Tampereen teknillinen yliopisto. Signaalinkäsittelyn laitos.
Opetusmoniste 2011:1
Tampere University of Technology. Department of Signal Processing.
Lecture Notes 2011:1

Heikki Huttunen

Signaalinkäsittelyn sovellukset

TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Tampereen teknillinen yliopisto. Signaalinkäsittelyn laitos.
Opetusmoniste 2011:1
Tampere University of Technology. Department of Signal Processing.
Lecture Notes 2011:1

Heikki Huttunen

Signaalinkäsittelyn sovellukset

Tampereen teknillinen yliopisto. Signaalinkäsittelyn laitos
Tampere 2011

ISBN 978-952-15-2635-0 (nid.)
ISBN 978-952-15-2636-7 (PDF)
ISSN 1459-4609

Esipuhe

Käsillä oleva moniste on tarkoitettu opetusmateriaaliksi Tampereen teknillisen yliopiston signaalinkäsittelyn laitoksen kurssille "Signaalinkäsittelyn sovellukset". Materiaali on kehittynyt nykyiseen muotoonsa luennoidessani aiheesta Jyväskylän yliopistossa lukuvuonna 1995–1996, Tampereen yliopistossa lukuvuonna 1999–2000 ja Tampereen teknillisessä yliopistossa lukuvuosina 1999–2011. Monisteen edellinen versio ilmestyi vuonna 2010, ja siihen verrattuna tässä painoksessa kappaleiden järjestys on muuttunut ja kappale 6 on uusi. Lisäksi tähän versioon on lisätty harjoitustehtäviä sekä tehty lukuisia pienempiä korjauksia.

Kurssi ja sen materiaali ovat jatkoa kurssille "Signaalinkäsittelyn menetelmät", jonka pääpaino on lineaaristen järjestelmien teoriassa. Tässä monisteessa keskitytään enemmän sovelluksiin. Moniste esittelee seuraavat aihepiirit: näytteenottotaajuuden muuntelu, adaptiivinen suodatus, spektrin estimointi, oppivat järjestelmät, tilastollinen signaalinkäsittely, signaaliprosessorit ja äänisignaalin pakkaaminen, epälineaarinen suodatus. Aihepiirien valinnassa on pyritty luomaan katsaus tärkeimpiin nykyaikaisten sovellusten käyttämiin menetelmiin.

Monet harjoitusassistentit, opiskelijat ja kollegat ovat olleet avuksi monisteen laatimisessa. Erityisesti mainittakoon Jari Niemi, Antti Niemistö, Matti Nykter ja Tuomo Pirinen TTY:ltä sekä Prof. Hisakazu Kikuchi Niigatan yliopistosta Japanista.

Lisäinformaatiota löytyy kurssin kotisivulta, jonka osoite on

`http://www.cs.tut.fi/kurssit/SGN-1251/.`

Sivulta löytyy myös monisteen tuorein versio PDF-muodossa.

Kurssin "Signaalinkäsittelyn menetelmät" materiaali puolestaan löytyy osoitteesta

`http://www.cs.tut.fi/kurssit/SGN-1201/.`

Monistetta on käytetty opetusmateriaalina vastaavilla kursseilla Tampereen AMK:ssa, Kemi-Tornion AMK:ssa, Jyväskylän AMK:ssa, Oulun AMK:ssa, Kuopion yliopistossa sekä TTY:n koordinoimassa DI-muuntokoulutuksessa Kuopiossa. Pyynnöstä materiaalin käytölle voidaan myöntää lupa myös muualla. Tällöin voin tarvittaessa toimittaa myös luentokalvot sekä harjoituksissa käytettäviä Matlab-skriptejä.

Tampereella, 16. helmikuuta 2012,
Heikki Huttunen
heikki.huttunen@tut.fi

Sisältö

Esipuhe **i**

1 Äärellisen sananpituuden vaikutukset **1**
1.1 AD-muunnoksen kvantisointivirhe 2
1.2 Suodatuksen vaikutus kvantisointikohinaan 4
1.3 Kertoimien pyöristämisen vaikutus 7

2 Spektrin estimointi **13**
2.1 FFT-menetelmä ja autokorrelaatiomenetelmä 13
2.2 Spektrin resoluutio 16
2.3 Spektrin ikkunointi 20
2.3.1 Sivukeilan korkeus 20
2.3.2 Kuuden desibelin kaistanleveys 21
2.4 Ikkunoiden vertailua 22

3 Näytteenottotaajuuden muuntelu **27**
3.1 Desimointi eli näytteenottotaajuuden alentaminen 28
3.2 Näytteenottotaajuuden pienentäminen useassa vaiheessa 31
3.3 Interpolointi eli näytteenottotaajuuden nostaminen 32
3.4 Näytteenottotaajuuden muunnos rationaalikertoimella 34
3.5 Interpoloinnin käyttö D/A-muunnoksessa 35
3.5.1 Nollannen asteen pitopiiri 35
3.5.2 Kohinanmuokkaus 38

4 Adaptiivinen suodatus **47**
4.1 Kohinan poisto 47
4.2 Tuntemattoman järjestelmän mallinnus 48
4.3 Ekvalisointi 49
4.4 Ennustus 49
4.5 Suodatusongelman teoreettinen optimiratkaisu 50
4.6 Gradienttimenetelmä ja LMS-algoritmi 53
4.7 Esimerkkejä LMS-algoritmin toiminnasta 54
4.7.1 Häiriötaajuuden estimointi ja poisto 55
4.7.2 Suppenemistesti 59

5 Oppivat järjestelmät **65**
5.1 Lähimmän naapurin luokittelija 66
5.2 Lineaarinen erottelija 67
5.3 Tukivektorikoneet 70
5.4 Hermoverkot 73
5.5 Esimerkki: merkkien luokittelu 78

6 Tilastolliset menetelmät signaalinkäsittelyssä **87**
6.1 Pienimmän neliösumman menetelmä 89
6.2 Sovellusesimerkkejä 92

7 Audiodatan kompressio **99**
7.1 Kuuloaistin ominaisuuksia 100
7.1.1 Kuuloaistin anatomia 100
7.1.2 Kuuloaistin mallintaminen 100
7.1.3 Kuulokäyrä 101
7.1.4 Peittoilmiö 102
7.2 Peittoilmiön hyväksikäyttö audion koodauksessa 103
7.2.1 Taajuusalueiden erottelu 104
7.2.2 Kvantisointikohinan peittyminen ja bittien jakaminen eri taajuuksille 108

8 Digitaaliset signaaliprosessorit **115**
8.1 Circular buffering 116
8.2 Kiinteän vai liukuvan pilkun aritmetiikka? 118
8.3 C:llä vai konekielellä? 118

9 Epälineaarinen suodatus **125**
9.1 Robustisuus 129
9.2 Muutamia epälineaarisia suotimia 129
9.2.1 Mediaanisuodin 129
9.2.2 Painotettu mediaanisuodin 130
9.2.3 L-suotimet 131

Luku 1

Äärellisen sananpituuden vaikutukset

Tähänastinen suodinten suunnittelu on tehty oletuksella, että käytössä on ääretön sananpituus ja laskentatarkkuus. Jopa Matlabilla suunniteltaessa saatavien suodinten kertoimet on esitetty verrattain suurella tarkkuudella. Kun suotimet käytännössä toteutetaan, täytyy kuitenkin kiinnittää huomiota käytettävän lukuesityksen vaikutukseen suodinten todelliseen käyttäytymiseen. Jos suotimet toteutetaan jollain korkean tason ohjelmointikielellä (C, Pascal, Fortran, Matlab) ja liukulukuaritmetiikkaa tukevalla suorittimella, pyöristysvirheet eri vaiheissa ovat suhteellisen pieniä. Signaalinkäsittelyprosessorit eivät useimmiten laske liukuluvuilla, koska kiinteän pilkun aritmetiikan laskutoimitukset vaativat vähemmän resursseja (virtaa, tilaa, jäähdytystä, jne.) ja ne on halvempi toteuttaa. Tyypillinen signaalinkäsittelyprosessori toteuttaa laskutoimitukset kahdeksan, kahdentoista tai kuudentoista bitin tarkkuudella, joten pyöristysvirheisiin on syytä kiinnittää huomiota.

Digitaalisen IIR-suotimen tärkeimmät äärellisestä sananpituudesta johtuvat virhelähteet on lueteltu seuraavassa.

- Kvantisointivirhe, joka syntyy muunnettaessa sisään tuleva analoginen signaali digitaaliseksi signaaliksi, jonka esityksessä käytetään verraten pientä bittimäärää.

- IIR-suotimen kertoimien esitys äärellisellä bittimäärällä aiheuttaa muutoksia taajuusvasteessa ja stabiilisuusominaisuuksissa.

- Äärellisen sananpituuden seurauksena järjestelmän laskutoimituksissa saattaa tulla ylivuotoa, jolloin tulokset menettävät merkityksensä lähes täysin.

- Suodatettaessa tarvittavien kertolaskuoperaatioiden tulokset pyöristetään tai katkaistaan käytetyn sananpituuden mukaisesti.

Jos aritmetiikka on toteutettu *kahden komplementtiin* (two's complement) perustuen, niin ylivuototilanteessa esimerkiksi kahden suuren positiivisen luvun summaksi saadaan itseisarvoltaan suuri negatiivinen luku. Tämä luonnollisestikin sotkee koko suodatuksen, sillä tällainen suuri virhe säilyy ja kertautuu jatkossa IIR-suodinten rekursiivisen rakenteen vuoksi.

1.1 AD-muunnoksen kvantisointivirhe

Muunnettaessa analoginen signaali digitaaliseksi on käytettävissä ainoastaan äärellinen määrä bittejä kunkin signaalin arvon esittämiseen. Merkitään käytettävää bittimäärää (merkkibittiä lukuunottamatta) muuttujalla b ja tarkastellaan tapausta, jossa diskreetin signaalin kaikki lukuarvot on jaettu tasavälisesti välille $[-1, 1]$.

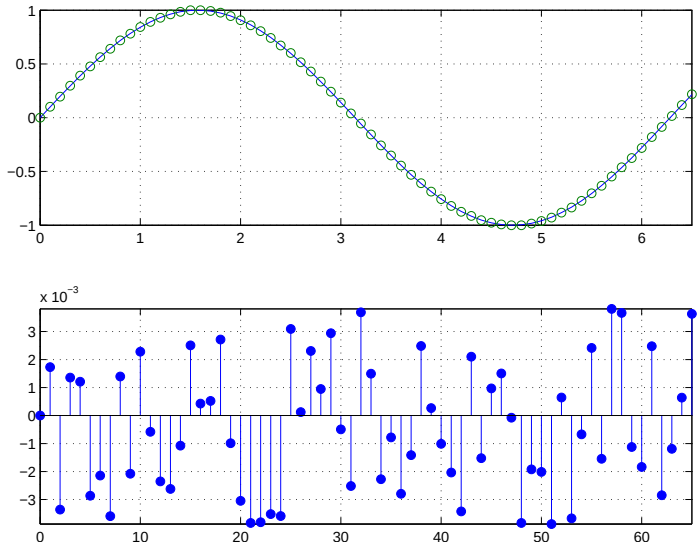
Jos käytössä on esimerkiksi seitsemän bittiä (+merkkibitti), voidaan kahden komplementtiaritmetiikalla esittää luvut $-128, -127, -126, \ldots, -1, 0, 1, \ldots, 125, 126, 127$. Nämä luvut skaalataan välille $[-1, 1]$ yksinkertaisesti jakamalla ne luvulla $2^7 = 128$. Näin saadaan kvantisointitasot $-\frac{128}{128}, -\frac{127}{128}, -\frac{126}{128}, \ldots, -\frac{1}{128}, \frac{0}{128}, \frac{1}{128}, \ldots, \frac{125}{128}, \frac{126}{128}, \frac{127}{128}$. Nämä kolme esitysmuotoa tapauksessa $b = 2$ on esitetty alla olevassa taulukossa.

binääriesitys	100	101	110	111	000	001	010	011
desimaaliesitys	-4	-3	-2	-1	0	1	2	3
kvantisointitaso	$-\frac{4}{4}$	$-\frac{3}{4}$	$-\frac{2}{4}$	$-\frac{1}{4}$	$\frac{0}{4}$	$\frac{1}{4}$	$\frac{2}{4}$	$\frac{3}{4}$

Taulukon tapauksessa kahden kvantisointivälin etäisyys on $\frac{1}{4}$ ja yleisesti ottaen se on 2^{-b} käytettäessä b:tä bittiä (+merkkibittiä). Kvantisoitaessa muodostuva virhe on enintään puolet tästä eli $2^{-b}/2$, jos käytetään pyöristystä lähimpään numeroon.[1] Tämä virhe $e(n)$ voidaan ajatella lisätyksi alkuperäiseen signaaliin $x(n)$, jolloin saadaan tulos

$$\hat{x}(n) = x(n) + e(n).$$

Alla olevissa kuvissa on esitetty sinisignaali yhtenäisellä viivalla ja tulos kvantisoitaessa seitsemään bittiin (+merkkibittiin) ympyröillä. Alemmassa kuvassa on kvantisointivirhe $e(n)$.



Tyypillisesti kvantisointivirheestä $e(n)$ tehdään seuraavat oletukset tilastollista analyysiä varten.

[1] Ylin kvantisointitaso on poikkeus tästä, koska esimerkiksi taulukon tilanteessa pitää kaikki luvut väliltä $[\frac{3}{4}, 1]$ pyöristää alaspäin lukuun $\frac{3}{4}$. Tämä poikkeus jätetään yleensä huomiotta, koska suuremmilla bittimäärillä virheen suuruusluokka on hyvin pieni ja se myöskin esiintyy hyvin harvoin.

1. Signaali $e(n)$ on stationaarinen satunnaisprosessi, eli sen tilastolliset ominaisuudet eivät muutu ajan myötä.
2. Signaalin $e(n)$ lukuarvot eivät riipu lukujonon $x(n)$ arvoista.
3. Signaalin $e(n)$ arvot ovat riippumattomia toisistaan, eli $e(n)$ on *valkoista kohinaa* (white noise).
4. Signaalin $e(n)$ arvot ovat jakautuneet tasaisesti välille $(-2^{-b}/2, 2^{-b}/2]$.

Nämä oletukset ovat voimassa, jos signaali on riittävän satunnainen. Ne eivät pidä paikkaansa kaikille signaaleille $x(n)$: jos esimerkiksi $x(n)$ on yksikköaskel, niin useimmat edellä mainituista oletuksista eivät ole voimassa. Näitä oletuksia käyttämällä saadaan kuitenkin johdettua yksinkertainen malli kvantisointivirheiden määrälle.

Jos oletetetaan järjestelmän käyttävän pyöristystä lähimpään lukuun, niin silloin signaalin $e(n)$ odotusarvo[2]

$$\mu_e = E[e(n)] = 0,$$

ja varianssi

$$\sigma_e^2 = E[(e(n) - \underbrace{\mu_e}_{=0})^2] = E[e(n)^2].$$

Odotusarvon (toisesta) määritelmästä[3] saadaan edelleen:

$$\begin{aligned} E[e(n)^2] &= \int_{-2^{-b}/2}^{2^{-b}/2} x^2 \frac{1}{2^{-b}}\, dx \\ &= 2^b \Big/_{-2^{-b}/2}^{2^{-b}/2} \frac{x^3}{3} \\ &= 2^b \left(\frac{2^{-3b}}{24} + \frac{2^{-3b}}{24} \right) \\ &= \frac{2^{-2b}}{12}. \end{aligned}$$

Kvantisoidun signaalin *signaali-kohinasuhde* (signal to noise ratio; SNR) määritellään signaalin tehon suhteena kohinan tehoon (desibeliasteikolla), ja se saadaan näiden varianssien suhteesta:

$$\mathrm{SNR} = 10 \log_{10} \frac{\sigma_x^2}{\sigma_e^2}.$$

Signaali-kohinasuhde on siis kvantisoinnissa b bittiin (+merkkibitti)

$$\begin{aligned} \mathrm{SNR} &= 10 \log_{10} \frac{\sigma_x^2}{2^{-2b}/12} \\ &= 10 \log_{10}(\sigma_x^2) + 10 \log_{10}(12) + 2b \cdot 10 \log_{10}(2) \\ &\approx 10 \log_{10}(\sigma_x^2) + 10.79 + 6.02b. \end{aligned}$$

[2]Kutakuinkin sama asia kuin lukujonon keskiarvo, s.o. $\lim_{n\to\infty} \frac{1}{2n+1} \sum_{k=-n}^{n} e(k)$.

[3]Jos tiedetään satunnaismuuttujan x jakauma $p(x)$, voidaan x:n odotusarvo laskea kaavasta $E[x] = \int_{-\infty}^{\infty} x p(x)\, dx$.

Signaali-kohinasuhde kasvaa siis noin 6 dB jokaista lisäbittiä kohden.

Yllä olevaa kaavaa sievempään muotoon ei ole mahdollista päästä tekemättä oletuksia signaalista $x(n)$ ja sen varianssista σ_x^2. Kulutuselektroniikkatuotteissa halutaan yleensä mahdollisimman hyvä signaali-kohinasuhde ja siksi teknisissä tiedoissa ilmoitettu SNR lasketaan mahdollisimman suuriamplitudiselle signaalille. Tällaiseksi sopii esimerkiksi sini- tai kosinisignaali amplitudilla 1, esimerkiksi

$$x(n) = \cos(2\pi \cdot 0.25 \cdot n).$$

Nyt signaalissa $x(n)$ toistuu jakso $\ldots, 1, 0, -1, 0, \ldots$, joten signaalin neliön odotusarvo on $\frac{1}{2}$. Samaan tulokseen päästään muillakin taajuuksilla kuin 0.25. Oletuksella $\sigma_x^2 = \frac{1}{2}$ saadaan signaali-kohinasuhteeksi

$$\text{SNR} \approx 10\log_{10}(\frac{1}{2}) + 10.79 + 6.02b = 6.02b + 7.78.$$

Esimerkiksi CD-soitin esittää näytteet 16 bitin tarkkuudella, joten tällä sinisignaalilla sen SNR$\approx 6.02 \cdot (16 - 1) + 7.78 \approx 98$ dB.

Toinen yleinen oletus signaalin $x(n)$ varianssista on, että sen amplitudi on skaalattu johonkin vakioarvoon. Käytännön tilanteissa täytyy nimittäin varautua siihen, että analoginen signaali, josta näytteitä otetaan, saa ajoittain hyvinkin suuria arvoja. Siksi sisääntulevan signaalin amplitudi on tapana kertoa jollain vakiolla A. Näin saatavan signaalin $Ax(n)$ varianssi on $A^2\sigma_x^2$, joten kvantisoinnin jälkeinen signaali-kohinasuhde on

$$\text{SNR} \approx 6.02b + 10.79 + 10\log_{10}(\sigma_x^2) + 20\log_{10}(A).$$

Nyrkkisääntönä voidaan sanoa, että valitsemalla $A = 1/(4\sigma_x)$ käytännössä eliminoidaan liian suurien arvojen saapumisen mahdollisuus. Tällöin skaalatun signaalin $Ax(n)$ varianssi on $\sigma_x^2/(16\sigma_x^2) = \frac{1}{16}$ ja signaali-kohinasuhde on desibeleissä

$$\text{SNR} \approx 6.02b + 10.79 + 10\log_{10}(\frac{1}{16}) = 6.02b - 1.25.$$

Muitakin skaalaustermejä toki käytetään ja kullekin niistä voidaan johtaa oma SNR-kaava. Mitä pienempi skaalaustermi A on, sitä pienemmäksi tulee myös SNR. Jos esimerkiksi halutaan signaali-kohinasuhteeksi tällä skaalauksella yli 80 dB, niin pitää olla $6.02b - 1.25 > 80$, eli

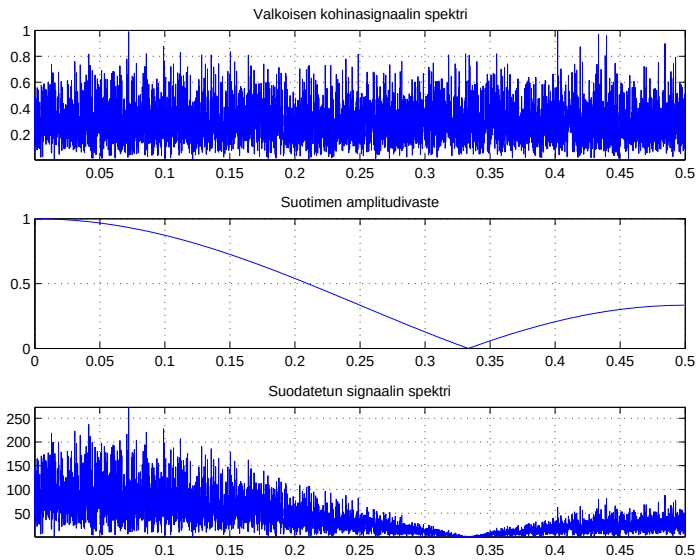
$$b > \frac{80 + 1.25}{6.02} = 13.50.$$

Näin ollen riittää valita 14-bittinen esitys (+merkki).

1.2 Suodatuksen vaikutus kvantisointikohinaan

Kulkiessaan digitaalisen suotimen läpi kvantisointikohina muuttuu kahdella tavalla: sen kokonaismäärä voi lisääntyä tai vähentyä ja toisaalta sen taajuusjakauma voi muuttua. Tässä kappaleessa keskitytään kvantisointikohinan kokonaismäärän muutokseen. Taajuusjakauman muutoshan on suoraviivainen prosessi, koska valkoista kohinaa suodatettaessa tuloksen spektri on täsmälleen saman näköinen kuin suotimen amplitudivastekin.

Kvantisointikohina käyttäytyy suodatuksessa kuten mikä tahansa muu signaalikin. Erityispiirteenä sillä on että kohina on valkoista, jolloin ennen suodatusta kaikkia taajuuksia on yhtä paljon. Esimerkiksi alla olevassa kuvassa on ylhäällä erään 10000 näytteen mittaisen kohinasignaalin taajuusjakauma. Kun se suodatetaan keskimmäisen kuvan amplitudivasteen mukaisella suotimella ($y(n) = [x(n)+x(n-1)+x(n-2)]/3$), saadaan alimman kuvan mukainen spektri. Selvästikin alimmassa kuvassa on amplitudivasteella kerrottuna ylimmän kuvan spektriarvot.



Kvantisointikohinan suodatusta tullaan myöhemmin käyttämään kappaleessa 3.5.2, jossa kvantisointikohinaa siirretään DA-muunnoksen yhteydessä korkeammille taajuuksille, jossa ihmiskorva ei sitä kuule.

Kvantisointikohinan määrän muutos suodatuksen yhteydessä on laskettavissa analyyttisesti. Tarkastellaan kvantisoidun signaalin $\hat{x}(n) = x(n) + e(n)$ suodatusta lineaarisella ja aikainvariantilla (LTI) järjestelmällä $\mathcal{F}(\cdot)$, jonka impulssivaste on $h(n)$. Koska järjestelmä on lineaarinen, niin suodatustuloksen

$$\mathcal{F}[\hat{x}(n)] = \mathcal{F}[x(n) + e(n)] = \mathcal{F}[x(n)] + \mathcal{F}[e(n)]$$

virhettä voidaan tarkastella erikseen. Suodatuksen jälkeinen pelkästään kvantisoinnista johtuva virhe on

$$f(n) = \mathcal{F}[e(n)] = \sum_{k=-\infty}^{\infty} h(k)e(n-k).$$

Tämän virheen odotusarvo on

$$\begin{aligned}
\mu_f &= E\left[\sum_{k=-\infty}^{\infty} h(k)e(n-k)\right] \\
&= \sum_{k=-\infty}^{\infty} h(k)E[e(n-k)] \\
&= \mu_e \sum_{k=-\infty}^{\infty} h(k).
\end{aligned}$$

Viimeiselle lausekkeelle saadaan myös käyttökelpoisempi muoto havaitsemalla z-muunnoksen määritelmästä, että

$$\mu_e \sum_{k=-\infty}^{\infty} h(k) = \mu_e H(1).$$

Varianssin laskenta on hieman hankalampaa, ja lausekkeen johto sivuutetaankin tässä yhteydessä. Lopputulos voidaan ilmaista seuraavasti. Kvantisointivirheestä johtuvan virheen varianssi σ_f^2 suodatuksen jälkeen voidaan esittää kolmella vaihtoehtoisella tavalla:

$$\begin{aligned}
\textbf{1:} \qquad \sigma_f^2 &= \sigma_e^2 \sum_{n=-\infty}^{\infty} |h(n)|^2 \\
\textbf{2:} \qquad \sigma_f^2 &= \frac{\sigma_e^2}{2\pi} \int_{-\pi}^{\pi} \left|H\left(e^{i\omega}\right)\right|^2 d\omega \\
\textbf{3:} \qquad \sigma_f^2 &= \frac{\sigma_e^2}{2\pi i} \oint_C H(z)H(z^{-1}) \frac{dz}{z},
\end{aligned}$$

missä σ_e^2 on sisäänmenevän (kvantisointivirheestä johtuvan) valkoisen kohinan varianssi, $h(n)$ on tarkasteltavan suotimen impulssivaste, $H(z)$ on sen siirtofunktio ja C on funktion $H(z)H(z^{-1})/z$ suppenemisalueella sijaitseva suljettu käyrä.

Erityisesti FIR-suotimen tapauksessa ensimmäinen tapa kolmesta on helpoin. Seuraavassa on kaksi esimerkkiä kvantisointivirheen käyttäytymisestä suodatettaessa IIR- ja FIR-suotimella.

Esimerkki: Oletetaan, että analoginen signaali muunnetaan digitaaliseksi $(1+7)$-bittisellä A/D-muuntimella, ja että näin saatu digitaalinen signaali suodatetaan IIR-suotimella, jonka siirtofunktio on

$$H(z) = \frac{1}{z + 0.5}.$$

Suodattimen toteuttava differenssiyhtälö on siis

$$y(n) = -0.5y(n-1) + x(n-1).$$

Mikä on ulostulossa oleva pelkästään kvantisoinnista johtuva virhe, kun äärellisestä laskentatarkkuudesta ja kertoimien äärellisestä esityksestä johtuvat virheet jätetään huomiotta?

Muunnosvaiheessa on käytössä 8 bittiä, joista yksi käytetään merkkibitiksi. Tässä tapauksessa siis $b = 7$. Muunnosvaiheen virheen varianssi saadaan edellä olleesta kaavasta,

$$\sigma_e^2 = \frac{2^{-2b}}{12} = \frac{2^{-14}}{12} = \frac{2^{-16}}{3} \approx 5.09 \cdot 10^{-6}.$$

Tätä vastaava keskiarvohan on 0, kun oletetaan, että analogisen signaalin arvot pyöristetään aina lähimpään lukuun suoran katkaisun asemesta. Ulostulon kvantisointivirheestä johtuvan kohinan varianssi voidaan johtaa ratkaisemalla järjestelmän impulssivaste:

$$h(n) = \left(-\frac{1}{2}\right)^{n-1} u(n-1)$$

ja sijoittamalla se edellä olleeseen kaavaan

$$\begin{aligned}
\sigma_f^2 &= \sigma_e^2 \sum_{n=-\infty}^{\infty} |h(n)|^2 \\
&= \sigma_e^2 \sum_{n=1}^{\infty} \left(\frac{1}{4}\right)^{n-1} \\
&= \sigma_e^2 \frac{1}{1-\frac{1}{4}} \\
&= \frac{4}{3}\sigma_e^2.
\end{aligned}$$

Kohinan varianssi kasvaa siis 4/3-kertaiseksi eli on kaikkiaan

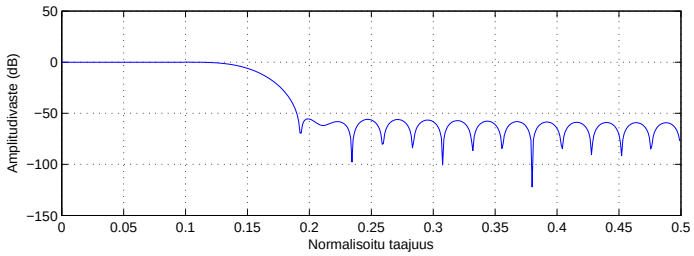
$$\sigma_f^2 = \frac{4}{3} \cdot \frac{2^{-16}}{3} \approx 6.78 \cdot 10^{-6}.$$

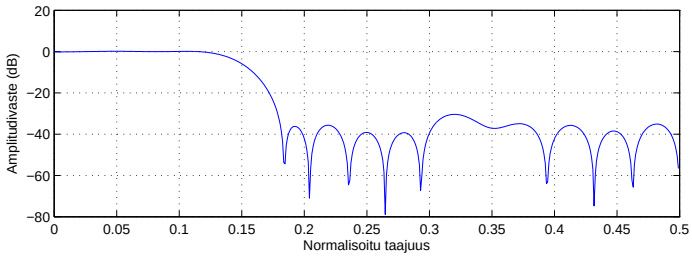
FIR-suodinten tapauksessa tilanne on yksinkertaisempi kuin IIR-suotimilla. Esimerkiksi FIR-suotimen, jonka impulssivasteen kertoimet ovat $1/3$, kun $n = 0, 1, 2$ ja 0 muulloin, ulostulossa olevan kvantisoinnista johtuvan kohinan varianssi on

$$\begin{aligned}
\sigma_f^2 &= \sigma_e^2 \sum_{n=-\infty}^{\infty} |h(n)|^2 \\
&= \sigma_e^2 \left(\left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2 \right) \\
&= \frac{3}{9}\sigma_e^2 = \frac{1}{3}\sigma_e^2.
\end{aligned}$$

1.3 Kertoimien pyöristämisen vaikutus

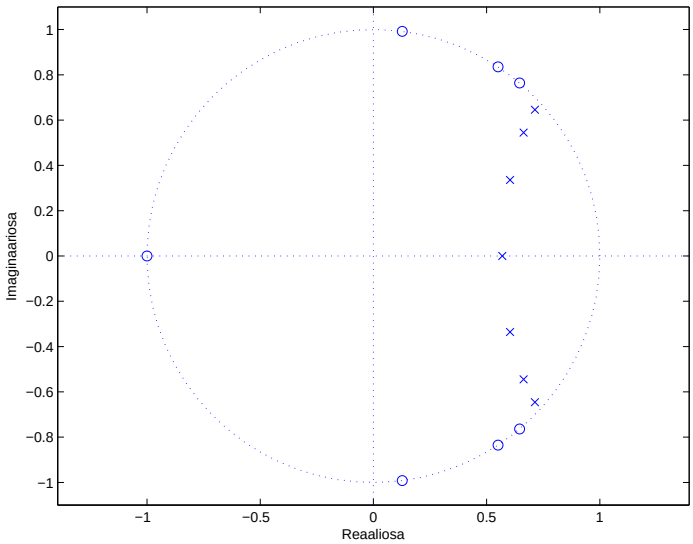
Suodinten suunnittelumenetelmien tavoitteena on suunnitella paras mahdollinen suodin tietyille kriteereille. Kun suodin sitten toteutetaan pienemmällä bittimäärällä, helpointa on pyöristää kertoimet lähimpään kvantisointitasoon. Kertoimia pyöristettäessä suodin muuttuu optimaalisesta aina huonompaan päin, jolloin suodin ei välttämättä toteutakaan annettuja vaatimuksia. Alla olevassa kuvassa on erään FIR-suotimen amplitudivaste ennen ja jälkeen kertoimien pyöristystä (1+7):n bitin tarkkuuteen.



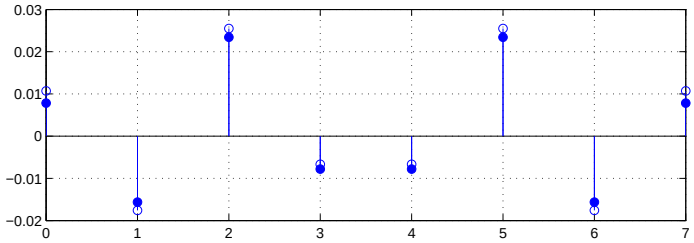


Amplitudivaste heikkenee selvästi ja suotimen toiminta ei välttämättä ole enää määrittelyjen mukaista.

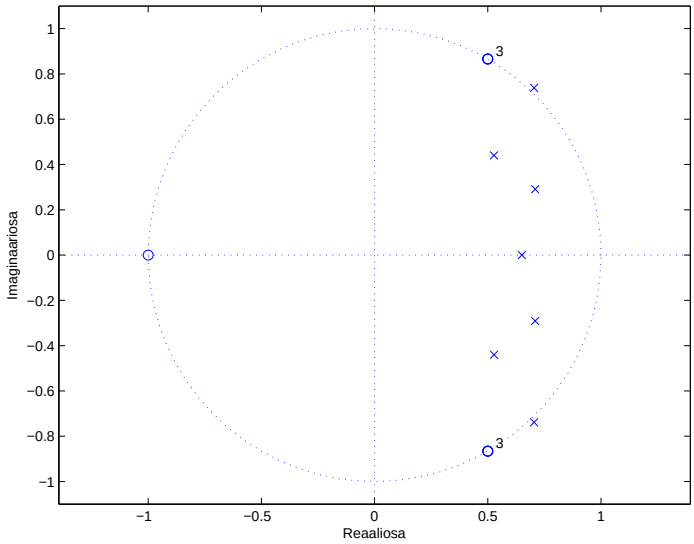
IIR-suotimen tapauksessa kvantisointi voi tuottaa vielä suuremman muutoksen suotimen toiminnassa: pahimmassa tapauksessa suotimesta tulee epästabiili. Alla olevassa kuvassa on edellisen kappaleen elliptisen IIR-suotimen napa-nollakuvio.



Kun suotimen kertoimet kvantisoidaan (1+7):n bitin tarkkuuteen, kertoimien muutos ei ole erityisen suuri. Alla olevassa kuvassa on kuvattu siirtofunktion osoittajan kertoimien muutos kvantisoitaessa. Alkuperäiset arvot on esitetty avoimilla ympyröillä ja kvantisoidut mustilla ympyröillä.



Suotimen napoihin ja nolliin tämä vaikuttaa kuitenkin paljon enemmän. Alla olevasta kuvasta nähdään, että kvantisoidusta suotimesta tuli epästabiili.



Useimmiten on olemassa parempiakin (1+7)-bittisiä suotimia kuin se, joka saadaan pyöristämällä kukin kerroin suoraan lähimpään (1+7)-bittiseen lukuun. Kertoimien muutoksen vaikutus yhteys ja nolliin ja suotimen amplitudivasteeseen on melko monimutkainen, joten suoraviivainen pyöristäminen ei välttämättä tuota parasta mahdollista amplitudivastetta. Kertoimien optimaaliseen sijoitteluun eri kvantisointitasoille perehdytään myöhemmillä signaalinkäsittelyn kursseilla.

Harjoitustehtäviä

1.1. Laske A/D-muunnoksessa syntyvän kvantisointikohinan varianssi kun käytössä on

(a) merkkibitti+3 bittiä,

(b) merkkibitti+7 bittiä,

(c) merkkibitti+11 bittiä,

(d) merkkibitti+15 bittiä.

1.2. Oletetaan, että analoginen näytteistettävä signaali skaalataan vakiolla $A = 1/(4\sigma_x)$. Mikä on digitaalisen signaalin signaali-kohinasuhde (SNR) kun käytössä on

(a) merkkibitti+3 bittiä,

(b) merkkibitti+7 bittiä,

(c) merkkibitti+11 bittiä,

(d) merkkibitti+15 bittiä?

1.3. Laske signaali-kohinasuhteen kaava, jos skaalauksessa käytetäänkin vakiota

$$A = \frac{1}{2\sigma_x}.$$

1.4. Oletetaan, että analoginen signaali on muotoa

$$x(t) = \sin(\omega t + \phi).$$

Tälle signaalille $\sigma_x^2 = 1/2$. Mikä on nyt signaali-kohinasuhteen kaava näytteenoton jälkeen, kun oletetaan, ettei analogista signaalia skaalata lainkaan? Mikä on SNR, kun käytetään 15 bittiä merkkibitin lisäksi?

1.5. *(Matlab)* Tutkitaan tässä tehtävässä kvantisointikohinaa Matlabilla.

(a) Avaa ensin testisignaali komennolla `load gong.mat`, joka lataa sen muuttujaan `y`. Kvantisoi testisignaali seitsemään bittiin (+merkkibittiin) komennolla `z=quant(y,1/2^7);`. Tulosta virhesignaali `y-z` ruudulle ja laske sen varianssi komennolla `var`). Vertaa tätä teoreettiseen tulokseen (tehtävä 1.1). Toista sama (1+3):lle, (1+11):lle ja (1+15):lle bitille.

(b) Toista testi testisignaalille `seiska.wav`, joka löytyy kurssin kotisivulta. Wav-tiedostot voi ladata Matlabiin komennolla `y = wavread('seiska.wav');`. Mistä erot kvantisointivirheen varianssissa eri testisignaaleille johtuvat?

1.6. *(Matlab)* Suunnittele Matlabilla elliptinen IIR-suodin, jonka estokaista on näytteenottotaajuudella normalisoituna $[0, 0.1]$ ja päästökaista $[0.15, 0.5]$. Estokaistan minimivaimennus on 50 dB ja päästökaistan maksimivärähtely 0.1 dB. Kvantisoi kertoimet (vektorit `b` ja `a`) (1+11):n bittiin ja tulosta ruudulle näin saadun suotimen taajuusvaste kvantisoimattoman kanssa samaan kuvaan. Kuinka paljon estokaistan vaimennus huononi?

1.7. Tarkastellaan ensimmäisen asteen IIR-suodinta, jonka siirtofunktio on

$$H(z) = \frac{1}{1 - 0.8z^{-1}}.$$

Oletetaan että analoginen signaali muunnetaan digitaaliseksi 8-bittisellä A/D-muuntimella (merkki+7 bittiä). Mikä on suotimen ulostulossa oleva pelkästään kvantisoinnista johtuva virhe, kun äärellisestä laskentatarkkuudesta ja kertoimien äärellisestä esityksestä johtuvat virheet jätetään huomiotta?

1.8. *(Matlab)* Luo normaalijakautunut satunnainen testisignaali komennolla `randn`. Pituus voi olla joitakin tuhansia näytteitä. Kvantisoi signaali kahdeksaan bittiin (merkki + 7 bittiä), ja laske empiirisesti näin saadun kvantisointikohinan varianssi. Suodata kvantisoitu sekä kvantisoimaton signaali tekstissä olevan esimerkin suotimella $y(n) = -0.5y(n-1) + x(n-1)$. Suodatuksen jälkeinen kvantisointikohina on nyt tulossignaalien erotus.

(a) Laske suodatuksen jälkeisen kvantisointikohinan varianssi ja vertaa tulosta esimerkissä laskettuun teoreettiseen tulokseen.

(b) Teoriassa kohinan varianssi kasvoi suodatettaessa $\frac{4}{3}$-kertaiseksi. Entä käytännössä?

1.9. Tarkastellaan ensimmäisen asteen IIR-suodinta, jonka siirtofunktio on

$$H(z) = \frac{1}{1 - az^{-1}}.$$

Tässä lausekkeessa a on jokin reaalivakio, jonka itseisarvo $|a| < 1$. Johda (vakiosta a riippuva) lauseke suotimen ulostulossa olevalle pelkästään kvantisoinnista johtuva virheelle, kun äärellisestä laskentatarkkuudesta ja kertoimien äärellisestä esityksestä johtuvat virheet jätetään huomiotta ja A/D-muunnin käyttää $(1 + 7)$ bitin esitystä? Mitä virheelle tapahtuu kun $|a|$ lähestyy ykköstä?

1.10. *(Matlab)* Käytettäessä IIR-suotimia äärellisellä laskentatarkkuudella, voi syntyä ilmiö nimeltä *rajasykli*. Tällöin järjestelmän vaste ei herätteen loputtua vaimene vaan jää ikuisesti värähtelemään jollain taajuudella nollan ympärille.

Lataa testiskriptin pohja osoitteesta

```
http://www.cs.tut.fi/kurssit/SGN-1251/rajasykli.m
```

Kirjoita kommentoidun for-silmukan tilalle koodi, jossa muuttujaan `y(n)` sijoitetaan suodatuksen tulos kvantisoinnin jälkeen. Kvantisointi tehdään viiteen bittiin (+merkkibitti) `quant`-funktiolla. Toteutettavan suotimen siirtofunktio on

$$H(z) = \frac{1}{1 + 0.5z^{-1}}.$$

Aja skripti, jolloin suotimen ulostulo tulostuu ruudulle. Vaikka heräte on nollassa heti kahden ensimmäisen näytteen jälkeen, tulos jää värähtelemään ikuisesti (tämän suotimen tapauksessa Nyquistin taajuudella).

Luku 2

Spektrin estimointi

Spektrin tarkastelu on useassa sovelluksessa tärkeä menettely analysoida signaalin ominaisuuksia. Tärkeitä sovelluskohteita löytyy esimerkiksi lääketieteestä, jossa voidaan EEG-signaalin spektristä yli 90-prosenttisella varmuudella päätellä, onko testattavalla Huntingtonin tauti, Parkinsonin tauti tai skitsofrenia. Myös puheentunnistus perustuu lähes täysin spektrin laskentaan ja siitä johdettuihin piirteisiin pääteltäessä mikä äänne kulloinkin on sanottu.

Annetun signaalin spektrin estimointi tarkoittaa yksinkertaisimmillaan signaalin muuntamista taajuustasoon diskreetin Fourier-muunnoksen avulla. Signaalin spektrillä tarkoitetaankin kuvaajaa, josta nähdään kaikki signaalin sisältämät taajuudet. Diskreetti Fourier-muunnos tarjoaa sinällään varsin hyvän estimaatin signaalin todellisesta spektristä, mutta on olemassa tilanteita, joissa pelkkä FFT ei toimi. Tällöin on käytettävä monimutkaisempia menetelmiä, joita tarkastellaan tässä luvussa.

Fourier-muunnos laskee signaalin amplitudin jakautumisen eri taajuuksille. Usein amplitudi ei ole kuitenkaan niin kiinnostava kuin tehon jakauma, ja yleensä spektrin estimoinnilla tarkoitetaan tämän ns. *tehospektrin* (engl. power spectral density) laskentaa. Koska teho on suoraan verrannollinen amplitudin neliöön, tehojakauman laskenta onnistuu yksinkertaisimmillaan korottamalla Fourier-muunnos toiseen potenssiin. Tarkkaan ottaen näin saatava signaalin $x(n)$ tehospektri $P_{xx}(e^{i\omega})$ määritellään siis kaavalla

$$P_{xx}(e^{i\omega}) = |X(e^{i\omega})|^2,$$

missä $X(e^{i\omega})$ on signaalin $x(n)$ diskreettiaikainen Fourier-muunnos:

$$X(e^{i\omega}) = \left| \sum_{n=-\infty}^{\infty} x(n) e^{-i\omega n} \right|^2 .$$

Ongelmaksi yo. kaavan laskennassa muodostuu Fourier-muunnoksen vaatima ääretön summa, jonka laskentaan on kaksi yleisesti käytettyä ratkaisua.

2.1 FFT-menetelmä ja autokorrelaatiomenetelmä

Yksinkertainen tapa on katkaista $x(n)$ ja laskea diskreetti Fourier-muunnos (eli FFT) vaikkapa $N = 1024$ näytteen mittaisesta pätkästä. Samalla kuitenkin oletetaan signaalin olevan jaksollinen.

Toinen ratkaisu on käyttää ns. Wiener-Khinchin-lausetta[1], jonka mukaan tehospektri saadaan myös autokorrelaatiofunktion $r_{xx}(k) = E[(x(n)x(n+k)]$ Fourier-muunnoksesta. Toisin sanoen, tehospektrille $P_{xx}(e^{i\omega})$ on myös kaava

$$P_{xx}(e^{i\omega}) = \sum_{k=-\infty}^{\infty} r_{xx}(k)e^{-i\omega k} = \sum_{k=-\infty}^{\infty} E[(x(n)x(n+k)]e^{-i\omega k}.$$

Käytännössä tässäkin tapauksessa joudutaan tekemään approksimaatio ja rajaamaan ääretön summa äärelliseksi sekä arvioimaan odotusarvoa $E[(x(n)x(n+k)]$ esimerkiksi $N = 1024$ näytteestä lasketulla keskiarvolla:

$$E[(x(n)x(n+k)] \approx \frac{1}{N-k} \sum_{n=0}^{N-k-1} x(n)x(n+k).$$

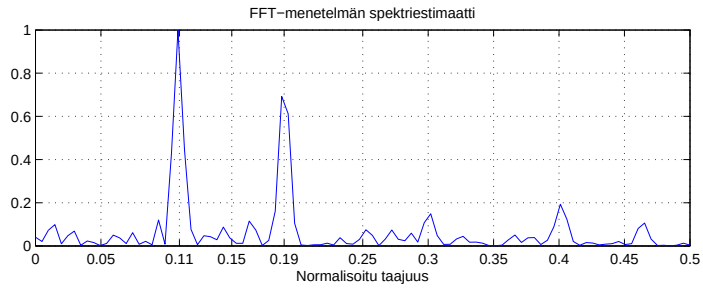
Huomaa, että keskiarvon laskennassa käytettävien tulojen lukumäärä vaihtelee. Esimerkiksi tapauksessa $N = 1024$ ja $k = 1022$ keskiarvo voidaan laskea vain kahdesta tulosta jossa tekijät ovat riittävän etäällä ($x(0) \cdot x(1022)$ ja $x(1) \cdot x(1023)$), kun taas esimerkiksi tapauksessa $k = 1$ keskiarvoon saadaan 1023 termiä. Tämä tekee menetelmän herkäksi reunimmaisten arvojen vaihtelulle, mutta tähänkin on olemassa monenlaisia ratkaisuja (esimerkiksi jätetään äärimmäiset keskiarvot kokonaan pois summasta).

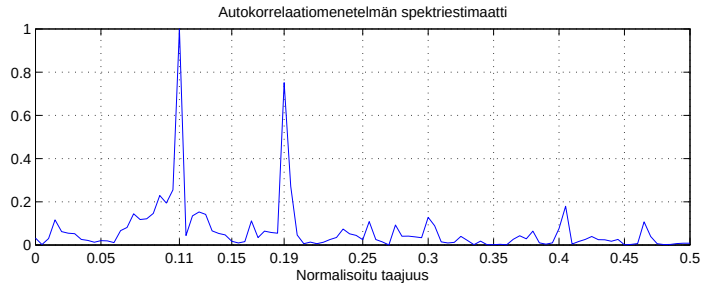
Molemmilla menetelmillä siis tulee virhettä: suora FFT:n laskenta olettaa signaalin jaksolliseksi (vaikka se ei ole), kun taas autokorrelaatiomenetelmä olettaa etteivät yli $N = 1024$ näytteen etäisyydellä olevat näytteet korreloi keskenään (vaikka näin yleensä ei olekaan).

Alla on esimerkki kahden sinisignaalin spektriestimaateista näillä menetelmillä. Testisignaali luotiin komennoilla

```
n = (0:99)';
x = cos(2*pi*0.125*n) + (1/sqrt(2))*sin(2*pi*0.19*n) + randn(size(n));
```

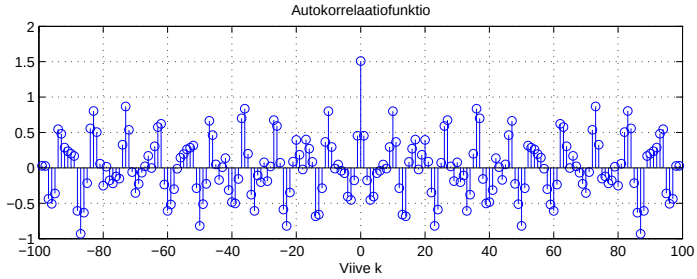
eli se koostuu kahdesta kohinaisesta sinisignaalista. Tuloksesta havaitaan, että molemmat menetelmät tunnistavat taajuuskomponentit hyvin.



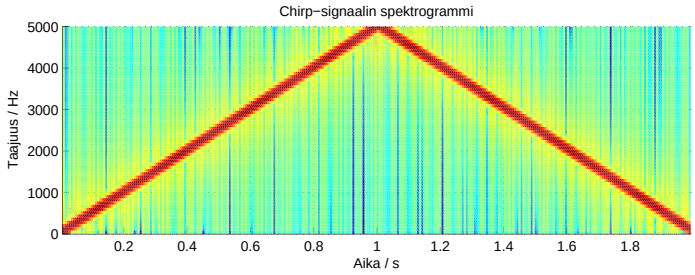


[1] http://en.wikipedia.org/wiki/Wiener-Khinchin_theorem

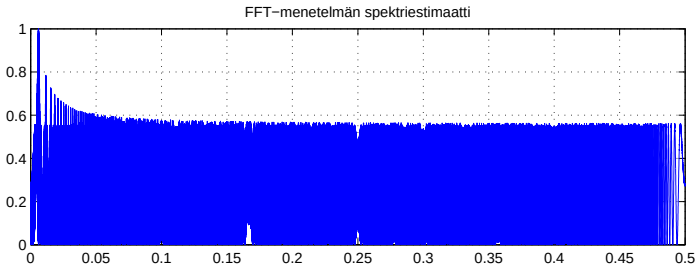
Alla olevassa kuvassa on autokorrelaatiomenetelmän laskema autokorrelaatiofunktio kaikille mahdollisille viiveille $k = -99, \ldots, 99$.

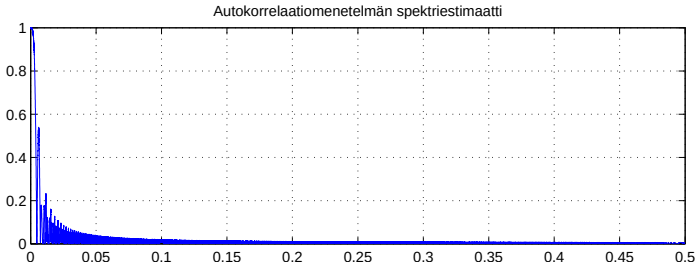


Autokorrelaatiomenetelmälle pahin tilanne on sellainen, jossa signaali on ei-satunnainen ja sen taajuussisältö muuttuu näytteen aikana. Esimerkki tällaisesta signaalista on ns. Chirp-signaali, jonka taajuus nousee tai laskee lineaarisesti. Alla olevassa kuvassa on kahden sekunnin mittainen Chirp-signaalin spektrogrammi.



Todellisuudessa signaali sisältää kaikkia taajuuksia suunnilleen yhtä paljon. Laskettaessa spektriestimaatti FFT:llä ja autokorrelaatiomenetelmällä saadaan alla olevat tulokset. Kuvasta nähdään selvästi että alempana oleva autokorrelaatiomenetelmän tulos tunnistaa parhaiten matalat taajuudet, mutta kadottaa korkeammat taajuudet. Ylempänä olevassa FFT:n tuloksessa taajuusjakauma on tasaisempi, eli se on lähempänä oikeaa.





Tärkein syy autokorrelaatiomenetelmän epäonnistumiselle on se, että signaalin taajuusjakauma on erilainen reunoilla kuin keskellä. Suurempien viiveiden autokorrelaatioita ei nimittäin pystytä laskemaan kuin reunoilta: signaalissa on 20000 näytettä, jolloin esimerkiksi korrelaatio $E[x(n)x(n+19000)]$ arvioidaan keskiarvona tuhannesta reunalla olevasta näytteestä, kun taas korrelaatio $E[x(n)x(n+1)]$ voidaan laskea keskiarvona 19999 näyt-

teestä (ml. keskellä olevat).

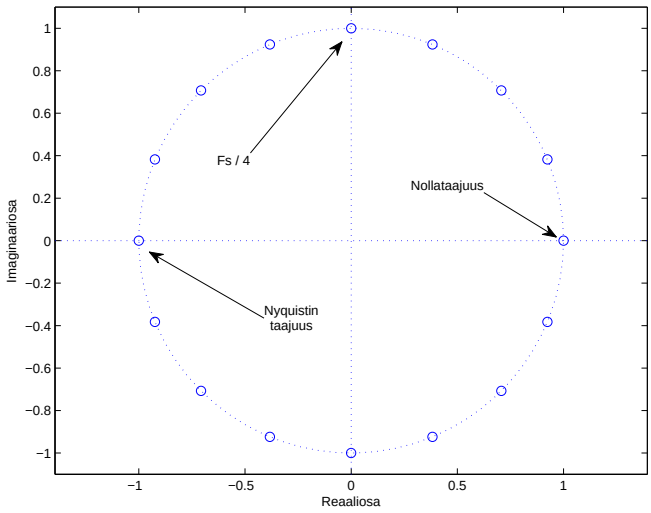
Nyrkkisääntönä voidaan sanoa, että autokorrelaatiomenetelmä toimii sitä paremmin mitä satunnaisempi analysoitava signaali on (esimerkiksi soinniton äänne puhesignaalissa; esim. 's' tai 'f'). Tällöin nimittäin peräkkäisten näytteiden korrelaatio on vähäinen, eikä menetelmän oletus kaukana olevien näytteiden korreloimattomuudesta haittaa. Mikäli taas signaali on ei-satunnainen (esimerkiksi soinnillinen äänne puheessa; esim. vokaali), toimii FFT yleensä paremmin.

Yksinkertaisuudestaan johtuen FFT on yleisempi menetelmä reaaliaikaisissa sovelluksissa (kuten spektrianalysaattoreissa tai viritysmittareissa). Sen sijaan tieteellisissä ei-reaaliaikaisissa sovelluksissa käytetään usein autokorrelaatiomenetelmää, koska se on mahdollista saada paljon tarkemmaksi kehittämällä autokorrelaation laskentaa tilastotieteen keinoin.

Seuraavassa tarkastellaan yksinkertaisuuden vuoksi ainoastaan FFT:n käyttöä spektriestimaatin laskennassa. Samat ratkaisut (esim. ikkunointi) voidaan yleistää myös autokorrelaatiomenetelmälle.

2.2 Spektrin resoluutio

Fourier-muunnos laskee kuinka paljon eri taajuuksia signalissa on mukana. Tämä tapahtuu esittämällä signaali eri taajuuksilla värähtelevien kompleksisten eksponenttifunktioiden painotettuna summana tai integraalina. Diskreetin Fourier-muunnoksen tapauksessa näitä funktioita on äärellinen määrä, eli yhtä monta kuin signaalissa on näytteitä. Jos signaalin näytteiden määrä on (parillinen) N, esitetään se kulmataajuuksilla $0, \frac{2\pi}{N}, \frac{4\pi}{N}, \frac{6\pi}{N}, \ldots, \frac{2(N-1)\pi}{N}$ värähtelevien eksponenttifunktioiden avulla. Esimerkiksi tapauksessa $N = 16$ taajuuksia on 16 ja ne ovat $0, \frac{\pi}{8}, \frac{2\pi}{8}, \ldots \frac{15\pi}{8}$. Koska eksponenttifunktion jakso on 2π, voidaan puolet näistä taajuuksista tulkita negatiivisiksi: $\frac{-7\pi}{8}, \frac{-6\pi}{8}, \ldots, \frac{-2\pi}{8}, \frac{-\pi}{8}, 0, \frac{\pi}{8}, \frac{2\pi}{8}, \ldots, \frac{7\pi}{8}, \frac{8\pi}{8}$. Reaalisilla lukujonoilla DFT on symmetrinen nollataajuuden suhteen, joten negatiivisilla taajuuksilla on sama amplitudi-informaatio kuin positiivisillakin. Näin ollen kuudestatoista kertoimesta jää jäljelle vain yhdeksän ei-negatiivista taajuuspistettä: $0, \frac{\pi}{8}, \frac{2\pi}{8}, \ldots, \frac{7\pi}{8}, \frac{8\pi}{8}$. Nämä taajuuspisteet jakautuvat tasaisesti nollataajuuden ja Nyquistin rajataajuuden $F_s/2$ välille alla olevan kuvan mukaisesti.

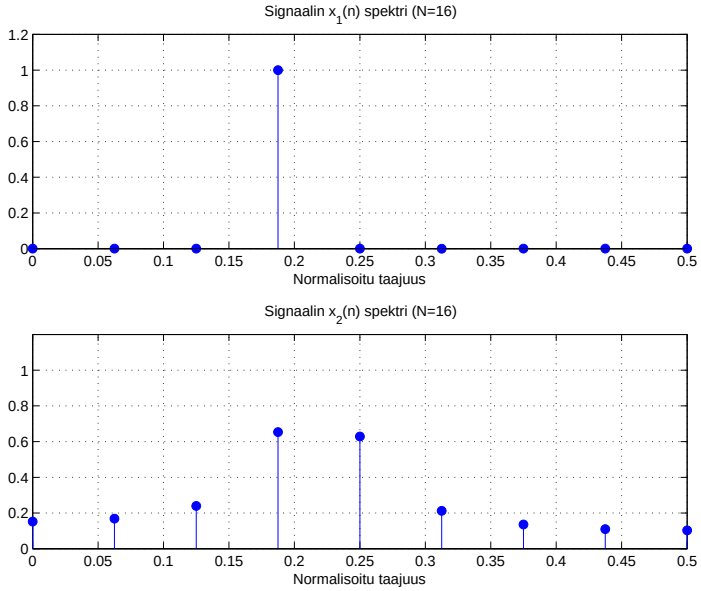


Jos esimerkiksi $F_s = 24$ kHz, Saadaan DFT:n avulla selville taajuuksien 0 Hz, 1500 Hz, 3000 Hz, 4500 Hz, 6000 Hz, 7500 Hz, 9000 Hz, 10500 Hz ja 12000 Hz amplitudit. Spektrin resoluutio eli taajuuspisteiden väli on siis 24/16 Hz. Yleisesti, jos signaalissa on (parillinen) N näytettä näytteenottotaajuudella F_s, saadaan DFT:stä $N/2 + 1$ taajuuspistettä: $k \cdot F_s/N$, missä $k = 0, 1, \ldots, N/2$. Taajuusresoluutio on F_s/N.

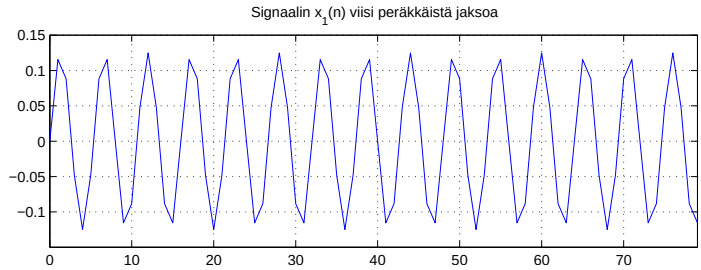
Koska taajuuspisteitä on rajallinen määrä, ei DFT pysty esittämään tarkasti kaikkia taajuuksia. Tarkastellaan esimerkiksi kahta sinisignaalia

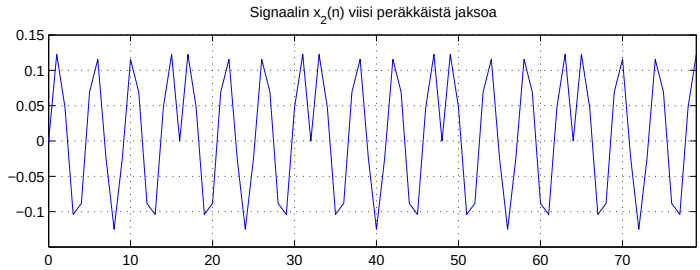
$$\begin{aligned} x_1(n) &= \frac{1}{8}\sin\left(\frac{6\pi n}{16}\right), \\ x_2(n) &= \frac{1}{8}\sin\left(\frac{7\pi n}{16}\right). \end{aligned}$$

Signaalin $x_1(n)$ taajuus $\omega_1 = \frac{6}{16}\pi$ voidaan esittää kuudentoista pisteen DFT:n avulla tarkasti, mutta signaalin $x_2(n)$ taajuutta $\omega_2 = \frac{7}{16}\pi$ ei voida. Signaalien DFT:n yhdeksän ensimmäistä kerrointa on esitetty alla olevassa kuvassa.



Kun signaalin taajuus osuu tarkalleen jonkin DFT:n taajuuden kohdalle, saadaan todellinen taajuus tarkasti. Kaikissa muissa tapauksissa signaalin energia jakautuu useamman DFT:n taajuuden kesken. Pohjimmiltaan tähän on syynä se, ettei signaalin $x_2(n)$ jakso ole kuusitoista, ja DFT olettaa signaalin olevan jaksollinen. Alla olevassa kuvassa on esitetty kummankin signaalin viisi peräkkäistä jaksoa.



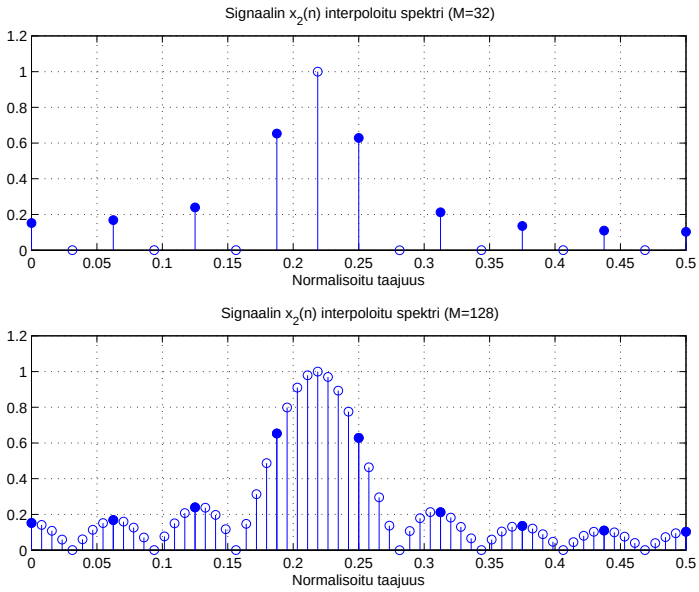


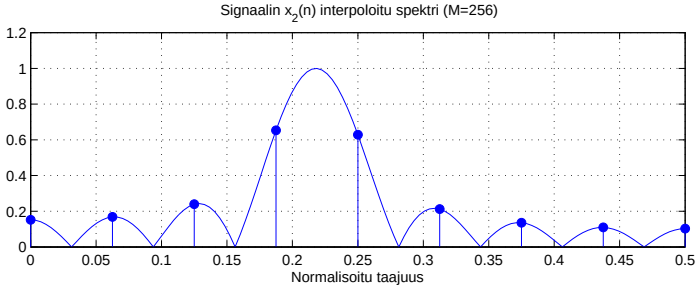
Alemmassa kuvassa signaalin todellinen jakso katkeaa kesken, vaikka DFT olettaakin signaalin olevan jaksollinen. Koska ilmiö on käytännössä mahdoton välttää täysin, sen vaikutusta pyritään lieventämään muilla menetelmillä.

Edellisen esimerkin tapauksen $\omega_2 = \frac{7}{16}\pi$ todellinen taajuus löydetään yksinkertaisella interpolointimenettelyllä. Lisäämällä nollia tarkasteltavan otoksen perään, saadaan DFT:hen lisää pisteitä (käytetään interpoloidun signaalin näytteiden määrästä jatkossa merkintää M). Tämä menettely on olennaisesti ainoastaan spektrin interpolointia, eikä tietenkään tuo siihen mitään uutta informaatiota. Kuitenkin näin saadaan tietyt signaalin piirteet paremmin näkyville. Edellisen esimerkin tilanteessa siis lisätään alkuperäisen kuudentoista näytteen mittaisen signaalin perään esimerkiksi 16 nollaa, jolloin saadaan $M = 32$ näytteen mittainen signaali (englanninkielinen nimitys operaatiolle on zero-padding). Tästä otetaan seuraavaksi DFT, jolloin saadaan spektri, jossa voidaan tarkemmin esittää signaalin taajuudet.

Alla olevassa kuvassa on esitetty alkuperäisen signaalin spektri, joka saadaan kun lisätään ensin 16 nollaa ja otetaan sen jälkeen DFT. Mustat pallot saavat samat arvot kuin interpoloimattomassa spektrissä ja valkoiset ovat uusia, interpoloinnin tuomia pisteitä. Seuraavassa kuvassa nollia on lisätty 112 kappaletta ($M = 128$) ja alimmassa 240 kappaletta ($M = 256$). Molemmissa on merkitty mustilla ympyröillä ne kohdat, jotka löytyvät jo kuudentoista mittaisella DFT:llä.

Kuvasta nähdään, että todellinen taajuus löytyy nyt korkeimman arvon kohdalta. Tämä johtuu siitä että sinisignaalin aiheuttama piikki ei nyt jakaudu niin ikävästi useamman komponentin osalle.





Tämä onkin tyypillinen menettely spektrin laskemisessa. Jos muunnettavan vektorin dimensio ei ole muotoa 2^L, niin silloin sen loppuun lisätään nollia siten, että saadaan dimensioksi luvun 2 potenssi. Tästä on se hyöty, että tällöin voidaan käyttää nopean Fourier-muunnoksen algoritmia (FFT). Lisäksi tällä menettelyllä saadaan parempi estimaatti signaalin spektristä. Yllä olevassa kuvassa päästään nollia lisäämällä parempaan tulokseen, mutta selvästikään tulos ei ole vielä signaalin varsinainen spektri. Todellinen spektri on ainoastaan yksi piikki, jonka pitäisi sijaita taajuudella $\omega = 7\pi/16$. Sen sijaan alimmassa kuvassa on signaalinkäsittelyssä hyvin tyypillinen $\sin(x)/x$ -muotoinen kuvaaja. Sama funktio tuli usein vastaan suunniteltaessa FIR-suotimia ikkunamenetelmällä. Selvitetään seuraavaksi miten kyseinen funktio liittyy spektrin estimointiin.

Kun signaalista $x(n)$ lasketaan DFT, niin tällöinhän varsinaisesta signaalista katkaistaan ensin sopivan mittainen osa (yleensä muotoa $N = 2^L$). Tällöin saadaan uusi signaali, joka on muotoa

$$v(n) = w(n)x(n),$$

missä

$$w(n) = \begin{cases} 1, & \text{kun } n = 0, 1, 2, \ldots, N-1, \\ 0, & \text{muulloin.} \end{cases}$$

Kuten diskreetin Fourier-muunnoksen tarkastelusta muistetaan, kertolasku aika-alueella vastaa konvoluutiota taajuusalueella, ts.

$$V(n) = \frac{1}{N} W(n) * X(n).$$

Siis tulokseksi saadaan spektri, joka on aidon spektrin konvoluutio ikkunan $w(n)$ diskreetin Fourier-muunnoksen kanssa:

$$V(n) = \frac{1}{N} \sum_{k=0}^{N-1} W(n-k)X(k),$$

missä lukujonot $W(n)$ ja $X(n)$ täytyy laskettaessa olettaa jaksollisiksi ja äärettömän pitkiksi.

Erityisesti siis meidän esimerkissämme todellinen spektri $X(n)$ on muotoa $\alpha\delta(n-\beta)$, jolloin

$$V(n) = \frac{\alpha}{N} \sum_{k=0}^{N-1} W(n-k)\delta(k-\beta) = \frac{\alpha}{N} W(n-\beta).$$

Näin ollen havaittu spektri $V(n)$ on siis ikkunan $w(n)$ siirretty ja skaalattu Fourier-muunnos. Sen lauseke on laskettu aikaisemmin, ja se on muotoa $\sin(x)/x$ siirrettynä ja skaalattuna.

Koska diskreetti Fourier-muunnos on lineaarinen operaatio, niin jokainen signaalin sisältämä taajuus voidaan ajatella muunnettavaksi erikseen ja tämän jälkeen laskettavaksi yhteen. Siksi jokaisen signaalin Fourier-muunnos muodostuu muotoa $\sin(x)/x$ olevien funktioiden summana, ja tarkastelu voidaan periaatteessa palauttaa yhden taajuuden tarkasteluun.

2.3 Spektrin ikkunointi

DFT:n laskennassa vain N taajuuskomponenttia voidaan siis esittää tarkasti ja kaikki muut taajuudet näkyvät yhtenä leveänä huippuna (*pääkeila, engl. main lobe*) sekä taajuuksien mukaan vaimenevina matalampina huippuina (*sivukeilat, engl. side lobe*). Ongelman poistamiseksi menetellään samoin kuin FIR-suodinten suunnittelussa, eli kerrotaan näytteet ikkunafunktiolla. Ikkunan tarkoituksena FIR-suodinten suunnittelussa on vähentää värähtelyä päästökaistalla ja estokaistalla. Nyt tarkoituksena on saada pääkeila mahdollisimman kapeaksi ja sivukeilat mahdollisimman mataliksi. Nämä osoittautuvat osin ristiriitaisiksi vaatimuksiksi, joten ikkunafunktion valinta on nytkin kompromissi. Ikkunafunktiota käytetään ennen varsinaisen DFT:n laskemista kertomalla ikkunan ja signaalin arvot komponenteittain keskenään.

Tilanne on sen verran erilainen FIR-suodinten suunnitteluun nähden, että käytettävät ikkunat poikkeavat osittain aikaisemmin tarkastelluista. Seuraavissa kappaleissa tarkastellaan tärkeimpiä ikkunoiden ominaisuuksia. Taulukon sarake *sivukeilan korkeus* kuvaa sivukeilojen voimakkuutta ja *kuuden desibelin kaistanleveys* ja kuvaa spektrin taajuusresoluutiota.

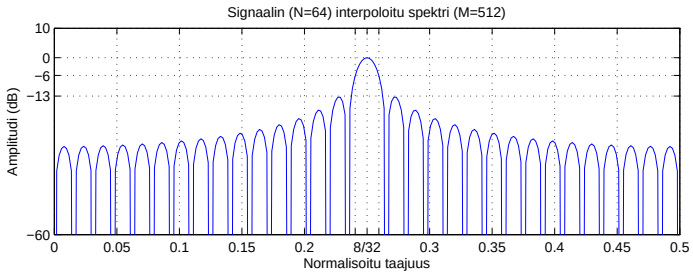
Suodinsuunnittelun yhteydessä tutustuttiin jo muutamiin ikkunoiden ominaisuuksiin. Tuolloin oli esillä saman tyyppinen taulukko kuin alla.

Ikkuna-funktion nimi	Sivukeilan korkeus (dB)	6 dB kaistan-leveys	Ikkunan lauseke $w(n)$, kun $\lvert n\rvert \leq (N-1)/2$
Suorakulmainen	-13	1.21	1
Bartlett	-25	1.78	$1 - \frac{2\lvert n\rvert}{N-1}$
Hanning	-31	2.00	$0.5 + 0.5\cos\left(\frac{2\pi n}{N}\right)$
Hamming	-41	1.81	$0.54 + 0.46\cos\left(\frac{2\pi n}{N}\right)$
Blackman	-57	2.35	$0.42 + 0.5\cos\left(\frac{2\pi n}{N}\right) + 0.08\cos\left(\frac{4\pi n}{N}\right)$

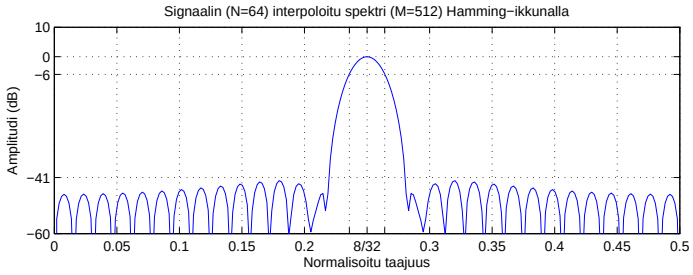
2.3.1 Sivukeilan korkeus

Taulukon toisessa sarakkeessa on esitetty pääkeilan ja korkeimman sivukeilan korkeuksien suhde. Tämä suhde riippuu vain ikkunasta eikä esimerkiksi näytteiden määrä N vaikuta siihen. Taulukon luku kertoo suoraan sen kuinka korkeita huippuja tulee yksittäisen spektripiikin ympäristöön. Esimerkiksi suorakulmaista ikkunaa käytettäessä korkein sivukeila on taulukon mukaan 13 dB pääkeilaa alempana. Sivukaistojen amplitudi on siis -13 dB suhteessa pääkaistaan, mikä lineaarisella asteikolla vastaa kertomista arvolla 0.2239. Sivukaistan korkeimmat huiput ovat näin ollen hieman reilu viidesosa pääkeilan korkeudesta. Alla olevassa kuvassa olevassa esimerkissä on kuvattu signaalin $x_2(n) = \sin(\frac{8\pi n}{16})$ spektri ilman ikkunointia desibeliasteikolla. Nyt käytössä on $N = 64$ näytettä, joka interpoloidaan $M = 512$:n näytteen mittaiseksi. Koska esimerkissä ei käytetä ikkunaa, tilanne

vastaa suorakulmaisen ikkunan käyttöä. Nyt korkein sivukeila on noin -13 desibelin korkeudella.



Hamming-ikkunaa käytettäessä sivukaistan huiput ovat vähintään 41 dB alempana. Desibeliasteikon luku -41 dB tarkoittaa lineaarisella asteikolla lukua 0.0089. Hamming-ikkunaa käytettäessä sivukaistan korkeimmat huiput ovat näin ollen enintään vajaa sadasosa spektripiikin korkeudesta. Tämä näkyy selvästi alla olevasta kuvasta. Kuvasta näkyy myös, että pääkeila on leveämpi kuin ilman ikkunointia.



Spektriä estimoitaessa tästä ilmiöstä käytetään englanniksi nimitystä *spectral leakage*, joka tarkoittaa että osa spektrin energiasta vuotaa ympäröiville taajuuksille. Tämä näkyy ylimääräisinä huippuina sellaisilla taajuuksilla, joilla ei todellisuudessa pitäisi olla mitään. Ikkuna on tässä mielessä sitä parempi, mitä pienempi luku on sarakkeessa "korkein huippu sivukaistalla".

2.3.2 Kuuden desibelin kaistanleveys

Edellisessä esimerkissä havaittiin ikkunoinnin kasvattavan pääkeilan leveyttä. Tämän ilmiön voimakkuus voidaan mitata *kuuden desibelin kaistanleveyden* (6 dB bandwidth) avulla. Nimensä mukaisesti se kertoo kuinka leveällä alueella ikkunan spektri on kuuden desibelin rajan yläpuolella. Desibeliasteikon luku -6 dB vastaa lineaarisella asteikolla noin puolikasta amplitudissa ja se on yleisesti käytetty raja spektriä estimoitaessa.

Sivun 20 taulukossa on ilmoitettu kuuden desibelin kaistanleveys muutamalle ikkunalle. Arvo kertoo kuinka monen DFT-kertoimen alueelle spektripiikki leviää (amplitudilla 1/2). Spektrin estimaattorin *resoluutio* on sitä parempi mitä pienempi tämä arvo on. Taulukosta nähdään, että suorakulmaisella ikkunalla saadaan aina paras resoluutio, mutta tällöin ongelmaksi muodostuu voimakas värähtely sivukaistoilla. Edellisen esimerkin spektrikuvaajissa ilman ikkunointia ja Hamming-ikkunalla on kuuden desibelin kaistanleveys merkitty katkoviivoilla.

Kun käytettävissä on N näytettä, on spektrin resoluutio eli vierekkäisten pisteiden etäisyys F_s/N tai normalisoituina taajuuksina $1/N$. Näin ollen esimerkiksi suorakulmaisella ikkunalla kuuden desibelin alueen leveys on normalisoituina taajuuksina $1.21/N$. Aiemmin olleessa esimerkissä oli $N = 64$, joten kuuden desibelin kaistanleveys oli normali-

soituina taajuuksina $1.21/64 \approx 0.0189$ (sivun 21 esimerkissä väli $[\frac{8}{32} - \frac{1.21}{2\cdot 64}, \frac{8}{32} + \frac{1.21}{2\cdot 64}] \approx [0.2405, 0.2595]$). Hamming-ikkunaa käytettäessä puolestaan kuuden desibelin kaistanleveys oli $1.81/64 \approx 0.0283$ (sivun 21 esimerkissä väli $[\frac{8}{32} - \frac{1.81}{2\cdot 64}, \frac{8}{32} + \frac{1.81}{2\cdot 64}] \approx [0.2359, 0.2641]$).

Kuuden desibelin kaistanleveyttä voidaan siis aina kaventaa kertoimia lisäämällä. Tämä muistuttaa siirtymäkaistan leveyden suhdetta kertoimien määrään FIR-suodinten suunnittelussa. Molemmissa tapauksissa taajuuskaistan leveys on kääntäen verrannollinen näytteiden määrään.

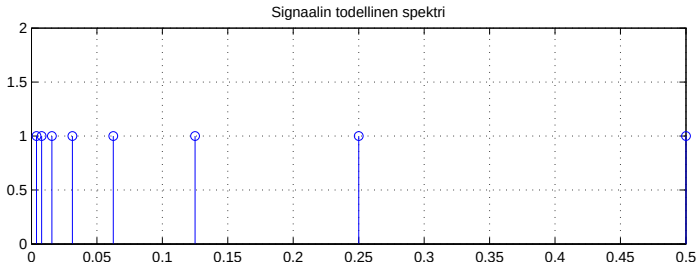
2.4 Ikkunoiden vertailua

Eri ikkunoilla saadaan siis laskettua monia niiden tehokkuutta kuvaavia parametreja, mutta vasta käytännön testit varmistavat valitun ikkunan toimivuuden. Kokeillaan lopuksi Matlabilla muutaman ikkunan tehokkuutta yksittäisten spektripiikkien tunnistamisessa. Testisignaalina käytetään eritaajuisten sinien summaa. Normalisoituina taajuudet ovat $\frac{1}{2}$, $\frac{1}{2^2}$, $\frac{1}{2^3}$, $\frac{1}{2^4}$, $\frac{1}{2^5}$, $\frac{1}{2^6}$, $\frac{1}{2^7}$ ja $\frac{1}{2^8}$. Ideana tässä on se, että näin voidaan verrata eri ikkunoiden kykyä erotella lähekkäin olevia taajuuksia toisistaan (spectral resolution). Tarkalleen ottaen testisignaali on seuraava:

$$x(n) = \sum_{k=1}^{8} \sin\left(\frac{2\pi n}{2^k} + \frac{\pi}{4}\right).$$

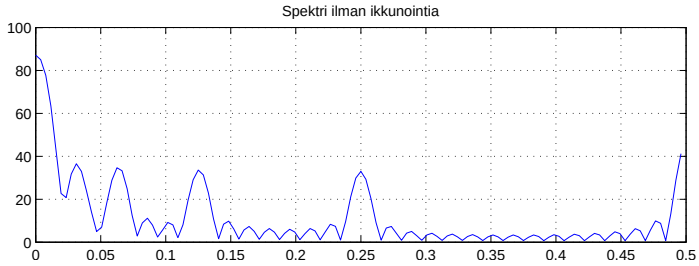
Kaikilla taajuuksilla on $\pi/4$:n suuruinen vaihesiirtymä siksi, että myös Nyquistin rajataajuudella oleva signaali voitaisiin esittää (mitä tapahtuisi ilman vaihesiirtoa?).

Spektri ideaalitapauksessa näyttäisi seuraavalta:

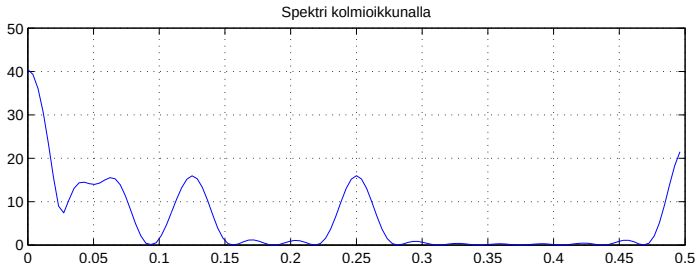


Spektriä estimoitaessa ikkunan pituudella on suuri merkitys ja valinta ei ole helppoa. Valitaan tässä esimerkissä näytteiden määräksi 64. Tämän mittainen datavektori kerrotaan pisteittäin eri ikkunafunktioilla ja lisätään loppuun vielä nollia niin paljon, että vektorista tulee 256:n mittainen.

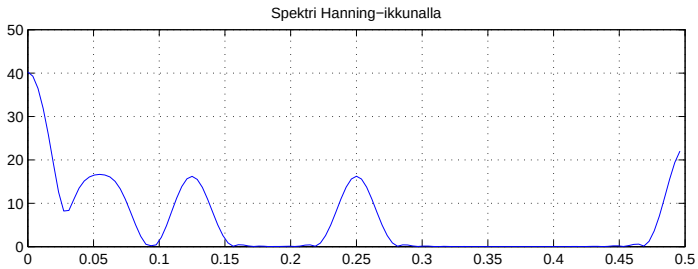
Ilman ikkunointia lopputulos näyttää seuraavalta:



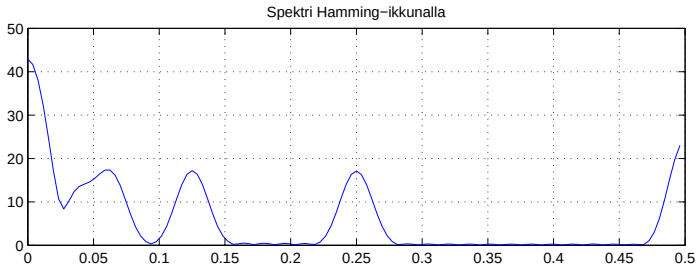
Kolmioikkunaa käytettäessä tulos on alla:



Seuraavaksi Hanning-ikkuna:



Ja lopuksi vielä tulos Hamming-ikkunaa käytettäessä.

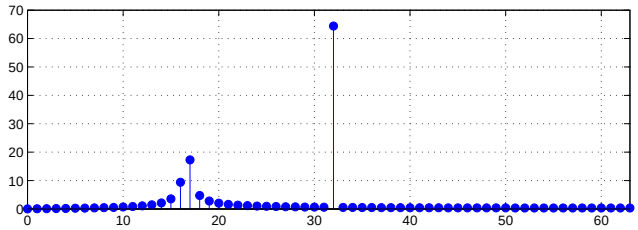


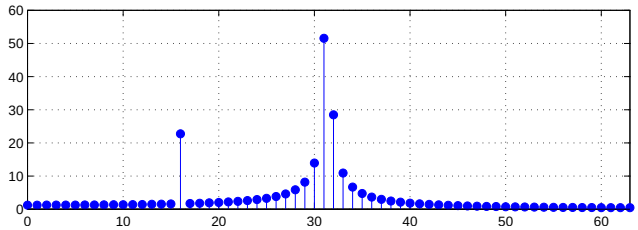
Kuvista huomataan, että ilman ikkunointia spektrissä on lukuisia huippuja sielläkin, missä todellisuudessa ei pitäisi olla mitään. Syynä on aiemmin mainittu värähtely sivukaistoilla. Toisaalta ilman ikkunointia saadaan paras spektripiikkien resoluutio. Resoluutiota voidaan arvioida kuuden desibelin kaistanleveyden avulla.

Harjoitustehtäviä

2.1. (a) Jatkuva-aikainen signaali $v(t)$ sisältää ainoastaan viittä kilohertsiä pienempiä taajuuksia. Se muunnetaan digitaaliseksi kymmenen kilohertsin näytteenottotaajuudella. Näin saadun signaalin $x(n)$ tuhannesta ensimmäisestä näytteestä lasketaan DFT ($X(n)$). Mitä alkuperäisen signaalin taajuutta vastaa DFT:n sadasviideskymmenes komponentti $X(150)$, kun $X(0)$ on nollataajuus?

(b) Alla olevassa kuvassa on kahden signaalin DFT:n itseisarvojen kuvaajat. Toinen signaali on $x_1(n) = \cos(0.5\pi n) + \frac{1}{3}\cos(0.26\pi n)$ ja toinen on $x_2(n) = \cos(0.49\pi n) + \frac{1}{3}\cos(0.25\pi n)$, missä $n = 0, 1, \ldots, 127$. Kuvassa on DFT:n itseisarvojen 64 ensimmäistä arvoa. Yhdistä nämä signaalit niille kuuluvien spektrien kanssa. Perustele.





2.2. *(Matlab)* Toteuta autokorrelaatioon perustuva spektriestimaattori. Laske menetelmälläsi spektri signaalista:

```
n = (0:99)';
x = cos(2*pi*0.125*n) + (1/sqrt(2))*sin(2*pi*0.19*n) + randn(size(n));
```

2.3. *(Matlab)* Spektrin estimointimenetelmän tehokkuutta voidaan tarkastella estimoimalla sopivasti suodatetun valkoisen kohinan spektriä. Luo 512:n näytteen mittainen satunnaisvektori (`help randn`) ja suodata se IIR-suotimella, jonka kertoimet ovat

```
b=0.01*[1,9,1];
a=[1,-1.3,0.845];
```

Nyt siis vektori `b` sisältää suotimen osoittajan (viivan yläpuolella) kertoimet ja `a` nimittäjän (viivan alla).

(a) Tee siis seuraava algoritmi:

i. Laske näytevektorin FFT.

ii. Ota tuloksesta itseisarvo (`abs`).

iii. Ota tulosvektorin alusta 256 ensimmäistä näytettä ja tulosta ne desibeliasteikolla ruudulle (lineaarisen asteikon luku x on desibeliasteikolla $20\log_{10}(x)$).

Vertaa saatua tulosta todelliseen spektriin, joka on käytetyn suotimen amplitudivaste. Tulosta amplitudivaste samaan kuvaan[2]. Tee koe kun $N = 32, 64, 128$ ja 256. Ei haittaa vaikka saatu kuvaaja olisi ylempänä tai alempana kuin todellinen spektri, muoto on tärkein.

(b) Yritä parantaa algoritmiasi kertomalla signaali ennen nollien lisäämistä Hamming-ikkunalla (`hamming(N)`).

[2]Ensin täytyy laskea amplitudivaste `[H,W]=freqz(b,a);` ja tulostaa `H`:n itseisarvon logaritmi kerrottuna 20:llä. Se saadaan samaan kuvaan antamalla ensin komento `hold on`. Lopuksi kannattaa antaa komento `hold off`.

(c) Laita koko proseduuri for-silmukan sisään ja laske sadan yo. tavalla generoidun signaalin spektrin keskiarvo. Tulosta kuvaaja ruudulle. Tämän pitäisi olla jo aika lähellä todellista spektriä.

Luku 3

Näytteenottotaajuuden muuntelu

Näytteenottotaajuuden muuntelusta A/D-muunnoksen jälkeen käytetään englannin kielessä nimitystä *multirate DSP*, mutta sille ei ole olemassa lyhyttä suomenkielistä nimeä. Yksinkertaisin menetelmä näytteenottotaajuuden muuntamiseksi on tietysti muuntaa signaali ensin analogiseksi ja tämän jälkeen muuntaa se takaisin digitaaliseksi uudella näytteenottotaajuudella. Tämä menettely johtaa kuitenkin ylimääräisiin kvantisointi- eli pyöristysvirheisiin, joita on syytä välttää. Ongelma on mahdollista ratkaista pelkästään digitaalisen signaalinkäsittelyn keinoin. Tällöin saadaan taatusti optimaalinen tulos, jossa ei ole mukana ylimääräisiä kvantisointikohinoita.

Ongelma on siis seuraava: on olemassa signaali, joka on muodostettu analogisesta signaalista näytteenottotaajuudella f. Tästä signaalista halutaan selvittää se digitaalinen signaali, joka on mahdollisimman lähellä sitä signaalia, joka olisi saatu näytteenottotaajuudella $\hat{f}$. Kouluesimerkki kyseisestä ongelmasta on muunnos CD-formaatista DAT-formaattiin. CD-levyillä data on nimittäin esitetty näytteenottotaajuudella 44.1 kHz. Sen sijaan DAT-nauhurin formaatti perustuu näytteenottotaajuuteen 48 kHz. Usein muunnos toki hoidetaan käytännössä muuntamalla signaali ensin analogiseksi ja edelleen digitaaliseksi taajudella 48 kHz. Parempaan tulokseen kuitenkin on mahdollista päästä multirate-signaalinkäsittelyn menetelmin.

Taajuuden muuntaminen on tarpeellista myös, kun signaalista on syystä tai toisesta otettu näytteitä liian korkealla taajuudella. Tällöin siis signaalissa ei ole läheskään niin suuria taajuuksia kuin näytteenottotaajuus mahdollistaa. Kun kuitenkin näytteenottotaajuus on suuri, tulee suunniteltujen suodintenkin aste tarpeettoman korkeaksi. Pienempi aste saavutetaan, jos taajuudet muunnetaan ensin pienemmäksi ja suodatetaan vasta sitten. Tämän jälkeen signaali voidaan tarvittaessa muuntaa jälleen suurempaan taajuuteen. Muuntaminen käytännössä tapahtuu kahden perusoperaation avulla:

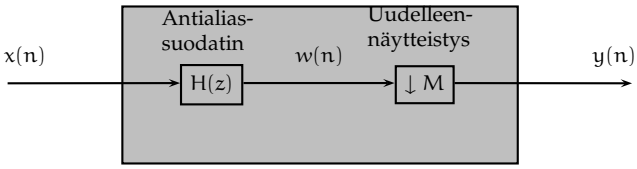
- *Interpolointi* kasvattaa signaalin näytteenottotaajuutta lisäämällä ylimääräisiä arvoja,
- *Desimointi* pienentää signaalin näytteenottotaajuutta poistamalla osan alkuperäisen signaalin arvoista.

Desimointi ja interpolointi muuntavat näytteenottotaajuutta jollain kokonaislukukertoimella (esimerkiksi 1 kHz $\mapsto$ 2kHz tai 1 kHz $\mapsto$ 0.5kHz, jne). Näitä operaatioita yhdistelemällä saadaan aikaiseksi kaikkia rationaalikertoimia vastaavat taajuusmuunnokset. Tarkastellaan lähemmin muunnosta CD-formaatista DAT-formaattiin. Käytössä on 44.1 kHz

ja tavoite olisi 48 kHz. Muunnoskerroin on $48/44.1 = 480/441 = 160/147$. Ensin taajuus korotetaan siis arvoon $44.1 \cdot 160 = 7056$ kHz interpoloimalla kertoimella 160. Tämän jälkeen signaali muunnetaan takaisin taajuuteen 48 kHz desimoimalla kertoimella 147. Merkittävää on, että muunnos tapahtuu juuri tässä järjestyksessä. Jos ensin olisi desimoitu, niin tuloksena olevassa signaalissa näytteenottotaajuus olisi ollut liian pieni, ja suurin osa signaalin informaatiosta olisi hävinnyt. Nyt tällaista informaatiohävikkiä ei tapahdu.

3.1 Desimointi eli näytteenottotaajuuden alentaminen

Oheinen kaavio esittää signaalin $x(n)$ desimointiin liittyviä vaiheita. Ennen varsinaista näytteenottotaajuuden pienentämistä signaali täytyy suodattaa alipäästösuotimella laskostumisen estämiseksi. Näin saadun signaalin näytteenottotaajuutta pienennetään jättämällä ainoastaan osa alkuperäisen signaalin arvoista jäljelle. Desimointioperaatiota merkitään alaspäin osoittavalla nuolella ja taajuuden muunnoskertoimella. Esimerkiksi näytteenottotaajuuden pudottamista kolmannekseen merkitsevä symboli on $\boxed{\downarrow 3}$. Kuvion tapauksessa alkuperäinen näytteenottotaajuus F_s pudotetaan desimoinnissa arvoon F_s/M. Tämä tehdään poimimalla joka M:s alkio desimoituun signaaliin.



Koska desimoitaessa näytteenottotaajuus pienenee, on laskostumisen vaara ilmeinen. Laskostuminenhan estetään ainostaan poistamalla signaalista taajuudet, jotka ovat suurempia kuin puolet näytteenottotaajuudesta. Tämän suorittava suodin (digital anti-aliasing filter) on siis sellainen alipäästösuodin, joka poistaa kaikki arvoa $F_s/2M$ suuremmat taajuudet. Normalisoiduissa taajuuksissa ilmaistuna alipäästösuotimen suunnitteluvaatimukset ovat seuraavat.

- Suotimen päästökaista on $[0, \frac{1}{2M} - \Delta f]$.
- Suotimen estokaista on $[\frac{1}{2M}, \frac{1}{2}]$.

Siirtymäkaistan leveys, Δf, määräytyy sovelluksen mukaan. Mitä kapeammaksi siirtymäkaista halutaan, sitä enemmän kertoimia suotimessa tulee olla. Myös vaimennusvaatimukset riippuvat sovelluksesta.

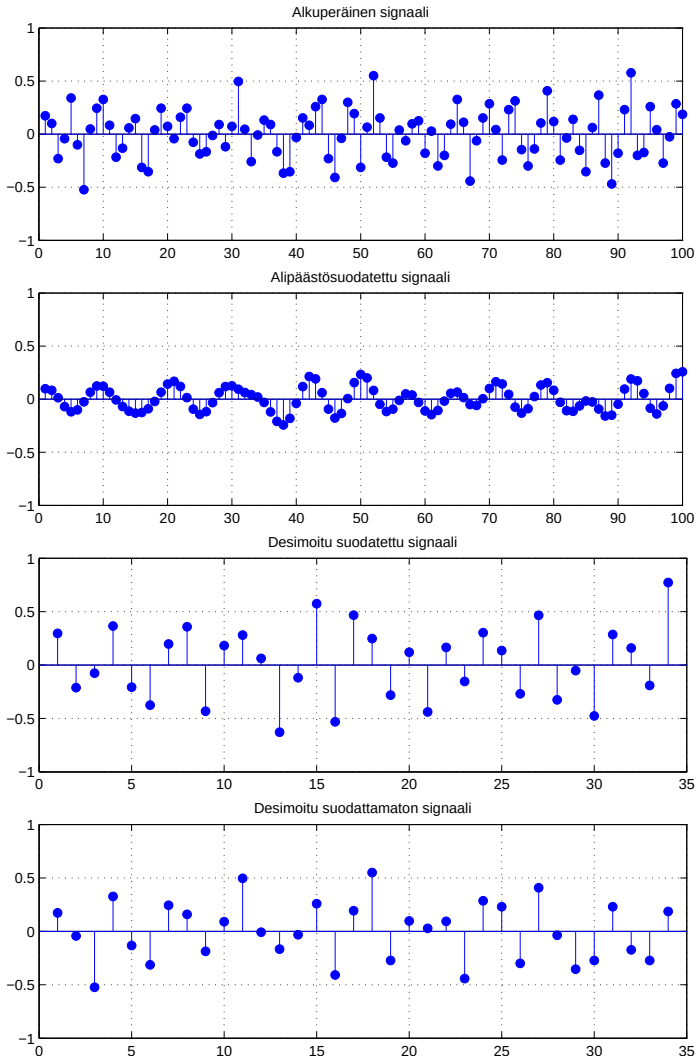
Kaavoina signaalin $x(n)$ desimointiprosessi kertoimella M signaaliksi $y(n)$ voidaan ilmaista seuraavasti:

$$\begin{aligned} w(n) &= \sum_{k=-\infty}^{\infty} h(k)x(n-k), \\ y(m) &= w(mM) = \sum_{k=-\infty}^{\infty} h(k)x(mM-k). \end{aligned}$$

Alla olevat kuvat esittävät desimoinnin vaikutusta aikatasossa. Alkuperäinen signaali $x(n)$ suodatetaan ensin alipäästösuotimella, jolloin saadaan signaali $w(n)$, joka on valmis

desimoitavaksi. Desimoitaessa kertoimella 3 otetaan uuteen signaaliin mukaan ainoastaan joka kolmas arvo. Vertailun vuoksi mukana on myös signaali, joka on saatu desimoimalla ilman alipäästösuodatusta.

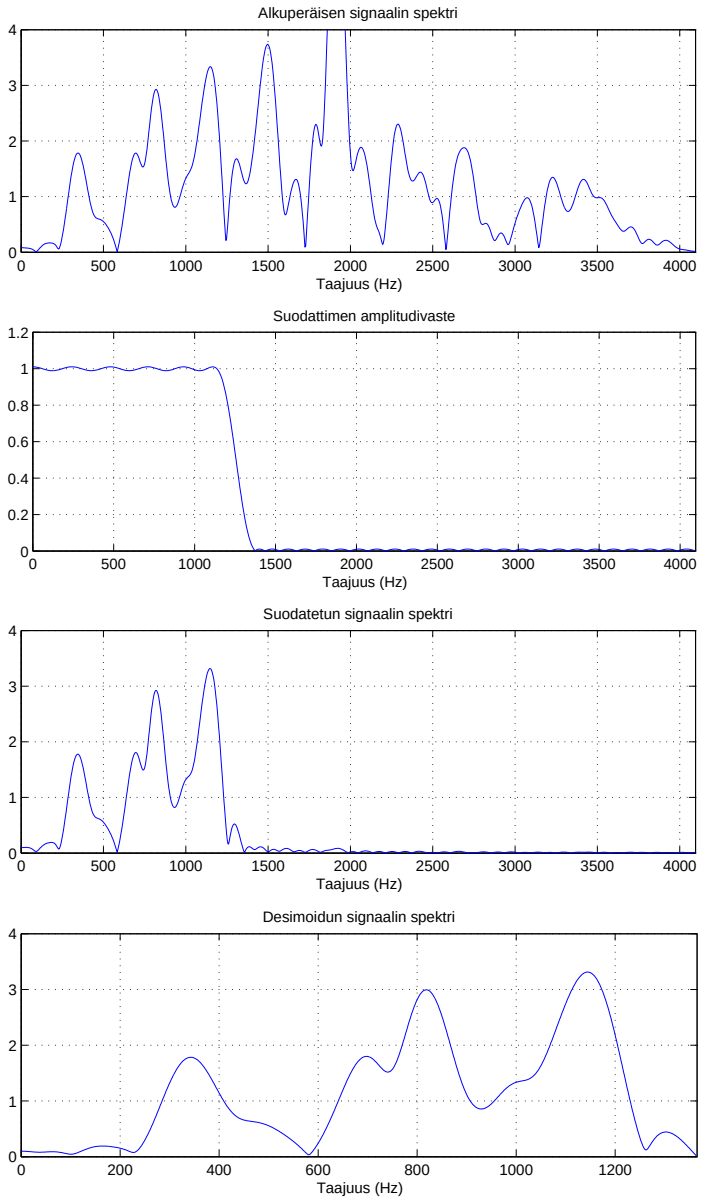
Toiseksi alimmassa kuvassa näytteet on kerrottu desimointikertoimella $M = 3$, jotta näytteiden suuruusluokka olisi sama kuin alkuperäisellä signaalilla. Ilman tätä kertolaskua näytteiden lukuarvot olisivat selvästi alkuperäisiä pienemmät, koska alipäästösuodatuksessa signaalin energia putoaa noin kolmannekseen. Yleensä tämä kertolasku jätetään pois kaavoista, koska se on helpointa toteuttaa suunnittelemalla suodin, jonka amplitudivaste päästökaistalla on M. Tällainen suodin saadaan normaalista alipäästösuotimesta yksinkertaisesti kertomalla sen jokainen kerroin luvulla M.

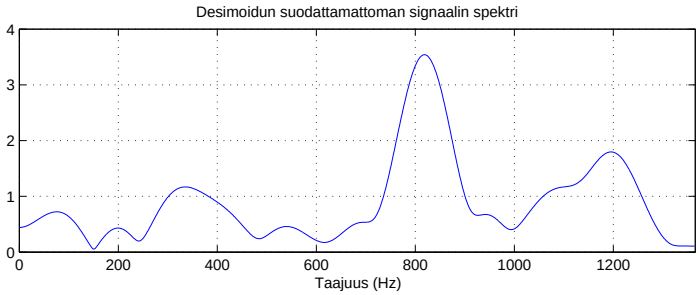


Desimointiprosessi selvinnee helpommin taajuustason kuvaajista. Ensimmäinen kuvaaja esittää alkuperäisen signaalin $x(n)$ spektriä (diskreetin Fourier-muunnoksen itseisarvoa sopivasti ikkunoituna ja interpoloituna) $|X(n)|$. Esimerkin tapauksessa näytteenottotaajuus on 8192 Hz. Tällöin siis suurin signaalin sisältämä taajuus on 4096 Hz. Kun tätä signaalia halutaan desimoida kertoimella 3, täytyy ensin poistaa $1365\frac{1}{3}$ Hz suuremmat taajuudet. Tätä varten suunnitellaan alipäästösuodin, jonka estokaista on väli $[\frac{1}{6}, \frac{1}{2}]$ (näytteenottotaajuuden suhteen normalisoituina taajuuksina) ja päästökaista nollasta johonkin

lukua 1/6 pienempään lukuun, riippuen kuinka paljon kertoimia on varaa käyttää. Oheisessa esimerkissä päästökaistan rajataajuudeksi valittiin 0.14, päästökaistan maksimivärähtelyksi 0.09dB (0.01 lineaarisella asteikolla) ja estokaistan minimivaimennukseksi 40dB (0.01 lineaarisella asteikolla). Tämän suotimen amplitudivaste on kuvattu oheisessa kuvassa. Tällä suotimella suodatettaessa saadaan signaali $w(n)$, jonka spektri on suotimen amplitudivasteen alla olevassa kuvassa. Kun signaalista $w(n)$ jätetään jäljelle vain joka kolmas arvo (ja kerrotaan näytteet luvulla 3), tuloksena on signaali $y(n)$, jonka spektri on seuraavassa kuvassa. Nyt siis uusi näytteenottotaajuus on 8192/3Hz.

Viimeinen kuvaaja esittää suodattamattoman desimoidun signaalin spektriä, jossa laskostumisilmiö on selvästi havaittavissa. Siinä oleva spektri poikkeaa alkuperäisen signaalin vastaavasta kaistasta, vaikka ne halutaan mahdollisimman lähelle toisiaan.





3.2 Näytteenottotaajuuden pienentäminen useassa vaiheessa

Desimaattoreiden toteutusta voidaan nopeuttaa pudottamalla näytteenottotaajuutta useammassa vaiheessa. Esimerkiksi näytteenottotaajuuden pudotus yhteen kymmenesosaan alkuperäisestä voidaan tehdä yhdessä vaiheessa kertoimella kymmenen tai kahdessa vaiheessa ensin kertoimella 5 ja sitten kertoimella 2. Kolmas mahdollisuus on pudottaa ensin kertoimella 2 ja sitten kertoimella 5. Kaikilla näillä menetelmillä tarvitaan eri määrä kertoimia, ja useimmiten suora pudotus kertoimella 10 tarvitsee enemmän kuin useammassa vaiheessa tehdyt operaatiot. Alla on esitetty kaaviot eri menetelmistä, kun lähtötaajuus on 20 kHz ja tavoite on 2 kHz. Oletetaan lisäksi, että signaalista täytyy säilyttää taajuudet 900 Hertsiin asti ja että suunnittelussa käytetään FIR-suodatinta ja Hamming-ikkunaa.

- Yhdessä vaiheessa: $x(n) \rightarrow \boxed{H(z)} \rightarrow \boxed{\downarrow 10} \rightarrow y(n)$
  Nyt päästökaista on väli $[0, 0.9]$ kHz ja estokaista $[1, 10]$ kHz. Siirtymäkaista normalisoituna on näin ollen väli $[0.045, 0.05]$, joten kertoimia tarvitaan $N = 3.3/\Delta f = 3.3/0.005 \approx$ **661**.

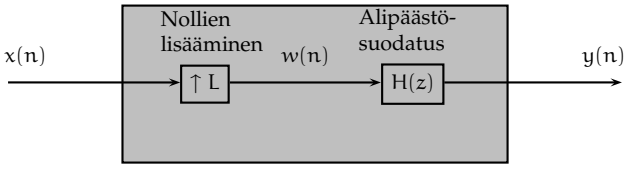
- Kahdessa vaiheessa: $x(n) \rightarrow \boxed{H_1(z)} \rightarrow \boxed{\downarrow 5} \rightarrow \boxed{H_2(z)} \rightarrow \boxed{\downarrow 2} \rightarrow y(n)$
  Nyt suotimen $H_1(z)$ päästökaista on väli $[0, 0.9]$ kHz ja estokaista $[2, 10]$ kHz. Siirtymäkaista normalisoituna on näin ollen väli $[0.045, 0.1]$, joten kertoimia tarvitaan $N_1 = 3.3/0.055 \approx 61$. Suotimen $H_2(z)$ päästökaista puolestaan on väli $[0, 0.9]$ kHz ja estokaista $[1, 2]$ kHz. Siirtymäkaista normalisoituna näytteenottotaajuudella 4 kHz on näin ollen väli $[0.225, 0.25]$, joten kertoimia tarvitaan $N_2 = 3.3/0.0250 \approx 133$. Yhteensä kertoimia tarvitaan siis **194**.

- Kahdessa vaiheessa: $x(n) \rightarrow \boxed{H_1(z)} \rightarrow \boxed{\downarrow 2} \rightarrow \boxed{H_2(z)} \rightarrow \boxed{\downarrow 5} \rightarrow y(n)$
  Suotimen $H_1(z)$ päästökaista on väli $[0, 0.9]$ kHz ja estokaista $[5, 10]$ kHz. Siirtymäkaista normalisoituna on näin ollen väli $[0.045, 0.25]$, joten kertoimia tarvitaan $N_1 = 3.3/0.205 \approx 17$. Suotimen $H_2(z)$ päästökaista puolestaan on väli $[0, 0.9]$ kHz ja estokaista $[1, 5]$ kHz. Siirtymäkaista normalisoituna näytteenottotaajuudella 10 kHz on näin ollen väli $[0.09, 0.1]$, joten kertoimia tarvitaan $N_2 = 3.3/0.01 \approx 331$. Yhteensä kertoimia tarvitaan siis **348**.

Näin ollen tehtävä on viisainta suorittaa kahdessa vaiheessa kertoimilla 5 ja 2 (tässä järjestyksessä). Yleisesti pitää paikkansa, että desimointikertoimet kannattaa sijoittaa laskevaan järjestykseen. Siksi järjestys $x(n) \rightarrow \boxed{H_1(z)} \rightarrow \boxed{\downarrow 2} \rightarrow \boxed{H_2(z)} \rightarrow \boxed{\downarrow 5} \rightarrow y(n)$ tuottaa aina enemmän kertoimia kuin $x(n) \rightarrow \boxed{H_1(z)} \rightarrow \boxed{\downarrow 5} \rightarrow \boxed{H_2(z)} \rightarrow \boxed{\downarrow 2} \rightarrow y(n)$, ja se olisi voitu alun perinkin jättää tarkastelematta.

3.3 Interpolointi eli näytteenottotaajuuden nostaminen

Signaalin interpoloinnin tarkoituksena on saada aikaiseksi signaali, joka vastaa alkuperäistä, mutta jonka näytteenottotaajuus on suurempi. Oheinen kaavio esittää lohkokaaviota interpolointioperaatiosta. Toisin kuin desimoinnissa, nyt ei tarvita esisuodatusta, koska näytteenottotaajuutta nostettaessa kaikki alkuperäiset taajuudet voidaan toki esittää.



Ensimmäinen operaatio on näytteenottotaajuuden nostaminen L-kertaiseksi. Tätä operaatiota merkitään nuolella ylöspäin ja interpolointikertoimella L, siis $\boxed{\uparrow L}$. Tämän operaation toteuttamisessa on lukuisia vaihtoehtoja; kuinka määritetään uudet ylimääräiset arvot? Matemaatikko alkaisi tässä tapauksessa luultavasti sovittaa polynomeja tai splinejä saadakseen uusia arvoja olemassaolevien arvojen välille. Signaalinkäsittelyssä tilanne hoidetaan kuitenkin toisin: tuntemattomien arvojen tilalle sijoitetaan nollat ja näin saatu signaali suodatetaan alipäästösuotimella. Nollia lisättäessä mukaan tulee hyvin suuria taajuuksia.

Suurten taajuuksien lisäämisen vaikutus on poistettava, ja se luonnollisesti tapahtuu alipäästösuodatuksella. Alkuperäisessä signaalissa suurin taajuus on $F_s/2$. Interpoloitaessa näytteenottotaajuuteen LF_s, suurin esitettävissä oleva taajuus on $LF_s/2$. Taajuutta $F_s/2$ suuremmat taajuudet on poistettava, koska ne ovat nollien lisäämisen tulosta.

Nollien lisäämisen jälkeen saatu signaali (näytteenottotaajuus LF_s) suodatetaan siis alipäästösuotimella, joka säilyttää taajuudet väliltä $[0, F_s/2]$ ja poistaa tätä suuremmat taajuudet. Koska nyt näytteenottotaajuus on interpoloinnin seurauksena LF_s, niin suotimen vaatimukset on normalisoitava tämän luvun suhteen. On siis suunniteltava alipäästösuodin, jonka vaatimukset ovat:

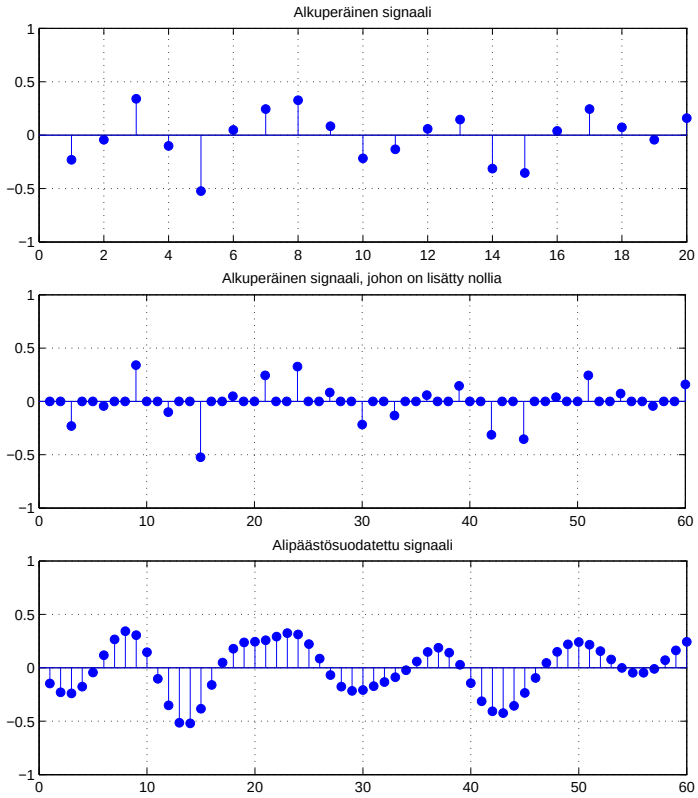
- päästökaista on normalisoituina taajuuksina ilmaistuna väli $[0, (F_s/2)/(LF_s) - \Delta f] = \left[0, \frac{1}{2L} - \Delta f\right]$. Siirtymäkaistan leveys Δf riippuu jälleen sovellutuksesta jossa interpolointia on tarkoitus soveltaa (Δf määrää osaltaan kertoimien määrän).

- estokaista on väli $[(F_s/2)/(LF_s), 1/2] = \left[\frac{1}{2L}, \frac{1}{2}\right]$.

Myös vaimennusvaatimukset määräytyvät enimmäkseen sovellutuksen mukaan, eikä niiden valinnasta voida antaa mitään yleistä ohjenuoraa. Koska alkuperäiseen signaaliin sijoitetaan $L - 1$ nollaa jokaista signaalin arvoa kohden, signaalin amplitudi putoaa suodatettaessa yhteen L:n osaan. Siksi suodatuksen jälkeen (tai sitä ennen) signaalin arvot on syytä kertoa luvulla L.

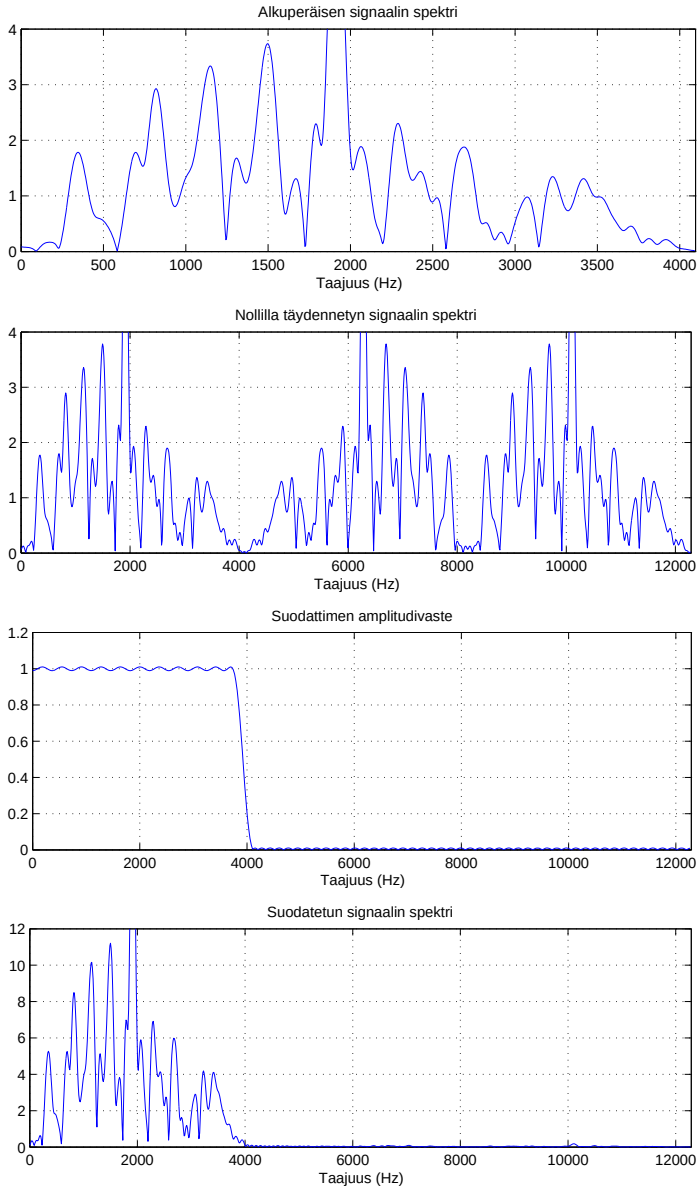
Kaavoilla esitettynä signaalin $x(n)$ interpolointiprosessi kertoimella L signaaliksi $y(n)$ on seuraava:

$$
\begin{aligned}
w(m) &= \begin{cases} x(m/L), & \text{kun } m = 0, \pm L, \pm 2L, \ldots \\ 0, & \text{muulloin,} \end{cases} \\
y(n) &= \sum_{k=-\infty}^{\infty} h(k) w(n-k).
\end{aligned}
$$



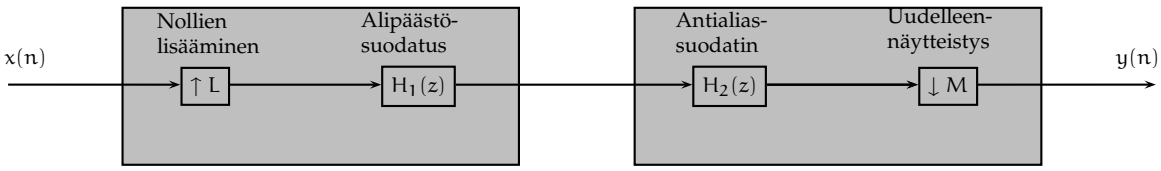
Yllä oleva kuva esittää esimerkkitapausta, jossa signaalin näytteenottotaajuus interpoloidaan kolminkertaiseksi ($L = 3$). Ensin signaaliin $x(n)$ jokaisen kahden peräkkäisen arvon väliin sijoitetaan $L - 1 = 2$ nollaa, jolloin saadaan signaali $w(m)$. Tämä suodatetaan alipäästösuotimella, jonka estokaista on $[\frac{1}{6}, 0.5]$, päästökaista $[0, 0.1526]$ ja maksimivärähtelyarvot samat kuin desimointiesimerkissä. Tuloksena saadaan ulostulosignaali $y(m)$. Kuten desimoinnin tapauksessakin, tämä signaali on lopuksi vielä kerrottava luvulla $L = 3$, jotta signaalin energia säilyisi. Vaihtoehtoisesti käytetyn alipäästösuotimen kertoimet voidaan kertoa luvulla L, jolloin päästökaistan taajuudet vahvistuvat kolminkertaisiksi. Tulos on alimmassa edellä olleista kuvista.

Taajuustasossa tilanne on oheisen kuvan mukainen. Ylin kuva esittää alkuperäistä signaalia, jonka näytteenottotaajuus on esimerkissämme 8192 Hz. Lisättäessä signaaliin nollia, saavutetaan signaali, jonka spektri $|W(e^{i\omega})|$ on seuraavassa kuvassa. Tästä on poistettava välillä 4096 Hz – 12288 Hz olevat taajuudet esimerkiksi suotimella, jonka amplitudivaste $|H(e^{i\omega})|$ on seuraavassa kuvassa. Tuloksena on signaali $y(n)$, jonka spektri on alimmassa kuvassa.

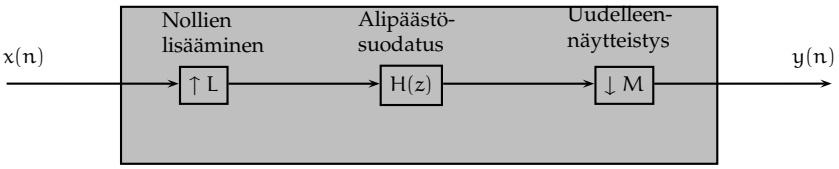


3.4 Näytteenottotaajuuden muunnos rationaalikertoimella

Kuten aiemmin mainittiin, rationaalikertoiminen näytteenottotaajuuden muunnos saadaan yhdistämällä interpolointi ja desimointi, tässä järjestyksessä. Jos siis näytteenottotaajuus halutaan $\frac{L}{M}$-kertaiseksi, interpoloidaan signaali ensin L-kertaiseksi ja desimoidaan tämän jälkeen kertoimella M. Tässä kannattaa laskennan säästämiseksi supistaa murtoluku $\frac{L}{M}$ niin pitkälle kuin mahdollista. Järjestelmän lohkokaavio on alla olevan kuvan mukainen.



Kaaviosta käy ilmi, että järjestelmässä on kaksi alipäästösuodatusta peräkkäin. Toinen näistä voidaan poistaa ja jättää jäljelle se, jonka suodatusvaatimukset ovat tiukemmat. Jos esimerkiksi kerroin $\frac{L}{M} = \frac{4}{3}$, tulee suotimen $H_1(z)$ päästökaistaksi $[0, \frac{1}{2L} - \Delta f] = [0, \frac{1}{8} - \Delta f]$ ja estokaistaksi $[\frac{1}{8}, \frac{1}{2}]$. Suotimen $H_2(z)$ vastaavat arvot ovat $[0, \frac{1}{2M} - \Delta f] = [0, \frac{1}{6} - \Delta f]$ ja $[\frac{1}{6}, \frac{1}{2}]$. Suodin $H_1(z)$ poistaa siis laajemman taajuusalueen kuin $H_2(z)$, joten pelkkä $H_1(z)$ riittää. Tässä yhteydessä täytyy luonnollisesti huomioida myös vaimennusvaatimusten toteutuminen. Alla oleva kaavio esittää näin saatavaa yksinkertaistettua järjestelmää.



3.5 Interpoloinnin käyttö D/A-muunnoksessa

Seitsemänkymmentäluvun puolivälin jälkeen alettiin tutkia mahdollisuuksia musiikin ja muun äänimateriaalin digitaaliseen tallentamiseen. Lopputuloksena syntyi CD-soitin, joka on nykyään yleisesti käytössä ympäri maailmaa. Seuraavassa tarkastellaan sen joitakin teknisiä yksityiskohtia ja erityisesti sen D/A-muunnosvaihetta, jossa digitaalinen signaali muunnetaan analogiseksi. Lisätietoja löytyy esimerkiksi Steiglitzin kirjasta[1] sekä Philipsin julkaisusta[2]. Kohinanmuokkausta tarkastellaan yksityiskohtaisemmin IEEE:n artikkelikokoelmassa[3].

CD-levyn halkaisija on 12 cm ja raidat ovat 1.6 mikrometrin etäisyydellä toisistaan. Audio-CD:n sisältämä informaatio koostuu 16 bitin näytteistä, joita on otettu 44100 Hertzin näytteenottotaajuudella. Koska stereoääni tarvitsee kaksi kanavaa, saadaan bittimääräksi $2 \times 16 \times 44100 \approx 1.41$ miljoonaa bittiä sekunnissa. Todellinen levyllä oleva bittimäärä on kuitenkin noin kolminkertainen. Ylimääräiset bitit ovat enimmäkseen virheenkorjausta varten. Pieni osa biteistä tarvitaan myös kappaleiden pituuksien ja mahdollisesti myös nimien tallentamiseen. Multirate-tekniikan yhteydessä mielenkiintoisin vaihe on CD-soittimen D/A-muunnos.

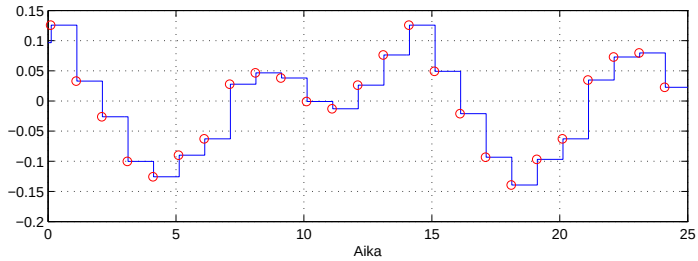
3.5.1 Nollannen asteen pitopiiri

Yksinkertaisimmillaan muunnos digitaalisesta signaalista analogiseksi tapahtuu nollannen kertaluvun pitopiirillä (engl. zero-order hold; ZOH tai sample-and-hold; S/H). Tällöin analogisen signaalin arvoksi asetetaan viimeksi tullut digitaalisen signaalin arvo. Oheisessa kuvassa esitetään erään digitaalisen signaalin muunnos analogiseksi nollannen kertaluvun pitopiirillä. Ympyrät kuvaavat digitaalisen signaalin arvoja ja viiva on analoginen approksimaatio.

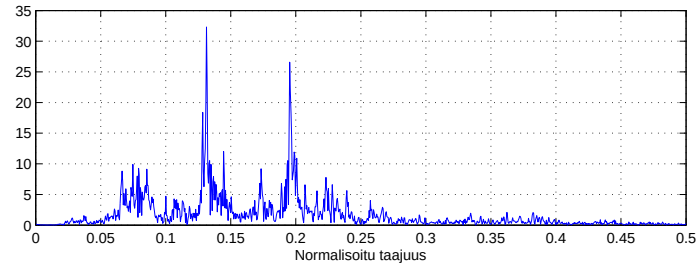
[1] Ken Steiglitz, *A Digital Signal Processing Primer, with Applications to Digital Audio and Computer Music*, Addison-Wesley, Menlo Park, CA, 1996.

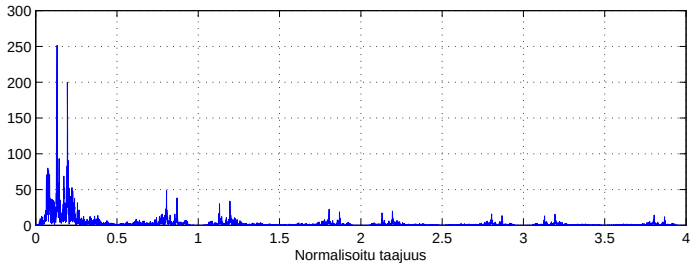
[2] "Compact Disc Digital Audio,"*Philips Technical Review*, vol. 40, no. 6, 1982.

[3] *Oversampling Delta-Sigma Data Converters: Theory, Design and Simulation*, toim. J. Candy ja G. Temes, IEEE Press, New York, 1992.



Äänisignaalin ollessa kyseessä mielenkiintoisinta on kuinka analogisen signaalin spektri vastaa alkuperäisen digitaalisen signaalin spektriä. Vastaus löytyy seuraavan kuvan simuloiduista spektreistä.





Ylemmässä kuvassa on digitaalisen signaalin spektri. Vaaka-akseli esittää taajuuksia näytteenottotaajuuden suhteen normalisoituina. Alempi kuvaaja esittää analogisen signaalin spektriä nollannen asteen pitopiirin jälkeen. Nyt signaalin energia jakautuu paljon laajemmalle alueelle siten, että alkuperäinen spektri (välillä $[0, 0.5]$) monistuu suuremmille taajuuksille. Välillä $[0.5, 1]$ on vaimentunut peilikuva alkuperäisestä spektristä, välillä $[1, 1.5]$ on alkuperäisen spektrin vaimennettu kopio, välillä $[1.5, 2]$ taas peilikuva vaimennettuna, jne. Kuvassa on esitetty ainoastaan alkuperäinen spektri ja sen seitsemän kopiota, mutta itse asiassa kopioita on äärettömän monta (pienemmillä ja pienemmillä energioilla).

Nollannen asteen pitopiirin käyttäytyminen taajuustasossa voidaan esittää analyyttisesti. Koska kyseessä on jatkuva-aikainen suodin, analyysimenetelmä poikkeaa hieman johdatuskursseilla tarkastelluista. Pitopiiri voidaan ajatella suotimeksi, jonka impulssivaste on jatkuva funktio

$$h(t) = \begin{cases} 1, & \text{kun } 0 \leq t < T, \\ 0, & \text{muulloin,} \end{cases}$$

missä T on kahden näytteenottohetken välinen aikaero (CD-soittimella $1/44100$ s). Analogisen suotimen tapauksessa konvoluutio määritellään kaavalla

$$y(t) = \int_{-\infty}^{\infty} h(u)x(t-u)du$$

eli tässä tapauksessa

$$y(t) = \int_0^T x(t-u)\,du.$$

Tällaisen suotimen taajuusvaste on

$$\begin{aligned} H(e^{i\omega}) &= \int_{-\infty}^{\infty} h(t)e^{-i\omega t}\,dt \\ &= \int_0^T e^{-i\omega t}\,dt \\ &= \Big/_0^T \frac{1}{-i\omega}e^{-i\omega t} \\ &= \frac{e^{-i\omega T}-1}{-i\omega}. \end{aligned}$$

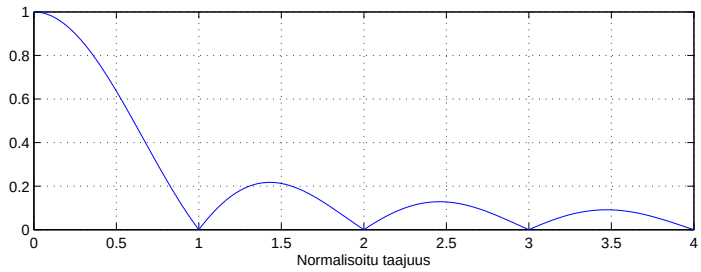
Viimeisin lauseke voidaan sieventää muotoon

$$H(e^{i\omega}) = Te^{-i\omega T/2}\,\mathrm{sinc}(\omega T/2),$$

jolloin amplitudivasteeksi tulee

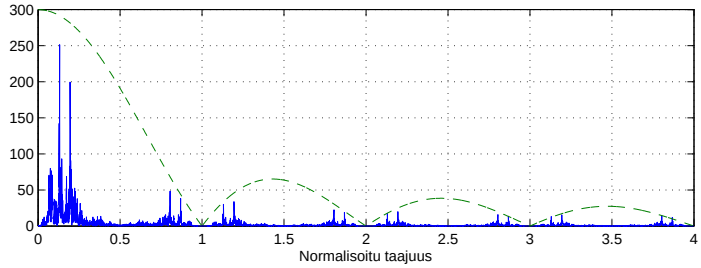
$$|H(e^{i\omega})| = T|\,\mathrm{sinc}(\omega T/2)|.$$

Tämän funktion kuvaaja on alla (nyt $T = 1$).



Digitaalisen signaalin näytteenottotaajuus on yo. asteikolla yksi ja Nyquistin rajataajuus näinollen kohdassa 1/2. Jokainen kuvaajan huippukohta tuottaa yhden vaimennetun kopion digitaalisen signaalin spektristä.

Amplitudivasteen kuvaaja sovitettuna signaalin spektriin on alla. Visualisointisyistä amplitudivaste on nyt kerrottu kolmellasadalla.



Näin havaittiin, että pelkkä pitopiiri tuottaa ylimääräisiä taajuuksia analogiseen signaaliin. CD-soittimen tapauksessa ylimääräinen energia esiintyy yli 22.05 kilohertsin taajuuksilla, joita ihmiskorva ei kuule. Ylimääräinen energia korkeilla taajuuksilla saattaa kuitenkin rasittaa vahvistinlaitteistoa, joten se on hyvä poistaa.

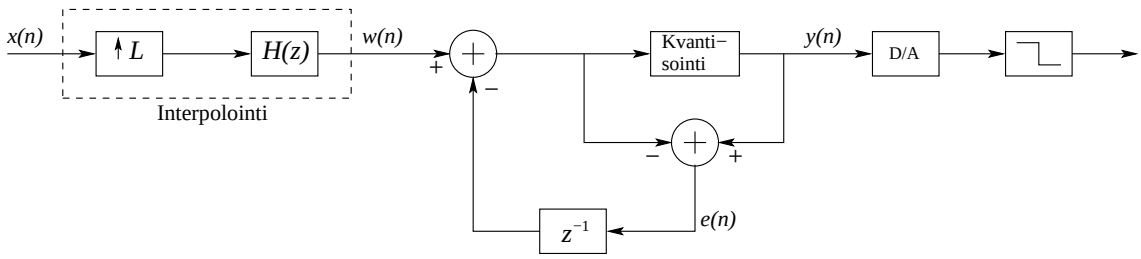
Korkeat taajuudet voitaisiin poistaa analogisella alipäästösuotimella, jonka päästökaista on esimerkiksi väli $[0, 20]$ kHz ja estokaista $[22.05, \infty)$ kHz (analogisella suotimilla käsiteltävillä taajuuksilla ei ole ylärajaa). Tällöin siirtymäkaistasta tulee kuitenkin verraten kapea, jolloin analogisesta suotimesta tulee hankala suunniteltava ja helposti kallis.

Multirate-menetelmiä käyttäen suodatus voidaan tehdä kahdessa vaiheessa, jolloin suotimista tulee yksinkertaisempia. Ensimmäisessä vaiheessa nostetaan digitaalisen signaalin näytteenottotaajuus nelinkertaiseksi (176.4 kHz). Tämä tapahtuu lisäämällä kahden näytearvon väliin kolme nollaa ja suodattamalla tulos alipäästösuotimella, jonka päästökaista on väli $[0, 20]$ kHz ja estokaista $[22.05, 88.2]$ kHz. Tämän suotimen toteutus riippuu halutusta tarkkuudesta, mutta esimerkiksi Philipsin kuvauksessa käytetään FIR-suodinta, jossa on 96 kerrointa. Kukin kerroin esitetään kahdentoista bitin tarkkuudella. Päästökaistalla suotimen amplitudivaste on nouseva niin, että nollataajuuden lähellä se on hieman alle nollan desibelin ja 20 kHz:n lähellä hieman yli nollan desibelin. Näin pyritään kompensoimaan korkeampien taajuuksien pientä vaimenemista nollannen asteen pitopiirissä.

Toisessa vaiheessa digitaalinen signaali muunnetaan analogiseksi nollannen asteen pitopiirillä ja suodatetaan lopuksi analogisella suotimella, joka poistaa suurille taajuuksille tulevan ylimääräisen energian. Nyt analogisen suotimen vaatimukset on helppo toteuttaa: päästökaista $[0, 20]$ kHz, estokaista $[88.2, \infty)$ kHz. Siirtymäkaistan leveys on nyt yli 30-kertainen aikaisempaan verrattuna. Itse asiassa siirtokaista voitaisiin venyttää yli 150:een kilohertsiin ilman ongelmia (miksi?).

3.5.2 Kohinanmuokkaus

Digitaalilaitteita käytettäessä informaation kaikkein luonnollisin esitysmuoto on binäärinen. Binääridatalla tehtävät operaatiot ovat nopeita ja niitä käyttävät laitteet yksinkertaisia suunnitella. Myös CD-soittimen alunperin 16-bittinen data muunnetaan nykyisissä soittimissa ensin binääriseksi ja vasta tämän jälkeen analogiseksi. Tällöin D/A-muunnin tuottaa vain kahta eri jännitetasoa ja se saadaan rakenteeltaan yksinkertaiseksi. Normaalisti signaalin muunnos binääriseksi aiheuttaisi voimakkaan kvantisointikohinan signaaliin, mutta siitä päästään eroon nostamalla signaalin näytteenottotaajuutta sekä siirtämällä kvantisointikohinaa korkeammille taajuuksille. Tällaisista menetelmistä käytetään nimeä *kohinanmuokkaus* (*engl. noise shaping*), ja ideana on nimenomaan siirtää kohinan energiaa suuremmille taajuuksille, josta se on helppo poistaa alipäästösuodatuksella.



Yksinkertaisin kohinanmuokkausmenettely on yllä olevan lohkokaavion mukainen. Ensimmäisessä vaiheessa signaali $x(n)$ interpoloidaan näytteenottotaajuudeltaan moninkertaiseksi. Tulos $w(n)$ kvantisoidaan ja järjestelmä laittaa muistiin syntyneen kvantisointivirheen $e(n)$. Kvantisointivirheen tallentaminen auttaa tasapainottamaan virhettä seuraavilla kierroksilla.

Yleensä kvantisointia mallinnetaan lisäämällä kvantisoitavaan signaaliin virhesignaali $e(n)$. Yllä olevan kuvan tapauksessa lohkokaaviosta saadaan yhtälö

$$y(n) = w(n) + e(n) - e(n-1).$$

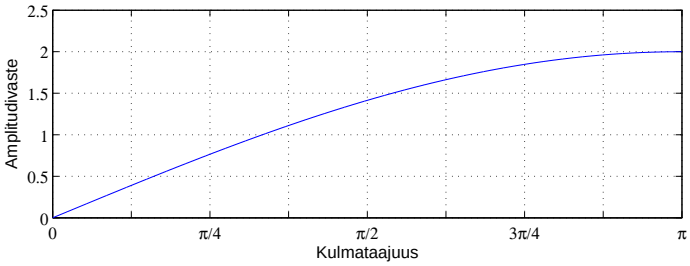
Ottamalla z-muunnokset puolittain saadaan yhtälö

$$Y(z) = W(z) + E(z)(1 - z^{-1}).$$

Näin ollen signaali $w(n)$ menee sellaisenaan järjestelmän läpi (ei suodatusta). Sen sijaan virhesignaali $e(n)$ kulkee lineaarisen järjestelmän $H(z) = 1 - z^{-1}$ läpi. Käytetään tästä ulostulosignaalista jatkossa merkintää $d(n)$:

$$d(n) = e(n) - e(n-1).$$

Järjestelmän $H(z)$ taajuusvaste on $H(e^{i\omega}) = 1 - e^{-i\omega}$, jonka itseisarvon kuvaaja on alla. Melko helposti voidaan osoittaa, että $|1 - e^{-i\omega}| = 2\sin(\frac{\omega}{2})$.



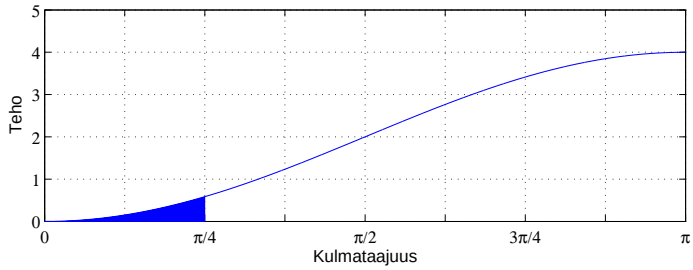
Kuviosta nähdään, että kohinan pienet taajuudet vaimenevat ja suuret taajuudet vahvistuvat. Koska signaali $w(n)$ on interpoloitu versio alkuperäisestä signaalista, sijaitsevat tärkeät taajuudet nimenomaan pienillä taajuuksilla, jossa kohina on vaimentunut. Signaalin $y(n)$ SNR riippuu nyt interpolointikertoimesta L ja siitä, kuinka moneen bittiin signaali kvantisoidaan.

Kohinan määrä järjestelmän $H(z)$ jälkeen voidaan laskea analyyttisesti käyttämällä hyväksi tietoa, että sen tehospektri riippuu valkoisen kohinan tapauksessa suoraan järjestelmän $H(z)$ taajuusvasteesta:

$$\begin{aligned} P_{dd}(e^{i\omega}) &= |H(e^{i\omega})|^2 \sigma_e^2 \\ &= \left(2\sin\left(\frac{\omega}{2}\right)\right)^2 \sigma_e^2 \\ &= 4\sigma_e^2 \sin^2\left(\frac{\omega}{2}\right), \end{aligned}$$

missä $\sigma_e^2 = 2^{-2b}/12$ ja b on bittien määrä kvantisoinnin jälkeen. Koska signaali sijaitsee interpoloinnin jälkeen taajuuksilla $\omega \in [0, \pi/L]$, meidän tarvitsee tietää tällä taajuusalueella olevan kohinan teho.

Esimerkiksi tapauksessa $L = 4$ tilanne olisi alla olevan kuvan mukainen. Käyrän alle jäävä merkitty alue vastaa signaalin kanssa samalla taajuusalueella olevan kohinan tehoa.



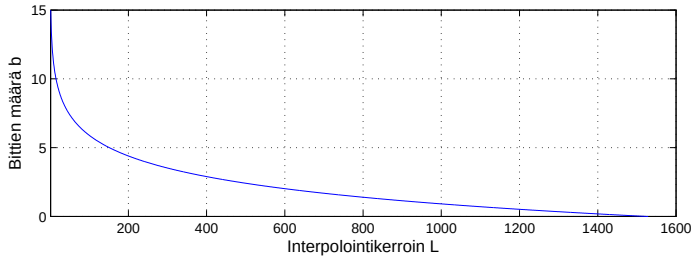
Se saadaan integroimalla,

$$\begin{aligned}\frac{1}{\pi}\int_0^{\pi/L} 4\sigma_e^2 \sin^2(\frac{\omega}{2})\,d\omega &= 4\cdot\frac{2^{-2b}}{12\pi}\int_0^{\pi/L}\sin^2(\frac{\omega}{2})\,d\omega \\ &= \frac{2^{-2b}}{3\pi}\left(\frac{\pi}{2L}-\frac{1}{2}\sin(\frac{\pi}{L})\right) \\ &= \frac{2^{-2b}}{6}\left(\frac{1}{L}-\frac{\sin(\pi/L)}{\pi}\right).\end{aligned}$$

Nyt voidaan kysyä, montako bittiä säästetään kun interpoloidaan tietyllä kertoimella L. Esimerkiksi CD-soittimen tapauksessa bittimäärä on $15+1$, ja kvantisointikohinan teho näin ollen $\frac{2^{-30}}{12}$. Samaan kvantisointikohinaan päästään interpoloinnin ja kohinanmuokkauksen avulla interpolointikertoimella L ratkaisemalla tarvittava bittimäärä b yhtälöstä

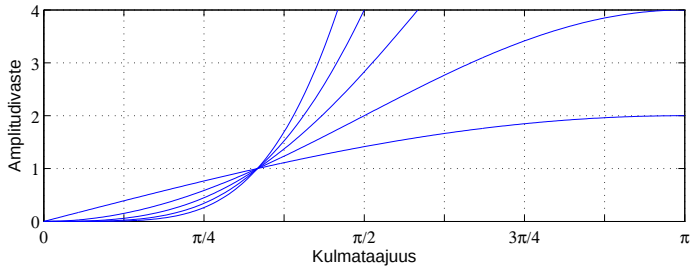
$$\frac{2^{-2b}}{6}\left(\frac{1}{L}-\frac{\sin(\pi/L)}{\pi}\right)=\frac{2^{-30}}{12}.$$

Bittien tarve eri kertoimilla on esitetty alla olevassa kuvassa. Arvoissa ei ole mukana merkkibittiä.



Kuviosta nähdään interpolointikertoimien olevan edelleenkin varsin suuria. Täysin binääriseen tilanteeseen ($b=0$) tarvitaan nyt 1523-kertainen interpolointi, mikä on liikaa. Järjestelmää on siis kehitettävä edelleen.

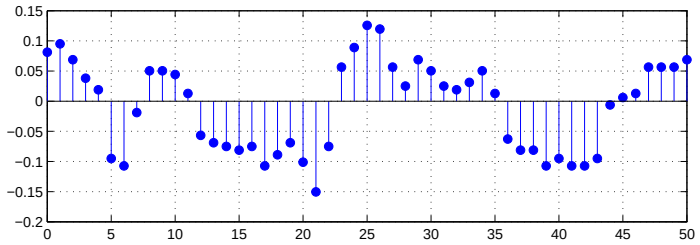
Kohinanmuokkaus tapahtuu siis yksinkertaisimmillaan viemällä kvantisointikohina järjestelmän $H(z)=1-z^{-1}$ läpi. Tulosta voidaan parantaa korottamalla siirtofunktio toiseen tai suurempaan potenssiin. Toisen asteen kohinanmuokkaus käyttää siis järjestelmää $H(z)=(1-z^{-1})^2=1-2z^{-1}+z^{-2}$, joka on melko helppo liittää aiemmin esillä olleeseen lohkokaavioon. Yleisemmin p:nnen asteen kohinanmuokkaus vie kohinan järjestelmän $H(z)=(1-z^{-1})^p$ läpi. Tällöin amplitudivasteeksi tulee $|H(e^{i\omega})|=(2\sin(\frac{\omega}{2}))^p$ ja amplitudivasteen kuvaajat arvoilla $p=1,2,\ldots,5$ on esitetty alla. Käyrät kohtaavat pisteessä ω, jossa on voimassa yhtälö $2\sin(\omega/2)=1$, eli pisteessä $\omega=\pi/3$, joka on normalisoituina taajuuksina $f=\frac{\omega}{2\pi}=\frac{1}{6}$.



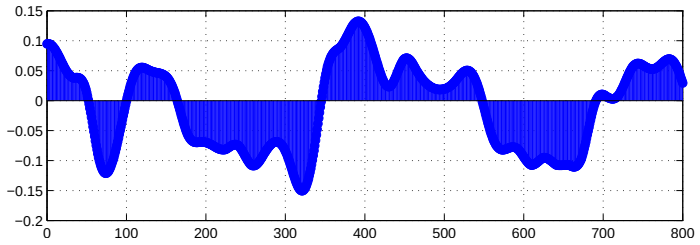
Näin ollen p:nnen asteen kohinanmuokkaimen kvantisointikohinan varianssi on

$$\frac{2^{2(p-b)}}{12\pi}\int_0^{\pi/L}\sin^{2p}(\frac{\omega}{2})\,d\omega.$$

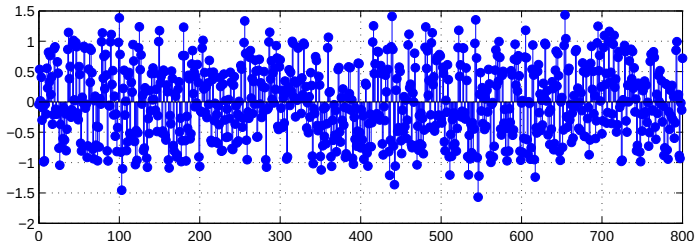
Tutkitaan lopuksi toisen asteen kohinanmuokkaajaa, kun testisignaalin alkuperäinen näytteenottotaajuus on 8192 Hz, $L = 4$ ja $b = 0$ eli kvantisoinnin tulos on binäärinen. Interpolointikerroin L on visualisointisyistä melko pieni. Alla olevassa kuvassa on pätkä testisignaalia.



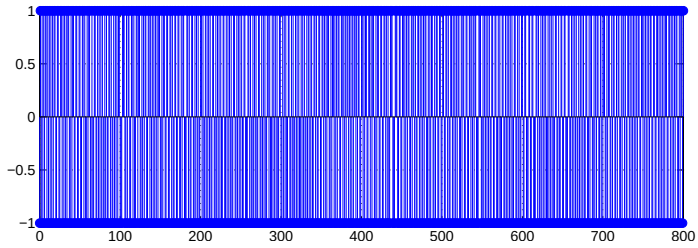
Ensimmäisessä vaiheessa signaali interpoloidaan; tulos on alla.



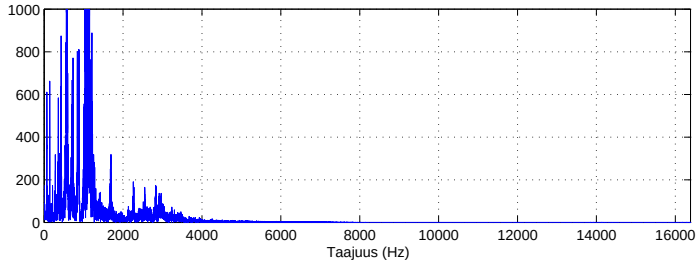
Interpoloinnin jälkeen signaali kvantisoidaan yhteen bittiin, eli käytännössä jäljelle jää vain näytteen merkki. Tällöin syntyy alla oleva virhesignaali. Huomaa, että kyseessä ei ole alkuperäisen signaalin merkki vaan mukana on myös edellisen askeleen virhesignaali.



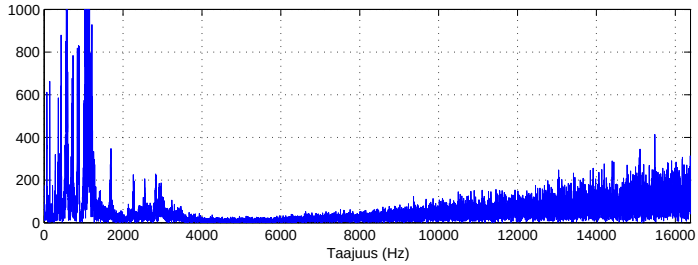
Järjestelmän ulostulosignaali näyttää seuraavalta.



Signaali ei näytä aikatasossa juurikaan samalta kuin alkuperäinen, mutta taajuustasossa yhtäläisyydet silti löytyvät. Ensin alkuperäisen interpoloidun signaalin spektri.



Kun tätä verrataan alla olevaan tulossignaalin spektriin, nähdään, että pienillä taajuuksilla spektrit ovat kokolailla samanlaisia. Lisäksi havaitaan suuremmilla taajuuksilla spektrin noudattavan amplitudivasteen $|H(e^{i\omega})| = 16\sin^4(\omega)$ muotoa. Nyt interpolointikerroin L ei ollut riittävän suuri, joten signaalin ja kohinan spektrit ovat osittain samalla taajuusalueella.



Seuraavaksi kvantisoitu signaali muunnetaan analogiseksi yksinkertaisella yksibittisellä D/A-muuntimella, jonka jälkeen analoginen signaali suodatetaan analogisella alipäästösuotimella. Analoginen suodin poistaa signaalista kohinan taajuudet. Jos interpolointikerroin ja järjestelmän aste olisivat olleet riittävän suuria, signaalin ja kohinan spektrit olisivat melko selkeästi erillään. Tällöin analogisen alipäästösuotimen suunnittelu olisi melko helppoa, koska siirtymäkaista saataisiin riittävän leveäksi.

Harjoitustehtäviä

3.1. Kuvitellun digitaalisen järjestelmän näytteenottotaajuus on 42 kHz. Jatkokäsittelyä varten signaali pitää muuntaa näytteenottotaajuuteen 6 kHz. Millaiset taajuusvaatimukset on asetettava antialias-suotimelle, jonka läpi signaali suodatetaan ennen varsinaista uudelleennäytteistystä? Signaalista tiedetään, että taajuuksien $0 - 2.5$ kHz tulee ehdottomasti säilyä desimoidussa signaalissa. (Siis missä on päästökaista ja estokaista?)

3.2. Signaali $x(n)$, jonka näytteenottotaajuus on 12 kHz pitää muuntaa signaaliksi, jonka näytteenottotaajuus on 15 kHz. Selvitä muunnoksen vaiheet lohkokaaviona käyttäen uudelleennäytteistystä ($\boxed{\uparrow L}$ ja $\boxed{\downarrow M}$) ja alipäästösuodatusta ($\boxed{H(z)}$). Esitä tarvittavien alipäästösuodinten päästö- ja estokaistojen sijainti, kun taajuudet väliltä $0 - 5.5$ kHz halutaan säilyttää.

3.3. Tavoitteena on pudottaa 12 kHz signaalin näytteenottotaajuus yhteen kilohertsiin useassa vaiheessa. Signaalin olennaisin informaatio sijaitsee taajuuskaistalla 0-450 Hz, joka tulee siis ehdottomasti säilyttää. Selvitä tarvittavien suodinten kertoimien yhteismäärä eri multistage-toteutuksissa (4 kpl riittää, jos tutkitaan vain ne tapaukset, joissa desimointikertoimet ovat laskevassa järjestyksessä), kun alipäästösuotimet suunnitellaan Hamming-ikkunalla (jolloin $N = 3.3/\Delta f$).

3.4. Kuinka monta kertolaskua kukin edellisistä toteutuksista tarvitsee sekunnissa (MPS, multiplications per second)? Tämä saadaan laskettua kaavasta

$$\mathrm{MPS} = \sum_{i=1}^{I} N_i F_i,$$

missä I on desimointilohkojen lukumäärä, N_i on i:nnen lohkon kertoimien määrä ja F_i on i:nnen lohkon sisääntulevan signaalin näytteenottotaajuus.

3.5. *(Matlab)* Toteutetaan tässä ja seuraavassa tehtävässä Matlabilla näytteenottotaajuuden muunnos kertoimella $\frac{2}{3}$. Lataa ensin testisignaali `laughter` (latautuu komennolla `load laughter` muuttujaan `y`). Signaalin näytteenottotaajuus on 8192 Hz.

Nostetaan signaalin näytteenottotaajuus ensin kaksinkertaiseksi. Tämä vaatii seuraavat vaiheet:

- Muodosta nollasignaali `z`, jonka pituus on kaksinkertainen alkuperäiseen nähden.
- Sijoita nollasignaalin joka toiseen paikkaan alkuperäisen signaalin arvot komennolla `z(1:2:end) = y;` Kuuntele tulos. Huomaa, että komennolle `soundsc` täytyy antaa myös näytteenottotaajuus; muutoin tulos kuulostaa pikkuoravilta.
- Suunnittele esim. astetta 100 oleva FIR-suodin (komento `fir1`) ja suodata syntyneet häiriöt pois.
- Kuuntele tulos ja totea, että se kuulostaa samalta kuin alkuperäinen.
- Tulosta kaikkien kolmen signaalin spektrogrammit komennolla `specgram`.

3.6. *(Matlab)* Pudota edellisen tehtävän tulossignaalin näytteenottotaajuus kolmasosaan seuraavasti.

- Suunnittele antialias-suodin, jonka aste on myös 100.
- Suodata signaali `z` kyseisellä suotimella.
- Ota tuloksesta talteen joka kolmas näyte.
- Kuuntele tulos näytteenottotaajuudella 5461 Hz, mikä on noin 2/3 alkuperäisestä.

- Tulosta kaikkien kolmen signaalin spektrogrammit komennolla `specgram`.

3.7. *(Matlab)* Toteutetaan Matlabilla ensimmäisen asteen kohinanmuokkain. Aloita Matlab-skripti lataamalla testisignaali Handel muuttujaan `y`. Sijoita `x=y;`, koska muuttujaa `y` tarvitaan myöhemmin. Interpoloi signaali kertoimella 4 käyttäen komentoa `interp`. Interpoloinnin tulos on lohkokaaviossa oleva signaali $w(n)$. Tätä signaalia lähdetään käsittelemään lohkokaavion mukaisesti. Alusta tätä varten muuttujat `e` ja `y` nolliksi: `e=zeros(1,length(w));`. Lohkokaavion mukainen kohinanmuokkaus voidaan esittää pseudokoodina seuraavasti:

```
for n := 2 to length(w)
    tmp  := w(n)-e(n-1);
    y(n) := sign(tmp);
    e(n) := y(n)-tmp;
end
```

Tulosta ruudulle ulostulosignaali ja sen spektri (`fft`:llä laskettuna) sekä virhesignaali. Desimoi signaali takaisin 8192 Hertsiin ja kuuntele tulos. Kvantisointikohina on selvästi kuultavissa signaalin taajuuskaistalla johtuen vain nelinkertaisesta interpoloinnista ja ensimmäisen asteen kohinanmuokkauksesta.

3.8. *(Matlab)* Muunna edellisen tehtävän kohinanmuokkaus toisen asteen versioksi, jossa pseudokoodi on seuraava.

```
for n := 3 to length(w)
    tmp  := w(n)-2*e(n-1)+e(n-2);
    y(n) := sign(tmp);
    e(n) := y(n)-tmp;
end
```

Tee vastaavat testit kuin edellisessä tehtävässä ja vertaa tulosta.

3.9. *(Matlab)* Muunna tehtävän 3.8 kohinanmuokkaus kolmannen asteen versioksi. Tee vastaavat testit kuin tehtävissä 3.7 ja 3.8 ja vertaa tulosta. Kuitenkin niin, että kvantisoit tuloksen $(1+1)$ bittiin. $(1+0)$-bittinen ratkaisu on numeerisesti epästabiili korkeamman asteen vuoksi. Siis ei

```
y(n) := sign(tmp);
```

vaan

```
y(n) := quant(tmp, 2^(-1));
```

3.10. Täydennä oheinen kohinanmuokkauksen lohkokaavio niin, että se esittää kolmannen asteen kohinanmuokkainta.

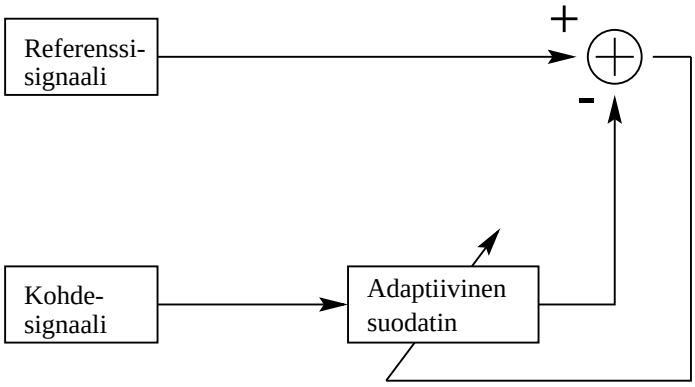
x(n)
↑L
H(z)
w(n)
Interpolointi
+
−
Kvanti−
sointi
y(n)
D/A
−
+

Luku 4

Adaptiivinen suodatus

Kun signaali ja kohina ovat eri taajuuskaistoilla, suodatusongelma on ratkaistavissa normaalein lineaarisen suodatuksen menetelmin. On kuitenkin tilanteita, joissa taajuuskaistat ovat osittain päällekkäin ja vaihtelevat ajan kuluessa. Lisäksi saattaa käydä niin, ettei suunnittelija tiedä vielä suunnitteluvaiheessa millaista kohinaa sovellusvaiheessa tulee vastaan. Tällöin suotimen täytyy oppia sovellutuksen olosuhteet ja *adaptoitua* niihin. Tämä tarkoittaa, että suodin muuttaa kertoimiaan olosuhteiden muuttuessa. Adaptiiviset järjestelmät eivät siis tarkalleen ottaen ole aikainvariantteja. Ne ovat kuitenkin lineaarisia ja ne esitetään konvoluution avulla.

Adaptiiviset suotimet voivat toimia lukuisien erilaisten konfiguraatioiden yhteydessä. Seuraava kuva esittää niistä yhtä.



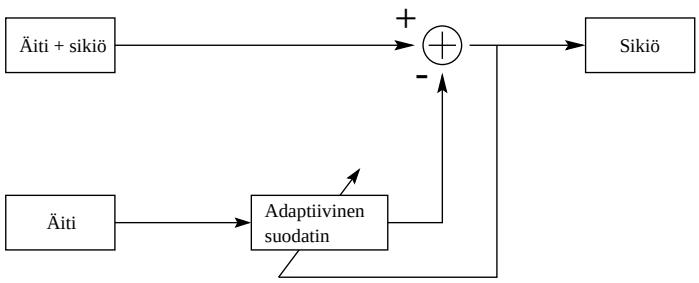
Kuvassa on kaksi signaalilähdettä, joista toisesta tulee suodatettava signaali (kohdesignaali) ja toisesta tulee jonkinlainen estimaatti toivotusta tulossignaalista (ns. referenssisignaali). Adaptiivinen suodatin pyrkii koko ajan minimoimaan referenssisignaalin ja suodatustuloksen välistä erotusta. Ensinäkemältä idea ei vaikuta kovinkaan järisyttävältä, koska toivottu tulossignaali täytyy tietää ennen suodatusta. Valitsemalla suodatettava signaali ja referenssisignaali sopivasti päästään kuitenkin mielenkiintoisiin tuloksiin. Seuraavassa tarkastellaan muutamia sovelluskohteita.

4.1 Kohinan poisto

Tietyn tyyppisiä häiriöitä voidaan poistaa signaalista adaptiivisin menetelmin. Adaptiivisuus tulee kyseeseen silloin, kun häiriön parametreja ei tiedetä tai ne vaihtelevat ajan myö-

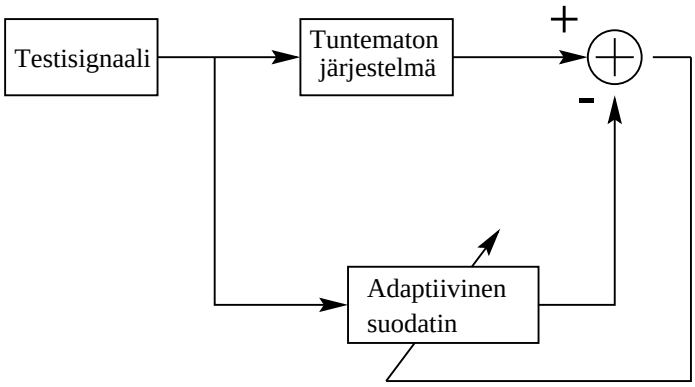
tä. Referenssisignaali sisältää tällöin signaalin, joka on tietyn tyyppisen häiriön vääristämä. Suodatettavana signaalina on toinen häiriösignaali, joka korreloi jollain lailla referenssisignaalin häiriön kanssa. Tällöin adaptiivinen suodin pyrkii minimoimaan referenssisignaalin ja häiriön välistä erotusta käyttäen hyväkseen kahden eri häiriön yhteisiä piirteitä. Jos tämä onnistuu, vähennyslaskun tuloksena on signaali, josta on poistettu alkuperäinen häiriö.

Esimerkkinä tämän tyyppisestä menettelystä on sikiön sydänäänien tunnistus. Ongelmana tunnistuksessa normaalisti on äidin sydänäänien voimakkuus. Adaptiivisessa järjestelmässä sijoitetaan yksi mikrofoni äidin sydämen lähelle ja toinen äidin vatsalle. Referenssisignaali tulee nyt vatsan mikrofonista ja sisältää varsinaisen signaalin (sikiön sydänäänet) sekä häiriökomponentin (äidin sydänääni). Häiriökomponentti voidaan poistaa toisen mikrofonin signaalin avulla. Äidin sydämen lähellä olevan mikrofonin signaali koostuu miltei pelkästään äidin sydänäänistä. Adaptiivinen algoritmi pyrkii minimoimaan kahden signaalin erotuksen ja samalla poistamaan äidin sydänäänet mahdollisimman tehokkaasti. Sikiön sydänäänet sen sijaan säilyvät sillä ne eivät juurikaan korreloi äidin sydänäänien kanssa. Siksi adaptiivinen suodatin ei tee niille mitään. Alla on tämän järjestelmän lohkokaavio.



4.2 Tuntemattoman järjestelmän mallinnus

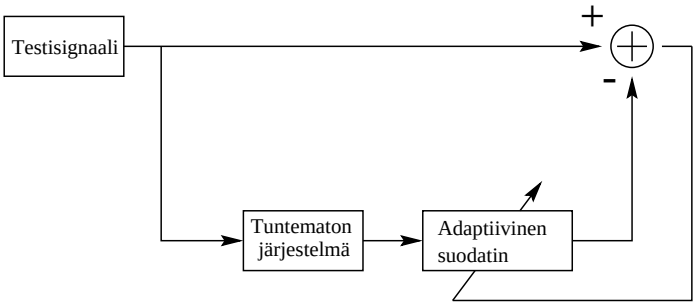
Adaptiivisella järjestelmällä voidaan mallintaa tuntematonta järjestelmää ja laskea sen ominaisuuksia. Tällöin suodatettavana signaalina on jokin (laajakaistainen, useita taajuuksia sisältävä) testisignaali ja referenssisignaalina on tuntemattoman järjestelmän ulostulo, kun herätteenä on kyseinen testisignaali. Jos adaptiivinen suodatin kykenee minimoimaan testisignaalin ja referenssisignaalin välisen erotuksen, se itse asiassa käyttäytyy samoin kuin tuntematon mallinnettava laite. Adaptiivisen suodattimen ominaisuudet ovat tällöin matemaattinen malli tuntemattoman järjestelmän ominaisuuksista. Alla on tällaisen järjestelmän lohkokaavio.



4.3 Ekvalisointi

Ekvalisoinnilla tarkoitetaan vääristyneen signaalin palauttamista mahdollisimman lähelle alkuperäistä. Vääristymisen syynä voi olla esimerkiksi kohinainen tiedonsiirtokanava. Sekä lähetin että vastaanotin tietävät jonkin laajakaistaisen testisignaalin, jota lähetetään säännöllisin väliajoin. Adaptiivinen suodatin suodattaa vastaanotettua testisignaalia ja referenssisignaalina on alkuperäinen testisignaali. Adaptioprosessin tuloksena on suodin, joka palauttaa kohinaisen signaalin mahdollisimman lähelle alkuperäistä signaalia täten minimoiden suodatustuloksen ja referenssisignaalin välisen virheen. Samaa suodatinta voidaan käyttää myös muun signaalin suodatukseen, jos kanavan ominaisuudet pysyvät riittävän samanlaisina seuraavan testisignaalin lähettämiseen saakka.

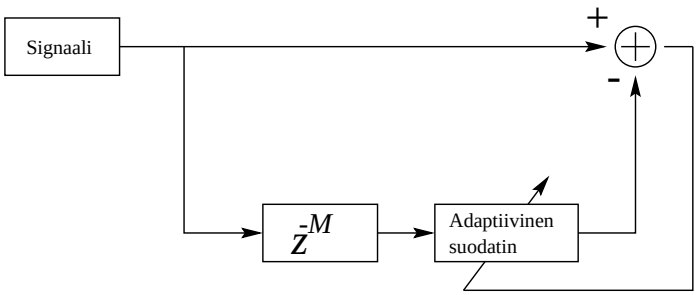
Tätä tekniikkaa käytetään GSM-puhelimissa ekvalisoimaan puhelimen liikkeistä ja signaalin heijastumisesta aiheutuvia häiriöitä. Tukiasema ja puhelin lähettävät säännöllisin väliajoin tiettyä testisignaalia, jonka molemmat tietävät. Kun kanava muuttuu puhelimen liikkuessa, molemmat sovittavat ekvalisaatiosuotimensa tilanteen mukaan. Lohkokaavio on alla.



4.4 Ennustus

Jos valitaan suodatettavaksi signaaliksi viivästetty versio referenssisignaalista, voidaan adaptiivista suodinta käyttää tulevien signaaliarvojen ennustamiseen. Suodatuksen edetessä suodin pyrkii minimoimaan referenssisignaalin ja viivästetyn version välistä eroa. Jos signaali on riittävän yksinkertainen (muodostuu esimerkiksi vain muutamista taajuuksista), suodin oppii ennustuksen varsin pian. Ennustavaa adaptiivista suodinta käytetään hyväksi esimerkiksi signaalin kompressiossa. Jos lähetin- ja vastaanottajapäässä käytetään

samaa ennustajaa, tarvitsee välillä lähettää ainoastaan ennustusvirhe. Jos ennustaja toimii hyvin, ennustusvirhe sisältää ainoastaan lähellä nollaa olevia arvoja, jotka saadaan pakattua pieneen tilaan. Informaatioteoreettisin termein ilmaistuna virhesignaalilla on pieni entropia. Lohkokaavio on alla.



4.5 Suodatusongelman teoreettinen optimiratkaisu

Ennen varsinaisten adaptoitumisessa tarvittavien algoritmien esittelyä tarkastellaan hieman yleistä tehtävänasettelua ja teoreettisia ratkaisuja sille.

Olkoon $x(n)$ suodatettava signaali ja $s(n)$ referenssisignaali. Merkitään suodatustulosta merkinnällä

$$y(n) = w(n) * x(n) = \sum_{k=0}^{N-1} w(k)x(n-k),$$

missä $w(n)$ on adaptiivisen suodattimen painovektori. Yleensä tämä painovektori muuttuu ajan funktiona, joten oikeampi merkintä olisi $w_n(k)$, tms., mutta yksinkertaisuuden vuoksi jätetään aikaindeksi jatkossa pois. Yksinkertaisin esitysmuoto konvoluutiosummalle saadaan vektorien sisätulona:

$$y(n) = \mathbf{w}^T\mathbf{x}(n) = \mathbf{x}^T(n)\mathbf{w},$$

missä $\mathbf{w} = (w(0), w(1), \ldots, w(N-1))^T$ ja $\mathbf{x}(n) = (x(n), x(n-1), \ldots, x(n-N+1))^T$.

Määritellään lisäksi virhesignaali

$$e(n) = s(n) - y(n) = s(n) - \mathbf{w}^T\mathbf{x}(n),$$

joka pyritään saamaan mahdollisimman lähelle nollasignaalia. Se, kuinka lähellä virhesignaali on nollasignaalia, määräytyy *kustannusfunktion* avulla.

Mahdollisia kustannusfunktioita voisivat olla esimerkiksi

$$c(\mathbf{w}) = E[|e(n)|]$$

ja

$$c(\mathbf{w}) = E[e^2(n)],$$

missä $e^2(n)$ on lyhyempi merkintä lukujonolle $(e(n))^2$, ja $E[e^2(n)]$ tarkoittaa kyseisen jonon odotusarvoa.

Koska tavoitteena on minimoida kustannusfunktio, sen on hyvä olla derivoituva. Siksi valitsemme jälkimmäisen kustannusfunktion (funktio $f(x) = x^2$ on kaikkialla derivoituva, mutta $f(x) = |x|$ ei).

Auki kirjoitettuna kustannusfunktio saa muodon

$$\begin{aligned} c(\mathbf{w}) &= E[e^2(n)] \\ &= E\left[(s(n)-y(n))^2\right] \\ &= E[s^2(n)] + \underbrace{E[y^2(n)]}_{(*)} - \underbrace{2E[s(n)y(n)]}_{(**)}. \end{aligned}$$

Tämän lausekkeen kaksi viimeistä odotusarvoa puretaan auki:

$$\begin{aligned} (*) &= E\left[(\mathbf{w}^T\mathbf{x}(n))^2\right] \\ &= E\left[\mathbf{w}^T\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}\right] \\ &= \mathbf{w}^T E[\mathbf{x}(n)\mathbf{x}^T(n)]\mathbf{w} \\ &= \mathbf{w}^T\mathbf{R}\mathbf{w}, \end{aligned}$$

ja

$$(**) = 2E[s(n)\mathbf{x}^T(n)\mathbf{w}] = 2E[s(n)\mathbf{x}^T(n)]\mathbf{w} = 2\mathbf{p}^T\mathbf{w},$$

missä matriisi $\mathbf{R} = E[\mathbf{x}(n)\mathbf{x}^T(n)]$ on $N \times N$ matriisi, josta käytetään nimeä *autokorrelaatiomatriisi* ja vektori $\mathbf{p} = E[s(n)\mathbf{x}^T(n)]$ on signaalien $s(n)$ ja $\mathbf{x}(n)$ *ristikorrelaatiovektori*. Esimerkiksi tapauksessa $N = 4$ saavat $\mathbf{R}$ ja $\mathbf{p}$ muodon

$$\mathbf{R} = E\begin{pmatrix} x(n)x(n) & x(n)x(n-1) & x(n)x(n-2) & x(n)x(n-3) \\ x(n-1)x(n) & x(n-1)x(n-1) & x(n-1)x(n-2) & x(n-1)x(n-3) \\ x(n-2)x(n) & x(n-2)x(n-1) & x(n-2)x(n-2) & x(n-2)x(n-3) \\ x(n-3)x(n) & x(n-3)x(n-1) & x(n-3)x(n-2) & x(n-3)x(n-3) \end{pmatrix}$$

ja

$$\mathbf{p} = E\begin{pmatrix} s(n)x(n) \\ s(n)x(n-1) \\ s(n)x(n-2) \\ s(n)x(n-3) \end{pmatrix}.$$

Matriisin $\mathbf{R}$ symmetrisyys ja yksinkertainen muoto tulevat hyvin ilmi eräälle testisignaalille lasketuista arvoista[1]:

$$\mathbf{R} = \begin{pmatrix} 0.0385 & 0.0291 & 0.0127 & -0.0023 \\ 0.0291 & 0.0385 & 0.0291 & 0.0127 \\ 0.0127 & 0.0291 & 0.0385 & 0.0291 \\ -0.0023 & 0.0127 & 0.0291 & 0.0385 \end{pmatrix}, \quad \mathbf{p} = \begin{pmatrix} 0.0224 \\ 0.0322 \\ 0.0311 \\ 0.0184 \end{pmatrix}.$$

Kustannusfunktio saa siis muodon $c(\mathbf{w}) = E[s^2(n)] + \mathbf{w}^T\mathbf{R}\mathbf{w} - 2\mathbf{p}^T\mathbf{w}$. Tämä funktio on derivoituva vektorin $\mathbf{w}$ jokaisen alkion suhteen ja sen gradientin nollakohta minimoi kustannusfunktion. Kustannusfunktio on aina paraboloidi.

Ensimmäinen derivoitava, $s^2(n)$ ei riipu kertoimista $w(0), w(1), \ldots, w(N-1)$, joten gradientista tulee nolla. Toinen termi sen sijaan riippuu ja gradientiksi tulee (yksi kerrallaan derivoimalla ja yhdistämällä tuloksen vektoriksi)

$$\frac{\partial}{\partial \mathbf{w}} E[y^2(n)] = \frac{\partial}{\partial \mathbf{w}} \mathbf{w}^T\mathbf{R}\mathbf{w} = 2\mathbf{R}\mathbf{w}.$$

[1] Koska signaali oli äärellisen mittainen, odotusarvo korvattiin 73109:n näytteen keskiarvolla.

Kolmannen termin gradientti puolestaan on

$$\frac{\partial}{\partial \mathbf{w}} E[2s(n)y(n)] \quad = \quad \frac{\partial}{\partial \mathbf{w}} 2\mathbf{p}^T\mathbf{w} = 2\mathbf{p}.$$

Näin ollen

$$\nabla c = \frac{\partial}{\partial \mathbf{w}} c(\mathbf{w}) = 2\mathbf{R}\mathbf{w} - 2\mathbf{p}.$$

Asetetaan gradientti nollaksi:

$$2\mathbf{R}\mathbf{w} - 2\mathbf{p} = 0$$

Tästä saadaan *normaaliyhtälöt*

$$\mathbf{R}\mathbf{w} = \mathbf{p},$$

ja jos $\mathbf{R}$ on kääntyvä,

$$\mathbf{w} = \mathbf{R}^{-1}\mathbf{p}.$$

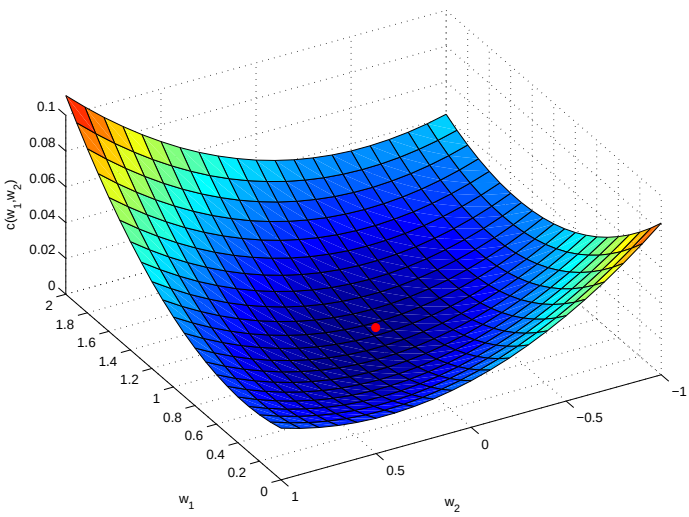
Esimerkiksi edellä olleen esimerkkimatriisin $\mathbf{R}$ käänteismatriisi on

$$\mathbf{R}^{-1} = \begin{pmatrix} 89.3205 & -101.4405 & 41.7160 & 7.2675 \\ -101.4405 & 203.9342 & -152.2110 & 41.7160 \\ 41.7160 & -152.2110 & 203.9342 & -101.4405 \\ 7.2675 & 41.7160 & -101.4405 & 89.3205 \end{pmatrix},$$

joten optimaaliset kertoimet tässä tapauksessa ovat[2]

$$\mathbf{w} = \mathbf{R}^{-1}\mathbf{p} = \begin{pmatrix} 0.1655 \\ 0.3282 \\ 0.5091 \\ -0.0053 \end{pmatrix}.$$

Jos suotimessa on vain kaksi kerrointa, voidaan kustannusfunktion kuvaaja myös piirtää. Kustannusfunktio harjoitustehtävän 4.1 signaaleille tässä tapauksessa on kuvattu alla.



[2]Tässä tapauksessa referenssisignaali $s(n)$ muodostettiin signaalista $x(n)$ suodattamalla se suotimella, jonka impulssivaste on $h(n) = \frac{1}{6}\delta(n) + \frac{2}{6}\delta(n-1) + \frac{3}{6}\delta(n-2)$. Tämän jälkeen signaaliin $s(n)$ lisättiin hieman kohinaa realistisemman kokeen luomiseksi.

Esimerkin tapauksessa suotimella on vain kaksi kerrointa: $\mathbf{w}^T = [w_1, w_2]$, joten kustannusfunktio voidaan piirtää niiden suhteen. Tässä tapauksessa matriisit $\mathbf{R}$ ja $\mathbf{p}$ ovat

$$\mathbf{R} = \begin{pmatrix} 0.0308 & 0.0156 \\ 0.0156 & 0.0308 \end{pmatrix}, \quad \mathbf{p} = \begin{pmatrix} 0.0259 \\ 0.0140 \end{pmatrix}.$$

Näiden avulla voidaan laskea optimipainot: $\mathbf{w} = \mathbf{R}^{-1}\mathbf{p} = [0.8220, 0.0367]^T$, jotka on merkitty punaisella pallolla yllä olevaan kuvioon. Kuviosta nähdään painojen tosiaankin sijaitsevan kustannusfunktion minimikohdassa. Lisäksi kuviosta havaitaan kustannusfunktion olevan paraboloidin muotoinen. Tämä muoto on voimassa myös pidemmille suotimille, mutta sen piirtäminen ei enää ole mahdollista.

Käytännön menetelmät pyrkivät välttämään matriisiyhtälön $\mathbf{Rw} = \mathbf{p}$ ratkaisemista sen laskennallisen kompleksisuuden vuoksi. Iteratiiviset menetelmät päivittävät painovektoria kustannusfunktion gradientin mukaan, ja kustannusfunktion muodon perusteella tämä johtaa aina optimipisteeseen. Tällaisia menetelmiä ovat *least mean squares* (LMS) ja *recursive least squares* (RLS) -algoritmit, joista edelliseen tutustutaan seuraavassa.

4.6 Gradienttimenetelmä ja LMS-algoritmi

Käytännön tilanteissa valmista tilastollista mallia ei useinkaan ole saatavilla ja tilastolliset ominaisuudet muuttuvat ajan myötä. Siksi myös autokorrelaatiomatriisin ja ristikorrelaatiovektorin estimaatteja täytyy päivittää jatkuvasti. Jos tämä on helposti tehtävissä, voidaan käyttää *gradienttimenetelmää* adaptiivisen suodatuksen ratkaisuna. Gradienttimenetelmä on iteratiivinen menetelmä, joka estimoi kullakin iteraatiokierroksella matriisia $\mathbf{R}$ ja vektoria $\mathbf{p}$ ja päivittää näiden perusteella adaptiivisen suotimen kertoimia $\mathbf{w}$ tietyn matkan negatiivisen gradientin suuntaan (siis sinne missä kustannusfunktio pienenee). Tämä voidaan ilmaista kaavamuodossa seuraavasti:

$$\mathbf{w} := \mathbf{w} - \mu \cdot \nabla c(\mathbf{w}), \tag{4.1}$$

missä $\nabla c(\mathbf{w})$ on kustannusfunktion gradientti ja μ on pieni positiivinen reaaliluku, joka kertoo kuinka pitkiä harppauksia gradientin määräämään suuntaan otetaan. Jos μ on liian pieni, adaptoituminen muuttuu liian hitaaksi, ja jos se on liian suuri, prosessi ei suppene kohti optimiratkaisua.[3]

Ongelmana yllä olevan kaavan käytössä on gradientin estimointi. Tarkastellaan asiaa lähemmin seuraavassa. Aikaisemmin laskettiin kustannusfunktion gradientille seuraava lauseke:

$$\nabla c(\mathbf{w}) = 2\mathbf{Rw} - 2\mathbf{p}.$$

Sijoittamalla $\mathbf{R} = E[\mathbf{x}(n)\mathbf{x}^T(n)]$ ja $\mathbf{p} = E[s(n)\mathbf{x}(n)]$ saadaan yhtälöt

$$\begin{aligned} \nabla c(\mathbf{w}) &= 2E[\mathbf{x}(n)\mathbf{x}^T(n)]\mathbf{w} - 2E[s(n)\mathbf{x}(n)] \\ &= 2E[\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w} - s(n)\mathbf{x}(n)] \\ &= 2E[\mathbf{x}(n)y(n) - s(n)\mathbf{x}(n)] \\ &= 2E[\mathbf{x}(n)(y(n) - s(n))] \\ &= -2E[e(n)\mathbf{x}(n)]. \end{aligned}$$

[3] Voidaan osoittaa, että LMS-algoritmi suppenee, jos $0 < \mu < \frac{1}{\lambda_{max}}$, missä λ_{max} on autokorrelaatiomatriisin $\mathbf{R}$ suurin ominaisarvo. Tämän luvun tarkka määrittäminen on käytännössä usein mahdotonta.

Viimeisintä odotusarvoa ei voida käytännössä arvioida tarkasti, joten on tyydyttävä käyttämään jotain likiarvoa sen sijaan. Osoittautuu, että algoritmi toimii vaikka odotusarvon tilalla käytettäisiin vain viimeisimmän ajanhetken tilannetta. Tällöin

$$\nabla c(\mathbf{w}) \approx -2e(n)\mathbf{x}(n). \tag{4.2}$$

Näin saadusta gradienttimenetelmän muodosta käytetään nimitystä *LMS-algoritmi*. Sen kerrointen päivittäminen voidaan tiivistää kaavaan

$$\mathbf{w} := \mathbf{w} + 2\mu e(n)\mathbf{x}(n),$$

joka on saatu sijoittamalla kaava (4.2) kaavaan (4.1).

Kokonaisuudessaan LMS-algoritmi on seuraavanlainen.

LMS-algoritmi

1. Alusta painovektori $\mathbf{w}$ sopivilla alkuarvoilla, esimerkiksi nollavektoriksi.
2. Laske ulostulo $y(n) = \mathbf{w}^T\mathbf{x}(n)$.
3. Laske virhe $e(n) = s(n) - y(n)$.
4. Päivitä suotimen kertoimia:
$$\mathbf{w} := \mathbf{w} + 2\mu e(n)\mathbf{x}(n).$$
5. Jos suodatettava signaali ei ole lopussa, aseta $n := n + 1$ ja palaa kohtaan 2.

4.7 Esimerkkejä LMS-algoritmin toiminnasta

Seuraavassa tarkastellaan LMS-algoritmin toimintaa keinotekoisessa adaptiivisen suodatuksen esimerkissä. Tarvittava rutiini on adaptiivisen suodatuksen toteuttava algoritmi, jonka muoto Matlabissa on

```
[y,w] = LMS (x, ref, mu, N)
```

Funktio saa siis lähtötietoina signaalin `x`, referenssisignaalin `ref`, suotimen pituuden `N` ja parametrin μ. Funktio palauttaa ulostulovektorin `y` ja viimeisen kierroksen painovektorin `w`, joka sisältää viimeisen iteraation painot (pituus on siis `N`). Signaalien `y` ja `ref` erotus on tällöin virhevektori `e`, joka ilmoittaa kuinka paljon suodatustulos poikkeaa referenssisignaalista `ref` kullakin iteraatiokierroksella. Seuraava Matlab-koodi toteuttaa LMS-suodatuksen, ja se on ladattavissa osoitteesta `http://www.cs.tut.fi/kurssit/SGN-1251/LMS.m`.

```
function [y,w]=LMS(x,ref,mu,N)

y=zeros(size(x));       % Alustetaan ulostulo nolliksi
w=zeros(1,N);           % Alustetaan suodin nolliksi

for n=N:length(x),
  data=x(n:-1:n-N+1);   % Otetaan pätkä suodatettavaa signaalia
```

```
  y(n)=w*data;          % Lasketaan konvoluutio painojen kanssa
  e=ref(n)-y(n);        % Lasketaan erotussignaali
  w=w+2*mu*e*data';     % Päivitetään painoja
end;
```

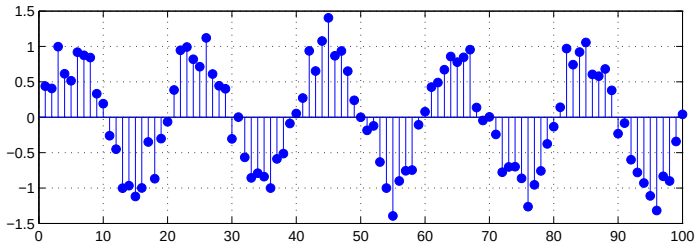
4.7.1 Häiriötaajuuden estimointi ja poisto

Tarkastellaan tilannetta, jossa testisignaali koostuu kahdesta taajuuskomponentista, joista toinen on häiriö ja toinen etsittävä signaali. Tämä yksinkertaistus vastaa esimerkiksi kappaleen 4.1 tilannetta.

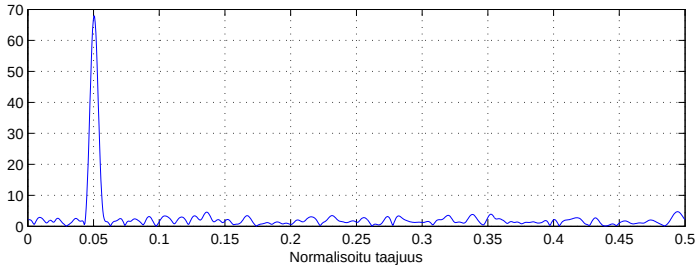
Olkoon

```
x=sin(0.05*2*pi*(1:256)) + 0.2*randn(1,256);
```

sisääntuleva signaali, josta pitäisi estimoida häiriökomponenttia. Tämä vastaa siis kappaleen 4.1 esimerkin tapauksessa äidin sydänääntä. Sen 100 ensimmäistä komponenttia ovat kuvassa:

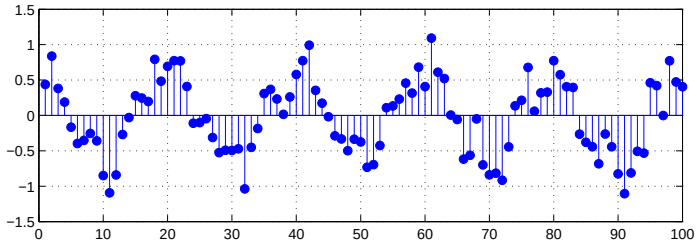


Sen spektri näyttää seuraavalta:

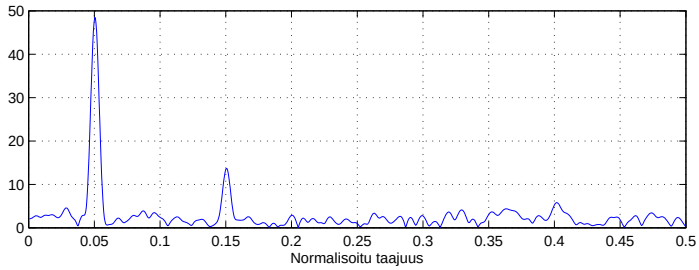


Oletetaan, että toisesta signaalilähteestä saadaan referenssisignaali, jossa on häiriötä sekä toinen sinisignaali summautuneena. Tämä voisi olla kappaleen 4.1 esimerkin tapauksessa äidin (taajuus 0.05) ja sikiön (taajuus 0.15) sydänäänten summa.

```
s = 0.7*cos(0.05*2*pi*(1:256)) + 0.2*sin(0.15*2*pi*(1:256))...
  + 0.2*randn(1,256);
```



Tämän signaalin spektri on seuraavassa:

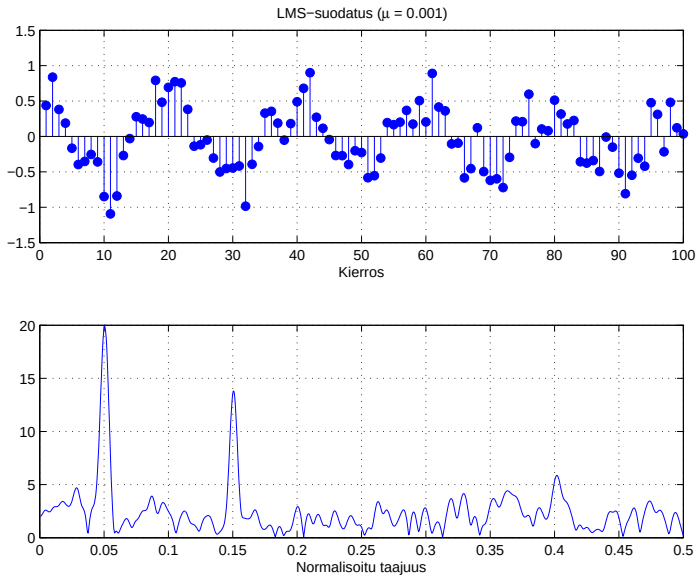


Adaptoitumisen tarkoituksena on siis tässä tapauksessa oppia löytämään (automaattisesti; ilman tietoa komponenttien taajuuksista tai amplitudeista) signaalista $x(n)$ signaalin $s(n)$ häiriökomponentti pienentämällä amplitudia ja muuttamalla vaihetta. Toivotun järjestelmän pitäisi oppia muuntamaan signaalin $x(n)$ komponentti $\sin(0.05 \cdot 2\pi n)$ signaalin $s(n)$ komponentiksi $0.7\cos(0.05 \cdot 2\pi n)$. Tämän vaatimuksenhan toteuttaa mikä tahansa suodin, jonka amplitudivaste taajuudella 0.05 on 0.7 (-3.10 dB) ja vaihevaste -90 astetta (eli sinin ja kosinin vaihe-ero).

Kutsutaan LMS-suodatusta niin, että suotimessa on 21 kerrointa ja parametri $\mu = 0.001$.

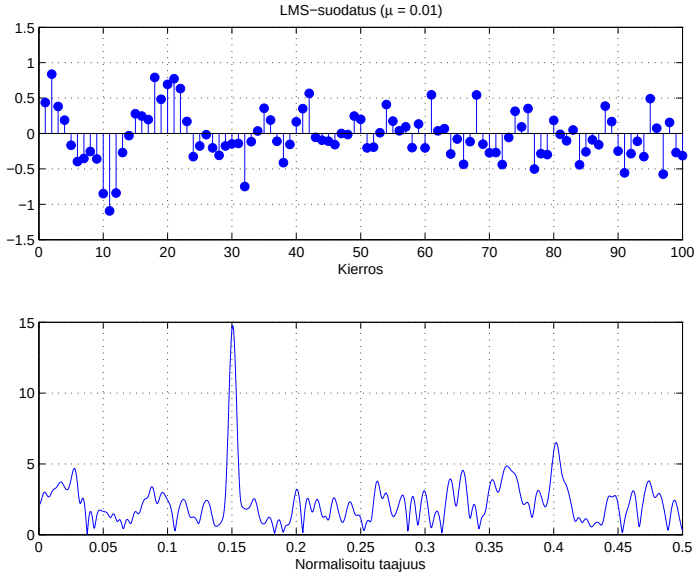
```
[y,w] = LMS (x, s, 0.001, 21);
```

Tuloksena on seuraava erotuskäyrä spektreineen. Ideaalitapauksessa spektrissä pitäisi olla jäljellä vain sikiön sydänkäyrä taajuudella 0.15.



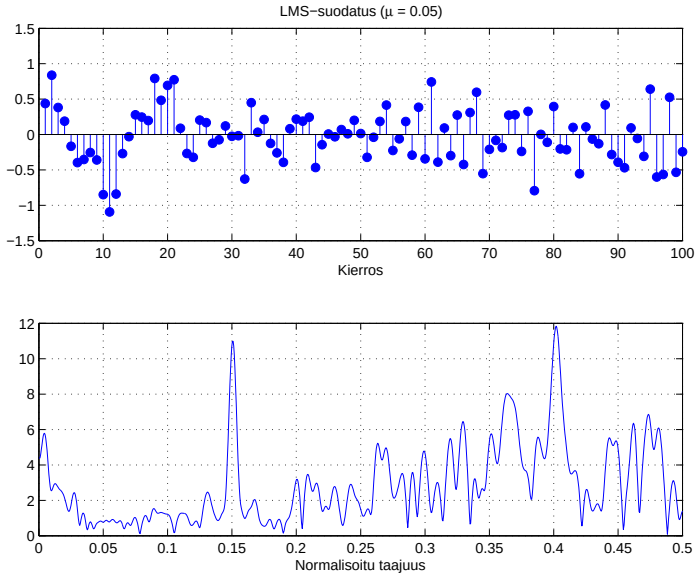
Suppeneminen vaikuttaa hieman hitaalta, ja häiriötaajuus on mukana etsittävää taajuutta voimakkaampana. Kokeillaan suurempaa arvoa parametrille μ.

```
[y,w] = LMS (x, s, 0.01, 21);
```



Nyt suodatus selvästikin kumoaa häiriökomponentin, jolloin varsinainen signaali tulee hyvin esiin. Kasvatetaan parametria μ edelleen

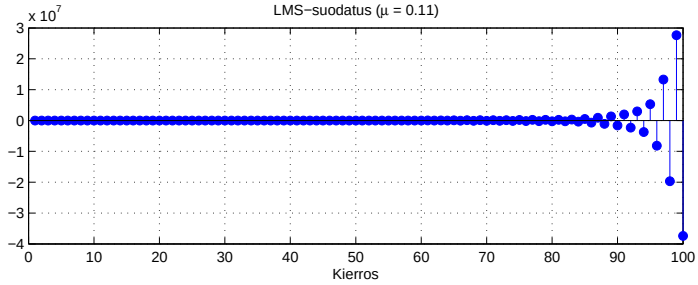
```
[y,w] = LMS (x, s, 0.05, 21);
```

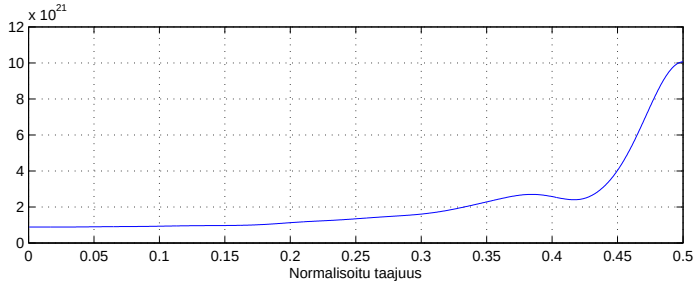


Tässä tapauksessa aikaisempi häiriökomponentti kyllä poistuu, mutta suodin vahvistaa samalla korkeampia taajuuksia, jolloin siellä olevan kohinan määrä kasvaa.

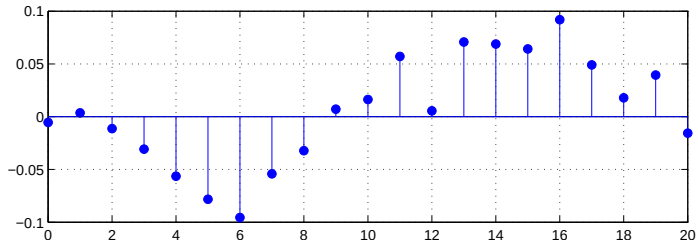
Seuraavalla parametrin μ arvolla prosessi ei selvästikään enää suppene:

```
[y,w] = LMS (x, s, 0.11, 21);
```

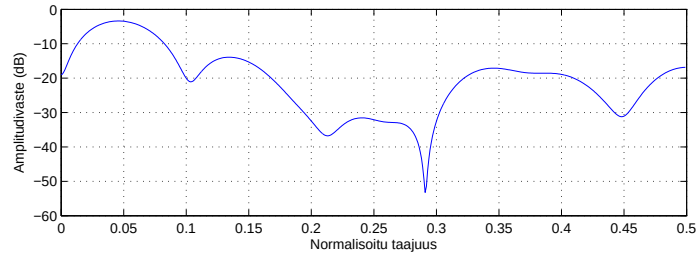




Tutkitaan nyt adaptiivisen suotimen kertoimia LMS-algoritmin viimeisellä kierroksella. Esimerkiksi tapauksessa $\mu = 0.01$ suotimen painot (vektorissa `w`) ovat alla olevan kuvan mukaiset.

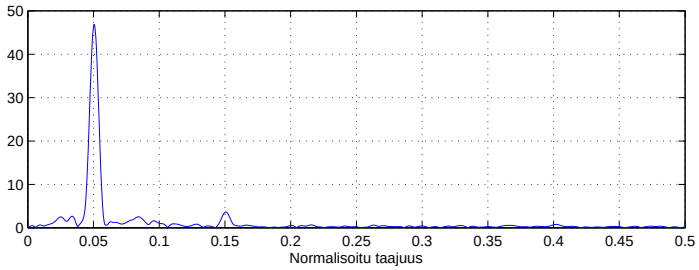


Siirtymällä taajuustasoon nähdään näitä painoja vastaavan suotimen amplitudivaste:



Meitä kiinnostaa erityisesti amplitudivaste taajuudella 0.05. Taajuusvasteen kaavalla laskettaessa Matlab antaa tulokseksi $W(e^{0.05\cdot 2\pi i}) = -0.0390 - 0.6152i$, jolloin amplitudivaste on 0.6165, mikä on melko lähellä toivottua arvoa. Vaihevaste on -93.6307 astetta, joka on myös melko lähellä oikeaa. Kohinan vaikutus kokeessa on melko suuri, joten vasteet vaihtelevat melko paljon toistettaessa koe.

Adaptiivisen suotimen ulostulon $y(n)$ spektri on alla olevan kuvan näköinen.



4.7.2 Suppenemistesti

Suppenemisen nopeus käy paremmin ilmi keinotekoisesta esimerkistä, jossa virhesignaali on mahdollista saada nollaksi sopivilla kertoimilla. Käytännön tilanteissa tämä on harvinaista, mutta esimerkkinä havainnollinen.

Olkoon suodatettava signaali $x(n)$ satunnaissignaali:

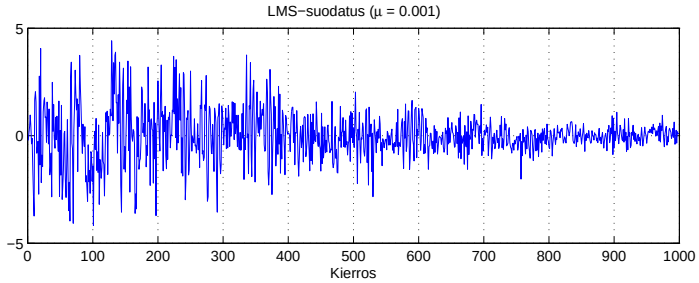
```
x = randn(1000,1);
```

Muodostetaan referenssisignaali $s(n)$ konvoluution avulla:

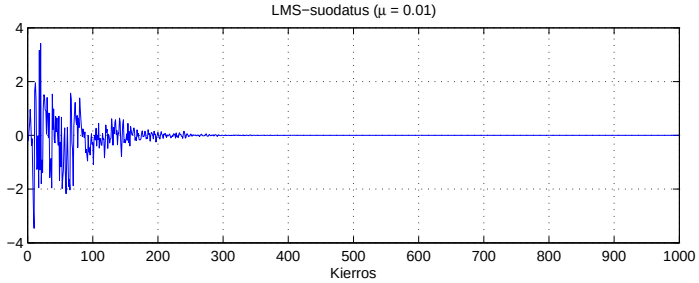
```
h = randn(1,9);
s = conv(x,h);
```

Konvoluutiossa käytetty impulssivaste $h(n)$ on satunnaisesti generoitu. Teoriassa LMS-suodatuksen virheen $e(n)$ pitäisi lähestyä nollaa kun suotimen painovektori $\mathbf{w}$ lähestyy impulssivasteen $h(n)$ arvoja.

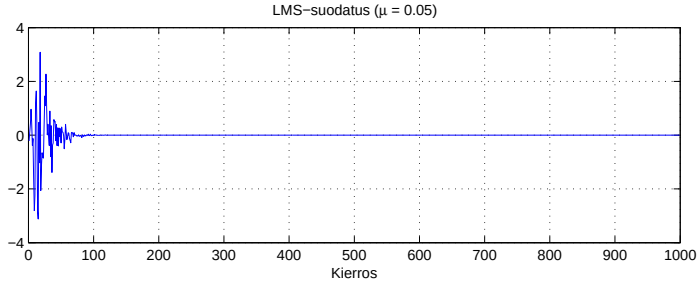
Kokeillaan ensin kertoimen μ arvolla 0.001. Virhesignaali on alla.



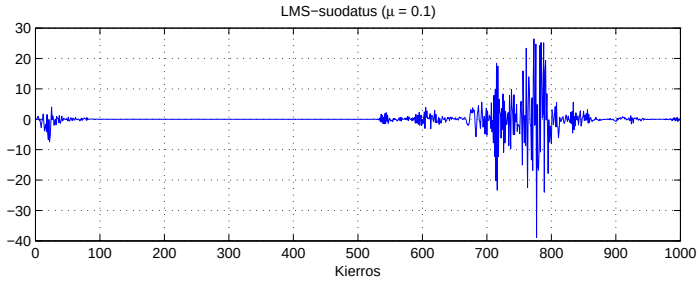
Virhe selvästi pienenee kierros kierrokselta, mutta on melko hidasta. Kokeillaan suuremmalla arvolla, $\mu = 0.01$.



Nyt suodatus oppii käytetyt kertoimet melko pian, noin parinsadan kierroksen jälkeen. Oppiminen nopeutuu edelleen kun kerrointa kasvatetaan: $\mu = 0.05$.



Kun kerrointa kasvatetaan edelleen, siitä tulee jo liian suuri, eikä prosessi enää suppene arvolla $\mu = 0.1$.

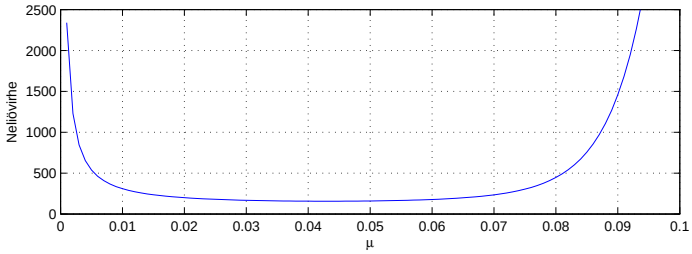


On mahdollista kokeilla useilla eri kertoimen μ arvoilla, kuinka nopeaa suppeneminen on. Suppenemisen nopeutta voidaan mitata yksinkertaisesti laskemalla referenssisignaalin $s(n)$ ja suodatustuloksen $y(n)$ erotuksen neliösumma, eli

$$\sum_{n=1}^{1000} (s(n) - y(n))^2.$$

Jos algoritmi suppenee nopeasti, $s(n)$ ja $y(n)$ ovat nopeasti lähellä toisiaan. Jos taas suppeneminen on hidasta tai prosessi ei suppene, on erotuksen neliösumma suuri.

Suppenemistesti ajettiin luvun μ arvoille $\mu = 0.001, 0.002, 0.003, \ldots, 0.1$, ja neliösummat näillä arvoilla näkyvät alla olevassa kuvassa.



Tässä tapauksessa optimiarvo näyttäisi olevan noin $\mu = 0.05$. Yleisesti ottaen optimiarvo riippuu kuitenkin signaaleista, eikä tämä lukuarvo ole optimi muissa tapauksissa. Käyrän muoto on kuitenkin hyvä muistaa, koska se pysyy suunnilleen samanlaisena muissakin tilanteissa.

Harjoitustehtäviä

4.1. *(Matlab)* Kopioi LMS-algoritmi itsellesi osoitteesta

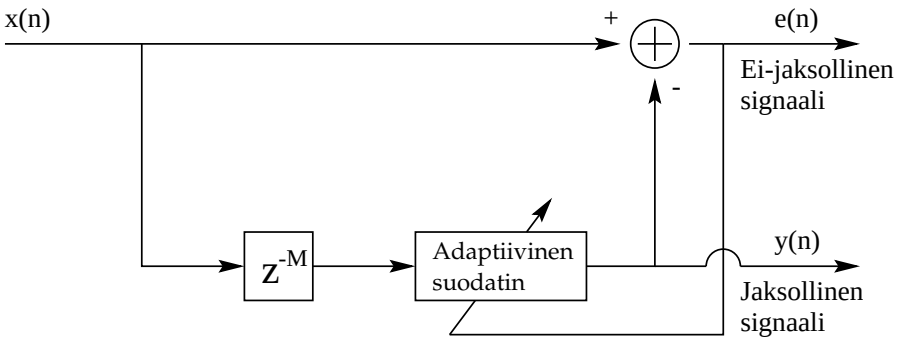
`http://www.cs.tut.fi/kurssit/SGN-1251/LMS.m`

Lisää nyt periodinen kohinakomponentti signaaliin `laughter`:

```
noisy=y+0.1*sin(2*pi*0.2*(1:length(y)))';
```

Jos kuuntelet signaalin, havaitset tasaisen ylimääräisen äänen taajuudella $0.2 \cdot 8192 = 1638.4$ Hz. LMS-algoritmi poistaa sen helposti, jos käytettävissä on alkuperäinen referenssisignaali. Sovella LMS-algoritmia signaaliin `noisy` siten, että `y` on referenssisignaalina. Suotimen pituus N voi olla muutaman kymmenen näytteen luokkaa. Tulosta ruudulle saadun suotimen amplitudivaste. Huomaa, että funktiota on kutsuttava isoilla kirjaimilla: `LMS`. Matlabissa on myös pienellä kirjoitettu funktio `lms`, joka toimii hieman eri tavalla.

4.2. *(Matlab)* Usein käytettävissä ei ole alkuperäistä referenssisignaalia, mutta LMS-algoritmi on tällöinkin käyttökelpoinen. Alla olevassa kuvassa 4.1 on tähän tilanteeseen sopiva järjestelmä. Kohinaisesta signaalista `noisy` täytyy nyt luoda viivästetty versio. Tehdään se lisäämällä alkuun nollia: `noisy2=[zeros(1,50),noisy']';` Sovella nyt kuvan 4.1 menettelyä jaksollisen kohinakomponentin poistamiseen signaalista kun suodatetaan viivästettyä versiota ja referenssisignaalina on viivästämätön kohinainen signaali. Tulosta amplitudivaste ruudulle. Kuuntele signaali, josta vihellys on poistettu. Kiinnitä huomiota siihen, mistä kohtaa lohkokaaviota kyseinen signaali saadaan.



Kuva 4.1: Jaksollisen häiriön poistoon sopivan järjestelmän lohkokaavio.

4.3. *(Matlab)* Normalisoitu LMS-algoritmi (NLMS) saadaan LMS-algoritmista muuttamalla suppenemisparametrin μ laskenta adaptiiviseksi. Muokkaa funktiota `LMS.m` siten, että painoja päivitettäessä parametrin `mu` tilalla käytetään uutta parametria `mu2`, joka lasketaan kaavasta

$$\texttt{mu2} = \frac{\texttt{mu}}{1 + \texttt{data'*data}}.$$

Sijoita tämä laskukaava juuri ennen riviä, jossa painot päivitetään. Muuta vielä rivillä

```
w=w+2*mu*e*data';          % Päivitetään painoja
```

olevan parametrin `mu` tilalle uusi parametri `mu2`. Kokeile uutta algoritmia tehtävän 4.2. tilanteeseen. Anna parametrina `mu` nyt joku luku väliltä 0–1, mieluummin läheltä ykköstä.

4.4. Oletetaan, että seuraavien termien arvot (vrt. kappale 4.5) on arvioitu kahdesta signaalista:

$$\mathbf{R} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \qquad \mathbf{p} = \begin{bmatrix} 7 \\ 8 \end{bmatrix}, \qquad E[s^2(n)] = 42.$$

Merkitään lisäksi

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}.$$

Laske ja sievennä kustannusfunktion $c(\mathbf{w})$ lauseke.

4.5. Oletetaan, että $\mathbf{R}$, $\mathbf{p}$ ja $\mathbf{w}$ ovat kuten edellisessä tehtävässä. Laske (ja sievennä) kustannusfunktion gradientti ∇c sekä kustannusfunktion minimoiva vektori $\mathbf{w}$.

4.6. Mitä tapahtuu adaptiivisen suotimen kertoimille, jos referenssisignaali $s(n)$ ja kohdesignaali $x(n)$ eivät korreloi lainkaan? Toisin sanoen jokaisella kokonaisluvulla k on voimassa ehto $E[s(n)x(n-k)] = 0$. Perustele.

4.7. *(Matlab)* Muunna funktiota `LMS.m` siten, että viimeisen kierroksen kerroinvektorin `w` sijasta palautetaankin matriisi, jossa kukin rivi sisältää yhden iteraatiokierroksen kertoimet:

$$\begin{pmatrix} \text{1. kierroksen kertoimet} \\ \text{2. kierroksen kertoimet} \\ \text{3. kierroksen kertoimet} \\ \vdots \\ \text{kierroksen } M \text{ kertoimet} \end{pmatrix} = \begin{pmatrix} w_1(0) & w_1(1) & w_1(2) & \cdots & w_1(N-1) \\ w_2(0) & w_2(1) & w_2(2) & \cdots & w_2(N-1) \\ w_3(0) & w_3(1) & w_3(2) & \cdots & w_3(N-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_M(0) & w_M(1) & w_M(2) & \cdots & w_M(N-1) \end{pmatrix}.$$

Muuta myös kertoimien alustus sellaiseksi, että ensimmäisen iteraatiokierroksen suotimen kertoimet ovat ykkösiä (komento `ones`, eikä `zeros`). Luo sitten komennolla `randn` kaksi keskenään korreloimatonta signaalia: referenssisignaali `s` ja kohdesignaali `x`, joiden pituus on 30000. Aja muutettu LMS-algoritmi näillä signaaleilla ja tulosta kertoimen $w_k(0)$ kuvaaja kun $k = 1, 2, 3, \ldots, M$ komennolla `plot(w(:,1))`. Saatko kuvaajan täsmäämään edellisen tehtävän tuloksen kanssa?

4.8. *(Matlab)* LMS-algoritmi on yksi ratkaisu adaptiivisen suodatuksen toteutuksessa. Toinen yleisesti käytetty algoritmi on nimeltään RLS-algoritmi, joka on lyhenne sanoista *Recursive Least Squares*. Tällä kurssilla emme paneudu lähemmin itse algoritmiin, mutta tutustumme sen käyttöön. Kopioi tiedosto `RLS.m` itsellesi verkkoselaimella osoitteesta `http://www.cs.tut.fi/kurssit/SGN-1251/RLS.m`.

Tutkitaan LMS- ja RLS-algoritmien eroja kokeellisesti simuloiduilla testisignaaleilla. Testiä varten kannattanee lisätä komennot Matlab-skriptiin, jonka kaikki komennot ajetaan kerrallaan.

Luodaan ensin satunnainen, normaalijakautunut kohinasignaali:

```
x=10*randn(2000,1);
```

Suodatetaan tulos, jotta saataisiin korrelaatio näytteiden välille (ilman sitä LMS ja RLS eivät toimi).

```
x=filter([1 -3 2],[1 -0.5],x);
```

Tämä on nyt kohdesignaali. Olkoon referenssisignaali nyt tämän suodatettu versio:

```
y=filter([1,2,3,4],1,x);
```

Lisätään lopuksi referenssisignaaliin kohinaa, jotta tehtävä vaikeutuisi hieman:

```
y=y+randn(size(y));
```

RLS-algoritmi saa neljä parametria: kohdesignaalin, referenssisignaalin, suotimen pituuden ja *unohduskertoimen* (engl. *forgetting factor*). Unohduskerroin $\lambda \in [0, 1]$ kertoo sen, kuinka pitkältä ajalta algoritmi muistaa signaalien arvot. Esimerkiksi kertoimella $\lambda = 0.99$ algoritmin muisti on kertaluokkaa

$$\sum_{k=0}^{\infty} 0.99^k = \frac{1}{1 - 0.99} = 100 \text{ näytettä.}$$

Mitä suurempi unohduskerroin, sitä pidempi muisti algoritmilla on. Suurilla kertoimilla algoritmi ei kykene mukautumaan lyhyen aikavälin muutoksiin. Toisaalta pienet kertoimet saavat algoritmin liiankin herkäksi pienille muutoksille.

Vertaile LMS- ja RLS-algoritmeja edellämainittujen kohde- ja referenssisignaalin tapauksessa. Suotimen pituudeksi riittää 4. Tulosta ruudulle molempien menetelmien virhesignaalit (referenssin ja suodatustuloksen erotus). Kokeile eri arvoja RLS:n unohduskertoimelle ja LMS:n suppenemisparametri μ:lle. Katso myös `help RLS`.

4.9. *(Matlab) Ekvalisointi.* Matlabin testisignaali `handel` on kulkenut erään (simuloidun) tiedonsiirtokanavan läpi ja samalla taajuudet ovat vääristyneet. Lataa (`wavread`) tulossignaali tiedostosta

```
http://www.cs.tut.fi/kurssit/SGN-1251/corrupt.wav
```

Tehtävänäsi on käyttää adaptiivista järjestelmää (LMS tai RLS valintasi mukaan) etsimään kanavan kertoimet, ts. etsimään se FIR-suodin, joka muuntaa signaalin `handel` signaaliksi `corrupt.wav`. Estimoi ajan säästämiseksi kanavan kertoimia signaaliparin 5000:sta ensimmäisestä näytteestä. Voit olettaa, että mallissa on kertoimia enintään 15. Ekvalisoi vääristynyt signaali kanavan käänteissuotimella[4], toisin sanoen

```
ekvalisoitu=filter(1,kanava,corrupt);
```

missä vektorissa `kanava` on estimoidut kanavan kertoimet ja `corrupt` on vääristynyt signaali. Kuuntele alkuperäinen, vääristynyt sekä ekvalisoitu signaali. Tulosta ruudulle kanavan impulssivaste ja taajuusvaste sekä toiseen ikkunaan ekvalisointisuotimen taajuusvaste (`freqz(1,kanava)`).

[4]Tämä onnistuu vain, jos kanavan $H(z)$ kaikki nollat ovat yksikköympyrän sisäpuolella. Tämä siksi, että ne muuttuvat käänteissuotimen $1/H(z)$ navoiksi. Tällaisen suotimen sanotaan olevan *minimivaiheinen*. Tämän tehtävän kanava valittiin tarkoituksella minimivaiheiseksi.

4.10. *(Matlab) Ekvalisointi II.* Toinen vaihtoehto ekvalisoinnille on estimoida adaptiivisella järjestelmällä suoraan kanavan käänteissuodinta (edellähän estimoitiin kanava ja *laskettiin* sen käänteissuodin). Tällöin siis pyritään etsimään se FIR-suodin, joka muuntaa signaalin `corrupt.wav` signaaliksi `handel`. Ongelmaksi muodostuu se, että teoriassa käänteissuodin on IIR-suodin, ja sitä estimoidaan FIR-suotimella. Etsi adaptiivisella menettelyllä paras tällainen FIR-suodin ja tulosta ruudulle kanavan impulssivaste. Vertaa tulosta edellisen tehtävän ekvalisointisuotimen taajuusvasteeseen. Nyt kannattaa valita pidempi suodin, luokkaa $30 - 40$ näytettä.

Amplitudivasteesta ei tule tällä menettelyllä tarkalleen oikeaa käänteiskanavaa, mutta toisaalta tulos ei ole myöskään epästabiili. Edellisen tehtävän ongelmana on oletus kanavan minimivaiheisuudesta, mikä ei useimmiten pidä paikkaansa.

4.11. *(Matlab) Ennustus.* Lataa Matlabin testisignaali `handel` ja erota sen alusta 10000:n näytteen mittainen pätkä: `y=y(1:10000);`. Tee tuloksesta kaksi uutta versiota: kohdesignaali `x`, jonka alussa on nolla ja referenssisignaali `s`, jonka lopussa on nolla (`x=[0,y']'; s=[y',0]';`). Ajettaessa LMS:ää näillä signaaleilla algoritmi minimoi suodatustuloksen ja referenssisignaalin erotusta. Koska kohdesignaali on viivästetty versio referenssisignaalista, algoritmi yrittää käytännössä ennustaa seuraavaa arvoa aikaisemmista arvoista. Aseta kertoimien määräksi $N = 11$ ja ala kasvattaa parametrin μ arvoa arvosta $\mu = 0.01$ alkaen kunnes tulos karkaa käsistä. Tulosta ennustusvirhe ja tutki kuinka hyvin ennustus onnistui.

4.12. Kappaleen 4.7.1 esimerkin viimeisessä kuvaajassa on adaptiivisen suotimen ulostulo $y(n)$. Tarkkaan katsottuna kuvaajassa näkyy pienempi huippu myös taajuudella 0.15. Miten tämä on mahdollista, kun suotimen heräte sisältää ainoastaan taajuutta 0.05?

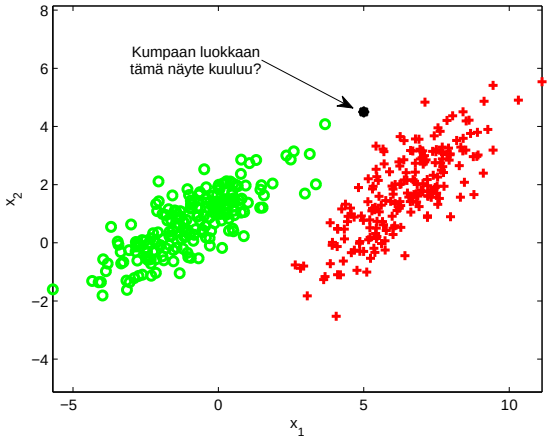
Luku 5

Oppivat järjestelmät

Kappaleessa 4 tutustuttiin järjestelmiin, jotka mukautuvat ulkoisiin olosuhteisiin ja oppivat suodatustehtävän parametrit tilanteen mukaan. Tämä menetelmä voidaan yleistää ns. *oppiviin järjestelmiin*, joille opetetaan laskennallinen ongelma esimerkkien avulla. Yksi merkittävä osa tähän luokkaan kuuluvia järjestelmiä ovat ns. *luokittelijat*, joiden täytyy esimerkkien perusteella oppia luokittelemaan moniulotteinen data ($\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbf{R}^n$) äärelliseen määrään luokkia: $y_1, y_2, \ldots, y_N \in \{1, 2, \ldots, M\}$.

Oppivien järjestelmien sovelluskohteisiin signaalinkäsittelyssä kuuluvat mm. tekstintunnistus ja puheentunnistus. Näiden lisäksi menetelmiä käytetään esimerkiksi hakukoneissa, lääketieteellisessä diagnostiikassa sekä mm. sähköpostisuodattimissa. Kaikille näille järjestelmille on yhteistä, että ne oppivat luokittelun esimerkkien avulla: esimerkiksi tekstintunnistuksessa luokittelijalle näytetään kustakin kirjaimesta satoja tai tuhansia esimerkkejä, joiden perusteella opetusalgoritmi valitsee luokittelijalle sopivat parametrit. Esimerkiksi käsinkirjoitettujen merkkien tunnistuksessa yleisesti käytetty MNIST-tietokanta koostuu yhteensä noin 70000 käsinkirjoitettua kirjainta esittävästä bittikartasta.

Yleisesti käytettyihin luokittelumenetelmiin kuuluvat mm. *Fisherin lineaarinen erottelija*, *tukivektorikone* sekä *hermoverkko*. Menetelmiä tarkastellaan jatkossa esimerkin avulla, jossa alla olevan kuvan mukaisten kaksiulotteisten joukkojen (punaiset ristit ja vihreät ympyrät) avulla pitäisi oppia päättelysääntö uusien näytteiden luokittelemiseksi (kuten kuvan musta pallo).



Ensinäkemältä ongelma saattaa vaikuttaa helpolta, koska tässä kaksiulotteisessa ta-

pauksessa kuka tahansa osaa piirtää kohtuullisen hyvän rajan joukkojen välille. Ongelmasta tulee kuitenkin haasteellisempi kun joukkojen dimensio kasvaa ja ne menevät osin päällekkäin. Esimerkiksi edellä mainitun MNIST-tietokannan näytteet ovat 20×20 pikselin kokoisia kuvia, joten avaruus on 400-ulotteinen.

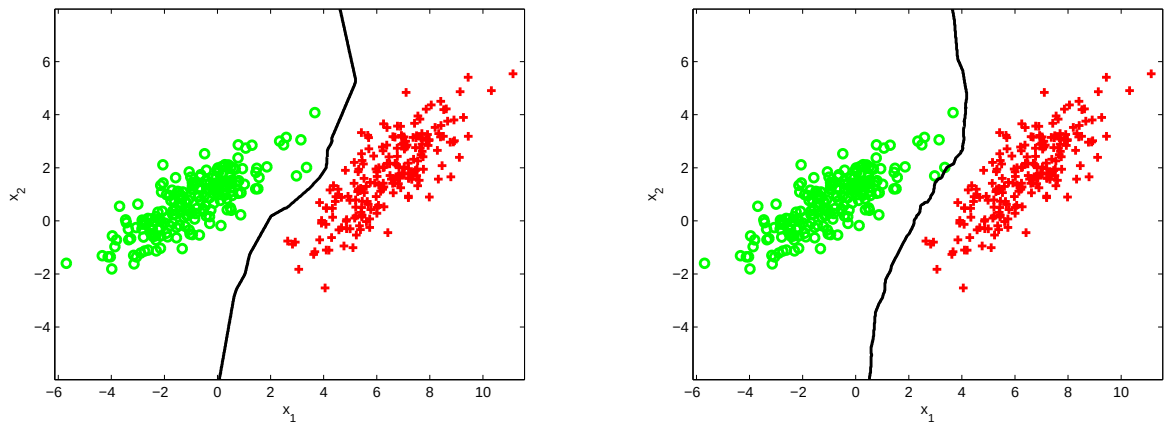
Tärkeitä käsitteitä luokittelussa sekä yleensäkin koneoppimisessa ovat *opetusjoukko* ja *testijoukko*. Opetusjoukkoa käytetään nimensä mukaisesti järjestelmän opetuksessa, mutta lopputuloksen toimivuutta arvioidaan erillisen testijoukon avulla. Todellisessa tilanteessahan luokittelija luokittelee enimmäkseen opetusjoukkoon kuulumattomia näytteitä, joten opetusjoukon luokitteluvirhe ei ole luotettava mittari. Esimerkiksi edellä mainittu MNIST-tietokannan 70000 merkkiä on jaettu 60000:n opetusmerkkiin ja 10000:n testikuvaan.

5.1 Lähimmän naapurin luokittelija

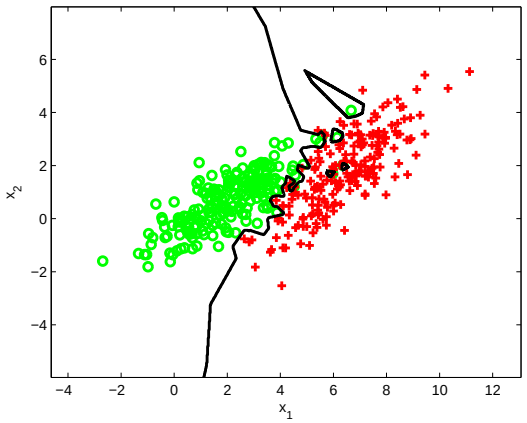
Yksinkertainen ja helposti toteutettava ratkaisu luokitteluongelmaan on niin sanottu lähimmän naapurin luokittelija (engl. Nearest Neighbor Classifier; NN-classifier). Nimensä mukaisesti menetelmä etsii opetusaineistosta samankaltaisimman näytteen ja valitsee luokiteltavan pisteen luokan sen perusteella. Yleisempi versio tästä varsin intuitiivisesta luokittelijasta on ns. k-NN-luokittelija, joka etsii opetusdatasta k samankaltaisinta näytettä ja valitsee näiden joukosta yleisimmän luokan.

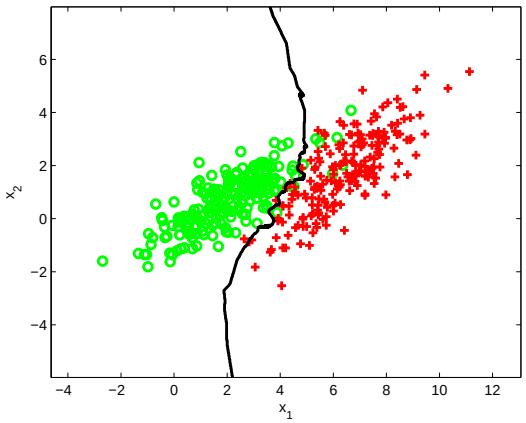
Menetelmä ei siis vaadi erillistä opetusvaihetta, vaan se muistaa koko opetusmateriaalin sellaisenaan. Tämä on myös menetelmän suurin ongelma: uuden näytteen luokkaa pääteltäessä täytyy koko aineisto käydä läpi ja etsiä sieltä lähimpänä olevat. Lisäksi koko aineiston täytyy olla muistissa, mikä voi tulla ongelmaksi suurilla aineistoilla. Pienille opetusdatan määrille k-NN sen sijaan toimii yleensä yhtä hyvin kuin mikä tahansa muukin luokittelija, ja onkin usein käytetty helpohkoissa luokitteluongelmissa, joissa luokat erottuvat toisistaan selvästi.

Alla olevassa kuvassa on piirretty k-NN-luokittelijan luokkarajat aiemmin esillä olleen kahden luokan testidatan tapauksessa. Vasemmanpuoleisessa kuvassa on 1-NN-luokittelijan tulos ja oikeanpuoleisessa 9-NN-luokittelijan tulos. Kuvasta nähdään, että suuremmilla parametrin k arvoilla menetelmästä tulee epäherkempi yksittäisille poikkeaville näytteille. Oikeanpuoleisessa kuvassa oikeanpuolimmaiset vihreät ympyrät eivät vaikuta luokkarajaan yhtä paljon, koska ne ovat melko irrallisia muista näytteistä ja näin ollen sattuman tulosta.



Jos luokat ovat enemmän päällekkäin, tulee 1-NN-luokittelijan luokkarajasta melko rikkonainen. Alla on esimerkki tällaisesta tilanteesta. Vasemmalla olevassa 1-NN-luokittelijan tuloksessa jokainen näytepiste vaikuttaa lopputulokseen, ja luokkaraja koostuu monesta irrallisesta osasta. Oikealla olevassa 9-NN-luokittelijan kuvassa luokkaraja on yhtenäinen ja luultavasti paremmin toimiva.





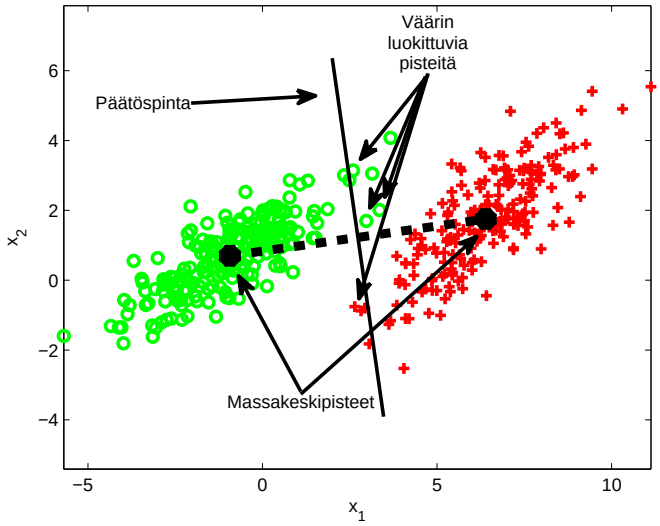
Seuraavaksi tutustutaan monimutkaisempiin luokittelijoihin, joiden esitysmuoto on k-NN-luokittelijaa kompaktimpi. Kompakti esitys voi olla esimerkiksi funktio $f(\mathbf{x})$, jonka arvo (esim. positiivinen vai negatiivinen) määrää näytteen $\mathbf{x}$ luokan. Tällöin luokan valinta on nopeampaa, mutta hyvän funktion f päättely vie enemmän aikaa. Tämä on yleensä toivottu toimintatapa, koska opetusvaihe ei ole reaaliaikainen eikä opetuksen kesto ole niin kriittinen tekijä kuin itse luokittelun.

5.2 Lineaarinen erottelija

Fisherin lineaarinen erottelija eli lineaarinen diskriminantti (*Linear Discriminant Analysis; LDA*) on julkaistu jo vuonna 1936, mutta on edelleen yleisesti käytetty ratkaisu pääasiassa yksinkertaisuutensa vuoksi. Perusideana on ratkaista se suora, jolle projisoitu moniulotteinen data maksimoi luokkien välisen varianssin (eli vie luokat mahdollisimman kauas toisistaan).

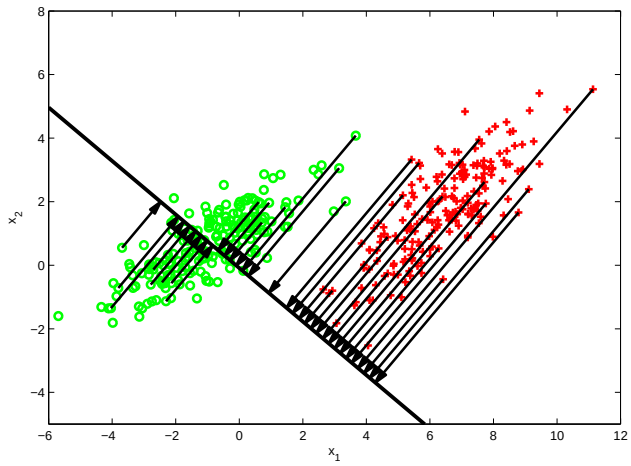
Ensimmäinen mieleen tuleva ratkaisu on vetää raja suunnilleen joukkojen puoliväliin. Kaksiulotteiselle datalle tämä voidaan määrittää käsinkin, mutta esim. sataulotteiselle datalle tarvitaan automaattinen menetelmä rajan (eli ns. päätöspinnan) määräämiseksi.

Yksinkertaisin ratkaisu on piirtää jana joukkojen massakeskipisteiden välille, ja vetää (ortogonaalinen) päätöspinta niiden puoliväliin. Tämä ratkaisu on esitetty alla olevassa kuvassa.



Tuloksesta nähdään, että ratkaisu ei ole paras mahdollinen, jos luokat ovat vähänkään limittäin. Nyt nimittäin vääriä päätöksiä jää valittiinpa rajan sijainti miten hyvänsä, jos suunta on kuvan mukainen.

Parempi ratkaisu olisi suora, jolle projisoitaessa pistejoukot tulevat toisistaan erillisiksi yksiulotteisiksi joukoiksi. Esimerkiksi alla olevassa kuvassa on edellä olleet kaksi pisteparvea projisoitu eräälle tällaiselle suoralle.[1]

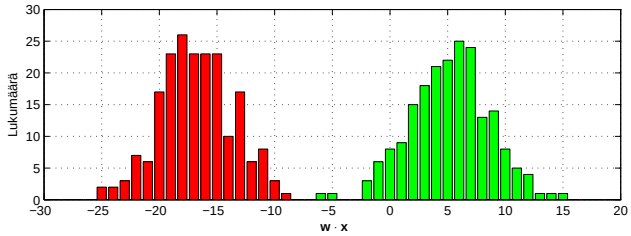


Kuvion perusteella havaitaan, ettei erottelun kannalta ole väliä missä suora sijaitsee, ainoastaan sen suunnalla on merkitystä. Tämän vuoksi riittää tarkastella pelkästään suoran suunnan määrävää vektoria $\mathbf{w}$. Pisteen $\mathbf{x}$ projektio vektorille $\mathbf{w}$ määritellään kaavalla

$$\mathrm{proj}_{\mathbf{w}}(\mathbf{x}) = \frac{\mathbf{w} \cdot \mathbf{x}}{|\mathbf{w}|^2}\mathbf{w}.$$

Projisoidun pisteen sijainti suoralla määräytyy pistetulon $\mathbf{w} \cdot \mathbf{x}$ perusteella, joten yksinkertaisuuden vuoksi riittää tarkastella pelkästään sitä. Suureen $\mathbf{w} \cdot \mathbf{x}$ jakauma on esitetty alla olevassa kuvassa.

[1] Muistin virkistämiseksi: pisteen projektio kuvaa sen lähimmäksi suoralla olevaksi pisteeksi.



Näyttäisi siis, että käytetty suora on hyvä valinta, ja esimerkiksi seuraava yksinkertainen algoritmi vaikuttaisi olevan hyvä luokittelija kyseiselle datalle:

$$\text{Pisteen } \mathbf{x} \text{ luokka} = \begin{cases} \text{vihreä ympyrä, jos} & \mathbf{w} \cdot \mathbf{x} \geq -7.5, \\ \text{punainen risti, jos} & \mathbf{w} \cdot \mathbf{x} < -7.5. \end{cases}$$

Kuinka vektori $\mathbf{w}$ sitten valitaan? Tässä yksinkertaisessa tapauksessa se onnistuu käsinkin, mutta 100-ulotteisten osin päällekkäin olevien luokkien tapauksessa automaattinen menetelmä on tarpeen. Fisher muotoili kysymyksen optimointiongelman muotoon: mikä on se vektori $\mathbf{w}$, jolle projisoitaessa luokkien välinen varianssi maksimoituu suhteessa luokkien sisäiseen varianssiin. Kyseinen vektori saadaan ratkaistua kirjoittamalla Fisherin määritelmän mukaiset lausekkeet auki ja sieventämällä tulos. Melko suoraviivaisen matriisilaskennan avulla voidaan osoittaa että tämä suure saa muodon[2]

$$J(\mathbf{w}) = \frac{\text{luokkien välinen varianssi}}{\text{luokkien sisäinen varianssi}} = \frac{\mathbf{w}^T(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T\mathbf{w}}{\mathbf{w}^T(\mathbf{C}_1 + \mathbf{C}_2)\mathbf{w}},$$

missä $\boldsymbol{\mu}_1$ ja $\boldsymbol{\mu}_2$ ovat luokkien keskipisteet ja $\mathbf{C}_1$ ja $\mathbf{C}_2$ niiden otoskovarianssimatriisit. Etsimällä gradientin nollakohta voidaan osoittaa, että tämän lausekkeen maksimoiva vektori $\mathbf{w}$ on

$$\mathbf{w} = (\mathbf{C}_1 + \mathbf{C}_2)^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2).$$

Edellä olleen esimerkin tapauksessa kovarianssimatriiseilla ja keskipisteillä on seuraavat arvot

$$\mathbf{C}_1 = \begin{pmatrix} 2.4775 & 1.3271 \\ 1.3271 & 1.1239 \end{pmatrix}, \quad \mathbf{C}_2 = \begin{pmatrix} 2.2666 & 1.7515 \\ 1.7515 & 2.1657 \end{pmatrix}, \quad \boldsymbol{\mu}_1 = \begin{pmatrix} -0.9330 \\ 0.6992 \end{pmatrix}, \quad \boldsymbol{\mu}_2 = \begin{pmatrix} 6.3992 \\ 1.7508 \end{pmatrix}$$

jolloin $\mathbf{w} = [-3.4077, 2.8695]^T$, joka on aiemmassa kuvassa olleen projektiosuoran suuntavektori. Huomaa, että kaikki samansuuntaiset (ja vastakkaissuuntaiset) vektorit kelpaavat yhtä lailla. Usein onkin tapana normalisoida vektori ykkösen mittaiseksi: $\mathbf{w}' = \mathbf{w}/\|\mathbf{w}\|$.

Näin saadun luokittelijan ohjelmallinen toteutus on äärimmäisen yksinkertainen. Uuden näytteen $\mathbf{x} = (x(1), x(2))$ luokka päätellään laskemalla pistetulo

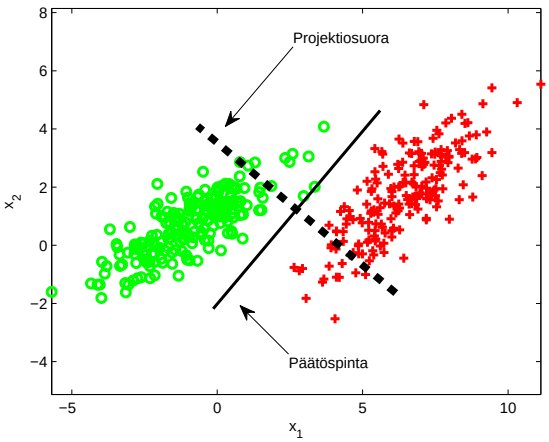
$$\mathbf{x} \cdot \mathbf{w} = -3.4077x(1) + 2.8695x(2)$$

ja vertaamalla tulosta kynnysarvoon c (esim. edellä ollut raja $c = -7.5$). Todellisessa tilanteessa kynnyksen valinta "kuvasta katsomalla"on liian epätarkkaa, joten tähänkin tarvitaan automaattinen menetelmä. Paras tapa on valita c niin, että väärin menneiden opetusnäytteiden osuus on mahdollisimman pieni (esimerkin tapauksessa kaikki kynnykset

[2]Huomaa, että kaavassa käytetään pistetulon kanssa yhtäpitävää matriisituloa $\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T\mathbf{x}$.

väliltä $c \in [-8.9267, -5.6639]$ tuottavat nollavirheen, joten ne kaikki kelpaisivat). Yksinkertaisempi tapa on valita kynnys joukkojen keskipisteiden puolivälistä (ks. harjoitus 5.3), jolloin saadaan arvo $c = -5.7984$.

Tällä kynnysarvolla saatava päätöspinta on alla olevassa kuvassa. Kuvasta nähdään menetelmän jakavan opetusjoukot kahtia täysin oikein.



Päätöspinnan lähellä on kuitenkin nyt kaksi vihreää pistettä, jotka luokittuvat juuri ja juuri oikein. Tätä ratkaisua turvallisemmalta tuntuisikin valinta, jossa marginaali päätöspinnan molemmin puolin olisi mahdollisimman suuri. Tähän ajatukseen perustuvat tukivektorikoneet, joita käsitellään seuraavaksi.

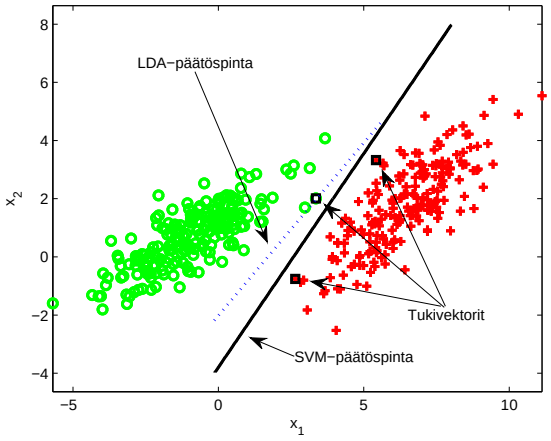
5.3 Tukivektorikoneet

Tukivektorikone (*engl. support vector machine (SVM)*) kehitettiin alunperin 1960-luvulla Moskovassa[3], mutta laajempi suosio syntyi vasta 1990-luvun lopulla tehokkaiden laskentamenetelmien sekä kehittyneemmän teorian myötä. Menetelmän ideana on valita se päätöspinta, joka maksimoi luokkien väliin jäävän marginaalin (eli on mahdollisimman keskellä pisteparvien välissä). Esimerkiksi yllä ollut LDA:n päätöspinta on piirretty luokkien keskipisteiden puoliväliin, mutta se sattuu olemaan lähempänä vihreitä ympyröitä kuin punaisia ristejä. SVM piirtää päätöspinnan niin, että etäisyys molempien luokkien lähimpiin alkioihin on mahdollisimman suuri.

Tukivektoreiden ja päätöspinnan ratkaisu on varsin matemaattinen ja tulee vastaan myöhemmillä kursseilla. SVM:stä on kuitenkin useita toteutuksia, joita voi käyttää vaikka algoritmin toiminta ei olisikaan täysin selvä[4]. Optimiratkaisu on kuvattu alla olevassa kuvassa. Kuvan sininen katkoviiva kuvaa LDA:n päätöspintaa ja musta yhtenäinen viiva SVM:n päätöspintaa, joka lepää ns. *tukivektoreiden* varassa. Kaikkiin kolmeen tukivektoriin on nyt sama etäisyys (eli marginaali) ja tukivektorit on merkitty mustilla neliöillä.

[3]V. Vapnik ja A. Lerner, "Pattern recognition using generalized portrait method," *Automation and Remote Control*, 1963.

[4]Suosittuja toteutuksia ovat mm. LibSVM sekä Matlabin Bioinformatics-toolboxin toteutus.



Tukivektorikoneet siis maksimoivat päätöspinnan ympärille jäävän marginaalin ja valitsevat tukivektorit sen mukaan. Tämä ominaisuus ei pääse täysin oikeuksiinsa lineaarisen päätöspinnan tapauksessa, koska LDA:n päätöspinta on usein melko lähellä SVM:n pintaa. Marginaalin maksimointi kuitenkin korostuu kun data on korkeampiulotteista, ja marginaalit tyypillisesti suuremmat.

Tähän havaintoon perustuu SVM:n suosio ns. *kernelitempun* (*engl. kernel trick*) yhteydessä, joka esiteltiin jo vuonna 1964 käytettäväksi LDA:n kanssa.[5] Se ei kuitenkaan saavuttanut merkittävää suosiota, koska se ei toimi LDA:n kanssa erityisen hyvin. Kernelitempun idea on kuvata data korkeampiulotteiseen avaruuteen ns. kernelin eli ytimen avulla. Ytimeksi sopii mikä tahansa tietyt ehdot toteuttava funktio, ja tarkoituksena on että data olisi luokiteltavissa korkeammassa ulottuvuudessa lineaarisen päätöspinnan avulla. Ratkaisua kutsutaan tempuksi sen vuoksi, että sen toteuttamiseksi dataa ei tarvitse eksplisiittisesti siirtää korkeampaan ulottuvuuteen, vaan riittää pelkästään korvata kaikki algoritmin pistetulot sopivalla funktiolla. Menetelmää voidaan näin ollen käyttää kaikilla menetelmillä, jotka perustuvat vektoreiden pistetuloon (kuten mm. LDA ja SVM).

Ytimiä ja sitä kautta erilaisia kuvauksia on olemassa lukuisia. Yksinkertaisimmat ytimet ovat ns. polynomiytimiä, jotka muodostetaan nostamalla pistetulo johonkin kokonaislukupotenssiin. Esimerkiksi toisen asteen polynomiydin vastaa normaalin pistetulon korvaamista sen toisella potenssilla:

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2 .$$

Esimerkin tapauksessa data oli kaksiulotteista, joten vektoreiden $\mathbf{x}_i = (\mathbf{x}_i(1), \mathbf{x}_i(2))$ ja $\mathbf{x}_j = (\mathbf{x}_j(1), \mathbf{x}_j(2))$ pistetulo määritellään kaavalla

$$\mathbf{x}_i \cdot \mathbf{x}_j = \mathbf{x}_i(1)\mathbf{x}_j(1) + \mathbf{x}_i(2)\mathbf{x}_j(2).$$

[5] M. Aizerman, E. Braverman, and L. Rozonoer, "Theoretical foundations of the potential function method in pattern recognition learning". *Automation and Remote Control*, 1964.

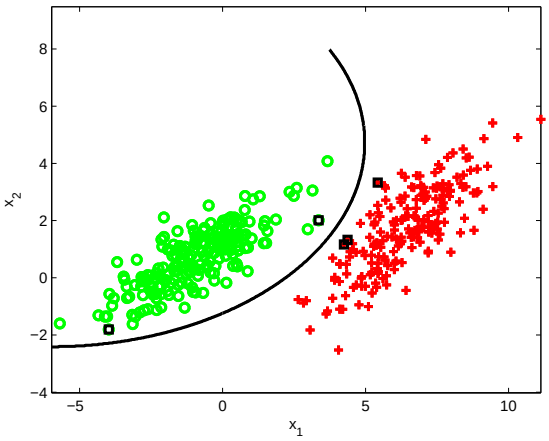
Pistetulon toinen potenssi on näin ollen lauseke

$$\begin{aligned}
k(\mathbf{x}_i, \mathbf{x}_j) &= \left[\mathbf{x}_i(1)\mathbf{x}_j(1) + \mathbf{x}_i(2)\mathbf{x}_j(2)\right]^2 \\
&= \left[\mathbf{x}_i(1)\mathbf{x}_j(1) + \mathbf{x}_i(2)\mathbf{x}_j(2)\right] \cdot \left[\mathbf{x}_i(1)\mathbf{x}_j(1) + \mathbf{x}_i(2)\mathbf{x}_j(2)\right] \\
&= \mathbf{x}_i^2(1)\mathbf{x}_j^2(1) + \mathbf{x}_i^2(2)\mathbf{x}_j^2(2) + 2\mathbf{x}_i(1)\mathbf{x}_j(1)\mathbf{x}_i(2)\mathbf{x}_j(2) \\
&= \mathbf{x}_i^2(1)\mathbf{x}_j^2(1) + \mathbf{x}_i^2(2)\mathbf{x}_j^2(2) + (\sqrt{2}\mathbf{x}_i(1)\mathbf{x}_i(2)) \cdot (\sqrt{2}\mathbf{x}_j(1)\mathbf{x}_j(2)) \\
&= (\mathbf{x}_i^2(1), \mathbf{x}_i^2(2), \sqrt{2}\mathbf{x}_i(1)\mathbf{x}_i(2))^T \cdot (\mathbf{x}_j^2(1), \mathbf{x}_j^2(2), \sqrt{2}\mathbf{x}_j(1)\mathbf{x}_j(2))^T
\end{aligned}$$

Kernelitempun idea on siis hoksata, että tämäkin lauseke on eräs pistetulo kolmiulotteisessa avaruudessa. Kuten viimeisestä lausekkeesta nähdään, niin samaan tulokseen päästään kuvaamalla alkuperäinen 2-ulotteinen vektori $\mathbf{x} = (\mathbf{x}(1), \mathbf{x}(2))$ kolmiulotteiseen avaruuteen kuvauksella

$$\begin{pmatrix} \mathbf{x}(1) \\ \mathbf{x}(2) \end{pmatrix} \mapsto \begin{pmatrix} \mathbf{x}(1)^2 \\ \mathbf{x}(2)^2 \\ \sqrt{2}\mathbf{x}(1)\mathbf{x}(2) \end{pmatrix}. \tag{5.1}$$

Johtopäätöksenä voidaan siis todeta, että pistetulojen nostaminen toiseen potenssiin vastaa normaalia SVM:ää kolmiulotteisessa tapauksessa, jossa näytteet on saatu kaavalla 5.1. Alla oleva kuva esittää luokittelun tulosta.



Edellisessä tapauksessa sama luokittelija olisi voitu toteuttaa ilman kernelitemppuakin, yksinkertaisesti suunnittelemalla kolmiulotteinen SVM-luokittelija kuvauksen 5.1 jälkeiselle datalle. Eksplisiittinen kuvaus korkeampaan dimensioon ei kuitenkaan ole mahdollista, jos dimensio on kovin suuri. Tällainen tilanne syntyy esimerkiksi käytettäessä epähomogeenista polynomiydintä

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d,$$

joka tuo mukaan kaikki enintään astetta d olevat monomit. Kuvaus 5.1 muuttuisi tässä tapauksessa muotoon

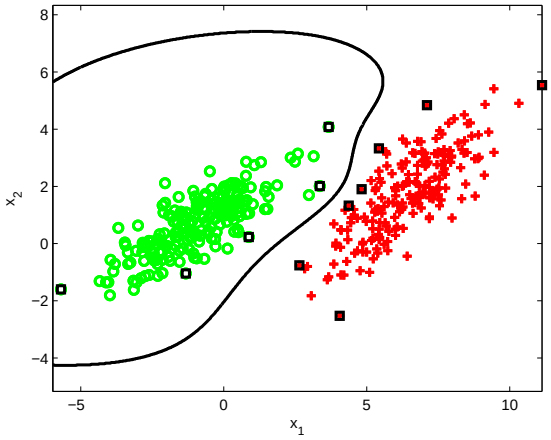
$$\begin{pmatrix} \mathbf{x}(1) \\ \mathbf{x}(2) \end{pmatrix} \mapsto \begin{pmatrix} \mathbf{x}(1)^2 \\ \mathbf{x}(2)^2 \\ \sqrt{2}\mathbf{x}(1)\mathbf{x}(2) \\ \sqrt{2}\mathbf{x}(1) \\ \sqrt{2}\mathbf{x}(2) \\ 1 \end{pmatrix}. \tag{5.2}$$

Jos nyt esimerkiksi alkuperäinen data olisikin viisiulotteista ja esim. $d = 8$, suoraan kuvaukseen tulisi kymmeniä termejä ja kernelitemppu alkaa vaikuttaa entistä houkuttelevammalta.

Äärimmäisenä esimerkkinä kernelitempun mahdollisuuksista on yleisesti käytetty ns. RBF-ydin (radial basis function), joka vastaa kuvausta *ääretönulotteiseen* avaruuteen. RBF-ydin korvaa vektoreiden pistetulon $\mathbf{x}_i \cdot \mathbf{x}_j$ funktiolla

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2),$$

missä parametri $\gamma > 0$. Voidaan osoittaa, että tämä temppu vastaa lineaarisen päätöspinnan hakua ääretönulotteisessa avaruudessa. Alla olevassa kuvassa on esimerkki päätöspinnasta RBF-ydintä käytettäessä tapauksessa $\gamma = 1$.



Vaikka edellä oletettiinkin, että luokat ovat eroteltavissa toisistaan lineaarisella päätöspinnalla (jotta marginaali olisi ylipäätään olemassa), ei käytännössä näin useinkaan ole. Tämän vuoksi SVM-algoritmi on yleistetty myös tilanteisiin joissa joukot menevät päällekkäin, ja kaikki järkevät toteutukset tukevat tätä.

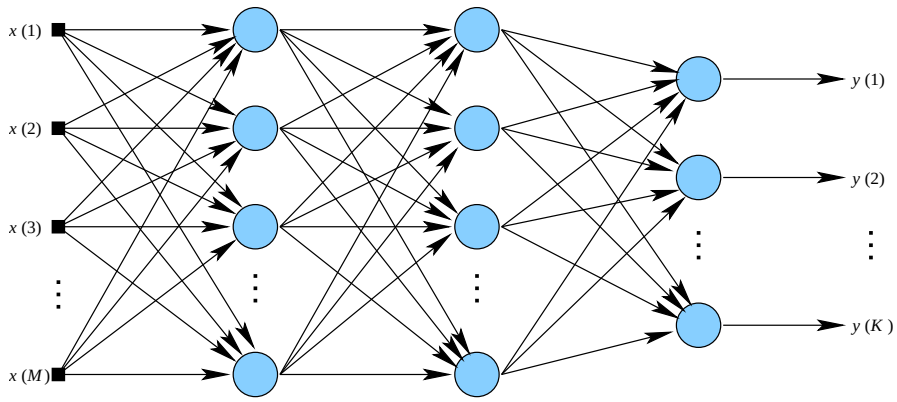
SVM on saavuttanut huomattavan suosion, mikä johtuu ensisijaisesti sen tavasta maksimoida luokittelumarginaali. Tämä nimittäin helpottaa opetusta ja saa luokittelijan toimimaan hyvin jo pienilläkin opetusjoukoilla. Muilla luokittelijoilla on nimittäin vaarana ns. *ylioppiminen*, josta on esimerkki hermoverkkojen yhteydessä. Yleisen käsityksen mukaan SVM ei ole yhtä herkkä opetusjoukon koolle kuin muut luokittelijat.

5.4 Hermoverkot

Hermoverkot perustuvat suurelta osiin samaan teoriaan kuin adaptiiviset suotimetkin: niiden parametrit lasketaan LMS-algoritmin yleistetyllä versiolla ja ne oppivat ratkaisemaan ongelmia annettujen esimerkkien perusteella.

Hermoverkkojen taustalla on nimen mukaisesti aivojen toiminnan mallintaminen. Aivot koostuvat suuresta määrästä toisiinsa yhteydessä olevia *neuroneja*. Myös keinotekoiset hermoverkot koostuvat suuresta määrästä yksinkertaisia laskentayksiköitä, jotka välittävät toisilleen tietoa. Hermoverkkojen tutkimus alkoi 1950-luvulla, jolloin tietokoneita alettiin käyttää aivojen toiminnan mallintamiseen. Vuonna 1958 esiteltiin *perseptronin* (engl. perceptron) käsite, joka oli malli yksittäiselle hermosolulle. Mallin yksinkertaisuus rajasi kuitenkin sen mahdollisuuksia, eikä tutkimus olennaisesti edennyt ennen vuonna 1974

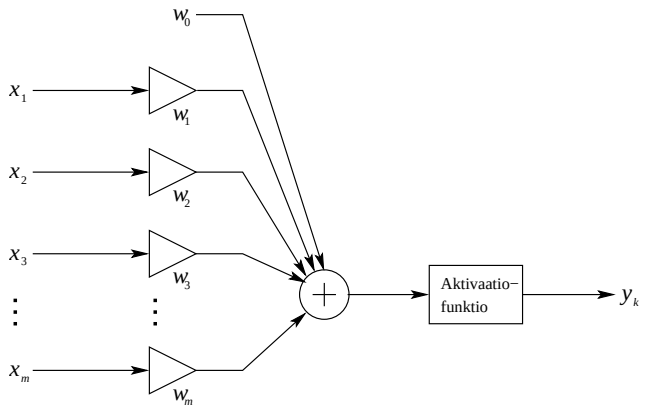
esiteltyä *backpropagation*-opetusalgoritmia, jonka avulla verkoksi yhdistettyjen perseptronien opetus tuli mahdolliseksi.



Yllä on kuva keinotekoisesta hermoverkosta. Hermoverkko koostuu *kerroksista (layer)*, joiden jokaisesta neuronista on yhteys jokaiseen seuraavan kerroksen neuroniin. Jokainen neuroni suorittaa parametriensa määräämän laskentaoperaation ja välittää saamansa tiedon eteenpäin seuraavalle kerrokselle.

Kuvassa vasemmalla on *herätekerros (input layer)*, jonka solmut vain välittävät lukuarvot $x(1), x(2), \ldots, x(M)$ hermoverkon ensimmäiselle kerrokselle. Ensimmäisen, toisen ja kolmannen (*piilokerros, hidden layer*) kerroksen solmut suorittavat eräitä laskuoperaatioita näille luvuille ja välittävät saamansa lukuarvot edelleen, kunnes ne lopulta saapuvat *ulostulokerrokselle (output layer)* oikealla.

Neuronin rakenne on kuvattu alla. Kuviosta havaitaan, että rakenne on itse asiassa tuttu FIR-suotimesta: herätearvot $x_1, x_2, \ldots, x_m$ kerrotaan painoilla $w_1, w_2, \ldots, w_m$ ja saadut tulos summataan yhteen. Erona FIR-suotimeen on summauksen jälkeen tuleva epälineaarinen aktivaatiofunktio, joka simuloi luonnossa neuronien välittämää aktivaatiota (yksinkertaistettuna hermosolu välittää muille tiedon onko se aktivoitunut vai ei). Lisäksi summauksessa on aina mukana vakiotermi w_0.



Yksittäisen neuronin toimintaa voidaan siis kuvata kaavalla

$$y_k = \psi\left(w_0 + \sum_{k=1}^{m} w_k x_k\right),$$

missä $\psi(\cdot)$ on aktivaatiofunktio.

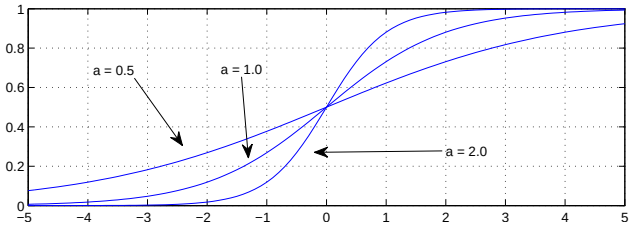
Useimmiten aktivaatiofunktiona käytetään jotain S-kirjaimen muotoista, ns. *sigmoidifunktiota*. Tällainen funktio on esimerkiksi *hyperbolinen tangentti*

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

ja *logistinen funktio*:

$$\psi(x) = \frac{1}{1 + e^{-\alpha x}},$$

missä parametri α määrää kuinka jyrkästi funktio nousee origon lähellä. Jatkossa keskitymme näistä funktioista jälkimmäiseen. Logistisen funktion kuva kolmella eri kertoimen α arvolla on alla.



Kertoimen α kasvaessa funktio lähestyy jatkuvaa askelfunktiota

$$u(x) = \begin{cases} 0, & \text{kun } x < 0, \\ 1, & \text{kun } x \geq 0. \end{cases}$$

Näin ollen aktivaatiofunktio päättää aktivoituuko neuroni vai ei, eli onko neuroniin tulevien ärsykkeiden määrä riittävän suuri. Toisin sanoen, toteuttaako herätteiden painotettu summa ehdon:

$$\sum_{k=1}^{m} w_k x_k \geq -w_0.$$

Tämä tulkinta tulee erityisen hyvin selville äärimmäisessä tapauksessa $\alpha \to \infty$, jossa aktivaatiofunktiosta tulee askelfunktio ja neuronin toiminta saa yksinkertaisen muodon:

$$y_k = \begin{cases} 0, & \text{kun } \sum_{k=1}^{m} w_k x_k < -w_0, \\ 1, & \text{kun } \sum_{k=1}^{m} w_k x_k \geq -w_0. \end{cases}$$

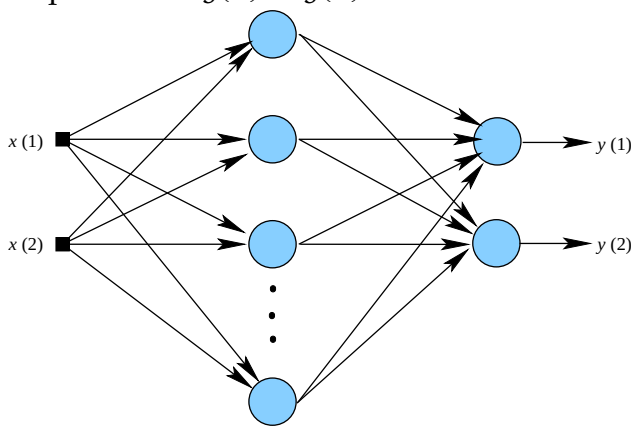
Neuroni siis aktivoituu, jos siihen kohdistuu riittävästi ärsykkeitä edellisen tason neuroneilta. Edelleen, kukin neuroni välittää tiedon aktivoitumisestaan seuraavan tason neuroneille ja vaikuttaa omalta osaltaan niiden aktivoitumiseen.

Koska hermoverkko koostuu suuresta määrästä yksinkertaisia laskentayksiköitä, on hermoverkon parametrien määrä yleensä hyvin suuri. Jos hermoverkossa on esimerkiksi kolme piilokerrosta (kuten sivun 74 kuvassa), ja eri kerroksilla on solmuja 100, 100, 100 ja 30 kappaletta, saadaan parametrien kokonaismääräksi huikeat $101 \cdot 100 + 101 \cdot 100 + 101 \cdot 30 = 23230$ kerrointa. Tällaista kerroinmäärää varten tarvitaan luonnollisesti automaattinen laskentamenetelmä.

Yleisimmin käytetty painojen laskentamenetelmä on englanninkieliseltä nimeltään *back-propagation algorithm*, josta käytetään joskus suomenkielistä käännöstä *takaisinlevitysmenetelmä*. Menetelmän juuret ovat LMS-algoritmissa, ja tässäkin tapauksessa virhefunktion minimi haetaan korjaamalla painoja gradientin minimin suuntaan.

Hermoverkkojen opetus on erillinen prosessi, joka tapahtuu ennen kuin verkkoa aletaan käyttää tuotantokäytössä. Tällaisesta opetuksesta käytetään nimitystä *off-line -oppiminen* vastakohtana esimerkiksi LMS-algoritmin *on-line -oppimiseen*, missä painot muuttuvat suodatuksen mittaan. Yleensä off-line -oppimiseen käytetään runsaasti aikaa; tavallinen hermoverkon opetus voi viedä tunteja tai jopa vuorokausia prosessoriaikaa. Opetukseen kuluva aika ei kuitenkaan ole ongelma: järjestelmän muuhun kehitystyöhön verrattuna aika on häviävän pieni, ja opetus voidaan tehdä esimerkiksi yöllä tai viikonloppuna.

Aiemmin esillä olleen kaksiulotteisen datan luokittelu voidaan tehdä alla olevan kuvan mukaisella verkolla. Herätteenään verkko saa kaksi lukua, koordinaatit x_1 ja x_2, jotka syötetään keskellä olevalle piilokerrokselle. Näiden solmujen laskennan tulokset syötetään edelleen ulostulokerroksen kahdelle solmulle, jotka kertovat kumpaan luokkaan näyte kuuluu: tavoitteena on nyt opettaa verkko niin, että luokan 1 näytteen tapauksessa $y(1) > y(2)$ ja luokan 2 tapauksessa $y(2) > y(1)$.[6]



Opetusta varten tarvitaan *opetusjoukko*, jossa on valmiiksi luokiteltuja esimerkkejä. Edellisen esimerkin tapauksessa verkolle näytetään kaksiulotteisia näytteitä ja sen ulostuloa verrataan haluttuihin ulostuloarvoihin. Esimerkiksi vihreän tähden tapauksessa halutut ulostulot voisivat olla

$$\begin{aligned} d(1) &= 0.9 \\ d(2) &= 0.1 \end{aligned}$$

ja punaisen ristin tapauksessa

$$\begin{aligned} d(1) &= 0.1 \\ d(2) &= 0.9. \end{aligned}$$

Halutuksi ulostuloksi ei kannata valita vektoria $(1, 0)$, koska sigmoidi saavuttaa ääriarvot 0 ja 1 vasta kun summa on $\pm\infty$. Käytännössä tämä ilmenisi siten, ettei verkko

[6]Toinen vaihtoehto verkon rakenteeksi olisi käyttää vain yhtä ulostulosolmua, jonka arvo olisi suuri luokan 2 näytteille ja pieni luokan 1 näytteille. Esimerkin kahden ulostulossolmun rakenne toimii kuitenkin myös kun luokkia on enemmän kuin kaksi.

oppisi luokittelua kunnolla. Halutun ulostulon $d(n)$ määräämisen jälkeen voidaan laskea verkon virhe jokaiselle opetusjoukon näytettä vastaavalle ulostulolle $y(n)$ kaavalla $e(n) = d(n) - y(n)$, kaikille $n = 1, 2, \ldots, K$. Itse virhefunktio on tavallisesti puolet virhevektorin neliösummasta: $\mathcal{E} = \frac{1}{2}\sum_{k=1}^{K} e^2(n)$.

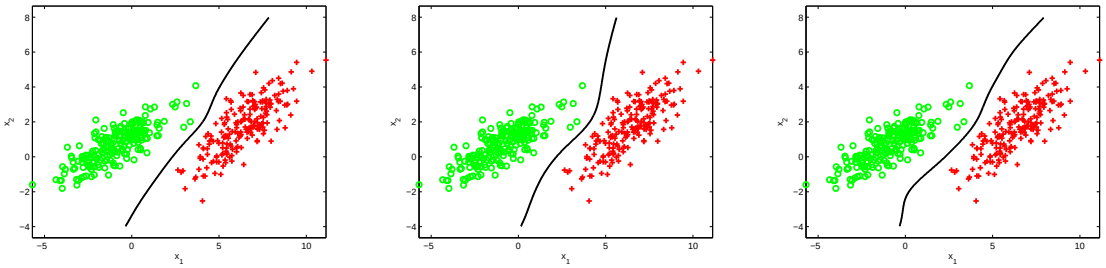
Virhefunktio $\mathcal{E}$ riippuu jokaisesta verkon painosta, ja on mahdollista johtaa kaavat funktion osittaisderivaatoille kunkin painon suhteen:

$$\frac{\partial \mathcal{E}}{\partial w_{jk}},$$

missä w_{jk} tarkoittaa kerroksen j painoa k. [7].

Kun virhe on laskettu, voidaan verkon kaikkia painoja päivittää negatiivisen gradientin suuntaan (eli sinne mihin virhe pienenee). Päivitys tehdään oikealta vasemmalle, eli lähempänä ulostuloa olevat painot päivitetään ensin taso kerrallaan. Myöhäisempien tasojen painokertoimet vaikuttavat nimittäin aikaisempien tasojen osittaisderivaattoihin, joten päivityksen täytyy alkaa ulostulon suunnasta.

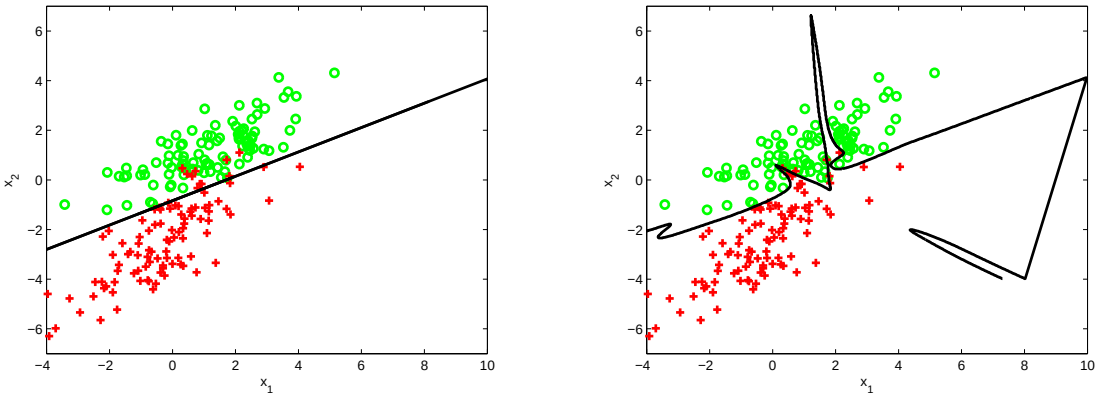
Painojen päivitys voidaan tehdä jokaisen esimerkin esittämisen jälkeen tai vasta kun kaikki opetusjoukossa olevat näytteet on ajettu verkon läpi. Hermoverkon opetuksen tulos riippuu näytteiden esitysjärjestyksestä, joka on useimmissa toteutuksissa satunnainen. Näin ollen kahta samanlaista hermoverkkoa ei käytännössä voi saada peräkkäisillä opetuskerroilla. Tämä tulee hyvin ilmi alla olevista kolmella peräkkäisellä opetuskerralla saatavista päätöspinnoista aiemmin esillä olleessa kaksiulotteisessa tapauksessa. Esimerkin tapauksessa piilokerroksella on 10 solmua.



Edellä olleet menetelmät muodostivat varsin erilaisia päätöspintoja. Yksinkertaisimmillaan päätöspinta on suora ja monimutkaisimmillaan se voi mutkitella hyvinkin paljon. Mitä monimutkaisempi luokittelija on, sitä monimutkaisempia alueita se kykenee erottelemaan toisistaan. Tällöin on kuitenkin pidettävä mielessä ylioppimisen vaara.

Alla oleva esimerkki havainnollistaa ylioppimista. Tässä tapauksessa luokat ovat lähempänä toisiaan ja menevät osin päällekkäin. Vasemmalla on LDA-luokittelijan päätöspinta ja oikealla on hermoverkon päätöspinta. Hermoverkko on tässä tapauksessa valittu tarkoituksella melko monimutkaiseksi, eli siinä on kaksi kahdenkymmenen solmun piilokerrosta peräkkäin. Kuvista nähdään, että hermoverkko oppii luokittelemaan opetusjoukon täydellisesti: monimutkainen päätöspinta jakaa kaikki vihreät ja punaiset näytteet omille puolilleen.

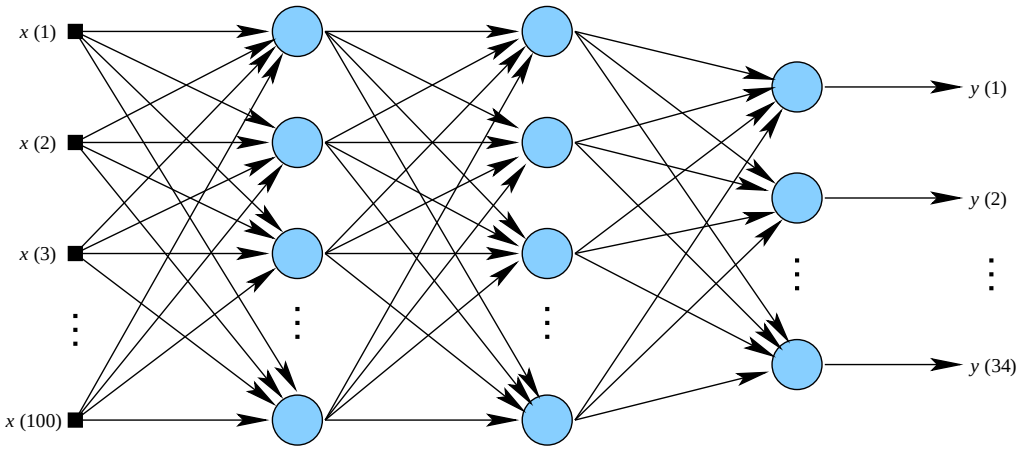
[7] Kaavat löytyvät esimerkiksi kirjasta: Haykin, Simon: *Neural networks: a comprehensive foundation*, Prentice Hall, 1999.



Suora päätöspinta toimii kuitenkin uudelle datalle tässä tapauksessa paremmin kuin hermoverkon monimutkaisempi käyrä. Päätöspinta tekee koukkaukset muutaman sattumalta mukaan tulleen poikkeuksellisen näytteen vuoksi, joita tuskin kannattaisi ottaa huomioon; seuraavassa näytteessä näitä näytteitä tuskin on ainakaan samoilla kohdilla.

5.5 Esimerkki: merkkien luokittelu

Tarkastellaan hermoverkon opetusta esimerkkitapauksessa, jossa hermoverkko tunnistaa bittikarttamuodossa olevia kirjaimia ja numeroita. Verkon rakenne tässä tapauksessa voi olla esimerkiksi alla olevan kuvan mukainen.



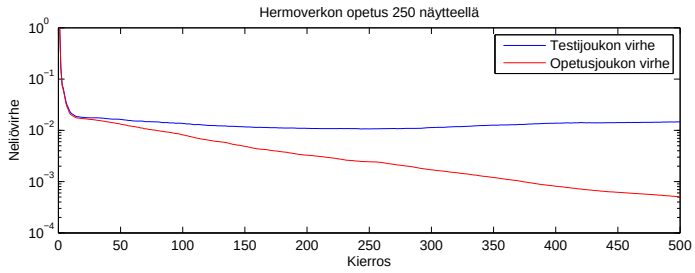
Kuvassa verkon herätteiden lukumääräksi on valittu 100, mikä vaatii kaikkien luokiteltavien kirjainten skaalaamisen 10×10-kokoon. Mikä tahansa muukin määrä on mahdollinen, mutta hyvin suuri herätteiden määrä tuottaa jo laskennallisiakin ongelmia. Verkolla on 34 ulostuloa, mikä vastaa englannin kielen kirjainten ja numeroiden lukumäärää ('1' ja 'I' sekä '0' ja 'O' tulkitaan samoiksi luokiksi).

Ulostulon lukuarvot voidaan tulkita varmuudeksi siitä kuuluuko kirjain luokkaan 1, 2, 3, . . . , 34. Jos esimerkiksi $y(1)$ on suuri ja muut ulostulot pieniä, tarkoittaa tämä merkin olevan '0'. Ulostulo $y(2)$ vastaa merkkiä '1', $y(11)$ vastaa merkkiä 'A', $y(12)$ merkkiä 'B' ja $y(34)'$ merkkiä 'Z'.

Opetusjoukon lisäksi on hyvä olla pienempi testijoukko, jonka avulla hermoverkon tehokkuutta arvioidaan. Testijoukko koostuu samanlaisista näytteistä kuin opetusjoukkokin, ja käytännössä se saadaan jakamalla opetusjoukko satunnaisesti kahtia esimerkiksi

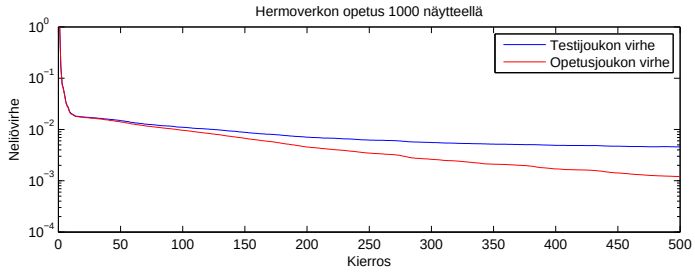
suhteessa 1:9. Opetuksen edetessä testijoukkokin luokitellaan säännöllisin väliajoin senhetkisellä hermoverkolla. Luokittelutuloksen virhe lasketaan ja näin saadaan arvio siitä, kuinka hyvin verkko osaa luokitella sellaiset näytteet, joita ei ole käytetty opetuksessa. Tällaisessa tilanteessahan verkkoa tullaan todellisuudessa käyttämään.

Testijoukkoa käyttämällä voidaan välttää hermoverkon *ylioppiminen*, joka ilmenee erityisesti pienillä opetusjoukoilla. Tällöin nimittäin hermoverkko alkaa oppia opetusjoukon epäolennnaisia piirteitä, joita ei ole joukon ulkopuolisilla näytteillä. Alla olevassa kuvassa on tyypillinen virhekäyrä ylioppimistapauksessa. Tässä esimerkissä hermoverkko opetettiin luokittelemaan 10×10 pikselin kokoisia bittikarttoja kirjaimiksi. Opetusjoukossa oli 250 ja testijoukossa 1000 etukäteen käsin luokiteltua kirjainta.



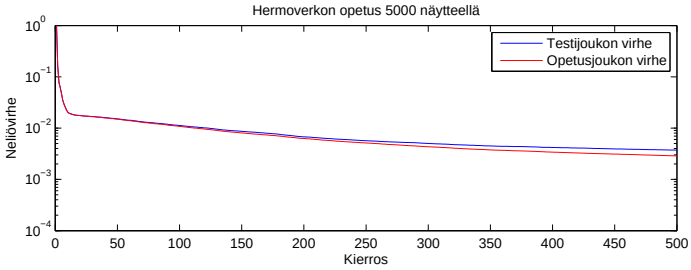
Kuvassa alempi käyrä esittää opetusjoukon näytteiden luokitteluvirhettä. Koska hermoverkon kertoimia muutetaan aina negatiivisen gradientin suuntaan, pienenee opetusjoukon virhe kierros kierrokselta. Ylempi käyrä esittää testijoukon näytteiden luokitteluvirhettä. Koska näitä näytteitä ei käytetä opetuksessa, luokitteluvirhe ei laskekaan monotonisesti, vaan voi jopa kasvaa ajoittain. Tällöin opetusjoukko on liian pieni tai huonosti valittu, koska opetusjoukon ja testijoukon virheet käyttäytyvät eri tavalla. Tällaisessa tapauksessa verkko oppii opetusjoukon liian hyvin, ja alkaa käyttää hyväkseen piirteitä, joita ei välttämättä ole opetusjoukon ulkopuolella. Verkko ei toimi testijoukossa läheskään yhtä hyvin kuin opetusjoukossa, ja virhe jää melko suureksi vaikka opetusta jatketaan. Usein testijoukon virhe alkaa vieläpä uudelleen kasvaa, kun ylioppiminen etenee riittävän pahaksi. Ylioppimiselta voidaan välttyä käyttämällä suurempaa ja edustavampaa opetusjoukkoa. Jos tämä ei ole mahdollista, paras vaihtoehto on keskeyttää opetus ennen kuin verkko on liian ylioppinut.

Seuraavassa kuvassa on sama testi tapauksessa, jossa opetusjoukon ja koko on kasvatettu moninkertaisiksi: opetusjoukossa on nyt 1000 näytettä ja testijoukossa samat 1000 näytettä kuin aikaisemminkin. Tässä tapauksessa ylioppiminen on vähäisempää, mutta joukkojen käyrien välillä on edelleen selvä ero.

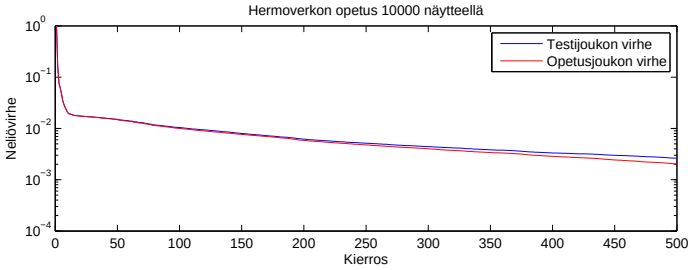


Kasvattamalla opetusjoukkoa edelleen ero opetusjoukon ja testijoukon välillä pienenee. Alla olevassa kuvassa on käytetty 5000 näytteen opetusjoukkoa ja samaa testijoukkoa

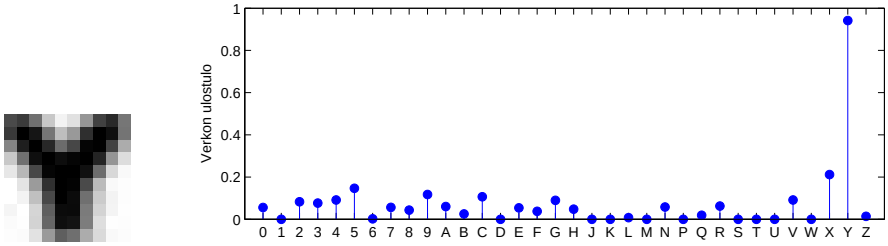
kuin aikaisemminkin. Ero käyrien välillä on aiempaa pienempi ja käyrät ovat hyvin lähellä toisiaan. Opetusjoukon virhe jää myös suuremmaksi kuin edellisessä tapauksessa, koska verkko ei opi luokittelua helpottavia vain opetusjoukossa olevia epäolennaisia piirteitä.



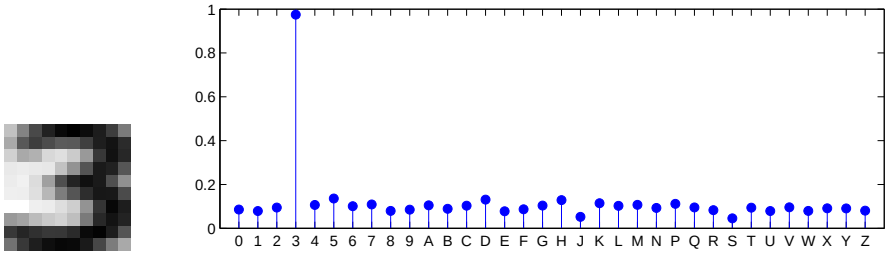
Neljännessä testissä opetusjoukon kokoa kasvatettiin 10000:n näytteeseen. Tulos on alla ja se on melko samanlainen kuin puolta pienemmällä opetusjoukolla. Testijoukon virhe kuitenkin pienenee edelleen.



Esimerkki opetetun hermoverkon ulostulovektorista on alla olevassa kuvassa oikealla Tässä tapauksessa sisään meni vasemmalla oleva bittikartta, joka tunnistettiin oikein kolmanteen (merkkiä 'Y' vastaavaan) luokkaan.



Toisena esimerkkinä on numero '3', joka on alla olevassa kuvassa vasemmalla. Oikealla on ulostulovektori, josta näkyy selvästi verkon oppineen luokittelun erinomaisesti myös näillä opetusjoukon ulkopuolisilla näytteillä.



Harjoitustehtäviä

5.1. *(Matlab)* Toteutetaan tässä tehtävässä lineaarinen erottelija kaksiulotteiselle keinotekoiselle datalle. Tehtävässä 5.5 sovelletaan menetelmää kuvankäsittelyyn.

Lataa kurssin sivuilta tiedosto

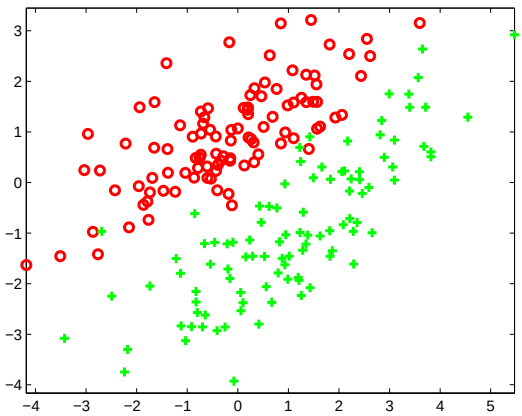
```
http://www.cs.tut.fi/kurssit/SGN-1251/testdata_fisher.mat
```

Tiedosto sisältää kaksi kaksiulotteista dataa sisältävää matriisia, joista kumpikin esittää yhtä luokkaa. Lataa tiedosto komennolla

```
load testdata_fisher.mat
```

jonka jälkeen Matlabissa on kaksi uutta muuttujaa: X1 $\in \mathbf{R}^{100\times 2}$ sisältää 100 kaksiulotteista datapistettä luokan 1 alkioita ja X2 $\in \mathbf{R}^{100\times 2}$ saman määrän luokan 2 alkioita. Molempien ensimmäinen sarake sisältää alkioiden x-koordinaatit ja toinen sarake y-koordinaatit.

Tulosta samaan kuvaan matriisin X1 pisteet punaisilla palloilla ja matriisin X2 pisteet vihreillä risteillä olevan kuvan mukaisesti.



5.2. *(Matlab)* Edellisen tehtävän kaksi luokkaa on tarkoitus nyt luokitella Fisherin diskriminantin avulla. Kyseinen menetelmä projisoi kaksiulotteisen datan sille suoralle, jossa joukot erottuvat toisistaan mahdollisimman hyvin. Tämän jälkeen ne on helppo erotella jakamalla vain suora kahteen osaan sopivasti.

Paras mahdollinen suora määritellään vektorina seuraavasti:

$$\mathbf{w} = (\mathbf{C}_1 + \mathbf{C}_2)^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

missä $\mathbf{C}_1$ ja $\mathbf{C}_2$ ovat luokkien 2×2-kokoiset kovarianssimatriisit ja μ_1 ja μ_2 ovat luokkien keskipisteet 2×1-vektoreina. Laske vektori $\mathbf{w}$. Kovarianssi saadaan funktiolla `cov` ja keskipiste funktiolla `mean`. Piirrä vektori ruudulle komennolla `line`. Komento `hold on` ennen piirtokomentoa pitää edellisen tehtävän kuvan mukana vertailun vuoksi. Anna myös komento `axis equal`, jotta koordinaatistosta tulisi neliö ja vertailusta helpompaa.

5.3. *(Matlab)* Jatkoa edelliseen tehtävään. Projektiosuoran suunnan lisäksi tarvitaan kohta, johon luokkien välinen raja vedetään. Yksinkertaisin tapa on sijoittaa se luokkien massakeskipisteiden puoliväliin. Käytännössä tämä tapahtuu projisoimalla data suoralle ja vertaamalla tulosta kynnysarvoon $c \in \mathbf{R}$:

$$\begin{cases} \text{Näyte } \mathbf{x} \text{ kuuluu luokkaan 1, jos } \mathbf{w} \cdot \mathbf{x} < c \\ \text{Näyte } \mathbf{x} \text{ kuuluu luokkaan 2, jos } \mathbf{w} \cdot \mathbf{x} \geq c \end{cases}$$

Laske kynnysarvo c seuraavasti:

- Laske mille arvolle μ_1 projisoituu.
- Laske mille arvolle μ_2 projisoituu.
- Luku c on näiden keskiarvo.

5.4. *(Matlab)* Laske montako prosenttia näytteistä saatiin luokiteltua oikein tehtävän 5.2 suuntavektorilla $\mathbf{w}$ ja tehtävän 5.3 kynnyksellä.[8]

5.5. *(Matlab)* Sovelletaan Fisherin diskriminanttia todellisen datan käsittelyyn. Lataa kurssin sivulta tiedosto

```
http://www.cs.tut.fi/kurssit/SGN-1251/canoe.jpg
```

Lataa se Matlabiin ja katso kuva:

```
imOrig = imread('canoe.jpg');
imshow (imOrig, []);
```

Kuvan kanootti on hyvin selkeästi erottuva punaisen värinsä vuoksi. Projisoidaan se Fisherin diskriminantilla sopivasti, jotta punainen erottuu muista väreistä mahdollisimman hyvin. Tätä varten tarvitaan opetusjoukko, joka saadaan kuvasta osoittamalla. Komento `[x,y] = ginput(1);` palauttaa kuvasta hiirellä osoitetun pisteen koordinaatit. Tee tämä kaksi kertaa niin että saat kaksi opetusjoukkoa: toisen kanootin alueelta ja toisen esimerkiksi veden pinnasta. Olkoot näin saadut koordinaatit vektoreissa `x1`, `y1` ja `x2`, `y2`. Irrota näin saatujen pisteiden ympäriltä 7×7-neliö, joiden pisteet muodostavat kaksi opetusluokkaa:

```
window1 = imOrig(y1-3:y1+3, x1-3:x1+3,:);
window2 = imOrig(y2-3:y2+3, x2-3:x2+3,:);
```

Huomaa, että data on nyt kolmiuloitteista: väri-informaatio esitetään RGB-arvoina, jonka dimensio on 3. Jotta tämä saataisiin Fisherin diskriminantin vaatimaan 3×49-matriisimuotoon, anna seuraavat komennot:

[8] Huomaa, että tositilanteessa virhe täytyisi arvioida muulla kuin opetusdatalla. Nythän esim. 1-NN-luokittelija saa täydellisen tuloksen testattaessa samalla datalla kuin opetuksessa, mikä ei silti tarkoita että se olisi hyvä millekään muulle näytejoukolle.

```
X1 = double(reshape(window1, 49, 3))';
X2 = double(reshape(window2, 49, 3))';
```

5.6. *(Matlab)* Käytä tehtävän 8.2 ohjelmakoodia saadaksesi kolmiulotteisen vektorin, joka kuvaa parasta projektiota annetun kahden värin optimaaliseksi erotteluksi. Mikä kyseinen vektori on?

5.7. *(Matlab)* Itse projektio tehdään kertomalla vektorin **w** komponenteilla kuvan R, G ja B -komponentit, esim. näin:

```
imGray = w(1)*double(imOrig(:,:,1)) +...
         w(2)*double(imOrig(:,:,2)) +...
         w(3)*double(imOrig(:,:,3));
```

Katso tuloskuva. Osoittamasi punaisen alueen pitäisi hohtaa kirkkaan valkoisena ja jälkimmäisenä osoitetun alueen miltei mustana.

Yritä erottaa punainen alue muista kynnystämällä tiettyä arvoa suuremmat arvot valkoisiksi ja muut mustiksi. Matlabissa on komento

```
imshow(imGray > kynnys, []);
```

Komennossa oleva parametri `kynnys` on raja, jota vaaleammat pisteet tulevat valkoisiksi ja muut mustiksi. Saatko kanootin irrotettua taustastaan?

5.8. Suunniteltaessa lineaarista luokittelijaa kaksiulotteiselle datalle saadaan opetusdatasta kahden luokan kovarianssimatriiseiksi ja keskiarvoiksi seuraavat:

$$\begin{aligned} \mathbf{C}_1 &= \begin{pmatrix} 2 & 0.1 \\ 0.1 & 0.4 \end{pmatrix} \quad \mathbf{C}_2 = \begin{pmatrix} 3 & -2 \\ -2 & 2 \end{pmatrix} \\ \boldsymbol{\mu}_1 &= \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad \boldsymbol{\mu}_2 = \begin{pmatrix} -7 \\ 2 \end{pmatrix} \end{aligned}$$

Laske projektiosuoran määräävä vektori **w**. Älä käännä matriisia koneella, vaan käsin muistisäännöllä

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

5.9. Jatkoa edelliseen tehtävään. Projektiosuoran suunnan lisäksi tarvitaan kohta, johon luokkien välinen raja vedetään. Yksinkertaisin tapa on sijoittaa se luokkien massakeskipisteiden puoliväliin. Käytännössä tämä tapahtuu projisoimalla data suoralle ja vertaamalla tulosta kynnysarvoon $c \in \mathbf{R}$:

$$\begin{cases} \text{Näyte } \mathbf{x} \text{ kuuluu luokkaan A, jos } \mathbf{w} \cdot \mathbf{x} < c \\ \text{Näyte } \mathbf{x} \text{ kuuluu luokkaan B, jos } \mathbf{w} \cdot \mathbf{x} \geq c \end{cases}$$

Laske kynnysarvo c seuraavasti:

- Laske mille arvolle μ_1 projisoituu.
- Laske mille arvolle μ_2 projisoituu.
- Luku c on näiden keskiarvo.

5.10. *(Matlab)* Kokeillaan tukivektorikoneen toimintaa luokittelussa. Lataa kurssin sivuilta tiedosto

```
http://www.cs.tut.fi/kurssit/SGN-1251/testdata_svm.mat
```

Kuten tehtävässä 5.1, tiedosto sisältää kaksi kaksiulotteista dataa sisältävää matriisia, joista kumpikin esittää yhtä luokkaa. Lataa tiedosto komennolla

```
load testdata_svm.mat
```

jonka jälkeen Matlabissa on kolme uutta muuttujaa: $X1 \in \mathbf{R}^{2\times 200}$ sisältää 200 kaksiulotteista datapistettä luokan 1 alkioita ja $X2 \in \mathbf{R}^{2\times 200}$ saman määrän luokan 2 alkioita. Tulosta pisteet ruudulle eri väreillä kuten tehtävässä 5.1.

SVM-luokittelijan opetus tapahtuu Matlabin Bioinformatics-toolboxin komennolla `svmtrain`. Funktiota varten data täytyy muokata seuraavasti:

```
training = [X1, X2]';
classes = [ones(length(X1), 1); zeros(length(X2), 1)];
```

Tämän jälkeen SVM-luokittelija opetetaan komennolla

```
svmStruct = svmtrain(training, classes, 'showplot', true);
```

SVM-luokittelun tulos saadaan komennolla

```
result = svmclassify(svmStruct, training);
```

Nyt siis muuttujassa `classes` on todelliset luokat ja muuttujassa `result` SVM:n ennustamat luokat. Laske montako prosenttia opetusnäytteistä meni oikein.

5.11. *(Matlab)* Funktiolle `svmtrain` voi määritellä eri kerneleitä seuraavasti:

```
svmStruct = svmtrain(training, classes, ...
   'showplot', true, ...
   'kernel_function', 'linear');
```

Viimeinen termi on oletuksena `'linear'`, mutta muita vaihtoehtoja ovat mm.

- `'quadratic'`: Neliöity kerneli
- `'polynomial'`: Polynomikerneli
- `'rbf'`: RBF-kerneli

Kokeile näitä kolmea kerneliä edellisen tehtävän tapaukseen ja laske montako prosenttia opetusnäytteistä menee oikein kussakin tapauksessa.

Toisinaan SVM-opetus ei konvergoi riittävän nopeasti ja `svmtrain` antaa ilmoituksen

```
Unable to solve the optimization problem...
```

Tämä korjautuu yleensä lisäämällä iteraatioiden määrää, mikä voidaan tehdä muuttujalla `'quadprog_opts'`. Tällöin loitsu on koko komeudessaan esim. muotoa

```
svmStruct = svmtrain(training, classes, ...
   'showplot', true, ...
   'kernel_function', 'linear', ...
   'quadprog_opts', optimset('MaxIter', 1000));
```

Luku 6

Tilastolliset menetelmät signaalinkäsittelyssä

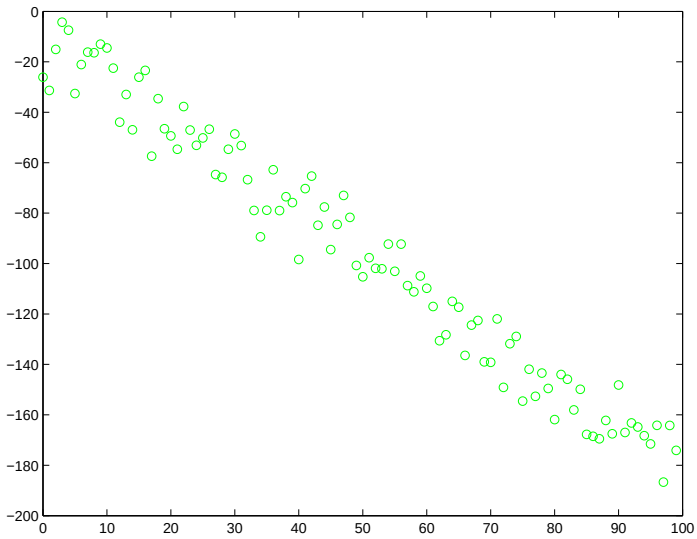
Tilastollisia menetelmiä sovelletaan laajasti signaalinkäsittelyssä. Eräässä mielessä myös kappaleen 5 hahmontunnistusmenetelmät voidaan lukea tähän luokkaan. Laajennetaan kuitenkin tätä tarkastelua tässä kappaleessa yleisempiin estimointiongelmiin.

Tilastollinen signaalikäsittely tarkastelee ongelmia, joissa ongelma voidaan kuvata tilastollisen *mallin* avulla. Tämä voi tarkoittaa esimerkiksi suureiden x ja y välisen suhteen mallintamista mahdollisimman tarkasti funktiolla f:

$$y = f(x) + \epsilon.$$

Mallin f lausekkeessa on tavallisesti joitakin parametreja, joiden arvo täytyy sovittaa mahdollisimman oikeiksi todellisten mittausten valossa. Termi ϵ on mallinnusvirhe, joka on virhe mallin ja todellisuuden välille. Käytännössähän malli $f(x)$ ei ole tarkalleen sama kuin suure y.

Estimointiongelma voi olla esimerkiksi alla olevan kuvan mukainen. Kuvan perusteella vaikuttaisi että pystyakselin suure $y(n)$ riippuu lineaarisesti vaaka-akselin suureesta $x(n)$.

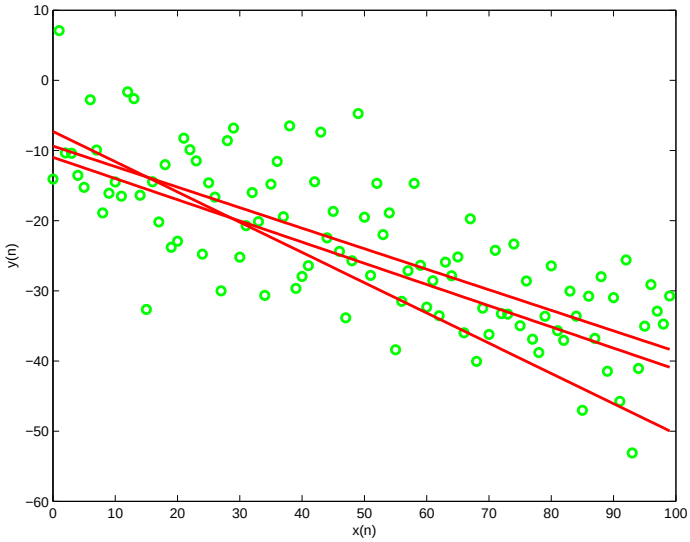


Tämä riippuvuus voidaan kuvata seuraavan mallin avulla:

$$y(n) = ax(n) + b + \epsilon, \tag{6.1}$$

josta täytyisi nyt päätellä mahdollisimman hyvät arvot parametreille a ja b annetun datan perusteella.

Erilaisilla parametrien a ja b arvoilla saadaan erilaisia kuvaajia, kuten alla olevasta kuvasta nähdään. Nyt estimointiongelma kiteytyy kysymykseen mikä niistä kuvaa suureiden $x(n)$ ja $y(n)$ riippuvuutta parhaiten (eli milloin ϵ menee mahdollisimman lähelle nollaa).



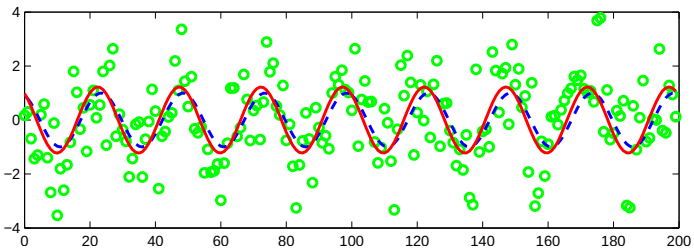
Voidaan osoittaa että pienin neliövirhe (eli ns. pienimmän neliörumman ratkaisu; engl. Least Squares solution) mallin ja todellisen datan välillä saadaan tässä tapauksessa kaavoilla

$$\hat{a} = -\frac{6}{N(N+1)}\sum_{n=0}^{N-1} y(n) + \frac{12}{N(N^2-1)}\sum_{n=0}^{N-1} x(n)y(n) \tag{6.2}$$

$$\hat{b} = \frac{2(2N-1)}{N(N+1)}\sum_{n=0}^{N-1} y(n) - \frac{6}{N(N+1)}\sum_{n=0}^{N-1} x(n)y(n). \tag{6.3}$$

Merkintä $\hat{a}$ tarkoittaa parametrin a estimaattia erotukseksi sen "oikeasta arvosta" a. Kaava ei ole erityisen helppolukuinen, ja myöhemmin onkin helpottavaa huomata kaavan olevan paljon selkeämpi matriisimuodossa[1].

Toinen esimerkki on tavallinen mm. tiedonsiirrossa. Tässä tapauksessa kuvassa oleva katkoviivalla piirretty sininen käyrä esittää lähetettyä sinisignaalia, josta kuitenkin vastaanotetaan vain kuvassa vihreillä ympyröillä piirretyt häiriölliset näytteet.



Vastaanottopäässä on nyt vastattava vastaaotettujen näytteiden perusteella mahdollisimman hyvin seuraaviin kysymyksiin:

[1]Sopivilla määrittelyillä kaava saadaan muotoon: $\begin{pmatrix}\hat{a}\\ \hat{b}\end{pmatrix} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$.

- Mikä oli lähetetyn signaalin amplitudi?
- Mikä oli lähetetyn signaalin taajuus?
- Mikä oli lähetetyn signaalin vaihe?

Täsmällisemmin ilmaistuna tämä tarkoittaa että on etsittävä hyvät arvot parametreille A, f ja ϕ seuraavassa mallissa

$$x(n) = A\cos(2\pi f n + \phi) + \epsilon,$$

Voidaan osoittaa että ns. *suurimman uskottavuuden ratkaisu* (engl. maximum likelihood estimator; MLE) saadaan kaavoista

$$\begin{aligned}
\hat{f} &= \text{se parametrin } f \text{ arvo joka maksimoi lausekkeen } \left|\sum_{n=0}^{N-1} x(n)e^{-2\pi i f n}\right|, \\
\hat{A} &= \frac{2}{N}\left|\sum_{n=0}^{N-1} x(n)e^{-2\pi i \hat{f} n}\right| \\
\hat{\phi} &= \arctan\frac{-\sum_{n=0}^{N-1} x(n)\sin(2\pi\hat{f}n)}{\sum_{n=0}^{N-1} x(n)\cos(2\pi\hat{f}n)}.
\end{aligned}$$

Käytännössä täytyy siis ensin löytää sinin taajuus ensimmäisellä kaavalla[2], jonka jälkeen voidaan estimoida sen amplitudi ja vaihe kahdella jälkimmäisellä kaavalla. Yllä olleessa testitapauksessa alkuperäiset arvot olivat $f = 0.04$, $A = 1$ ja $\phi = 0.4$. Häiriöllisistä vihreistä näytteistä parametrien arvoiksi arvioitiin $\hat{f} = 0.0401$, $\hat{A} = 1.2249$ ja $\hat{\phi} = 0.6425$. Näin saatuja arvoja vastaava sinisignaali on esitetty yllä olevassa kuvassa punaisena yhtenäisenä käyränä.

Tärkeä osa tilastollista signaalinkäsittelyä on ratkaisujen optimaalisuuden tarkastelu. Molempien edellä olleiden esimerkkien tapauksessa löydettiin optimaalinen ratkaisu suljetussa kaavamuodossa. Näin ollen ilmaan ei jää kysymystä olisiko mahdollisesti olemassa vieläkin paremmat arvot. Esimerkkien optimaalisuuskriteerit olivat kuitenkin erilaiset: ensimmäinen esimerkki on optimaalinen neliövirheen mielessä ja jälkimmäinen oli suurimman uskottavuuden ratkaisu. Seuraavaksi tutustutaan näistä kahdesta yksinkertaisempaan menetelmään lähemmin.

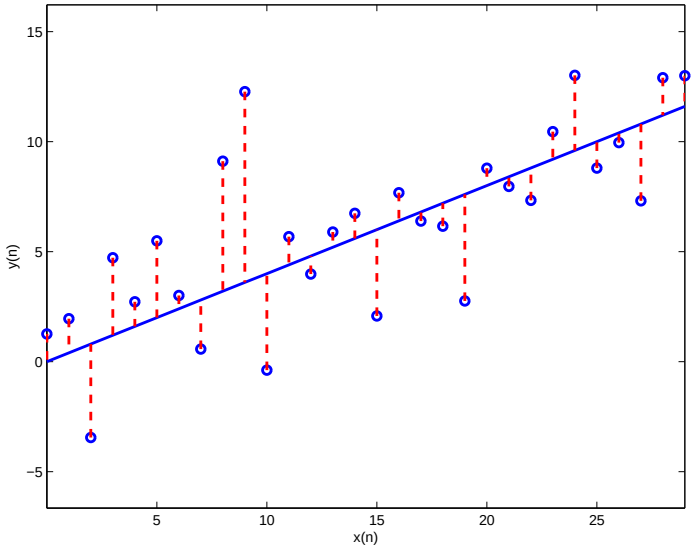
6.1 Pienimmän neliösumman menetelmä

Pienimmän neliösumman menetelmä (PNS) on peräisin vuodelta 1795, jolloin kaikkien aikojen suurimmaksi matemaatikoksi sanottu C.F. Gauss kehitti sen vain 18-vuotiaana. Ensimmäinen sovellus oli kääpiöplaneetta Ceresin radan lausekeen selvittäminen yksittäisistä mittauksista vuonna 1801.

Nimensä mukaisesti pienimmän neliösumman menetelmä pyrkii minimoimaan neliöetäisyyden mallin ja näytepisteiden välillä. Alla olevassa kuvassa olevat siniset ympyrät

[2]Huomaa, että tämä on itse asiassa diskreettiaikaisen Fourier-muunnoksen kaava. Maksimoiva taajuus voidaan päätellä esimerkiksi kokeilemalla kaikkia arvoja väliltä $0...2\pi$ vaikkapa tuhannesosan välein.

kuvaavat yksittäisiä mittauksia, ja sininen käyrä kuvaa siihen sovitettavaa mallia (vrt. kaava 6.3). PNS-sovite on nyt se suora, jolla kuvaan piirrettyjen katkoviivojen yhteispituus on mahdollisimman pieni.



Käytännön syistä etäisyyden mittana täytyy käyttää neliövirhettä. Matemaattisesti ilmaistuna optimaalinen suora on se, jonka parametrit a ja b minimoivat lausekkeen:

$$\sum_{n=0}^{N-1} (y(n) - (a \cdot x(n) + b))^2$$

PNS-ratkaisu on helppo löytää silloin kun malli on lineaarinen. Jos näin ei ole, ratkaisun löytyminen on tapauskohtaista, ja voidaan joutua turvautumaan numeerisiin menetelmiin. Lineaarisessa tapauksessa malli voidaan esittää kätevästi matriisimuodossa. Esimerkiksi suoran malli $y(n) = ax(n) + b + \epsilon$ voidaan matriisimuodossa ilmaista näin:

$$\begin{pmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{pmatrix} = \begin{pmatrix} x(0) & 1 \\ x(1) & 1 \\ x(2) & 1 \\ \vdots & \vdots \\ x(N-1) & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} + \boldsymbol{\epsilon}.$$

Vastaavasti toisen asteen malli $y(n) = ax(n)^2 + bx(n) + c + \epsilon$ voidaan ilmaista kaavalla

$$\begin{pmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{pmatrix} = \begin{pmatrix} x(0)^2 & x(0) & 1 \\ x(1)^2 & x(1) & 1 \\ x(2)^2 & x(2) & 1 \\ \vdots & \vdots & \\ x(N-1)^2 & x(N-1) & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} + \boldsymbol{\epsilon}.$$

Yleisesti voidaan löytää ratkaisu kaikille muotoa $\mathbf{y} = \mathbf{H}\mathbf{c} + \boldsymbol{\epsilon}$ oleville malleille. Todistus on samanlainen gradientin nollakohdan ratkaisu kuin optimiratkaisun etsintä kappaleessa 4, ja lopputuloskin on hyvin samanlainen: optimikertoimet saadaan kaavasta

$$\mathbf{c} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}.$$

Itse asiassa kaava on vain yleisempi versio LMS-algoritmin yhteydessä johdetusta kaavasta $\mathbf{w} = \mathbf{R}^{-1}\mathbf{p}$.

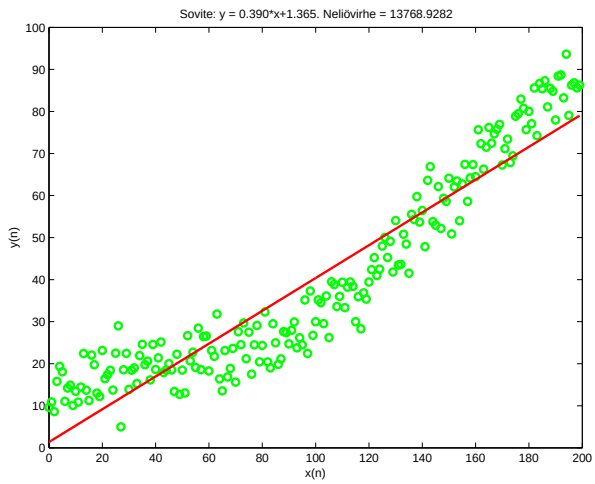
Lineaarisen mallin tapauksessa PNS-estimaatin ratkaisu on siis hyvin yksinkertaista. Matlabilla tarvitaan kaksi koodiriviä: jos mittausdata on pystyvektoreissa $\mathbf{x} \in \mathbf{R}^{N\times 1}$ ja $\mathbf{y} \in \mathbf{R}^{N\times 1}$, löytyvät mallin $\mathbf{y} = \mathbf{H}\mathbf{c} + \boldsymbol{\epsilon}$ kertoimet $\mathbf{c}$ kaavariveillä:

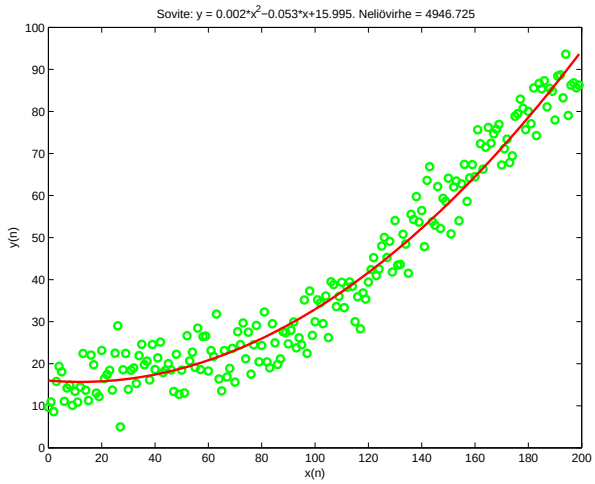
```
H = [x, ones(size(x))];
c = inv(H' * H) * H' * y;
```

Toisen asteen malli saataisiin aikaiseksi riveillä

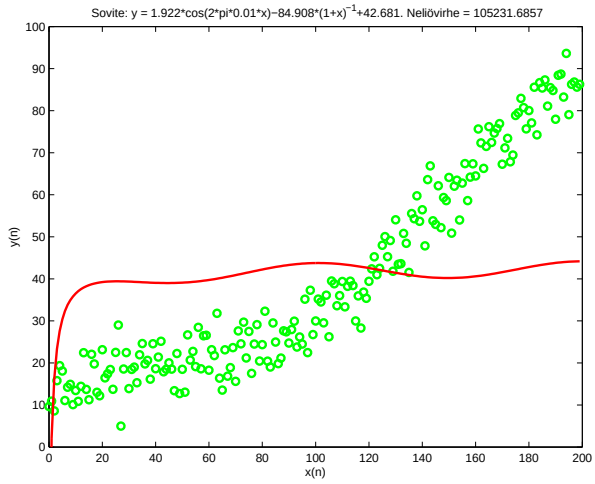
```
H = [x.^2, x, ones(size(x))];
c = inv(H' * H) * H' * y;
```

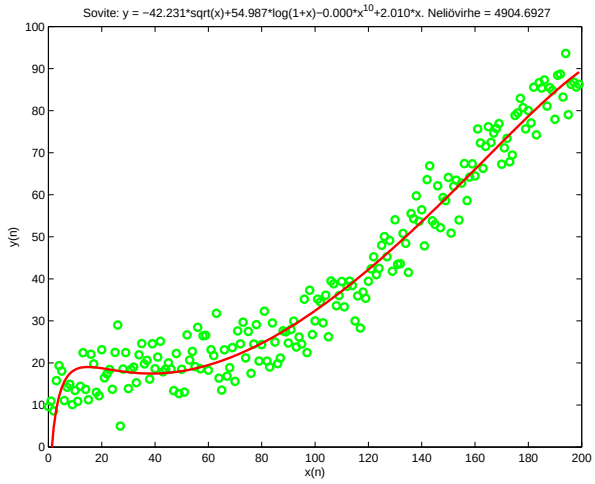
Erään testidatan tapauksessa ensimmäisen ja toisen asteen sovitteet ovat alla olevien kuvien mukaisia. Kuvaajat on piirretty komennolla `plot(x,y,'go',x,H*c,'r-');`.





Matriisiin $\mathbf{H}$ on helppo lisätä hyvinkin mielikuvituksellisia termejä, vaikka kaikki eivät välttämättä olekaan järkeviä. Alla yritetään mallintaa edellä ollutta dataa kosinin, käänteisfunktion, logaritmin, neliöjuuren sekä kymmenennen potenssin avulla. Jokaisessa kuvassa on myös sovitteen neliövirhe, jonka perusteella voi tehdä päätelmiä funktiojoukkojen soveltuvuudesta kyseisen datan mallintamiseen.



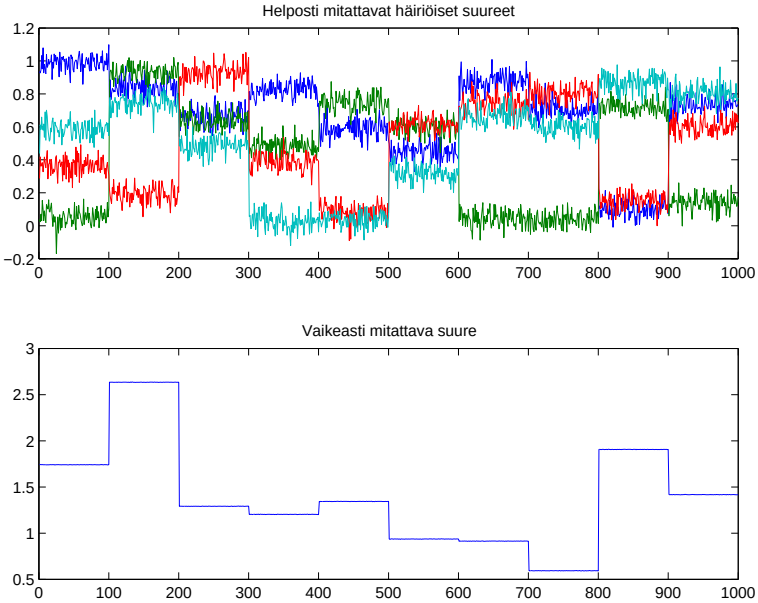


6.2 Sovellusesimerkkejä

Alussa olleiden esimerkkien lisäksi estimointia ja mallinnusta on sovellettu lukuisissa ongelmissa. Tarkastellaan seuraavassa lyhyesti muutamia.

Suureen approksimointi muiden mittausten avulla. Eräässä hankkeessa asiakkaalla oli laboratoriossa käytettävä menetelmä, jonka avulla saatiin eräs sovelluksen kannalta tärkeä suure mitattua laboratorio-olosuhteissa. Mittausmenetelmä oli hankala ja kallis, mutta sillä saatiin tarkka tulos. Hankkeen tavoitteena oli selvittää voitaisiinko eräistä helpommin suoritettavista mittauksista jotenkin päätellä summittainen arvo kiinnostavalle mittaukselle, jolloin mittauksia voitaisiin tehdä myös laboratorion ulkopuolella.

Esimerkki mittausdatasta on alla olevassa kuvassa kymmenessä eri tilanteessa. Ylemmän kuvan neljästä mittauksesta pitäisi päätellä approksimaatio alemman kuvan arvoille.

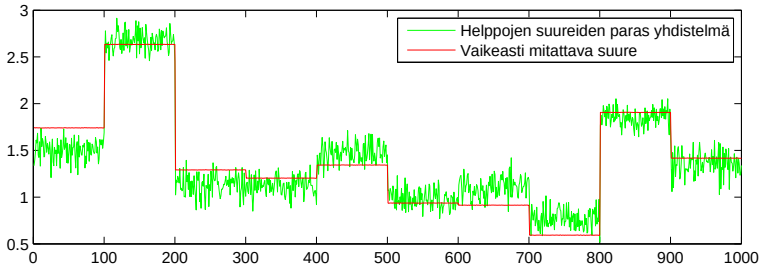


Hankkeessa toteutettiin yksinkertainen PNS-estimaattori alemman kuvan arvon approksimointiin. Tässä tapauksessa malli on

$$\underbrace{\begin{pmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} x_1(0) & x_2(0) & x_3(0) & x_4(0) & 1 \\ x_1(1) & x_2(1) & x_3(1) & x_4(1) & 1 \\ x_1(2) & x_2(2) & x_3(2) & x_4(2) & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1(N-1) & x_2(N-1) & x_3(N-1) & x_4(N-1) & 1 \end{pmatrix}}_{\mathbf{H}} \cdot \underbrace{\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix}}_{\mathbf{c}} + \boldsymbol{\epsilon},$$

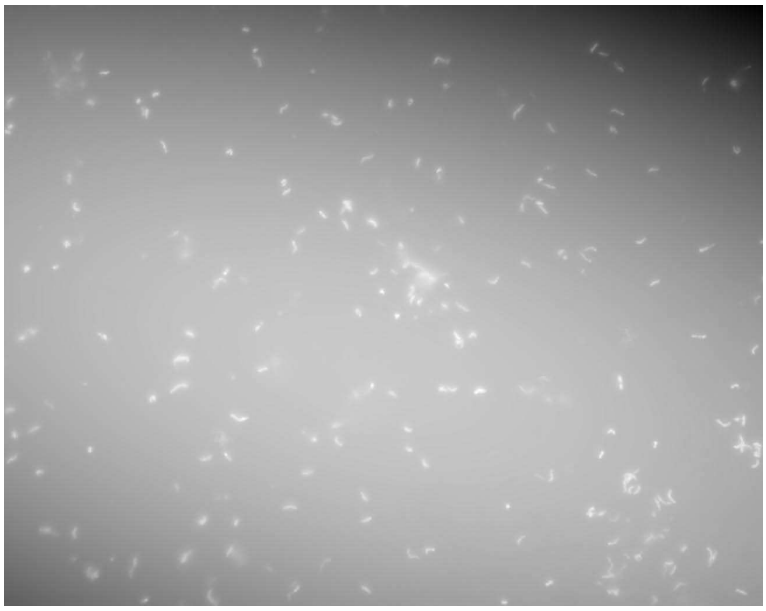
missä $x_k(j)$ tarkoittaa kanavan k mittausta ajanhetkellä j.

Yllä olevan datan tapauksessa PNS-kertoimet ovat $\mathbf{c} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y} = (0.8271, 0.6780, -1.0849, 1.1581, 0.4838)^T$ ja approksimaatio on alla olevan kuvan näköinen. Approksimaatio on kuvassa vihreällä ja tavoitearvo punaisella.



Kuvan perusteella näyttäisi että vaikeasti mitattavan suureen ennustus on mahdollista kohtuullisella tarkkuudella. On kuitenkin huomattava, että yllä oleva kuva on luultavasti liian optimistinen, koska samaa dataa käytettiin sekä opetukseen että testaukseen. Tämähän johtaa kappaleessa 5 tavattuun ylioppimiseen. Erilaiset ristiinvalidointimenetelmät (*engl.* cross validation) ovat yleinen tapa arvioida oppivan järjestelmän toimivuutta tuntemattomalle datalle. Edellisessä tapauksessa voitaisiin aina jättää yksi kuvan kymmenestä tasosta pois opetuksesta ja testata pois jätetyllä tasolla. Ennustustarkkuus kullekin opetuksesta vuorollaan pois jätetylle tasolle antaa paljon realistisemman kuvan ennustimen todellisesta tarkkuudesta kuin ennustustulos opetusdatalla.

Mikroskooppikuvan taustan kompensointi. TTY:n signaalinkäsittelyn laitoksen systeemibiologian tutkimusryhmässä kehitettiin automaattinen solulaskuri mikroskooppikuville[3]. Menetelmän yhtenä vaiheena on kuvan taustan kompensointi. Kuten alla olevasta esimerkkikuvasta nähdään, kuvan tausta ei ole tasaisesti valaistu.



Tämä ongelma voidaan saada hallintaan mallintamalla taustan kirkkauserot sopivalla funktiolla ja vähentämällä tulos alkuperäisestä kuvasta. Taustan lähempi tarkastelu osoittaa, että harmaasävyt saavat aina suurimman arvonsa lähellä kuvan keskustaa ja sävy muuttuu tummemmaksi reunoille mentäessä. Tämän vuoksi päätettiin käyttää paraboloidia mallintamaan valaistusta. Paraboloidin kaava on

$$z(x, y) = c_1x^2 + c_2y^2 + c_3xy + c_4x + c_5y + c_6.$$

[3]J. Selinummi, J. Seppälä, O. Yli-Harja, and J. Puhakka, "Software for quantification of labeled bacteria from digital microscope images by automated image analysis," *BioTechniques*, Vol. 39, No 6, 2005, pp 859-863.

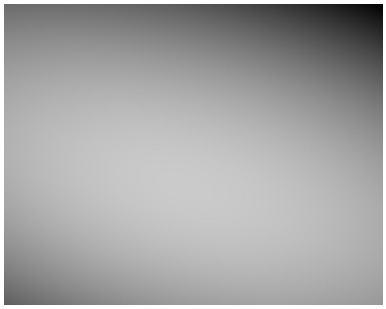
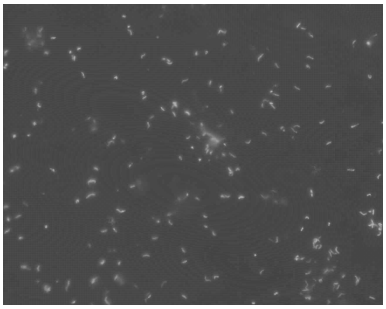
Näin ollen on etsittävä ne kertoimet $c_1, c_2, \ldots, c_6$, joilla neliösumma mallin ja todellisen kuvan välillä minimoituu. Nyt koko malli näyttää seuraavalta:

$$\underbrace{\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix}}_{\mathbf{z}} = \underbrace{\begin{pmatrix} x_1^2 & y_1^2 & x_1y_1 & x_1 & y_1 & 1 \\ x_2^2 & y_2^2 & x_2y_2 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_N^2 & y_N^2 & x_Ny_N & x_N & y_N & 1 \end{pmatrix}}_{\mathbf{H}} \cdot \underbrace{\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{pmatrix}}_{\mathbf{c}} + \boldsymbol{\epsilon}, \tag{6.4}$$

missä z_k tarkoittaa todellisen kuvan harmaasävyarvoa pisteessä (x_k, y_k). Nyt indeksi k käy kuvan kaikki pikselit läpi jossain järjestyksessä. PNS-sovituksen tuloksena saadaan kertoimet $\mathbf{c} = (-0.000080, -0.000288, 0.000123, 0.022064, 0.284020, 106.538687)$, eli sovitettu malli kokonaisuudessaan on

$$z(x, y) = -0.000080x^2 - 0.000288y^2 + 0.000123xy + 0.022064x + 0.284020y + 106.538687$$

Malli on esillä alla olevassa kuvassa keskellä, ja selvästikin se noudattaa alkuperäisen kuvan valaistuksen vaihtelua. Valaistus saadaan nyt kompensoitua yksinkertaisesti vähentämällä malli alkuperäisestä kuvasta.

-

=


Harjoitustehtäviä

6.1. Kappale on hetkellä $t(0) = 0$ s vapaassa pudotuksessa ja sen alkunopeus pystysuunnassa on v_0 ja paikka $s(0) = 0$. Paikasta tehdään kaksi muutakin mittausta hetkillä 2 s ja 4 s, jolloin saadaan seuraava taulukko.

Aika t	Paikka $s(t)$
0	0
2	22
4	83

Fysiikan kaavat antavat seuraavan yhteyden paikalle $s(t)$, alkunopeudelle v_0 ja vakiokiihtyvyydelle a:

$$s(t) = v_0 t + \frac{1}{2}at^2$$

Kaava voidaan muuntaa meille tutuksi malliksi seuraavasti:

$$\begin{pmatrix} s(0) \\ s(1) \\ s(2) \end{pmatrix} = \begin{pmatrix} t(0) & \frac{1}{2}t(0)^2 \\ t(1) & \frac{1}{2}t(1)^2 \\ t(2) & \frac{1}{2}t(2)^2 \end{pmatrix} \begin{pmatrix} v_0 \\ a \end{pmatrix} + \epsilon.$$

Laske käsin PNS-estimaatti $(\hat{v_0}, \hat{a})^T$ sivun 90 kaavalla. Kiihtyvyyden pitäisi olla lähellä gravitaatiovakiota g.

6.2. *(Matlab)* Lataa kappaleessa 6.1 käytetty testidata osoitteesta

`http://www.cs.tut.fi/kurssit/SGN-1251/LSData.mat`

Tiedostossa on kaksi muuttujaa, vektorit $\mathbf{x}$ ja $\mathbf{y}$. Laske PNS-malli muuttujien välille niin että

(a) malli on 1. astetta (s.o. mukana on vakio ja 1. asteen termi),

(b) malli on 2. astetta (s.o. mukana on vakio ja 1-2. asteen termit),

(c) malli on 3. astetta (s.o. mukana on vakio ja 1-3. asteen termit).

Laske kussakin tapauksessa mallinnusvirhe, eli $\sum_{n=0}^{N-1} (\mathbf{y} - \mathbf{Hc})^2$ ja tulosta pisteparvi sekä sovite sivun 91 komennoilla.

6.3. Mittaukset $x(n)$ ja $y(n)$ riippuvat toisistaan suunnilleen lineaarisesti, ja niistä on tehty seuraavat mittaukset:

n	0	1	2
$x(n)$	7	9	2
$y(n)$	11,6	14,8	3,5

Suureiden välinen riippuvuus halutaan selvittää mallintamalla suure $y(n)$ seuraavasti:

$$y(n) = ax(n) + b.$$

Laske käsin pienimmän neliösumman ratkaisu parametreille a ja b.

6.4. *(Matlab)* Lataa kappaleessa 6.2 käytetty testikuva osoitteesta

`http://www.cs.tut.fi/kurssit/SGN-1251/Microscope.jpg`

ja lataa se Matlabiin komennolla `imread`. Mallinnetaan kuvan taustaa paraboloidilla kaavan (6.4) mukaisesti.

Jotta voitaisiin muodostaa matriisi $\mathbf{H}$, täytyy ensin muodostaa vektorit $\mathbf{x}$ ja $\mathbf{y}$, joissa on kaikki kuvan x- ja y-koordinaattien kombinaatiot (eli yhtä monta kuin kuvassa

pikseleitä). Nämä saadaan Matlabissa kätevimmin muodostamalla seuraavanlaiset matriisit:

$$\mathbf{X} = \begin{pmatrix} 1 & 2 & 3 & 4 & \dots & 1300 \\ 1 & 2 & 3 & 4 & \dots & 1300 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & 3 & 4 & \dots & 1300 \end{pmatrix} \qquad \mathbf{Y} = \begin{pmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 2 & 2 & 2 & 2 & \dots & 2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1030 & 1030 & 1030 & 1030 & \dots & 1030 \end{pmatrix}$$

Matlabissa on valmis komento tähän: `[X, Y] = meshgrid(1:1300, 1:1030);`. Nyt sinulla on siis kolme matriisia: yhdessä on kuvan kaikki x-koordinaatit, toisessa kuvan y-koordinaatit ja kolmannessa sen z-koordinaatit (eli harmaasävyarvot). Kysymys kuuluu kuinka x:n ja y:n avulla voidaan ennustaa z.

Muodosta kaavan (6.4) mukainen matriisi $\mathbf{H}$. Tätä varten matriisien $\mathbf{X}$, $\mathbf{Y}$ sekä kuvan elementit täytyy järjestää vektoreiksi. Järjestyksellä ei ole väliä, kunhan se on sama kaikille kolmelle. Helpoimmalla pääsee komennoilla

```
x = X(:);
y = Y(:);
z = double(I(:));
```

jotka ottavat vektoriin matriisin kunkin pystyrivin yksi kerrallaan. Yllä oletettiin että kuva on muuttujassa `I`.

6.5. *(Matlab)* Jatkoa edelliseen tehtävään. Laske kertoimet $\mathbf{c}$ edellisen tehtävän mallille. Laske mallin ulostulo eli $\mathbf{Hc}$. Tulos on vektori, joten muokkaa se `reshape`-komennolla takaisin alkuperäisen kuvan muotoiseksi 1030×1300-matriisiksi ja vertaa tulosta alkuperäiseen kuvaan. Ota myös alkuperäisen kuvan ja taustan erotus ja tulosta se ruudulle.

6.6. *(Matlab)* PNS-menetelmä on herkkä yksittäisille suurille muutoksille, ja yksittäinen suuri mittavirhe voi saada sovitteen järjettömäksi. Testaa tehtävän 6.2 toisen asteen ratkaisun toimintaa kun muutat yhden opetusnäytteen järjettömän suureksi, esim. sijoituksella `y(1) = 1000;`. Aja tämän jälkeen PNS-sovitus ja tulosta vastaava kuva kuin tehtävässä 6.2. Yhden poikkeavan näytteen vuoksi skaala saattaa muuttua aika suureksi, joten muuta se samaksi kuin tehtävässä 6.2 komennolla `axis([0, 200, 0, 100]);`.

Matlabissa on komento `robustfit`, joka on epäherkempi versio PNS-sovituksesta. Tutustu sen käyttöön ja tee vastaava toisen asteen sovite kuin PNS-tapauksessakin. Nyt tuloksen pitäisi olla vastaava kuin tehtävässä 6.2 yhdestä poikkeuksellisen suuresta näytteestä huolimatta.

Robustista sovituksesta löytyy lisätietoa esimerkiksi hakutermillä *robust regression*.

6.7. *(Matlab)* PNS-menetelmällä voi estimoida parametreja myös moniulotteiselle datalle. Hae verkosta tiedosto

`http://www.cs.tut.fi/kurssit/SGN-1251/vectorField.mat`

ja lataa se Matlabiin. Tiedostossa on neljä muuttujaa: `X`, `Y`, `DX` ja `DY`. Nämä esittävät vektorimuotoista dataa, niin että `(DX,DY)` on pisteessä `(X,Y)` olevan vektorin pituus. Kuva näistä saadaan Matlabin komennolla `quiver(X, Y, DX, DY)`.

Vektoridatassa on kohinaa, ja halutaankin muodostaa matala-asteinen polynomimalli, joka sisältää olennaisimman tiedon vektoreiden suunnista ja suuruuksista. Mallin avulla ymmärretään vektoreiden suuntien muodostuminen sekä saadaan sivutuotteena kohinaton vektorikenttä.

Käytetään toisen asteen polynomimallia sekä DX:lle että DY:lle:

$$\begin{aligned} DX(m,n) &= a_1X(m,n)^2 + b_1Y(m,n)^2 + c_1X(m,n)Y(m,n) + d_1X(m,n) + e_1Y(m,n) + f_1 \\ DY(m,n) &= a_2X(m,n)^2 + b_2Y(m,n)^2 + c_2X(m,n)Y(m,n) + d_2X(m,n) + e_2Y(m,n) + f_2 \end{aligned}$$

Nämä molemmat saadaan siististi esitettyä yhtenä yhtälönä:

$$[DX, DY] = [X^2, Y^2, XY, X, Y, 1] \cdot C.$$

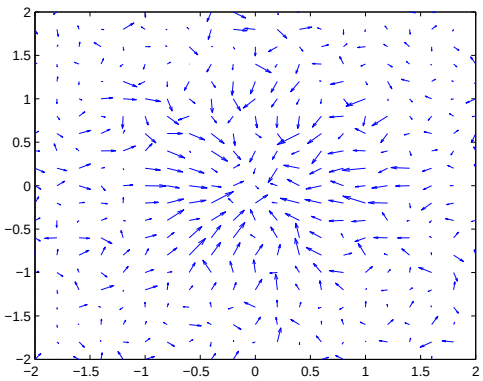
Matlabin syntaksilla matriisi **H** sekä toivottu ulostulo **Z** saadaan siis komennolla

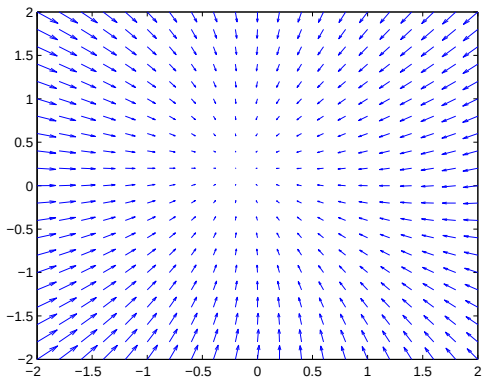
```
H = [X(:).^2, Y(:).^2, X(:).*Y(:), X(:), Y(:), ones(size(X(:)))];
Z = [DX(:), DY(:)];
```

Tästä voidaan ratkaista mallin matriisin $\mathbf{C} \in \mathbf{R}^{6\times 2}$ kertoimet tutulla kaavalla

$$\mathbf{C} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T \cdot \mathbf{Z}.$$

Edelleen mallin mukainen ulostulo saadaan kaavalla $[DX', DY'] = \mathbf{HC}$. Alla on kuvat alkuperäisestä datasta sekä mallin ulostulosta. Tee samanlaiset kuvat Matlabilla. Huomaa että ennusteen pystyvektorit DX ja DY täytyy muuttaa matriisiksi `reshape`-komennolla.





Luku 7

Audiodatan kompressio

Ensimmäisen johdatuskurssin lopuksi tarkasteltiin kuvien kompressiota häviöllisin ja häviöttömin menetelmin. Tuolloin tutustuttiin JPEG-standardiin sekä MPEG-standardin videokoodaukseen. MPEG-standardi määrittelee myös menetelmiä audion koodaukseen ja niihin tutustutaan seuraavaksi.

Kappaleessa 3.5 tutustuttiin CD-soittimen ominaisuuksiin ja mainittiin muun muassa, että pakkaamaton CD-audiodata sisältää noin 1.41 miljoonaa bittiä sekuntia kohti. Näin suuri datamäärä muodostuu ongelmaksi muun muassa digitaalisen audion siirrossa, esimerkiksi digitaaliradiota käytettäessä. Suuri osa CD-soittimen biteistä on tarpeettomia kuuntelijan kannalta. Uusista kompressoiduista digitaalisen audion formaateista (esim. mp3-standardi) puhuttaessa oletetaan usein kompression vain heikentävän äänenlaatua. Näin ei suinkaan tarvitse olla, vaan kompression avulla voidaan esimerkiksi siirtää vähemmän tarpeellisia bittejä ihmisen kuuloaistin kannalta merkityksettömiltä alueilta tärkeämmille alueille. Näin on mahdollista parantaa (havaittua) äänenlaatua (esimerkiksi keskiäänien dynaamista aluetta) bittimäärän pysyessä samana.

Tässä kappaleessa esitetään tärkeimmät periaatteet digitaalisen audion pakkaamisessa. Erityisesti tarkastellaan huomattavan suosion saanutta MPEG-standardin äänikoodauksen osuutta. Kyseinen äänikoodauksen standardi jaetaan kolmeen tasoon (layer) niiden kompressiotehon mukaan. Taso 1 tuottaa huonoimman kompressiosuhteen, mutta on yksinkertaisin toteuttaa ja tehovaatimukset ovat pienimmät. Jos laitteistotehoa riittää eikä reaaliaikainen pakkaaminen ole välttämätöntä, voidaan käyttää korkeampia tasoja. Suurempi kompressioteho vaatii suunnittelijoilta enemmän paneutumista enkooderin ohjelmointiin (enemmän ohjelmakoodia) sekä laitteistolta enemmän laskentatehoa. Standardin määrittelyn mukaan ensimmäisen tason on määrä tuottaa suunnilleen CD-tasoinen ääni käyttäen 384 kilobittiä sekuntia kohti. Tasoille 2 ja 3 tämä bittimäärä on 192 kb/s ja 128 kb/s. Äänen laatu mitataan kuuntelutesteissä, ja tulos on CD-tasoinen, jos kuuntelijoiden havaitsema ero alkuperäisen ja pakatun version välillä ei ole tilastollisesti merkittävä. Konetehon kasvaessa käytännön standardiksi useissa sovelluksissa on muodostunut taso 3 sen alhaisemman bittimäärän vuoksi. Reaaliaikaiset sovellukset käyttävät alempaa tasoa; esimerkiksi digitaaliradio (DAB) perustuu tasoon 2.

Häviöllisen kuvanpakkauksen standardit oli mahdollista esittää melko vähäisin esitiedoin. Tärkeintä osaa näyttelivät kuvien muunnokset, erityisesti diskreetti kosinimuunnos eli DCT. Muunnoksen jälkeen osa muunnoskertoimista jätettiin koodaamatta tai koodattiin pienemmällä tarkkuudella.

Audiokoodauskin voitaisiin esittää vastaavalla tavalla. Tällöin riittäisi käydä läpi audiosignaalille tehtävä muunnos ja mainita, että osa kertoimista koodataan pienemmällä tarkkuudella. Bittivähennys pohjautuu kuitenkin monipuoliseen ja -tieteiseen teoriaan ihmisen kuuloaistin havaintoprosessista (tätä tutkiva tieteenala on nimeltään *psykoakustiikka*). Tämä teoria näyttelee niin olennaista osaa audiokoodauksessa ja erityisesti sen tulevaisuudennäkymissä, että myös se on syytä käsitellä. Näin ollen ennen algoritmista vaihetta tutustutaankin äänen ja äänihavainnon väliseen suhteeseen.

7.1 Kuuloaistin ominaisuuksia

7.1.1 Kuuloaistin anatomia

Anatomian kannalta ihmiskorva jakautuu kolmeen osaan, ulko-, väli- ja sisäkorvaan.[1] Ulkokorvaan kuuluvat korvalehti sekä korvakäytävä. Äänisignaalien kannalta korvalehti vahvistaa edestä ja sivuilta tulevia ääniä (n. 5 dB) samalla vaimentaen takaa tulevia ääniä. Erityisesti korvalehti vahvistaa keskiääniä ja amplitudivasteen huippu on noin 2.5 kHz:n kohdalla. Tämän lisäksi myös korvakäytävä vahvistaa keskiääniä 5-20 dB ja vahvistushuippu on noin kolmen kilohertsin kohdalla. Vaikka keskiäänten vahvistuminen osittain kumoutuukin välikorvassa, saapuu sisäkorvaan keskiääniltään vahvistunut äänisignaali.

Sisäkorvan ääniä aistiva osa on nimeltään *simpukka*, jonka sisäosassa on *tyvilevy*. Yksinkertaistettuna tyvilevy on ikäänkuin pitkä soittimen kieli, joka on jäykempi ja kapeampi alkupäästä kuin loppupäästä. Näin ollen suuritaajuiset ääniaallot saavan tyvilevyn kapean osan värähtelemään (vrt. kitaran ohuet kielet) ja pienet taajuudet vaikuttavat enemmän leveään loppupäähän (vrt. kitaran paksut kielet). Tyvilevyn liikkeitä on tunnistamassa 15000 karvasolua, joiden liikkeet välittyvät aivoihin. Aivot saavat näin tiedon värähtelyn maksimiamplitudin sijainnista ja muodostavat sen perusteella tulkinnan taajuudesta.

7.1.2 Kuuloaistin mallintaminen

Kehitettäessä matemaattisia malleja kuuloaistille on toimivaksi ratkaisuksi osoittautunut olettamus, jonka mukaan kuulokäytävä jakaa ääniaallot eri taajuuskomponentteihin eräänlaisten kaistanpäästösuodinten avulla. Erään tutkimuksen tuloksena[2] julkaistiin taulukko 24:sta kaistanpäästösuodattimen rajataajuudesta. Tutkimuksessa oletetettiin suodinten olevan lähellä ideaalisia kaistanpäästösuotimia, mutta myöhempi tutkimus on osoittanut tämän olettamuksen vääräksi.

Melko hyviin tuloksiin päästään kuitenkin mallintamalla kuuloaistia ei-ideaalisilla kaistanpäästösuotimilla. Tällaisten suodinten kaistanleveyden on havaittu riippuvan niiden keskustaajuudesta. Jos tätä riippuvuutta havainnollistetaan täyslogaritmisella asteikolla, voidaan voimakkaasti yksinkertaistaen sanoa kaistanleveyden kasvavan suunnilleen eksponentiaalisesti suhteessa tarkasteltavaan taajuuteen. Näin ollen spektrin estimoinnin resoluutio on paras pienillä taajuuksilla ja heikoin suurilla taajuuksilla.

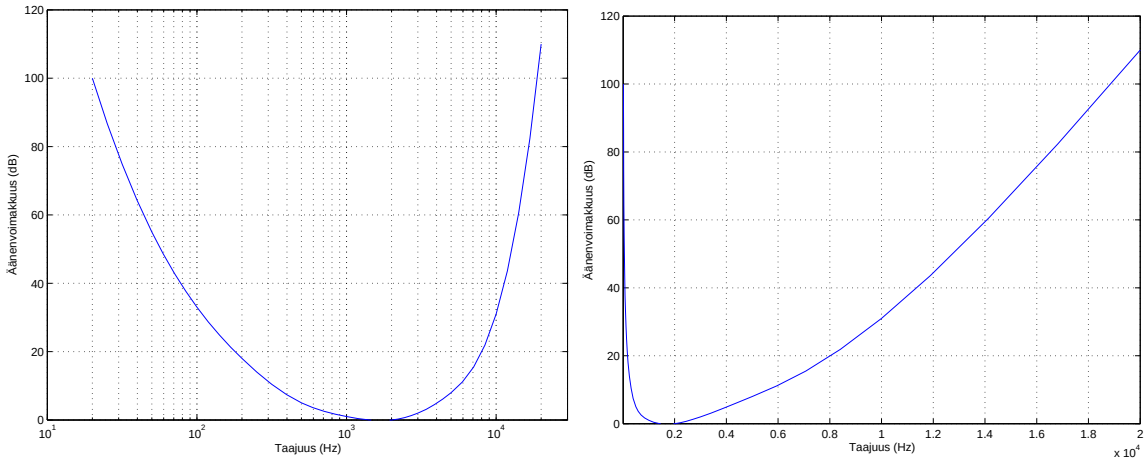
[1] Lisätietoja löytyy esim. E. Haug, O. Sand, Ø. Sjaastad, K. Toverud, *Ihmisen fysiologia*, WSOY 1995.

[2] B. Scharf, "Critical bands", teoksessa *Foundations of modern auditory theory*, Academic press, New York, 1970.

Äänisignaalin jako eri taajuuskaistoihin on pohjana audion koodaukselle. Koodaustehokkuuden kannalta tähän on kaksi syytä. Ensinnäkin tällöin voidaan pakkaus optimoida sopivaksi ihmisen kuulokäyrän mukaan ja toiseksi voidaan käyttää apuna *peittoilmiötä*.

7.1.3 Kuulokäyrä

Kuuntelutesteissä on havaittu ihmisen kuuloaistin herkkyyden noudattavan suunnilleen seuraavien kuvien mukaista kuulokäyrää.



Vasemmanpuoleinen kuva esittää kuulokäyrää logaritmisella asteikolla. Oikeanpuoleisessa kuvassa sama käyrä on esitettynä lineaarisella asteikolla. Logaritminen asteikko on yleisemmin käytetty, koska se kykenee kuvaamaan kuuloaistin toimintaa havainnollisemmin.

Kuvien pystyakseli esittää pienintä havaittua äänenpainetta kullakin taajuudella. Esimerkiksi puhdas sinisignaali taajuudella 500 Hz on mahdollista kuulla, jos sen äänenpäine on vähintään 5 dB. Käyrän alapuolelle jääviä taajuus-äänenvoimakkuus-yhdistelmiä ei ihmiskorva kuule.

Pystyakselilla on desibeleissä ilmaistu äänenvoimakkuus. Äänenvoimakkuudet ilmaistaan suhteessa pienimpään kuultavissa olevaan ääneen, jonka voimakkuus on näin ollen 0 dB (n. 1-3 kHz taajuudella, vaihtelee lähteen mukaan). Toisinaan käytetään merkintää "dB SPL", jossa SPL on lyhenne sanoista *sound pressure level*. Nimi viittaa äänenpainetasoon ja sitä kautta äänen intensiteettiin, jonka yksikkö on W/m^2. Nolla desibeliä määritellään intensiteetiksi 10^{-12} W/m^2. Näin ollen esimerkiksi lukeman 50 dB tarkoittama intensiteetti saadaan yhtälöstä

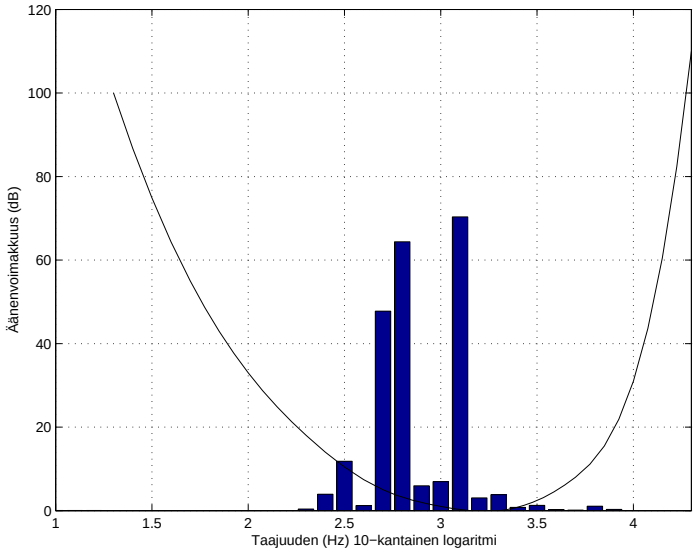
$$10 \log_{10} \frac{x}{10^{-12}} = 50,$$

eli

$$x = 10^{-7}\ W/m^2.$$

Kuulokäyrää hyväksi käyttämällä voitaisiin jo säästää bittejä jättämällä kaikki sen alle jäävät äänet tallentamatta. Alla oleva kuva vertaa erään näytesignaalin taajuuskomponentteja kuulokynnykseen. Kuviosta nähdään, että hyvin pienet ja hyvin suuret taajuudet jäävät kuulokynnyksen alapuolelle ja voidaan näin ollen esimerkiksi suodattaa pois signaalista. Suodatuksen tuloksena on yksinkertaisempi signaali, mikä näkyy entropian pienenemisenä.

Todellisuudessa tilanne ei ole aivan näin yksinkertainen, sillä kuulokynnystä kuvaava käyrä pätee puhtaille siniäänille, joilla on siis vain yksi taajuuskomponentti. Esimerkin puhesignaalin taajuussisältö on monimutkaisempi, jolloin jotain saattaa jäädä kuuluviin. Lisäksi käyrän alle jäävät komponentit tulevat esiin, jos käyttäjällä on mahdollista kontrolloida äänenvoimakkuutta. Näin ollen on tarkasteltava samanaikaisten äänten yhteisvaikutusta kuulohavaintoon.



7.1.4 Peittoilmiö

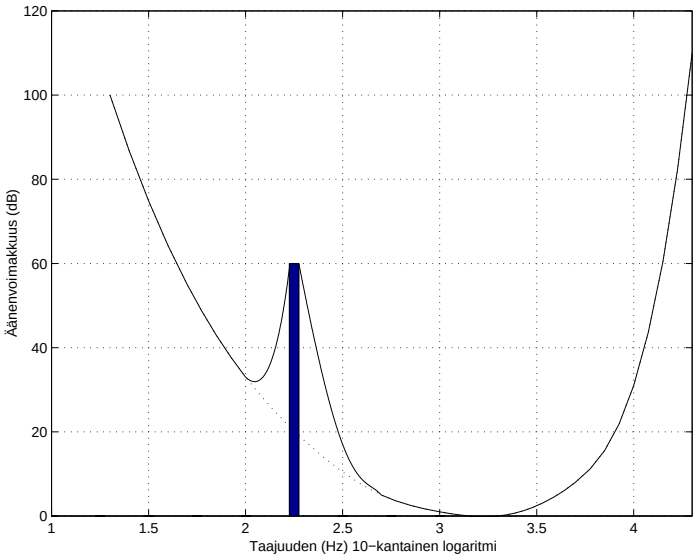
Koodauksen kannalta kuulokynnystä hyödyllisempi kuuloaistin ominaisuus on peittoilmiö (maskeeraus, masking effect). Tuomainen[3] määrittelee peittoilmiön toisen signaalin (maskeeraajan, maskin) aiheuttamaksi häirinnäksi kohdesignaalin havaitsemisessa. Maskaajasignaali on siis voimakkaampi ääni, joka peittää heikomman äänen alleen. Käytännössä tämä ilmenee esimerkiksi siten, että kovaäänisen musiikin (maski) soidessa normaali puhe (kohdesignaali) ei kuulu.

Yleensä peittoilmiö jaetaan kahteen eri tyyppiin, *samanaikaiseen* ja *ei-samanaikaiseen*. Samanaikaisessa peittoilmiössä sekä maskaajasignaali että kohdesignaali kuullaan yhtäaikaa. Jos maskaajasignaali on kohdesignaalia voimakkaampi, kohdesignaali peittyy kuulumattomiin. Samalla peittyy osa maskaajasignaalia ympäröivistä taajuuksista oheisen kuvan mukaisesti.

Kuvassa on esitetty yksi voimakas taajuuspiikki taajuudella $10^{2.25} \approx 178$ Hz. Tällöin kuulokynnys muuttuu siten, että korva ei kuule ympäröiviä taajuuksia kuulokynnyksen mukaisesti vaan huonommin. Nyt kaikki kuvan yhtenäisen viivan alle jäävät äänet (taajuusvoimakkuusyhdistelmät) jäävät kuulumattomiin. Älykäs koodausmenetelmä ymmärtää tällöin koodata kyseiset taajuudet pienemmällä tarkkuudella ja säästää näin bittejä.

Yksittäisen äänen muodostama maskikuvio riippuu myös äänen luonteesta. Kohinatyyppiset ja laajakaistaiset äänet peittävät muita ääniä alleen tehokkaammin kuin säännölliset äänet.

[3] J. Tuomainen, "Psykoakustiikka: akustiikan ja havainnon välinen suhde", teoksessa *Revonsuo et al. (toim.), Mieli ja aivot: kognitiivinen neurotiede*, Kognitiivisen neurotieteen tutkimusyksikkö, Turun yliopisto, Turku, 1996.



Ei-samanaikainen peittoilmiö tarkoittaa sitä, että voimakas yksittäinen ääni peittää muita lähitaajuuksilla olevia ääniä myös jonkin aikaa sen jälkeen kun maskaajaääni itse on loppunut. Tämä johtunee siitä, että korva ei ehdi aivan heti palautua entiselleen eikä havaitse heikkoja lähitaajuuksien ääniä aivan täydellisesti. Ei-samanaikaisen peittoilmiön kesto on kymmenien tai satojen millisekuntien luokkaa. Esimerkiksi 80 ms = 0.08 s vastaa 44.1 kHz:n näytteenottotaajuudella 3528 näytettä. Näin ollen muutaman tuhannen seuraavan näytteen kyseiset taajuudet voidaan koodata pienemmällä tarkkuudella. Bittisäästö voi näin muodostua merkittäväksi. Ei-samanaikainen peittoilmiö ei kuitenkaan ole yhtä voimakasta kuin samanaikainen, eikä sen avulla voida näin ollen saavuttaa yhtä merkittäviä bittisäästöjä.

Yllättävä havainto on, että ei-samanaikainen peittoilmiö toimii myös toiseen suuntaan. Tällöin voimakas maskaajaääni peittää lähitaajuudet myös *ennen* alkamistaan. Tuomainen viittaa selitysmalliin, jonka mukaan ilmiö johtuu lyhytaikaisen sensorisen muistin (ns. kaikumuistin) toiminnasta. Riippumatta syistä, ilmiötä on helppo hyödyntää ohjelmistotasolla samaan tapaan kuin peittoilmiötä yleensäkin.

7.2 Peittoilmiön hyväksikäyttö audion koodauksessa

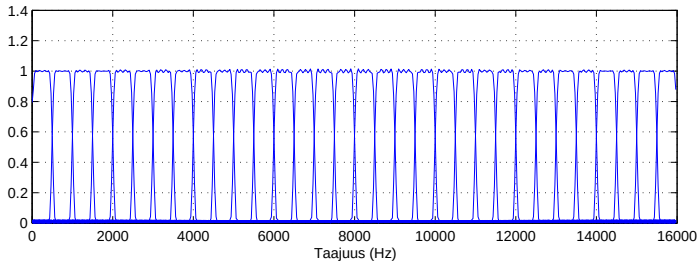
Molempiin peittotyyppeihin on olemassa matemaattiset mallit, jotka toimivat kohtuullisen hyvin. MPEG-standardikin määrittelee kaksi ohjelmarutiinia, jotka laskevat annetun signaalin kokonaismaskin. Kokonaismaski tarkoittaa eri taajuuskomponenttien yhteisvaikutusta. MPEG-standardin malli 1 on yksinkertaisempi ohjelmoitava ja nopeampi laskettava, mutta toisaalta epätarkempi. Näin ollen se soveltuu tilanteisiin, joissa laskennan on oltava nopeaa ja bittimäärän ei tarvitse olla optimaalinen. Malli 2 on monimutkaisempi ja tarkempi.

Eräs peruslähtökohdista MPEG-standardointityössä on ollut jättää mahdollisimman paljon vapauksia enkooderin sunnittelijalle. MPEG määrittelee vain bittivirran, mutta ei menetelmiä sen luomiseksi. Standardin mallit ovat vain (hyviä) esimerkkejä. Mikään ei pakota pakkausohjelman suunnittelijaa käyttämään kumpaakaan niistä. Nopea (joskin tuhlaileva) ratkaisu on olla käyttämättä kumpaakaan psykoakustisista malleista. Toisaalta ke-

hittyneiden pakkausohjelmien algoritmit perustuvat viritettyihin psykoakustisiin malleihin, jotka ovat standardimalleja tarkempia. Mitä tarkempi malli on, sitä enemmän bittejä voidaan jättää pois ilman kuuluvaa eroa alkuperäiseen. Lisätietoja aiheesta löytyy verkosta vaikkapa hakusanoilla "*psychoacoustic model MPEG*".

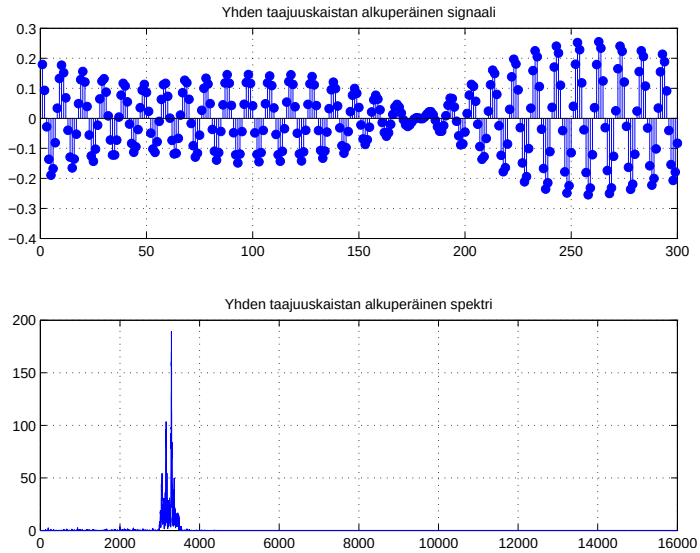
7.2.1 Taajuusalueiden erottelu

Jotta peittoilmiötä voitaisiin käyttää tehokkaasti hyväksi, äänisignaali jaetaan ensin eri taajuuskomponentteihin. MPEG-standardin mukaan tämä tapahtuu 32:lla kaistanpäästösuotimella, joiden päästökaistat ovat yhtä leveät ja erilliset. Jos esimerkiksi näytteenottotaajuus on 32 kHz, jokaisen suotimen päästökaistan leveys on $16000/32 = 500$ Hz. Ensimmäisen suotimen päästökaista on siis 0–500 Hz, toisen 500–1000 Hz ja niin edelleen. Tällaisesta suodinkokoelmasta käytetään nimeä *suodinpankki* (*filter bank*) ja erään suodinpankin suodinten amplitudivasteet on esitetty alla olevassa kuvassa.



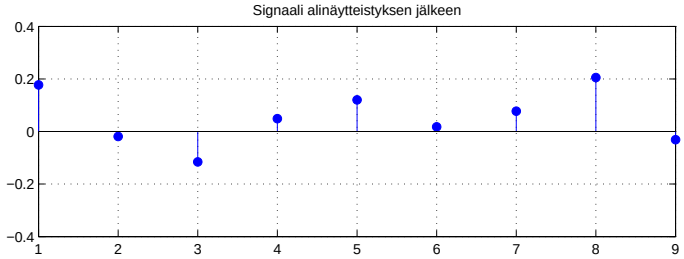
Äänisignaali suodatetaan jokaisella suodinpankin suotimella, ja tuloksena saadaan 32 signaalia joista kukin esittää yhtä taajuuskaistaa. Näiden 32:n signaalin näytteenottotaajuus pudotetaan kertoimella 32 eli yhteen kilohertsiin. Tällöin korkeammat taajuudet laskostuvat taajuusalueelle 0–500 Hz. Tämä ei kuitenkaan haittaa, sillä tuon taajuusalueen informaatio suodatettiin pois aiemmin. Näin ollen laskostuminen ei sekoita taajuusinformaatiota ja alkuperäinen signaali voidaan palauttaa *moduloimalla* desimoitua versiota.

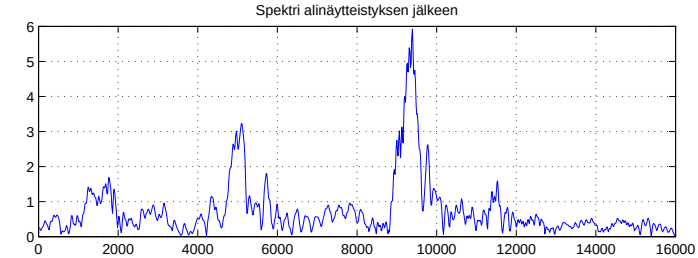
Alla olevassa kuvassa on yksi 500 Hz levyinen kaista alkuperäisestä signaalista esitettynä aika- sekä taajuusalueella.



Kuvasta nähdään, että kapeakaistaisuudesta johtuen signaali on melko yksinkertainen suureen näytteenottotaajuuteen nähden. Kun signaali desimoidaan kertoimella 32, taa-

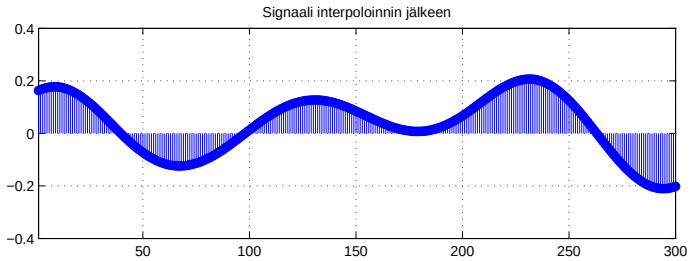
juusalue 3000–3500 Hz laskostuu taajuuksille 0–500 Hz. Samalla näytteiden määrä putoaa yhteen kolmaskymmeneskahdesosaan alkuperäisestä. Tulos on esitetty alla olevassa kuvassa.

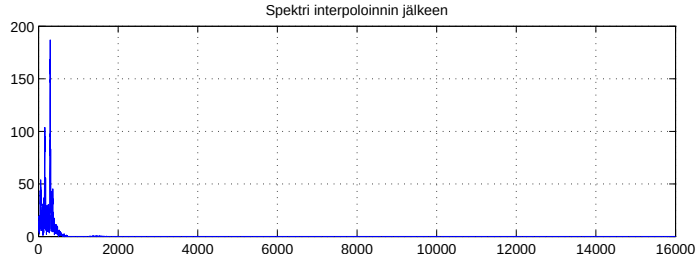




Lopputuloksena on siis 32 signaalia, joiden näytteenottotaajuus on 1 kHz ja joista kukin esittää yhtä taajuuskaistaa alkuperäisestä signaalista. Näytteet on esitetty 16:lla bitillä kuten aiemminkin. Periaatteessa tässä vaiheessa alkuperäinen signaali on vielä täydellisesti rekonstruoitavissa. Lisäksi näytearvojen lukumäärä on sama kuin ennen jakoa eri taajuuskaistoille (alussa yksi signaali näytteenottotaajuudella 32 kHz, nyt 32 signaalia näytteenottotaajuudella 1 kHz). Enkooderi tallentaa signaalin nimenomaan tässä esitysmuodossa pienennettyään ensin kunkin taajuuskaistan bittimäärää seuraavan kappaleen periaatteiden mukaan.

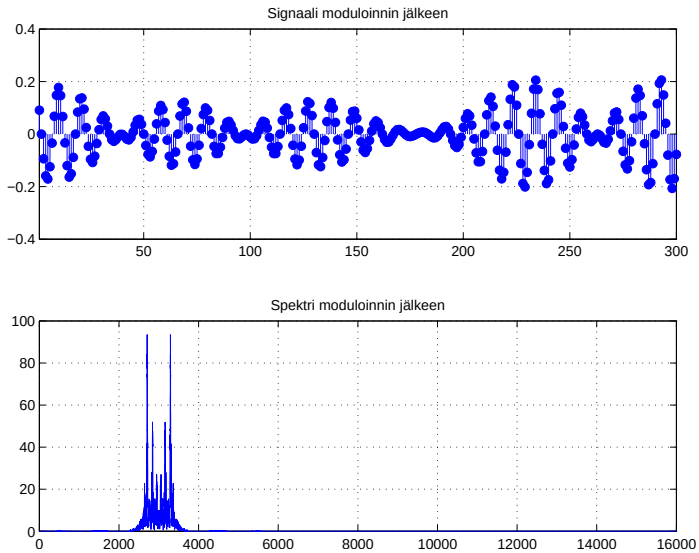
Tästä voidaan dekoodausvaiheessa saada palautettua miltei tarkalleen alkuperäinen signaali. Ensimmäisessä vaiheessa interpoloidaan kertoimella 32, jolloin saadaan alla olevan kuvan mukainen signaali.



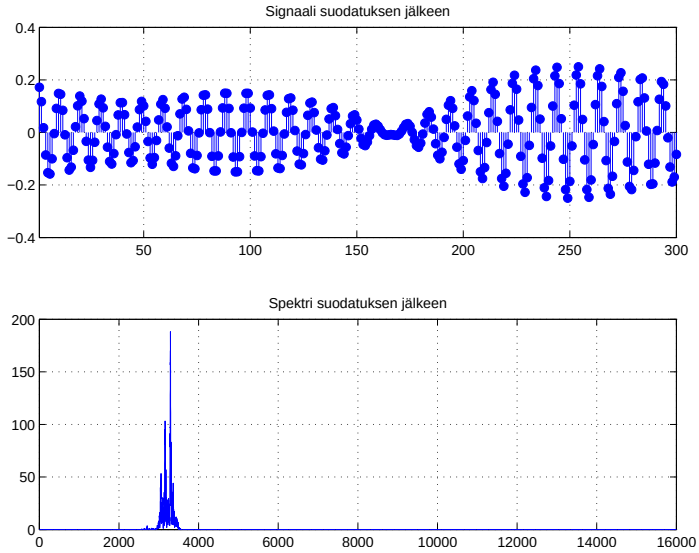


Tämän jälkeen signaali moduloidaan kosinisignaalilla, jonka taajuus on 3000 Hz. Kertolasku aikatasossa vastaa konvoluutiota taajuustasossa, ja nyt konvoluutio otetaan siis signaalin spektrin ja kosinin spektrin (yksi piikki taajuudella 3000 Hz) välillä. Tuloksena

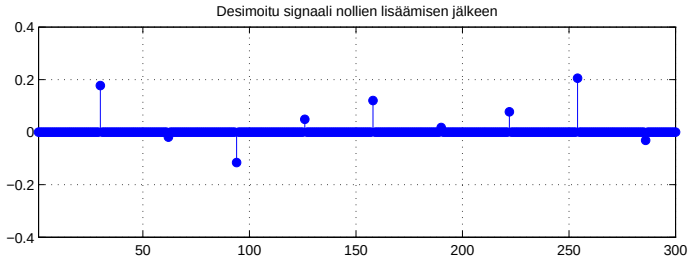
alkuperäinen spektri siirtyy 3000 Hz suuremmalle taajuudelle. Tulos näkyy alla olevassa kuvassa.

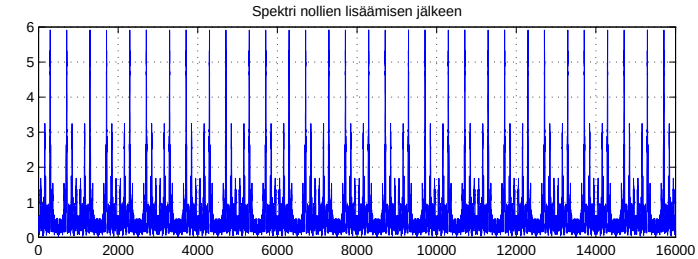


Spektristä nähdään, että myös negatiiviset taajuudet siirtyvät taajuusasteikolla 3000 Hz ylöspäin, mutta ne voidaan poistaa ylipäästösuodatuksella. Tulos on esitetty alla ja se on hyvin lähellä alkuperäistä. Vähäiset muutokset johtuvat desimoitaessa tapahtuvasta laskostumisesta. Todellisessa järjestelmässä nekin kumoutuvat muiden taajuuskaistojen laskostumisilmiöiden kanssa.

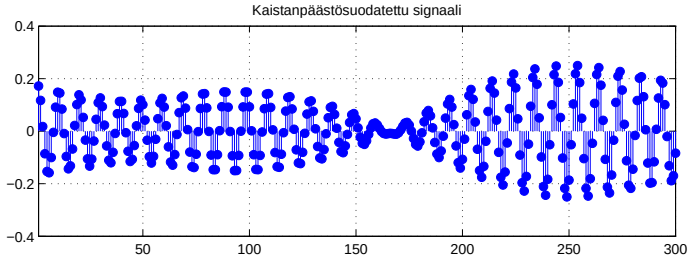


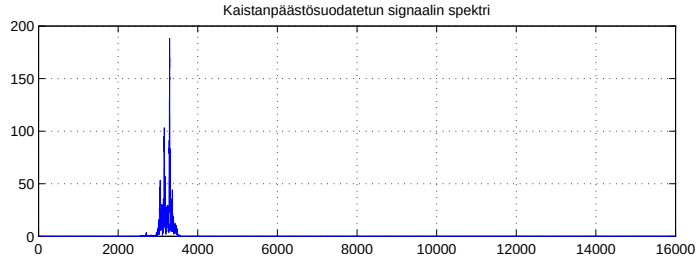
Rekonstruktiovaihetta voidaan yksinkertaistaa jättämällä modulointi pois. Tämä onnistuu tarkastelemalla näytteenottotaajuuden kasvattamista (lohkokaaviossa $\boxed{\uparrow N}$) lähemmin. Tämän operaation jälkeen (ennen interpoloinnin jälkeistä alipäästösuodatusta) signaalin spektri näyttää seuraavalta.





Kuten interpoloinnin yhteydessä havaittiin, nollien lisääminen muodostaa kopioita alkuperäisestä spektristä korkeammille taajuuksille. Koska alkuperäinen taajuuskaista sijaitsi korkeammalla taajuudella, voidaan yhtä hyvin käyttää siellä valmiiksi olevaa kopiota. Alipäästösuodatuksen ja moduloinnin asemesta voidaan siis käyttää vain yhtä kaistanpäästösuodatusta, joka erottelee kaistalla 3000 Hz–3500 Hz olevan kopion. Tulossignaali spektreineen on alla.

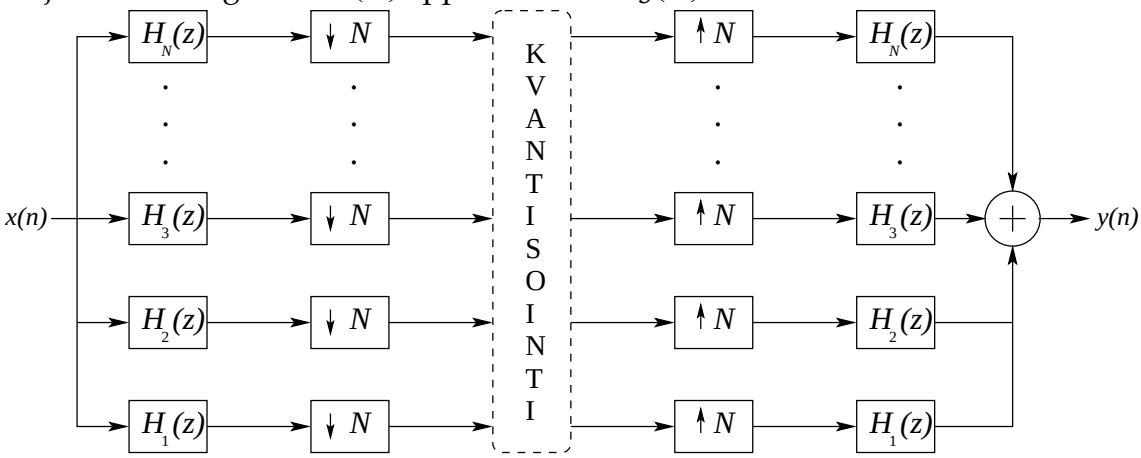




Joka toinen spektrin kopio on alkuperäisen spektrin peilikuva. Tämä ei kuitenkaan haittaa, sillä samat spektrit peilautuivat jo kerran laskostuessaan näytteenottotaajuutta pienennettäessä. Näin ollen niiden spektrit ovat todellisen spektrin peilikuvia silloin, kun näytteenottotaajuutta aletaan nostaa. Peilautuessaan toisen kerran nollia lisättäessä ne kääntyvät automaattisesti oikein päin.

Suodinpankin lohkokaavio on alla olevassa kuvassa. Vasemmalla signaali $x(n)$ jaetaan N:llä kaistanpäästösuotimella taajuuskaistoihin ja yksittäisiä taajuuskaistoja esittävien signaalien näytteenottotaajuus pudotetaan kertoimella N. Taajuuskaistoja esittävät signaalit voidaan nyt pakata, eli ne voidaan esittää pienemmällä tarkkuudella kuten seuraavassa kappaleessa esitetään. Kun alkuperäinen signaali halutaan palauttaa kompressoidusta versiosta, nostetaan taajuuskaistojen näytteenottotaajuudet ensin nollia lisäämällä samak-

si kuin alkuperäisellä signaalilla. Näin saaduista signaaleista irrotetaan oikea taajuuskaista kaistanpäästösuodatuksella. Tässä voidaan käyttää samaa suodinta, jolla kaista alunperin erotettiin signaalista $x(n)$. On tosin mahdollista käyttää muutakin suodinta, mutta suotimet $H_1(z), H_2(z), \ldots, H_N(z)$ kelpaavat yhtä hyvin. Näin saadut taajuuskaistat summataan yhteen ja saadaan signaalin $x(n)$ approksimaatio $y(n)$.

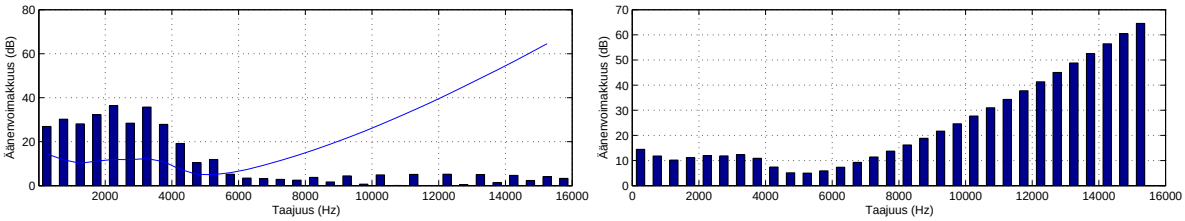


Pakkaamisen kannalta olennaisin vaihe on seuraava. Siinä taajuuskaistojen esittämiseen tarvittavia bittimääriä aletaan vähentää. Bittivähennys tapahtuu siitä esitysmuodosta, jossa on 32 yhden kilohertsin näytetaajuudella olevaa signaalia.

7.2.2 Kvantisointikohinan peittyminen ja bittien jakaminen eri taajuuksille

Tässä vaiheessa on käytössä rutiini (psykoakustinen malli), joka laskee kokonaismaskin parametrina saamansa signaalin perusteella. Tätä mallia voidaan verrata signaalin taajuussisältöön. Alla olevista kuvista vasemmanpuoleinen esittää erään signaalin spektriä (pystypylväät) sekä siitä ja kuulokäyrästä laskettua kokonaismaskia (yhtenäinen käyrä)[4]. Oikeanpuoleisessa kuvassa on kokonaismaski. Kokonaismaski noudattaa siis kuulokäyrän muotoa, mutta signaalin voimakkailla taajuuksilla nousee hieman ylemmäs.

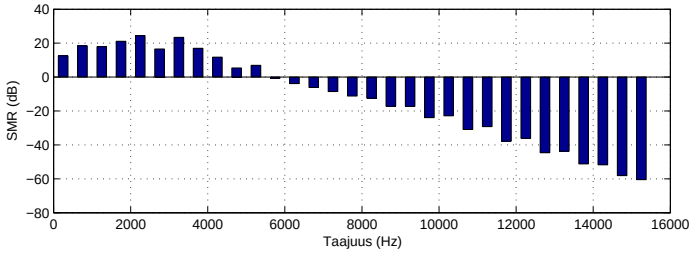
Kokonaismaskista nähdään kullakin taajuusalueella voimakkain signaali, joka alkuperäiseen voidaan lisätä ilman että kuuntelija havaitsee sitä. Näin ollen kuvaaja ilmaisee myös, kuinka suuri määrä *kvantisointikohinaa* signaalin eri taajuuksilla voi olla. Tästä voidaan päätellä pienin bittimäärä jolla kukin 32:sta eri taajuuskaistoilla olevasta signaalista täytyy esittää ilman havaittavaa laadun heikkenemistä.



Signaalin sisältämiä taajuuksia on tapana vertailla kokonaismaskiin tarkastelemalla näiden erotusta. Seuraavassa kuvassa on kyseinen erotus (siis ylhäällä vasemmassa ku-

[4]Esimerkin signaali on todellinen äänisignaali, mutta kokonaismaski on vain esimerkki eikä sitä ole laskettu millään käytössä olevalla rutiinilla.

vassa olevat pylväät miinus oikeassa kuvassa olevat pylväät). Tästä erotuksesta käytetään nimeä *signaali-maski-suhde* (signal-to-mask ratio, SMR).



Signaali-maski-suhdetta voidaan vertailla siihen signaali-kohina-suhteeseen, joka syntyy esitettäessä signaali b:llä bitillä. Käytetään tästä merkintää $\mathrm{SNR}(b)$. Tällöin riittävä ehto sille, ettei bittivähennyksestä seuraava kvantisointikohina kuulu, on

$$\mathrm{SMR} < \mathrm{SNR}(b). \tag{7.1}$$

Signaali-kohina-suhde riippuu signaalin amplitudista, mutta tässä esimerkissä voidaan olettaa vastaava malli kuin kappaleessa 1. Tuolloin päädyttiin kaavaan

$$\mathrm{SNR}(b) \approx 6.02b - 1.25.$$

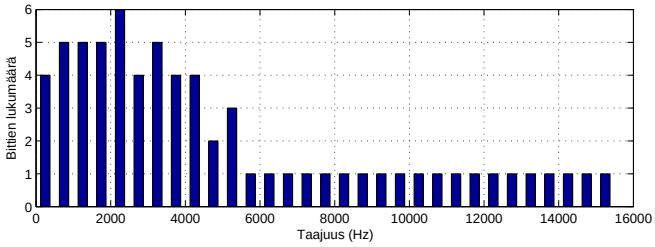
Sijoittamalla tämä lauseke kaavaan (7.1), saadaan epäyhtälö

$$\mathrm{SMR} < 6.02b - 1.25.$$

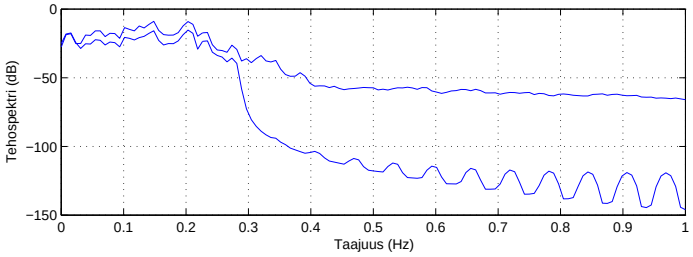
Tästä on helppo ratkaista b:

$$b > \frac{\mathrm{SMR} + 1.25}{6.02}.$$

Tämän kaavan mukaiset minimibittimäärät eri taajuuskaistoille on esitetty seuraavassa kuvassa.



Käytännön tilanteissa voisi olla varminta antaa bittejä hieman avokätisemmin myös korkeille taajuuksille. Jos kokeeksi kullekin taajuudelle annetaan yksi ylimääräinen bitti yllä olevien lisäksi, saadaan koodauksen ja dekoodauksen jälkeen seuraava taajuussisältö.



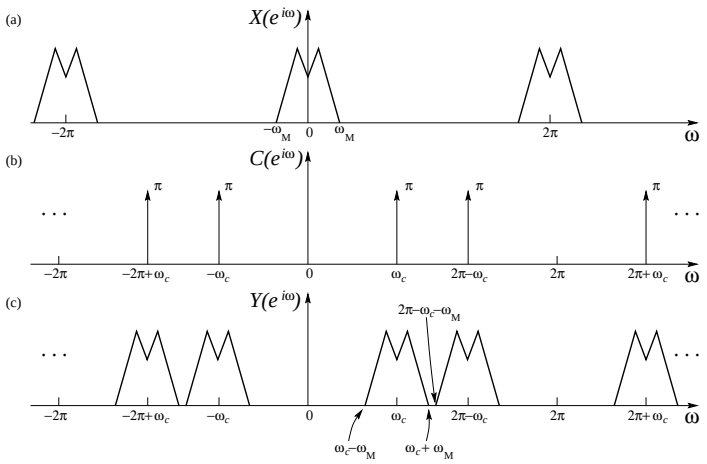
Ylempi kuvaaja esittää alkuperäisen signaalin spektriä ja alempi häviöllisen koodauksen tulossignaalin spektriä. Bittejä on nyt koko 32000 näytteen mittaiseen signaaliin käytetty 60000, eli 1.88 bittiä/näyte alkuperäisen 16 bittiä/näyte sijasta[5]. Ero näkyy selvästi, ja on myös kuultavissa äänisignaalissa, mutta se on yllättävän pieni ottaen huomioon kompressiosuhteen ja miltei arvaamalla valitun kokonaismaskin. Tämän lisäksi näyte oli pitkä verrattuna todellisten koodereiden käyttämiin ikkunoihin (mp3:ssa tyypillinen ikkunan pituus on $64 - 1024$ näytettä). Siksi myös taajuudet olivat enemmän hajallaan.

Esimerkissä bittejä jaettiin signaalin taajuussisällön mukaan. Tästä seuraa, että bittitarve vaihtelee ajan myötä. Tällaisesta koodauksesta käytetään MPEG:in yhteydessä nimeä *variable bitrate* (VBR) ja sopiva suomennos voisi olla *vaihtuvan bittinopeuden koodaus*. Useissa mp3-koodereissa ei kuitenkaan ole sen hyödyllisyydestä huolimatta mahdollisuutta VBR:n käyttöön, vaan bittinopeus on kiinnitetty etukäteen.

Jos bittinopeus on kiinnitetty, kooderilla on käytössään tietty bittimäärä kullekin ikkunalle. Tällöin ne jaetaan taajuuksien kesken iteratiivisesti luovuttamalla yksi bitti kerrallaan sille kaistalle, jonka SMR:n ja SNR:n erotus[6] kulloinkin on suurin. Toisinaan bittejä on enemmän kuin tarvitaan ja toisinaan niitä voitaisiin käyttää enemmänkin. MPEG:in tasot 1 ja 2 eivät vaadi dekoodereilta mahdollisuutta VBR:n käyttöön, mutta jokaisen tason 3 dekooderin täytyy tukea VBR:ää.

Harjoitustehtäviä

7.1. Kun kapeakaistainen signaali $x(n)$ halutaan siirtää korkeammalle taajuusalueelle, käytetään modulointia. Tällöin signaali $x(n)$ kerrotaan pisteittäin *kantoaallolla* $c(n) = \cos(2\pi f_c n)$, missä f_c kertoo kuinka paljon signaalin $x(n)$ taajuuksia halutaan nostaa (esimerkiksi arvolla $f_c = 0.25$ taajuudet nousevat puolella Nyquistin rajataajuudesta). Millainen spektri on tuloksena, kun signaali $x(n) = \sin(0.1 \cdot 2\pi n)$ moduloidaan kantoaallolla, missä $f_c = 0.3$? Piirrä tilanteesta vastaavat kolme kuvaajaa kuin kuvassa (kuvassa normalisoidun taajuuden f_c tilalla on kulmataajuus ω_c, joten on muistettava, että $\omega_c = 2\pi f_c$). Vapaaehtoinen lisäkysymys: osaatko sanoa mikä DFT:n ominaisuus selittää modulaation?



[5]Entropiakoodaus parantaa kompressiosuhdetta edelleen.

[6]Tästä erotuksesta käytetään nimeä *noise-to-mask ratio*: $\mathrm{NMR} = \mathrm{SMR} - \mathrm{SNR}(b)$.

7.2. Jotta signaalin spektri pysyisi muuttumattomana, eivät vierekkäiset spektrin kopiot saa mennä päällekkäin modulaatiossa. Mitkä kaksi epäyhtälöä kuvan muuttujien ω_c ja ω_M välillä täytyy tällöin olla voimassa? Jos haluat, voit muotoilla epäyhtälöt myös muuttujien f_c ja f_M avulla, missä $\omega_c = 2\pi f_c$ ja $\omega_M = 2\pi f_M$.

7.3. *(Matlab)* Kopioi itsellesi tiedosto `http://www.cs.tut.fi/kurssit/SGN-1251/seiska.mat` ja lataa se Matlabiin. Ääninäytteen näytteenottotaajuus on 16000 Hz. Suunnittele `fir1`-rutiinilla alipäästösuodin (n. sata kerrointa), jonka rajataajuus on kahdeksasosa näytteenottotaajuudesta eli neljäsosa Nyquistin rajataajuudesta. Suodata muuttujassa `x` oleva signaali suunnitellulla suotimella. Tällöinhän tulossignaalin taajuudet ovat välillä $0 - 2$ kHz ja välillä $2 - 8$ kHz on tyhjää tilaa (katso spektrogrammi). Tehtävän 7.2 merkinnöillä ilmaistuna $f_M = \frac{1}{8}$. Minkä muodon tehtävän 7.2 kaksi ehtoa nyt saavat ja missä rajoissa kantoaallon taajuuden f_c tulee olla? Moduloi nyt suodattamasi signaali samanmittaisella kantoaallolla $c(n) = \cos(2\pi f_c n)$, missä f_c on mainittujen rajojen välissä, ja katso spektrogrammi. Modulointi nosti signaalin spektriä ylöspäin siten, että nollataajuus nousi f_c:n verran korkeammalle. Tee `remez` tai `fir1`-rutiinilla suodin, joka poistaa negatiiviselta puolelta taajuuksille $0 - f_c$ tulleet haamutaajuudet. Kuuntele tulos ja katso spektrogrammi. Tee samat testit myös muutamalla muulla sopivalla muuttujan f_c arvolla.

7.4. *(Matlab)* Edellisen tehtävän moduloidut signaalit saadaan palautettua alkuperäisiksi *demoduloimalla* ne, eli käytännössä kertomalla ne uudelleen kantoaallolla ja poistamalla muut kuin alkuperäisessä signaalissa olleet taajuudet $[0, f_M]$. Tee testit edellisen tehtävän kantoaalloilla ja vertaa demoduloinnin tulosta alkuperäiseen spektrogrammissa ja kuuntelemalla.

7.5. *(Matlab)* Laaditaan MPEG-standardin mukainen suodinpankki, jota voidaan käyttää taajuuksien erottelussa 32:lle kaistalle. Oletetaan, että näytteenottotaajuus on 32 kHz, jolloin yksi taajuuskaista on 500 Hz. Luo ensin *prototyyppisuodin*, jonka perusteella muut suotimet suunnitellaan[7]. Kyseinen suodin on alipäästösuodin, joka päästää läpi taajuudet 0-250 Hz ja poistaa muut (negatiiviselta puolelta mukaan tulevat 'haamutaajuudet' -250-0 Hz, siis yhteensä 500 Hz). Otetaan siis päästökaistaksi 0-180 Hz ja estokaistaksi 320-16000 Hz. Suunnittele Remez-algoritmilla tällainen suodin, jossa on 512 kerrointa (siis 512 on kerrointen lkm., ei aste). Kaistojen painot voit itse valita. Kerro lopuksi kaikki kertoimet kahdella.

Jos em. suotimen kertoimista käytetään merkintää $h(n)$, kun $n = 0, 1, \ldots, 511$, saadaan j:nnellä taajuuskaistalla oleva suodin kaavasta

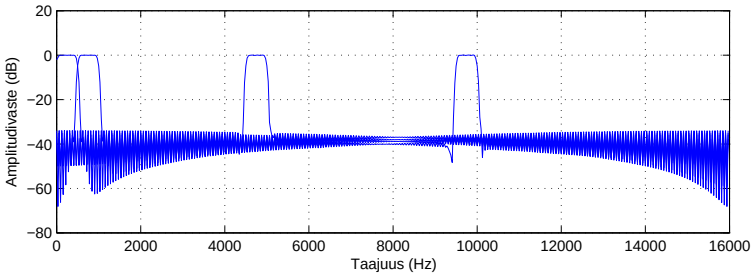
$$h_j(n) = h(n) \cdot \cos\left(\frac{(2j-1)(n-16)\pi}{64}\right),$$

missä $j = 1, 2, 3, \ldots, 32$. Kyseessä on siis prototyyppisuotimen modulointi.

Muodosta näin saaduista suotimista 32×512-matriisi, jossa kullakin 32:lla vaakarivillä on j:nnen suotimen 512 kerrointa. Ensin kannattanee luoda sopivan kokoinen nollamatriisi (`help zeros`) ja sitten for-silmukassa täyttää rivi kerrallaan. Tulosta

[7]Standardin prototyyppisuodin ei ole juuri tämä suodin, mutta vastaava.

kokeeksi muutaman suotimen taajuusvaste. Alla olevassa kuvassa on esimerkiksi kaistoilla 1 (0-500 Hz), 2 (500-1000 Hz), 10 (4500-5000 Hz) sekä 20 (9500-10000 Hz) operoivien suodinten amplitudivasteet.



7.6. *(Matlab)* Lataa signaali `handel` ja ota siitä sekunnin mittainen näyte alusta (8192 näytettä). Irrota sen yhdeksäs taajuuskaista edellisessä tehtävässä suunnittelemallasi suotimella. (Näytteenottotaajuus on toinen, joten taajuusalue on alempana kuin aikaisemmassa tehtävässä. Tällä ei ole tämän tehtävän kannalta olennaista merkitystä.) Tulosta tuloksen spektri ruudulle. Voit nyt hätätapauksessa tehdä spektriestimaatin pelkällä FFT:llä. Ota tuloksesta talteen ainoastaan joka 32:s näyte (vaikkapa kaksoispisteen avulla, `help colon`)[8]. Interpoloi tämä signaali alkuperäisen mittaiseksi, ts. kertoimella 32 (`help interp`). Tulosta spektri ruudulle. Moduloi lopuksi viimeisintä signaalia kosinisignaalilla

$$c(n) = \cos\left(\frac{9-1}{32} \cdot \pi n\right).$$

Luku '9' viittaa yhdeksänteen taajuuskaistaan ja '32' on kaistojen yhteismäärä. Tulosta moduloidun signaalin spektri ruudulle. Huomaat, että mukaan on tullut myös spektrin peilikuva negatiiviselta puolelta. Suodata saamasi signaali alkuperäisellä suotimella, jolloin peilikuvan pitäisi poistua spektristä (tämä on yksi mahdollinen tapa päästä eroon peilikuvasta, muitakin on).

7.7. (a) Erään signaalin taajuuskaistalla 5000 Hz – 5500 Hz oleva äänenvoimakkuus on 13.5 dB. Kyseisen kaistan kokonaismaski on puolestaan 5.2 dB. Montako bittiä/näyte tarvitaan (monisteen mukaisilla kaavoilla) taajuuskaistalla olevan signaalin koodaamiseen, jotta eroa alkuperäiseen ei kuulla?

(b) Taajuuskaistalla 7000 Hz – 7500 Hz oleva äänenvoimakkuus on 14.1 dB ja kaistan kokonaismaski on 16.0 dB. Montako bittiä tarvitaan, jotta eroa alkuperäiseen ei kuulla?

7.8. *(Matlab)* Tehtävässä 7.5. suunniteltiin suodinpankki, joka jakaa signaalissa olevat taajuudet 32:een eri taajuuskaistaan. Suunnittele tällä skriptillä kyseinen suodinpankki. Lataa signaali `handel` muuttujaan `y` käskyllä `load handel` ja jaa se taajuusalueisiin suodinpankillasi (for-silmukassa, tms.). Muodosta suodatustuloksista matriisi, jossa on 32 kappaletta 73113 näytteen mittaisia signaaleita.

7.9. *(Matlab)* Ilman psykoakustista mallia jokaista kaistaa käsitellään samalla tavalla. Kokeillaan tässä tehtävässä psykoakustisen mallin vaikutusta.

[8] Tämä olisi nyt yksi niistä signaaleista, jotka talletetaan.

Lataa kurssin kotisivulta tiedosto

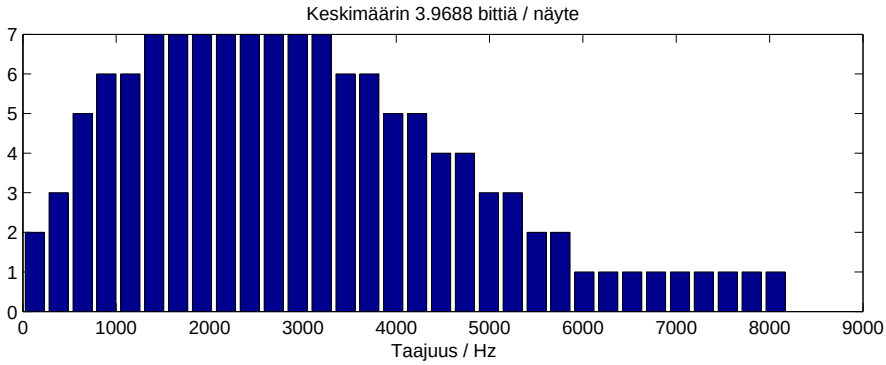
`http://www.cs.tut.fi/kurssit/SGN-1251/Bittijakauma.mat`

ja lataa se Matlabiin komennolla `load`. Vektoriin `bits` latautuu tällöin 32 kokonaislukua, jotka on esitetty alla olevassa kuvassa.

Kvantisoi kukin tehtävän 7.8 tulossignaaleista vektorin `bits` ilmoittamaan määrään ja summaa tulokset yhteen. Kvantisointia ennen kukin kaista täytyy skaalata välille $[0, 1]$. Esim. kaistalla k oleva signaali x kvantisoidaan y:ksi tähän tapaan:

```
M = max(abs(x));
y = M * quant(x / M, 2^(-bits(k)));
```

Näin saatu tulos on siis kvantisoitu keskimäärin 3.9688 bittiin / näyte. Kuuntele tulos ja vertaa sitä siihen mitä saadaan jos kvantisoidaan alkuperäinen signaali suoraan neljään bittiin ilman jakoa 32 kaistaan. Ero ei ole suuri, mutta pitäisi selvästi kuulua.



Luku 8

Digitaaliset signaaliprosessorit

Tässä kappaleessa tutustutaan digitaalisiin signaaliprosessoihin (engl. *digital signal processor; DSP*), eli prosessoreihin, jotka on suunniteltu nimenomaisesti digitaalisen signaalinkäsittelyn tarpeisiin.

Tietokoneiden suorittamat tehtävät voidaan jakaa kahteen luokkaan: yleiseen tiedon käsittelyyn (järjestäminen, haku, jne.) sekä aritmeettisiin operaatioihin. Perinteisesti työasemissa käytettävät yleiskäyttöiset prosessorit on suunniteltu tehokkaiksi ensisijaisesti ensimmäisessä tehtävissä. Signaaliprosessoreissa sen sijaan on erityisesti panostettu aritmeettisten tehtävien nopeaan suoritukseen. Tämä saadaan aikaiseksi toteuttamalla piiritasolla tiettyjä käskyjä, jotka tulevat usein vastaan DSP-intensiivisissä sovelluksissa. Prosessorin tulee nimittäin olla nopea kertolaskuissa ja summauksissa, sillä esimerkiksi FIR-suodatuksessa konvoluutio lasketaan tulojen summana. Usein signaaliprosessorit on optimoitu myös kompleksilukulaskentaa edellyttävän FFT:n laskemista varten.

Lisäksi signaaliprosessorien arkkitehtuuri on suunniteltu tehokkaaksi suurien datamäärien jatkuvaan käsittelyyn. Signaaliprosessorilla tiettyyn tehtävään liittyvän suoritusajan tulee nimittäin olla ennustettavissa ja mieluiten kiinteä datasta riippumatta. Prosessointi onkin signaaliprosessoreissa yleensä reaaliaikaista ja jatkuvaa. Jos prosessori on esimerkiksi osana kuulolaitetta, prosessorin pitää suoriutua tehtävästään pitkiä aikoja yhtäjaksoisesti ilman katkoksia, jotka aiheutuvat siitä, ettei prosessori pysy mukana AD-muuntimen vauhdissa.

Perinteiset yleiskäyttöiset prosessorit noudattavat von Neumannin 1940-luvulla ehdottamaa *von Neumann -arkkitehtuuria*, jossa suoritettava ohjelma ja käsiteltävä data ovat samassa muistilaitteessa. Esimerkiksi konvoluution perusoperaatio, eli kahden luvun kertominen keskenään vaatii tällöin kolme kellojaksoa (kahden luvun haku muistista ja tuloksen siirto muistiin). Koska suotimen kertoimet pysyvät kuitenkin vakioina, saadaan laskusta helposti nopeampi käyttämällä erillisiä muisteja datalle ja ohjelmalle. Tällöin suotimen kertoimet sijaitsevat ohjelmamuistissa. Prosessori voi yhden kellojakson aikana hakea sekä datamuistista että ohjelmamuistista, jolloin kerrottavien lukujen hakuaika pienenee puoleen. Erillisten muistien arkkitehtuuria kutsutaan *Harvard-arkkitehtuuriksi*, koska rakenne esiteltiin ensimmäisen kerran Harvardin yliopistossa Kaliforniassa.

Viime vuosikymmeninä signaaliprosessoreissa huomio on kiinnittynyt entistä enemmän signaaliprosessoreiden mobiilikäyttöön. Prosessoreiden täytyy olla pieniä ja kevyitä, ja niiden täytyy tuottaa vähän lämpöä ja kuluttaa vähän virtaa. Osittain tästä syystä sig-

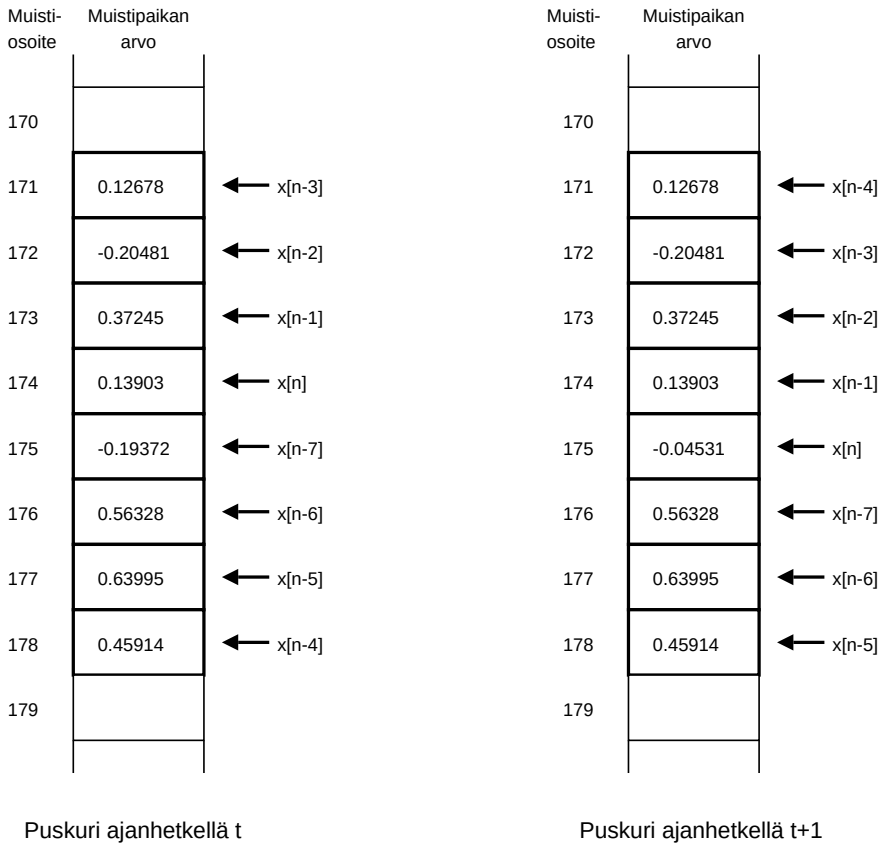
naaliprosessoreista on tullut hyvin halpoja yleiskäyttöisiin prosessoreihin nähden[1]. Halpaan hintaan vaikuttaa myös se, että tiettyä tehtävää varten valmistaja valitsee halvimman ja yksinkertaisimman tehtävään sopivan prosessorityypin, eikä tehokkainta markkinoilta löytyvää mallia. Pöytäkoneissahan tehoa ei koskaan ole liikaa, koska maksimitehoa tarvitaan aina jossain tilanteessa. Signaaliprosessorissa käsiteltävää dataa tulee sen sijaan yleensä vakiotahdilla, joten ylitehokkaan prosessorin käytössä ei ole järkeä.

8.1 Circular buffering

Signaalinkäsittelyä käyttävissä laitteissa prosessointi on yleensä siis reaaliaikaista. Toisin sanoen ulostulosignaali tuotetaan samaan aikaan kuin sisääntulosignaali saapuu laitteeseen. Ulostulo on siis saatava tuotettua lähes välittömästi herätteen tultua laitteeseen. Esimerkiksi puhelimessa viiveen tulee olla alle 10 millisekuntia, jotta puhuja tai kuuntelija eivät huomaa viivettä.

Toteutettaessa reaaliaikainen FIR-suodatus, muistissa täytyy olla kerrallaan yhtä monta signaalin näytettä kuin on suotimen kertoimien määrä. Näitä näytteitä päivitetään jatkuvasti kun uusia näytteitä saapuu. Koska näytteenottotaajuudet voivat olla suuria (esim. 44.1 kHz), tarvitaan tehokas rakenne näytteiden tallettamista ja päivittämistä varten.

Yleisesti käytössä on niin sanottu *circular buffering* -rakenne. Periaate näkyy alla olevasta kuvasta, jossa on esitetty kahdeksan näytteen puskurirakenne. Puskurin sisältö on esitetty kahdella peräkkäisellä ajanhetkellä.



[1]Kirjoitushetkellä tyypillinen listahinta on parinkymmenen dollarin luokkaa.

Rakenteen ideana on, että puskurin ensimmäisen muistipaikan ajatellaan olevan kiinnitetyn puskurin viimeiseen muistipaikkaan. Toisin sanoen kuvan esimerkissä muistipaikat 171 ja 178 ovat peräkkäisiä aivan kuten esimerkiksi muistipaikat 174 ja 175 ovat peräkkäisiä. Tietoa siitä, missä viimeisin näyte on, ylläpidetään osoittimen avulla. Kun saadaan uusi näyte, osoitin siirretään seuraavaan muistipaikkaan. Puskurirakenteen tehokkuus perustuu siihen, että vain yhtä näytettä tarvitsee päivittää uuden näytteen saapuessa.

Esimerkki: Tarkastellaan FIR-suodatuksen toteuttamista circular buffering -rakennetta käyttäen.

1. Lue arvo A/D-muuntimelta ja generoi keskeytys;
2. Käsittele keskeytys ja siirrä signaalin näyte sisääntulosignaalin puskuriin;
3. Päivitä puskurin osoitinta;
4. Käy läpi kaikki suodattimen kertoimet:
   (a) lue kertoimen arvo muistista
   (b) päivitä kertoimiin osoittavan osoittimen arvo seuraavaan kertoimeen
   (c) lue näytteen arvo puskurista
   (d) päivitä näytteisiin osoittavan osoittimen arvo seuraavaan näytteeseen
   (e) kerro näytteen arvo kertoimen arvolla
   (f) lisää tulos summa-rekisteriin (engl. *accumulator*)
   (g) toista kunnes kaikki suotimen kertoimet on käyty läpi
5. Siirrä tulos ulostulorekisteriin ja edelleen D/A-muuntimeen.

Signaaliprosessoreissa on olennaista, että esimerkin algoritmissa toistuvat operaatiot pystytään toteuttamaan nopeasti. Tärkeitä ovat erityisesti kohtaan 4 kuuluvat vaiheet, eli muistista lukeminen, osoittimen päivittäminen, lukujen kertominen ja yhteenlasku. Useissa prosessoreissa arkkitehtuuri onkin suunniteltu niin, että kaikki nämä toiminnot saadaan suoritettua yhden kellojakson aikana eli käytännössä siis yhdellä konekäskyllä. Tämä konekäsky on nimeltään MAC-operaatio (Multiply and ACcumulate), ja se kertoo kaksi lukua keskenään ja lisää tuloksen summamuuttujaan yhden kellojakson aikana. Perinteissä mikroprosessoreissa jokainen vaihe vaatisi oman käskynsä. Nykyään MAC-operaatioita voidaan suorittaa useampia yhdessä kellojaksossa, jolloin suodatus nopeutuu entisestään.

Vastaavanlaisia operaatioita on viime aikoina tullut myös yleiskäyttöisiin prosessoreihin. Intel esitteli vuonna 1997 laajennetun käskykannan MMX-prosessorit, joissa on MAC-operaatiota vastaava komento kokonaisluvuille. Lisäksi MMX-laajennuksen avulla yhden kellojakson aikana voidaan suorittaa useita kokonaislukukertolaskuja. Sittemmin Intel on laajentanut käskykantaa myös liukuluvuille SSE, SSE2 ja SSE3 -laajennusten myötä. Myös muilla prosessorivalmistajilla on nykyisin vastaavia tekniikoita käytössä.

8.2 Kiinteän vai liukuvan pilkun aritmetiikka?

Digitaalinen signaaliprosessori voidaan toteuttaa käyttäen joko kiinteän pilkun (engl. *fixed point*) tai liukuvan pilkun (engl. *floating point*) aritmetiikkaa. Molempia esitystapoja käyttäviä prosessoreja on yleisesti saatavilla.

Perinteisesti liukulukulaskenta on ollut hitaampaa kuin kokonaislukulaskenta, mutta nykyaikaisissa signaaliprosessoreissa ero on merkityksetön. Niinpä kiinteän pilkun aritmetiikan käytöstä saatava merkittävin hyöty on laitteen halvempi hinta sekä pienempi koko ja virrankulutus. Liukuvan pilkun järjestelmien etuja ovat laskennan suurempi tarkkuus ja suurempi käytössä oleva lukualue. Lisäksi näiden laitteiden ohjelmointi on helpompaa, sillä ohjelmoijan ei yleensä tarvitse huolehtia ylivuotojen ja alivuotojen tai pyöristysvirheiden välttämisestä.

Usein kiinteän pilkun aritmetiikkaa kannattaa kuitenkin käyttää. Jos esimerkiksi A/D-muunnoksessa käytetään joka tapauksessa pientä bittimäärää, ei ole välttämättä järkevää käyttää liukulukuja. Esimerkiksi usein videosignaali esitetään kahdeksalla bitillä. Vastaavasti jos toteutettavat algoritmit ovat melko yksinkertaisia, kannattaa harkita kiinteän pilkun aritmetiikkaa. Jos taas algoritmit ovat monimutkaisia, ne on paljon helpompi toteuttaa liukuvan pilkun aritmetiikkaa käyttäen. Yleisesti taajuustason algoritmit (kuten FFT) ovat melko monimutkaisia, ja ne kannattaa toteuttaa liukulukuja käyttäen.

Valintaan vaikuttaa useimmiten myös taloudellinen näkökohta. Käytettäessä kiinteän pilkun aritmetiikkaa, laitteen tuotantokustannukset saadaan painettua alas, mutta vastaavasti laitteen suunnittelukustannukset saattavat nousta korkeiksi. Liukuvan pilkun aritmetiikkaa käytettäessä tilanne on päinvastainen.

Myös prosessorin tehonkulutus on usein ratkaiseva valintaperuste. Monissa sovelluksissa laitteen on toimittava pitkiä aikoja samalla virtalähteellä (akut, paristot, tms.), jolloin tehon kulutus on ratkaiseva tekijä. Kiinteän pilkun prosessorien tehonkulutus on huomattavasti liukuvan pilkun prosessoreita vähäisempi. Tällaisissa sovelluksissa voi joskus jopa olla kannattavaa simuloida liukulukulaskentaa ohjelmallisesti kiinteän pilkun prosessorilla.

8.3 C:llä vai konekielellä?

Yleisesti signaaliprosessoreja ohjelmoidaan joko C-kielellä tai suoraan symbolisella konekielellä (assembler). Nyrkkisääntönä on, että C-kielellä ohjelmointi on helpompaa ja nopeampaa kuin konekielellä, kun taas taitava ohjelmoija saa tuotettua konekielellä nopeampaa koodia kuin C-kielellä.

Yleisprosessoreja ohjelmoitaessa assemblerin käyttö on nykyisin melko harvinaista. Signaaliprosessoreissa sen käyttö on kuitenkin edelleen hyvin yleistä, koska tyypillisesti ohjelmat ovat lyhyitä (helppoja hallita), ja nopeus on kriittinen vaatimus. Kääntäjät eivät nimittäin ole täydellisiä, eivätkä ne välttämättä pysty hahmottamaan C-kielisen ohjelman perimmäistä tarkoitusta ja tehokkainta käännöstä. Esimerkiksi FIR-suodin toteutetaan C-kielellä for-silmukan sisällä. C-kielisessä koodissa ei kuitenkaan ole enää tietoa siitä, että kyseessä on konvoluutio, joka voidaan toteuttaa tehokkaasti MAC-operaatioiden avulla. Kääntäjä joutuukin tutkimaan silmukan alku- ja loppuehtoja sekä sen sisällä käytettävien muuttujien riippuvuuksia toisistaan. Aina kääntäjä ei löydä summausoperaatiota, joka oli-

si tehokas toteuttaa MAC-operaatiolla tai pysty löytämään riippumattomia lohkoja, jotka olisi mahdollista suorittaa rinnakkain.

Useat kääntäjät muuntavat C-koodin ensin Assembler-kieliseksi, josta on mahdollista tarkastaa kuinka kääntäjä tulkitsi C-kielisen koodin.

Harjoitustehtäviä

8.1. Toteutetaan reaaliaikainen FIR-suodatin TI:n DSP Starter Kitillä C-kielisenä rutiinina. Tutustutaan tässä tehtävässä kortin käyttöönottoon.

Alkuvalmistelut. Tarkista ensin kaikki kytkennät. Tietokoneen takana olevasta äänikortin vihreästä pistokkeesta kytketään johto DSP-kortin sisäänmenoon (kotelossa lukee IN, tms.) Kotelossa on myös toinen 3,5 mm:n pistoke (OUT), johon liitetään kaiuttimien johto. Näin ollen tietokoneen tuottama ääni menee DSP-kortille, joka käsittelee sen ja lähettää edelleen ulostulon kautta kaiuttimiin. Tarkista myös, että tietokoneen takaa USB-väylästä menee vaaleanharmaa johto kortille. Tarkista vielä, että kortti saa virtaa (vihreä ledi palaa). Käytä ohutta mustaa virtajohtoa varmuuden vuoksi irti, jotta kortti on perustilassa, eikä esimerkiksi edellisen ryhmän ohjelma jää sinne pyörimään.

Kortin toimintaa testataan jatkossa soittamalla Windows Media Playerillä jompi kumpi esimerkkikappaleista (tai nettiradio tms.). Myös Matlabissa on testisignaaleita (esimerkiksi komennot `load handel.mat; soundsc(y, 8192);`). Testisignaali prosessoinnin jälkeen pitäisi saada kuulumaan kaiuttimista. Tietokoneissa on sisäiset kaiuttimet, joten voit kuunnella alkuperäisen signaalin irrottamalla audiokaapelin koneen takaa. Tällöin kone käyttää sisäisiä kaiuttimia.

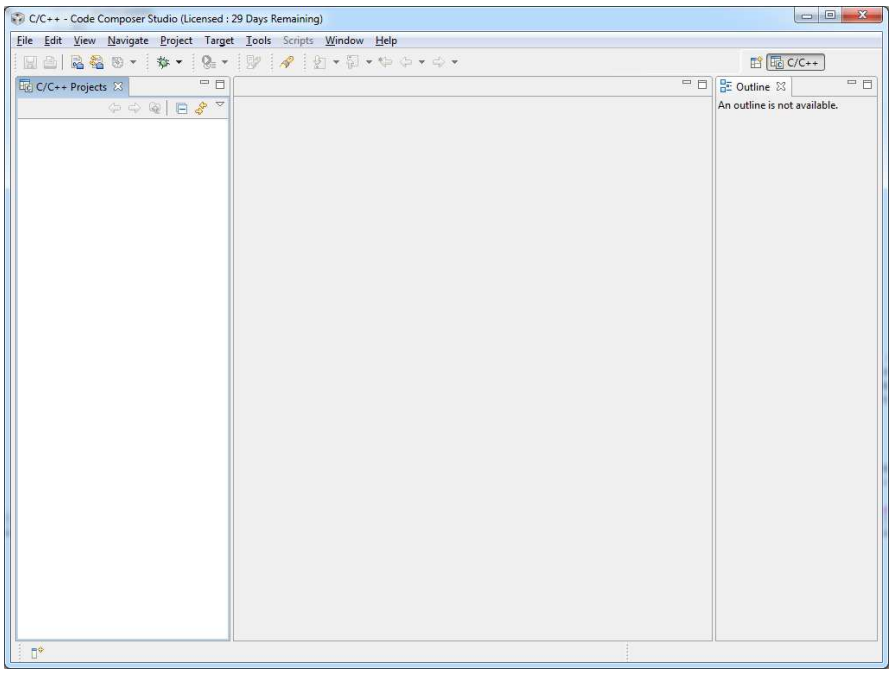
Ohjelmisto. Ohjelmointityökalu (Code Composer Studio 4.2; CCS) on asennettu, ja kuvake löytyy työpöydältä:

Ohjelman pohjana käytetään esimerkkiohjelmaa, joka löytyy osoitteesta

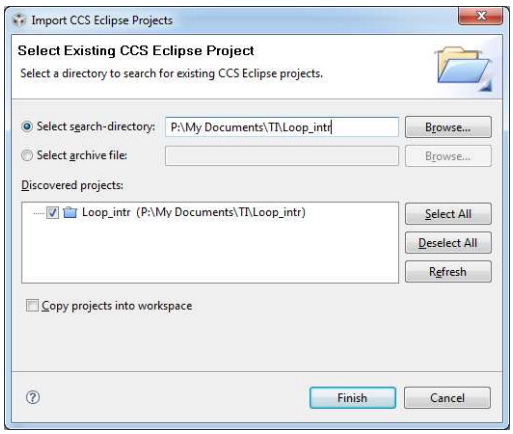
`http://www.cs.tut.fi/kurssit/SGN-1251/TI-tyo.zip`

Pura tiedosto esim. hakemistoon `P:\My Documents\TI\`.

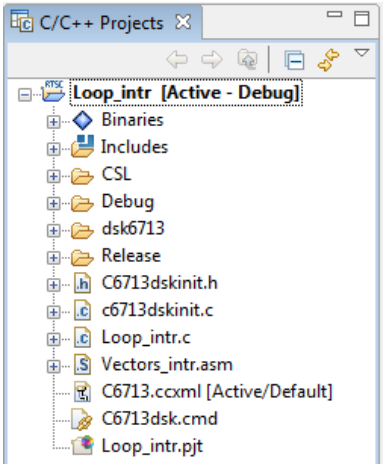
Laitteiston testaus. Käynnistä Code Composer Studio. Ensimmäisellä käynnistyskerralla ohjelma kysyy oletushakemistoa uusille projekteille, mihin kannattaa antaa oma kotihakemistosi tai sen alikansio (esim. `P:\My Documents\TI\`). Tämän jälkeen aukeaa musta Welcome-screen, jonka oikeassa ylälaidassa on CCS:n ikoni ja teksti "start using CCS". Klikkaa ikonia ja alla olevan kuvan mukainen työpöytänäkymä aukeaa.



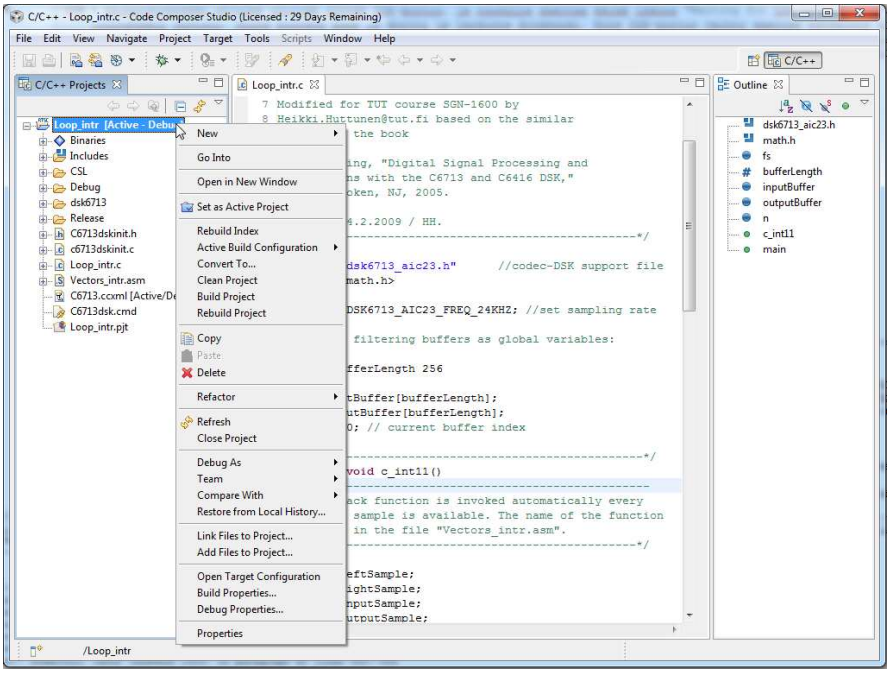
Avaa seuraavaksi lataamasi ja purkamasi projektipohja jonka purit hakemistoon `P:\My Documents\TI\`. Tämä onnistuu valikoista: Project → Import Existing CCS/CCE Eclipse Project. Aukeaa dialogi, jolle täytyy antaa hakemisto, josta projekti löytyy (siis `P:\My Documents\TI\Loop_Intr\` kohtaan Select search-directory). Jos kaikki meni oikein, CCS löytää valmiin projektipohjan ja dialogi näyttää seuraavalta.



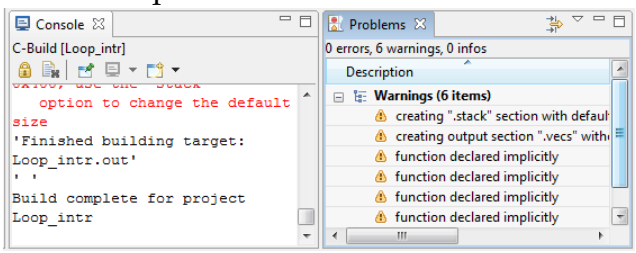
Ikkunan vasemmassa laidassa on tämän jälkeen alla olevan kuvan mukainen puu. Muokattava C-ohjelmakoodi on nyt tiedostossa `Loop_intr.c`. Avaa tiedosto ja tarkista että sieltä löytyy funktio `interrupt void c_int11()`. Laite kutsuu tätä funktiota joka kerta kun uusi näyte saapuu A/D-muuntimelta, eli 24000 kertaa sekunnissa (oletusasetuksilla näytteenottotaajuus on 24 kHz).



Ohjelman raakaversio kopioi järjestelmään tulevan signaalin sellaisenaan ulostuloon, joten on hyvä testata sillä että kaikki toimii kuten pitääkin. Ohjelman voi kääntää esim. klikkaamalla hiiren oikealla napilla avointa projektia (`Loop_intr`), jolloin aukeaa alla olevan kuvan mukainen valikko.



Ohjelma käännetään ajettavaan muotoon valitsemalla yo. valikosta "Build Project"(tai näppäinyhdistelmällä Ctrl-B). Jos kaikki meni oikein, ruudun alalaidassa olevissa ikkunoissa pitäisi olla alla olevan mukainen näkymä.



Jos Problems-ikkunassa on virheitä (errors), tulee ne korjata ennen kuin ohjelma kääntyy. Ohjelma voidaan siirtää prosessorille ja laittaa ajoon klikkaamalla alla

olevan kuvan mukaista Debug-nappia ja sieltä valintaa Debug Active Project.

8.2. Jatkoa edelliseen tehtävään. Edellinen tehtävä oli testi, jolla testattiin laitteiston toimintaa. Seuraavaksi toteutetaan FIR-ylipäästösuodin muokkaamalla ohjelmakoodia tiedostossa `Loop_intr.c`. Avaa kyseinen tiedosto.

Tiedoston alussa on määrittely

```
interrupt void c_int11()
```

Tämä on funktio, jota järjestelmä kutsuu aina kun uusi näyte on valmis käsiteltäväksi (siis 24000 kertaa sekunnissa). Funktio saa uusimmat näytteet riveillä

```
leftSample = input_left_sample();
rightSample = input_right_sample();
```

Funktio pitää itse muistissaan aiemmin tulleet näytteet taulukossa

```
short inputBuffer[bufferLength];
```

Funktio tallentaa myös vanhat ulostulonäytteet, mutta niitä ei tässä tehtävässä käytetä (koska suodin of FIR eikä IIR).

Funktion keskivaiheilla on rivi:

```
outputSample = inputSample;
```

Tämän rivin tilalle tulee FIR-suotimen toteuttava ohjelmalohko, jonka ulostulo sijoitetaan muuttujaan `Loop_intr.c`. Ohjelmalohko on yksinkertainen for-silmukka, jonka sisällä lasketaan konvoluutiosumma. Konvoluutio lasketaan ylipäästösuotimen impulssivasteen kanssa, joka suunnitellaan Matlabissa.

Tee siis seuraavat vaiheet.

(a) Suunnittele Matlabissa FIR-ylipäästösuodin ikkunamenetelmällä eli komennolla `fir1`. Suotimen rajataajuus on 3000 Hz ja laitteen näytteenottotaajuus on 24000 Hertsiä. Suotimessa tulee olla 31 kerrointa. Ennen kertoimien laskentaa anna komento `format long`, jotta tarkkuus olisi riittävä.

(b) Kopioi kertoimet[2] tiedoston `Loop_intr.c` alkuun, jossa määrittelet taulukon

```
float h[] = {0.001201261290430, 0.002048894418557,... jne. };
```

(c) Poista yllä mainittu rivi (`outputSample = inputSample`), ja laita sen tilalle for-silmukka, esimerkiksi näin:

[2]Copy-paste operaatio voi helpottua tallentamalla kertoimet ensin tekstitiedostoon (`save -ascii`) ja kopioimalla ne Notepadista. Matlabissa on myös komennot `fopen`, `fprintf`, jne., jolla tiedoston saa muotoiltua miten tahansa.

```
sum = 0.0;                  /* Tyyppi float */
for (k = 0; k < N; k++)     /* indeksin k tyyppi int */
{
   sum = sum + jotain;
}

outputSample = (short) sum;
```

For-silmukan sisällä oleva termi "`jotain`" täytyy korvata konvoluutiosummassa olevalla tulolla. Konvoluution määritelmähän kuuluu 31 näytteen mittaisen FIR-suotimen tapauksessa seuraavasti:

$$y(n) = \sum_{k=0}^{30} h(k)x(n-k).$$

Huomaa kuitenkin, että muoto $x(n-k)$ ei kelpaa sellaisenaan, koska käytössä on circular buffering, ja indeksi voi mennä negatiiviseksi. Jos näin käy, oikea arvo löytyykin indeksillä `n - k + bufferLength`.

(d) Käännä ja aja ohjelma. Soita Windows Media Playerin esimerkkitiedosto. Kaiuttimista pitäisi nyt kuulua ylipäästösuodatettu versio kappaleesta.

Yleisiä ongelmia

- Kirjoita C-kieltä, älä C++:aa. Esim. muuttujat saa määritellä vain lohkon alussa, s.o. aaltosulun jälkeen.
- Muista joka kerralla kääntää ja ladata ohjelma uudelleen laudalle.
- Indeksointi for-silmukan sisällä menee helposti väärin. Muista, että C-kielen taulukon indeksointi alkaa nollasta eikä ykkösestä kuten Matlabissa. Voit siis esim. tässä tapauksessa viitata taulukkoon `inputBuffer` indekseillä $0, 1, \ldots, 255$. Jos viittaat negatiivisella indeksillä, ohjelma luultavasti silti kääntyy ja toimii. Virhe ilmenee ainoastaan esim. napsahduksina kun ulostuloon pääsee vääriä arvoja silloin tällöin.
- Jos kertoimet ovat väärin, voi tuloksena olla ylivuoto.

Luku 9

Epälineaarinen suodatus

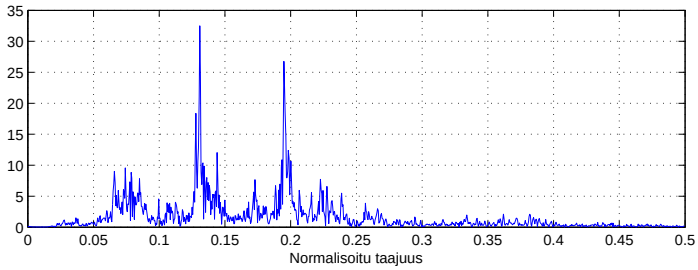
Tähänastinen kurssi on keskittynyt lineaarisiin suodatusmenetelmiin, eli sellaisiin suotimiin $\mathcal{F}(x(n))$, joille on voimassa

$$\mathcal{F}(ax(n) + by(n)) = a\mathcal{F}(x(n)) + b\mathcal{F}(y(n)),$$

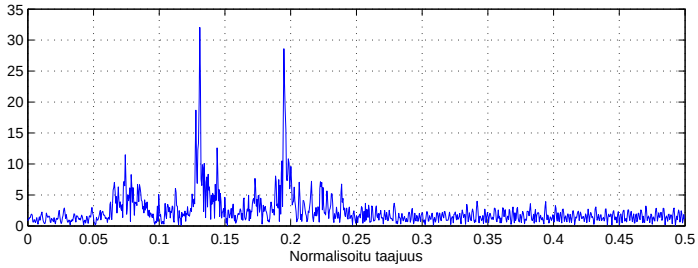
kaikille reaaliluvuille a ja b ja kaikille signaaleille $x(n)$ ja $y(n)$. Myös sovellukset käyttävät pääosin lineaarisia suotimia. Tästä huolimatta herää kysymys siitä, mitä tapahtuisi jos yllä olevasta rajoituksesta luovuttaisiin. Ilmeisesti tällöin menetettäisiin mahdollisuus käyttää lineaaristen suodinten tehokasta suunnittelukoneistoa, mutta mitä saataisiin tilalle? Näihin kysymyksiin tutustutaan seuraavassa.

Lineaarisia suotimia suunniteltaessa taustalla on ollut niiden taajuustulkinta. Esimerkiksi alipäästösuotimen tärkein ominaisuus on pienten taajuuksien säilyminen ja suurten taajuuksien poistuminen. Suotimen toiminnan ymmärtämisen kannalta tämä on selvä etu. Toisaalta se myöskin rajoittaa suodinten toimintaa, sillä esimerkiksi samalla taajuuskaistalla olevia signaalia ja kohinaa ei voida erottaa toisistaan lineaarisin menetelmin.

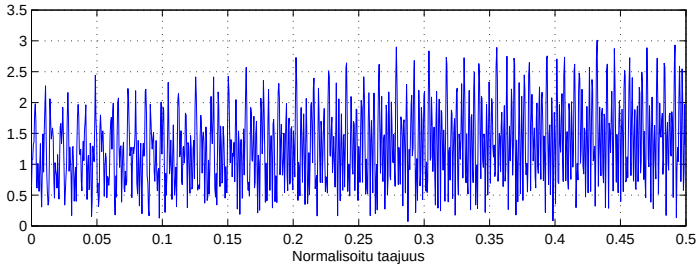
Tarkastellaan alla olevaa signaalin spektriä.



Kyseinen signaali on tallennettu 16 bitin tarkkuudella. Oletetaan, että se kulkee kohinaisen kanavan läpi ja että kukin bitti muuttuu todennäköisyydellä p. Alla on tulossignaalin spektri kun $p = 1/1000$. Tällöin keskimäärin joka tuhannes bitti on muuttunut.



Kohina ei ole kovinkaan selvästi havaittavissa yllä olevassa spektrissä, mutta jos tarkastellaan pelkän virhesignaalin spektriä, tulos on seuraavan näköinen.



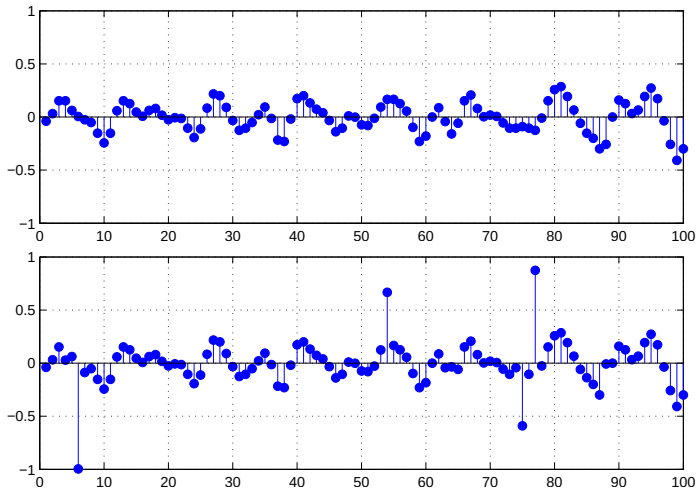
Kuviosta nähdään, että tarkasteltu kohinamalli aiheuttaa kohinaa kaikille taajuuksille. Näin ollen mikään lineaarinen suodatin ei voi poistaa kohinaa tehokkaasti.

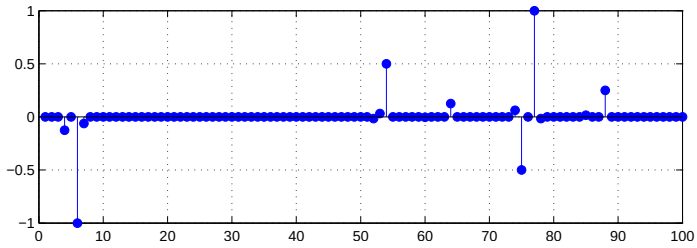
Edellä olleessa esimerkissä olleesta kohinasta käytetään nimitystä *bittivirhe*. Jos oletetaan, että bitit esittävät posiviivisia kokonaislukua (unsigned int), muun muassa seuraavat virheet ovat mahdollisia.

Lähetetty:	0011100001011100	Kymmenjärjestelmässä 14428
Vastaanotettu:	0011100001011101	Kymmenjärjestelmässä 14429
Vastaanotettu:	0011100000011100	Kymmenjärjestelmässä 14364
Vastaanotettu:	0011110001011100	Kymmenjärjestelmässä 15452
Vastaanotettu:	1011100001011100	Kymmenjärjestelmässä 47196

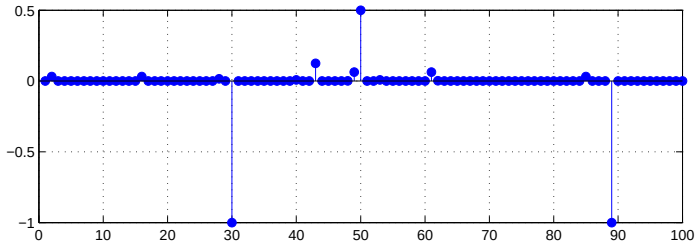
Jos bittivirhe muuttaa eniten merkitsevää (äärimmäisenä vasemmalla olevaa) bittiä, on näytearvon muutos $2^{15} = 32768$. Koska lineaariset suotimet perustuvat painotetun keskiarvon laskentaan, vaikuttaa tämä muutos suodatuksessa useisiin vasteen näytteisiin. Lineaarinen suodatin levittääkin bittivirhettä laajemmalle alueelle samalla pienentäen sen amplitudia.

Alla oleva kuvasarja havainnollistaa bittivirhettä aikatasossa. Ylin kuva on alkuperäinen signaali, seuraavassa on lisätty bittivirhettä (virheen todennäköisyys 0.01) ja kolmas kuva esittää näiden kahden erotusta, ts. varsinaista bittivirhettä.

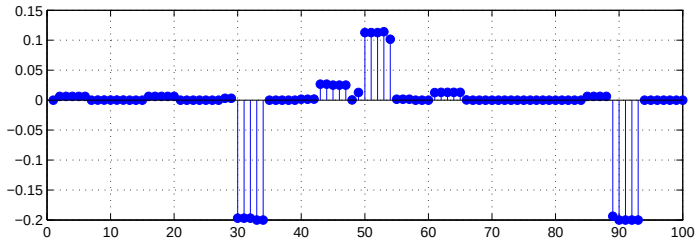




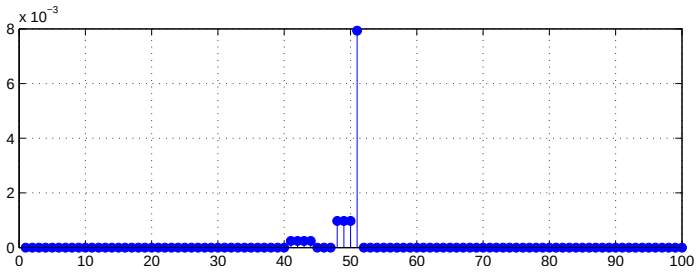
Koska bittivirhe ja varsinkin sen suodatus aikatasossa riippuu jossain määrin signaalista, on suodatusta usein tapana tarkastella vakiosignaalin yhteydessä. Alla olevassa kuvassa on bittivirhettä lisätty nollasignaaliin todennäköisyydellä 0.01.



Jos tämä suodatetaan viiden mittaisella FIR-suotimella tuloksena on alla olevan kuvan mukainen signaali. Suotimen kertoimet on valittu yhtä suuriksi siten, että niiden summa on yksi (valinta tehtiin näin paremman puutteessa; taajuuksiin perustuvasta suunnittelustahan ei ole mitään apua nyt). Kuvasta nähdään, että jokaisesta signaalissa olevasta impulssista on tullut viiden mittainen impulssi, jonka amplitudi on yksi viidesosa alkuperäisestä. Vakiosignaalin tapauksessa kohinan poistuminen ei ole kovinkaan hyvä.



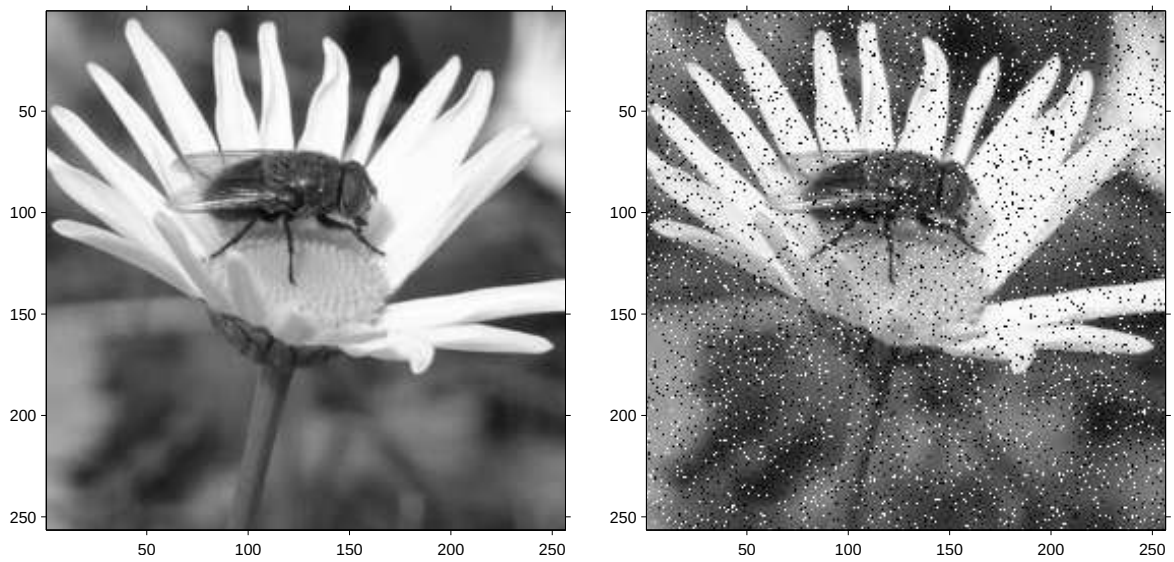
Jos kohinan tiedetään muodostuvan yksittäisistä impulsseista, halutaan suotimen olevan mahdollisimman immuuni suurimmille ja pienimmille arvoille ikkunan sisällä. Esimerkki tällaisesta suotimesta on yleisimmin käytetty epälineaarinen suodin: *mediaanisuodin*. Mediaanisuodin valitsee ulostuloksi suuruusjärjestyksessä keskimmäisen. Alla on suodatustulos mediaanisuotimella kun ikkunan pituus on viisi.



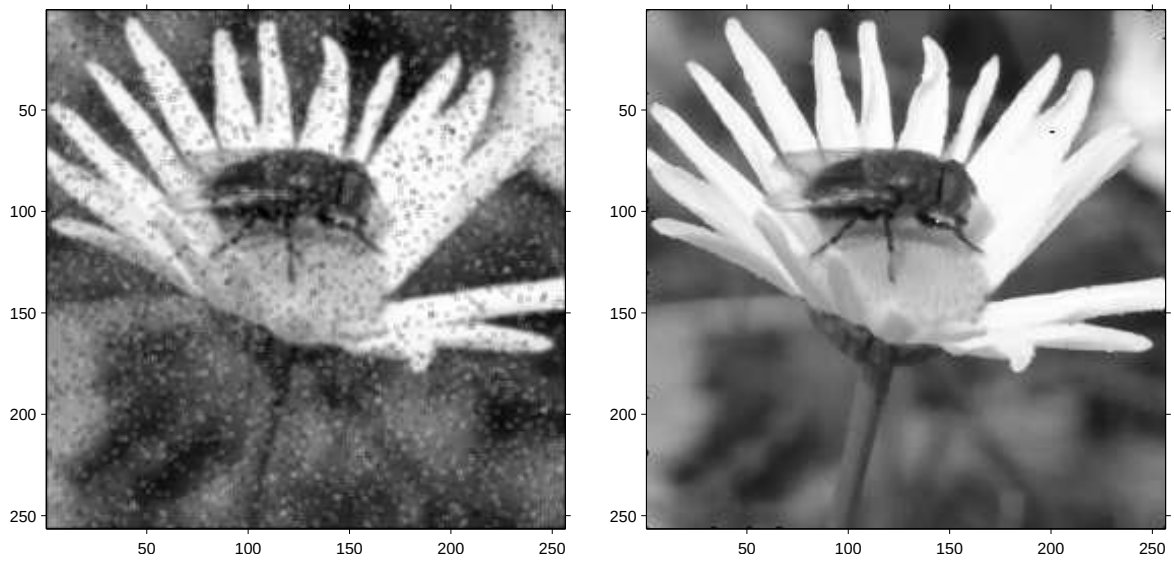
Kuvasta nähdään, että lähes kaikki impulssit poistuvat. Mediaanisuodin toimiikin erinomaisesti yksittäisten impulssien poistossa vakiosignaalista. Tästä ei toki voida vetää mitään johtopäätöksiä suotimen soveltuvuudesta muunlaisille kuin vakiosignaaleille. Lineaarisista suotimista poiketen mediaanilla ei ole varsinaista taajuustulkintaa, joten se sopii verraten huonosti esimerkiksi äänenkäsittelyyn ja muihin taajuuksien suodatukseen pe-

rustuviin sovelluksiin. Sen sijaan sitä käytetään jossain määrin mm. kuvissa esiintyvän kohinan poistoon.

Alla on esimerkki tilanteesta, jossa mediaanisuodin toimii erinomaisesti. Vasemmalla olevaan alkuperäiseen kuvaan lisätään keinotekoisesti ns. *salt & pepper*-tyyppistä kohinaa, jossa kukin pikseli muuttuu satunnaisesti täysin valkoiseksi tai mustaksi tietyllä todennäköisyydellä (tässä 4 % + 4 %).



Suodatettaessa kohinainen kuva 3×3-keskiarvosuotimella, jää jokaisesta impulssista kuvaan kopio suotimen impulssivasteesta (ks. kuva vasemmalla). Mediaanilla suodatettaessa kuvasta poistuvat miltei kaikki impulssit (ks. kuva oikealla). Kuvaan jää ainoastaan yksittäisiä impulsseja kohtiin, joissa alkuperäisessä kuvassa oli rykelmä samansuuntaisia impulsseja. Näistäkin päästäisiin eroon käyttämällä suurempaa ikkunaa. Tästä herääkin kysymys montako impulssia eri suotimet eri kokoisilla ikkunoilla "sietävät", ja voidaan määritellä robustisuuden käsite.



9.1 Robustisuus

Epälineaarisen suodatuksen yhteydessä puhutaan suotimen *robustisuudesta* (engl. *robustness*). Robustisuudella tarkoitetaan suotimen olevan epäherkkä herätteen muutoksille. Aiemmin olleen esimerkin perusteella voitaisiin väittää mediaanisuotimen olevan keskiarvosuodinta robustimpi. Tämä näkyy siinä, ettei bittivirheen lisääminen alkuperäiseen nollasignaaliin muuta mediaanisuotimen ulostuloa yhtä paljon kuin keskiarvosuotimen.

Useimmiten robustisuudesta puhutaan kvalitatiivisessa mielessä. Tämä tarkoittaa, että sanotaan esimerkiksi suotimen A olevan robustimpi kuin B. Suodinten suunnittelun ja analyysin kannalta olisi mielekkäämpää sanoa vaikkapa suotimen A olevan kaksi kertaa niin robusti kuin suodin B, tai jopa että suotimen A robustisuus on 0.4 kun taas suotimen B on 0.2. Tällaiseen käyttöön sopivia robustisuusmittoja on kehitelty ja eräs niistä on suotimen *murtumapiste*.

Suotimen murtumapiste (engl. *breakdown point*) on suurin määrä muutoksia ikkunan sisällä olevissa arvoissa, joka ei vielä voi saada ulostuloa selvästi alkuperäistä suuremmaksi tai pienemmäksi. Toinen tapa esittää asia on kysyä kuinka monta ikkunan sisällä olevaa arvoa voidaan korvata hyvin suurilla tai pienillä arvoilla, jotta ulostulo ei vielä muutu merkittävästi. Useimmiten "hyvin suuri arvo" ja "merkittävä muutos" tarkoittavat arvoa $\pm\infty$. Lisäksi tulos on tapana jakaa ikkunan alkioiden lukumäärällä.

Esimerkiksi viiden mittaisen mediaanisuotimen murtumapiste on $\frac{2}{5}$. Tämä lasketaan seuraavasti. Jos ikkunan sisällä on viisi äärellistä arvoa, joista yksi korvataan ∞:llä, mediaani on äärellinen (järjestetty vektori on muotoa $(*, *, *, *, \infty)$, missä '$*$' on jokin äärellinen arvo). Jos kaksi arvoa korvataan ∞:llä, mediaani on edelleen äärellinen (järjestetty vektori on muotoa $(*, *, *, \infty, \infty)$). Mutta jos kolme arvoa korvataan ∞:llä, mediaanista tulee ääretön (järjestetty vektori on muotoa $(*, *, \infty, \infty, \infty)$, josta valitaan keskimmäinen). Kaksi ääretöntä on siis suurin määrä, jolla ulostulosta ei tule ääretöntä. Tämä jaetaan ikkunan alkioiden määrällä 5, jolloin saadaan murtumapisteeksi $\frac{2}{5}$.

Vastaavalla tavalla havaitaan seitsemän mittaisen mediaanisuotimen murtumapisteeksi $\frac{3}{7}$, koska kolme ääretöntä ei vielä saa mediaania äärettömäksi, mutta neljä saa. Yleisemmin $2k+1$-pituisen mediaanisuotimen murtumapiste on $k/(2k+1)$, kun $k = 1, 2, 3, \ldots$.

Viiden mittaisen keskiarvosuotimen murtumapiste sen sijaan on nolla, sillä ensimmäinen ∞ saa keskiarvon äärettömäksi.[1]

9.2 Muutamia epälineaarisia suotimia

Tässä kappaleessa tarkastellaan lyhyesti muutamia yleisimmin käytettyjä epälineaarisia suotimia.

9.2.1 Mediaanisuodin

Aiemmin tutustuttiin jo mediaanisuotimeen. Täsmällinen määritelmä on seuraava.

N:n pisteen (merkitään $N = 2k+1$) mediaanisuotimen vaste herätteelle $x(n)$ on

$$y(n) = \mathrm{MED}\{x(n-k), x(n-k+1), \ldots, x(n-1), x(n), x(n+1), \ldots, x(n+k)\},$$

[1] Äärettömien kanssa laskemista on tässä yksinkertaistettu olettamalla laskusääntö $a + \infty = \infty$. Määritelmät voi toki tehdä täsmällisemminkin, mutta tällä kurssilla tyydytään yksinkertaisempaan käsittelyyn.

missä "MED" tarkoittaa annetun joukon mediaania, s.o. järjestyksessä keskimmäistä alkiota. Jos alkioita on parillinen määrä, MED antaa kahden järjestyksessä keskimmäisen alkion keskiarvon.

Esimerkiksi

$$\begin{aligned} \mathrm{MED}\{1,4,3,2,5\} &= 3, \\ \mathrm{MED}\{1,4,3,2\} = \frac{2+3}{2} &= 2.5. \end{aligned}$$

Mediaanin hyvänä ominaisuutena on voimakas robustisuus.

9.2.2 Painotettu mediaanisuodin

Mediaanisuodinta käytettäessä jokaisella ikkunan sisällä olevalla arvolla on sama paino. Jos tilannetta verrataan FIR-suodatukseen, vastaava tilanne tulisi vastaan kertoimien ollessa samat. Tämä rajoite vältetään tarkastelemalla *painotettua mediaanisuodinta*. N:n pisteen painotetun mediaanisuotimen vaste herätteelle $x(n)$ painoilla $(w_1, w_2, \ldots, w_N)$ on

$$\begin{aligned} y(n) = \mathrm{MED}\{&w_1 \diamondsuit x(n-k), w_2 \diamondsuit x(n-k+1), \ldots, w_k \diamondsuit x(n-1), w_{k+1} \diamondsuit x(n), \\ &w_{k+2} \diamondsuit x(n+1), \ldots, w_N \diamondsuit x(n+k)\}, \end{aligned}$$

missä merkintä $r \diamondsuit x$ tarkoittaa näytteen x monistamista r kertaa, esimerkiksi $3 \diamondsuit 2 = 2, 2, 2$.

Nyt on huomattava, että tavallinen joukon määritelmä ei enää riitä, vaan on käytettävä *multijoukkoa*, jossa toisto on sallittu. Tavallisella joukon käsitteellä joukot $\{2 \diamondsuit 3, 3 \diamondsuit 4\}$ ja $\{3, 4\}$ ovat samat, mutta multijoukkoa käytettäessä $\{2 \diamondsuit 3, 3 \diamondsuit 4\} = \{3, 3, 4, 4, 4\}$.

Olkoon ikkunan sisällä alkiot $(17, 18, 13, 25, 24)$. Painoilla $(1, 2, 3, 2, 1)$ ulostulo lasketaan seuraavasti. Ensin näytteet monistetaan painojen mukaisesti. Tuloksena saadaan multijoukko $\{1 \diamondsuit 17, 2 \diamondsuit 18, 3 \diamondsuit 13, 2 \diamondsuit 25, 1 \diamondsuit 24\} = \{17, 18, 18, 13, 13, 13, 25, 25, 24\}$. Tämän jälkeen joukko järjestetään ja saadaan $(13, 13, 13, 17, 18, 18, 24, 25, 25)$. Näistä valitaan keskimmäinen, joka on 18 (sattumoisin sama kuin mediaani).

Painoja käyttämällä on mahdollista muuttaa mediaanin ominaisuuksia. Muutos on aina vähemmän robustiin suuntaan, sillä voidaan osoittaa mediaanin murtumapisteen olevan suurin kaikista painotetuista mediaaneista. Näin ollen muut painotetut mediaanit muuttavat vähemmän alkuperäistä signaalia ja poistavat samalla vähemmän kohinaa.

Mediaani- ja keskiarvosuotimilla on mielenkiintoinen yhteys. Joukon $\{x_1, x_2, \ldots, x_N\}$ keskiarvo on se reaaliluku θ, joka minimoi summan

$$\sum_{k=1}^{N} (x_k - \theta)^2.$$

Joukon $\{x_1, x_2, \ldots, x_N\}$ mediaani puolestaan on se reaaliluku θ, joka minimoi summan

$$\sum_{k=1}^{N} |x_k - \theta|.$$

Näennäisistä eroavaisuuksistaan huolimatta näille voidaan siis löytää hyvin samankaltaiset esitykset. Näistä esityksistä on hyötyä teorian kannalta, sillä niiden avulla voidaan

osoittaa mediaanin olevan tilastollisesti paras suodin (vakiosignaalin tapauksessa), kun signaalissa oleva kohina noudattaa *Laplace-jakaumaa*, esimerkiksi

$$p(x) = e^{-2|x|}.$$

Keskiarvo puolestaan on paras suodin, kun signaalissa oleva kohina noudattaa *Gaussin jakaumaa*, esimerkiksi

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}.$$

Yllä olevia muotoja vastaavat esitykset löydetään painotetulle keskiarvosuotimelle (eli FIR-suotimelle) ja painotetulle mediaanille.

Joukon $\{x_1, x_2, \ldots, x_N\}$ painotettu keskiarvo painoilla $(w_1, w_2, \ldots, w_N)$ on se reaaliluku θ, joka minimoi summan

$$\sum_{k=1}^{N} w_k (x_k - \theta)^2.$$

Joukon $\{x_1, x_2, \ldots, x_N\}$ painotettu mediaani painoilla $(w_1, w_2, \ldots, w_N)$ on puolestaan se reaaliluku θ, joka minimoi summan

$$\sum_{k=1}^{N} w_k |x_k - \theta|.$$

Painotetun mediaanisuotimen murtumapiste saadaan lausekkeesta

$$\frac{1}{N} \max \left\{ m \in \mathbf{Z}_+ : \sum_{k=N-m+1}^{N} w_{(k)} < 0.5 \sum_{k=1}^{N} w_k \right\},$$

missä suotimen painot ovat $w_1, w_2, \ldots, w_N$. Lausekkeessa haetaan suurin sellainen kokonaisluku m, että m:n suurimman painon summa ei vielä ole yli puolet kaikkien painojen summasta.

9.2.3 L-suotimet

On olemassa myös muita ratkaisuja painokertoimien lisäämiseksi mediaaniin. Yksi tällainen ratkaisu muodostaa L-suodinten joukon.

Merkitään ikkunan alkioita vektorina $(x_1, x_2, \ldots, x_N)$ ja näistä järjestämällä muodostettua vektoria merkinnällä $(x_{(1)}, x_{(2)}, x_{(3)}, \ldots, x_{(N)})$, missä $x_{(1)} \leq x_{(2)} \leq x_{(3)} \leq \ldots \leq x_{(N)}$. Tällöin L-suodatuksen ulostulo painoilla $(w_1, w_2, \ldots, w_N)$ on

$$y = \sum_{k=1}^{N} w_k x_{(k)}.$$

Ilman järjestämistä L-suodin olisi tavallinen FIR-suodin.

Olkoon ikkunan sisällä alkiot $(17, 18, 13, 25, 24)$. Painoilla $(0.1, 0.2, 0.4, 0.2, 0.1)$ ulostulo lasketaan seuraavasti. Ensin näytteet järjestetään: $(13, 17, 18, 24, 25)$, ja järjestetyn vektorin näytteet kerrotaan vastaavilla painoilla ja summataan:

$$0.1 \cdot 13 + 0.2 \cdot 17 + 0.4 \cdot 18 + 0.2 \cdot 24 + 0.1 \cdot 25 = 19.2.$$

Valitsemalla sopivat painot L-suotimesta saadaan keskiarvosuodin, mediaanisuodin tai ns. α-viritetty-suodin (engl. α-trimmed mean). Se on L-suodin, jonka painot ovat

$$w_k = \begin{cases} \frac{1}{(1-2\alpha)N}, & \text{kun } \alpha N + 1 \leq k \leq (1-\alpha)N, \\ 0, & \text{muulloin.} \end{cases}$$

Hankalan näköisistä lausekkeista huolimatta α-suotimen idea on yksinkertainen. Ikkunasta poistetaan tietty määrä suurimpia ja pienimpiä alkioita ja lopuista otetaan keskiarvo. Esimerkiksi seitsemän pisteen α-viritetty-suodin parametrin α arvolla $\frac{2}{7}$ on L-suodin kertoimilla $(0, 0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, 0)$. Nyt nimittäin painot saadaan lausekkeesta

$$w_k = \begin{cases} \frac{1}{(1-2\cdot\frac{2}{7})7}, & \text{kun } \frac{2}{7}\cdot 7 + 1 \leq k \leq (1-\frac{2}{7})7, \\ 0, & \text{muulloin,} \end{cases}$$

eli sievennettynä

$$w_k = \begin{cases} \frac{1}{3}, & \text{kun } 3 \leq k \leq 5, \\ 0, & \text{muulloin.} \end{cases}$$

Parametri α määrää siis poistettavien näytteiden osuuden ikkunan kaikista näytteistä. Mitä suurempi sen arvo on, sitä enemmän suuria ja pieniä arvoja poistetaan ja sitä lähempänä tulos on mediaanisuodinta.

Koska L-suotimen ulostulo lasketaan sisääntulevien arvojen lineaarikombinaationa (eli kertolaskulla ja yhteenlaskulla), tulee sen murtumapisteeksi sama kuin FIR-suotimella eli nolla. Poikkeuksena ovat ne L-suotimet, joiden ensimmäinen ja viimeinen kerroin (w_1 ja w_N) ovat nollia (ks. harjoitus 9.2).

Harjoitustehtäviä

9.1. *(Matlab)* Lataa Matlabiin kuva `cameraman` komennolla

```
X = imread('cameraman.tif');
```

Saat kuvan ruudulle komennolla `imagesc(X)`. Lisää nyt kuvaan keinotekoista kohinaa komennolla `imnoise`. Valitse kohinamalliksi 'salt & pepper' parametrillä $D = 0.06$. Katso miltä tulos näyttää ja suodata se kaksiulotteisella 5×5-FIR-suotimella (`filter2`), jonka kertoimet ovat yhtäsuuret (kerroinmatriisi `M=ones(5,5)/25;`). Vertaa tulosta 5×5-mediaanisuotimella saatavaan tulokseen (`medfilt2`).

9.2. Laske yhdeksän pisteen α-viritetyn suotimen murtumapiste kun $\alpha = \frac{3}{9}$.

9.3. Tarkastellaan suodatustilannetta, jossa ikkunan sisällä on näytteet $x_1, x_2, \ldots, x_N$ (N on positiivinen kokonaisluku). Osoita, että lauseke

$$\sum_{k=1}^{N} (x_k - \theta)^2$$

saa pienimmän arvonsa kun parametri θ saa arvon

$$\theta = \frac{1}{N}\sum_{k=1}^{N} x_k.$$

Vihje. Etsi derivaatan nollakohdat.

9.4. *(Matlab)* Outlierien poisto äänisignaalista on haasteellisempaa kuin kuvasta, koska epälineaaristen suodinten vaikutus taajuusspektriin on vaikeasti ennustettavissa. Pelkkä outlier-arvojen poisto tapahtuukin yleensä turvallisimmin tunnistamalla niiden sijainti jollain tilastollisella testillä, ja suodattamalla vain nämä kohdat esimerkiksi mediaanisuotimella.

Toteutetaan tässä tehtävässä Matlabilla yksinkertainen outlier-tunnistin. Lataa esimerkkisignaali osoitteesta

`http://www.cs.tut.fi/kurssit/SGN-1251/scratch.mat`

Muuttujassa `x` on rahinainen versio Matlabin Handel-testisignaalista. Toteuta yksinkertainen outlier-tunnistin, jossa piste $x(n)$ tulkitaan poikkeavaksi arvoksi jos

$$\frac{|x(n) - m|}{s} > 2, \tag{9.1}$$

missä m on koko signaalin keskiarvo (komento `mean`) ja s sen keskihajonta (komento `std`)[2]. Korvaa kaikki näin tunnistetut outlierit viiden peräkkäisen näytteen mediaanilla for-silmukassa. Kuuntele tulos ja vertaa sitä rahinaiseen.

9.5. *(Matlab)* Edellisen tehtävän kaltaisissa *ilmaisuongelmissa* tunnistimen hyvyyttä voidaan arvioida sen mukaan montako tapausta tunnistettiin oikein (true positive rate; TPR) ja toisaalta montako väärää osumaa tuli (false positive rate; FPR). Täsmällisemmin nämä määritellään seuraavasti:

$$\begin{aligned} \text{TPR} &= \frac{\text{tunnistetut todelliset outlierit}}{\text{kaikki todelliset outlierit}} \\ \text{FPR} &= \frac{\text{virheellisesti tunnistuneet ei-outlierit}}{\text{kaikki näytteet}} \end{aligned}$$

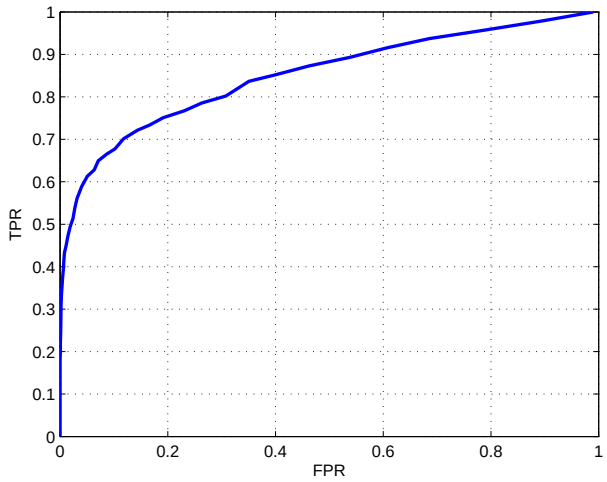
Nämä suureet selvästikin riippuvat toisistaan, koska mitä herkemmin menetelmä tunnistaa, sitä enemmän vääriä sekä oikeita tunnistuksia tulee.

(a) Laske TPR- ja FPR-arvot edellisen tehtävän tapaukselle. Tätä varten tarvitset outlierien todelliset sijainnit. Tässä tapauksessa häiriö oli itse lisättyä, ja outlierien sijainnit löytyvät tiedoston `scratch.mat` muuttujasta `e` (arvo `true` = outlier, arvo `false` = ei-outlier).

(b) Kokeile kuinka TPR ja FPR muuttuvat jos laitat kaavan (9.1) oikealla puolella olevan Hervanta-vakion 2 tilalle luvut 1.5 tai 2.5.

[2]Huomaa, että tulos olisi parempi jos koko signaalin keskiarvon ja hajonnan tilalla olisi paikallinen keskiarvo ja hajonta esim. 100:sta lähimmästä näytteestä

Kaikilla kynnyksen eri arvoilla saatavat FPR-TPR-kombinaatiot on kuvattu alla olevassa kuvassa. Esimerkiksi parametrin arvolla 1 saadaan TPR = 0.79 ja FPR = 0.26, mikä on yksi piste käyrällä. Kohdassa b saamasi tulosten pitäisi olla myös käyrällä. Tästä ns. *ROC-käyrästä* voidaan arvioida tunnistimen tehokkuutta: mitä suurempi käyrän alle jäävä pinta-ala, sen parempi.



Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland