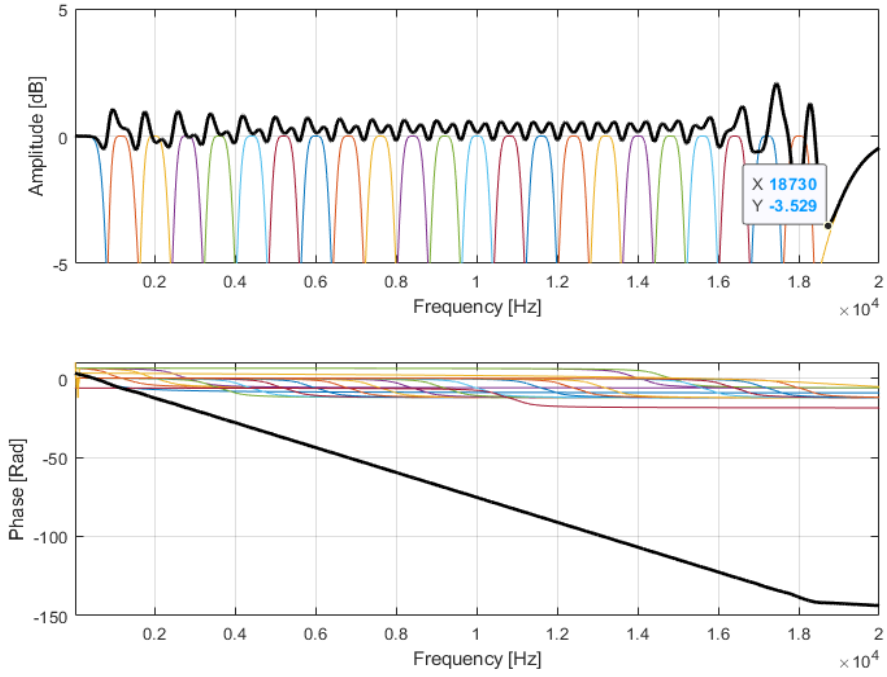
3.4. ábra. Háromsávos negyedfokú Linkwitz-Riley szűrőelrendezés Bode-diagramja f_1 = 6680 Hz és f_2 = 13340 Hz vágási frekvenciákkal

A sáváteresztő szűrőt másodfokúra csökkentettem és az így fellépő fáziskioltás miatt az alul- és felülateresztő szűrők fázisát megfordítottam, ezzel sokkal jobb eredményt elérve. A 3.6. ábra mutatja, hogy a maximális erősítés alig haladja meg a 0.5 dB-t így ezt választottam végleges elrendezésnek.



3.6. ábra. 3 sávos javított szűrőelrendezés Bode-diagramja f_1 = 6680 Hz és f_2 = 13340 Hz vágási frekvenciákkal

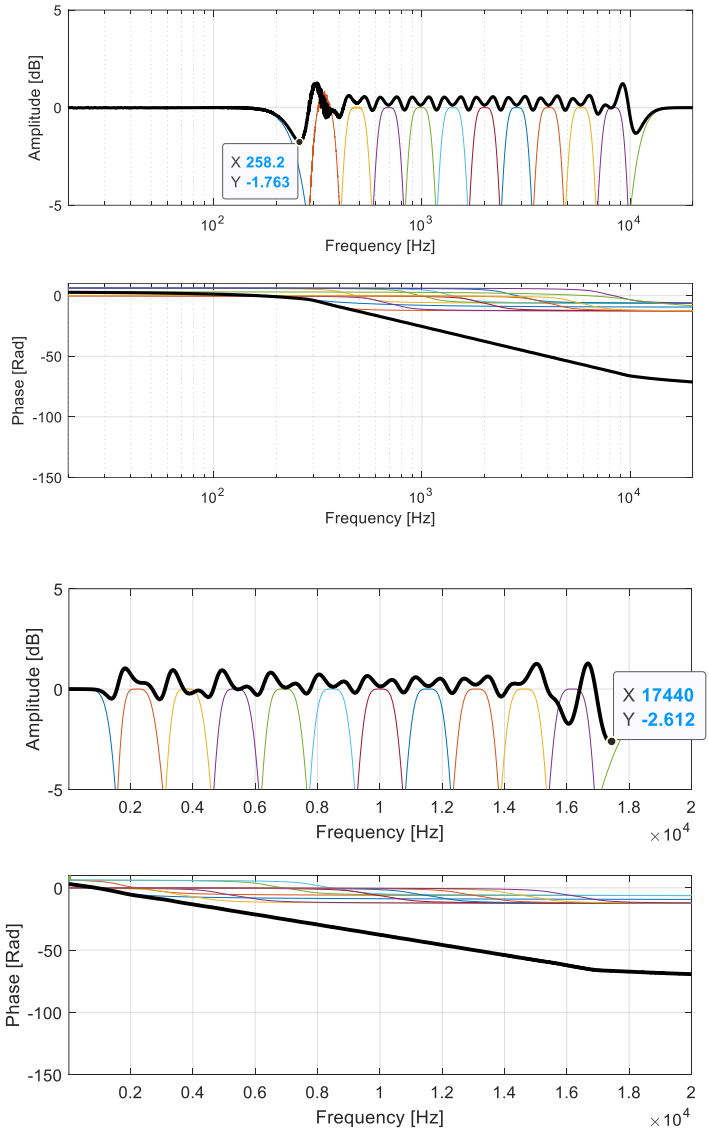
Végül teszteltem 24 sávra is a megtervezett rendszert, melynek az eredményeit a 3.5. ábra szemlélteti. A legnagyobb eltérés 18700 Hz-nél történik, ahol -3.5 dB-t is eléri az erősítés, viszont ez a hiba elfogadható, főleg, hogy ilyen magas frekvenciatartományban van. Az átlagerősítés 0.03 dB, ami bőven eleget tesz az elvárásoknak így ezt a szűrőelrendezést használtam a megvalósításom során.



3.5. ábra. 22 sáváteresztő, egy aluláteresztő szűrő, egy felüláteresztő szűrő és összegüknek Bode-diagramja lineáris eloszlású vágásifrekvencia közökkel.

Eddig a szűrőelrendezés lineárisan volt paraméterezve, viszont a hallásunk logaritmikus, így, ha azt szeretnénk hallani, hogy a sávok ugyanakkora tartományt fednek le, akkor logaritmikusan kell felosztani őket. 12 sávos szétosztással vizsgáltam az elrendezést, viszont a logaritmikus osztás miatt feljebb kellett vennem a legalsó vágási frekvenciát, mivel a MATLAB *Butterworth* modellje nem paraméterezhető túl kis távolságú vágási frekvenciákkal.

A 3.7. ábra mutatja, hogy az elvárásoknak megfelelően működik a logaritmikus osztás, és majdnem 1 dB-lel kisebb a maximális ingása az erősítésnek, így ezt a frekvenciakiosztást alkalmaztam a megvalósított szoftverben.



3.7. ábra. logaritmikus (felül) és lineáris (alul) frekvenciakiosztással vizsgált szűrőelrendezés

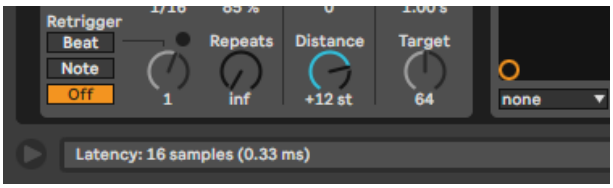
4 VST implementáció

Fontos volt számomra, hogy ne csak egy modell valósuljon meg, hanem egy használható eszköz, mely kompatibilis a zeneiparban használatos szoftverekkel. Emiatt döntöttem úgy, hogy a végleges szoftveremet VST pluginként (Virtual Studio Technology) fogom implementálni. A technológiát Steinberg Media Technologies Kft. [5] fejlesztette ki 1996-ban az akkori Cubase Digital Audio Workstation (DAW) szoftverükhöz, hogy külsős partnerek szintetizátorokat és effekteket írhassanak a DAW-hoz. Azóta folyamatosan fejlesztik és az iparág szabványának tekinthető. A struktúra két fő elemből áll, a hosztból (DAW) és a pluginekből, melyek egyfajta master-slave kapcsolatban vannak egymással és mind követik a VST szabványt, aminek segítségével kommunikálnak egymással.

4.1. ábra. Ableton Live 10 DAW és VST-k integrációja egy projekten belül (Az alsó sorban láthatóak a VST-k egy midi sávon sorban elhelyezve ezek közül a Phasis VST grafikus felülete látható)

A DAW szolgáltatja a fő munkakörnyezetet és felelős, hogy minden időben szinkronizálva legyen, ne legyenek csúszások az egyes sávok között. Ehhez egy projektben felhasznált pluginekről mind tudnia kell, hogy hány mintányi a késleltetésük a többi sáv megfelelő kompenzálásához.

Mint ahogy a 4.1. ábra szemlélteti, a bal alsó sarokban az Ableton Live kiírja az adott plugin késleltetését, ez esetben egy phase effekt van nyitva, emiatt késik 16 mintát



4.2. ábra. Native Instruments Phasis VST effekt késleltetése

jel.

A VST szabvány két fő információ kezelésére nyújt fejlesztői környezetet. Az audió pufferek és MIDI üzenetek átadására / átvételére. Emellett egyéb informatív adatok átküldéséért is felelős, mint például a mintavételi frekvencia, a puffer mérete, audiócsatornák száma, a projekt tempója (BPM-ben), a plugin állapota a munkakörnyezetében (aktív, inaktív), éppen hanyadik ütemnél vagyunk a hosztban és ezekhez hasonló információk. Látható, hogy ennek az interfésznek köszönhetően a digitális világban is létrejött egy hasonló munkakörnyezet, mint a korábbi analóg hardverek világban.

A VST SDK-t (softver development kit) C++ nyelven fejlesztik, így felépítése objektumorientált. Az SDK-val fejleszthető pluginok szerkezete általában két fő osztályra épül, a processzorra és az editorra. A processzor feladata a DAW-val való kapcsolattartás és audiófeldolgozás, míg az editor a grafikus felületért felelős, és működése szoros kapcsolatban van a processzorral.

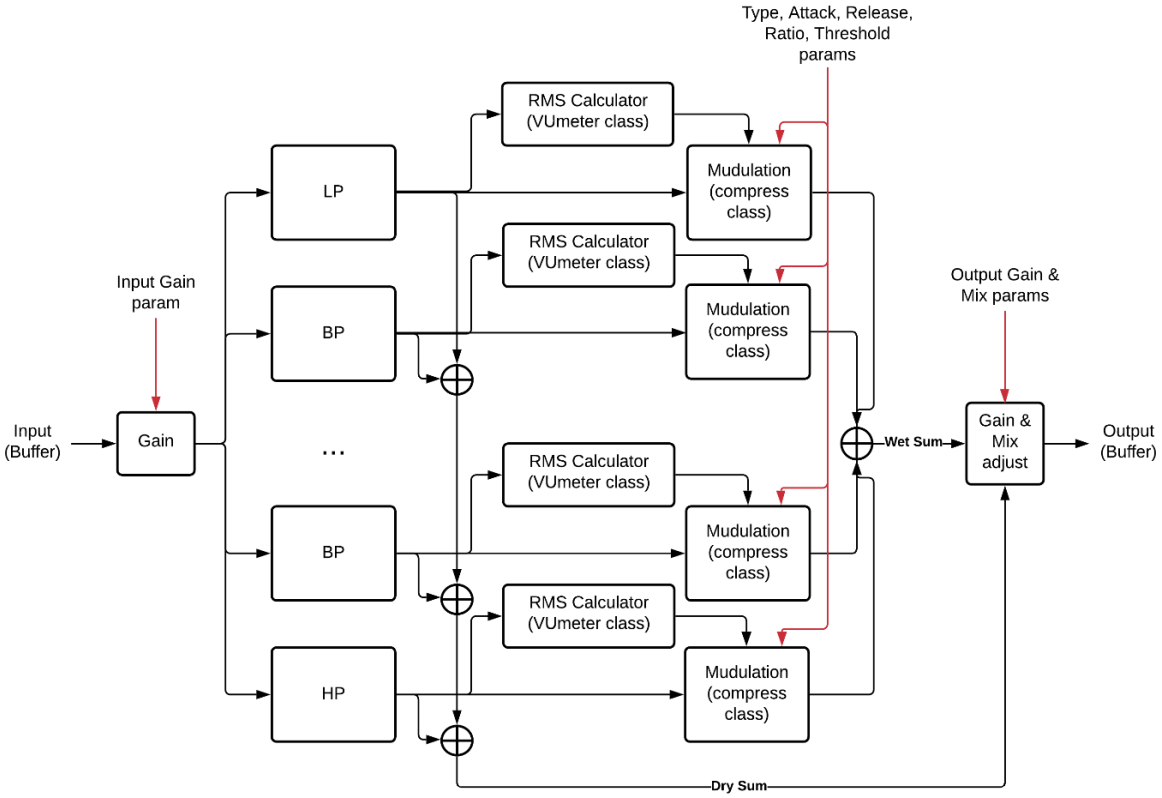
Mivel a mintákat blokkonként kapjuk, így a megvalósított szoftveremben végig a blokkos feldolgozásra törekedtem. A dinamikaprocesszáláshoz három osztályt hoztam létre, melyek alapvető működése két fő függvény köré épülnek: a *'setup'*, mely egyszer a szoftver megnyitásakor, vagy újrainicializásakor fut le a megfelelő puffer méretek beállítására és ehhez hasonló paraméterek megadására. A másik függvény a *'process'*, ami minden alkalommal, amikor új puffer érkezik, meg hívódik, és itt zajlik a valós idejű feldolgozás.

A `VUmeter` osztály, a jelszint mérésére szolgál. RMS mérést végez blokkonként több csatornára, melynek az időállandóját paraméteresen lehet állítani, az én megvalósításomban 100 ms megfelelőnek beállításnak bizonyult. A feldolgozott blokkból olyan többcsatornás kimenetet kaptunk, melyek az adott pontban a jel szintjének

dB-ben mért értékének feleltethető meg. Így az amplitúdómoduláció folyamán könnyebb volt a számolás, és nem kellett mégegyszer átváltani az értékeket.

A `compress` osztály nevével ellentétben nem csak kompresszálást végez, hanem az összes megvalósított dinamika szabályozást, egy külső paraméter függvényében, hogy ezt valós időben tudjuk változtatni. Hasonlóan a jelszintméréshez itt is tömbös feldolgozást alkalmaztam, így itt is a *'process'* függvény lefutásra egy többcsatornás jel tömbjére történik a kiszámolás. Fontos még, hogy a *'VUmeter'* egy objektuma a *'compress'* osztálynak. Itt történik a 2.3-as pont alatt részletezett processzálás, melyek közül a halkító és hangosító kompresszort, expandert, limitert, zajzárat és egy egyedi hangosító – halkító kompresszor kombinációját valósítottam meg, mely esetében, ha a *threshold* alá kerül a jelszint, hangosítja, ha pedig fölé akkor halkítja a jelet.

A legfontosabb osztály, ami a teljes processzálás egyben tartásáért felelős a `channelSplit` osztály. Feladata a sávok másolása, megszűrése a megtervezett elrendezéssel és szabályozása, végül pedig összeadása. A 4.3. ábra illusztrálja, hogy miután megszűrjük a jelet, egy külön sávban összeadom, ez amiatt van, mert kiviteleztem egy mix paramétert is a szoftverben, mellyel a nyers és processzált jel között lehet a keverést befolyásolni, viszont a teljesen nyers jellel ez nem kivitelezhető, mivel a szűrést



4.3. ábra. Többsávos dinamikaszabályozó folyamatábrája

követően fázistolás fog létrejönni. Így a megszűrt jelet adtam össze, hogy azonos fázisban legyen a kimenettel.

A szűrőket külső könyvtár [6] segítségével valósítottam meg, így hasonlóképpen tudtam használni a Butterworth modellt, mint MATLAB környezetben. Mivel végig szem előtt tartottam, hogy az emberi hallás miként működik, így a sávok vágási frekvenciáját logaritmikus intervallumokra osztottam 20 Hz és 20 kHz között.

5 Grafikus felület és vizualizáció megvalósítása

Sokan alulértékelik a kezelőfelület fontosságát, viszont egy ergonomikus, intuitív interfész legalább olyan fontos, mint a mögötte levő háttérfolyamatok. Így törekedtem egy reszponzív, információgazdag GUI (graphical user interface) létrehozására.

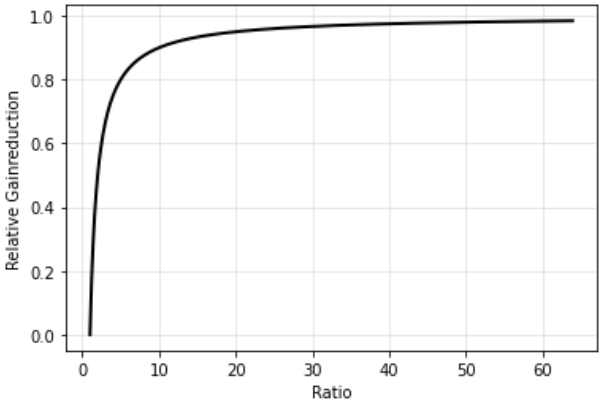
Mivel a VST SDK egyedül bitmap-os vizualizációt tett lehetővé, így a spektrum vizualizációja miatt kénytelen voltam egy másik keretrendszerre átváltani. A JUCE megfelelőnek bizonyult, ugyanis vizualizációhoz használható moduljai sokkal jobban személyre szabhatóak és a VST szabványt is támogatja, így a korábban megírt osztályokat könnyedén áthelyezhettem az új környezetbe. A JUCE kifejezetten audió szoftverek fejlesztésére lett tervezve, emiatt nagyon sok beépített függvénye van, amik használatát lehetőség szerint próbáltam minél jobban elkerülni, és saját megvalósításra törekedni.

5.1 Paraméterek

Első feladatom volt meghatározni, hogy mik azok a paraméterek, melyeket valós időben szeretném, hogy lehessen változtatni. Korábban, a 4.3. ábra szemléltette, hogy mely alapvető kontrollálási lehetőségekkel láttam el a szoftverem, viszont mivel több kompresszorból áll az effekt, így azt is el kellett döntenem, hogy melyek legyenek globális, és melyek lokális, tehát kompresszoronként változtatható paraméterek:

- A bemeneti és kimeneti erősítés természetesen két csúszkával megoldható. Fontosságuk, hogyha erősebben szeretnénk az effekt modulációját hallani a jelen, akkor a bemenő jelet érdemes növelni, viszont így sok esetben a kimenetet csökkenteni kell a megfelelő kivezérlés eléréséért. Ezeket a paramétereket ±32 dB-es intervallumon lehet mozgatni.
- A dinamikaszabályozók vezérléséhez a felfutási és elengedési idő mellett a *ratiot* és típust globális paramétereknek választottam, hogy az effekt összhangzása egységes jellegű legyen és ne legyen túl bonyolult a kezelőfelület sem. A felfutási és elengedési időket logaritmikusan skáláztam, mivel fontos, hogy finomhangolni tudjuk a rövid időegységek területén az effektet. Hasonlóan az időegységek skálázásához a *ratio* is logaritmikus osztással lett megvalósítva, mivel, ha azt vizsgáljuk, hogy a *ratio* függvényében mekkora redukciót alkalmazzunk, akkor

észrevehetjük, hogy egy hiperbolát járunk be ekkor, ahogy ezt az 5.1. ábra is mutatja. Igaz a logaritmus és hiperbola közel sem ugyanaz a függvény, viszont hasonlóan viselkednek [1, ∞) tartományban (mivel megfelelően transzformálva van a hiperbola) így helyettesíthetőek. Főként, hogy ez csak egy állítható paraméter skálázása és ennek a közelítésnek a célja kizárólag ergonómiai okokból történik.



5.1. ábra. Relatív kompenzálás a ratio függvényében

- A *threshold* paramétereket választottam végül egyedül lokálisnak, így ezzel kellően precíz vezérlést érhetünk el a szoftveren és egyben megmarad az ergonómiája a kezelésnek. Ezeket úgy helyeztem el, hogy a spektrumon az adott sávon és adott erősítésen helyezkedjenek el, így pontosan látni, hogy mikor kerül áteresztő tartományba az adott sáv.
- Végül a mix paramétert egy egyszerű lineáris csavarható felülettel valósítottam meg, mely 0 – 100 % között állítható.

5.2 Spektrumvizualizáció megvalósítása

Az audió szoftverek vezérléséhez sokszor vizuális visszacsatolást is kapcsolnak, hogy könnyebb legyen megtalálni a problémás területet, vagy csak hogy egyszerűbben meg lehessen érteni, hogy mit csinál az adott effekt vagy szintetizátor. Erre többféle megoldást is alkalmaznak, ami általában az eszköz funkciójától függ, ahogy azt az 5.2. ábra is mutatja. Csak az effektek grafikus megjelenítését fogom részletezni, mivel a szakdolgozatom ezzel a kérdéskörrel foglalkozik.

A késleltetés alapú effektek működésének vizualizációja kifejezetten nehéz és bonyolult. Sok esetben meg se próbálkoznak ezzel, néhány esetben a flanger és phaser

effekteket az egymáshoz viszonyított késleltetések mozgásával, többnyire egymáshoz relatívan mozgó pontokkal vizualizálják, vagy átviteli függvényükkel, ami általában egy fésűszűrő. A zengetőket jellegzetesen impulzusválaszukkal szokták vizualizálni. A szűrőket az átviteli karakterisztikájukkal és sok esetben a jel spektrumával vizualizálják.

A dinamikaszabályozók vizualizációja általában kétféleképpen történik: vagy a jelleggörbe grafikonja látható az adott beállítással, vagy az aktuális erősítést mutatják egy skálán.

5.2. ábra. Ableton Live 11 alapvető effekteinek kezelőfelülete

Mivel az általam megvalósított dinamikaszabályozó frekvenciafüggő így amellett a megoldás mellett döntöttem, hogy két spektrumot láthassunk (a nyers jel és a processzált jel) és a threshold értékeket ráhelyezem úgy, hogy a sík mindkettő számára ugyanúgy

legyen skálázva és azonos szélsőértékekkel legyen vizualizálva. Így a problémás frekvenciákat nem csak hallás alapján tudjuk processzálni, hanem vizuálisan is tudjuk vizsgálni a jelet.

Ahhoz, hogy a jel spektrumát meg tudjuk jeleníteni, az idő szerinti analízisből át kell transzformálnunk a jelet frekvencia szerinti ábrázolásra. Ezt a jel diszkrét Fourier transzformáltjával (DFT) lehet megtenni, viszont ennek a számítási bonyolultsága N elemre $O(N^2)$, ami számunkra nem elfogadható, így egy nagyságrendekkel hatékonyabb algoritmust használtam, a Fast Fourier Transform (FFT) metódust. Ennek bonyolultsága N elemre $O(N * log(N))$, ami látható, hogy a nagyobb adathalmazokra effektívebben számol. Így ez tökéletes nekünk, ugyanis nagyságrendileg, ha a mintavételi frekvencia $4*10^4$ Hz, akkor a DFT-vel 10^8 nagyságrendű számolást kellene elvégeznünk másodpercenként míg az FFT-vel csak 10^5 nagyságrendűt. A megvalósításhoz az FFTW könyvtárat [7] használtam. Ahhoz, hogy a transzformáció kimenete érvényes legyen, először a bemeneti adatokat elő kell készíteni.

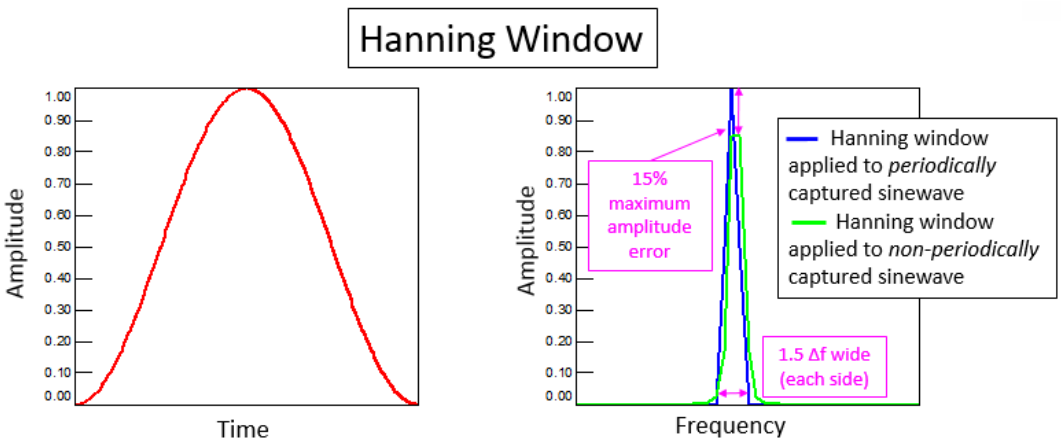
Az FFT megfelelő működéséhez garantálni kell, hogy mindig ugyanannyi mintát kapjon, emellett az effektív működéséhez optimális, ha 2^n darabnyi ez a szám$(n \in N)$. Sajnos a plugin által kapott pufferek ezeket a feltételeket több szempontból sem tudják kielégíteni, így létre kellett hoznom egy körpuffert, mely konvertálja a bejövő puffereket FFT kompatibilis méretűre. Ehhez létrehoztam egy osztályt `circularBuffer` névvel, mely felelős ezért a feladatért. Az inicializáláskor meg kell adni, a puffer kívánt méretét és hogy hányszor legyen nagyobb ennél. Ez amiatt fontos, hogy legyen a pufferben tartalék, és amíg ír, addig is lehessen érvényes kimenet. Ezt az értéket 2-re vettem, ami elegendőnek bizonyult. A megvalósításhoz két puffert alkalmaztam: A körpuffert, amibe egyenesen íródik a bejövő adat, és a kimeneti puffert, melybe csak az érvényes kimenetet másolom és ezt adom át az FFT számolásához. Hogy minél kevesebb legyen a késleltetés, a puffer írásához és olvasásához azt a megoldást használtam, hogy minden egyes bemásolt (a plugin által szolgáltatott) új puffer után azonnal bemásoltam a kimeneti pufferbe az utolsó érvényes FFT méretnyi mintát. Itt figyelni kellett, hogy mikor érjük el körpuffer végét és kezdődik előröl a címezés. Látható, hogy így a kimeneti puffer a körpufferen csúsztatva megy végig és így elkerülhető, hogy véletlenül a kimeneti puffer közepén legyen az aktuális írófej.

Ennek megvalósításában a legnehezebb feladat az volt, hogy az olvasás és írás két teljesen különböző ütemben történik. Az audió pufferek $\frac{mintavételi\ frekvencia}{puffer\ méret} \sim 100\ Hz$ körüli, a grafikus felület pedig általam beállított, 60 Hz frekvenciával hívódnak. Ez a probléma ott jelentkezett, hogy néha előfordult az az eset, hogy miközben az FFT épp olvasta ki a kimeneti pufferből az adatokat, azt közben a körpuffer elkezdte felülírni az új adatokkal, ezzel egy ugrást képezve a vizsgált jelen. A problémát úgy orvosoltam, hogy a kimeneti puffer valójában két puffer méretnyi, és csak azt a részét adom oda az FFT-nek, ami érvényes és közben a másik íródhat.

Az így kapott pufferek mérete ezzel a megoldással nem függtek többé a kapott bejövő pufferektől, viszont azt nem tudtuk garantálni, hogy a puffer két vége 0-ban érjen véget (tehát minden az ablakban levő hullám 0 fázistolással kezdődjön, és egész számú többszöröseként tartózkodjon a vett mintában). Ez az FFT számításnál amiatt fontos, mert ha ez nem teljesül, akkor a kiszámolt spektrumon megjelenhetnek olyan komponensek is, amik nem voltak a mintavételezett jelben. Ennek eliminálásához érdemes ablakozást használni, amihez én a Hanning-ablakozást választottam [8], mely egy emelt koszinuszablak, ahogy az egyenlet is mutatja,

$$Y[i] = X[i] \times \frac{1}{2}\left(1 - cos\left(2\pi * \frac{i}{N-1}\right)\right)$$

ahol i a minta sorszáma nullától $(N - 1)$ -ig és N az ablak mérete. Ahogy azt a 5.3. ábra is mutatja, ennek a típusú ablakozásnak az előnye, hogy ha nem fázisban mintavételezünk egy szinusz jelet, akkor is maximum 15%-kal térhet el az eredeti jel amplitúdójától az ablakozott jel.



5.3. ábra. idő (bal) és frekvencia (jobb) szerinti ábrázolása a Hanning-ablaknak és esetleges amplitúdó hibájának [8]

Az FFT kiszámolását a grafikus felülethez csatoltam, így csak akkor számolta ki a jel transzformáltját, ha szüksége volt az ábrázolásához. Fontos még, hogy a spektrum egy sávra tudja kiszámolni a jelet egyszerre, ezért, hogy ne legyen bonyolult a kapott ábra a sok ábrázolandó függvény miatt, a bal és jobb oldali csatornákat átlagoltam és így ábrázoltam.

A transzformációt követően a kapott puffert még át kell alakítani a vizualizációhoz kompatibilissá, ugyanis egy komplex elemekből álló puffert számol ki, amiből nekünk csak az amplitúdóra van szükségünk. Az így kapott amplitúdókat még át kellett skáláznunk a mintavételezés függvényében, tehát $\frac{N}{2}$ -vel el kell osztanunk őket és a Hanning-ablakozás miatt vesztett a jel az energiájából [9] ezért ebben az esetben 2-vel fel kell szoroznunk a kapott amplitúdókat tehát összesen $\frac{4}{N}$ -nel kell szoroznunk. Végül átváltottam az egészet decibel mértékegységekre, az ismert $20 \times \log(x)$ képlettel.

Az amplitúdókat hasonlóan a felfutási és lefutási időhöz, átlagoltam, mivel a megvalósított FFT számolás valós ideje miatt, nagyon gyorsan mozog a szem számára.

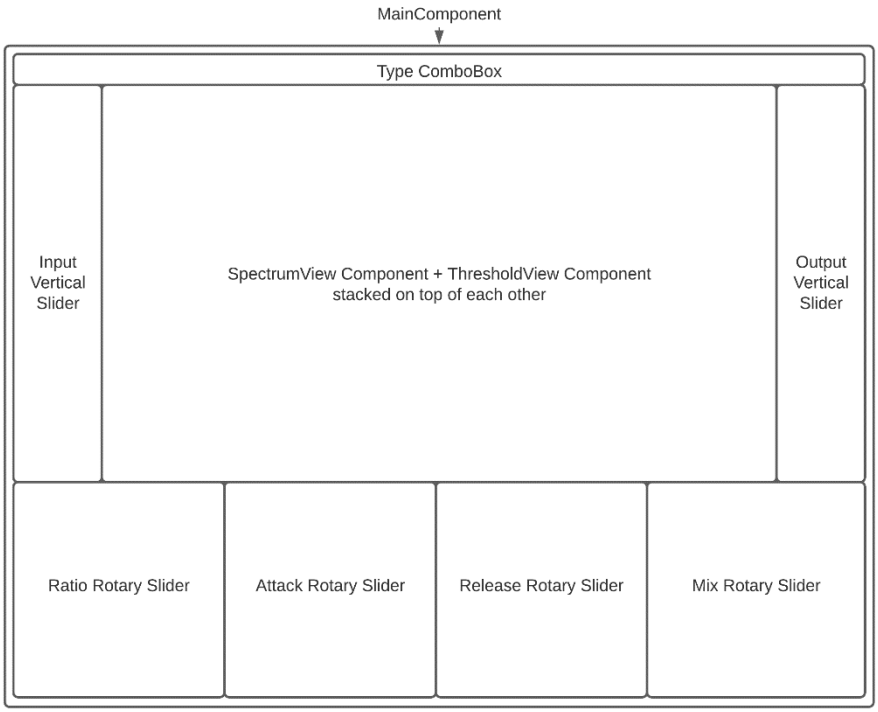
A mintavételezés miatt a kiszámolt spektrum a minták felére tükrös, ugyanis az FFT nullától mintavételi frekvenciáig lineárisan szétosztva N (ablakméretnyi) darab komplex vektort számol ki, emiatt nekünk csak az első $\frac{N}{2}$ darabnyi minta kell. A kiszámolt amplitúdókhoz az adott frekvenciák meghatározásához a fentebb írt lineáris eloszlás miatt $\frac{i \times f_s}{N}$ képletet használtam, ahol az i a minta sorszámát jelöli.

A teljes FFT processzáláshoz az általam megvalósított `FFTCalculator` osztály felelős, ami az audióprocesszor egyik publikus objektuma, így az editor hozzá tud férni a függvényeihez bármikor.

5.3 Grafikus felület implementálása JUCE-ban

A plugin vezérléséhez – leszámítva a dinamikaszabályozó típusát meghatározó leugró ablakot – a JUCE beépített *Slider* osztályát [10] használtam, aminek személyre szabhatósága elegendő volt a teljes vezérlés megvalósításához. a *threshold* vezérlését is ezzel az osztállyal oldottam meg úgy, hogy csak az aktuális pontot hagytam láthatónak és a többi részét átlátszóvá tettem. Az alsó forgatható paraméterek is a *Slider* osztály egyik változata. A dimanikaszabályozó típusát felül lehet kiválasztani, ha a típus nevére

rákattintunk nyílnak le a lehetőségek. Ezt egy *ComboBox* nevű modullal valósítottam meg. A kezelőfelület moduljainak elrendezését az 5.4. ábra mutatja.



5.4. ábra. A szoftver moduljainak elrendezése a grafikus felületen

Hasonlóan a VST SDK- hoz itt is az editor felelős a grafikus felületért. A JUCE környezetben ezt egy fő *Component* osztállyal valósították meg, melyre különböző modulokat lehet illeszteni (*Slider* és *ComboBox*), többek között új *Component* osztályokat is. A *Component* osztálynak két fő függvénye van, melyek felelősek a megjelenítésért: a *paint* és *resized*. A *resized* függvény akkor hívódik meg, ha létrehozzák vagy átméretezik a plugint. A *paint* pedig a plugin megnyitásakor és minden olyan alkalommal, amikor meghívják a *repaint* függvényt, hívódik meg. Emiatt érdemes a dinamikus elemek elhelyezkedését a *resized* függvényben megadni, míg a *paint*-ben a statikus, például szövegek és háttér grafikákat megvalósítani.

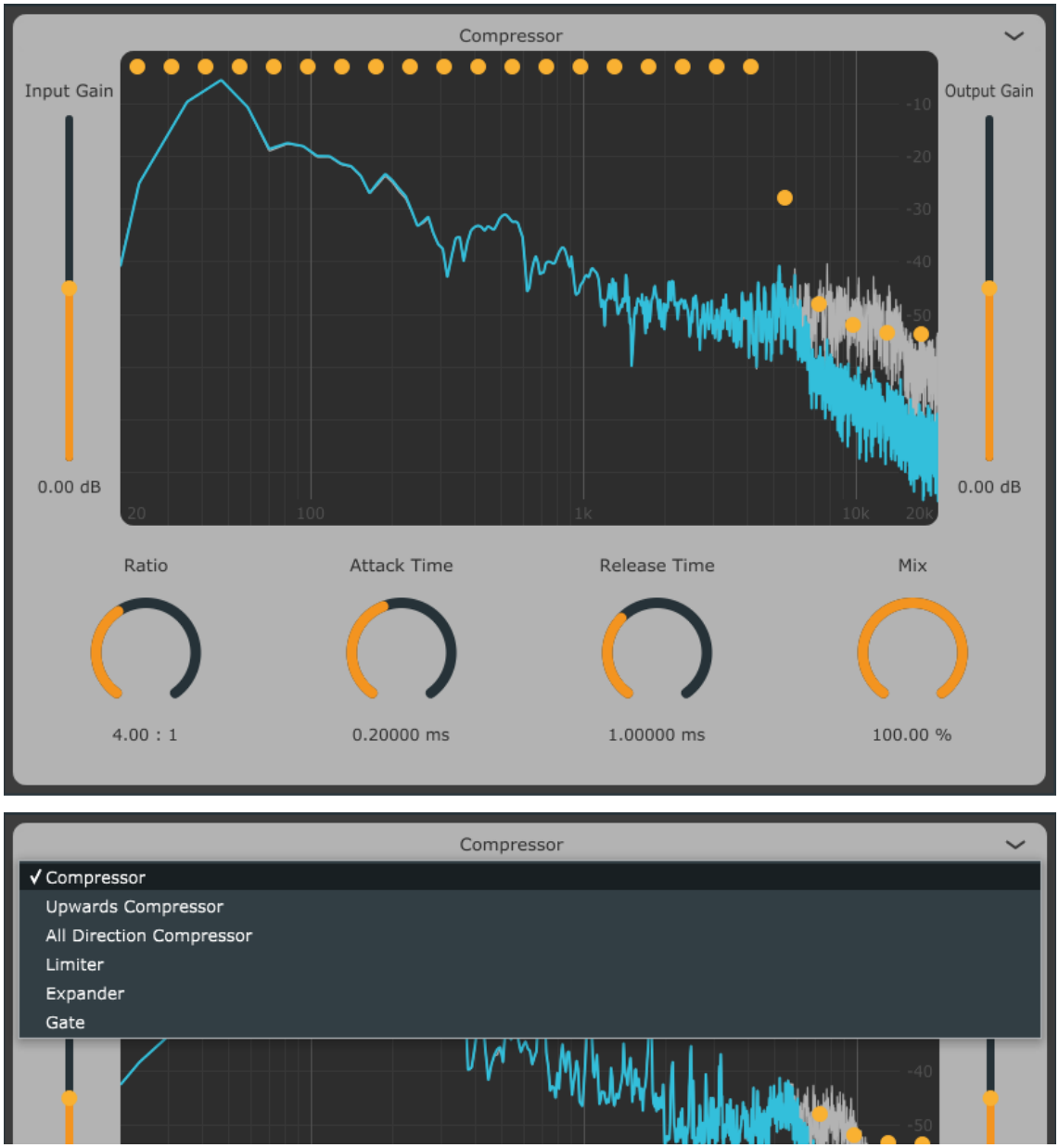
A spektrum vizualizálásához létrehoztam a `SpectrumView` osztályt, ami a leszármazottja a *Component* és *Timer* osztályoknak. A *Timer* osztály segítségével lehet animálni különböző grafikai elemeket, mivel az osztálynak létezik egy `timerCallback` függvénye, mely minden, a konstruktorában meghatározott időközönként (én 60 Hz-re állítottam) meghívódik, amit én felülírtam a saját megvalósításommal. Először elkéri az adott, FFT által kiszámolt puffereket és ezután a *repaint* függvényt hívom meg, hogy frissítsem a grafikát.

A *paint* függvényben JUCE által implementált pontokat hozok létre a kiszámolt koordinátákkal, melyet utána egy *path* nevű modullal összekötöttem, ezzel létrehozva a spektrumot.

Mivel a *processor* osztály hozza létre az editort, így azt a megoldást választottam, hogy létrehozásakor átadja saját magát a *Component*-nek, így a *processor* publikus változói és függvényei elérhetőek lesznek számára. Első feladat a grafikon megfelelő skálázása, és szélsőértékeinek meghatározása volt. A vízszintes tengelyt logaritmikus, 20 Hz és 20 kHz közötti tartományon skáláztam az emberi hallás módja és határai miatt. A függőleges tengelyt 0 és -90 dB közötti tartományon lineárisan paramétereztem, mert ezzel jól le lehetett fedni az ember számára hallható dinamikatartományt.

Mivel az amplitúdókat már átskáláztam dB értékekre ezzel csak az volt a dolgom, hogy a [0, -90] es tartományon megjelenítsem. A frekvenciák meghatározását a vízszintes tengelyen úgy valósítottam meg, hogy először [0, 1] intervallumra logaritmikusan átskáláztam, amit utána már csak be kellett szoroznom a felület szélességével.

Hogy jobban lássuk a szoftver működését 2 spektrumot vizualizáltam, a szabályozatlan összeadott bemenő jelet (*dry*) és a szabályozott kimeneti jelet (*wet*). Emellett egy *gridet* is létrehoztam, a pontosabb analizálás érdekében. A színek és paraméterek kinézetét az Ableton Live dizájnja ihlette ahogy azt az 5.5. ábra is mutatja.



5.5. ábra. A megvalósított plugin grafikus felülete (felül) és ComboBox működésének módja (alul)

6 Tesztelés és összehasonlítás

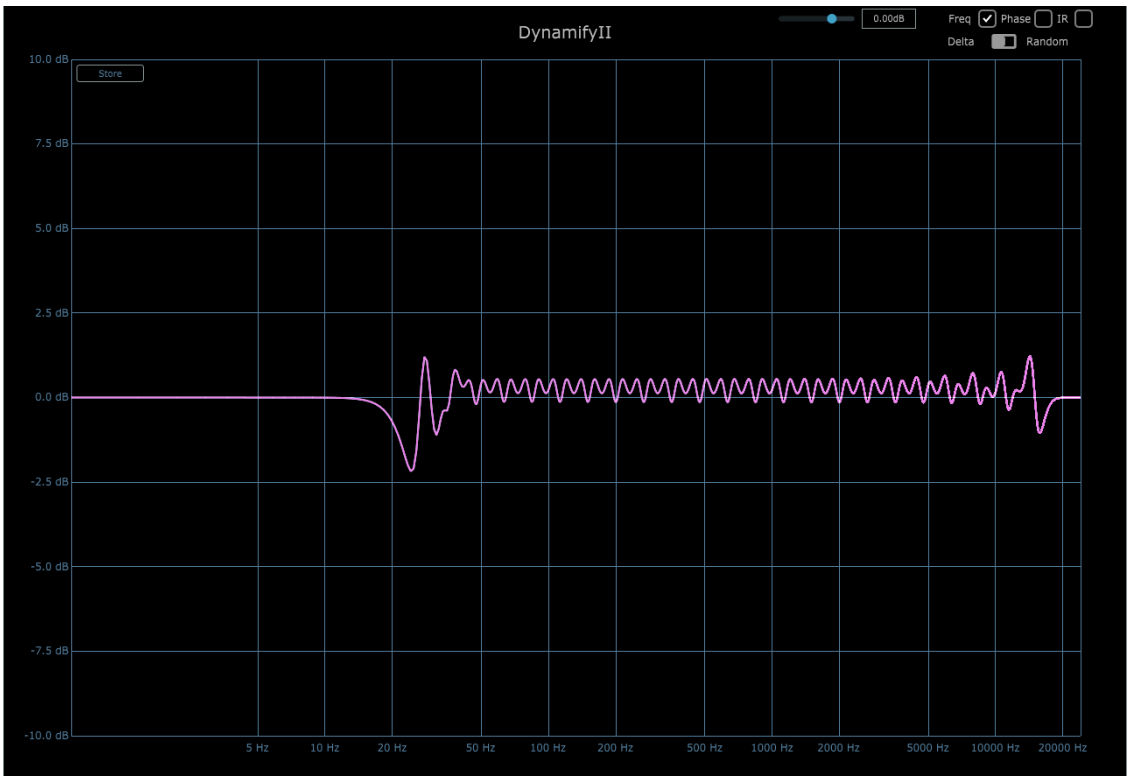
6.1 Megvalósított szoftver tesztelése

A szoftverfejlesztés egyik legfontosabb szakasza a tesztelés. Ezen a ponton dől el, hogy az adott szoftver eleget tesz-e az elvárásoknak, és kiadható állapotban van-e. Ez általában a szoftver fejlesztése közben folyamatosan zajlik, miután az adott funkció elkészül. Ez az én esetemben sem volt máshogy, viszont a szakdolgozatom szerkezeti felépítése miatt jobbnak láttam, ha ez egy külön pontot kap.

A különböző funkciók tesztelését a szűrőelrendezés vizsgálatával kezdem, ami után a dinamikaszabályozókat vizsgálom és végül a megvalósított FFT vizualizációt fogom értékelni. Ehhez a DDMF által fejlesztett Plugindoctor szoftvert fogom használni.

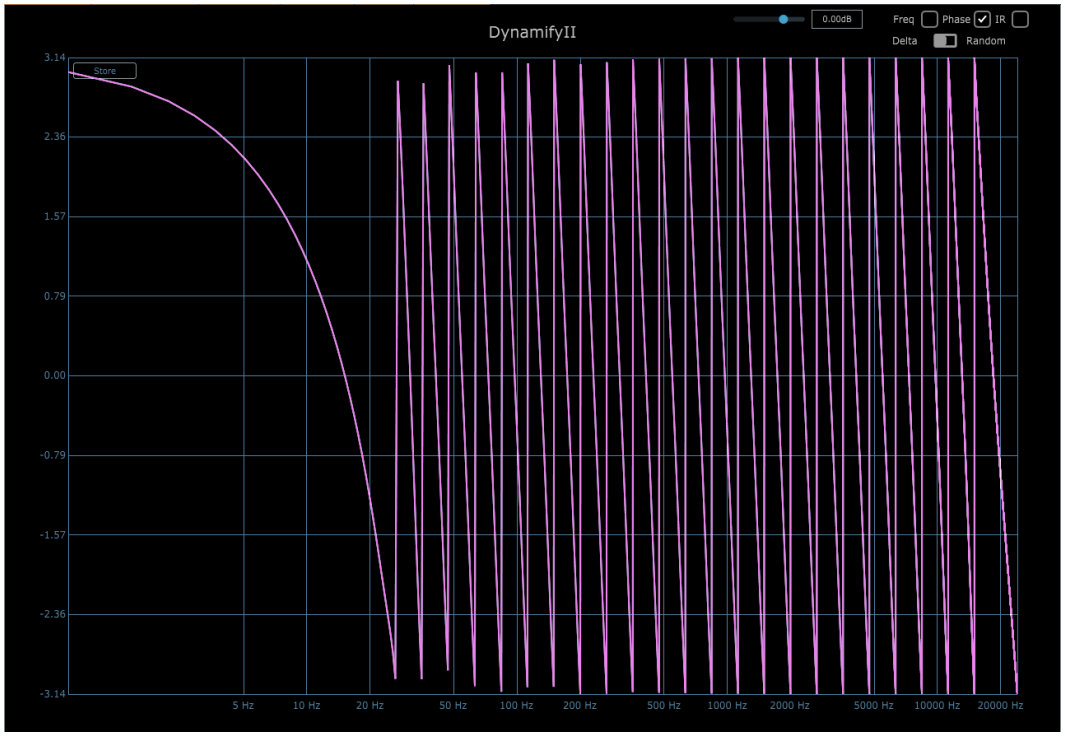
6.1.1 Szűrőelrendezés

A plugin mix paraméterét nullára véve vizsgáltam az rendszer átviteli függvényét. Amint azt a 6.1. ábra szemlélteti, az amplitúdó kilengés nem haladja meg a 2.5 dB-t, ami bőven várakozáson felüli.

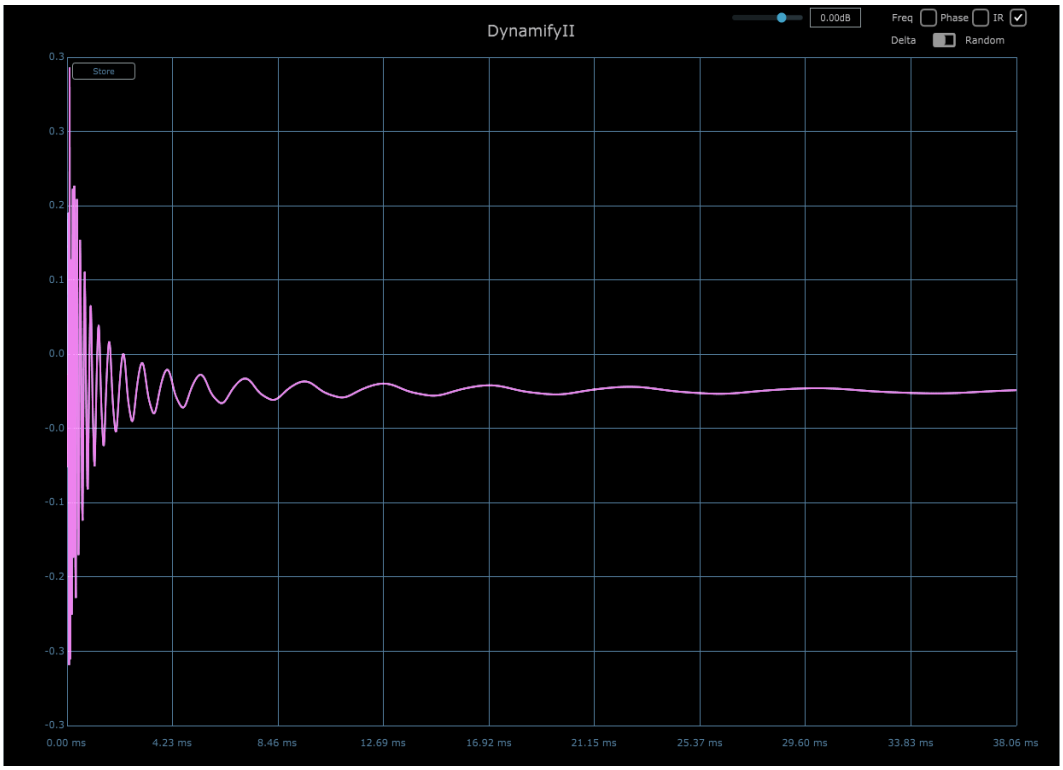


6.1. ábra. Megvalósított plugin átviteli függvénye.

A fázismenet is eleget tesz az elvárásoknak, mint ahogy a 6.3. ábra mutatja, többször körbefordul a fázis, ezzel egy frekvenciafüggő késleltetést víve a rendszerbe mely az impulzusválaszon jól látható (6.2. ábra). Emellett az is leolvasható, hogy 20 ms alatt közel teljesen lecseng a válasz, ami elfogadható.



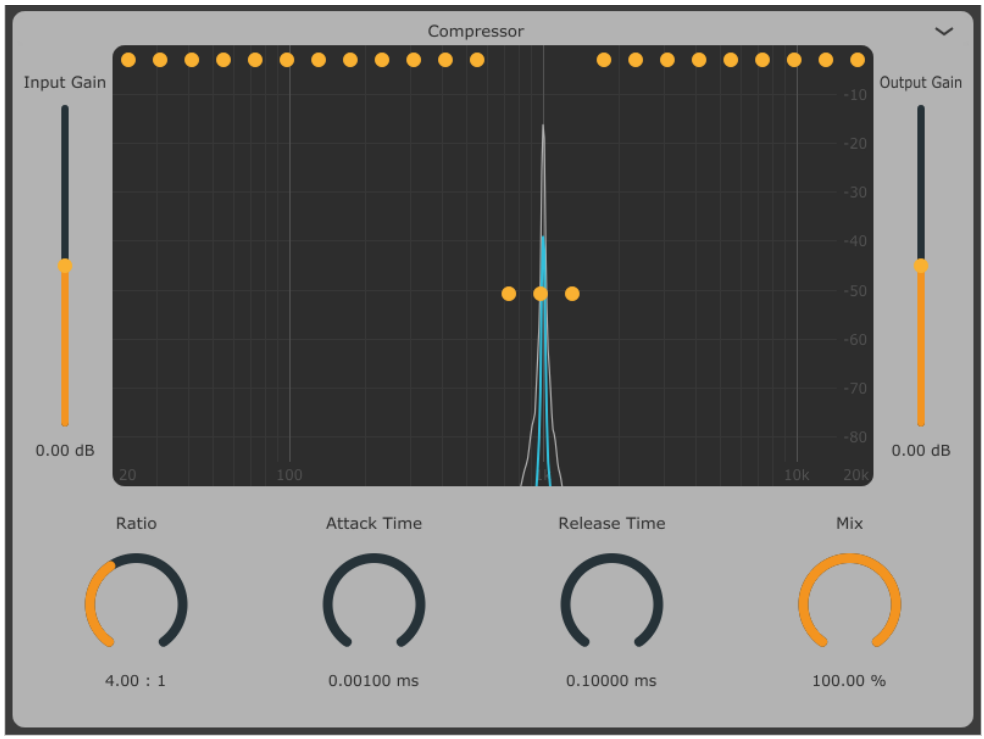
6.3. ábra. Megvalósított plugin fázismenete



6.2. ábra. Megvalósított plugin impulzusválasza

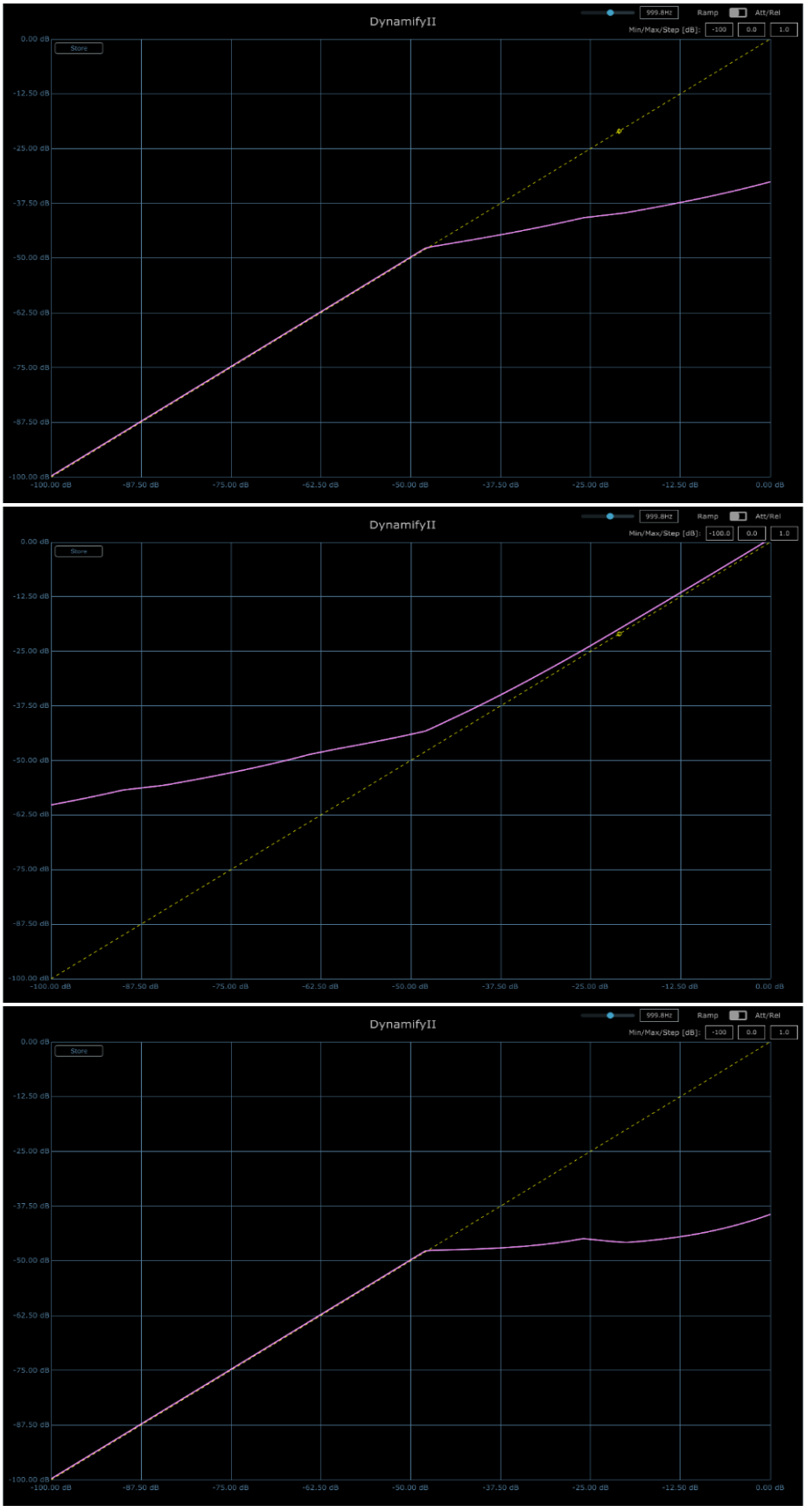
6.1.2 Dinamikaszabályozás

Ehhez a Plugindoctor dynamics analízis opcióját választottam, mely segítségével az adott szoftver jelleggörbe grafikonját tudja meghatározni. A jobb felső sarokban látható az analizáló jel és skála. Én 1000 Hz-es szinuszjelet választottam [-100 dB, 0 dB] intervallumon 1 dB-es lépésközzel. A plugin 4:1 ratioval, minimális felfutási és lefutási idővel, és az adott frekvencián és körülötte a *threshold* értékét szemmértékkel a -10 dB-es értékre állítottam ahogy azt a 6.4. ábra is mutatja.

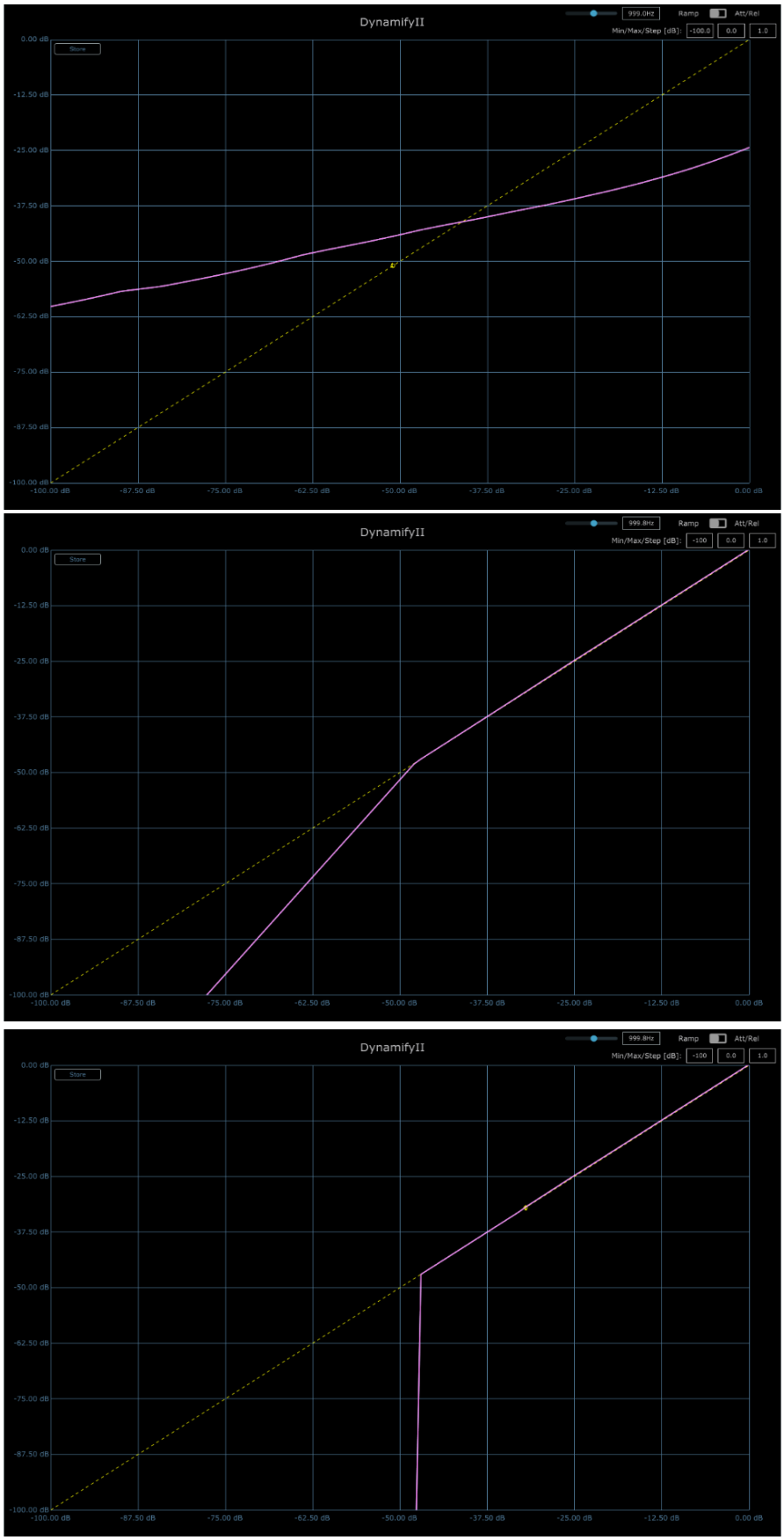


6.4. ábra. Megvalósított plugin beállításai jelleggörbe vizsgálata közben

A 6.5. ábra és a 6.6. ábra mutatja a megvalósított dinamikaszabályozók viselkedését. Azonnal feltűnik az összes diagrammon, hogy nem teljesen -50 dB-nél van a töréspont, hanem felette. Ez lehet több okból is. Az egyik, hogy a *threshold* értéket nem tudjuk pontosan beállítani, emellett a *slider* skálázása sem tökéletes, mivel, ha megfigyeljük a fölső pontban 0 dB értékre vannak állítva viszont a grafikonon olyan -2 dB körül helyezkednek el. Egy másik tényező a sávok RMS számolásával történik, ugyanis lehet, hogy nem teljesen az áteresztő tartományban tartózkodik a jel, akkor nem biztos, hogy pontos értéket mér. Emiatt húztam le a mellette levő sávok thresholdjait is.



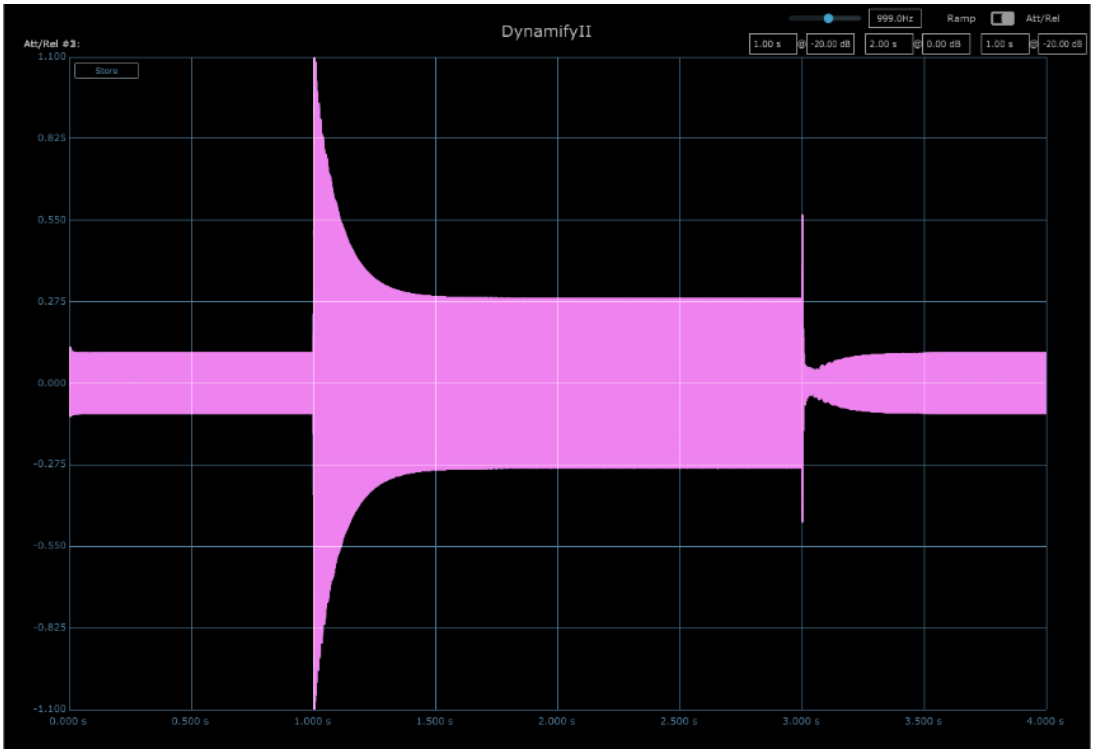
6.5. ábra. Felülről lefelé: halkító és hangosító kompresszor, végül limiter jelleggörbe grafikonja



6.6. ábra. Felülről lefelé: kétirányú kompresszor, expander és zajzár jelleggörbe grafikonja

Egy másik feltűnő jelenség a grafikonokon, hogy nem tökéletesen egyenletes a jelek osztása, szorzása. Ez amiatt lehet, mert a sávoknak van egy kicsi átfedése a szűrők tökéletlensége miatt, így több sávban is előtűnhet ugyanaz a jel más amplitúdóval, ahol a processzálás másként fog működésbe lépni és ezeket a kisebb torzulásokat láthatjuk ezeket a helyeken.

A 6.7. ábra mutatja az attack és release működését, melyen látható, hogy egy kis váratlan tüske megjelenik a release kezdetén, viszont ez nem volt hallható számomra, és nem értem pontosan, hogy miért mutathatja a grafikon.

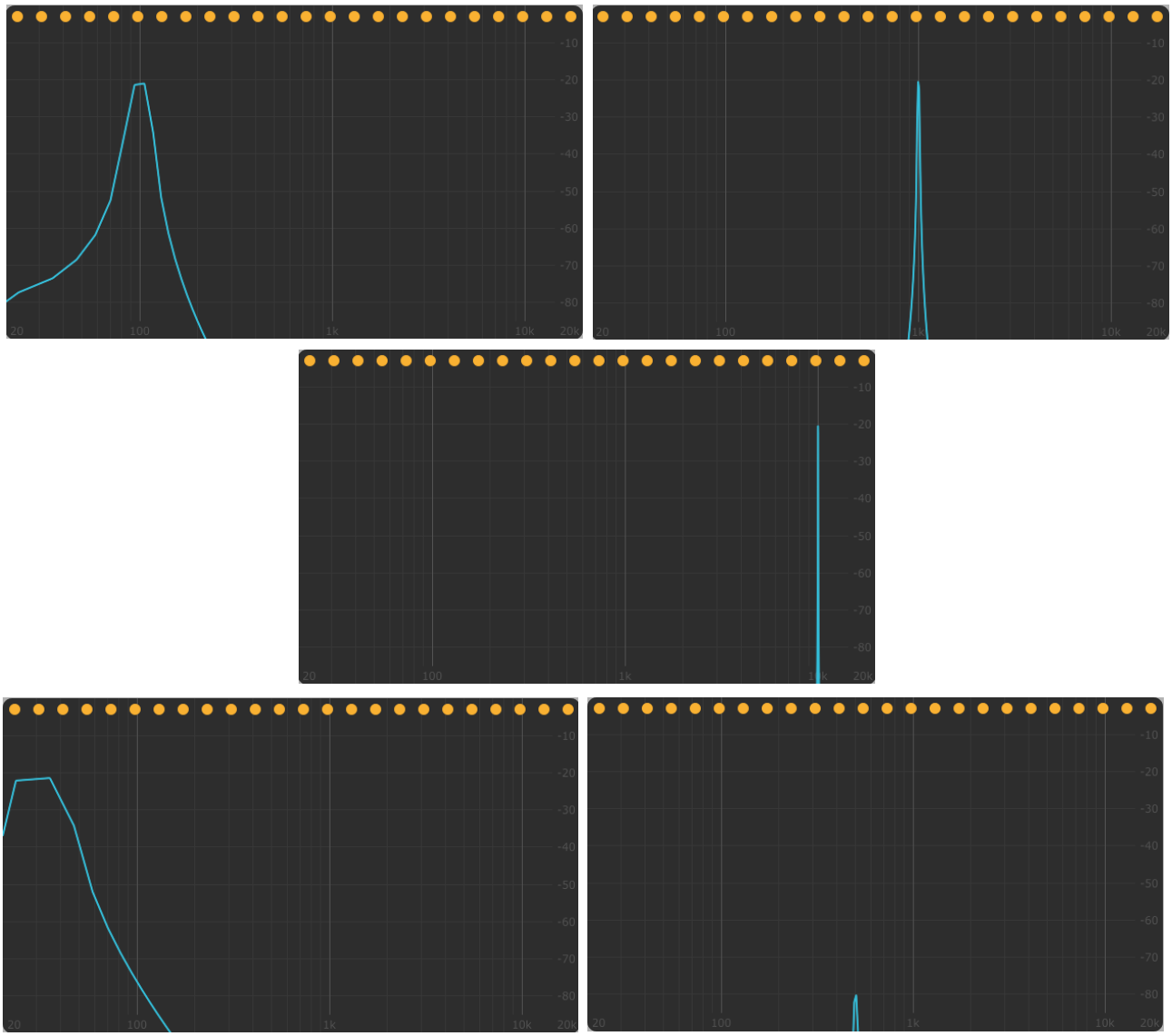


6.7. ábra. Attack és release idő szemléltetése amplitúdó-idő grafikonon, mindkettő 100 ms-ra állítva

6.1.3 Vizualizáció és vezérlés

Itt a spektrum pontosságát vizsgáltam olyan jelekkel, amiknek pontosan tudtam, hogy milyen paraméterei vannak. A 6.8. ábra mutatja, hogy közel hibátlanul ábrázolja a megvalósított spektrum az analizálásra használt jeleket. Észrevehető, hogy a frekvencia csökkentésével a felbontás romlik, ami logikus, mivel logaritmikusan ábrázoltuk a lineárisan kiszámolt pontokat. Érdemes még megvizsgálni a 30 Hz-es tartományt a 6.8. ábra segítségével, ahol már az alacsony felbontás miatt nehéz megmondani, hogy pontosan milyen frekvenciájú a szinusz és az amplitúdó is torzul az ablakozás hibája

miatt. Az én megvalósításomban 4096 mintára számolom ki az FFT-t, így 2048 pontot ábrázolok. Ennek növelésével lehet javítani a felbontáson, viszont ezzel növekszik a számítási igény (nem jelentősen) és a késleltetése a vizualizációnak, így én megfelelőnek találtam ezt a mennyiséget. A megvalósított spektrum vizualizáció minden területen megfelelt az elvárásoknak.



6.8. ábra. Szinuszjelek vizsgálata, először -20 dB amplitúdójú sorra 100 Hz, 1000 Hz, 10000 Hz, 30 Hz és végül -80 dB amplitúdójú szinuszjel 500 Hz frekvencián

6.2 Összehasonlítás hasonló funkcionalitású szoftverekkel

Piackutatásom érdekes fordulata volt, hogy ugyanilyen funkcionalitású szoftvert nem találtam. A piacon levő többsávos dinamikaszabályozók többnyire 3 sávra, maximum 5 sávra képesek bontani a jelet, egyedül 2 olyan szoftvert találtam, amely közelíti az általam megvalósított eszközt: Az Oeksound által programozott Soothe2 és a FabFilter által fejlesztett Pro MB. Így ezeket a szoftvereket fogom összehasonlítani az általam megvalósított pluginnal.

Mindkét szoftver jellegzetesen a zenei utómunka során használatos, emiatt csak apró változtatásokat hajtanak végre a jelen. A Pro MB egy maximum 6 sávra bontható kompresszor, míg a Soothe2 a jel spektrumát figyelve az adott frekvencián kompresszál, sajnos pontosan nem tudni mekkora felbontásban.

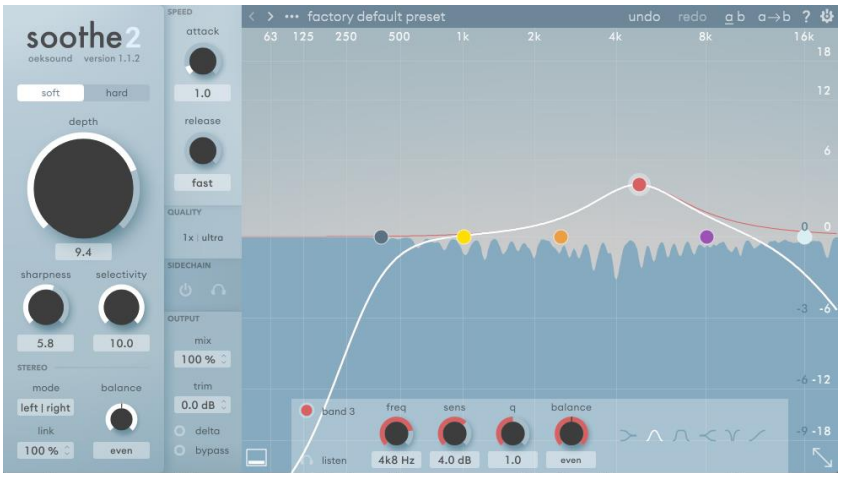
Mielőtt elkezdeném teszteli a szoftvereket, érdemes a kezelőfelületüket először megismerni, hogy jobban megértsük, miért viselkedik az adott módon az effekt. A Fabfilter Pro MB közel minden paraméterét sávonként külön lehet állítani, emellett a sávok számát is módosítani lehet maximum 6-ig. Érdekes, hogy a ratio helyett egy range paramétert valósítottak meg, amivel a kompresszió intervallumát lehet megadni. Valószínűleg amiatt döntöttek így, mert ez egy felhasználóbarátabb kifejezés. Emellett a felfutási és lefutási időkről is egyedül százalékos információt adnak.

6.9. ábra. Fabfilter Pro MB plugin kezelőfelülete

A Soothe2 ehhez képest még jobban eltávolodott a paraméterek számszerűsítéséről. Egyedül olyan információkat tudunk meg, hogy a „semmi” és a „nagyon” között hol tartózkodunk. A használati utasításából idézve:

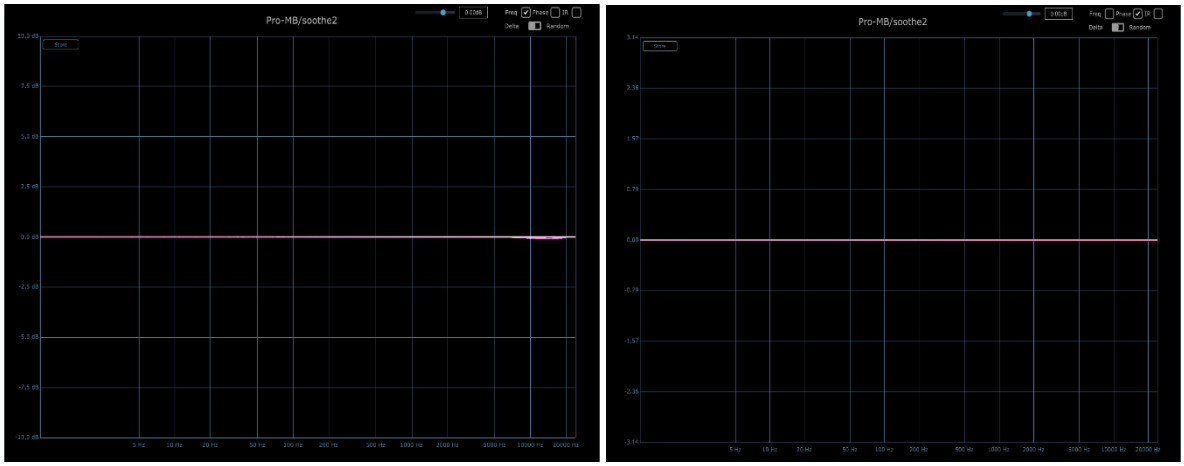
„The depth is shown in dB (-18 to 18) for convenience. Please note that the dB value displayed is referential to make it more familiar and intuitive to use, but it does not represent the absolute amount of changes the processing does.” [12]

Tehát összegezve, a „mélységet” szabályozó paraméter számozásának semmi köze a dB-értékekhez. Később írják is, hogy akár 60 dB-es redukció is létrejöhet egy sávban. Ahogy a 6.10. ábra mutatja, a jobb oldalt lehet befolyásolni, hogy mely frekvenciákat kompresszálja jobban a plugin.



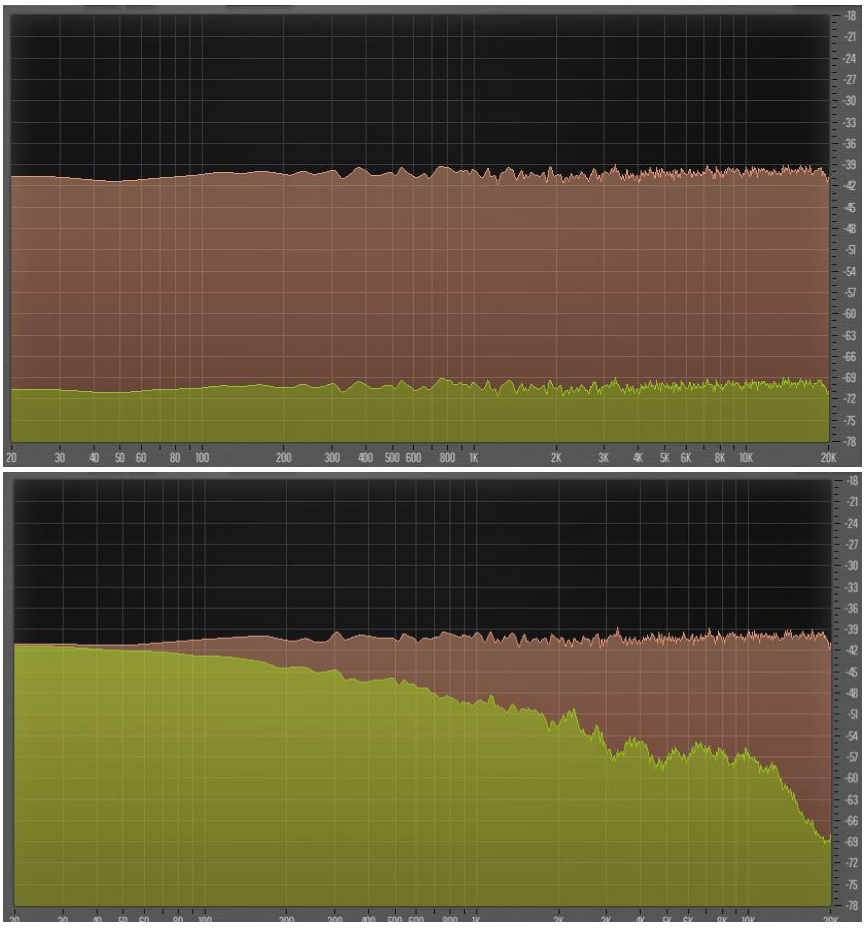
6.10. ábra. Oeksound Soothe2 plugin kezelőfelülete

Ezek tudatában először megmértem az átviteli karakterisztikájukat úgy, hogy csak be vannak kapcsolva az effektek, de nem csinálnak semmit. A 6.11. ábra mutatja, hogy nem fog se amplitúdó-, se fázistorzulás történni. Emiatt arra merek következtetni, hogy egyfajta dinamikus equalizerekként vannak ezek a szoftverek kivitelezve, egyedileg modellezve a szűrőket.



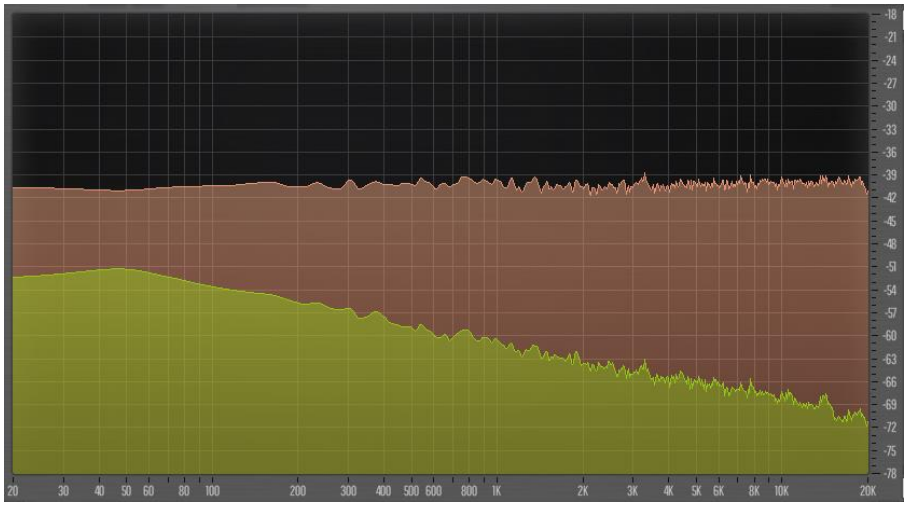
6.11. ábra. Pro MB (sötét rózsaszín) és Soothe2 (világos rózsaszín) átviteli karakterisztikája (balra az amplitúdó, jobbra a fázis)

A kompresszálási tartomány vizsgálatára a Plugindoctor nem bizonyult alkalmasnak, mert az analizáló jelet nem lehet állítani, és nem feltétlen kapunk reális eredményt. Ehhez Ableton Live-ban vizsgáltam meg fehér zajjal a különböző szoftverek viselkedését, úgy, hogy minden sávban kompresszálniuk kelljen. A Voxengo tulajdonában levő Span ingyenes spektrumanalizátorát használtam erre a tesztre, mivel ez képes egyszerre több spektrumot is megjeleníteni, így könnyen össze tudjuk hasonlítani a zaj spektrumát és a szoftver módosításait. Mindkét szoftvernél próbáltam a lehető legnagyobb kompresszálást elérni, amiknek az eredményeit a 6.12. ábra mutatja. Látható, hogy a Pro MB a teljes spektrumon ugyanannyit kompresszál sávfüggetlenül. A range paramétert 30 dB-re állítottam az összes sávon és leolvasható, hogy -40 dB-es átlagerősségű jelből -70 dB értékűre csökkent. A Soothe2 ennek ellenében sokkal csekélyebb és közel sem konstans csillapítást hajtott végre, bár a leírása alapján főként egy-egy szinuszos kiugrás kompenzálására használható effektíven. Ezekhez képest az általam megvalósított szoftver valahol a kettő között helyezkedik el, ahogy ezt a 6.14.



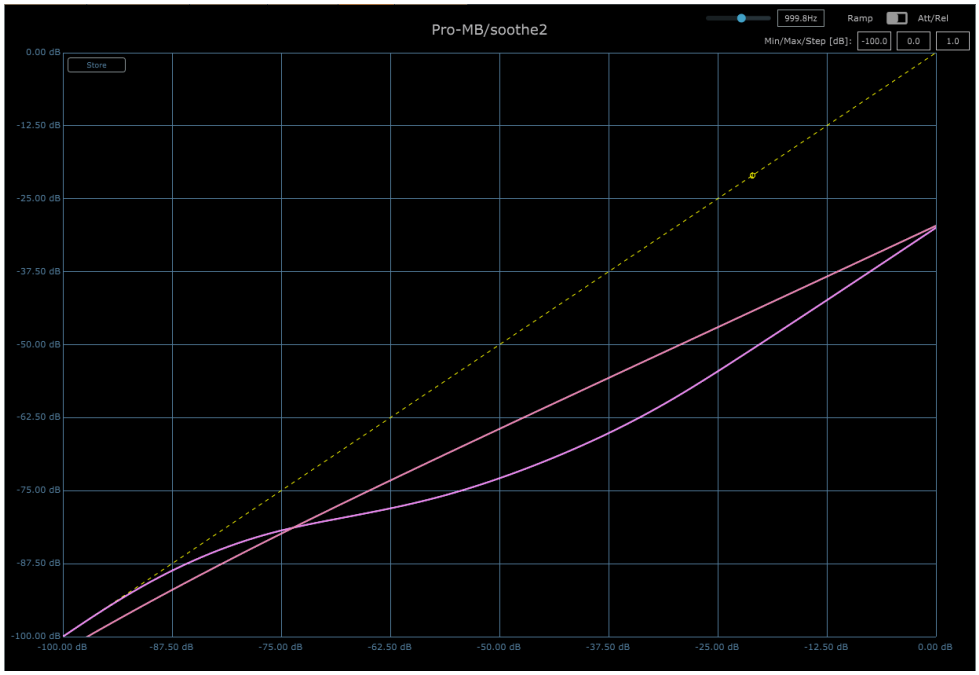
6.12. ábra. Pro MB (felül) és Soothe2 (alul) kompresszálási tartományon belüli fehérzaj tesztek (piros: kompresszálatlan, zöld: kompresszált jel)

ábra mutatja. Ez az eredmény a logaritmikus sávfelosztás miatt logikus és megfelel az elvárásainknak, mivel a kisfrekvenciás tartományban kevesebb teljesítmény jut egy sávra, így az RMS számláló nem mér akkora amplitúdót. Valószínűleg a Pro MB vagy másik számolási módot használ a jelszint mérésére, vagy kompenzálja a sávokat.



6.14. ábra. Általam megtervezett szoftver válasza fehérzajra

Ezután ugyanezzel a beállításokkal megmértem Plugindoctorral a jelleggörbe grafikonjukat, amiknek az eredményét a 6.13. ábra mutatja. Mivel a Pro MB szoftver *thresholdja* -90 dB-re volt állítva, érthető, hogy ilyen hamar elkezdi a kompresszálást és az is szépen látszik, hogy 30 dB-nél nagyobb nem lesz soha az eltérés. Érdekes viszont,



6.13. ábra. Pro MB (lilás rózsaszín) és Soothe2 (világos rózsaszín) jelleggörbe grafikonja

hogy a Soothe2 görbéjében nem látható egy *threshold* pont, hanem egy folytonos, egyre erősödő kompresszálást láthatunk.

Összességében látható, hogy az általam megvalósított szoftver drasztikusabb processzálási lehetőségei miatt főleg a hangok színezésére és *sound design* elemek megvalósítására alkalmasabb, míg az előbb tesztelt pluginok kiválóan használhatóak precíz editálásra, úgy, hogy közben a jel többi területe minimálisan változik.

7 Összegzés

A végleges megvalósított szoftver egy sokoldalú dinamikaszabályozó lett, mely egyediség a piacon, így sikerült elérnem célomat ezzel a projekttel. Valószínűleg amiatt nincs több hasonló szoftver, mivel a számítási igénye viszonylag nagy ahhoz képest, hogy ez „csak egy” effekt, amit minden sávra rá kéne tudni tenni. Ennek ellenére felhasználási területe nagyon sokrétű.

A különböző megvalósított módok tökéletesen kiegészítik egymást: Ha egy de-essert szeretnénk létrehozni, érdemes a zajzár funkcióval megkeresni, hogy melyik az a tartomány, ahol csak az ’sz’ és ’z’ hangok vannak. Ezt követően átállítjuk kompresszor módra és az elvárt hatás gyönyörűen hallható lesz. Másik alkalmazási lehetőség, ha egy dobszólóban a túl sok a pergődob lecsengése, akkor megkereshetjük, melyik frekvencia tartományban mozog, és az expanderrel csökkenthetjük úgy, hogy a lábdob és a cintányérok sértetlenül maradnak. Egy harmadik alkalmazási lehetőség, ha egy adott szintetizátor nem elég agresszív, akkor a *minden irányú* kompresszorral és a *thresholdok* megfelelő beállításával akár túlzásba is eshetünk, de erre van a mix paraméter, aminek segítségével enyhén adagolni tudjuk az adott effektet, ezzel egyfajta paralel kompresszálást alkalmazunk. Ezekhez csatoltan elérhetőek a *tests* mappában hanganyagok. Először mindig megmutatom az eredetit és utána a moduláltat. A de-esser esetén három verzió hallható egymás után, mert megmutatom a zajzárral kiszűrt hangot is.

A program forráskódja és exportja a *code* mappában megtekinthető.

7.1 Továbbfejlesztési lehetőségek

Természetesen egy szoftvert örökké lehet fejleszteni, így ami általában meghatározza, hogy végül mi minden valósuljon meg, az az időkorlát. Ez nálam is hasonlóképp volt. Mind hibajavítási, mind plusz funkciók implementálására lenne még lehetőség. Talán a legfőbb hiba a limiter megvalósításában volt, amiben nem is vagyok biztos, hogy kivitelezni lehetne jobban a többsávos megoldásban, ugyanis ennek az effektek a fő célja a teljes jel szabályozása, hogy semmiképp nem lógjon túl egy adott szinten.

A későbbiekben pedig mindenképp érdemes flexibilisebbé tenni, hasonlóan a Fabfilter Pro MB-hez (sávoknént külön állíthatóvá tenni az összes paramétert), és néhány globális változót is betenni, konkrétan egy csúszkát az összes *threshold* állítására. Érdemes lehet még egy dinamikus megvalósítást létrehozni, amivel a szabályozók a megmért szinthez állítják a *thresholdot* és *ratiot*. Az általános dinamikaszabályozók egyéb funkcióit is még érdemes lehet implementálni, többek között a knee-t és a sidechain funkciót. Ezek a szoftver alapszerkezetén már nem változtatnak, csak teljesebbé teszik.

Irodalomjegyzék

[1] Univeral Audio, Lynn Fuston: UA'S Classic 1176 Compressor - A History, https://www.uaudio.com/blog/analog-obsession-1176-history/ (hozzáférés dátuma: 2021. november 22.)

[2] Rane Corporation: Dynamics Processors – Technology & Application Tips, https://www.ranecommercial.com/legacy/pdf/ranenotes/Dynamics_Processors.pdf (hozzáférés dátuma: 2021. november 22.)

[3] Wikipedia: Dynamic range compression, https://en.wikipedia.org/wiki/Dynamic_range_compression (hozzáférés dátuma: 2021. november 22.)

[4] Rane Corporation: Linkwitz-Riley Crossovers: A Primer, https://mail.ampslab.com/PDF/linkwtiz_riley.pdf (hozzáférés dátuma: 2021. november 22.)

[5] Steinberg: VST: Seamless integration for virtual instruments and effects, https://www.steinberg.net/technology/ (hozzáférés dátuma: 2021. november 25.)

[6] Vinnie Falco: A Collection of Useful C++ Classes for Digital Signal Processing, https://github.com/vinniefalco/DSPFilters (hozzáférés dátuma: 2021. november 27.)

[7] Matteo Frigo, Steven G. Johnson: FFTW Documentation, https://www.fftw.org/fftw3.pdf (hozzáférés dátuma: 2021. december 3.)

[8] Siemens: Window Types: Hanning, Flattop, Uniform, Tukey, and Exponential, https://community.sw.siemens.com/s/article/window-types-hanning-flattop-uniform-tukey-and-exponential (hozzáférés dátuma: 2021. december 3.)

[9] Siemens: Window Correction Factors, https://community.sw.siemens.com/s/article/window-correction-factors (hozzáférés dátuma: 2021. december 3.)

[10] JUCE: Documentation, https://docs.juce.com/master/index.html (hozzáférés dátuma: 2021. december 4.)

[11] DDMF: Plugin Doctor, https://ddmf.eu/plugindoctor/ (hozzáférés dátuma: 2021. december 5.)

[12] Oeksound: Soothe2 user manual, https://storage.googleapis.com/oeksound-downloads/soothe2/soothe2_ManualFAQ.pdf (hozzáférés dátuma: 2021. december 6.)