Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Simonek Péter

REKURZÍV SPEKTRUMBECSLŐ ELJÁRÁSOK MEGVALÓSÍTHATÓSÁGA BEÁGYAZOTT RENDSZEREKBEN

KONZULENS

Dr. Orosz György

BUDAPEST, 2017

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Simonek Péter**, 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, 2017. 12. 08.

...……

Simonek Péter

Kivonat

Beágyazott jelfeldolgozási feladatok során számos esetben van szükség egy periodikus jel spektrumának ismeretére, különösen annak abszolút értékére. A spektrális együtthatók számítása hagyományosan Diszkrét Fourier Transzformációval (DFT) történik. Ez azonban időnként hátrányos a blokkos feldolgozás miatt. Célszerű tehát megvizsgálni, milyen lehetőségek vannak rekurzív spektrumszámítási eljárások használatára, amelyek minden, a jelből vett minta után, a szükséges számításokkal frissítik a spektrális együtthatók értékét.

Szakdolgozatomban annak járok utána, megvalósítható-e egy ilyen algoritmus egy nem nagy számítási sebességű beágyazott rendszeren.

Munkám során megismertem a rekurzív spektrumszámító eljárásokhoz használt állapotteres jelmodellt, majd két különböző, ennek alapján a Fourier-együtthatókat megadni képes algoritmust. Matlab környezetben megvizsgáltam ezek viselkedését abból a nézőpontból, mennyire lenne praktikus ezeket implementálni egy beágyazott eszközön. Szakirodalmi hivatkozások alapján mindkét algoritmus esetén lehetőség van a számítási igényt csökkentő egyszerűsítésekre. Megvizsgálva ezen javaslatokat arra jutottam, hogy a két algoritmus egyszerűsített változata nevezéktantól eltekintve formailag megegyezik.

Az így eredményül kapott eljárást implementáltam egy beágyazott fejlesztőkártyán egy C nyelvű szoftver formájában. A programozás során további egyszerűsítő, hatékonyságnövelő módosításokat tettem.

Végül mérésekkel megbizonyosodtam róla, hogy az elkészült rendszer működik, emellett pedig néhány jellemzőjét is megvizsgáltam. Méréseim során azt is figyeltem, milyen hatása van a fixpontos aritmetikájú processzoron annak, ha lebegőpontos változókat is használunk a programozás során, és milyen annak, ha ezek nélkül oldjuk meg a feladatot.

Abstract

The spectrum of a periodic signal, especially its absolute value is needed in many cases in embedded signal processing tasks. The calculation of the spectral coefficients is traditionally made with Discrete Fourier Transformation (DFT), however, it can be disadvantageous sometimes due to the block processing operation. So it is expedient to examine, what kind of possibilities exist about using recursive spectrum calculation methods. These algorithms refresh the value of spectral coefficients with the necessary calculations after every sample of the signal.

In my thesis, I investigate the feasibility of implementation of such recursive algorithms in an embedded system with moderate computational capacity.

During my work I collected information about the state-space signal representation used in recursive spectrum calculation methods. Then I examined two different algorithms, that could produce the Fourier-coefficients according to this signal model. I made examinations in Matlab environment from the viewpoint of practical implementation of algorithms in an embedded system. I considered the effect of some recommended simplifications, and I came to the result, that the two methods became the same one.

I implemented the simplified algorithm on an embedded development board in the form of a C language software. During the programming I made further simplifier, efficiency increasing modifications.

Finally I investigated the performance of the system by measurements. Beside that, I also examined some of its properties. During my measurements I paid attention to what effect it had on the fixed point arithmetic processor, if floating point variables had been also used at programming, and what effect it had, if the problem had been solved without them.

1 Bevezetés

A digitális jelfeldolgozás egy folyamatosan fejlődő tudományterület, amely alkalmazása egyre több területre terjed ki. Eredményeinek felhasználói közé tartozik például a méréstechnika, az akusztika, a távközlés, stb. A fejlődéshez alapvetően hozzájárult, hogy egyre nagyobb számítási teljesítményű jelfeldolgozást támogató jelprocesszorok és mikrovezérlők jelentek meg egyre elérhetőbb áron, ezzel lehetővé téve kis, olcsó beágyazott rendszereken való felhasználását.

A jelfeldolgozási problémák során nagyon gyakran van szükség a jelek frekvenciatartománybeli vizsgálatára vagy manipulálására. Ez néha csak egy szűrési feladatot jelent, ahol az időtartománybeli minták és egy előre megtervezett szűrőegyüttható-készlet segítségével a megoldás realizálható a pontos spektrum ismerete nélkül is, azonban ez máshol szükségessé válhat. A spektrum kellően nagy felbontású meghatározása számításigényes művelet lehet, ezért a hardver- és szoftverterv elkészítésénél érdemes ezzel kapcsolatos megfontolásokat tenni.

A számításokat végző vezérlőnek érdemes ilyenkor egy jelfeldolgozási műveletekre optimalizált jelprocesszort (Digital Signal Processor – DSP) választani. Ez kényelmessé teszi a szoftver megírását, hiszen számos támogatást ad célunk megvalósításához: lebegőpontos számábrázolás, MAC (Multiply And Accumulate) utasítás, bitreverse címzés; emellett valószínűleg a teljesítménnyel sem lesz probléma. A DSP-k hátránya azonban, hogy ára főleg más opciókkal összevetve magas lehet, ezért ha használata nem feltétlenül szükséges, érdemes lehet elkerülni és egy egyszerűbb mikrokontrollert használni. Manapság a mikrokontrollerek számos olcsóbb típusa rendelkezik jelfeldolgozást támogató hardveres egységekkel, a fentebb említett képességek is részben vagy egészben igazak rájuk. Teljesítményük is sok esetben elég lehet. Emellett gyakran szoftveres támogatás is van hozzájuk, például elérhetőek előre megírt függvénykönyvtárak optimalizált, hatékony jelfeldolgozási algoritmusokkal. Programozásuk is már régóta megvalósítható magas szintű, elterjedt nyelveken. Mindezeket egybevéve érdemes megfontolni használatukat.

További kérdés lehet a spektrumszámítási algoritmus megválasztása. Mintavételezett rendszerekben a legelterjedtebb módszer a jól ismert Diszkrét Fourier Transzformáció (DFT). Ennek előnye, hogy a teljes spektrumot kiszámítja, így egy

általános megoldást kínál a problémára. Vannak azonban hátrányai is: egyrészt egy *M* pontos DFT-hez *M* mintára van szükségünk, amely egyrészt jelentős memóriát foglalhat el, másrészt szükségessé teszi, hogy bevárjuk az összes minta megérkezését, így az adatfeldolgozás csak blokkosan valósítható meg. Számításigénye is nagy lehet. Mindezek miatt érdemes megvizsgálni más algoritmusokat is.

Periodikus jeleknél a spektrumban csupán az egyenkomponenst és a harmonikusokat kell meghatároznunk, hiszen a többi helyen az amplitúdó nulla. Ez feleslegessé teszi, hogy a DFT-vel a teljes spektrumot kiszámítsuk, elég, ha csak ezeken a frekvenciákon értékeljük ki. Erre kínálnak alternatívát a rekurzív spektrumbecslő eljárások. Ezek az alapharmonikus frekvencia ismeretében képesek a periodikus jel Fourier-sorának együtthatóit kiszámítani. Minden beérkező minta után újraszámítják az együtthatókat, így kiküszöbölik az esetenként elfogadhatatlan blokkos feldolgozást is. Előnyös tulajdonságai miatt tehát érdemes megvizsgálni megvalósíthatóságának kérdését beágyazott rendszerekben.

Dolgozatomban ismertetek két különböző, ám bizonyos egyszerűsítések után ugyanazt eredményező rekurzív spektrumszámító algoritmust. Ezekhez előbb a 2. fejezetben ismertetem a két eljárás által használt jelmodellt, majd külön-külön megvizsgálom őket a 3. fejezetben Matlab[1] segítségével, figyelembe véve néhány megvalósíthatósági szempontjukat beágyazott rendszereken, mint például számításigényesség, zajérzékenység. Az eredményeket látva néhány, a szakirodalomban javasolt, a funkcionalitást megőrző egyszerűsítéssel élve olyan alakra hozom őket, amely lényegében ugyanaz, így egyesítem a két modellt, és így implementációs szempontból is kedvezőbb eredményre jutok.

A 4. fejezetben bemutatok egy, a tapasztalatok felhasználásával készített valós implementációt, mely egy manapság könnyen elérhető fejlesztőkártyára írt C nyelvű szoftver. Ennek során bemutatom a felhasznált hardvert és a fejlesztői környezetet, valamint egy már kész keretrendszert, amelybe beillesztem az általam leprogramozott egyszerűsített algoritmust. Ismertetem az ezt megvalósító függvényeket a forráskód lényegi részeinek elemzésével, törekedve a gyakorlati szempontból jelentősebb hatással bíró megoldások kiemelésére. Ezt követően néhány mérési eredménnyel igazolom a megvalósítás működőképességét, valamint jellemzem annak gyakorlati tulajdonságait.

Végezetül összegzem az elvégzett munkát és kitekintek a továbbfejlesztési lehetőségekre.

2 Jelmodell

2.1 Periodikus jelek felírása Fourier-együtthatókkal

Az általam vizsgált rekurzív spektrumbecslő eljárások periodikus jelek vizsgálatára alkalmasak, ezért elsőként ezek leírásának módját ismertetem[6].

A periodikus jelek felírhatóak a periódusidejükből számított alapharmonikus frekvenciájú és annak egész számú többszöröseit jelentő felharmonikus frekvenciájú súlyozott szinuszos komponensek összegeként, ez a jel Fourier-sora. Ennek nyomán az y_n diszkrét időtartománybeli jel Fourier-sora komplex írásmódban:

$$y_n = \sum_{k=-P}^{P} X_k c_{k,n} \tag{1}$$

ahol:

$$X_k = A_k e^{j\varphi_k} \tag{2}$$

$$c_{k,n} = e^{j2\pi f_1 kn} \tag{3}$$

$c_{k,n}$ tehát a komplex körforgó egységvektor, X_k pedig a Fourier-együttható, ami két fontos tulajdonságát hordozza az adott frekvenciájú komponensnek: az amplitúdóját (A_k) és a fázisát (φ_k). Az f_1 a folytonos jel alapharmonikus frekvenciájának a mintavételi frekvenciához (f_s) viszonyított aránya. A mintavételi tétel korlátozza a sávszélességet, így a jel véges komponensből előállítható (P korlátos). Mindezekből látható, hogy a jel előállítható f_1, P, A_k és φ_k (ahol k = -P ... P) ismeretében. Vizsgálataim során feltételeztem, hogy f_1 vagy a kiszámításához (4) alapján szükséges f_{alap} alapharmonikus frekvencia ismert.

$$f_1 = \frac{f_{alap}}{f_s} \tag{4}$$

A felharmonikus frekvenciák az $f_k = kf_1$ helyeken vannak.

A dolgozatban végig valós peridikus jelekkel foglalkozom, melyeknél az azonos $|f_k|$ abszolút értékű, de előjelükben különböző frekvenciákhoz tartozó Fourier-együtthatók komplex konjugált párt alkotnak. A nem nulla amplitúdójú komponensek számának (N) maximális értéke az alábbi módon számítható:

$$N_{max} = \begin{cases} 2P+1, & \text{általában} \\ 2P+2, & \text{ha } (P+1)f_1 = 0{,}5 \end{cases} \tag{5}$$

ahol P a $\pm f_k$ frekvenciákhoz tartozó komplex konjugált párok maximális száma, és egyben a valós értékű szinuszos komponenseké is. Az (5) összefüggésben a második eset speciális, ekkor a mintavételi frekvencia fele az alapharmonikus egész számú többszöröse, ekkor még itt is van egy komponens. Ekkor az (1) összefüggés is módosul:

$$x_n = \sum_{k=-P}^{P+1} X_k c_{k,n} \tag{6}$$

P maximális értékét a mintavételi tétel szabályozza, a következő módon adható meg[15][4]:

$$Pf_1 < 0{,}5 \leq (P+1)f_1 \tag{7}$$

Érdemes még megjegyezni, hogy a jelben meglévő valós szinuszos komponens amplitúdója úgy adódik, hogy vesszük a hozzá tartozó $\pm f_k$ frekvenciájú komplex komponensek amplitúdóinak összegét: A_k+A_{-k}. Ez az oka, hogy szükség van a negatív frekvenciájú komponensekre is a valós jel rekonstrukciójához.

2.2 A koncepcionális jelmodell

Az, hogy a jel véges számú szinuszos komponens szuperpozíciójaként áll elő, adja a jelmodell alapgondolatát, miszerint tekintsük a jelforrást egy állapotváltozós leírással jellemzett rendszernek, ahol az állapotváltozók reprezentálják az egyes komponenseket[7]. Ez két megközelítéssel is megvalósítható: idővariáns és időinvariáns modellel, azonban mindkettő felírható az alábbi egyenletekkel:

$$\boldsymbol{x}_{n+1} = \boldsymbol{A}\boldsymbol{x}_n \tag{8}$$

$$y_n = \boldsymbol{C}\boldsymbol{x}_n \tag{9}$$

ahol $\boldsymbol{A}$ egy NxN-es kvadratikus mátrix, $\boldsymbol{x}$ az N db állapotváltozót tartalmazó oszlopvektor, $\boldsymbol{C}$ pedig egy N elemű sorvektor.

A továbbiakban bevezetek egy új indexelést f_k-ra. Legyen:

$$f_k = \begin{cases} \left(k - \dfrac{N+1}{2}\right) f_1, & k = 1 \ldots N, \quad \text{ha nincs } \dfrac{f_s}{2} \text{ tag} \\ \left(k - \dfrac{N}{2}\right) f_1, & k = 1 \ldots N, \quad \text{ha van } \dfrac{f_s}{2} \text{ tag} \end{cases} \tag{10}$$

Így ezután:

$$c_{k,n} = e^{j2\pi f_k n} \tag{11}$$

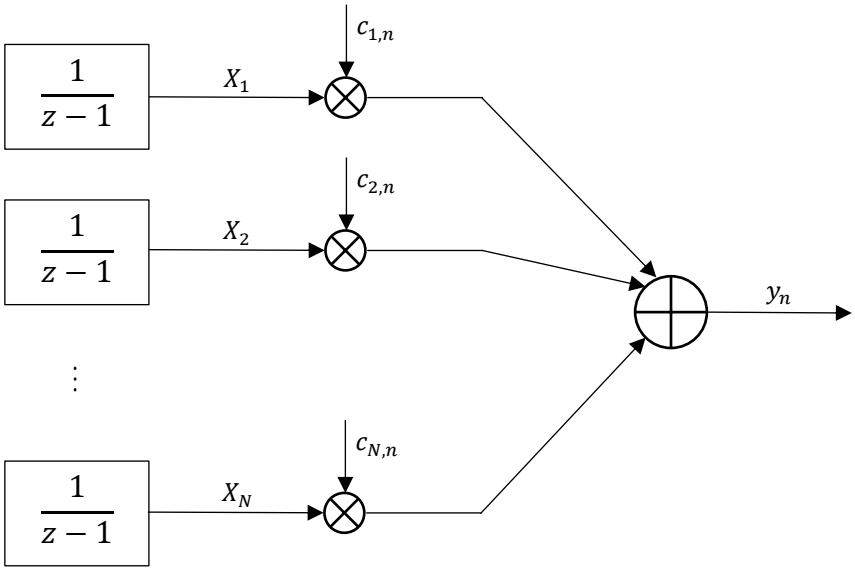
Fontos azonban kiemelni az egyértelműség kedvéért, hogy $f_1 \neq f_{k=1}$. Az f_1-el jelölt változó ugyanis ahogy eddig, a továbbiakban is mindig a (4) szerint értelmezett normalizált alapharmonikus frekvencia, míg $f_{k=1}$ a legnegatívabb vizsgált normalizált frekvencia. A továbbiakban külön rá nem fogok hivatkozni, mivel számunkra fontos speciális funkciója nincsen, csak általánosan f_k-ra.

Az idővariáns modellben az állapotváltozók a Fourier-együtthatók. A jelforrás a 2.1. ábra szerint modellezhető. A kezdetben nulla értékű integrátorok bemenetére a kezdeti n_0 ütemben X_k értéket, majd utána ismét folyamatosan nullát adunk. Az integrátor így végig szolgáltatja az X_k értéket, amit beszorzunk még $c_{k,n}$ körforgó vektor pillanatnyi értékével. Ezeket összegezve pedig megkapjuk az y_n jelet. Ekkor az integrátorok kimenetei képezik az $\boldsymbol{x}$ vektort, amely így változatlan marad. A modellt leíró mátrixok értéke pedig:

$$\boldsymbol{A} = diag(1\ 1 \ldots 1) \tag{12}$$

$$\boldsymbol{C}_n = [c_{1,n}\ c_{2,n} \ldots\ c_{N,n}] \tag{13}$$

Jól látható, hogy a komplex körforgó vektorral súlyozó $\boldsymbol{C}_n$ kicsatolóvektor értéke függ attól, melyik ütemben vagyunk, ezért tehát a modell idővariáns, mivel az állapotegyenleteket minden ütemben frissíteni kell. Fontos még megjegyezni, hogy az idővariancia nem vonatkozik a frekvenciákra, azok állandóak[15][2][4].



2.1. ábra: Jelforrás koncepcionális modellje

Az időinvariáns modellben ezzel ellentétben az állapotváltozók az egyes harmonikus komponensek adott ütembeli értékei, vagyis

$$x_{k,n} = X_k c_{k,n} \tag{14}$$

Ehhez az $\boldsymbol{A}$ mátrixnak már tartalmaznia kell a bázisfüggvénnyel ($c_{k,n}$) való súlyozást, $\boldsymbol{C}$-nek viszont csak egy összegzést kell végrehajtani. Így

$$\boldsymbol{A} = diag(z_1\ z_2 \dots\ z_N) \tag{15}$$

$$\boldsymbol{C} = [1\ 1 \dots 1] \tag{16}$$

$$z_k = \frac{c_{k,n+1}}{c_{k,n}} = e^{j2\pi f_k}, \quad k = 1 \dots N \tag{17}$$

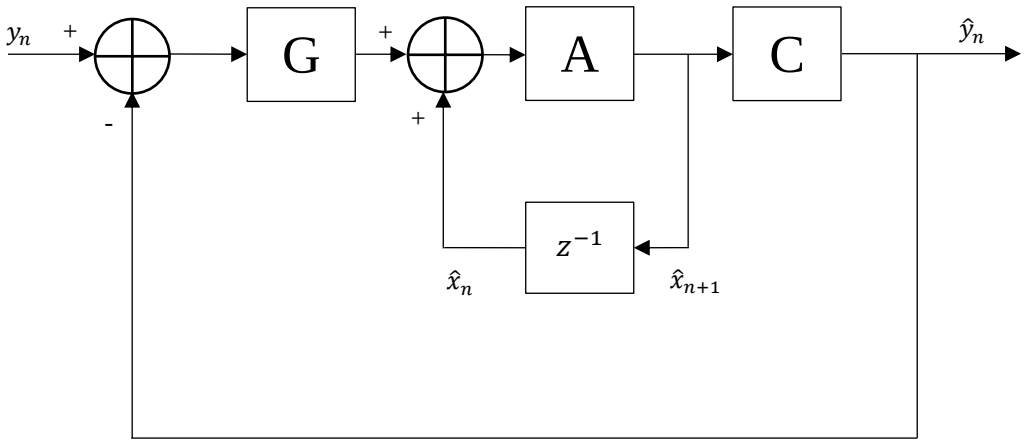
Jól látható, hogy ebben a modellben az állapotvektor ütemről ütemre változik, míg az $\boldsymbol{A}$ és $\boldsymbol{C}$ mátrixok konstansok, ezért a modell időinvariáns[15][4].

Mindkét jelmodell teljesen leírja a jelforrást, mindkettő használható, alkalmazástól függ, melyik célszerűbb. Esetemben végig az idővariáns megközelítést alkalmazom majd. Ennek oka, hogy beágyazott rendszerekben az időinvariáns modellben az állapotegyenletekben $\boldsymbol{A}$-nak például a számábrázolásból eredő pontatlansága olyan állandó rendszeres hibát okozhat, amely önmagát erősíti.

2.3 Megfigyelő tervezése

A cél tehát a Fourier-együtthatók meghatározása, amihez a megfigyelő-elméletet veszem segítségül. Az állapot-megfigyelők a megfigyelt rendszer struktúráját ismerve képesek meghatározni annak állapotváltozóit[15][2]. A megfigyelő blokkvázlatát mutatja a 2.2. ábra. Az $\boldsymbol{A}$ és $\boldsymbol{C}$ mátrixokat ismerjük, hiszen azok egyeznek a jelmodellbeli azonos nevűekkel. Ezekhez ismernünk kell f_1-et és N-et. f_1-et ismertnek feltételeztük, N-et pedig annak ismeretében választhatjuk, hogy hányadik felharmonikusig szeretnénk vizsgálni a spektrumot, természetesen betartva, hogy $N \leq N_{max}$. Megjegyzem, hogy azt a speciális esetet kivéve, amikor $\frac{f_s}{2}$-n is van komponens és a teljes spektrumot akarjuk vizsgálni, N-et mindig érdemes páratlannak választani, hiszen így a harmonikus komponensek pozitív és negatív megfelelői is jelen vannak, biztosítva ezzel a szimmetriát és a helyes számítást. N-et éppen ezért mindig eszerint választom meg a továbbiakban.

A megfigyelő blokkvázlatából az is látható, hogy a kérdéskör felfogható egy szabályozástechnikai problémaként is. Az alapjel y_n, amit szeretnénk minél gyorsabban és minél pontosabban követni a szabályozott jellel, amely a becsült $\hat{y}_n$. A szakasz egyezik a jelforrás állapotváltozós leírásával, és $\boldsymbol{G}$ az alkalmazott szabályzó.



2.2. ábra: Megfigyelő blokkvázlata

A megfigyelőt az alábbi egyenletekkel írhatjuk le:

$$\hat{\boldsymbol{x}}_{n+1} = \boldsymbol{A}\hat{\boldsymbol{x}}_n + \boldsymbol{G}_n(y_n - \hat{y}_n) \tag{18}$$

$$\hat{\boldsymbol{x}}_0 = \boldsymbol{0} \tag{19}$$

$$\hat{y}_n = \boldsymbol{C}_n\hat{\boldsymbol{x}}_n \tag{20}$$

Ezeket kiegészítve a (15) és(16) egyenletekkel levezethető az alábbi összefüggés a becsült és a valós állapotváltozók vektorára:

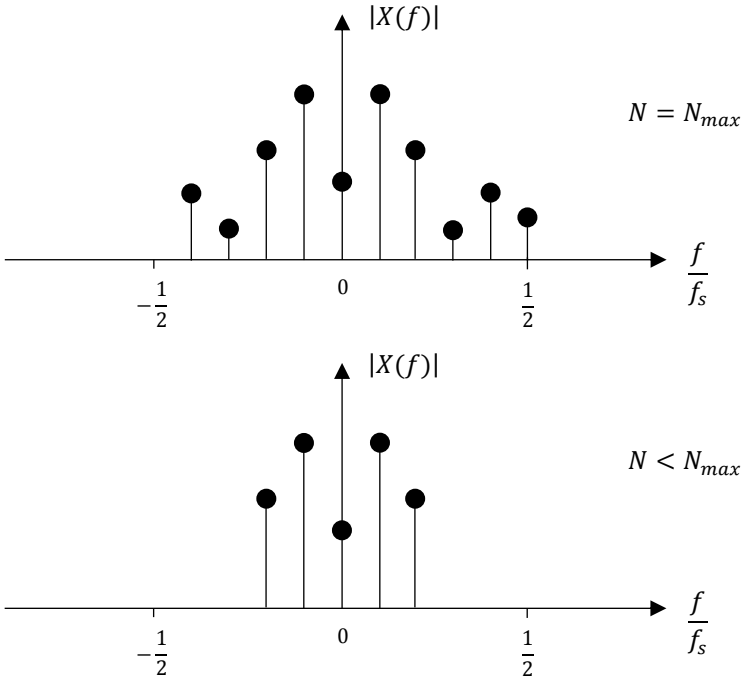
$$\hat{\boldsymbol{x}}_{n+1} - \boldsymbol{x}_{n+1} = (\boldsymbol{A} - \boldsymbol{G}_n\boldsymbol{C}_n)(\hat{\boldsymbol{x}}_n - \boldsymbol{x}_n) \tag{21}$$

Egyedül a $\boldsymbol{G}$ becsatolómátrixot kell megterveznünk, és ezt úgy kell megválasztani, hogy a megfigyelő és a valós jelforrás állapotváltozóinak különbsége minél gyorsabban nullához tartson[2].

3 Két rekurzív spektrumbecslő eljárás

Ebben a fejezetben két különböző elméleti hátterű rekurzív spektrumbecslő algoritmust mutatok be, majd azok egyesítését bizonyos egyszerűsítések megtétele után. Mindkettőnél úgy járok el, hogy először ismertetem az általános leírást, majd ezeknél ismertetek néhány problémakört, amelyek vizsgálataim során előjöttek és fontosnak bizonyultak. Végül a szakirodalom alapján javasolt egyszerűsítéseket követően olyan alakra hozom az általános összefüggéseket, hogy azok megfeleltethetőek egymásnak, és ezt az egyezőséget szimulációkkal is igazolom.

A vizsgálat során több paraméter változásának hatását fogom vizsgálni, melyek befolyásolják a működést vagy a megvalósíthatóságot. Az egyik ilyen paraméter mindkét algoritmusban szerepel, ez a vizsgált komponensek száma, az *N*. Ez lényegében azt fejezi ki mekkora sávszélességre vagyunk kíváncsiak a jelből. Tekintve, hogy ez a konkrét algoritmusok ismerete nélkül is fontos szempontnak látszik, *N* változtatásának hatását az eredményül kapott spektrumra itt külön kiemelve egy ábrán illusztrálom.



3.1. ábra: N változtatásának hatása a kapott spektrumra

Látható, hogy $N = N_{max}$ esetén a teljes sávszélességet lefedjük, míg $N < N_{max}$ esetén bizonyos felharmonikusokat már nem számítunk ki.

3.1 Rezonátoros algoritmus

Az felhasznált algoritmus [3]-ból származik. A módszer lényege, hogy minden f_k komponenshez tartozik egy rezonátorcsatorna a megfigyelőben, amelyben egy integrátor kimeneti értékeként előáll és tárolódik az adott komponens Fourier-együtthatója. Ezen érték segítségével és az integrátor kimenetén lévő $c_{k,n}$ szorzóval (mely a $\boldsymbol{C}_n$ vektor k. eleme) állítjuk elő az adott frekvenciájú komponenst a kimenetre a becsült jelhez, mely így az egyes rezonátorcsatornák kimenetének összegeként áll elő. Az $\hat{y}_n$ jelet a bemenetre negatívan visszacsatolva kapjuk a hibajelet, amely (23) szerint értelmezhető. A hibajelet minden rezonátorcsatornára továbbítjuk, ahol a $\boldsymbol{G}_n$ vektor megfelelő eleme, $g_{k,n}$ biztosítja, hogy az integrátor bemenetére olyan érték kerüljön, amely az f_k frekvenciájú komponens becsült és valós Fourier-együtthatója közti eltérést a régi értékkel összegezve csökkenti. A rezonátoros megfigyelő blokkdiagramját mutatja a 3.2. ábra.

A rendszer egyenletei a következőek:

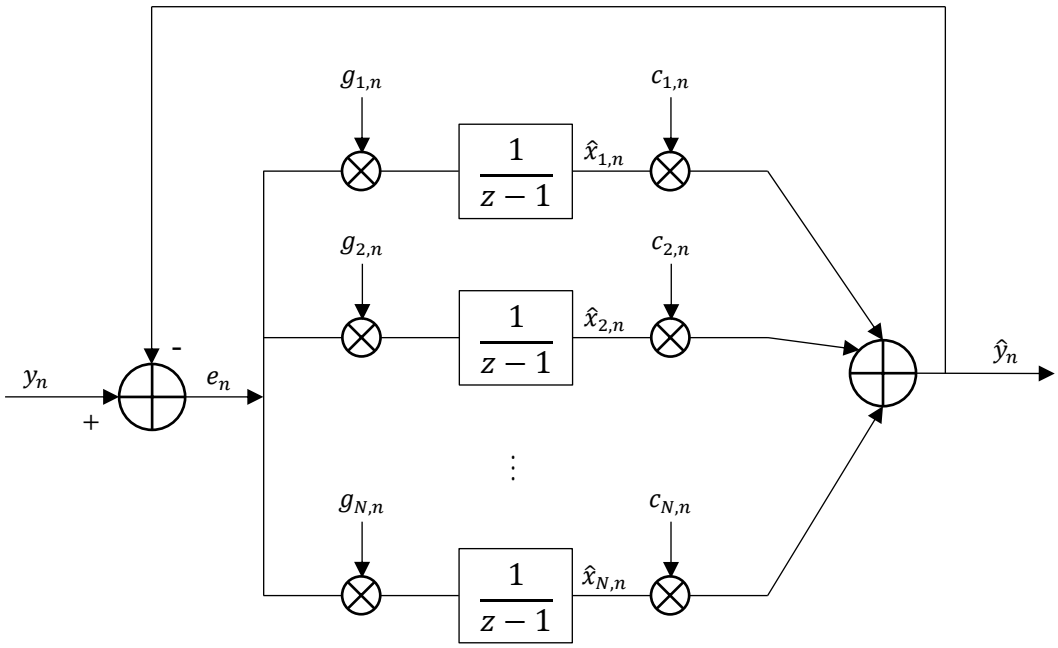
$$\hat{y}_n = \boldsymbol{C}_n \hat{\boldsymbol{x}}_n \tag{22}$$

$$e_n = y_n - \hat{y}_n \tag{23}$$

$$\hat{\boldsymbol{x}}_{n+1} = \hat{\boldsymbol{x}}_n + e_n \boldsymbol{G}_n \tag{24}$$

Szemléletesen az alábbi módon összegezhetjük a működést:

- Először becslést adunk a jel értékére az addigiak alapján becsült Fourier-együtthatókból.
- A jel valós és becsült értékéből kiszámítjuk a becslési hibát.
- A becslési hiba alapján módosítjuk az $\hat{\boldsymbol{x}}_n$ becsült együtthatókat.



3.2. ábra: Rezonátoros megfigyelő blokkdiagramja

3.1.1 A megfigyelő becsatolómátrixának megtervezése

A $g_{k,n}$ együtthatók számítása az alábbi összefüggések szerint történik[3][4], ahol z_k megegyezik a (17) egyenletben írtakkal.

$$g_{k,n} = conj(c_{k,n}) \cdot \frac{\Psi_k}{D_k} \tag{25}$$

$$\Psi_k = \prod_{i=1}^{N} (1 - p_i z_k^{-1}) \tag{26}$$

$$D_k = \prod_{i=1, i \neq k}^{N} (1 - z_i z_k^{-1}) \tag{27}$$

$$p_k = \sqrt[N]{(1-\alpha)} \cdot e^{j2\pi f_k} \tag{28}$$

Néhány kifejezés jelmagyarázata a könnyebb érthetőségért:

- *n*: időváltozó, a jelfeldolgozási ütem sorszáma
- *k*: a rezonátorcsatorna sorszáma, a hozzá tartozó frekvencia értéke f_k (ld. (10) egyenlet), értéke 1-től *N*-ig terjed
- α: bátorsági tényező, elemzését ld. később

- *N*: rezonátorcsatornák száma, ami egyezik a vizsgált harmonikus komponensekével

Az egyenletekben p_k a megfigyelő pólusait jelöli[15][4].

Jól látható, hogy a kifejezésekben majdnem minden adott, egyedül *N* és α értékét választhatjuk meg. *N*-ről már volt szó, most vizsgáljuk meg α-t! Értéke a rendszer beállásának sebességét határozza meg. Kisebb α lassabb beállást eredményez, ugyanakkor a zajérzékenységet is csökkenti. A tervezésnél a két szempontot együttesen mérlegelve kell megválasztani. Értékét az alábbi intervallumból lehet választani:

$$0 < \alpha \leq 1 \tag{29}$$

Érdemes megvizsgálni a két szélső helyzetet. $\alpha = 0$ esetén az egyenkomponenshez tartozó szorzótényező a Ψ_k szorzatban nulla értéket vesz fel, hiszen ekkor $p_{DC} = \sqrt[N]{(1-0)} \cdot e^{j2\pi \cdot 0} = 1$ és $z_{DC}^{-1} = e^{-j2\pi \cdot 0} = 1$, ezért az egész szorzat eredménye zéró. Ez azt eredményezi, hogy minden *k*-ra $g_{k,n} = 0$, ami nyilvánvalóan lehetetlenné teszi a megfigyelő működését, ezért ez az érték ki is van zárva a választható tartományból. $\alpha = 1$ esetén könnyen belátható, hogy $p_k = 0$, és emiatt $\Psi_k = 1$. Ez a beállítás azt eredményezi, hogy a megfigyelő *N* ütem alatt beáll, úgynevezett véges beállású megfigyelőt kapunk. Ez a beállási sebesség szempontjából előnyös tulajdonság, azonban, mint később látni fogjuk, problémás lehet.

3.1.2 A becsatoló-együtthatók

3.1.2.1 Az együtthatók számításának folyamata

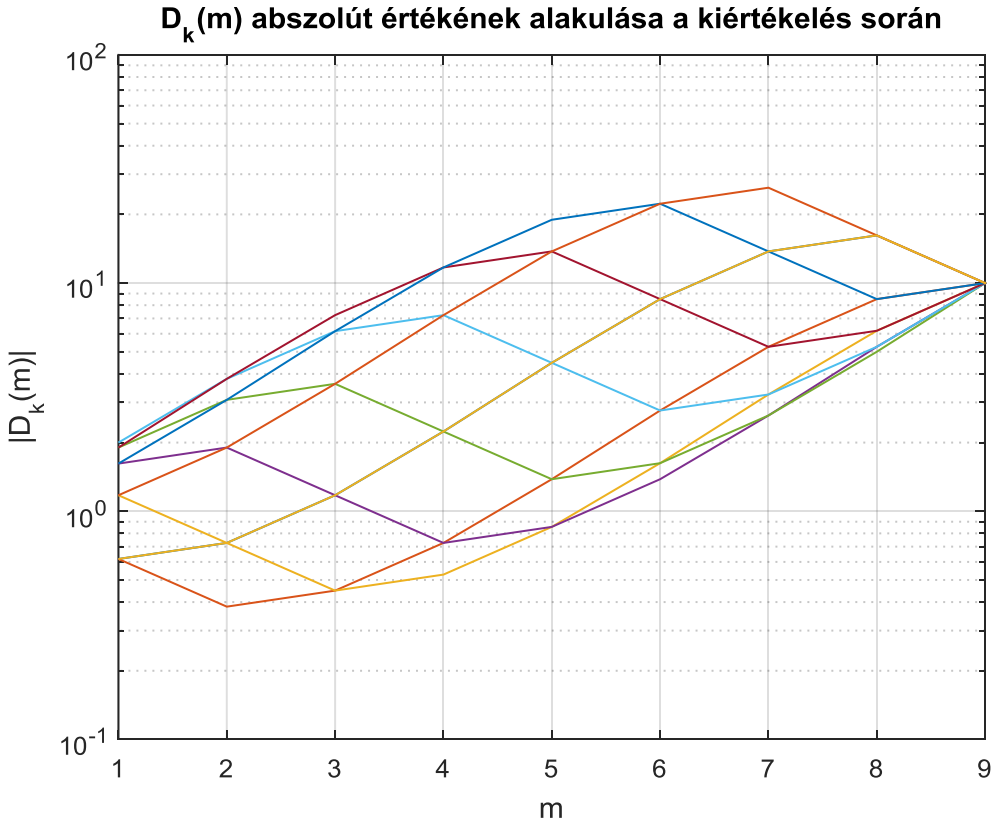
A 3.1.1 fejezetben bemutatott összefüggések láthatóan összetettek és bonyolultak. Számításigényük nagy *N*-ek esetén már jelentős lehet, és a kifejezésekből az sem látszik első ránézésre, hogy az általunk választott paraméterek módosítása milyen hatással van a végeredményre. Ezek miatt megvizsgáltam több szempontból ezen egyenletek kiértékelésének folyamatát.

Elsőként csak a (27) egyenletet vizsgáltam, ami a $g_{k,n}$ együttható nevezője. D_k értéke *N*-1 db szorzótényező szorzataként adódik. A szorzótényezők egy különbség eredményeiként adódnak, amiben $z_i z_k^{-1}$ szorzat egy egységkörön lévő komplex szám, tehát önmagukban nem nagy abszolút értékű számok.

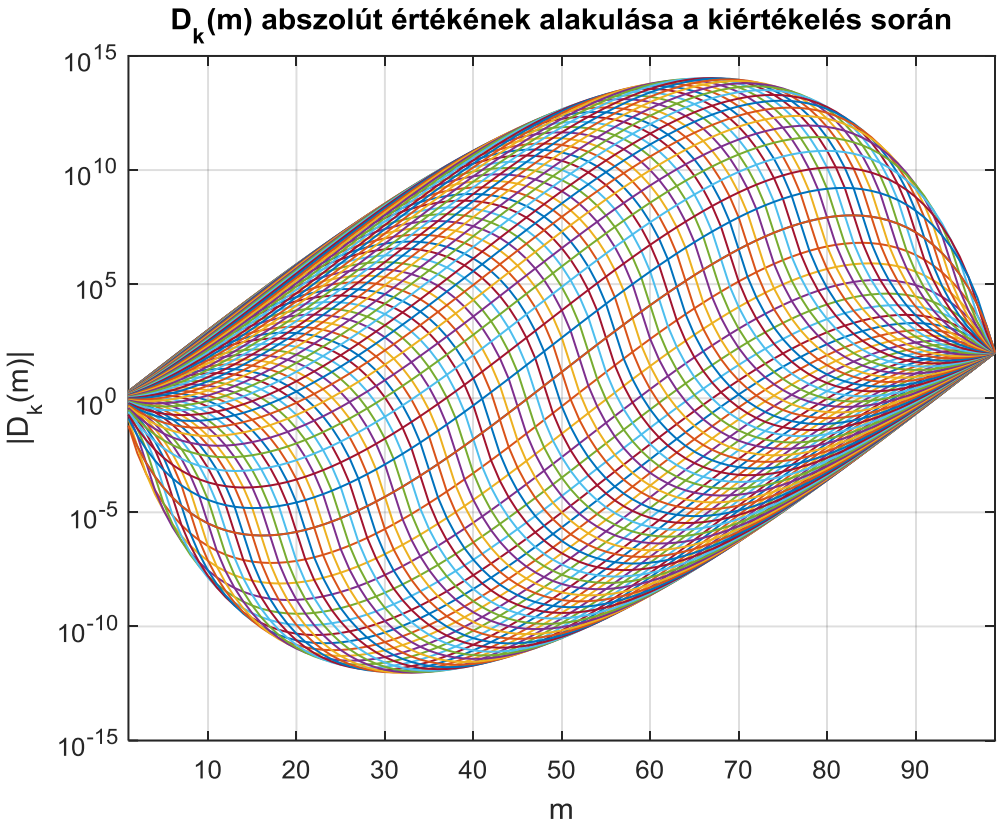
A szorzat vizsgálatához választottam egy f_1 értéket, majd az ehhez tartozó N_{max} számú komponenst feltételeztem. Ezután elkezdtem kiértékelni a szorzatot úgy, hogy minden szorzás után eltároltam az aktuális *m*. részeredmény abszolút értékét. Elsőként a stratégiám nagyon egyszerű volt: a (27) egyenletbeli produktumban a tényezőket $i = 1$-től *N*-ig sorban szoroztam össze, természetesen kihagyva az $i = k$ tényezőt (ld. (30)).

$$D_k = \underbrace{(1 - z_1 z_k^{-1})(1 - z_2 z_k^{-1}) \dots (1 - z_m z_k^{-1})}_{D_k(m)} \dots (1 - z_N z_k^{-1}) \tag{30}$$

Ezt minden *k*-ra elvégeztem, utána pedig egy közös grafikonon ábrázoltam az eredményeket (ld. 3.3. ábra és 3.4. ábra). Az egyes görbék azt mutatják, hogyan alakult az adott *k*-hoz tartozó részproduktum abszolút értéke az addig elvégzett szorzásműveletek (*m*) számának függvényében.



3.3. ábra: $|D_k(m)|$ értékének alakulása f_1 = 0,1 mellett



3.4. ábra: $|D_k(m)|$ értékének alakulása f_1 = 0,01 mellett

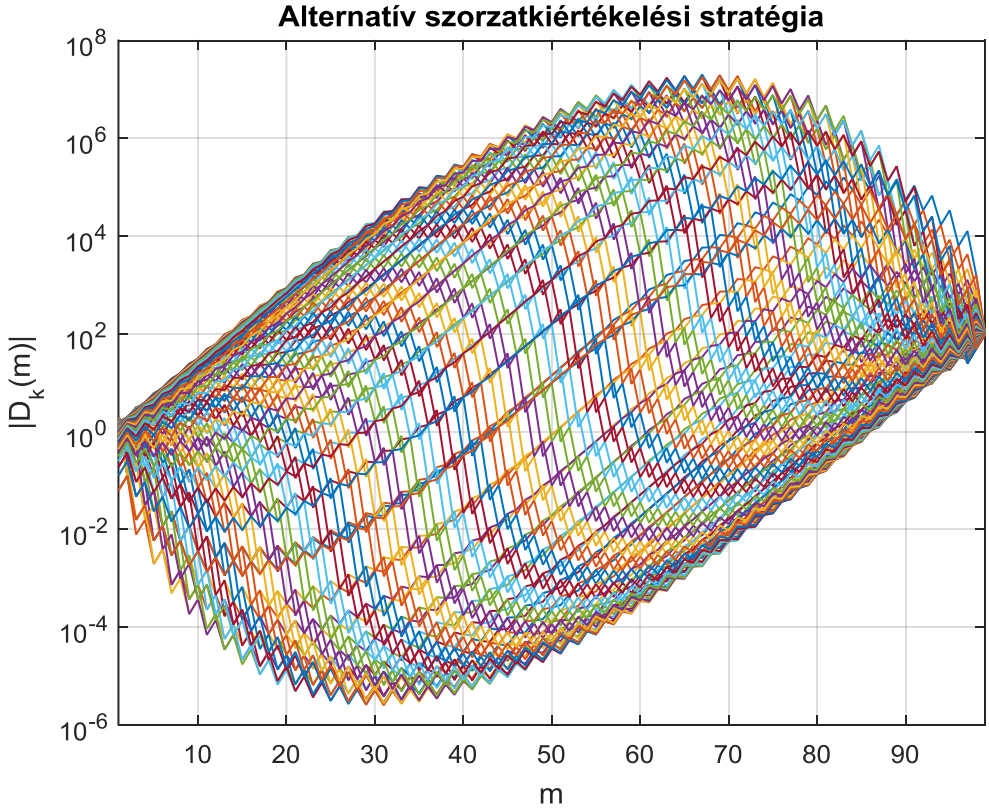
Látható, hogy $f_1 = 0{,}1$ esetén még nincs semmi különösebben figyelemreméltó jelenség, azonban $f_1 = 0{,}01$ esetén már a részeredmények maximuma és minimuma közt kb. 27 nagyságrend különbség van. Ez nagyon jelentős különbség, amely főleg fixpontos számábrázolás esetén nagy gondot okozhat egy beágyazott rendszeren, ráadásul az, hogy egy jelet frekvenciájának százszorosával mintavételezünk, nem is tartozik az extrém esetek közé. Még kisebb normalizált alapharmonikus frekvenciáknál pedig a tendencia folytatódik, $f_1 = 0{,}001$ esetén már a maximum a 10^{140}, a minimum a 10^{-138} nagyságrendbe esik, ami 279 nagyságrend átfogást jelent. Ezek már lebegőpontos számábrázolásnál is problémát jelentenek, az egyszeres pontosságú számábrázolás határain már kívül is esnek[16].

A teljes produktum, tehát D_k végeredménye viszont se nem nagyon nagy, se nem nagyon kicsi szám, ezért meg lehet próbálni a szorzatot másképpen is kiértékelni, hogy a részeredmények ne legyenek ennyire szélsőségesek. Egy másik stratégiával is elvégeztem a számításokat: ekkor a szorzótényezőket nem sorban, hanem a sor elejéről és közepétől indulva a szorzótényezőket felváltva vettem és haladtam végig, ahogyan azt (31) mutatja.

$$\left.\begin{aligned} D_k = (1 - z_1 z_k^{-1})\left(1 - z_{N/2+1} z_k^{-1}\right) \\ \cdot (1 - z_2 z_k^{-1})\left(1 - z_{N/2+2} z_k^{-1}\right) \\ \cdot \ldots \cdot \left(1 - z_{m/2} z_k^{-1}\right)\left(1 - z_{N/2+m/2} z_k^{-1}\right) \\ \cdot \ldots \cdot \left(1 - z_{N/2} z_k^{-1}\right)(1 - z_N z_k^{-1}) \end{aligned}\right\} D_k(m) \qquad (31)$$

Természetesen páratlan m-re is értelmezve van $D_k(m)$ ilyenkor a (31) egyenletben lévő kéttényezős blokkokból vagy még a következő első tényezőjét hozzávesszük, vagy az utolsó másodikját elhagyjuk.

Az eredményt mutatja a 3.5. ábra $f_1 = 0{,}01$ esetére. Ezen jól látszik, hogy a részeredmények által befogott tartomány jelentősen csökkent, azonban így is széles, tehát ez részben segít a problémán, de teljesen nem oldja meg.

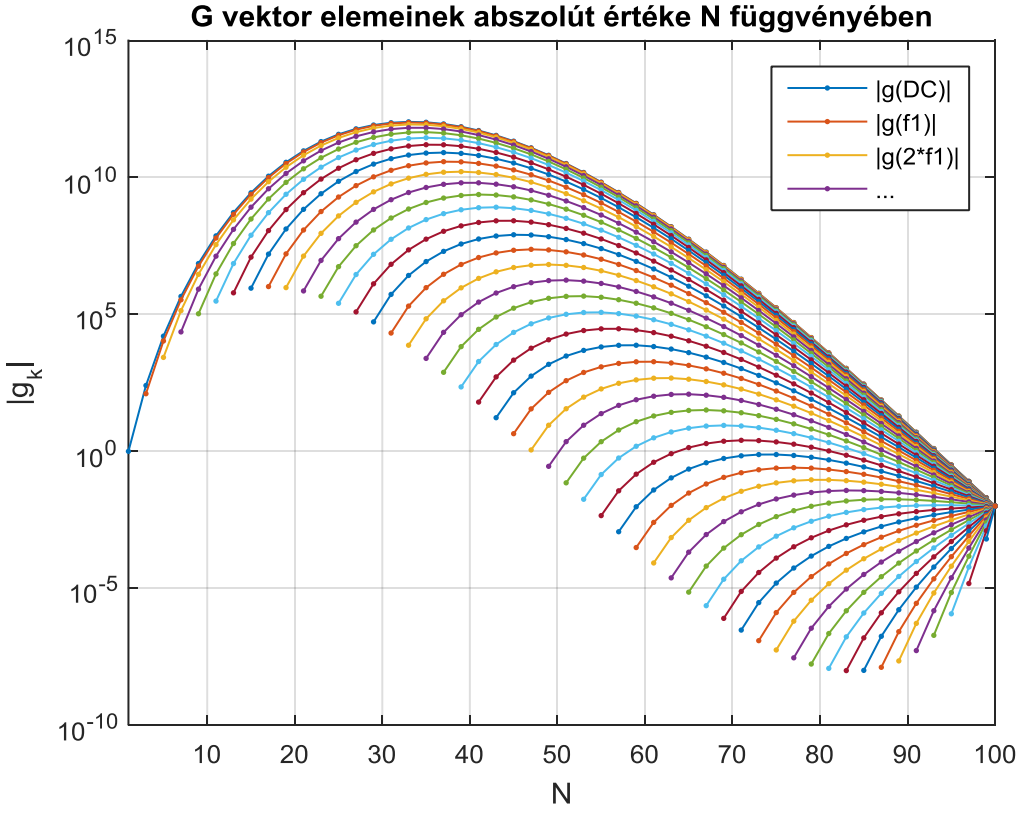


3.5. ábra: $|D_k(m)|$ értékének alakulása $f_1 = 0{,}01$ és alternatív kiértékelési stratégia mellett

3.1.2.2 Az együtthatók értékei és azok következményei

Következőnek az vizsgáltam, hogy $g_{k,n}$ abszolút értéke hogyan függ N-től. Ehhez először tekintsük a (25) egyenletet! $conj(c_{k,n})$ értéke nyilván nem befolyásolja, hiszen az egy egységkörön elhelyezkedő komplex szám, tehát a $\frac{\Psi_k}{D_k}$ tört határozza meg. A korábbi

diszkusszió alapján a tört számlálója $\alpha = 1$ esetén 1, így a kizárólag *N*-től való függés vizsgálatához ezt az egyszerű esetet vettem alapul. Ugyanúgy választottam egy f_1 értéket, majd 1-től N_{max}-ig minden páratlan számú *N*-re (ill. ha $\frac{f_s}{2}$ komponens is van, akkor a páros N_{max}-ra is) megnéztem az egyes komponensek abszolút értékét. A kísérlet eredményét mutatja $f_1 = 0{,}01$ mellett a 3.6. ábra, amely az egyenkomponens és a harmonikusok $g_{k,n}$ együtthatóinak abszolút értékét mutatja *N* függvényében. A görbék értelmezési tartománya nyilván eltérő, hiszen például $3f_1$ frekvenciájú rezonátorcsatorna nem létezik $N < 7$ esetén. Érdemes még azt is megjegyezni, hogy a grafikonon nem láthatunk N_{max} db görbét, mivel egy frekvencia pozitív és negatív megfelelőjéhez tartozó értékek megegyeznek. A grafikonról úgy olvashatók le a $g_{k,n}$ abszolút értékek egy megadott *N* esetén, hogy vesszük a görbék és az *N* egyenes metszéspontjait.



3.6. ábra: A becsatolóvektor elemeinek abszolút értéke *N* függvényében

A probléma itt is hasonló. A $g_{k,n}$ együtthatók abszolút értéke *N* függvényében felvehet nagyon magas és alacsony értékeket is, ráadásul ezen nem lehet a számítási mód optimalizálásával sem segíteni, hiszen ezek már végeredmények. A tendencia ekkor is ugyanaz: f_1 csökkentésével a közrefogott tartomány gyorsan nő. További probléma

forrása lehet, hogy a legtöbb rögzített *N* esetén a komponensek együtthatói szintén széles tartományon változnak, így a számábrázolási formátum optimalizálása sem vezetne eredményre egy beágyazott kis- vagy közepes erőforrású processzoron. Ez alól azonban kivételt képeznek a kis, 1 körüli *N*-ek és az N_{max} körüliek, ugyanis itt még nincsenek nagyon szélsőséges értékek, és az együtthatók nagyságrendben viszonylag közel helyezkednek el egymáshoz. Nagyon kis *N* használata nyilván csak durva, pontatlan mérést tesz lehetővé, ráadásul csak felharmonikusban szegény jeleknél alkalmazható, ezért érdemes minél nagyobb *N*-t választani, lehetőleg a teljes spektrumot megvizsgálni.

Láttuk tehát, hogy $\alpha = 1$ esetén súlyos numerikus problémákba ütközhetünk, ezért számításaimat kiegészítettem ennek a paraméternek a vizsgálatával. Itt már két paramétertől (N,α) is függ az eredmény, ezért 3 dimenziós az ábra. Most már csak egy kitüntetett $g_{k,n}$ komponens abszolút értékét vizsgáltam, amely az egyenkomponensé, máskülönben az ábra áttekinthetetlen lenne. Két másik indok is van erre: egyrészt csak ez az egy együttható értelmezett minden elképzelhető *N*-re, másrészt a 3.6. ábra tanulságai szerint minden *N*-re ennek az abszolút értéke a legmagasabb értékű az együtthatók közt, ezért ez egy kritikus paraméter.

3.7. ábra: A becsatolóvektor elemeinek abszolút értéke *N* és *α* függvényében

Az eredményeket jól mutatja a 3.7. ábra. Látható, hogy az $\alpha = 1$ eset kiugrik a többi közül. Már 1-hez nagyon közeli, de annál kisebb α esetén is nagyságrendekkel csökkennek az együtthatók. α további csökkentése pedig még lejjebb nyomja az együttható abszolút értékét. Fény derül arra is, hogy $\alpha < 1$ használata esetén már nem kell feltétlenül nagyon kicsi vagy nagyon nagy *N*-et választani ahhoz, hogy ez az együttható ne vegyen fel szélsőséges abszolút értéket, az eredmények nem fognak be jelen viszonylatban széles tartományt. Mindezek alapján tehát nem érdemes $\alpha = 1$ választással élni, amennyiben a számábrázolási tartomány problémát okozhat, például beágyazott rendszerekben.

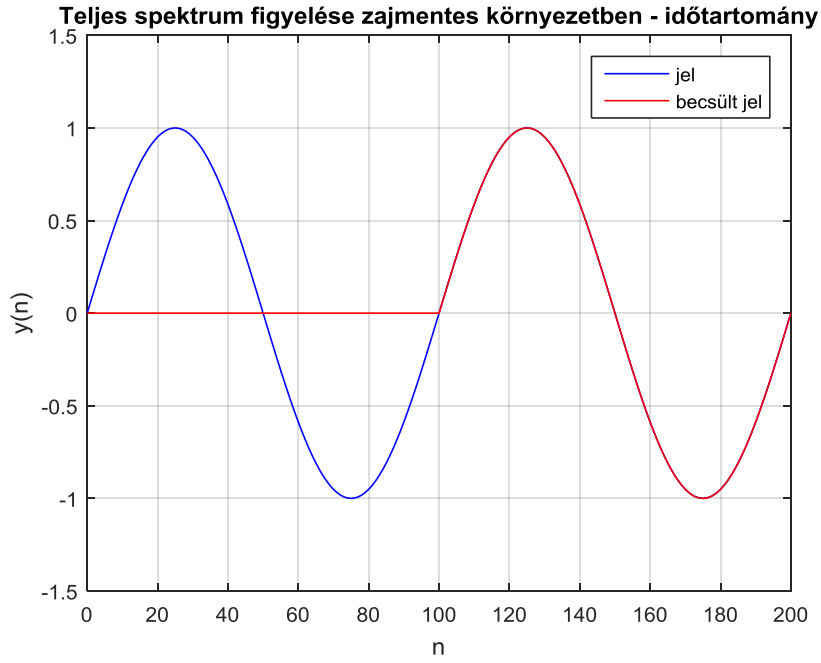
Diszkusszióra érdemes még az, hogy mekkora számítási igénnyel lép fel a megfigyelő együtthatóinak kalkulációja az n. ütemben. A (25) egyenletben egyedül $c_{k,n}$ értéke függ n-től, így célszerű $\frac{\Psi_k}{D_k}$ értékét offline, előre kiszámítani, és a rekurzív algoritmusban csupán a $conj(c_{k,n})$-nel való szorzást elvégezni. Ilyen módon az online számítási igény ütemenként nagyjából *N*-el egyenesen arányos.

3.1.3 Véges beállást eredményező becsatoló-együtthatókkal tervezett megfigyelő működése

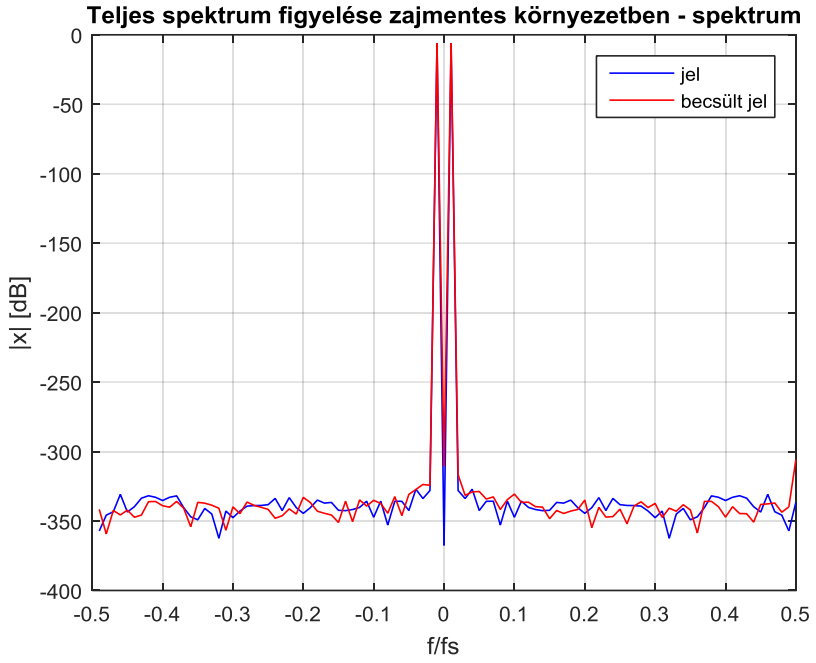
Ebben a fejezetben áttérek arra, hogyan működik a 3.1.2 fejezetben ismertetett módon megtervezett rezonátoros struktúrájú megfigyelő.

Vizsgálódásom során egy egyszerű $f_1 = 0{,}01$ mintavételi frekvenciához viszonyított frekvenciájú szinuszjelet használtam hasznos jelként, melynek amplitúdója egységnyi. A különböző beállításokkal végzett próbák során egyrészt lementettem a megfigyelő által kalkulált $\hat{y}_n$ értékeket, majd időtartományban ábrázoltam az így kiadódott hullámformát az eredeti jellel együtt. Emellett kiszámítottam a jel spektrumát is DFT-vel (a Matlab fft függvényével) referenciaként úgy, hogy a DFT pontok ugyanazokra a normalizált frekvenciákra essenek, amelyeket a megfigyelő is vizsgál, majd ezt szintén együtt ábrázoltam a teszt végére kiadódó $|\hat{x}_k|$ értékekkel.

Egyelőre a jel legyen zajmentes! Először megvizsgáltam akkor, ha $N = N_{max}$ és $\alpha = 1$. Az eredményeket mutatja a 3.8. ábra és a 3.9. ábra.



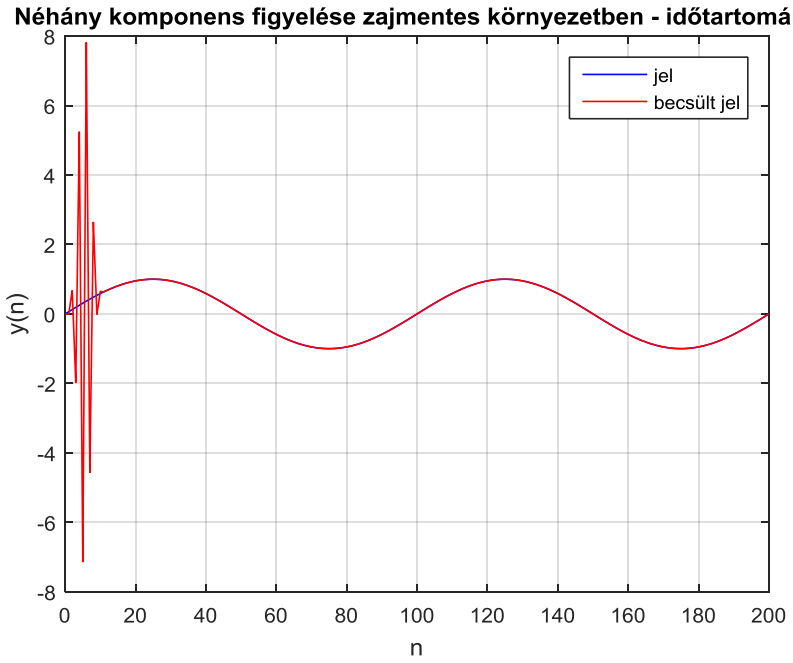
3.8. ábra: Zajmentes jel vizsgálata véges beállást eredményező g együtthatókkal, $N = N_{max}$



3.9. ábra: Zajmentes jel becsült és valós spektruma, $N = N_{max}$

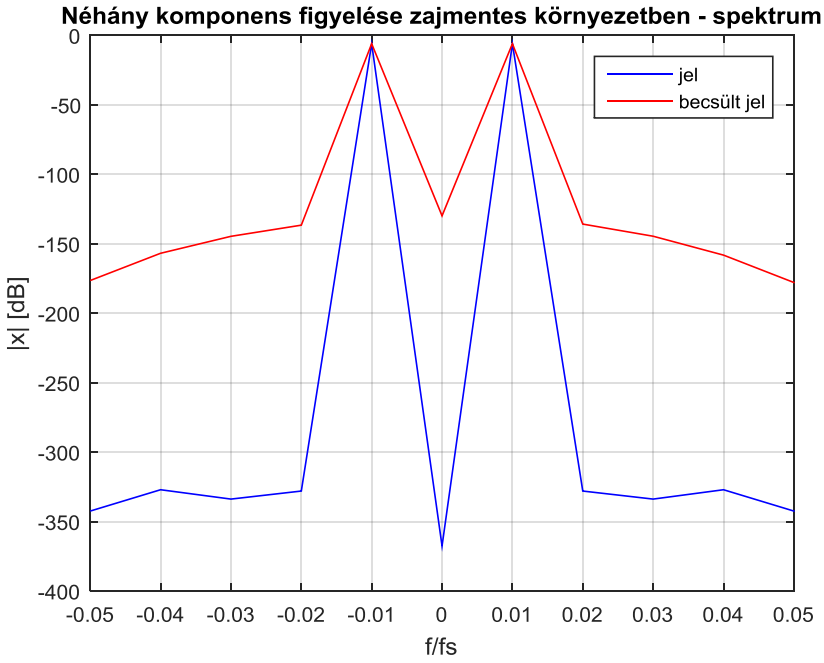
A megfigyelő megfelelően működik. A jel egy periódusa után követni tudja a megfigyelt jelet, és a becsült Fourier-együtthatók is megfelelnek a valóságosnak, a szinuszjelünk két csúcsa $\pm f_1$-nél jól elkülönül az alapzajtól. Ezután megváltoztattam N-et 11-re. Ez viszonylag kis N érték (ld. 3.1. ábra), de tekintve, hogy csak alapharmonikus szinuszunk van, elvileg így is minden nem nulla komponenst „lát” a megfigyelő. Az eredmények a 3.10. ábra és a 3.11. ábra szerintiek.

Több tanulság is van. Egyrészt a megfigyelő itt már egy periódus eltelte előtt ráállt a jelre, egészen pontosan 11 ütem után. Ez várható is volt, hiszen $\alpha = 1$ esetén $\Psi_k = 1$, így a megfigyelő összes pólusa zérus. Így ez egy véges beállású megfigyelő, amely N lépés alatt beáll. A beállása tehát nagyon gyors, azonban két problémát is észrevehetünk. Egyrészt a kezdeti lépésekben jelentős túllövések láthatóak a becsült jelben, ami nem előnyös. Másrészt bár a becsült spektrumban továbbra is jól elkülönülnek az alapharmonikus csúcsai, a többi részen nagyságrendekkel nagyobbak a Fourier-együtthatók a valósnál, mintha zajosabb lenne a szinusz.

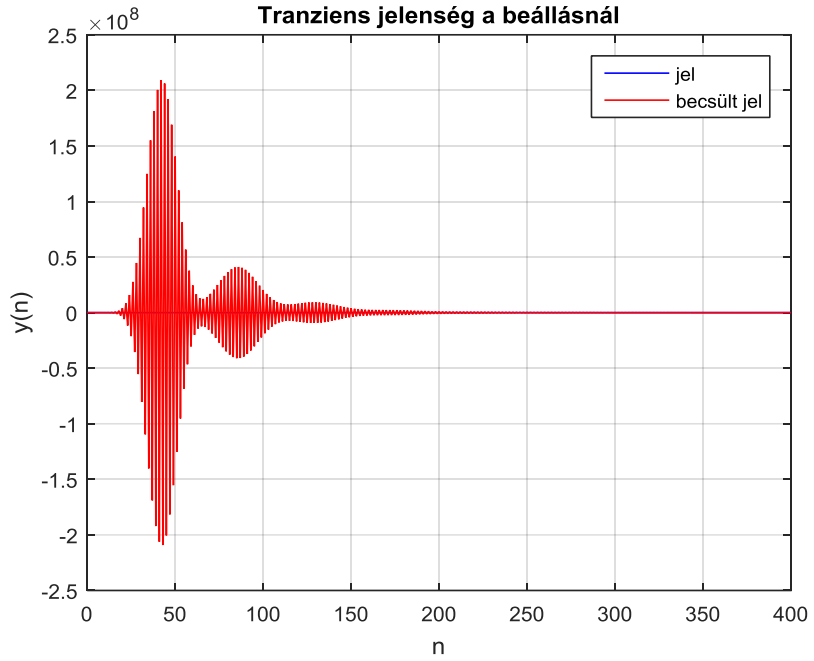


3.10. ábra: Zajmentes jel vizsgálata véges beállást eredményező g együtthatókkal, N = 11

Nem csak kis N-eknél fordulnak elő kellemetlen jelenségek a beállás során. Ha a másik irányból, a teljes spektrum vizsgálatából indulunk el, és fokozatosan csökkentjük N-et, egy idő után szintén nagy tranziensek jelennek meg a futás kezdetén. Egy ilyen példát mutat a 3.12. ábra. Ennek feltehetően numerikus okai lehetnek, hiszen mint azt korábban láttuk, $\boldsymbol{G}_n$ vektor elemeinek abszolút értéke nagyon változatos.



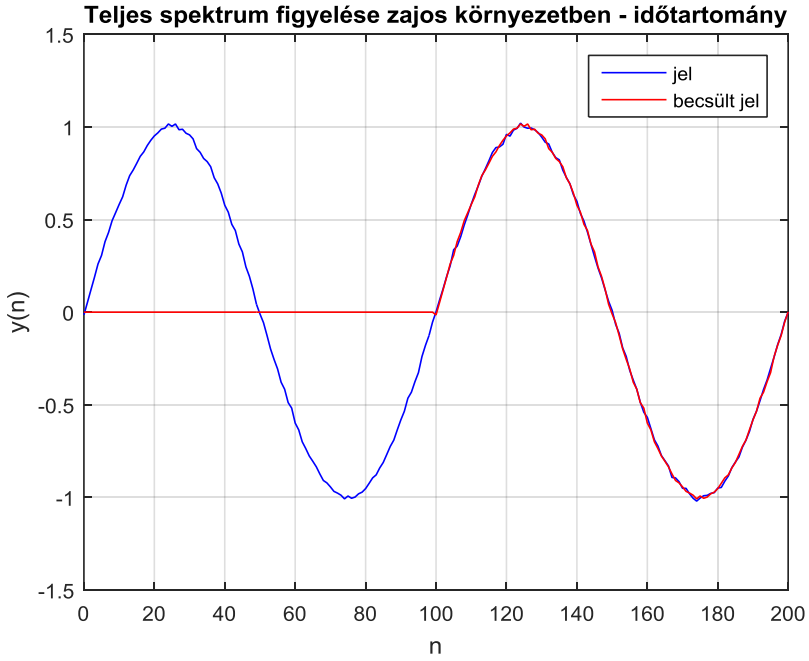
3.11. ábra: Zajmentes jel becsült és valós spektruma, N = 11



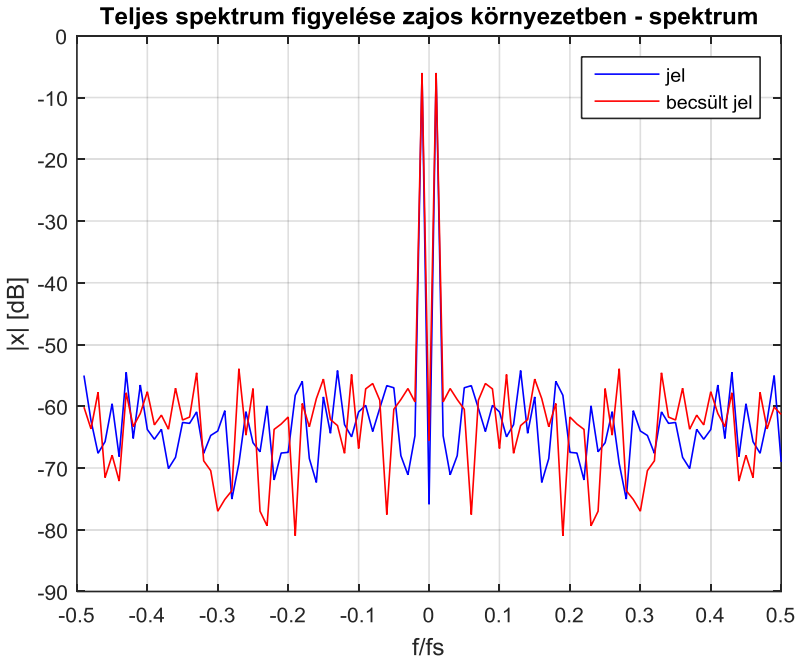
3.12. ábra: Zajmentes jel, N = 85. Bár nem látszik, a megfigyelő később tényleg felveszi a jel alakját

Idáig feltételeztük, hogy a jelen nincs zaj. Ez nem reális egy beágyazott rendszerben, hiszen számos külső és belső zaj csatolódhat be a mintavételezett analóg jelbe, az analóg-digitális átalakító (ADC) kvantálása is zajként jelentkezik, stb. Ezért a továbbiakban zajt adtam a szinuszjelhez. A felhasznált zajt a Matlab wgn függvényével generáltam, ami fehér zajt generál, teljesítménynek -40 dB-t állítottam, amelynél a jel ránézésre bőven felismerhető marad, a különbség a zajtalanhoz képest nem látványos.

Az eddigi tapasztalatok alapján az előnyösebbnek tűnő nagy N-ekkel folytattam, és mint kiderült, $N = 99$ *és* 100 mellett kalkulálja ki igazán jól a spektrumot. Ilyenkor az időtartománybeli jel is alapvetően rendben van, bár 99 esetén zajosabbnak látszik. Az $N = 100$ választással kapott eredményeket mutatja a 3.13. ábra és a 3.14. ábra.



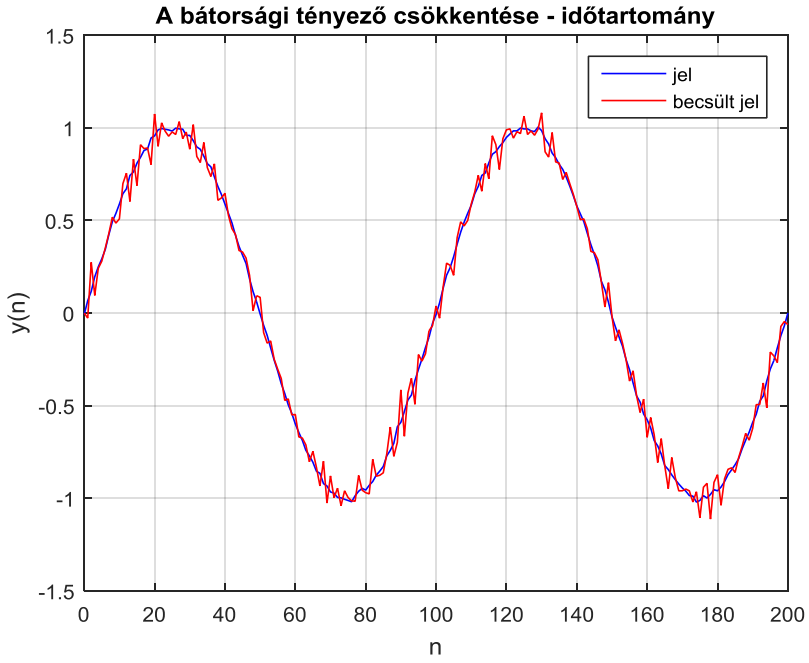
3.13. ábra: Teljes spektrum figyelése (N = 100) zajos jellel, α = 1 mellett



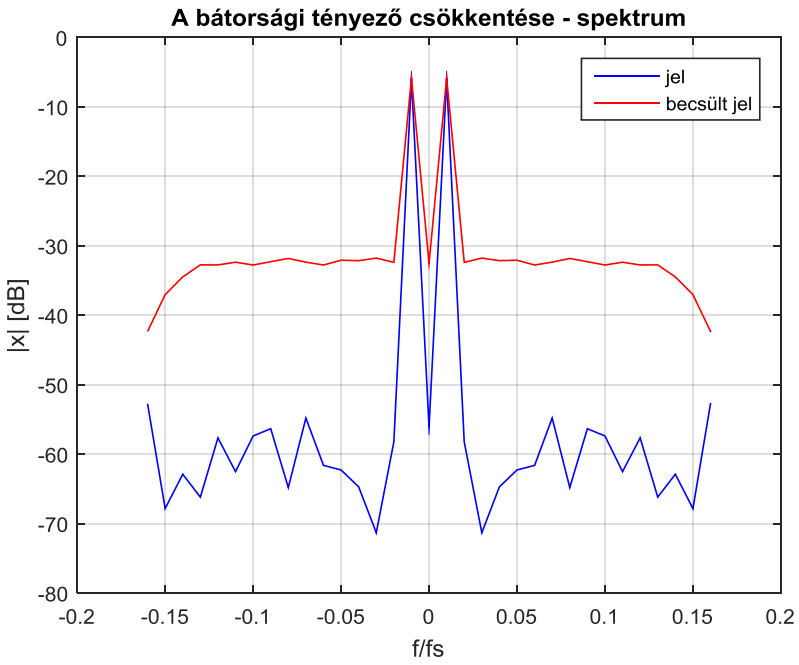
3.14. ábra: Teljes spektrum figyelése (N = 100) zajos jellel, α = 1 mellett

Még egy paraméter van, mellyel eddig nem foglalkoztam, ez pedig az α. A továbbiakban a zajos jel becslését $\alpha < 1$ választással figyeltem. Ahogyan azt az előző fejezetben láthattuk, ennek drasztikus hatása volt $g_{k,n}$ abszolút értékére már 1-hez egészen közeli, de annál kisebb α-nál is.

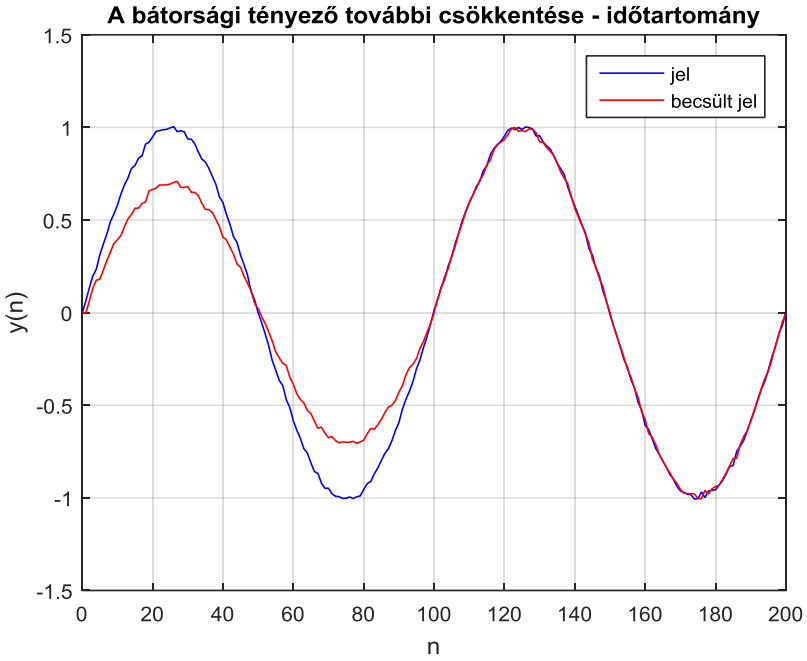
Először olyan N-ekkel próbálkoztam, amelyek előzetesen problémásnak tűntek a becsatoló vektor együtthatói miatt, hogy lássam, ilyen esetekben a $g_{k,n}$ értékekre gyakorolt jótékony hatás mellett milyen eredmény adódik a működésben. Az eredmény megnyugtató volt, ugyanis még a legproblémásabb $N = 33$ választással is már $\alpha = 0{,}99$ beállítással képes követni a jelet. A spektrumon pedig megjelenik, ráadásul nagyjából amplitúdóhelyesen a szinuszhoz tartozó két csúcs (ld. 3.15. ábra és 3.16. ábra). Az idő- és frekvenciatartománybeli becslés is zajos ugyan kissé, de α további csökkentésével ez is kiküszöbölhető (ld. 3.17. ábra és 3.18. ábra, ahol $\alpha = 0{,}7$).



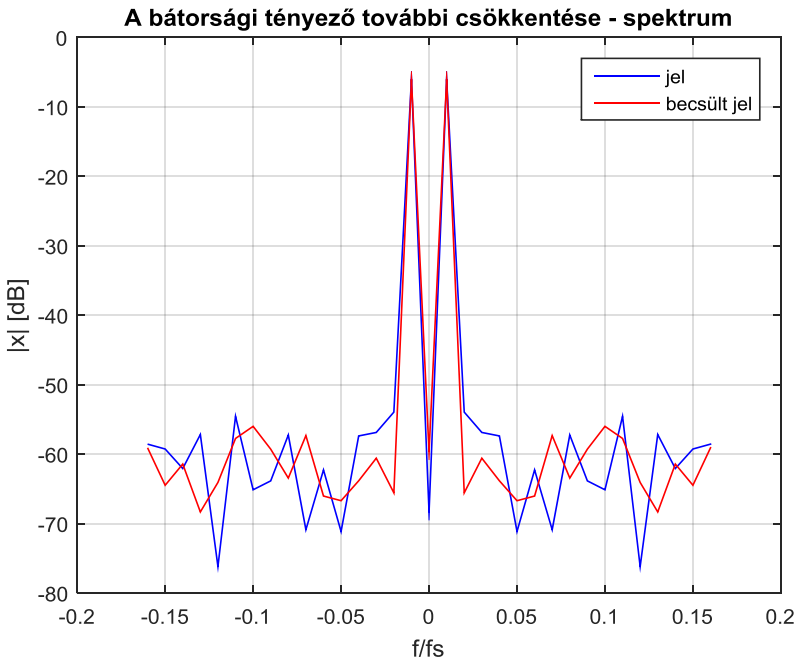
3.15. ábra: *N* = 33, α = 0,99



3.16. ábra: N = 33, α = 0,99



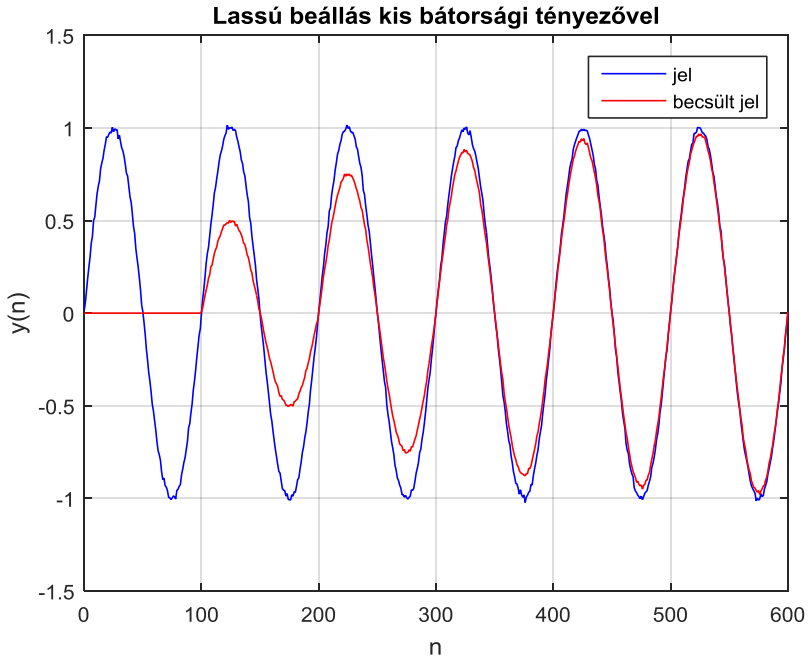
3.17. ábra: N = 33, α = 0,7



3.18. ábra: N = 33, α = 0,7

Láttuk tehát, hogy α csökkentésével érzéketlenebb lesz a zajra a megfigyelő, a spektrumban pedig csökken a felharmonikusok zajból eredő amplitúdója. Lényegében egy zajszűrést valósítunk így meg. Ennek azonban negatív következménye is van: nő a beállási idő. Ez a kettő következik egymásból, hiszen a zajra való érzéketlenség azt jelenti, hogy a spektrum változásait nem követi gyorsan. A zajnál ez nyilván előnyös, hiszen így az apró, egy érték körül ingadozó változásokra nem reagál. A jelnél viszont így csak lassan veszi fel a Fourier-együtthatók értékét a kezdetben nullának inicializált $\hat{\boldsymbol{x}}$. Ez alapján tehát α egy bátorsági tényezőként értelmezhető. A lassú beállás jelenségét mutatja a 3.19. ábra.

A tapasztalatokat összegezve tehát numerikus szempontokat figyelembe véve érdemes lehetőleg a teljes spektrumot vizsgálni és nem csak a kisfrekvenciás összetevőket, vagyis célszerű az $N = N_{max}$ választás, amennyiben ezt a számítási kapacitás lehetővé teszi. Zajos környezetben α értékét érdemes 1-nél kisebbre állítani, megfontolva a zaj nagyságát és a kívánt beállási időt.



3.19. ábra: N = 100, α = 0,5 mellett lassú a beállás

3.1.4 A becsatoló-együtthatók számítása egyenletes rezonátoreloszlásnál

A (25)-(28) egyenletek kiértékelése és a velük megtervezett megfigyelő használata, mint láttuk, bizonyos esetekben nehézségekbe ütközik. Azonban a $\frac{\Psi_k}{D_k}$ tört értéke egy rendkívül egyszerű alakká válik, ha teljesül az $f_1 = \frac{1}{N}$ feltétel. Ekkor ugyanis[15]:

$$\left.\frac{\Psi_k}{D_k}\right|_{f_1=\frac{1}{N}} = \frac{\alpha}{N} \tag{32}$$

És így

$$g_{k,n} = \frac{\alpha}{N} \cdot conj(c_{k,n}) \tag{33}$$

adódik. Ez a $\boldsymbol{G}_n$ vektor együtthatóinak nagyon egyszerű és gyors számítását teszi lehetővé. Ha visszanézünk, például a 3.3. ábra is ezt mutatja, ahol N_{max}-ra igaz volt, hogy reciproka f_1-el egyezik meg, és így a görbék az N_{max} helyen egy pontban találkoztak, amelynek értéke pont 0,01. Megjegyzem azonban, hogy bár az eddig vizsgált esetekben és ábrákon ez mindig így volt, ez nincs szükségképpen így, például $f_1 = 0{,}015$ esetén nem igaz, hogy $f_1 = \frac{1}{N_{max}}$, és a görbék nem is egy pontban végződnek.

3.1.5 Gyakorlati alkalmazásokban használt együtthatók

Mint az előző fejezetben láttuk, az $f_1 = \frac{1}{N}$ feltétel teljesülése esetén a $\boldsymbol{G}_n$ vektor elemeinek értéke egy nagyon egyszerűen és gyorsan számítható alakra (33) egyszerűsödik, ugyanis $\boldsymbol{C}_n$-t csak egy skalárral kell megszorozni. Ez egyrészt egy sokkal egyszerűbben implementálható képlet, másrészt a számítási igényt jelentősen csökkenti a kezdeti inicializálások során, ráadásul függetleníti *N*-től is, hiszen a becsatoló vektor minden elemét ugyanazzal az $\frac{\alpha}{N}$ értékkel kell megszorozni. Ez bizonyos alkalmazásokban, ahol a kezdeti feléledés ideje nem lényegtelen, kifejezetten előnyös lehet.

További előny, hogy ez a kifejezés lehetővé teszi, hogy az $\frac{\alpha}{N}$ skalárszorzót kiemeljük $\boldsymbol{G}_n$-ből, és ahelyett, hogy annak frissítésekor megszoroznánk vele $\boldsymbol{C}_n$-t, inkább az e_n hibajelet szorozzuk meg vele és így adjuk a rezonátorcsatornákra.

A gyakorlatban ezekért a (33) formát gyakran alkalmazzák olyan esetekben is, amikor a feltétele nem teljesül. Vagyis minden *N*-re:

$$g_{k,n} = \frac{\alpha}{N} \cdot conj(c_{k,n}) \tag{34}$$

És ezt használjuk fel az algoritmusban:

$$\hat{y}_n = \boldsymbol{C}_n \hat{\boldsymbol{x}}_n \tag{35}$$

$$e_n = y_n - \hat{y}_n \tag{36}$$

$$\hat{\boldsymbol{x}}_{n+1} = \hat{\boldsymbol{x}}_n + e_n \boldsymbol{G}_n \tag{37}$$

A 3.3 fejezetben bemutatom, hogy ez az egyszerűsítés nem rontja el az algoritmus működőképességét sem.

3.2 Kálmán-szűrős megközelítés

3.2.1 Idővariáns abszolút értékű Kálmán-szűrős algoritmus

Egy másik lehetséges rekurzív spektrumbecslő megközelítést kapunk a Kálmán-szűrő elméletének felhasználásával. Az algoritmus és részletes elemzése [5]-ben található időinvariáns jelmodellhez valós számokkal való leírással, azonban én itt áttérek a tömörebb és szemléletesebb komplex leírásra. Ennek az is oka, hogy így könnyebb összevetni a rezonátoros algoritmussal. A benne szereplő $\boldsymbol{A}$ és $\boldsymbol{C}$ mátrixok átértelmezésével az ott leírtak kevés változtatással felhasználhatóak idővariáns

jelmodellhez is. Az algoritmus értelmezéséhez azonban ki kell egészítenünk a jelmodellt sztochasztikus folyamatokkal az alábbi módon:

$$\boldsymbol{x}_{n+1} = \boldsymbol{A}\boldsymbol{x}_n + \boldsymbol{H}\boldsymbol{w}_n \tag{38}$$

$$y_n = \boldsymbol{C}_n\boldsymbol{x}_n + v_n \tag{39}$$

$\boldsymbol{w}_n$ és v_n nulla körüli fehér zajok, amelyek teljesítik, hogy

$$\mathbb{E}\left(\begin{bmatrix}\boldsymbol{w}_n \\ v_n\end{bmatrix} \quad [\boldsymbol{w}_m^T \quad v_m^T]\right) = \begin{bmatrix}\boldsymbol{Q}\delta_{nm} & 0 \\ 0 & R\delta_{nm}\end{bmatrix} \tag{40}$$

ahol $\boldsymbol{Q} \geq \boldsymbol{0}, R > 0$. Továbbá feltételezzük, hogy $\boldsymbol{x}_0$ egy $\bar{x}_0$ körüli gaussi valószínűségi változó, amelynek kovarianciája $\boldsymbol{P}_0 \geq 0$ [5].

A fenti összefüggésekben $\boldsymbol{w}_n$ értelmezhető úgy, mint állapotzaj, amely az állapotváltozók bizonytalansága. Ez eredhet például a jelgenerátorbeli számítások során történt kerekítésekből, kvantálásokból, vagy a valós megfigyelt jel paramétereinek ingadozásából, változásából. A v_n értelmezése egyszerűbb: ez a jelbe becsatolódott zaj.

Ezekkel a feltételezésekkel és meghatározásokkal az algoritmus egyenletei[5]:

$$\hat{\boldsymbol{x}}_{n+1|n} = (\boldsymbol{A} - \boldsymbol{K}_n\boldsymbol{C}_n)\hat{\boldsymbol{x}}_{n|n-1} + \boldsymbol{K}_n y_n \tag{41}$$

$$\hat{\boldsymbol{x}}_{0|-1} = \hat{\boldsymbol{x}}_0 \tag{42}$$

$$\boldsymbol{K}_n = \boldsymbol{A}\boldsymbol{\Sigma}_{n|n-1}\boldsymbol{C}_n^H(\boldsymbol{C}_n\boldsymbol{\Sigma}_{n|n-1}\boldsymbol{C}_n^H + R)^{-1} \tag{43}$$

$$\boldsymbol{\Sigma}_{n+1|n} = \boldsymbol{A}\left[\boldsymbol{\Sigma}_{n|n-1} - \boldsymbol{\Sigma}_{n|n-1}\boldsymbol{C}_n^H\left(\boldsymbol{C}_n\boldsymbol{\Sigma}_{n|n-1}\boldsymbol{C}_n^H + R\right)^{-1} \cdot \boldsymbol{C}_n\boldsymbol{\Sigma}_{n|n-1}\right]\boldsymbol{A}^T + \boldsymbol{H}\boldsymbol{Q}\boldsymbol{H}^H \tag{44}$$

$$\boldsymbol{\Sigma}_{0|-1} = \boldsymbol{P}_0 \tag{45}$$

A fenti egyenetekben $\boldsymbol{K}_n$ a Kálmán-erősítés, $\boldsymbol{\Sigma}_{n+1|n}$ pedig az $\hat{\boldsymbol{x}}_{n+1|n}$ kovarianciája.

Az egyenletekben a felső indexben lévő *H* operátor a transzponált konjugáltját jelöli. Az alsó indexben lévő *p|p-1* alakú jelölések értelmezése: *p*. ütembeli értéke a (*p-1*). ütembeli mintával bezárólag szerzett ismeretek alapján.

Ezekből pedig már könnyen kalkulálható az n. ütemre adott $\hat{y}$ becslés:

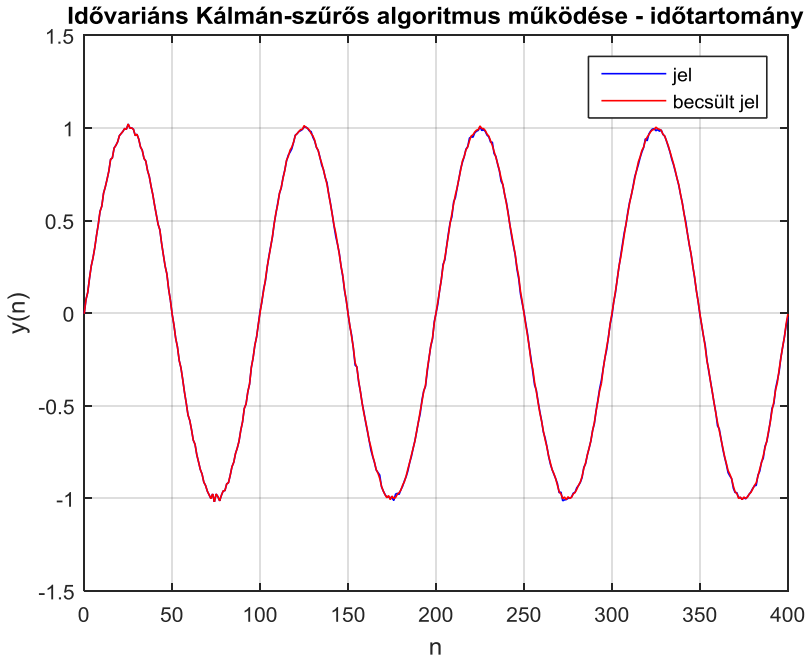
$$\hat{y}_n = \boldsymbol{C}_n\hat{\boldsymbol{x}}_{n+1|n} \tag{46}$$

Néhány pontban itt is összegezném a működést:

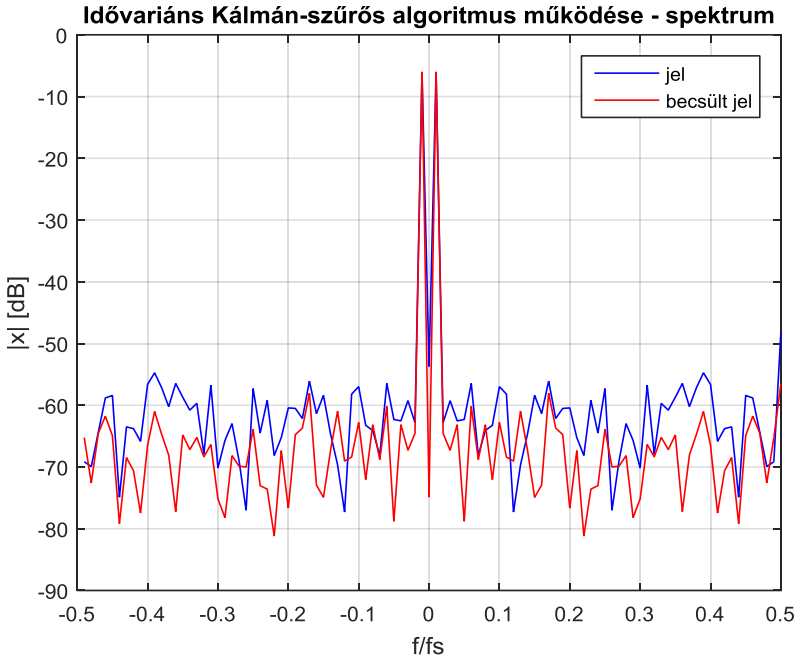
- Frissítjük a becsült állapotvektor kovarianciáját.
- Meghatározzuk az új Kálmán-erősítést.
- A Kálmán-erősítés és az új minta alapján frissítjük a becsült állapotvektort.
- Becslést adunk a jelre a becsült Fourier-együtthatókból.

Néhány kezdeti érték kiválasztását még el kell végezni a használat előtt. $\hat{\boldsymbol{x}}_0 = 0$ választás indokolt, hiszen kezdetben semmilyen információnk nincs a jelről. $\boldsymbol{P}_0$-al kapcsolatban előzetes ismereteink nincsenek, ezért a $\boldsymbol{P}_0 = \boldsymbol{I}$ választással éltem, mivel az egy nagyságrendbe esik a megfigyelt jel amplitúdójával. $\boldsymbol{Q}$-t kezdetben válasszuk a csak nullákat tartalmazó mátrixnak[5]! R-el kapcsolatban lehetnek előzetes ismereteink, ez ugyanis a jelforrás zajosságával áll összefüggésben. Ezt „üres” méréssel, vagyis amikor a jelforrást a rendszerünk bemenetére kötjük, de jelet nem adunk rá, meg lehet becsülni. Mivel jelen esetben -40 dB-es zajjal vizsgálódtam, ezért $R = 0{,}01$ választással éltem.

Mindezek felhasználásával megvizsgáltam a Kálmán-szűrős algoritmus működését ahhoz hasonlóan, ahogyan a rezonátorosét is. Ezúttal a zajmentes jelek vizsgálata nem rejt különösebb érdekességeket önmagában, ezért rögtön a zajos esetre térek. A rezonátoros algoritmushoz hasonlóan ezt is egy $f_1 = 0{,}01$ normalizált frekvenciájú szinusszal vizsgáltam, és ugyanúgy lementettem az időtartománybeli hullámformákat, valamint a jel DFT-vel számított és az algoritmus működéséből születő spektrumát. Az eredményeket a 3.20. ábra és a 3.21. ábra mutatja.



3.20. ábra: *N* = 100, *R* = 0,01, -40 dB fehér zaj



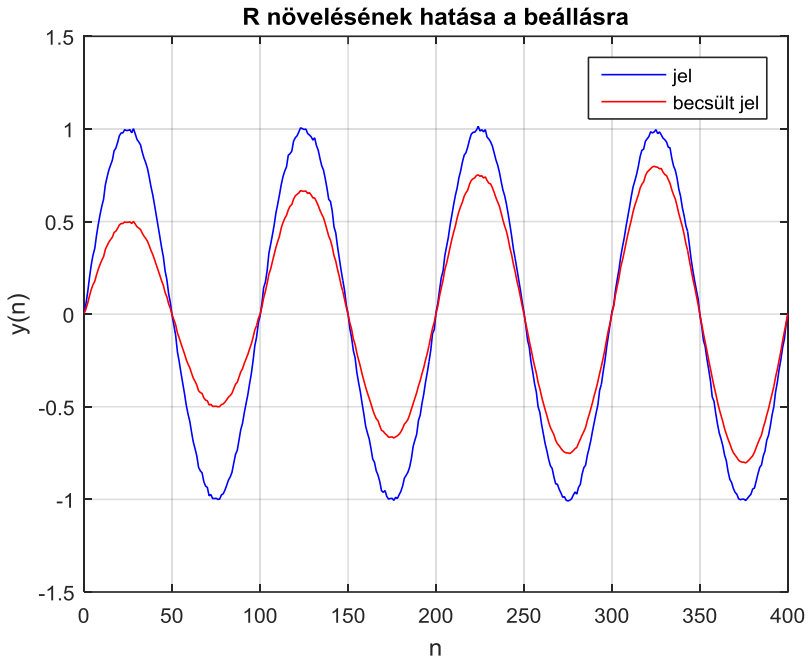
3.21. ábra: *N* = 100, *R* = 0,01, -40 dB fehér zaj

A rendszer időtartományban jól követi a jelet, még simább, kevésbé zajos is a becslés. A spektrumon sem látunk semmilyen problémát, jól látszik a szinuszhoz tartozó csúcs, és még amplitúdóhelyes is. További vizsgálatok után kiderült, ez az algoritmus nem érzékeny *N* változtatására sem, a hullámforma és a spektrum is helyes marad. Ez a megközelítés tehát meglehetősen stabil és robusztus. Meg kell azonban jegyeznünk, hogy az összehasonlíthatóság a rezonátoros algoritmussal abból a szempontból sérül, hogy az

annál tapasztalt numerikus problémák $\alpha = 1$ esetén adódtak, ami a Kálmán-szűrős megközelítés esetén a zajmentes esetet jelenti, amelynek vizsgálatát a bemeneti zaj feltételezése ($R > 0$) eleve kizárja. A fenti vizsgálatok tehát a rezonátor alapú eljárás esetén $\alpha < 1$ esetnek felelnek meg, ahol az is kellően robusztus volt.

Érdemes még megemlíteni, hogy R hasonló funkciójú paraméter, mint a rezonátoros megfigyelőnél α, azzal a különbséggel, hogy itt R növelésével lassul a beállás. Erre mutat példát a 3.22. ábra, amely ugyanazokkal a beállításokkal készült, mint a 3.20. ábra, kivéve, hogy $R = 100$.

Az algoritmus hátránya viszont, hogy számításigényes. A fentebb részletezett számításokat minden ütemben el kell végezni, hiszen a Kálmán-erősítés együtthatói és $\boldsymbol{\Sigma}$ idővariánsak. A nagy számítási igény pedig korlátozó tényezőként léphet fel egy beágyazott rendszerben.



3.22. ábra: *R* növelése lassítja a beállást

3.2.2 A szűrő-divergencia problémája

Az idővariáns Kálmán-szűrős megvalósításnak van egy másik problémája is a számításigény mellett, ez pedig a szűrő-divergencia[5]. A jelenség lényege, hogy a Kálmán-szűrős rendszer az idő előrehaladtával egyre inkább csak az általa számított $\hat{\boldsymbol{x}}$-ot veszi figyelembe az állapotvektor frissítésekor, figyelmen kívül hagyva a beérkezett y_n

mintát, tehát lényegében függetlenedik a megfigyelt jeltől. Ennek oka, hogy a Kálmán-erősítés együtthatói a nullába tartanak[5]:

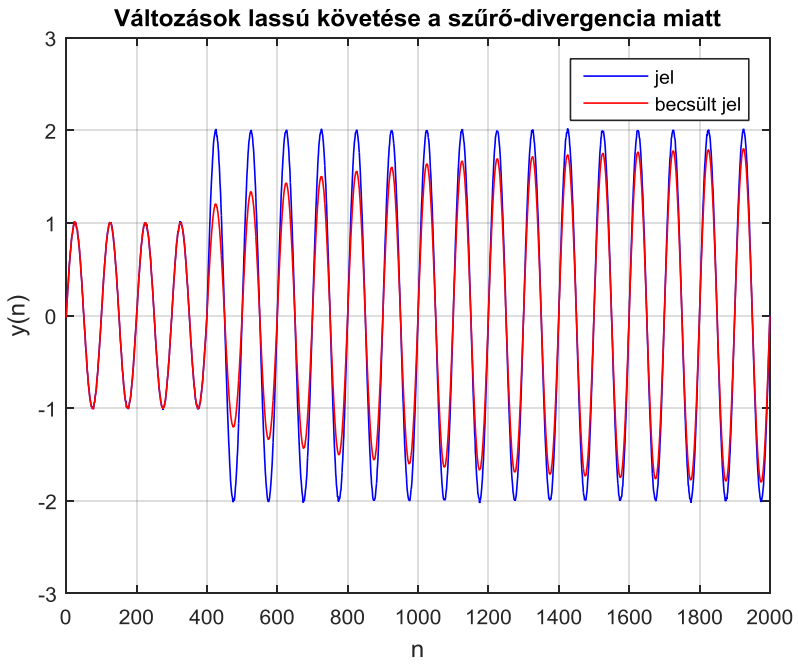
$$\lim_{n\to\infty} \boldsymbol{K}_n = \boldsymbol{0} \tag{47}$$

Ez azért van, mert $\boldsymbol{Q} = \boldsymbol{0}$, ami azt feltételezi, hogy az állapotváltozók nem változnak időben, tehát ha egyszer megtaláltuk a pontos értéküket, akkor azt kell tartani.

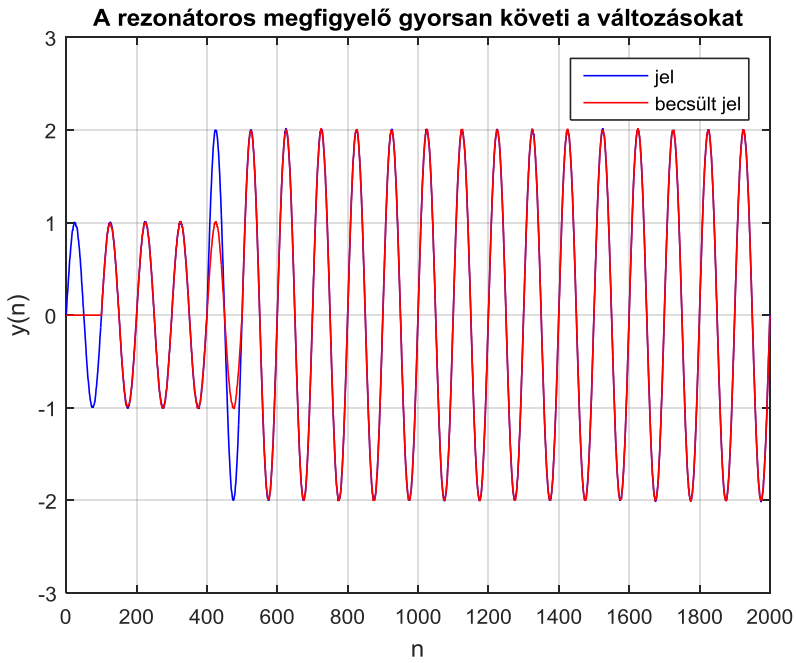
Könnyen belátható, hogy (47) hatása a beérkező minták figyelmen kívül hagyása, ha megvizsgáljuk a (41) egyenletet. Ha $\boldsymbol{K}_n = \boldsymbol{0}$, akkor az alábbi alakra egyszerűsödik:

$$\hat{\boldsymbol{x}}_{n+1|n} = \boldsymbol{A}\hat{\boldsymbol{x}}_{n|n-1} \tag{48}$$

Ez nem baj, ha a jelforrás állapotváltozói valóban sosem változnak, ekkor ugyanis a beállás után nincs szükség változások adaptálására, és így a jelenség egy zajszűrésként viselkedik, ahogyan az észrevehető a 3.21. ábra görbéin is. A teljes változatlanság feltételezése azonban nem igazán életszerű. Nézzük meg, mi történik, ha az f_1 frekvenciájú egységnyi amplitúdójú szinuszjel amplitúdója hirtelen kétszeresére vált. Ezt az esetet mutatja a 3.23. ábra. Látható, hogy a változást nehezen követi a rendszer, csak nagyon sokára sikerül nagyjából átállnia az új amplitúdóra. Ez természetesen annál több időbe telik, minél régebb óta fut a program.



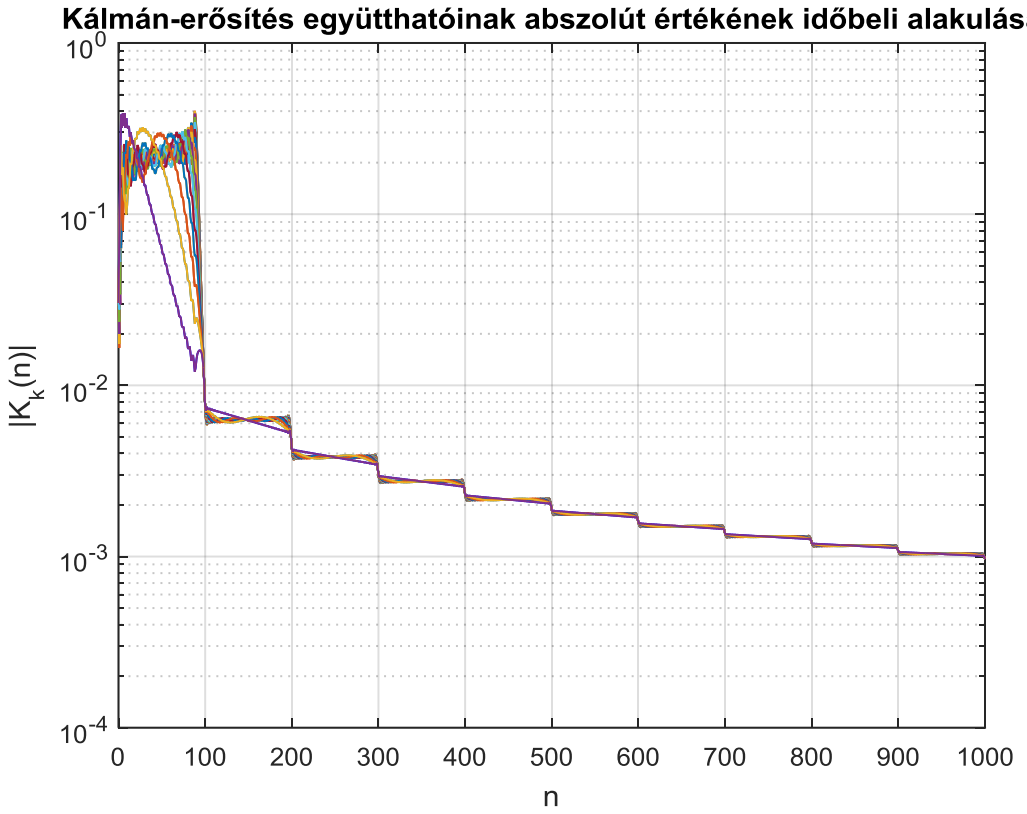
3.23. ábra: Az idővariáns abszolút értékű Kálmán-szűrős algoritmus csak nagyon lassan adaptálja a változásokat a szűrő-divergencia miatt



3.24. ábra: A rezonátoros megfigyelő sokkal gyorsabban rááll az új jelalakra

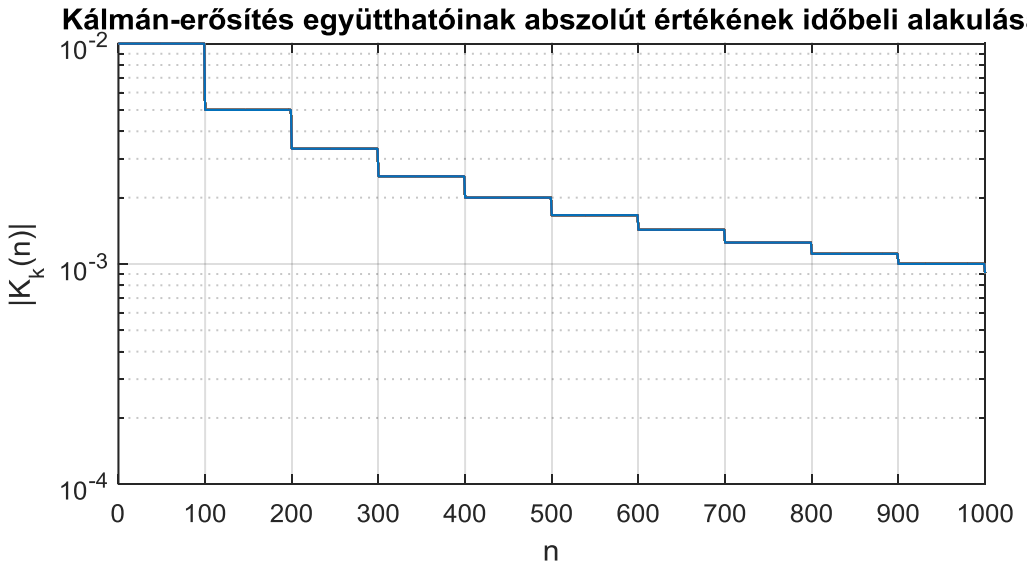
Összehasonlításképpen a 3.24. ábra mutatja, hogy a rezonátoros algoritmusnak ez nem okoz ekkora gondot, az alapharmonikus frekvencia 1 periódusideje után adaptálja az új jelet.

Megvizsgáltam a Kálmán-erősítés együtthatóinak időbeli változását is. A tesztelés során lementettem az erősítés-együtthatók abszolút értékét, majd az ütemszám függvényében ábrázoltam őket. Az eredményt mutatja $N = 33$ mellett a 3.25. ábra, amin jól látszik a csökkenő tendencia. Az értékek közt nincsenek szélsőséges eltérések egy ütemen belül, nagyjából ugyanolyan sebességgel tartanak a nullába. Észrevehetjük még, hogy periódusidőnként egy-egy nagyobb, ugrásszerűen meredek csökkenés következik be, így egy lépcsős-szerű függvényt hozva létre.



3.25. ábra: Szűrő-divergencia (*N* = 33)

Az $f_1 = \frac{1}{N}$ eset itt is különleges, ugyanis a Kálmán-erősítés együtthatóinak abszolút értéke ilyenkor minden ütemben megegyezik, így ekkor egyetlen görbét látunk csak az időbeli változást mutató grafikonon, amely minden periódus alatt konstans, majd a végén ugrik egyet a nulla felé.



3.26. ábra: Szűrő-divergencia (*N* = 100)

3.2.3 A modell egyszerűsítése, időinvariáns abszolút értékű Kálmán-erősítés használata

Láttuk tehát, hogy az idővariáns abszolút értékű Kálmán-szűrős megközelítés előnyei ellenére olyan negatívumokkal rendelkezik, amelyek problémássá tehetik felhasználását. A legjobb az lenne, ha sikerülne tervezni egy időinvariáns abszolút értékű Kálmán-erősítést, mégpedig egy $\overline{\boldsymbol{\Sigma}}$ állandó értékű $\boldsymbol{\Sigma}_n$ választásával[5].

A levezetést mellőzve, ha élünk a

$$\overline{\boldsymbol{\Sigma}} = \boldsymbol{P} = \varepsilon \boldsymbol{I}, \;\; \varepsilon > 0 \tag{49}$$

választással, amiben implicite benne van az is, hogy most már $\boldsymbol{Q} \neq \boldsymbol{0}$, tehát van állapotzaj, akkor $\boldsymbol{K}_n$ generálható a

$$\boldsymbol{K}_n = \boldsymbol{A}\overline{\boldsymbol{\Sigma}}\boldsymbol{C}_n^H(\boldsymbol{C}_n\overline{\boldsymbol{\Sigma}}\boldsymbol{C}_n^H + R)^{-1} = \boldsymbol{A}\varepsilon\boldsymbol{I}\boldsymbol{C}_n^H(\boldsymbol{C}_n\varepsilon\boldsymbol{I}\boldsymbol{C}_n^H + R)^{-1} \tag{50}$$

egyenlet szerint[5]. Ezt pedig behelyettesítve a (41) egyenletbe az algoritmus működőképes. $\boldsymbol{I}$ az *NxN*-es egységmátrix.

Fontos megjegyezni, hogy az időinvarianca csak a Kálmán-erősítés abszolút értékére igaz, nem általánosságban. Az egyetlen időfüggő tag a képletében a $\boldsymbol{C}_n$ vektor, emiatt azonban továbbra is minden ütemben ki kell számolni $\boldsymbol{K}_n$-t. Teljes időinvariancia csak időinvariáns jelmodell mellett lehetséges.

3.3 A két algoritmus egyesítése

Eddig bemutattam két különböző eredetű algoritmust, melyekkel kapcsolatban a szakirodalom alapján érdemes volt különböző egyszerűsítésekkel élni. Itt megismétlem az ezek után kapott összefüggések vektoriális alakjait, majd bemutatom, hogy ezek csupán formailag térnek el egymástól.

$$\boldsymbol{G}_n = \frac{\alpha}{N}\boldsymbol{C}_n^H \tag{51}$$

$$\boldsymbol{K}_n = \boldsymbol{A}\varepsilon\boldsymbol{I}\boldsymbol{C}_n^H(\boldsymbol{C}_n\varepsilon\boldsymbol{I}\boldsymbol{C}_n^H + R)^{-1} \tag{52}$$

Az egyenletekben $0 < \varepsilon, \alpha \leq 1$.

3.3.1 Az egyszerűsített modellek azonossága

Mivel az (52) egyenlet alakja bonyolultabb, ezt kezdem el egyszerűsíteni. Mivel $\boldsymbol{A} = \boldsymbol{I}$, és $\boldsymbol{I}$ az *NxN*-es egységmátrix, rögtön egyszerűbb alakra hozható:

$$\boldsymbol{K}_n = \varepsilon \boldsymbol{C}_n^H (\boldsymbol{C}_n \varepsilon \boldsymbol{C}_n^H + R)^{-1} \tag{53}$$

Következő lépésben vegyük észre, hogy mivel $\boldsymbol{C}_n^H$ a $\boldsymbol{C}_n$ transzponált konjugáltja, a $\boldsymbol{C}_n \varepsilon \boldsymbol{C}_n^H$ kifejezés értéke egy skalár: εN. Emiatt viszont $(\boldsymbol{C}_n \varepsilon \boldsymbol{C}_n^H + R)^{-1}$ értéke is egy skalár, tehát az egyenlet átírható a következő alakba:

$$\boldsymbol{K}_n = \frac{\varepsilon}{\varepsilon N + R} \boldsymbol{C}_n^H \tag{54}$$

Lényegében készen is vagyunk. Kiderült, hogy $\boldsymbol{G}_n$ és $\boldsymbol{K}_n$ is $\boldsymbol{C}_n^H$ vektor skalárszorosa. A két vektor elemeinek értéke is megegyezik, ha

$$\alpha = \frac{\varepsilon}{\varepsilon N + R} = \frac{1}{N + \frac{R}{\varepsilon}} \tag{55}$$

Ebből az összefüggésből az is kiderül, hogy ε szerepe hasonló α-éhoz. Szintén bátorsági tényezőként fogható fel, amely értékét növelve a beállás gyorsul, ugyanakkor a zajra érzékenyebb lesz a megfigyelő. Ugyanakkor nem szabad elfelejteni, hogy a Kálmán-szűrős megközelítésben nem ez az egyetlen ilyen funkciójú változó, R is erre szolgál, így mindkettő helyes megválasztása szükséges (voltaképpen az R és ε aránya szükséges). Az is belátható még, hogy az egyenlőség $\alpha = 1$ esetén csak úgy állhat fenn, ha $R = 0$, vagyis zajmentes bemeneti csatornát feltételezünk, viszont ekkor ε értékétől független ez az egyenlőség.

Az algoritmusok közt különbség mindössze abban van, hogy a Kálmán-szűrős a már frissített állapotvektor ismeretében adja meg becslését $\hat{y}_n$-ra (46) szerint, ez azonban nem szükségszerű, ugyanis $\hat{\boldsymbol{x}}_{n|n-1}$ szerint is adhatná, amelyet az állapotvektor frissítéséhez is felhasznál. A továbbiakban ezért áttérek erre a módszerre, vagyis

$$\hat{y}_n = \boldsymbol{C}_n \hat{\boldsymbol{x}}_{n|n-1} \tag{56}$$

így a Kálmán-szűrős algoritmus a (41) átrendezésével

$$\hat{\boldsymbol{x}}_{n+1|n} = \boldsymbol{A}\hat{\boldsymbol{x}}_{n|n-1} + \boldsymbol{K}_n\left(y_n - \boldsymbol{C}_n \hat{\boldsymbol{x}}_{n|n-1}\right) \tag{57}$$

ugyanúgy működik, mint a rezonátoros:

- Megbecsüli a jel értékét az addigiak alapján számolt Fourier-együtthatókkal.
- A jel valós és becsült értékéből kiszámítja a becslési hibát.

- A becslési hiba alapján frissíti az $\hat{\boldsymbol{x}}$ állapotvektort.

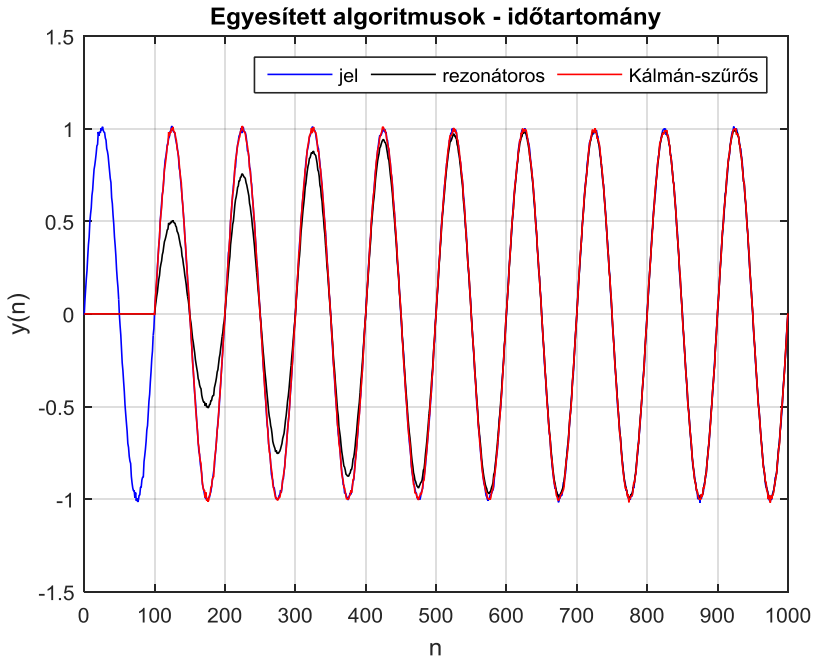
Fontos még kiemelni a későbbi implementáció miatt, hogy mivel a jel egy rezonátorcsatornán áthaladva $c_{k,n}$-nel és annak konjugáltjával is megszorzódik, más nem nulla képzetes részű komplex számmal viszont nem, a csatorna kimenetén lévő számnak valósnak kell lennie.

3.3.2 Az egységesen megtervezett megfigyelő használata

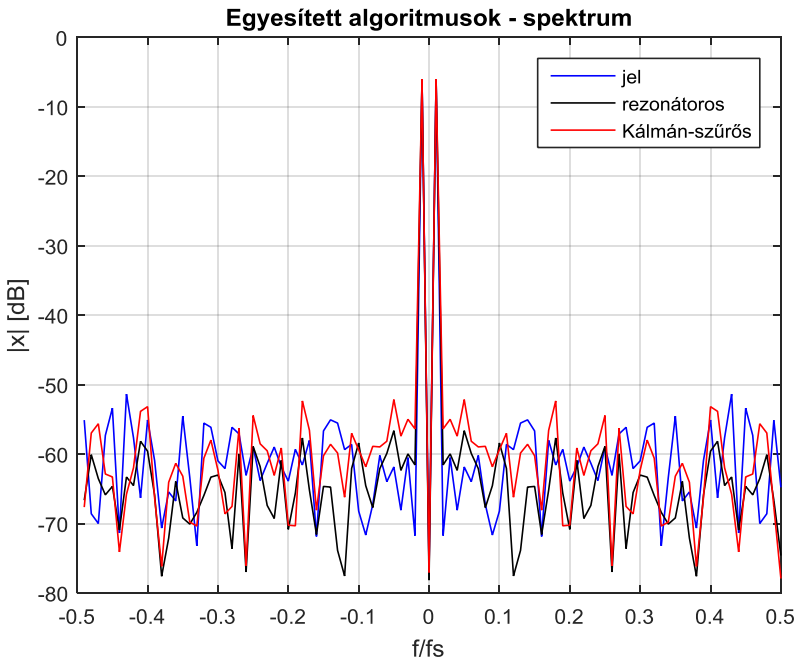
Az előző fejezetben kiderült, hogy a két egyszerűsített algoritmus megfelelő paraméterválasztással megegyezik, ezért a továbbiakban együtt vizsgálom őket. Vizsgálójelem továbbra is az $f_1 = 0{,}01$ normalizált frekvenciájú, egységnyi amplitúdójú, -40 dB fehér zajjal terhelt szinuszjel. *R*-et továbbra is 0,01-nek választom.

Két esetet vizsgálok, az egyik, amikor a $\boldsymbol{C}_n^H$-t szorzó skalárok (α és $\frac{\varepsilon}{\varepsilon N+R}$), nem egyeznek meg. Egy ilyen esetet mutat be a 3.27. ábra és a 3.28. ábra.

Amint az látható, mindkét algoritmus működik, egy periódus után mindkettő elkezdett beállni a jelre. A rezonátoros lassabb volt, de ez érthető is, hiszen $\boldsymbol{G}_n$ skalárszorzója kisebb volt, mint a Kálmán-erősítésé, viszont cserébe valamelyest kevésbé zajos a spektruma. Az alapharmonikus becsült Fourier-együtthatói is kellően pontosak.

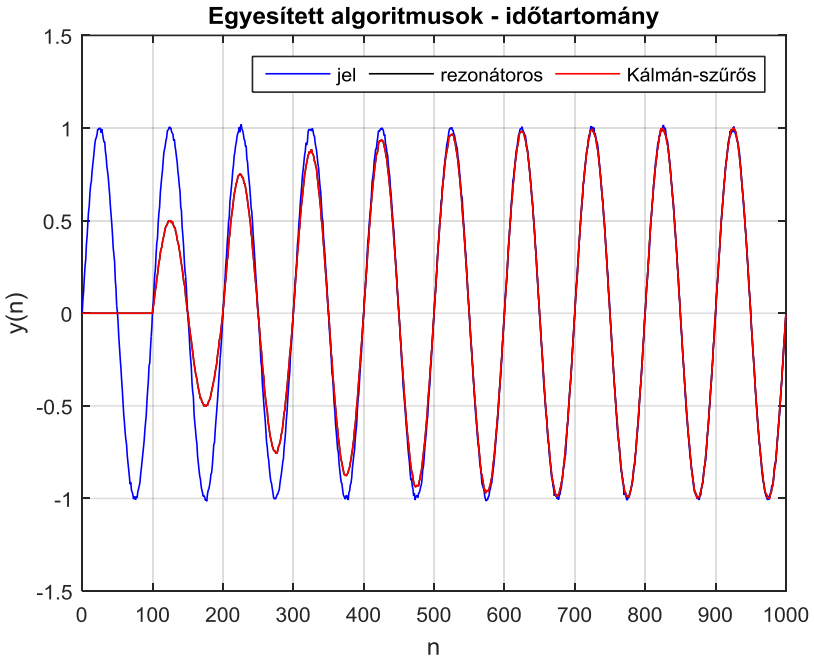


3.27. ábra: *N* = 100, α = 0,5, ε = 0,1

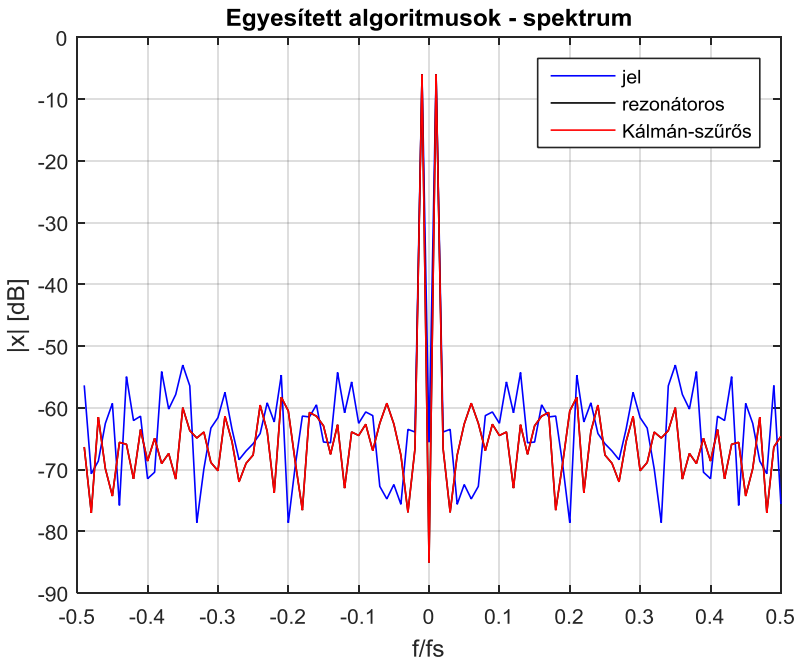


3.28. ábra: N = 100, α = 0,5, ε = 0,1

A második esetben már úgy állítottam be a paramétereket, hogy a skalárszorzók egyezzenek. Ekkor mind időtartományban, mind a spektrumon teljesen fedi egyik a másikat (ld.3.29. ábra és 3.30. ábra).

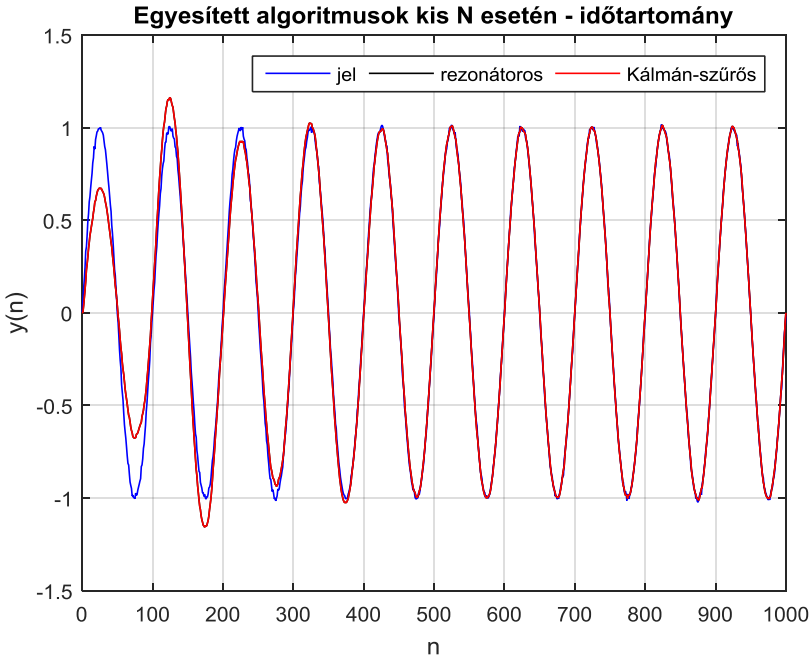


3.29. ábra: Egyesített algoritmusok teljes spektrum vizsgálatánál, α = 0,5



3.30. ábra: Egyesített algoritmusok teljes spektrum vizsgálatánál, α = 0,5

Még egy dologról érdemes szót ejteni. Az eddigi vizsgálatok $N = N_{max} = 100$ választással folytak. Megnéztem mi történik ha N értékét csökkentjük. Kismértékű csökkentésnél még nem látszik számottevő változás, azonban jelentősebb változtatásnál már az időtartományban megjelenik egy kismértékű hullámzás az amplitúdóban a beállás során. Ezt illusztrálja $N = 11$ esetében a 3.31. ábra.



3.31. ábra: Egyesített algoritmusok, N = 11

4 Rekurzív spektrumbecslés beágyazott szoftverterve

4.1 A felhasznált beágyazott környezet

Mint azt a korábbiakban láttuk, egy viszonylag kis számításigényű, egyszerű algoritmussal megvalósítható a rekurzív spektrumszámítás. A továbbiakban tehát a cél ennek egy beágyazott környezetben történő megvalósítása.

Első lépésként szükségem volt egy, a feladatra alkalmas fejlesztőkártyára és fejlesztőkörnyezetre, amelyekkel a feladat megoldható. Röviden ismertetem, milyen képességek voltak fontosak az eszközök részéről, majd a továbbiakban a teljesség igénye nélkül, a lényeges paraméterekre koncentrálva bemutatom a kiválasztott fejlesztőkártyát és fejlesztőkörnyezetet.

Kezdjük a hardver tulajdonságaival! Magától adódik, hogy a számítási teljesítmény fontos, hiszen az algoritmus műveleteit el kell végezni, és tekintve, hogy ezekkel két minta beérkezése között végezni kell, a megfelelő utasításkészlet és a maximális órajel fontos szempont. A számításokat kellő pontossággal is kell elvégezni, ezért a szóhosszúság is felmerül kérdésként. Nyilván bonyolult és lassú algoritmusokkal egy 8 vagy 16 bites processzoron is elérhető a kellően nagy felbontású számábrázolás, ezt azonban lehetőség szerint érdemes lenne elkerülni, és ennél nagyobb szóhosszúságú mikrovezérlőt választani. További diszkusszió tárgya, hogy szükség van-e lebegőpontos támogatásra. Természetesen előny, ha van, ugyanakkor az algoritmus egyszerűsége miatt véleményem szerint nélkülözhető, és fixpontos számábrázolással is megoldható a feladat.

Tekintve, hogy a beágyazott rendszernek a külvilággal, ahonnan a mintavett jelet kapja, kapcsolatot kell tartania, néhány periféria megléte is elengedhetetlen. Mivel analóg jelet mintavételezünk, természetesen szükség van egy ADC-ra, és mivel a becsült- vagy a hibajelet ki is szeretnénk vezetni, egy digitális-analóg átalakítóra (DAC) is. A mintavételezés pontos időzítést igényel, ezért ezt is érdemes hardverrel és nem szoftverrel intézni. Legalább egy időzítő (Timer) így mindenképpen szükséges. Az is előnyös, ha a fejlesztőkártyán van néhány nyomógomb és LED, amikkel futási időben tudunk kommunikálni a rendszerrel.

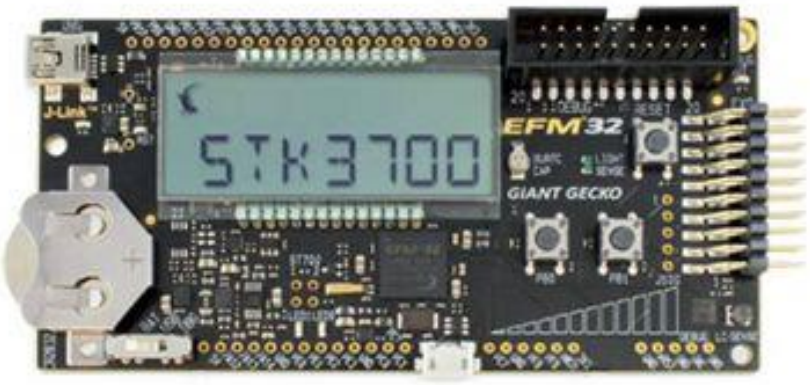
A fejlesztőkörnyezettel szemben is vannak elvárásaink olyan általános tulajdonságok mellett, mint a könnyen kezelhető, ergonomikus felhasználói felület.

Elsőként például jó lenne, ha rendelkezne olyan fordítóval, amely a C nyelvű programozást lehetővé teszi. A fejlesztés könnyítése miatt fontos a debug lehetőségek választéka is, előnyös például, ha biztosítva van a változók futás közbeni nyomon követése, a program debuggerből történő megállításának lehetősége (természetesen ezek hardveres támogatást is igényelnek). Nyilván mindezek miatt célszerű elterjedt, népszerű architektúrájú mikrokontrollert választani. Az is segítség lenne, ha a perifériák beállításához és használatához lennének előre megírt függvények szoftvertámogatás formájában.

A következő két alfejezetben bemutatom a választott eszközöket, mindkettőnél figyelve arra, mennyire elégítik ki az elvárásokat.

4.1.1 Fejlesztőkártya

A választás a Silicon Labs EFM32GG-STK3700 Giant Gecko Starter Kit-jére[8] esett.



4.1. ábra: EFM32GG-STK3700 Giant Gecko Starter Kit képe[8]

Ez egy viszonylag olcsó (a dolgozat írásakor ára 29,99 $ a Silicon Labs honlapján), könnyen elérhető, kezdőknek is szánt fejlesztőkártya, amely tartalmaz egy EFM32GG990F1024 típusú mikrokontrollert, számos szenzort és elektronikus részegységet.

Az említett mikrovezérlő az EFM32 családhoz tartozik, amely magja ARM Cortex-M3-as processzor[9]. Ez 32 bites, fixpontos műveletvégző egységgel rendelkezik, hardveres osztóval és egy órajel-ciklus alatt végző szorzóval, lebegőpontos támogatást viszont nem ad. Ez utóbbi nyilván nem előny, azonban nem is volt fontos szempont, hogy külön hardveres lebegőpontos támogatást adjon a rendszer. A szóhosszúság viszont

kellően széles. Az utasításkészletben is találhatunk DSP támogató elemeket. Az egyik a Multiply with Accumulate (`MLA`) utasítás, amely összevon egy szorzást és egy összeadást. A másik, ami esetünkben hasznos, az utasításszintű logikai jobbra és balra shiftelés lehetősége (`LSR`, `LSL`). Általánosságban az ARM Cortex-M sorozat minden tagjának, közte az M3-asnak, az architektúrája széles körben ismert és használt az iparban, ezért számos fejlesztőkörnyezet támogatja.

A mikrokontroller számos perifériát is tartalmaz a processzor mellett[10], most csak a számunkra fontosakat emelem ki. Egyrészt van benne 4 db 16 bites Timer, ami mint láttuk, elengedhetetlen esetünkben. A chip rendelkezik még 12 bites szukcesszív approximációs (SAR) ADC-vel, amely másodpercenként maximum 1 millió mintát képes digitalizálni, és egy szintén 12 bites DAC-vel is, amely másodpercenként maximum 500 ezer mintát tud kiadni. Ezeket a sebességeket nem fogjuk tudni kihasználni, így ezek bőven elégségesek számunkra. Az előző pontban említett nélkülözhetetlen perifériák tehát rendelkezésre állnak.

Az órajelet tekintve számos beállítás közül lehet választani, esetünkben nyilván a legnagyobb frekvenciájút érdemes választani, amely kristállyal generált 48 MHz.

A processzormag tartalmaz még 1 MB Flash-t és 128 kB RAM memóriát is.

Még néhány dolgot kiemelnék a panelről, ami nem a mikrokontrollerrel kapcsolatos[11]. Egyrészt van rajta két felhasználói nyomógomb és LED is, ezek szükség esetén használhatóak. Másrészt, ami számunkra még fontosabb, a kártyán van egy board controller, ami felelős a debugger kezeléséért. Ennek segítségével lehet a futtatható kódot a mikrovezérlőre tölteni, a program futásába beavatkozni, a változókat nyomon követni. Mindez nagyban megkönnyíti a szoftverfejlesztés során a hibakeresést. Ez a board controller UART-on kommunikál a mikrokontrollerrel és USB-n keresztül a PC-s fejlesztőkörnyezettel. A panel tápellátása bár akár elemmel is megoldható, esetünkben szintén ezen az USB kapcsolaton keresztül történik.

Mindezekből látható tehát, hogy a hardver teljesíti a követelményeinket, ugyanakkor még távol áll egy DSP-től. Így a panel nem nevezhető sem erőforrásokban szegénynek, sem jelfeldolgozási szempontból kifejezetten gazdagnak.

4.1.2 Fejlesztőkörnyezet

A Silicon Labs saját fejlesztőkörnyezetet is kínál a termékein való szoftverfejlesztéshez, amit Eclipse[12] alapokra építenek. Ez a Simplicity Studio[13], amelynek én az aktuális, 4-es verzióját használtam munkám során.

Ez a program lehetővé teszi a C nyelvű projekt alapú szoftverfejlesztést. Mivel a Silicon Labs saját fejlesztőkártyáihoz ajánlja a fejlesztőkörnyezetet, számos támogatást ad azok használatához. A telepítés során olyan csomag telepítését ajánlja fel, amelyik a meglévő eszköz programozását könnyíti meg. A támogatás része többek között egy hardver konfigurátor, amelyben a projekthez szükséges perifériákat lehet nagyon egyszerűen ki- és bekapcsolni, paramétereiket beállítani. Számos beépített könyvtárt is találhatunk, amelyekben előre megírt függvények vannak bonyolult, de gyakran előforduló számításokhoz, perifériaműveletekhez. Ezek közül most csak egyet emelnék ki, amely kifejezetten hasznos lesz számomra: a DSP_Lib könyvtárat. Ebben bonyolult, jelfeldolgozási feladatokban gyakori műveletekre vannak optimalizált függvények, például konvolúcióra, szűrésekre, stb. Én kétfajta függvényt fogok innen használni: a gyors szinusz-, koszinusz számítóakat, és a lebegőpontos-fixpontos számok átalakítását végzőket. Ezeket majd a kód ismertetésénél ismét kiemelem.

A fejlesztőkörnyezet fejlett debuggerrel is rendelkezik, ami lehetővé teszi a változók értékének nyomon követését, a futás felfüggesztését, a kódban való lépkedést, stb.

Mindezekből látható, hogy ez a fejlesztőkörnyezet lehetővé teszi a mikrokontroller kényelmes programozását, így emellett döntöttem.

4.2 Az elkészült program

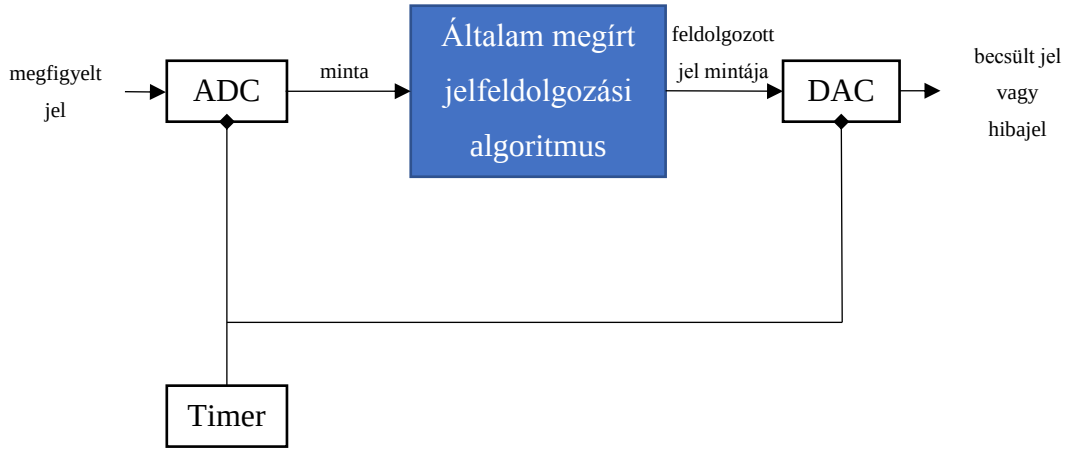
4.2.1 Keretrendszer

Munkám során támogatásul megkaptam egy a tanszéken oktatási célokra használt demonstrációs keretrendszert a fejlesztéshez, amit felhasználtam. Röviden bemutatom ez miből áll, meddig terjednek a funkciói, és hogyan illesztettem bele ebbe a rekurzív spektrumbecslő algoritmusomat.

Ez a keretrendszer lényegében két részből áll. Az első a mikrokontroller inicializálása. Ennek részeként beállítja az órajelet a maximális 48 MHz-re, inicializálja

az ADC-t és a DAC-t, valamint a mintavételezést ütemező Timert. Végül engedélyezi a Timer megszakítás (interrupt, IT) kérését.

A másik rész a Timer IT-kezelő függvénye. Ez a mintavételezés ütemében jelentkező Timer IT során hívódik meg. Kiírja a DAC-re az általunk az előző meghívódása során beállított értéket, és beolvassa az ADC-ről az új 12 bites mintát egész szám formátumban. Ezzel az értékkel meghív egy általunk választott függvényt, amely visszatérési értékét eltárolja egy változóban, amelyet a következő meghíváskor a DAC-ra kiír. Amennyiben az egyik nyomógomb le van nyomva, akkor ezt még módosítja, hogy ne a feldolgozott értéket, hanem az eredeti, ADC-ről kapott mintát írja majd ki (lényegében oszcilloszkóp funkció). Végezetül törli az IT-kérést.



4.2. ábra: A keretrendszer által nyújtott szolgáltatások a jelfeldolgozáshoz

A keretrendszer tehát az inicializálások után megvalósít egy mintavételezést, viszont a mintákon önmagában nem végez semmilyen feldolgozást, csupán meghív velük egy függvényt, amely ezt majd intézi. Az én munkám pedig ennek a jelfeldolgozási folyamatnak az implementálása volt, amit a következő fejezetekben mutatok be.

4.2.2 A szoftverterv összefoglalása

4.2.2.1 Lebegőpontos és fixpontos változók

A szoftverterv ismertetéséhez először vizsgáljuk meg a számábrázolás problémakörét. Mint láttuk, a processzorunk 32 bites fixpontos aritmetikával rendelkezik. Ez azonban nem jelenti azt, hogy ne lehetne akár lebegőpontos, egyszeres pontosságú

`float` típusú változókat használni a programban. A fordító szoftveresen megoldja, hogy ilyen számokkal is tudjunk műveleteket végezni, bár természetesen így a kód sokkal lassabb lesz, hiszen a szükséges konverziók plusz műveleteket igényelnek a futás során. Előnye ennek viszont az, hogy a programozás közben úgy kezelhetjük a lebegőpontos számokat, mintha a processzor tudná kezelni őket, nem kell külön ügyelnünk rájuk.

A feladat tehát megoldható lebegőpontos változók használatával is, és bizonyos részeken ez nem is okoz problémát. Például az inicializálások közben, amikor N_{max} értékét számítjuk (folyamatábrát ld. később), a lassúság nem okoz gondot, hiszen ekkor még nincs mintavételezés, nem kell sietni, hogy a következő mintáig készen legyünk. Igaz ugyan, hogy a feléledési idő nő, de ez a jelen kísérleti jellegű feladatban elhanyagolható szempont. Ezeknél a részeknél így használtam is `float` típusú változókat.

A jelfeldolgozó rekurzív algoritmusnál ugyanakkor kritikus szempont a gyorsaság, így itt már mindenképpen érdemes a processzoron hatékonyan futtatható fixpontos aritmetikát használni a kódolás során is, bár ez komplikáltabb.

Ezen a ponton felmerül a kérdés, hogy milyen fixpontos számábrázolási formátumot érdemes használni. Mindenképpen érdemes kettes komplemensben ábrázolni a számokat, hiszen így az összeadás-kivonás egységesen kezelhető, és a túlcsordulás is könnyen figyelhető. Általánosságban Q*m*.*n* formátumról beszélhetünk, ahol *m* az egészrész-, *n* a törtrész bitek száma. Ilyenkor[14]:

- az ábrázolható tartomány: $[(-2^{m-1}); 2^{m-1} - 2^{-n}]$
- a felbontás: 2^{-n}

Nyilván rengeteg féle megoldás lehetséges, azonban a DSP_Lib könyvtár függvényei csak két típust támogatnak ezek közül: ezek a Q1.15 és Q1.31, amik esetünkben is megfelelőek, így ez nem gond. A kettő közül végül a Q1.15-öt választottam (a C nyelvű kódban ennek típusa `q15_t`, amely a valóságban egy integer, csupán a jelölés szemléletessége miatt lehet így is használni), mivel ez a 32 bites processzoron egyszerűbben kezelhető, és tekintve, hogy a minták 12 bitesek, a felbontása is elégséges.

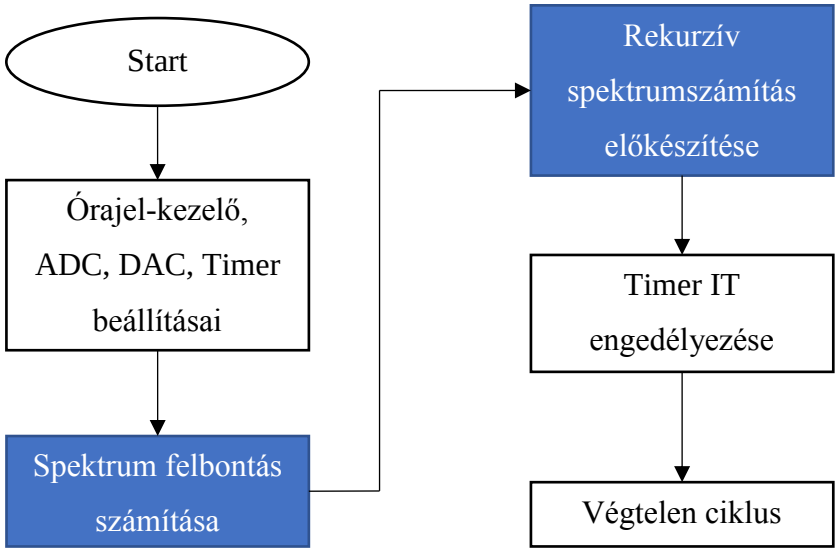
Mivel az összevetés tanulságos lehet, a jelfeldolgozási programrészt megírtam lebegőpontos változók használatával és anélkül, kizárólag fixpontos változókkal is. Az előkészítő, inicializáló programrészekben mindkét esetben használtam lebegőpontos

változókat. A továbbiakban e két változatra lebegőpontos és fixpontos változatként hivatkozom.

4.2.2.2 A főprogram

A program ismertetését a főprogrammal, a `main` függvénnyel kezdem. Tekintsük rögtön annak a lényegi részeit bemutató folyamatábráját (4.3. ábra)! Az ábrán kék háttérrel szerepelnek az általam implementált részek, és fehér háttérrel a keretrendszer alkotóelemei. Mivel a rekurzív spektrumbecslés szempontjából lényegesebbek, az előbbiek ismertetésére szorítkozok.

A program globális változóként kapja meg a kívánt N és α értékét, és az ismert f_{alap}-ot. A keretrendszerben a mintavételi frekvencia is beállításra kerül, innen ez is ismert. A „Spektrum felbontás számítása” rész ezekből kiszámítja N_{max} értékét, továbbá azt is, hogy van-e a vizsgálni kívánt frekvenciák közt $\frac{f_s}{2}$ komponens, ha igen, ezt egy flaggel (`fs2_flag`) jelzi a program későbbi részei számára. Ha a megadott N nagyobb a megengedett maximálisnál, akkor értékét N_{max}-ra csökkenti, ha pedig értéke páros $\frac{f_s}{2}$ komponens nélkül, úgy 1-el csökkenti, majd kiszámítja f_1 értékét (4) szerint. Mivel ez utóbbi változó lebegőpontos típusú, és a továbbiakban szükséges lesz, a fixpontos változatban ezt még konvertálja `q15_t` típusúra a DSP_Lib `arm_float_to_q15` függvényével



4.3. ábra: Főprogram folyamatábrája

Ezután ebben a részben a program feltölt egy `normalized_frequencies` nevű tömböt *N* alapján a valós frekvenciák mintavételi frekvenciára normált értékeivel. Ezen a ponton van egy fontos egyszerűsítés az algoritmushoz. Ahogyan a periodikus jelek Fourier-soráról írtam, a $\pm f_k$ frekvenciák $\hat{\boldsymbol{x}}$ együtthatói komplex konjugáltak, így azokat a megfelelő $c_{k,n}$-nekkel megszorozva, amelyek szintén egymás konjugáltjai, a kapott valós számok (ld. 3.3.1 fejezet) megegyeznek. Mivel mindkét frekvenciához tartozó rezonátorcsatorna ezzel az értékkel járul hozzá a valós $\hat{y}_n$ becsült jelhez, felesleges mindkétszer kiszámolni, elég csak mondjuk a pozitív f_k-hoz tartozót, majd azt kettővel megszorozni. Vagyis:

$$\hat{x}_n(f_k) = conj\big(\hat{x}_n(-f_k)\big) \tag{58}$$

$$c_n(f_k) = conj\big(c_n(-f_k)\big) \tag{59}$$

Tehát:

$$\hat{x}_n(f_k) \cdot c_n(f_k) + \hat{x}_n(-f_k) \cdot c_n(-f_k) = 2 \cdot \Re\{\hat{x}_n(f_k) \cdot c_n(f_k)\} \tag{60}$$

Így a programban a negatív frekvenciák elhagyásával számítási teljesítmény spórolható. Ezen okokból már a `normalized_frequencies` tömbbe sem kerülnek negatív frekvenciák.

A főprogram másik általam írt része, a „Rekurzív spektrumszámítás előkészítése” kevesebb érdekességet rejt. Ez a rekurzív algoritmushoz beállít néhány változót kezdeti értékére:

- Az $\hat{\boldsymbol{x}}$-nak megfelelő `x_kalap` tömb elemeit nullára.
- A $\boldsymbol{C}_n$vektornak megfelelő `C` tömb elemeit a valós $(1 + 0j)$-re (ill. a fixpontos változatban, mivel 1 már kívül esik az ábrázolható tartományon, $1\text{-}2^{-15}$ értékűre, ami nagyon jól közelíti 1-et). Mivel komplex számok, a `C` és az `x_kalap` tömb elemeinek típusa is az általam definiált `Complex` struktúra az alábbi kód szerint:

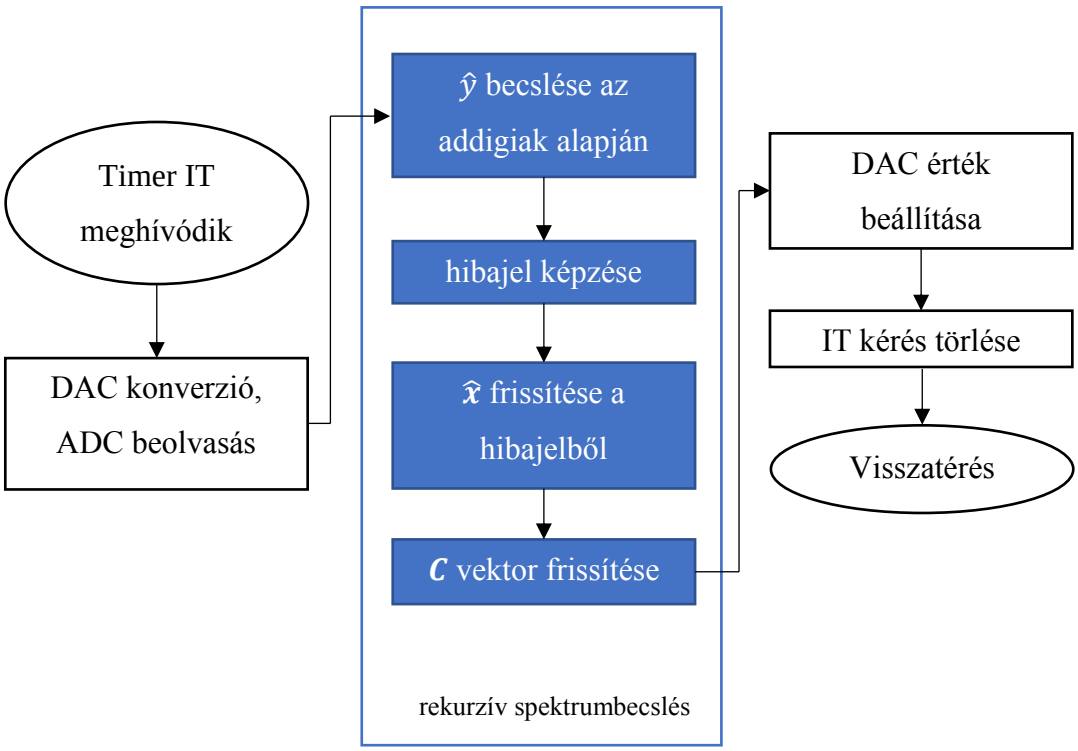
```
typedef struct complex {
     q15_t real;
     q15_t imaginary;
} Complex;
```

A lebegőpontos változatban `real` és `imaginary` típusa `float`.

- Egy `forgo_vektor` nevű tömb elemeit nullára. Az ebben a tömbben tárolt értékek fogják a komplex egységnyi hosszú $c_{k,n}$ körforgó vektor valós tengellyel bezárt szögét jelenteni, 2π-re normálva. Ennek majd $\boldsymbol{C}_n$ aktuális valós és képzetes részének számításánál lesz jelentősége.

4.2.2.3 Timer IT-kezelő

A Timer IT rutin a mintavételezés szerinti ütemben hívódik meg. Rögtön tekintsük is a folyamatábráját (4.4. ábra)! Itt szintén kékkel vannak kiemelve az általam írt részek.



4.4. ábra: Timer IT-kezelő folyamatábrája

Mint ahogyan az látható a rekurzív spektrumbecslés folyamata négy fő részből áll. Ezeket egy függvény fogja össze, a `calculate_xy_values(q15_t y)`, melynek fixpontos változatbeli kódját teljes egészében a Függelékben tüntetem fel. A kommentek ott jól jelölik az egyes részek kezdetét. A változónevek beszédesek, egyedül talán a `multiplier` szorul magyarázatra, ez az $\frac{\alpha}{N}$ szorzótényező.

Haladjunk végig sorban a kód részletein. Az első egység $\hat{y}$ becslése.

```
y_kalap = mul_complex(C[0],x_kalap[0]).real;
if(fs2_flag)
{
     for(i=0; i<N-1; i++)
          y_kalap += 2*mul_complex(C[i],x_kalap[i]).real;
     y_kalap += mul_complex(C[N-1],x_kalap[N-1]).real;
}
else
{
     for(i=0; i<N; i++)
          y_kalap += 2*mul_complex(C[i],x_kalap[i]).real;
}
```

Itt több dolgot érdemes megemlíteni. Egyrészt látható a korábban már említett egyszerűsítés, hogy a negatív frekvenciákhoz tartozó csatornák számítását elhagyom, és helyette a pozitívhoz tartozók eredményét megduplázom. Ez alól kivétel a DC komponens, és ha van, az $\frac{f_s}{2}$ komponens, ezeket a függvény külön is kezeli.

Megjegyzem, hogy ugyan a korábbiak alapján a mul_complex függvény mindig valós számmal kellene, hogy visszatérjen, numerikus okokból a képzetes rész nem feltétlenül lesz zéró. Azonban akár teljesül ez, akár nem, a visszatérési érték valós részének vétele egyrészt logikailag is helyes, másrészt eltüntethet egy ilyen numerikus hibát.

Még ugyanez az egység felhasznál egy másik függvényt, ez a `mul_complex`, amelyet két, az általam definiált `Complex` típusú szám összeszorzására írtam. Ennek kódját a következő fejezetben elemzem ki részletesen.

A második rész talán a legegyszerűbb, ebben csupán a hibajel kiszámítása történik, majd, mint az 3.1.5 fejezetben említett lehetőséggel élve annak a kiemelt $\frac{\alpha}{N}$-nel való megszorzása. Az ide tartozó forráskód:

```
err = y-y_kalap;
err_after_G = ((multiplier*err) >> 15);
```

Ezután a harmadik részben következik az $\hat{\boldsymbol{x}}$ vektor frissítése. Itt egy további egyszerűsítésre nyílik lehetőség annak nyomán, hogy az $\frac{\alpha}{N}$-el való szorzást kiemeltük. Ekkor ugyanis $\boldsymbol{G}_n = \boldsymbol{C}_n^T$, vagyis $g_{k,n} = conj(c_{k,n})$, és mivel a `Complex` típus algebrai alakjában tárolja a komplex számokat, $\boldsymbol{G}_n$ elemeinek előállítása csupán annyiból áll, hogy vesszük $\boldsymbol{C}_n$ elemeit a `C` tömbből és a képzetes részüket megnegáljuk. Így nagyon kevés számításigénnyel megoldható a feladat, és $\boldsymbol{G}_n$-t nem is kell külön tárolni, vagyis a memóriaigény is csökkent.

Az `x_kalap` tömb frissítése ezzel már könnyen megoldható:

```
for(i=0; i<N; i++)
{
	x_kalap[i].real = x_kalap[i].real + ((err_after_G*C[i].real) >>
15);
	x_kalap[i].imaginary = x_kalap[i].imaginary -
((err_after_G*C[i].imaginary) >> 15);    //mínusz a G = conj(C) miatt
}
```

Az $\frac{\alpha}{N}$-el súlyozott hibajelet megszorozzuk külön $\boldsymbol{G_n}$ valós és külön a képzetes részével, majd ezeket külön-külön hozzáadjuk `x_kalap` adott elemének régi értékének valós és képzetes részéhez.

Végül nézzük a negyedik részt, a `C` tömb frissítését. Ezt egy külön függvény intézi, a `C_calc()`. Ennek forráskódja:

```
void C_calc(void)
{
	int i;

	for(i=0; i<N; i++)
	{
		//körforgó vektor frissítése
		forgo_vektor[i] += normalized_frequencies[i];

		if(forgo_vektor[i] < 0)   //túlcsordulás történt
			forgo_vektor[i] = -(-0x8000 - forgo_vektor[i]);

		C[i].real = arm_cos_q15(forgo_vektor[i]);
		C[i].imaginary = arm_sin_q15(forgo_vektor[i]);
	}
}
```

Ennek a résznek a működése viszonylag egyszerű. Elsőként a normalizált frekvenciával, ami egy 0 és 0,5 közti szám, „továbbforgatja” a körforgó vektort, amely az adott $\boldsymbol{C}$ vektorbeli harmonikus fázishelyzetét reprezentálja. Amennyiben a `forgo_vektor` tömb adott eleme eléri az 1-et, túlcsordul, értéke negatív lesz, ebből könnyen lehet tudni, hogy a fazor megtett egy teljes kört. Ekkor a függvény a túlcsordulás mértéke alapján ismét leképezi a vektor állását a 0 és $1\text{-}2^{-15}$ értékekkel határolt tartományra, a következő körben pedig ismét a normalizált frekvenciával lépteti majd előre, és így tovább.

Mivel $c_{k,n}$ egységnyi abszolút értékű komplex szám, valós értéke a fázisának koszinusza, képzetes része pedig annak szinusza. Ezeket az `arm_cos_q15` és az `arm_sin_q15` függvények keresik elő gyorsan, amelyek a DSP_Lib könyvtárból származnak. Ezek egy 0 és $1\text{-}2^{-15}$ közötti `q15_t` paraméterrel dolgoznak, mint a

fázishelyzet 2π-re normált értékével. Mindezekből már látható, hogy milyen praktikus az időbeli változás `forgo_vektor`-beli reprezentálása.

A `calculate_xy_values` függvény végül az `y_kalap` becsült értékkel tér vissza, ami majd a DAC-ra kerül, ez azonban nem szükségszerű, a hibajellel is visszatérhet, és a mérések során volt is olyan alkalom, amikor így használtam.

Még egy lényegesebb dolgot említenék. Az ADC 0 és +3,3 V között tud mérni, ezért mintái 12 bites pozitív értékek, nekem azonban a Q1.15 számábrázolás miatt 15 bit törtrész áll rendelkezésre. Így, hogy finomabb felbontást nyerjek, és kihasználjam a teljes számábrázolási tartományt, a mintákat, mielőtt átadnám a függvénynek, 3 bittel balra shiftelem, majd emiatt természetesen a visszatérési értéket a DAC-nak visszashiftelem jobbra 3 bittel, ahogyan az alábbi kódsor mutatja:

```
DAC_data_out = (calculate_xy_values((q15_t)ADC_data_in << 3)) >> 3;
```

4.2.2.4 A fixpontos és lebegőpontos változatok eltérései

A legfőbb különbség demonstrálásához tekintsük a korábban már említett `mul_complex` függvény forráskódját!

```
Complex mul_complex(Complex one, Complex other)
{
     Complex result;
     result.real      =      ((one.real*other.real)      >>      15)      -
((one.imaginary*other.imaginary) >> 15);
     result.imaginary    =    ((one.real*other.imaginary)    >>    15)    +
((one.imaginary*other.real) >> 15);
     return result;
}
```

Bizonyítható, hogy Q1.15-ös számok ugyanúgy szorozhatóak, mint az egészek, de az eredményt 15 bittel jobbra kell shiftelni, ha az eredményt ismét Q1.15-ös számként szeretnénk. A függvényben `q15_t` típusú, azaz valójában integer számok szorzása történik, viszont emiatt minden szorzás részművelet után van egy 15 bites jobbra shiftelés a kódban. Természetesen összeadás és kivonás után erre nincs szükség. Most következzen ugyanez a függvény a lebegőpontos változatból:

```
Complex mul_complex(Complex one, Complex other)
{
     Complex result;
     result.real = one.real*other.real - one.imaginary*other.imaginary;
     result.imaginary         =         one.real*other.imaginary         +
one.imaginary*other.real;
     return result;
}
```

Miután megvizsgáljuk, látható, hogy teljesen ugyanaz, kivéve, hogy itt nincs szükség shiftelésekre. Itt azonban a háttérben `float` típusú változókkal kell dolgoznia a processszornak, ami mint a mérési eredményeken látni fogjuk, sokkal lassabb.

Logikai különbség még annyi van a két változat között, hogy a számábrázolás miatt a fixpontos verzióban az ADC mintái normálva vannak 0 és 1 közé, a lebegőpontosnál erre nincs szükség. Tekintve viszont, hogy a fixpontos számokon végzett műveletek során azok egész számokként vannak kezelve, a kódolás során ez viszonylag kevés különbséget eredményez, ezért is foglalkoztam végig a korábbiakban kizárólag a fixpontos forráskódjával.

Mindezeken kívül csupán olyan apróbb eltérések vannak, mint a DSP_Lib könyvtárból származó függvények neveinek eltérése, ezek azonban nem rejtenek újabb tanulságot, így nem mutatom be ezeket részletesen.

4.3 Mérési eredmények

4.3.1 A mérési eljárás

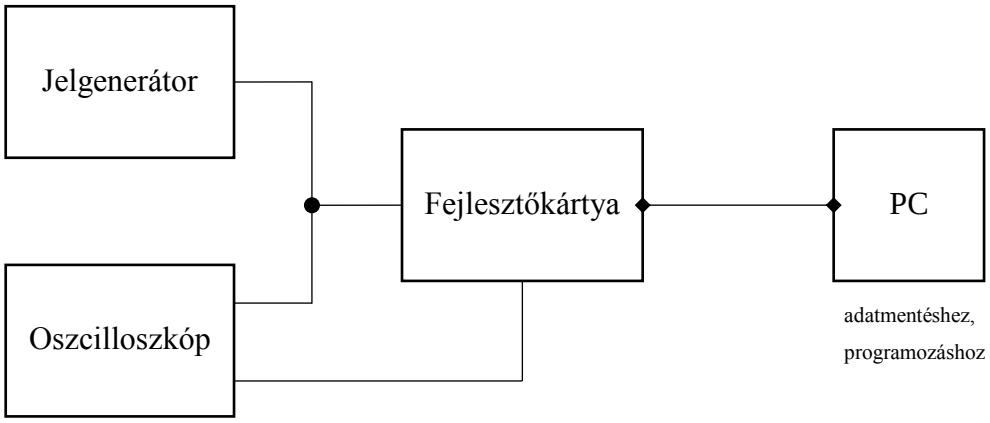
Röviden ismertetem a mérési eljárást.

A fejlesztőkártya ADC bemenetére egy jelgenerátor segítségével +2 V ofszettel rendelkező, 50 Hz-es 2,5 Vpp-s szinusz és háromszög jeleket kapcsoltam. Ezt a bemenetet és a DAC kimenetet, amelyre vagy a becsült-, vagy a hibajel kerül, egy oszcilloszkóp két csatornáján figyeltem. A következő fejezetekben lévő oszcilloszkóp ábrákon mindig a felső görbe tartozik a bemenethez, az alsó a kimenethez.

A mérések során mindig az $N = N_{max}$ beállítást használtam, hiszen az eddigiek alapján ez tűnt a legoptimálisabbnak, és ez adja a legtöbb információt is számunkra.

A mérés közben futtattam a mikrokontrolleren az előzőekben ismertetett jelfeldolgozó programot. A panel USB-n össze volt kapcsolva egy laptoppal, ezen a fejlesztőkörnyezet debuggerében tudtam elindítani és leállítani a futást.

A periodikus jelek spektrumának Fourier-együtthatóit úgy nyertem ki, hogy hagytam egy ideig futni az algoritmust, hogy a beállás biztosan megtörténjen, majd megállítottam a futást és az `x_kalap` tömb elemeit kimásoltam egy külön fájlba. Ezeket később Matlabban feldolgoztam, átskáláztam dBV-ra és ábrázoltam.



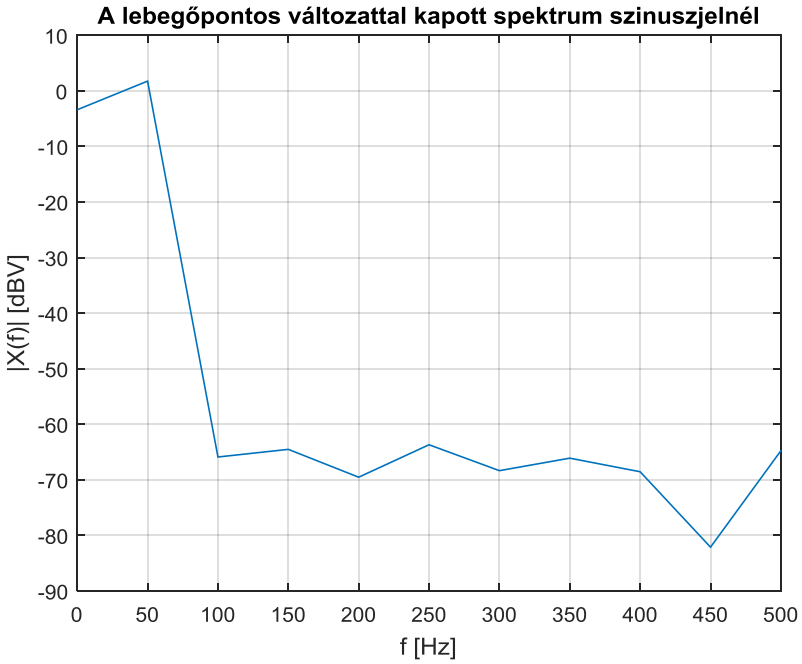
4.5. ábra: Mérési elrendezés sematikus rajza

4.3.2 A lebegőpontos változat eredményei

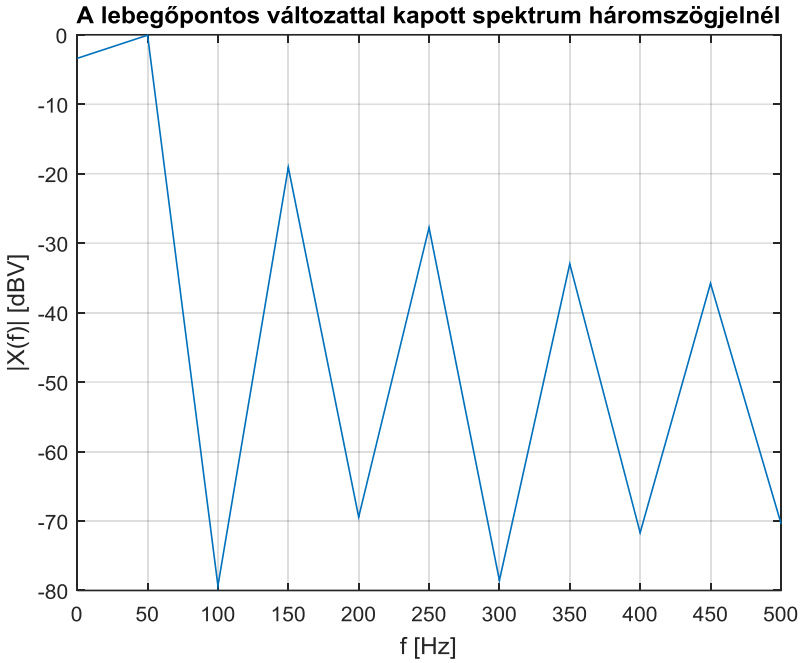
Kezdjük a lebegőpontos változattal! A várakozás az eddigiek alapján egyértelmű: működőképesnek kellene lennie, de csak kisebb mintavételi frekvencia esetén, mint a fixpontos esetében.

Vizsgálataim során maximálisan 1 kHz mintavételi frekvenciáig volt képes követni a jelet, efelett már nem volt elég ideje a számítások elvégzésére, így a becsült jel szétesett. A spektrumok viszont szépen kirajzolódnak szinusz és háromszögjel esetén is (4.6. ábra és 4.7. ábra), tehát, bár jóval kisebb spektrális felbontás érhető el vele, ez is használható.

Jól látszik, hogy a szinusznál egy kiemelkedő érték van az alapharmonikusnál, a háromszögnél pedig a páratlan harmonikusoknál vannak sorban kiemelkedő, de a frekvencia növekedésével csökkenő csúcsok, ahogyan az várható. Természetesen jelentős DC komponens is van mindkét esetben, hiszen a bemenő jel ofszetjével értem el, hogy az ADC-re ne kerülhessenek negatív feszültségek.

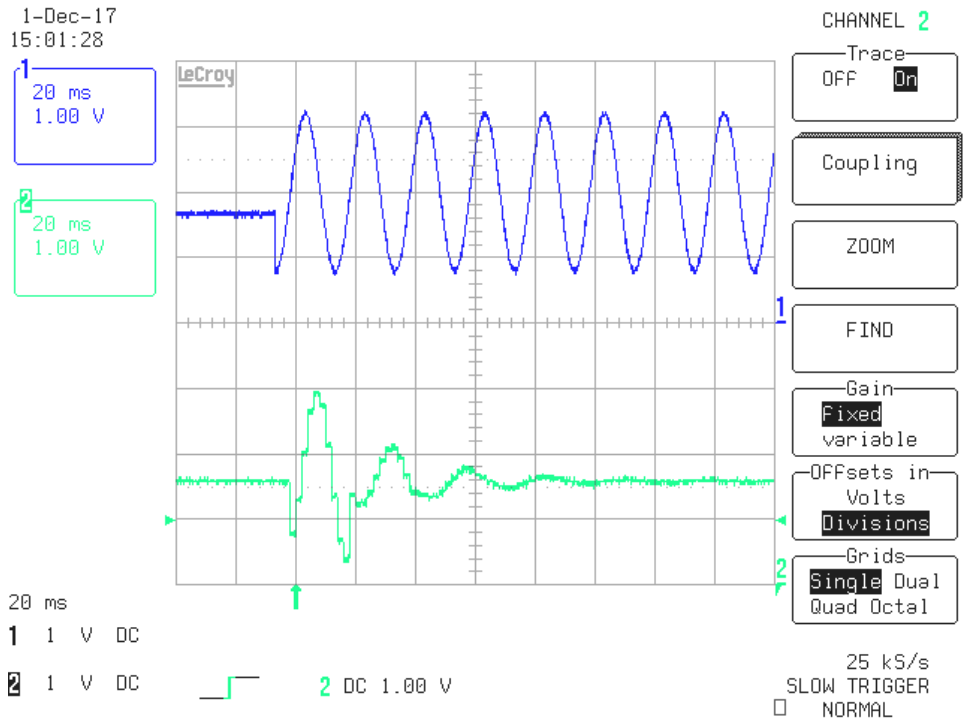


4.6. ábra: Szinuszjel spektruma a lebegőpontos változatban



4.7. ábra: Háromszögjel spektruma a lebegőpontos változatban

Bemutatom még a beállási folyamatot (4.8. ábra). Ehhez a hibajelet vezettem ki a DAC-ra. Látható, hogy a beállás rendben, hamar megtörténik ($\alpha = 1$ volt az ábra készítésekor). A görbe lépcsőssége amiatt van, hogy a jel frekvenciájának a f_s-re normált értéke viszonylag nagy szám (ekkor $f_s = 500$ Hz volt), így a DAC nulladrendű tartószervként való viselkedése szabad szemmel is jól látható.



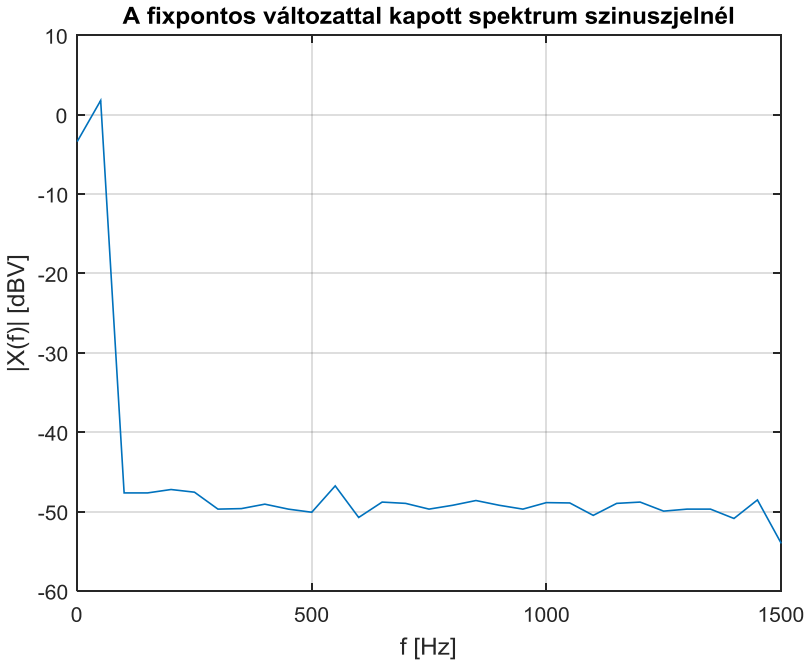
4.8. ábra: A megfigyelő beállása a lebegőpontos változatban, kis f_s mellett

4.3.3 A fixpontos változat eredményei

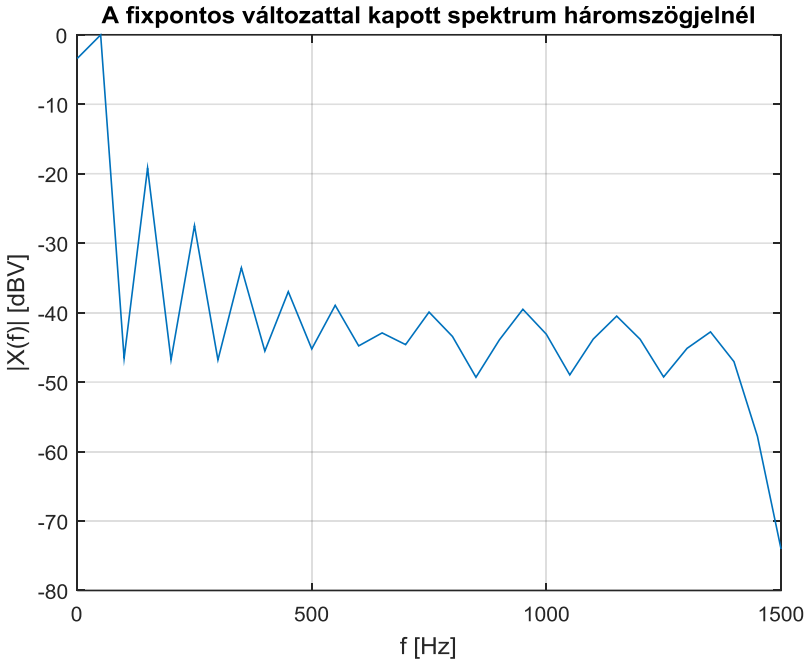
A fixpontos változatnál nyilván nagyobb mintavételi frekvenciában lehet reménykedni, és a mérések ezt vissza is igazolták.

Az algoritmus 3 kHz-es mintavételi frekvenciáig működött igazán jól, bár még efelett is alapvetően jónak tűnt a becsült jel, ekkor már megjelentek rajta periodikus zavarok, amik vagy numerikus okokból adódhattak vagy abból, hogy időnként már kevés lehetett a rendelkezésre álló idő két minta között a sok komponens kiszámítására.

A következő ábrákon lássuk, hogyan néz ki ezzel a változattal a szinusz és a háromszögjel spektruma!



4.9. ábra: Szinuszjel spektruma a fixpontos változatban



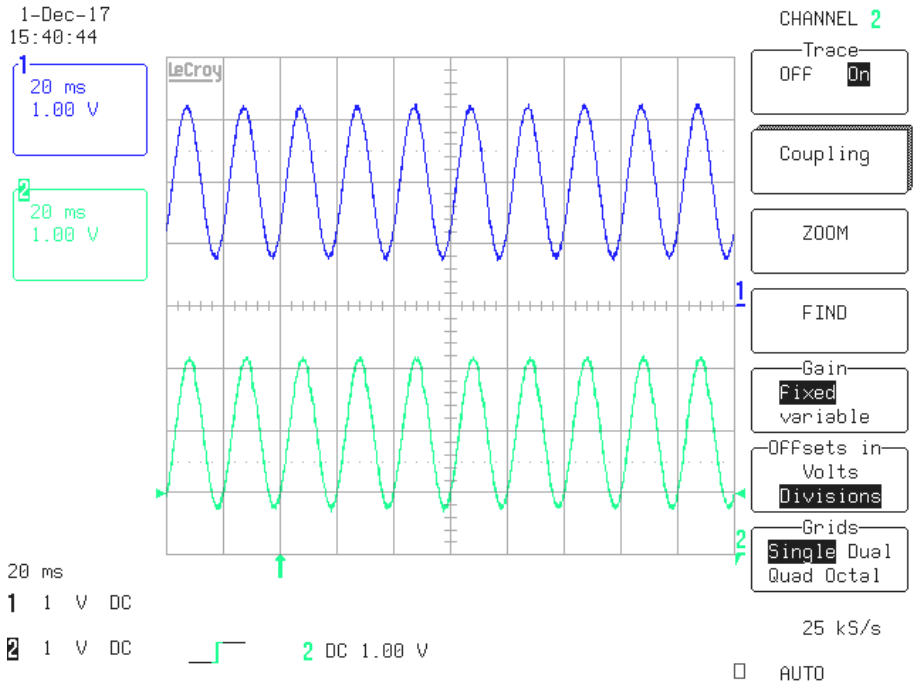
4.10. ábra: Háromszögjel spektruma a fixpontos változatban

A spektrumok mintázata itt is helyes, bár a háromszögjelé magasabb frekvenciákon nem ideális, ez azért lehet, mert itt már az együtthatók olyan kicsik, hogy inkább a zaj dominál.

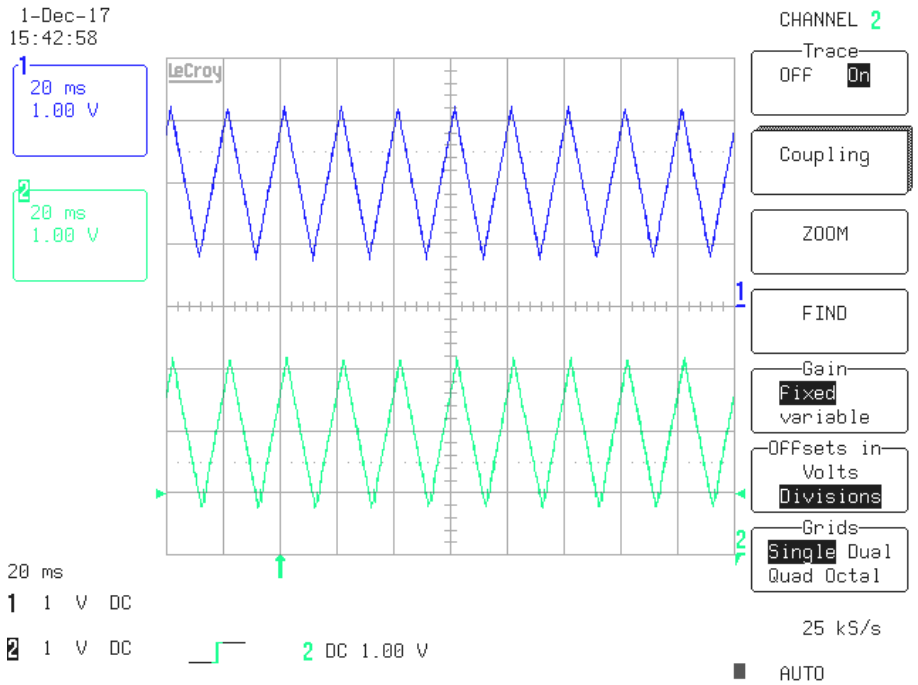
Ezen a ponton érdemes összevetni a fixpontos és a lebegőpontos változat precizitását. Az azonos típusú jelekhez tartozó spektrumok (4.6. ábra és 4.9. ábra, ill. 4.7.

ábra és 4.10. ábra) összevetésével két fő tanulság állapítható meg: egyrészt a jelhez tartozó komponenseket kb. ugyanannyinak mérték, tehát ebben nincs különbség. Másrészt viszont az alapzajszint, amit a szinuszok spektrumairól a felharmonikusok értékéből lehet megbecsülni, míg a lebegőpontosnál -60 és -70 dB között van, addig a fixpontosnál -40 és -50 dB között. A lebegőpontos jel-zaj viszonya tehát kb. 20 dB-el, vagyis egy nagyságrenddel nagyobb. Ez nem meglepő, hiszen az egyszeres pontosságú lebegőpontos számábrázolás jóval finomabb felbontást tesz lehetővé a számítások során. Cserébe viszont nyilván jóval kevesebb komponenst lehet így egyáltalán megmérni. Ennek a két jellemzőnek a viszonyát az alkalmazástól függően mérlegelni kell, amikor arról kell dönteni, melyik változatot használjuk.

Az oszcilloszkópon a becsült jel mindkét esetben szintén jól közelíti az eredetit. Ekkora mintavételi frekvenciánál már lépcsőzetesség sem látszik benne, szép sima görbéket láthatunk.

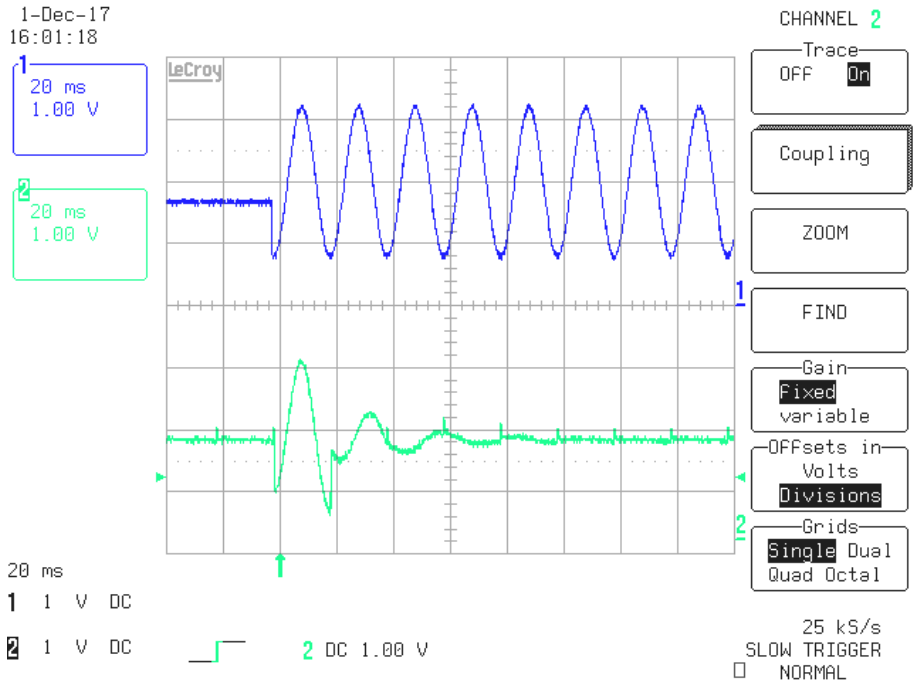


4.11. ábra: A megfigyelt szinusz és a becsült jel

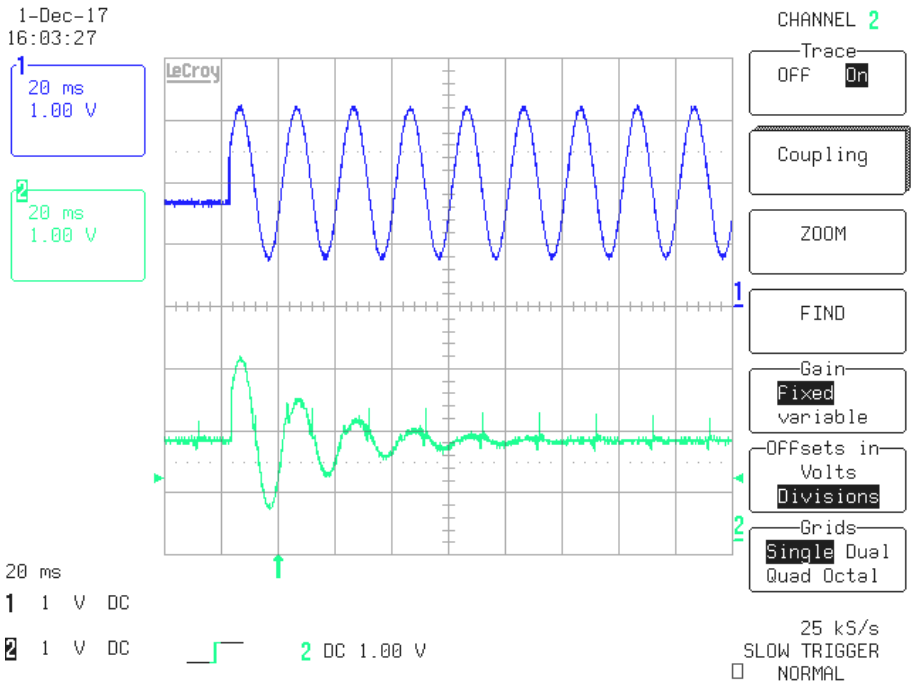


4.12. ábra: A megfigyelt háromszögjel és a becsült jel

Ennél a résznél még szemléltetném α állításának hatását. Két esetet néztem, az egyikben $\alpha = 1$, a másikban $\alpha = 0{,}5$. A 4.13. ábra és a 4.14. ábra jól mutatják, hogy a beállás tényleg szemmel láthatóan lassabb kisebb α esetén.



4.13. ábra: Beállás $\alpha = 1$ esetén



4.14. ábra: Beállás $\alpha = 0{,}5$ esetén

4.3.4 A futási idők összehasonlítása

Végezetül érdemes számszerűen is megvizsgálni, mennyi futási időt igényel a kétfajta algoritmus.

A keretrendszernek van még egy funkciója, melyet idáig nem említettem: megméri, hány órajel hosszan futott a `calculate_xy_values` függvény, és ezt egy változóban tárolja annak következő meghívásáig. Ennek segítségével két módon mértem meg a fixpontos és a lebegőpontos változat futási idejét: először a fordítóban kikapcsoltam az optimalizálást, másodszor pedig maximálisra állítottam. Természetesen ekkor olyan egyező paramétereket adtam meg mindkét változatnak, amikkel még képes működni: $f_s = 500$ Hz, $f_{alap} = 50$ Hz, $N = N_{max}$.

A lebegőpontos változat optimalizálás nélkül 24423 órajel-cikluson keresztül futott, és ez az érték optimalizált fordítással is csak 20674-re csökkent, tehát arányait tekintve nem jelentősen. Ebből látszik, hogy a lebegőpontos műveletvégzés nehézségeit nem lehet nagyon kioptimalizálni, az mindenképpen megmarad problémának.

Ezzel szemben a fixpontos optimalizálás nélkül is csak 3390 órajel-ciklust igényelt, azzal együtt pedig már csak 1131-et, ami arányait tekintve további jelentős javulás. A különbség tehát a várakozásoknak megfelelően számszerűleg is nagyon

jelentős a lebegőpontoshoz képest, egy nagyságrenddel kevesebb a számításigény fixpontos számábrázolás esetén.

Összefoglalva tehát a számítási idők:

számításigény [órajel]	fordítói optimalizáció nélkül	fordítói optimalizációval
fixpontos	3390	1131
lebegőpontos	24423	20674

Meg kell jegyezni, hogy a fent említett értékek nem teljesen állandóak, a függvény újbóli meghívásakor eltérhetnek. Az ingadozásuk azonban értékükhöz képest csekély, és itt csupán a nagyságrendi eltéréseket szerettem volna érzékeltetni, ezért az időbeli változásuktól eltekintettem.

5 Összefoglalás, kitekintés

Munkám során végigjártam egy jelfeldolgozási feladat megvalósításának lépéseit az elméleti háttér megismerésétől kezdve annak Matlabos szimulációin át a beágyazott rendszeren való programozásig, ill. a kész szoftver mérésekkel való vizsgálatáig.

A témával való foglalkozásom kezdetén megismerkedtem a valós értékű jelek spektrális leírásának állapotváltozós megközelítésével, majd az ehhez tervezett megfigyelő tervezésének alapjaival. Ezután tanulmányoztam a szakirodalomban fellelhető két különböző rekurzív spektrumbecslő eljárást, azok egyszerűsítési lehetőségeivel. Matlabos szimulációkkal megvizsgáltam ezek működését, közben figyelve egy beágyazott rendszeren való realizálhatóság szempontjaira. Ezután megtettem a javasolt egyszerűsítéseket és megmutattam, hogy ezek után a két algoritmus egyező alakra hozható. Miután meggyőződtem az egyszerűsített algoritmus működőképességéről, annak az implementálása mellett döntöttem.

A rekurzív spektrumbecslő eljárás realizálásához választottam egy fixpontos aritmetikájú processzort tartalmazó beágyazott rendszert, majd felhasználva egy készen kapott keretrendszert, lekódoltam C nyelven az algoritmust. A programozás során további, a hatékonyságot növelő egyszerűsítéseket, módosításokat tettem. Két változatot is készítettem, az egyik lebegőpontos változókkal, a másik csak fixpontos változókkal valósítja meg a rekurzív algoritmust. Végül a rendszert egy kísérleti elrendezésben végzett mérésekkel jellemeztem, igazoltam annak valós körülmények közti működőképességét, és összehasonlítottam különböző számábrázolási formátumok hatását.

Továbbfejlesztési lehetőségként mindenképpen érdemes lenne a szoftvert kipróbálni más beágyazott rendszereken is. A vizsgálatok során érdemes lenne a kisebb és a nagyobb teljesítményű rendszerek felé is elmozdulni, ezzel lehetővé téve ezek összehasonlítását. Kisebb rendszerek szempontjából például érdekes lenne egy 16 bites processzor, a nagyobb teljesítmény irányában pedig a lebegőpontos támogatás használata.

A vizsgálatokhoz használt fejlesztőkártyán érdekes lenne továbbá megfigyelni, milyen hatással lenne a pontosságra és a sebességre, ha Q1.15 helyett Q1.31 formátumú fixpontos változókat használnánk.

Irodalomjegyzék

[1] MATLAB, *fejlesztőkörnyezet*, https://www.mathworks.com/products/matlab.html (2017. november)

[2] Orosz György: *Adaptív jelfeldolgozó eljárások megvalósítása szenzorhálózatban*, diplomaterv, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest, 2006.

[3] Péceli Gábor: *A Common Structure for Recursive Discrete Transforms*, IEEE Transactions on Circuits and Systems, Vol. CAS-33, No. 10, pp. 1035-1036, October 1986.

[4] Orosz György: *Rezonátor alapú jelfeldolgozás szenzorhálózatokban*, doktori disszertáció, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest, 2012.

[5] Robert R. Bitmead, Ah Chung Tsoi, Philip J. Parker: *A Kalman Filtering Approach to Short-Time Fourier Analysis*, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-34, No. 6, pp. 1493-1501, December 1986.

[6] Fodor György: *Hálózatok és rendszerek*, egyetemi jegyzet, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest, 2014.

[7] Gene H. Hostetter: *Recursive Discrete Fourier Transformation*, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-28, No. 2, pp. 184-190, April 1980.

[8] Silicon Labs: *EFM32 Giant Gecko Starter Kit*, ismertető weblap, https://www.silabs.com/products/development-tools/mcu/32-bit/efm32-giant-gecko-starter-kit (December, 2017)

[9] Silicon Labs: *Cortex-M3 Reference Manual – EFM32 Microcontroller Family*, February 4th, 2011, https://www.silabs.com/documents/public/reference-manuals/EFM32-Cortex-M3-RM.pdf

[10] Silicon Labs: *EFM32GG990 Datasheet*, March 21st, 2016, https://www.silabs.com/documents/public/data-sheets/EFM32GG990.pdf

[11] Silicon Labs: *User Manual – Starter Kit EFM32GG-STK3700*, October 10th, 2013, https://www.silabs.com/documents/public/user-guides/efm32gg-stk3700-ug.pdf

[12] Eclipse, *fejlesztőkörnyezet*, http://www.eclipse.org/ (December, 2017)

[13] Simplicity Studio, *fejlesztőkörnyezet*, https://www.silabs.com/products/development-tools/software/simplicity-studio (December, 2017)

[14] Wikipedia: *Q (number format)*, https://en.wikipedia.org/wiki/Q_(number_format), (December 4th, 2017)

[15] Péceli Gábor: *Valós idejű jelkiértékelés mérési eljárásokban*, doktori értekezés, Budapesti Műszaki Egyetem, Budapest, 1988.

[16] Wikipedia: *Single-precision floating-point format*, https://en.wikipedia.org/wiki/Single-precision_floating-point_format, (December 7th, 2017)

Függelék

A spektrális megfigyelőt megvalósító jelfeldolgozási kódrészlet a következő:

```
q15_t calculate_xy_values(q15_t y)
{
	int i;
	q15_t y_kalap, err, err_after_G;

	//y_kalap számítása az eggyel korábbi mintából számított x_kalap és C-
ből
	y_kalap = mul_complex(C[0],x_kalap[0]).real;
	if(fs2_flag)
	{
		for(i=0; i<N-1; i++)
			y_kalap += 2*mul_complex(C[i],x_kalap[i]).real;
		y_kalap += mul_complex(C[N-1],x_kalap[N-1]).real;
	}
	else
	{
		for(i=0; i<N; i++)
			y_kalap += 2*mul_complex(C[i],x_kalap[i]).real;
	}

	//hibajel képzése és szorzása alfa/N-el
	err = y-y_kalap;
	err_after_G = ((multiplier*err) >> 15);

	//x_kalap frissítése
	for(i=0; i<N; i++)
	{
		x_kalap[i].real = x_kalap[i].real + ((err_after_G*C[i].real) >>
15);
		x_kalap[i].imaginary	=	x_kalap[i].imaginary	-
((err_after_G*C[i].imaginary) >> 15);	//mínusz a G = conj(C) miatt
	}

	//C frissítése a következő ütemre
	C_calc();

	return y_kalap;
}
```