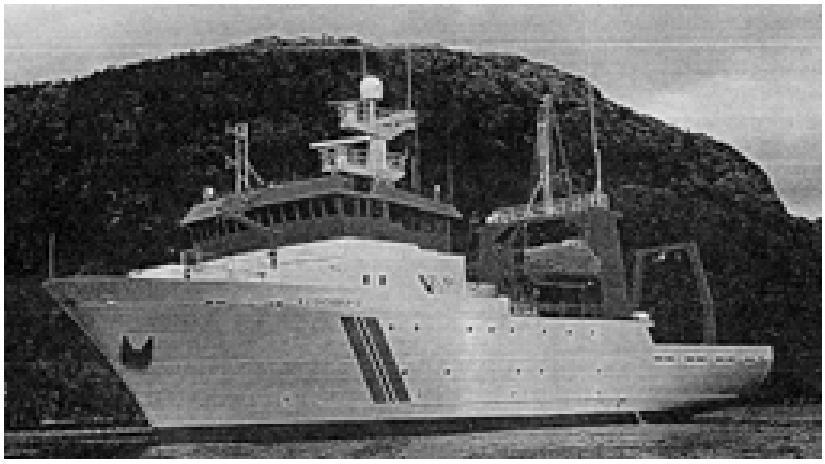
Adaptiv støykansellering av egenstøy på sonar

Jens-Kristian Haug

November 1995



Sammendrag

Denne hovedfagsoppgaven er basert på et reelt problem ved den parametriske sonaren på Forsvarets Forskningsinstitutts forskningsskip H. U. Sverdrup. Sonarens mottakende frekvensområde 500-5000Hz er overlagret med frekvenskomponenter fra skipets egenstøy. Problemet er av en slik art at frekvensområdet 500-1000Hz nå er filtrert bort før signalprosesseringen i sonaren. Det er ønskelig å redusere støykomponentene i dette frekvensområdet, fordi informasjonen som ligger der er spesielt interessant.

For å løse problemet er det tatt utgangspunkt i adaptive støykanselleringsteknikker. Utgangen av filteret i støykanselleringssystemet er et estimat av støykomponentene. Dette blir trukket fra sonarsignalet, og man får et forbedret sonarsignal med reduserte støykomponenter. 3 adaptive algoritmer basert på 2 filterstrukturer er valgt ut for evaluering på bakgrunn av et sett kriterier; minste midlere kvadrats metode og transversalt filter (LMS), rekursiv minste kvadrats metode og transversalt filter (RLS) og gradient adaptiv gitter metode og gitter-filter (GAL). For å få en grunnleggende forståelse av algoritmenes fordeler og ulemper, er utledningen av disse algoritmene vist.

En adaptiv støykansellerer er avhengig av godt samsvar (korrelasjon) mellom referansesignal og støy i primærsignal. Derfor er det utført målinger ombord på H. U. Sverdrup for å finne en plassering av referanseføleren som tilfredstiller et slikt krav. En plassering av referanseføleren ble funnet, der støykomponentene viste høy korrelasjon med støyen i sonarsignalet (primærsignalet). Opptak av signalene med tanke på senere simuleringer ble også gjort.

Filtersimuleringene med de tre algoritmene er gjort på kunstige og reelle data. Simuleringene på kunstige stasjonære data ble utført for å sammenligne algoritmene og bekrefte de teoretiske forhold fra litteraturen. Simuleringene ble utført med forskjellig filterorden og konvergeringsfaktor. Det ble vist at RLS hadde meget rask konvergensrate, og LMS og GAL algoritmene hadde omtrent like gode konvergensegenskaper. Med bakgrunn i littera-

turen kunne man forvente at GAL algoritmen ville konvergere raskere enn LMS algoritmen. Med det gitte støysignalet, en sinuskomponent, viste det seg imidlertid at LMS algoritmen hadde optimale konvergensegenskaper, tilnærmet lik egenskapene til GAL algoritmen.

Simuleringene med reelle data ble utført med signaler fra opptakene på H. U. Sverdrup. Det ble vist i simuleringene at forskjellene mellom algoritmene ble mindre ved reelle signaler. LMS algoritmen hadde like bra, om ikke bedre, egenskaper enn de andre algoritmene ved reelle ikke-stasjonære signaler. Alle algoritmenes støykansellering hadde en positiv effekt på sonarsignalet; støyen ble redusert og det ønskede signalet fra returnert bunnekko ble tydeligere.

Andre egenskaper, slik som kompleksitet og stabilitet ble diskutert, og LMS algoritmen ble foreslått valgt til videre arbeid på bakgrunn av denne vurderingen.

Til slutt er det gitt et forslag til implementering av det adaptive støykanselleringssystemet, med vekt på forståelse og vurdering av aktuelle digitale signalprosessorer. Det er vist at den adaptive støykanselleringsprosessen holder seg innenfor tiden bestemt av samplingsintervallet.

Forord

Denne hovedfagsoppgaven er skrevet av undertegnede for Forsvarets Forskningsinstitutt, avdeling for undervannsforsvar. Det er tatt utgangspunkt i et eksisterende problem, og denne hovedoppgaven består av teoretisk bakgrunn og analyse for implementering av et system som løser dette problemet.

Jeg har hatt særdeles god veiledning fra både ekstern veileder, forskningssjef Tor Knudsen ved FFI-U og intern veileder, professor Sverre Holm, Institutt for Informatikk, Universitetet i Oslo. Deres veiledning bidro til at jeg fikk en tidlig forståelse og oversikt over problemet, og jeg nøt godt av gode forklaringer på problem som dukket opp i skriveprosessen.

I oppgaven ble det gjort opptak av signaler fra sonar og vibrasjonsmåler. Jeg vil i den sammenheng takke Elling Tveit, forsker ved FFI-U, som var behjelpelig med innsamling av disse dataene.

I prosessen med korrekturlesning har jeg fått god hjelp av følgende personer; Kristin Kvernsveen, FFI-E og Ola Tørudbakken, SINTEF-SI. Jeg ønsker å takke disse for gode innspill på slutten av skriveprosessen.

Sist, men ikke minst, ønsker jeg å takke to som gjorde det mulig å gjennomføre denne oppgaven på tilmålt tid; min kone Helle Skaun Haug og min datter Mathilde. Jeg er umåtelig takknemlig for deres tålmodighet og forståelse i en hektisk tid.

Jens-Kristian Haug

I sandhed, en yderst begavet mand,
næsten alt, hvad han siger, går over ens forstand.

Henrik Ibsen; Peer Gynt, 4, handling.

[Denne siden er blank med hensikt]

Innhold

[Denne siden er blank med hensikt]

Kapittel 1

Problemdefinering og løsningsforslag

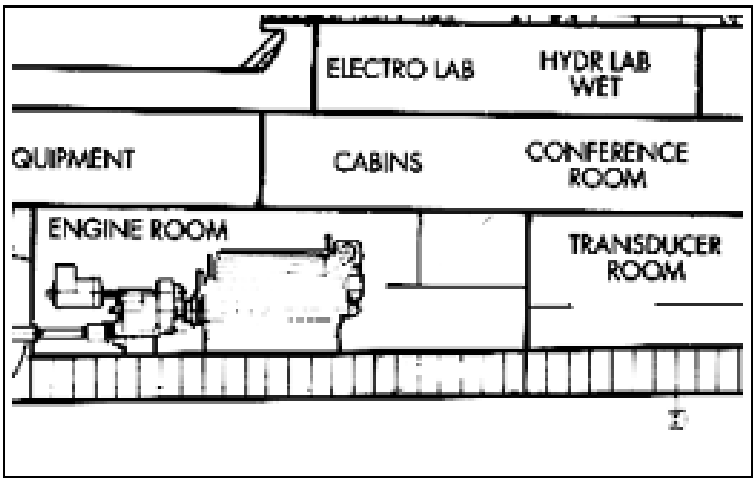
Avsnitt i dette kapittelet

Forsvarets Forskningsinstitutt's (FFI) forskningsskip "H. U. Sverdrup" er utstyrt med en parametrisk sonar til bruk for topografiske og seismiske undersøkelser. Denne sonaren har per idag ikke full utnyttelse av sitt mottakende frekvensområde, 500Hz til 5000Hz, fordi egenstøy fra skipet påvirker spesielt frekvensområdet 500 til 1000Hz. Problemet er av en slik art at man i dag har valgt å filtrere bort frekvensområdet mellom 500-1000Hz. Siden dette området inneholder viktig informasjon, jo lavere frekvens desto dypere bunn-penetrasjon, er det ønskelig å få redusert denne støypåvirkningen mest mulig. Denne oppgaven vil være grunnlaget for implementeringen av et system for å løse dette problemet.

Før forslag til problemløsning blir diskutert, vil det være nyttig å gi en kort forklaring av den parametrisk sonaren, samt å forklare begrepene egenstøy og vibrasjon.

1.0.1 Parametrisk sonar

En parametrisk sonar er en aktiv sonar som utnytter vannets ikke-lineære egenskaper til å generere en lavfrekvent lydkilde med smal strålebredde, uten å øke dimensjonene til transduseren. Prinsippet er at når to lydbølger med frekvens f_1 og f_2 med avstand Δf blir generert i den samme retningen, interfereres de og det blir dannet en sum og en differansefrekvens. Sumfrekvensen er av liten interesse, siden den raskt blir absorbert på grunn



Figur 1.1: Utsnitt fra skipet som viser transduserens plassering i forhold til maskin og gearboks. Transduserens plassering er vist nederst til høyre i figuren, gearboks er plassert til venstre i maskinrom.

av større absorbsjon ved høyere frekvenser. Differansefrekvensen derimot har en del verdifulle egenskaper som gjør den nyttig for praktiske applikasjoner i ekkolodd, seismikk og undervannskommunikasjon. Denne differansefrekvensen utgjør lydkilden i den parametriske sonaren, og har følgende egenskaper;

- Den har ingen sidelobeutstråling utenfor hovedstrålen. Dette har praktisk nytteverdi i for eksempel undervannskommunikasjon.
- Den har en smal og konstant strålebredde over et bredt bånd.
- Den har variabel båndbredde.
- En stor proporsjonal forandring i $(f_1 - f_2)$ kan oppnås ved en liten proporsjonal forandring i f_1, f_2 eller begge.
- Kavitasjon er ikke problem, fordi primærfrekvensene er høye. Kavitasjon oppstår når transduserens effektutsendelse skaper områder med negativt trykk i vannet rundt transduseren. Hulrom[1] med luft oppstår, kollapser kort tid etter, og forringer sonarsignalet.
- Ulempen er 40 til 70dB tap i differansefrekvenskilden sammenlignet med primærfrekvensene

Den parametriske sonaren på H. U. Sverdrup har to primærfrekvenser, den ene på 18kHz, og den andre har variabel frekvens slik at den sekundære frekvensen er fra 500 til 5000Hz. Strålebredden på primærfrekvensen er 3.5^o og på sekundærfrekvensen 5^o.

[1] engelsk: cavities

1.0.2 Egenstøy - hva og hvordan

Egenstøy er støy fra eget skip som påvirker hydrofonen. Støyen avhenger av hydrofonens direktivitet, plassering og montering. I hovedsak opererer man med tre typer egenstøy; hydrodynamisk støy, propellstøy og maskinstøy, og disse støybidragene når transduseren over svært varierte akustiske veier. I forhold til oppgavens problemstilling kan man si følgende for hvert av disse bidragene [Urick 83].

Hydrodynamisk støy

Dette er støy forårsaket av vannets flyt forbi skipets struktur og rundt sonardomen. Støyen har lavt nivå ved lave hastigheter, og øker sterkt ved høye hastigheter. Støyens bidrag i denne aktuelle situasjonen vil være liten, da bruk av sonaren skjer ved lave hastigheter.

Propellstøy

Propellstøy har sitt utspring i propellens kaviterende egenskap. Kavitasjon oppstår i dette tilfellet når propellens rotasjon skaper områder på propellen med negativt trykk. Propellstøyen påvirker sonaren gjennom vannet. Ved grunt vann kan også propellstøy reflekteres fra bunnen. Propellstøyen bidrar ikke mye til egenstøyen i denne situasjon fordi;

- Den har lavt nivå ved lave hastigheter
- Ved høy hastighet har den sitt viktigste område ved lave frekvenser, utenfor det aktuelle frekvensbånd 500-1000Hz.

I [Hovem 80] er det vist et forslag til filtrering av bredbåndet propellstøy på sonarsignalet.

Maskinstøy

Maskinstøy har sitt utspring i de utallige bevegelige deler i maskineriet. Man kan kort nevne turbinblader, sylindereksplosjon og geartenner som gir et linjekomponent frekvensbidrag og pumpe/ventil-kavitasjon og mekanisk friksjon som gir et kontinuerlig spekter bidrag.

I motsetning til det andre støyproduserende utstyret i maskinrom, er ikke gearboksen satt opp på støtdempende puter. Dette medvirker til at vibrasjonen fra tannhjulenes interaksjon forplanter seg i skipets skrog til sonarens opphengning. I figur 1.1 kan man få et inntrykk av mulige transmisjonsveier for gearboksens vibrasjoner. Tidligere undersøkelser ved FFI-U [Tveit 90] har vist at denne støypåvirkningen gir linjekomponenter i det aktuelle frekvensområdet, der den fundamentale frekvensen (førsteharmoniske) er relatert til antall tenner i kontakt per sekund, og i tillegg kommer overharmoniske av denne. Det er disse frekvensene som er hovedstøybidraget i det aktuelle frekvensområdet 500-1000Hz.

1.0.3 Hva er vibrasjon?

Vibrasjon er et fenomen som kan observeres som svingninger om et likevektspunkt. Vibrasjon skyldes overføring av energi i en struktur, som igjen er et resultat av påføring av krefter. Vibrasjon er ofte et biprodukt av en ellers nyttig operasjon, og den er i så måte veldig vanskelig å bli kvitt. Vibrasjonen kan observeres i følgende domener:

- Tidsdomenet, der det ses på forandring av amplituden med hensyn på tiden.

- Frekvensdomenet, beskrevet ved frekvensspekter. Frekvensen til bevegelsen er definert som antall bevegelsessykler i perioden ett sekund.

Disse to er relatert matematisk ved fouriertransformasjon. I det aktuelle tilfellet skyldes vibrasjonen bevegelser av komponenter i gearboksen. Dette frekvensbildet er komplisert på grunn av det store antall tannhjulkombinasjoner, men det vil senere bli vist hvordan tydelige linjekomponenter skiller seg ut i frekvensanalysen. Dette er frekvenser som sannsynligvis har sitt utspring i hovedtannhjulene i gearboksen.

Støyen har en komplisert gangvei mot transduseren, og flere strukturer kan underveis øve påvirkning på signalet. Strukturers egensvingninger kan være med på å forsterke og skape harmoniske svingninger.

1.1 Forslag til løsning av problem

For å løse problemet riktig må man kjenne den aktuelle situasjonen godt. Forskningsskipet H. U. Sverdrup blir operert på følgende måte; motoren kan kjøres på to omdreininger, 800 og 600 omdreininger i minuttet (RPM[2]), og skipets hastighet blir ut fra dette styrt ved å forandre propellens angrepsvinkel i forhold til vannet. Dermed står man igjen med to sett frekvenser fra gearboksen (fundamental og harmoniske) med utgangspunkt i henholdsvis 600 og 800 RPM, som påvirker transduseren og gir frekvenskomponenter i området 500-1000Hz. Foruten den innlysende problemløsningen å sette gearboksen på støtdempende oppheng, er dette en typisk filtreringsoppgave. Fordi man må forvente en stadig forandring i motorens omdreiningstall, er en vanlig filtreringsoppgave utelukket. Adaptive teknikker virker derimot mer attraktive. Adaptivt betyr tilpassende, og brukes i filtreringsøyemed om systemer som kan forandre filterkoeffisientene på bakgrunn av en referansekilde. I løsningen av dette problemet synes adaptive teknikker å være det riktige valg, og den korrekte betegnelsen på teknikken for den aktuelle omgivelsen er adaptiv støykansellering.

Denne oppgaven er grunnlaget for å løse et praktisk problem. Viktige spørsmål det søkes svar på i oppgaven er:

- Er det mulig å løse problemet med en adaptiv støykansellerer?
- Hvilken adaptiv filteralgoritme er best for å løse problemet?

Oppgavens form og utførelse er i sterk grad influert av disse punktene. I tillegg føles det naturlig å legge opp forslag til implementering av systemet.

Følgende fremgangsmåte ser jeg som nødvendig for å løse oppgavens problem;

- **Analyse av teorien for adaptiv støykansellering.** Flere adaptive algoritmer eksisterer, så det vil begrenses til et par sentrale. Det er nødvendig med en dyp forståelse av algoritmene, derfor er utledningene til algoritmene vist med stor nøyaktighet. Denne teorien danner grunnlaget for de senere simuleringer.
- **Måling for å finne riktig referanseplassering.** Dette er et viktig arbeid for å sikre god filtrering av støyen. Det vil derfor legges stor vekt på å finne riktig referanseplassering.
- **Test av algoritmer på syntetiske og oppsamlede data.** I disse kapitlene arbeides det med å finne svar på hovedspørsmålene i oppgaven. Simuleringene gir også god innsikt i egenskapene til de forskjellige algoritmene utledet i teoridelen.

[2]Rotations Per Minute

- **Forslag til konstruksjon av støykansellerer med nødvendig maskinvare.** Denne delen av oppgaven vektlegges mindre, men vil gi forslag til en implementering av et adaptivt støykanselleringssystem. Det gis særskilt oppmerksomhet til systemets sentrale regneenhet.

[Denne siden er blank med hensikt]

Kapittel 2

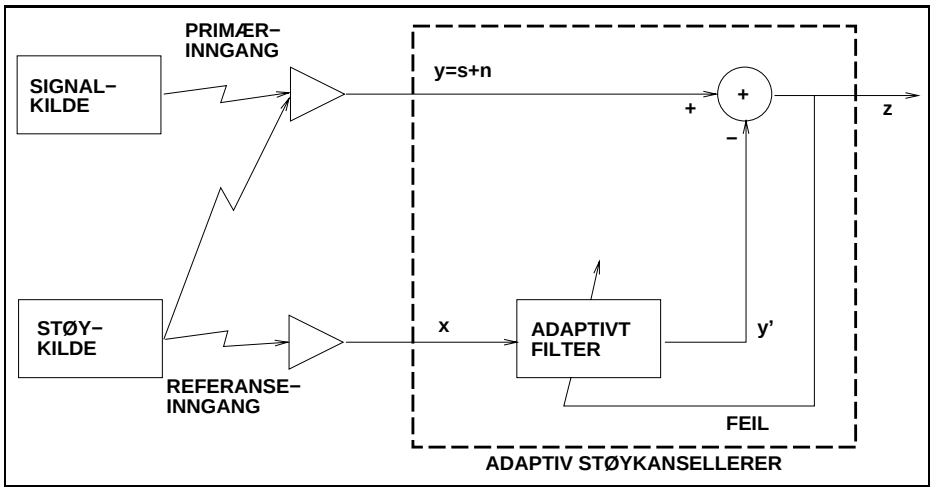
Algoritmer for adaptiv støykansellering

Avsnitt i dette kapittelet

Digitale filtre er i bruk for mange formål, blandt annet for å eliminere støy i spesielle frekvensbånd. De er karakterisert ved et sett filterparametre, som bestemmer filterets overføringsfunksjon. Vanlig filtrering er basert på forhåndskjennskap til både signal og støy. Adaptiv filtrering derimot, har evne til å justere sine parametre automatisk, og en trenger dermed ikke i utgangspunkt forhåndskjennskap til signal og støy. Adaptiv filtrering betyr et filter der filterkoeffisientene tilpasser seg (adapterer) i henhold til et referansesignal.
Man kan kort sette opp bruksområdene for et adaptivt støyfilter slik;

- når filterkoeffisientene må forandres for å tilpasse seg skiftende forhold.
- når det er spektralt overlapp mellom signal og støy.
- hvis støybåndet er ukjent eller varierer i tid.

Et adaptivt filter har følgende spesielle karakteristika.



Figur 2.1: Adaptiv støykansellerer

- Filterkoeffisientene kan variere med tiden.
- En adapsjonsalgoritme overvåker omgivelsene til filteret og sørger for at filteret tilpasser seg, slik at det er optimalt i den forstand at et feilmål minimaliseres.

I systemet for adaptiv støykansellering, vist i figur 2.1, er det to innganger. Primærinngangen består av ønsket signal og støy, $s + n$, mens referanseinngangen består av støy, x. Signalene n og x er korrelerte, det vil si samsvarende.

Målet er nå å redusere effekten av støyen i primærinngangen ved hjelp av referanseinngangen. Dette gjøres ved hjelp av et filter som produserer en utgang y', som blir et estimat av støyen n i primærsignalet. y' blir så subtrahert fra primærsignalet $s+n$, og vi får dermed en total utgang av systemet ved $z = s + n - y'$.

Det adaptive filteret justerer sin impulsrespons ved en algoritme som reagerer på et feilsignal fra systemets utgang. Dette feilsignalet er identisk med utgangen z.

2.0.1 Historie

Paul Lueg beskrev i 1936 de grunnleggende ideene for aktiv støykontroll. Han beskrev prinsippet for å måle et lydfelt med en mikrofon, manipulere signalet og mate dette inn i en sekundær kilde.

Det tidligste arbeidet med adaptiv støykansellering ble utført fra 1957 til 1960 av Howell og Appelbaum. De konstruerte og bygget et system for antenne sidelobekansellering. Widrow og Hoff utviklet i 1959 minste midlere kvadrat metoden, og brukte algoritmen i studiet rundt et mønstergjenkjenningsskjema. Andre personer arbeidet samtidig i USA, Sovjet-Unionen og Storbritannia med lignende prosjekter. En metode med nær forbindelse til minste midlere kvadrats metode er gradient adaptiv gitter metoden, beskrevet første gang i [Griffiths 78]. Forskjellen mellom disse ligger på det strukturelle plan, der den ene er basert på en transversal filterstruktur og den andre på en gitter filterstruktur.

I 60-årene ble arbeidene intensivert, og flere arbeider ble publisert om emnet adaptiv støykansellering. Bell Telephone Laboratories begynte tidlig med adaptiv ekkokansellering rundt 1965. Samtidig ble et adaptivt støykanselleringssystem konstruert og bygget ved

Stanford i 1965 av to studenter. De kansellerte 60Hz interferens på utgangen av en elektrokardiografisk forsterker og opptaker.

Siden det har adaptiv støykansellering blitt brukt i en rekke sammenhenger. De siste årenes utvikling av digitale signalprosessorer, har medført at dette bruksområdet har blitt utvidet til applikasjoner med krevende utregninger. Denne utviklingen på maskinfronten har også bidratt til en utvikling av raskere algoritmer.

2.0.2 Valg av filterstrukturer og algoritmer for uttesting i oppgaven

Etter å ha gjennomgått det foreliggende antall adaptive algoritmer, er det nødvendig å starte med en kort oversikt over disse. På bakgrunn av denne oversikten kan man da begrense antallet av aktuelle algoritmer for dette prosjektet. De to hovedfilterstrukturene for adaptive filtre er transversal (direkte), og gitter. I tillegg finnes blandt annet de mindre brukte rekursive filterstrukturer, ulineære filtre og nevrale nett, men disse vil ikke behandles i denne oppgaven. Man kan sette opp følgende adaptive algoritmer basert på transversal og gitter filterstruktur.

- **Minste midlere kvadrats metode** og transversalt filter. Dette er den klassiske algoritmen. Metoden har avveining mellom rask konvergens og liten feiljustering. Algoritmen er lite komplisert; $2N+1$ addisjoner/multiplikasjoner per iterasjon. Algoritmen er robust og numerisk stabil forutsatt riktige parameterverdier. [Haykin 91], [Haykin 84], [Honig & Messerschmitt 84], [Ifeachor & Jervis 93], [Kjølås 82], [Larssen 94], [Lebeda 90], [Treichler 87], [Widrow et. al. 75], [Widrow et. al. 76].

- **Rekursiv minste kvadrats metode** og transversalt filter. Dette er en variant av minste kvadrat algoritmen. Den konvergerer raskt, og har i teorien minimal feiljustering. Algoritmen er komplisert; N^2 addisjoner/multiplikasjoner per iterasjon. Spesielt bruksområde er talepredikskjon. Algoritmen er numerisk ustabil. [Cioffi & Kailath 84], [Haykin 91], [Haykin 84], [Honig & Messerschmitt 84], [Lebeda 90], [Treichler 87].

- **Rask minste kvadrats metode** og transversalt filter. Dette er også en variant av minste kvadrat metoden. Kompleksiteten er mindre enn ved rekursiv minste kvadrats metode, men algoritmen har problemer med numerisk ustabilitet. Algoritmen har hurtig konvergens. [Haykin 91] og [Lebeda 90].

- **QR dekomposisjon** og transversalt filter. Algoritmen er hurtig, men har relativt stor kompleksitet; $9N^2$ addisjoner/multiplikasjoner per iterasjon. Algoritmen er numerisk stabil. [Haykin 91] og [Lebeda 90].

- **Kompleks minste midlere kvadrats metode** og transversalt filter. Algoritmens kompleksitet er som ved minste midlere kvadrats metode, og konvergensegenskapene kan sammenlignes med rekursiv minste kvadrats metode. Algoritmen er stabil ved riktig parametervalg. [Haykin 91].

- **Kalman filtrering** og transversalt filter. Denne algoritmen er mye brukt innenfor fagfeltet kybernetikk. Algoritmen har samme konvergensegenskaper og kompleksitet som rekursiv minste kvadrats metode. Den er numerisk stabil. [Haykin 91] og [Lebeda 90].

- **Gradient adaptiv gitter** og gitter filterstruktur. Her kombineres den enkle minste midlere kvadrat metoden med stabilitets- og konvergensegenskapene ved gitter-filteret. Konvergensegenskapene er bedre enn ved transversal filterstruktur. Algoritmen er stabil med kompleksitet $14N$ addisjoner/multiplikasjoner per iterasjon. [Griffiths 78], [Wagner 90], [Haykin 91] og [Honig & Messerschmitt 84].

- **Minste kvadrat metoder** og gitter-filter. En rekke algoritmer som kombinerer stabiliteten i gitter-filteret med rask konvergens i minste kvadrat metoder. Algoritmene har gode konvergensegenskaper og er numerisk stabil. Kompleksitet er $14N$ addisjoner/multiplikasjoner per iterasjon. [Friedlander 82], [Haykin 91], [Ingebretsen 86] og [Lebeda 90].

For å begrense omfanget av arbeid i hovedoppgaven, velges 3 algoritmer ut fra oversikten. Valget tas på bakgrunn av følgende punkter, satt opp etter viktighetsgrad.

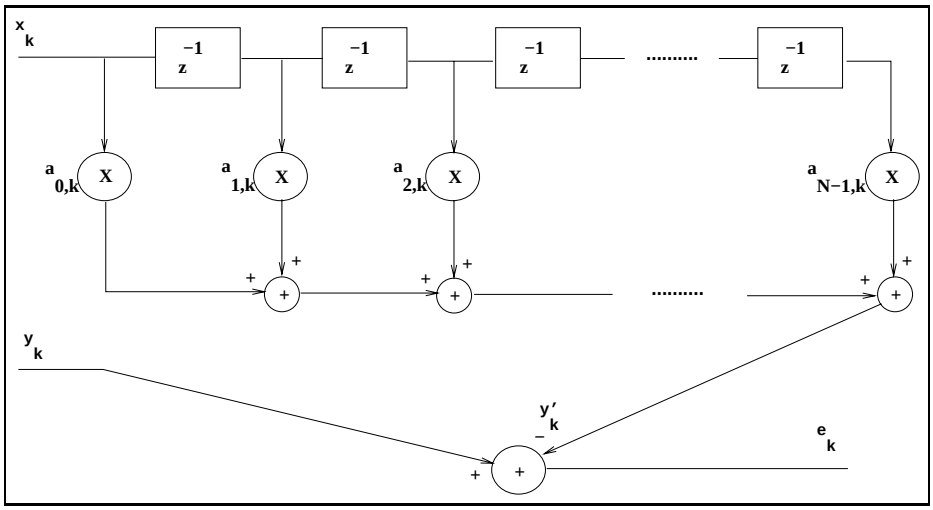
1. Er algoritmen tilpasset støykansellering? Er det erfaringer med slikt arbeid?
2. Algoritmen bør inneha gode stabilitetsegenskaper i et ikke-stasjonært miljø.
3. Utvalget skal representere en sammenligning av egenskapene til transversal og gitter filterstruktur.
4. Utvalget skal representere en sammenligning av egenskapene til minste midlere kvadrat metoden og minste kvadrat metoden.
5. Det er å foretrekke liten kompleksitet for å lette senere implementering.
6. Enkle og intuitive algoritmer.
7. Historisk interesse, stort erfaringsgrunnlag.

Adaptiv støykansellering er ikke så avhengig av rask konvergens som for eksempel ved talepredikskjon. Det betyr at alle algoritmene har konvergensegenskaper som tilfredstiller kravene ved adaptiv støykansellering. Dette er imidlertid et av denne oppgaves hovedspørsmål, som det senere skal finnes utdypende svar på. Punkt 1 tillater derfor å velge de enkleste algoritmene i oversikten. Punkt 2 er usikkert, og vil bli gjenstand for undersøkelse i denne oppgaven. Derfor vektlegges ikke dette foreløbig. For å tilfredstille punkt 3 og 4 føles det naturlig å velge følgende tre algoritmer;

- Minste midlere kvadrats metode. (LMS[1])
- Rekursiv minste kvadrats metode. (RLS[2])
- Gradient adaptiv gitter metode. (GAL[3])

De andre punktene synes da også å være oppfylt i stor grad. Det må understrekes at de andre metodene også kan være velegnet for adaptiv støykansellering, men utelates i oppgaven av plasshensyn.

[1] Least Mean Square
[2] Recursive Least Square
[3] Gradient Adaptive Lattice



Figur 2.2: Transversal filterstruktur

2.1 Algoritmer basert på transversalt filter

Den transversale filterstrukturen er den enkleste og mest brukte adaptive filterstruktur. Filterstrukturen er vist i figur 2.2. Dette er en FIR[4]-struktur. Strukturen består av en kjede av forsinkelsesledd, z^{-1}, som lagrer de N (filterorden) seneste inngangsverdiene x_k. Disse multipliseres med filterkoeffisientene, $a_{i,k}, i = 0, \ldots, N-1$, og summeres; [Ifeachor & Jervis 93]

$$y'_k = \sum_{i=0}^{N-1} a_{i,k} x_{k-i} \tag{2.1}$$

Går man tilbake til figur 2.1, ser man at feilsignalet, z ,brukes for styring av det adaptive filteret. Det er ønskelig å minimalisere feilsignalet, som heretter betegnes ε. Derfor kvadreres og midles feilsignalet over et ensemble av en realisasjon av prosessen, slik får man feilmålet, midlere kvadratisk feil (MSE[5]), også betegnet ξ.

$$\xi = E[\varepsilon^2]. \tag{2.2}$$

Adapsjonsalgoritmens formål er nå å minimere ξ.

Det er to hovedklasser adaptive algoritmer for den transversale filterstruktur, alt etter hvordan de minimerer ξ. Det er gradientbaserte algoritmer og minste kvadrat (LS[6]) algoritmer. Gradientbaserte algoritmer (steepest descent) estimerer gradienten til ξ.

$$\bigtriangledown \xi = \frac{\delta \xi}{\delta a_k} \tag{2.3}$$

Filterkoeffisientene innstilles slik at ξ beveger seg i den negative gradients retning. Den transversale filterstruktur sikrer at det eneste minimum for ξ er det globale, slik at ξ nærmer seg dette minimum.

[4]Finite Impulse Respons
[5]Mean Square Error
[6]Least Square

LS er en eksakt metode, i forhold til gradientalgoritmene. Den minimerer kostfunksjonen v_k, som består av summen av kvadratet av feilmålet, over vinduet W [Lebeda 90].

$$v_k = \sum_{i \in W} \lambda^{k-i} \varepsilon_i^2 \tag{2.4}$$

der $\lambda \leq 1$ er en vektfaktor. Denne minimeringen, som involverer tidsmidling, kan utføres eksakt med de data som er til rådighet, det vil si primærsignalet og referansesignalet. På grunn av dette konvergerer minste kvadrat metoder vanligvis hurtigere enn gradientalgoritmene.

2.1.1 Gradientbasert algoritme

Ut ifra figur 2.2 kan inngangsvektoren settes opp.

$\mathbf{x}_k = [x_k x_{k-1} \dots x_{k-N+1}]^T$,

der $[\cdots]^T$ betegner transponering av vektoren, og på samme måte kan man sette opp filterkoeffisientene

$\mathbf{a}_k = [a_{0,k} a_{1,k} \dots a_{N-1,k}]^T$

Summasjonen i filteret blir

$$y'_k = \sum_{i=0}^{N-1} a_{i,k} x_{k-i} = \mathbf{a}_k^T \cdot \mathbf{x}_k \tag{2.5}$$

Feilsignalet er definert som differansen mellom primærsignalet og utgangen av filteret, se figur 2.1.

$$\varepsilon_k = y_k - y'_k = y_k - \sum_{i=0}^{N-1} a_{i,k} x_{k-i} = y_k - \mathbf{a}_k^T \cdot \mathbf{x}_k \tag{2.6}$$

Man finner så et uttrykk for oppdatering av filterkoeffisientene. Kvadratet av feilen er gitt ved

$$\varepsilon_k^2 = (y_k - \mathbf{a}_k^T \cdot \mathbf{x_k})^2 = y_k^2 + \mathbf{a}_k^T \mathbf{x}_k \mathbf{x}_k^T \mathbf{a}_k - 2 y_k \mathbf{a}_k^T \mathbf{x}_k \tag{2.7}$$

Man finner nå middelkvadratfeilen ξ ved å ta forventningen på begge sider av ligning 2.7 under forutsetning av at $\mathbf{x}_k$ og y_k er stasjonære

$$\xi = E[\varepsilon_k^2] = E[y_k^2 + \mathbf{a}_k^T \mathbf{x}_k \mathbf{x}_k^T \mathbf{a}_k - 2 y_k \mathbf{a}_k^T \mathbf{x}_k] \tag{2.8}$$

$$\xi = E[y_k^2] + \mathbf{a}_k^T \mathbf{R} \mathbf{a}_k - 2 \mathbf{p} \mathbf{a}_k^T \tag{2.9}$$

der man har autokorrelasjonsmatrisen $\mathbf{R}$ og krysskorrelasjonsvektoren $\mathbf{p}$

$$\mathbf{R} = E[\mathbf{x}_k \mathbf{x}_k^T] = \begin{bmatrix} r_0 & r_1 & \cdots & r_{N-1} \\ r_1 & r_0 & \cdots & r_{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ r_{N-1} & r_{N-2} & \cdots & r_0 \end{bmatrix}, \mathbf{p} = E[y_k \mathbf{x}_k] = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{N-1} \end{bmatrix}$$

$\mathbf{R}$ inneholder de N første verdier av autokorrelasjonsfunksjonen for $\{x\}$,$r_0 \ldots r_{N-1}$. Tilsvarende inneholder $\mathbf{p}$ de N første verdier av krysskorrelasjonsfunksjonen mellom y_k og $\mathbf{x}_k$.

Man ser ut fra ligning 2.9 at ξ er en kvadratisk funksjon av filterkoeffisientene $\mathbf{a}_k$. Dette er en direkte følge av den transversale filterstruktur. Funksjonen ξ er en elliptisk hyperboliode, der middelkvadratfeilen endrer seg når filterkoeffisientene justeres. Flaten kalles prestasjonsflaten, og den har et globalt definert minimum. Oppgaven til adapsjonsalgoritmen er å styre mot dette minimum når man befinner seg på et hvilket som helst punkt på flaten. Dette gjøres ved å estimere gradientvektoren for de nåværende filterkoeffisientene i det punkt man står i og deretter endre filterkoeffisientvektoren i gradientens negative retning. Siden flaten er elliptisk og ikke sirkulær, er denne retningen ikke den direkte til dette minimumspunktet, men avvikelsen er av mindre betydning, selv om konvergenstiden blir påvirket.

Dette antyder dog ett av gradientalgoritmenes problemer: Konvergenstiden er avhengig av inngangssignalet. Når spekteret i inngangssignalet avviker meget fra et flatt, hvitt spektrum, spres egenverdiene for autokorrelasjonsmatrisen, og prestasjonsflaten blir elliptisk. Da har ikke gradienten retning mot minimumspunktet og filterkoeffisientene blir justert i en ikke optimal retning. Styrken av oppdateringen må da dempes og man får langsom konvergens [Lebeda 90]. I kapittelet som omhandler simulering på reelle data vil jeg vise hvordan man unngår dette ved å bruke en normalisert algoritme.

For å finne filterkoeffisientenes minimum, starter man med gradienten av feilfunksjonen ξ

$$\bigtriangledown \xi = \frac{\delta \xi}{\delta \mathbf{a}_k} = \frac{\delta(E[y_k^2])}{\delta \mathbf{a}_k} - 2\frac{\delta(\mathbf{p}\mathbf{a}_k^T)}{\delta \mathbf{a}_k} + \frac{\delta(\mathbf{a}_k^T \mathbf{R} \mathbf{a}_k)}{\delta \mathbf{a}_k} \tag{2.10}$$

Siden $E[y_k^2]$ er en konstant og ikke en funksjon av $\mathbf{a}_k$, så er gradienten av denne null. Videre har man at

$$\frac{\delta(\mathbf{p}\mathbf{a}_k^T)}{\delta \mathbf{a}_k} = \mathbf{p}, og \frac{\delta(\mathbf{a}_k^T \mathbf{R} \mathbf{a}_k)}{\delta \mathbf{a}_k} = 2\mathbf{R}\mathbf{a}_k \tag{2.11}$$

Kombinerer man dette får man at $\bigtriangledown\xi$ blir

$$\bigtriangledown \xi = \frac{\delta \xi}{\delta \mathbf{a}_k} = -2\mathbf{p} + 2\mathbf{R}\mathbf{a}_k \tag{2.12}$$

Optimalitet oppnåes ved å finne $\mathbf{a}_{k,opt}$ slik at gradienten blir null.

$$\begin{aligned} 0 &= -\mathbf{p} + \mathbf{R}\mathbf{a}_{k,opt} \\ \mathbf{a}_{k,opt} &= \mathbf{R}^{-1}\mathbf{p} \end{aligned} \tag{2.13}$$

Målet her er å justere filterkoeffisientene gjennom en passende algoritme for å finne det optimale punkt på prestasjonsflaten. Man kan også si at det optimale estimatet velger filterkoeffisienter slik at feilen blir ortogonal på dataene. Ortogonalitet er forklart i 2.2.1. Wiener-Hopf ligningen fra ligning 2.13 har begrenset praktisk anvendelse, fordi

- den krever at matrisen $\mathbf{R}$ og vektoren $\mathbf{p}$ er kjent på forhånd.
- den har en tidskrevende matriseinversjon.
- hvis signalene er ikke-stasjonære vil $\mathbf{R}$ og $\mathbf{p}$ forandre seg med tid og $a_{k,opt}$ må utregnes kontinuerlig.

Dermed kommer man til LMS algoritmen, som beregner $\mathbf{a}_{k,opt}$ sample for sample uten å måtte regne ut $\mathbf{R}$ og $\mathbf{p}$ eksplisitt.

Parametere:	
	N=filterorden
	μ=konvergeringsfaktor
Initialisering:	
	$\mathbf{a}_0 = 0$, filterkoeffisienter
Signaler	
	$\mathbf{x}_k$: referansesignalvektoren
	y_k: primærsignalet
For k = 0, 1, ...	
	$\varepsilon_k = y_k - \mathbf{a}_k^T \cdot \mathbf{x}_k$
	$\mathbf{a}_{k+1} = \mathbf{a}_k + 2\mu\varepsilon_k \mathbf{x}_k$

Tabell 2.1: Oversikt over LMS algoritmen for transversal filterstruktur

LMS algoritmen

Dette er en av de mest brukte algoritmene i adaptiv støykansellering. Den går også under navnet gradientbasert algoritme.
I LMS algoritmen oppdateres filterkoeffisientene sample for sample ved at neste filterkoeffisientvektor $\mathbf{a}_{k+1}$ er lik den nåværende $\mathbf{a}_k$ pluss en forandring proporsjonal til den negative gradient

$$\mathbf{a}_{k+1} = \mathbf{a}_k - \mu \bigtriangledown \xi \tag{2.14}$$

der μ kalles skrittlengden, en faktor som kontrollerer stabilitet og konvergensrate. Algoritmen gir ikke en eksakt løsning på problemet å minimere midlere kvadrats feil, men approksimerer løsningen. Isteden for å beregne ensemble midlingen, går man ut ifra stasjonæritet i korte tidsrom og utfører tidsmidling på dataene. Estimatet av de optimale filterkoeffisientene får dermed en stokastisk karakter, derfor kalles LMS algoritmen også for stokastisk gradient algoritmen.

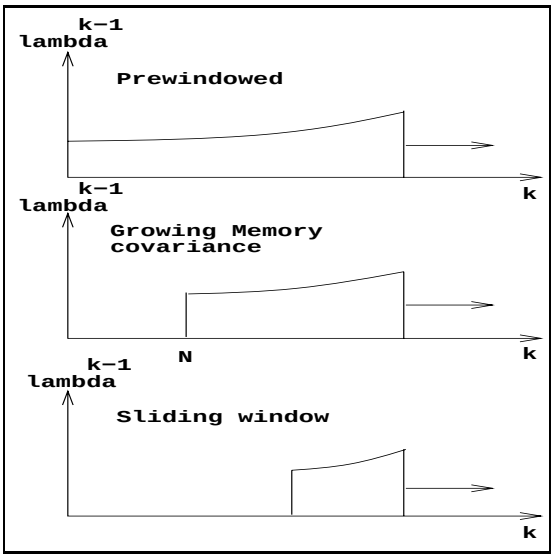
Man antar i algoritmen at ε_k^2 er et estimat av $E[\varepsilon_k^2]$ (ξ), det vil si ser bort fra forventningsoperatoren, og deriverer ε_k^2 med hensyn på $\mathbf{a}_k$. Forholdet mellom virkelig, $\bigtriangledown\xi$, og estimert, $\bigtriangledown\xi'$, gradient blir

$$\bigtriangledown \xi = \frac{\delta E[\varepsilon_k^2]}{\delta \mathbf{a}_k}, \bigtriangledown\xi' = \frac{\delta \varepsilon_k^2}{\delta \mathbf{a}_k} = 2\varepsilon_k \frac{\delta \varepsilon}{\delta \mathbf{a}_k} \tag{2.15}$$

$$\begin{aligned} 2\varepsilon_k \frac{\delta \varepsilon}{\delta \mathbf{a}_k} &= 2\varepsilon_k \frac{\delta[y_k - \mathbf{a}_k^T \mathbf{x}_k]}{\delta \mathbf{a}_k} \\ &= -2\varepsilon_k \frac{\delta[\mathbf{a}_k^T \mathbf{x}_k]}{\delta \mathbf{a}_k} \\ &= -2\varepsilon_k \mathbf{x}_k \end{aligned} \tag{2.16}$$

De to siste trinnene er mulig fordi hverken y_k eller $\mathbf{x}_k$ er påvirket av forandringer i $\mathbf{a}_k$.
Ved å sette gradientestimatet inn i steden for den virkelige gradient i ligning 2.14 får man filter-oppdaterings algoritmen, som sammen med ligningen 2.6 utgjør LMS algoritmen, vist i tabell 2.1.

Algoritmen er enkel, den utfører kun $2N + 1$ addisjoner/multiplikasjoner per vektoppdatering, der N er filterorden.



Figur 2.3: Typer av vindu

2.1.2 Algoritme basert på minste kvadrat metoden

I forrige avsnitt ble det vist at LMS algoritmen estimerer gradienten til prestasjonsflaten upresist. I dette kapittelet skal det bli sett nærmere på konseptet minste kvadrat (LS). Dette er en familie av algoritmer som er interessant i forbindelse med signaler som forandrer seg raskt. De kalles minste kvadrat fordi de minimerer summen av kvadratet av et feilsignal. LS-minimasjonen ble introdusert i avsnitt 2.1, der man minimerer kostfunksjonen v_k over vinduet W.

$$\mathbf{v}_k = \sum_{i \in W} \lambda^{k-i} \varepsilon_i^2 \tag{2.17}$$

Man opererer som regel med tre vindustyper;

- Prewindowed
- Growing Memory Covariance
- Sliding Window

Vindusfunksjonene er vist i figur 2.3, for nærmere beskrivelse henvises til [Haykin 91] side 373. Felles for disse vinduene er at de kan legge mindre vekt på tidligere data. Vektfaktoren λ er normalt litt mindre enn 1, hvis $\lambda = 1$ får man et rektangulært vindu. Typiske verdier er mellom 0.98 og 1. $\lambda = 1$ kan brukes i de tilfeller der inngangssignalene er stasjonære. Utsignalet kan bli ustabilt hvis man velger λ for liten, fordi man da legger for mye vekt på de siste dataene. Den effektive vinduslengden L får man som [Lebeda 90]

$$L = \frac{1}{1 - \lambda} \tag{2.18}$$

Ligninger for LS - filteret

Utledningen er gjort med hensyn på den enkleste vindusfunksjonen, prewindowed data. Ligningene kalles også normalligningene. Ligningene utledes ved å komme fram til $\frac{\delta \mathbf{v}_k}{\delta \mathbf{a}_k}$, og ved å sette dette lik null løse for den optimale filterkoeffisient-vektoren $\mathbf{a}_k$. En alternativ måte kunne vært å løse normalligningene fra prinsippet om ortogonalitet. Utgangspunktet for utledningen er ligning 2.4, og de aktuelle verdiene basert på figur 2.1 settes inn i denne [Haykin 91].

$$\begin{aligned}
\mathbf{v}_k &= \sum_{i \in W_{prew}} \lambda^i \varepsilon_i^2 \\
&= \sum_{i=1}^{k} \lambda^{k-i} (y_i - y_i')^2 \\
&= \sum_{i=1}^{k} \lambda^{k-i} (y_i - \mathbf{a}_k^T \cdot \mathbf{x}_i)^2 \\
&= \sum_{i=1}^{k} \lambda^{k-i} (y_i^2 + \mathbf{a}_k^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{a}_k - 2 y_i \mathbf{a}_k^T \mathbf{x}_i)
\end{aligned} \tag{2.19}$$

Hvis ligningen deriveres mhp. filterkoeffisientvektoren $\mathbf{a}_k$ og settes lik null, minimeres $\mathbf{v}_k$

$$\frac{\delta \mathbf{v}_k}{\delta \mathbf{a}_k} = \sum_{i=1}^{k} \lambda^{k-i} 2 \mathbf{x}_i \mathbf{x}_i^T \mathbf{a}_k - 2 y_i \mathbf{x}_i \tag{2.20}$$

$$\sum_{i=1}^{k} \lambda^{k-i} (2 \mathbf{x}_i \mathbf{x}_i^T \mathbf{a}_{k,opt} - 2 y_i \mathbf{x}_i) = 0 \tag{2.21}$$

Etter å ha ordnet dette uttrykket får vi

$$\sum_{i=1}^{k} \lambda^{k-i} \mathbf{x}_i \mathbf{x}_i^T \mathbf{a}_{k,opt} = \sum_{i=1}^{k} \lambda^{k-i} y_i \mathbf{x}_i \tag{2.22}$$

$$\phi_k \mathbf{a}_{k,opt} = \theta_k \tag{2.23}$$

der man har den tidsmidlede autokorrelasjonsmatrisen ϕ og krysskorrelasjonsvektoren θ

$$\phi_k = \sum_{i=1}^{k} \lambda^{k-i} \mathbf{x}_i \mathbf{x}_i^T \tag{2.24}$$

$$\theta_k = \sum_{i=1}^{k} \lambda^{k-i} y_i \mathbf{x}_i \tag{2.25}$$

Den optimale løsningen av normalligningen for filterkoeffisientene blir da

$$\mathbf{a}_{k,opt} = \phi_k^{-1} \theta_k \tag{2.26}$$

Denne løsningen krever N^3 multiplikasjoner og addisjoner, som ofte er uakseptabelt i sanntid.

RLS algoritmen

Ovenfor ble det vist at LS algoritmen ikke egner seg for filtrering i sanntid. Men algoritmen er likevel et fundament for utviklingen av den rekursive metoden i dette avsnittet. Utledningen av algoritmen er vist på basis av [Cioffi & Kailath 84].

Man har en kontinuerlig strøm av data, og siden man ønsker å oppdatere $\mathbf{a}_k$ med de nye dataene, kan man ta i bruk rekursive metoder. Det utnyttes at autokorrelasjonsmatrisen ϕ_k er positiv definit, det vil si at det for en vilkårlig vektor $\mathbf{u}$ gjelder at $\mathbf{u}^T\phi\mathbf{u} > 0$ [Lebeda 90].

Hvis $i = k$ isoleres i de to ligningene 2.24 og 2.25 kan man sette opp følgende;

$$\phi_k = \lambda[\sum_{i=1}^{k-1} \lambda^{k-1-i}\mathbf{x}_i\mathbf{x}_i^T] + \mathbf{x}_k\mathbf{x}_k^T \tag{2.27}$$

$$\theta_k = \lambda[\sum_{i=1}^{k-1} \lambda^{k-1-i}y_i\mathbf{x}_i] + y_k\mathbf{x}_k \tag{2.28}$$

Man ser at uttrykket innenfor hakeparantesene er lik henholdsvis ϕ_{k-1} og θ_{k-1}. Derfor kan det settes opp rekursjoner for oppdateringen slik;

$$\phi_k = \lambda\phi_{k-1} + \mathbf{x}_k\mathbf{x}_k^T \tag{2.29}$$

$$\theta_k = \lambda\theta_{k-1} + y_k\mathbf{x}_k \tag{2.30}$$

For å løse den inverse ϕ_k, som man trenger i filteroppdateringen, kan man bruke matriseinversjonslemmaet beskrevet i [Haykin 91] s 480, som sier at ved

$$\mathbf{A} = \mathbf{B}^{-1} + \mathbf{C}\mathbf{D}^{-1}\mathbf{C}^T \tag{2.31}$$

der $\mathbf{A}$ og $\mathbf{B}$ er positiv definit $M * M$ matriser, $\mathbf{D}$ er en positiv definit $N * N$ matrise og $\mathbf{C}$ en $M * N$ matrise, kan man uttrykke den inverse av matrisen $\mathbf{A}$ slik;

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{B}\mathbf{C}(\mathbf{D} + \mathbf{C}^T\mathbf{B}\mathbf{C})^{-1}\mathbf{C}^T\mathbf{B} \tag{2.32}$$

Bruker man følgende definisjoner; $\mathbf{A} = \phi_k, \mathbf{B}^{-1} = \lambda\phi_{k-1}, \mathbf{C} = \mathbf{x}_k, \mathbf{D} = 1$ får man

$$\phi_k^{-1} = \lambda^{-1}\phi_{k-1}^{-1} - \frac{\lambda^{-2}\phi_{k-1}^{-1}\mathbf{x}_k\mathbf{x}_k^T\phi_{k-1}^{-1}}{1 + \lambda^{-1}\mathbf{x}_k^T\phi_{k-1}^{-1}\mathbf{x}_k} \tag{2.33}$$

Dette kan forenkles ved å sette

$$\mathbf{P}_k = \phi_k^{-1} \tag{2.34}$$

og

$$\mathbf{v}_k = \frac{\lambda^{-1}\mathbf{P}_{k-1}\mathbf{x}_k}{1 + \lambda^{-1}\mathbf{x}_k^T\mathbf{P}_{k-1}\mathbf{x}_k} = \phi_k^{-1}\mathbf{x}_k = \mathbf{P}_k\mathbf{x}_k \tag{2.35}$$

Dermed kan 2.33 forenkles til

$$\mathbf{P}_k = \lambda^{-1}\mathbf{P}_{k-1} - \lambda^{-1}\mathbf{v}_k\mathbf{x}_k^T\mathbf{P}_{k-1} \tag{2.36}$$

Når det nå er funnet et uttrykk for den inverse autokorrelasjonsmatrisen, går man videre med oppdateringen av filterkoeffisientene fra ligning 2.26. For å oppdatere filterkoeffisientene starter man med følgende;

$$\mathbf{a}_k = \phi_k^{-1}\theta_k = \mathbf{P}_k\theta_k = \mathbf{P}_k[\lambda\theta_{k-1} + \mathbf{x}_k y_k] \tag{2.37}$$

Parametere:	
	N=filterorden
	λ=glemmefaktor
	σ liten konstant
Initialisering:	
	$\mathbf{P}_{-1} = \sigma^{-1}\mathbf{I}$
	$\mathbf{a}_0 = 0$, filterkoeffisienter
Signaler	
	$\mathbf{x}_k$: referansesignalvektor
	y_k: primærsignalet
For k = 0, 1, ...	
	$\mathbf{v}_k = \frac{\lambda^{-1}\mathbf{P}_{k-1}\mathbf{x}_k}{1+\lambda^{-1}\mathbf{x}_k^T\mathbf{P}_{k-1}\mathbf{x}_k}$
	$\varepsilon_k = y_k - \mathbf{a}_{k-1}^T\mathbf{x}_k$
	$\mathbf{a}_k = \mathbf{a}_{k-1} + \mathbf{v}_k\varepsilon_k$
	$\mathbf{P}_k = \lambda^{-1}\mathbf{P}_{k-1} - \lambda^{-1}\mathbf{v}_k\mathbf{x}_k^T\mathbf{P}_{k-1}$

Tabell 2.2: Oversikt over RLS algoritmen for transversal filterstruktur

Hvis man nå setter $\mathbf{P}_k$ bare inn i den første delen av ligningen, får man etter utregning

$$\mathbf{a}_k = \mathbf{a}_{k-1} - \mathbf{v}_k\mathbf{x}_k^T\mathbf{a}_{k-1} + \mathbf{P}_k\mathbf{x}_k y_k \tag{2.38}$$

Siden $\mathbf{P}_k\mathbf{x}_k$ er lik vektoren $\mathbf{v}_k$, får man den rekursive oppdateringen av $\mathbf{a}_k$;

$$\begin{aligned} \mathbf{a}_k &= \mathbf{a}_{k-1} + \mathbf{v}_k[y_k - \mathbf{x}_k^T\mathbf{a}_{k-1}] \\ &= \mathbf{a}_{k-1} + \mathbf{v}_k\varepsilon_k \end{aligned} \tag{2.39}$$

der

$$\varepsilon_k = y_k - \mathbf{a}_{k-1}^T\mathbf{x}_k \tag{2.40}$$

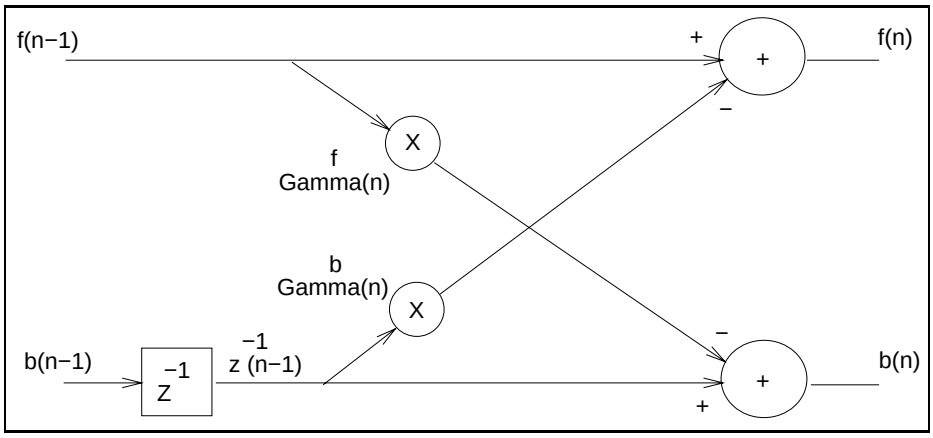
Man kan nå sette opp en fullstendig RLS algoritme i tabell 2.2.

I forhold til LMS, ser man at feilen ε_k blir multiplisert med den oppdaterte gainvektoren $\mathbf{v}_k$. Gainvektoren medfører at vektene blir oppdatert bedre i retning av det optimale punkt på prestasjonsflaten. $N * N$ matrisen ϕ_k må utregnes og inverteres for hver iterasjon/repetisjon. Dette gir operasjoner av N^2 orden, som er en god del mer enn ved LMS algoritmen.

I forhold til LMS algoritmen konvergerer RLS algoritmen hurtig uansett inngangssignalet. Ulemper ved RLS algoritmen kan deles i to. For det første kan man få såkalt "blow up", som kan opptre hvis $\mathbf{x}_k$ er null i lengre tid [Ifeachor & Jervis 93]. Da vil ϕ_k^{-1} vokse eksponensiellt som en følge av divisjon med λ ($<$1).

$$\lim_{k\to\infty}\phi_k^{-1} = \lim_{k\to\infty}(\frac{\phi_k^{-1}}{\lambda}) \tag{2.41}$$

Dessuten kan numeriske problemer ved korte vinduslengder føre til at algoritmen divergerer [Lebeda 90]. En løsning på det numeriske problemet er UD-faktorisering av ϕ^{-1}-matrisen. Dette foregår ved at man faktoriserer $\phi^{-1} = \mathbf{U}\mathbf{D}\mathbf{U}^T$ og utvikler rekursjoner for $\mathbf{U}$ og $\mathbf{D}$ matrisene direkte. $\mathbf{U}$ er en triangulær matrise med 1'ere i hoveddiagonalen, og $\mathbf{D}$ har kun elementer i hoveddiagonalen. Dette koster ekstra beregninger, men algoritmen har stadig kvadratisk sammenheng mellom filterorden og kompleksitet.



Figur 2.4: Blokk i gitter-strukturen

2.2 Algoritme basert på gitter-filteret

Gitter-strukturen er et alternativ til den transversale filterstruktur beskrevet i forrige avsnitt. Den blir i dette avsnittet beskrevet på FIR-form, selv om man også kan utlede gitter-IIR[7] filtre. Gitter-filteret er bygget opp av blokker som er satt i en kaskadestruktur, også kalt gitterprediktor. Figur 2.4 viser en slik blokk. Gitter-filterets struktur er forklart mer utdypende i [Honig & Messerschmitt 84] og [Haykin 91].

Man kan betrakte gitter-strukturen som en Gram-Schmidt ortogonalisering på inngangsdataene. Når man bruker minste midlere kvadrat metoden kombinert med gitter filterstruktur, vil ortogonaliseringen føre til raskere konvergens i forhold til det transversale filteret. [Honig & Messerschmitt 84] side 106.

Gitter-strukturen er definert ved ligningene

$$f(n) = f(n-1) - \Gamma(n)^b z^{-1} b(n-1) \tag{2.42}$$

$$b(n) = z^{-1} b(n-1) - \Gamma(n)^f f(n-1) \tag{2.43}$$

$f(n)$ er prediksjonsfeilen i forlengs retning, $b(n)$ er prediksjonsfeilen i baklengs retning og n er et uttrykk for filterorden. I de fleste gitter-filtrene er $\Gamma(n)^b = \Gamma(n)^f = \Gamma(n)$, denne størrelsen kalles også PARCOR[8] eller refleksjonskoeffisienter.

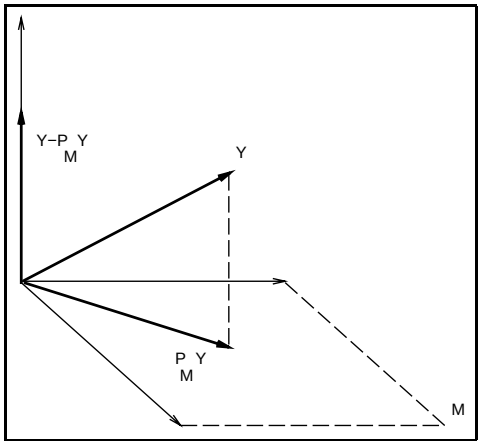
I neste avsnitt følger utledningen av gitter-filterets ligninger for adaptiv støykansellering basert på en geometrisk betraktning av signalene. Deretter kommer en beskrivelse av den aktuelle algoritmen for gitter-filteret, gradient adaptiv gittermetoden.

2.2.1 Utledning av gitter-filterets ligninger

Ortogonalitetsprinsippet

Ortogonalitet står sentralt i utledningen av ligningene for gitter-filteret. Det vil derfor forklares kort her. I et Euklidisk rom, [Honig & Messerschmitt 84] side 87, er en vektor et

[7] Infinite Impulse Respons
[8] partial correlation



Figur 2.5: Projeksjon i Hilbert rom

punkt i rommet, definert ved dens koordinater. En vektor i rommet er definert ved $\mathbf{X} \leftrightarrow (x_1, x_2, \ldots, x_n)$, der $\mathbf{X}$ er vektoren og $x_1, x_2, \ldots, x_n$ er komponentene i det n-dimensjonale rom. To vektorer i et Euklidisk rom er ortogonale hvis:

$$< \mathbf{X}, \mathbf{Y} >= 0 \tag{2.44}$$

der $< \mathbf{X}, \mathbf{Y} >$ er det indre produkt definert som

$$< \mathbf{X}, \mathbf{Y} >= \sum_{i=1}^{n} x_i y_i = ||\mathbf{X}||||\mathbf{Y}||cos\theta \tag{2.45}$$

det vil si produktet mellom lengden av $\mathbf{X}$, lengden av $\mathbf{Y}$ og cosinus til vinkelen mellom dem. I ligning 2.44 følger da at $\mathbf{X}$ og $\mathbf{Y}$ må være vinkelrett på hverandre.

Det defineres nå et underrom M i et k-dimensjonalt vektorrom H (Hilbert rom), se figur 2.5 [Honig & Messerschmitt 84] side 92. Man plasserer den ønskede signalvektoren $\mathbf{Y}$ i H, og projekserer $\mathbf{Y}$ på underrommet M. Det optimale estimat er den projekserede vektoren $\mathbf{Y}' = P_M\mathbf{Y}$, der P betyr projeksjonsoperator. Feilvektoren $\varepsilon = \mathbf{Y} - P_M\mathbf{Y} = \mathbf{Y} - \mathbf{Y}'$ er ortogonal (vinkelrett) på M

$$< \mathbf{Y} - P_M\mathbf{Y}, \mathbf{X} >= 0 \tag{2.46}$$

der $\mathbf{X}$ er en vilkårlig vektor i M. Dette kalles ortogonalitetsprinsippet, som sier at ved optimal innstilling av filterkoeffisientene er feilen ε ortogonal på filterestimatet, og dermed korteste avstand mellom $\mathbf{Y}$ og $P_M\mathbf{Y}$.

Utledning

I prediksjonsproblemet finner man et estimat av $\mathbf{Y}$ utfra $\mathbf{Y}$'s projeksjon i underrommet $M(1, n)$, utspent av vektorene $z^{-1}\mathbf{Y}, \ldots, z^{-n}\mathbf{Y}$. Et gitter-filter finner en ortogonal basis for dette underrommet $M(1, n)$ [Ingebretsen 86].
Forlengs feil er gitt som

$$f(n) = \mathbf{Y} - \sum_{i=1}^{n} a_i^f(n) z^{-i}\mathbf{Y} \tag{2.47}$$

og baklengs feil er gitt som

$$b(n) = z^{-n}\mathbf{Y} - \sum_{i=1}^{n} a_i^b(n) z^{-i+1}\mathbf{Y} \tag{2.48}$$

der $f(n)$ og $b(n)$ er feilene estimert av henholdsvis tidligere og etterfølgende verdier.

Som kjent ønsker man å gjøre feilen ortogonal på dataene. Ut fra projeksjonsteoremet, [Honig & Messerschmitt 84] side 94, er den vektoren i underrommet $M(1,n)$ som er nærmest til $\mathbf{Y}$, projeksjonen av $\mathbf{Y}$ på $M(1,n)$.

$$f(n) = \mathbf{Y} - P_{M(1,n)}\mathbf{Y} \tag{2.49}$$

Dette er forlengsfeilen til den optimale prediktoren.

Det blir nå satt opp to dekomposisjoner av underrom som er basisen for gitter-filteret. To underrom dannes ved å addere ortogonale komponenter til underrommet $M(1,n)$.

$$M(0,n) = M(1,n) \oplus \{f(n)\} \tag{2.50}$$

$$M(1,n+1) = M(1,n) \oplus \{z^{-1}b(n)\} \tag{2.51}$$

Siden $M(0,n)$ er definert som $z^{-0}\mathbf{Y}, \ldots, z^{-n}\mathbf{Y}$ og $M(1,n)$ er definert som $z^{-1}\mathbf{Y}, \ldots, z^{-n}\mathbf{Y}$, ser man at $M(1,n)$ mangler komponenten $\mathbf{Y}$, som finnes i $\{f(n)\}$, derfor er den første dekomposisjonen gyldig. På lignende måte kan man vise gyldigheten av den andre dekomposisjonen [Honig & Messerschmitt 84].

Siden både $f(n)$ og $z^{-1}b(n)$ er ortogonale på underrommet $M(1,n)$ og man har følgende definisjon fra [Honig & Messerschmitt 84]; hvis to ortogonale underrom M_1 og M_2 i Hilbert rommet H og en vilkårlig vektor $\mathbf{X}$ i H, kan projeksjonen $\mathbf{X}$ på $M_1 \oplus M_2$ uttrykkes som:

$$P_{M_1 \oplus M_2}\mathbf{X} = P_{M_1}\mathbf{X} + P_{M_2}\mathbf{X}. \tag{2.52}$$

Man får dermed

$$\begin{aligned} P_{M(1,n+1)}\mathbf{Y} &= P_{M(1,n)}\mathbf{Y} + P_{\{z^{-1}b(n)\}}\mathbf{Y} \\ P_{M(0,n)}(z^{-n-1}\mathbf{Y}) &= P_{M(1,n)}(z^{-n-1}\mathbf{Y}) + P_{\{f(n)\}}(z^{-n-1}\mathbf{Y}) \end{aligned} \tag{2.53}$$

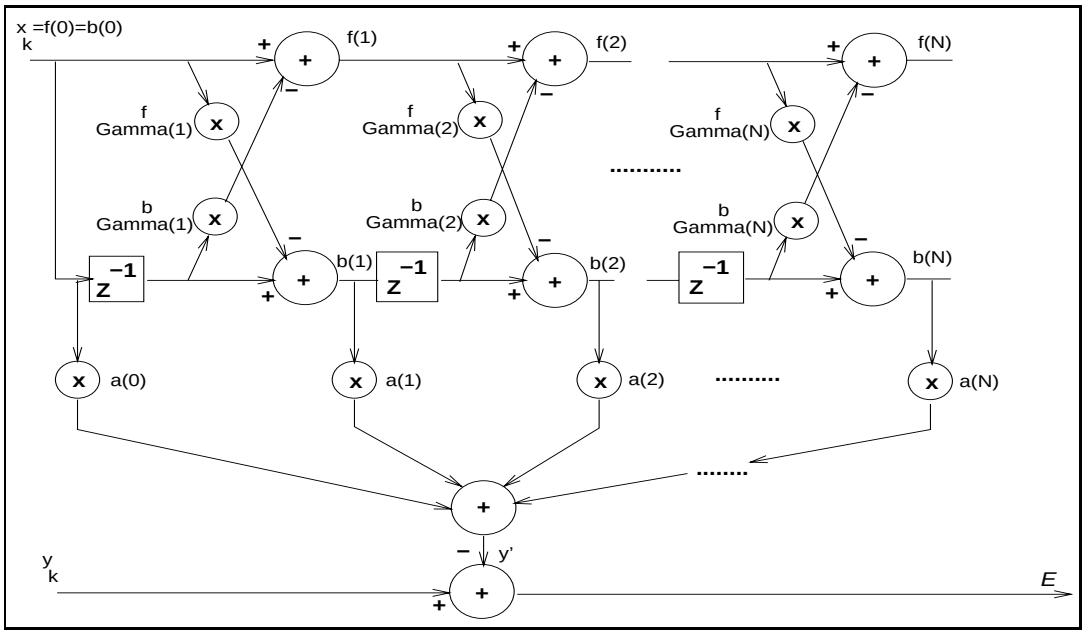
Forlengs feil kan således utledes fra 2.49:

$$\begin{aligned} f(n+1) &= \mathbf{Y} - P_{M(1,n+1)}\mathbf{Y} \\ &= \mathbf{Y} - P_{M(1,n)}\mathbf{Y} - P_{\{z^{-1}b(n)\}}\mathbf{Y} \\ &= f(n) - P_{\{z^{-1}b(n)\}}\mathbf{Y} \\ &= f(n) - \Gamma^b(n+1)z^{-1}b(n) \end{aligned} \tag{2.54}$$

Man får den siste overgangen i 2.54 fordi $P_{\{z^{-1}b(n)\}}\mathbf{Y}$ er på formen $\Gamma^b(n+1)z^{-1}b(n)$ for en konstant $\Gamma^b(n+1)$, som danner lengden på $\mathbf{Y}$'s projeksjon i underrommet $z^{-1}b(n)$.

Baklengsfeil utledes på lignende måte, se [Honig & Messerschmitt 84] eller [Ingebretsen 86], man får da:

$$b(n+1) = z^{-1}b(n) - \Gamma^f(n+1)f(n) \tag{2.55}$$



Figur 2.6: Struktur for adaptiv støykansellering med GAL algoritmen

2.2.2 Algoritme for gitter-filteret

Ved bruk av gitter-strukturen for støykansellering kobler man baklengs prediksjonsfeil $\mathbf{b}_k$ med filterkoeffisienter og summerer for å oppnå et estimat av n, se figur 2.1. Figur 2.6 gir et bilde av et slikt støykanselleringssystem, y' er utgangen av filteret og representerer et estimat av n.

Nå vil utledningen av gradient adaptiv gitter metoden bli vist. I [Friedlander 82] er flere minste kvadrat metoder for gitter-filteret utledet.

GAL algoritmen

Minste midlere kvadrat algoritmen for det transversale filteret ble gjennomgått i avsnitt 2.1.1, og fra dette prinsippet blir GAL algoritmen for gitter-filteret utledet. Metoden står beskrevet i [Griffiths 78] og [Haykin 91]. Gitter-Filteret skal konvergere hurtigere enn transversalfilteret fordi man estimerer både $\mathbf{a}_k$ og $\Gamma(n)$, se figur 2.6. Når $\Gamma(n)$ er bestemt foretar gitter-filteret en ortogonalisering av inngangsdataene, dermed forenkles filterkoeffisientene's ($\mathbf{a}_k$) estimasjon av støykomponenten n i y, se figur 2.1. Denne fordelen blir sterkt redusert hvis inngangssignalet er ukorrelert [Honig & Messerschmitt 84].

Utledningen av algoritmen tar utgangspunkt i kostfunksjonen ut fra hvert steg, se figur 2.4, og adapterer $\Gamma_k(n)$ slik at summen av forlengs og baklengs feil blir minimert. Man får et uttrykk for kostfunksjonen

$$J(n) = E[|f_k(n)|^2 + |b_k(n)|^2] \tag{2.56}$$

Gradienten av denne kostfunksjonen blir [Haykin 91] side 356

$$\bigtriangledown J(n) = 2E[f_k(n)b_{k-1}(n-1) + b_k(n)f_k(n-1)] \tag{2.57}$$

Parametere:

N=filterorden

μ=konvergeringsfaktor

Initialiser:

$\mathbf{a}_0 = 0$, filterkoeffisienter

$f_0(n) = b_0(n) = 0$

$\xi_0(n-1) =$ liten konstant

$\Gamma(n)(1) = 0$

Signaler

x_k: referansesignalet

y_k: primærsignalet

for k=0,1..

$f_k(0) = b_k(0) = x_k$

for n=0,1..

$$f_k(n) = f_k(n-1) - \Gamma_k^T(n) b_{k-1}(n-1)$$

$$b_k(n) = b_{k-1}(n-1) - \Gamma_k(n) f_k(n-1)$$

$$\xi_k(n-1) = \xi_{k-1}(n-1) + (|f_k(n-1)|^2 + |b_{k-1}(n-1)|^2)$$

$$\Gamma_{k+1}(n) = \Gamma_k(n) - \tfrac{1}{\xi_k(n-1)}[f_k(n-1) b_k^T(n) + b_{k-1}(n-1) f_k^T(n)]$$

end for n..

$$\varepsilon_k = y_k - \mathbf{a}_k^T \cdot \mathbf{b}_k$$

$$\mathbf{a}_{k+1} = \mathbf{a}_k + 2\mu\varepsilon_k \mathbf{b}_k$$

Tabell 2.3: Oversikt over GAL algoritmen

Man bruker øyeblikksestimater av forventningen:

$$\hat{\bigtriangledown} J(n) = 2[f_k(n) b_{k-1}(n-1) + b_k(n) f_k(n-1)] \tag{2.58}$$

Dermed kan man sette opp et uttrykk for oppdatering av refleksjonskoeffisientene $\Gamma(n)$:

$$\Gamma_{k+1}(n) = \Gamma_k(n) - \frac{1}{2}\mu_k(n)\hat{\bigtriangledown} J(n) \tag{2.59}$$

$\hat{\bigtriangledown} J(n)$ settes inn i 2.59, og man får:

$$\Gamma_{k+1}(n) = \Gamma_k(n) - \mu_k(n)[f_k(n) b_{k-1}(n-1) + b_k(n) f_k(n-1)] \tag{2.60}$$

Gitter-filterets konvergeringsfaktor $\mu_k(n)$ blir valgt slik at

$$\mu_k(n) = \frac{1}{\xi_k(n-1)} \tag{2.61}$$

der $\xi_k(n-1)$ representerer den totale energi av forlengs og baklengs prediksjonsfeil ved steg n, målt opp til og inkluderende tid k.

$$\xi_k(n-1) \quad = \quad \sum_{i=1}^{N} [|f(n-1)(i)|^2 + |b(n-1)(i-1)|^2]$$

Algoritme	Kompleksitet	Konvergens	Stabilitet
LMS	$2N+1$	Langsom	God
RLS	$4N^2+5N+2$	Rask	Mindre god
GAL	$14N$	Rask(avhengig av $\mathbf{x}_k$)	God

Tabell 2.4: Kort oversikt over algoritmenes egenskaper

$$\xi_k(n-1) = \xi_{k-1}(n-1) + |f_k(n-1)|^2 + |b_{k-1}(n-1)|^2 \tag{2.62}$$

Man har nå fått en oppdatering av prediksjonsfeilene i gitter-filteret sett ut fra figur 2.6, og man kan bruke LMS algoritmen til å oppdatere filterkoeffisientene.
Den fullstendige GAL algoritmen er vist i tabell 2.3.

2.3 Oppsummering for videre arbeid

Dette kapittelet har vist utledningen av de tre aktuelle algoritmene for adaptiv støykansellering. Vist i tabellene 2.1, 2.2 og 2.3 danner disse algoritmene basisen for simuleringene i senere kapitler. Hva kan foreløbig sies om de forskjellige algoritmene?

- **LMS**. Dette er den enkleste og mest brukte metoden for adaptiv støykansellering. Den har sin basis i det transversale filteret. Kompleksiteten er $2N+1$ addisjoner/multiplikasjoner per iterasjon, og algoritmen er stabil med oppfylte betingelser (for eksempel riktig konvergensrate). Det ble vist i avsnitt 2.1.1 at algoritmen kun approksimerer minimum midlere kvadrats feil, dette introduserer en avveining mellom rask konvergens og minimum feiljustering. Dette vil kommenteres ytterligere i avsnitt 4.3.1.

- **RLS.** Denne algoritmen har sin basis i den transversale filterstruktur. Algoritmen har kompleksitet $4N^2+5N+2$, men har mindre gode stabilitetsegenskaper. Aktuelle tilfeller der ustabilitet kan oppstå er hvis $\mathbf{x}_k$ er null i lengere tid. Algoritmen er også var ovenfor numeriske problemer ved korte vinduslengder. RLS algoritmen har rask konvergens og har i teorien minimum feiljustering.

- **GAL.** Algoritmen har sin basis i gitter filterstruktur. Algoritmen konvergerer noe hurtigere enn LMS algoritmen, [Haykin 91] side 359, fordi det i gitter-filteret estimeres to sett filteroppdateringer; filterkoeffisienter og refleksjonskoeffisienter. Denne fordelen gjelder bare hvis egenverdiene i autokorrelasjonsmatrisen til inngangssignalet $\mathbf{x}_k$ er sterkt ulike [Griffiths 78]. Algoritmen har kompleksitet $14N$ addisjoner/multiplikasjoner per iterasjon og gode stabilitetsegenskaper.

Tabell 2.4 viser en oversikt over disse egenskapene.

Før jeg starter med simuleringer av disse algoritmene, vil jeg nå rette blikket mot inngangssignalene i det adaptive støykanselleringssystemet. Som tidligere nevnt, er det en absolutt nødvendighet at man har høy korrelasjon mellom signalene x og n i figur 2.1. Korrelasjonen avhenger av hvor man plasserer referanseføleren. I neste kapittel følger en rapport basert på en undersøkelse av aktuelle plasseringer av denne referanseføleren. I tillegg er det gjort opptak av signaler for senere uttesting av algoritmene på et reelt datasett.

Kapittel 3

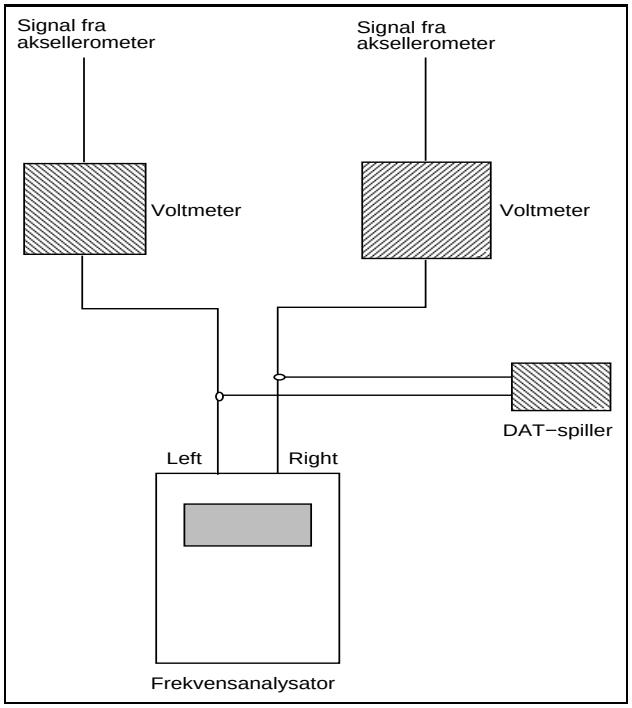
Måling av referanseplassering

Avsnitt i dette kapittelet

I figur 2.1 på side 8 er det skissert et adaptivt støykanselleringssystem. Hittil har det blitt konsentrert om hva som skjer i det adaptive filteret, nå skal det foretas en undersøkelse av inngangssignalene; primærsignalet og referansesignalet.

I teorien i kapittel 2 er det hentydet to forhold som i realiteten henger nøye sammen. Det er korrelasjonen mellom støydelen av primærsignalet og støyen i referansesignalet, og kravet om en god plassering (godt entydig støysignal) for måleren av referansesignalet.

Dette kapittelet omhandler en analyse av målinger av disse signalene gjort ombord på H. U. Sverdrup i samarbeid med Elling Tveit, FFI-U. Avslutningsvis vil det bli konkludert med en plassering av referanseføleren som tilfredstiller kravet til høy korrelasjon mellom referansesignalet og støydelen av primærsignalet.



Figur 3.1: Oppsett av måleinstrumentene for sammenligning av to plasseringer av referanseføleren

3.1 Utstyr brukt ved målingene

Følgende nødvendige utstyr for målingene ble lånt ut fra FFI-U.

- Sony DAT[1]-spiller TCD-D3, sn. 76675
- Hewlett Packard 3560 Dynamic Signal Analyzer, FFI no. 203845.
- Bruel & Kjær. Electronic Voltmeter 2425, FFI no. 202934.
- Bruel & Kjær. Electronic Voltmeter 2425, FFI no. 202865.
- 2 stk piezoelektriske akselerometere.

Et oppsett av dette utstyret er vist i figur 3.1. Oppsettet viser bruk av to akselerometere for sammenligning av plasseringer. Et annet oppsett som ble mye brukt er ett signal fra akselerometer og ett fra sonaren. Utover i oppgaven vil referanseføler også bli brukt for akselerometer. Nyansen vil bestå i at sistnevnte er en fysisk betegnelse, mens referanseføleren har en funksjonell betydning.

DAT-spilleren var en enkel "walkman"-type med 2 kanalers opptaksfunksjon. Det hendige formatet var praktisk i prosessen. Fordelen med to-kanalers opptak er at man får synkronisert samplingen av signalene. Dataene ble tatt opp på 4 taper, hver på 2 Gigabyte. Dette er relativt mye informasjon, og bare en brøkdel av dette brukes senere i oppgaven, men for senere evaluering er det nyttig med et omfattende datasett. Samplingsfrekvensen er 48kHz.

Frekvensanalysatoren var et nyttig instrument under opptakene. Man fikk kontinuerlig informasjon av frekvensspekteret slik at opptakene ble gjort under gunstigst mulig forhold.

[1] Digital Audio Tape

Frekvensanalysatoren var også nyttig for tilbakemelding for evaluering og kommentering underveis i arbeidsprosessen.

Begrensninger ved aksellerometere

Vibrasjon har sitt utgangspunkt i tannhjulene i gearboksen. Denne vibrasjonen transmitteres så gjennom skroget til svingerhuset, der man kan måle den med aksellerometere. Egenskapene til aksellerometerene er avhengig av flere faktorer. Av disse kan man nevne temperatur, magnetisme, fuktighet og lyd. I denne målingen har aksellerometeret ikke hatt nevneverdig påvirkning fra disse faktorene, bortsett fra varmepåvirkningen ved plassering av aksellerometer på gearboks. Siden aksellerometeret er festet til test-stedet med kitt, kan varmen gjøre kittet mykt og virke inn på aksellerometerets følsomhet. Ved en permanent aksellerometerføler kan man enten bruke en fastskrudd montering på test-stedet, eller man kan bruke en tilkobling basert på varmebestandig lim.

3.2 Måleresultater

For å få mulighet til å filtrere sonarsignalets støykomponenter adaptivt, er det nødvendig med god informasjon fra referanseføleren. Størst reduksjon av støyen får man når signalet fra referansekanalen er mest mulig lik støyen i sonarsignalet. I dette kapittelet er det vurdert plasseringer av referansekanalen, og et hovedspørsmål i den sammenheng er om man er best tjent med å plassere denne nær støyårsaken eller nær svingeren. En annen løsning kunne være å benytte ett av svingerelementene til referansehydrofon, beskrevet i [Knudsen 78]. Dette vil ikke bli behandlet her, fordi man får tilstrekkelig informasjon om støysignalet ved bruk av aksellerometermåler.

Nå følger en teoretisk forklaring av metoder brukt i evalueringen av signalene, dernest følger en oversikt over hvilke plasseringer som har vært evaluert, før det kommenteres og diskuteres rundt de forskjellige plasseringene for referanseføleren.

3.2.1 Teoretisk bakgrunn for beregning av plott

Hva ønsker man å se ut av opptakene? For det første er det nødvendig å finne de frekvenskomponentene som påvirker sonarsignalet, dette kan finnes med spektralanalyse. For det andre er det ønskelig å se om det finnes samsvar mellom referansesignal og sonarsignal, til dette kan man bruke koherensestimat.

Spektralanalyse

Spektralanalyse beskriver frekvensinnholdet til et deterministisk eller stokastisk signal med et endelig antall sampler. Effekttetthetsspekteret (ETS) av et stasjonært stokastisk signal x(n) er relatert matematisk til korrelasjonssekvensen til diskret-tid Fourier-transformen

$$P_{xx}(f) = \sum_{m=-\infty}^{\infty} R_{xx}(m) e^{-j f m} \tag{3.1}$$

Det er mange måter å estimere ETS på. Disse måtene kan deles i hovedgruppene parametrisk modellering og klassisk spektralestimering. For denne oppgaven er det brukt en funksjon i matlab, psd.m, som tar utgangspunkt i en klassisk metode utviklet av Welch.

Dette er en variasjon av midlet periodogram, som midler ved å beregne flere periodogrammer og summerer, man bruker datavinduer for hver datablokk og overlapper datablokkene. Redusert varians oppnåes ved at man beregner flere periodogrammer. Datavinduet reduserer lekkasje ved å redusere sidelobenivået av spektralestimasjonen for smalbåndede komponenter. Ved overlapping av datablokkene, blir variansen ytterlige redusert.
Uttrykket er utviklet fra ligningen for midlet periodogram [Kay 88] side 72.

$$\hat{P}_W(f) = \frac{1}{K}\sum_{m=0}^{K-1}\hat{P}_M^m(f) \tag{3.2}$$

der K er antall uavhengige datablokker

$$K = \frac{N-L}{S} + 1 \tag{3.3}$$

og N er datalengden, L er blokklengden og S er antall overlapp. Videre har man at

$$\hat{P}_M^m(f) = \frac{1}{UL}\left|\sum_{m=0}^{L-1} w[n]x_m[n]e^{-j2\pi fn}\right|^2 \tag{3.4}$$

der $w[n]$ er datavinduet, med lengde L. Dataene segmenteres i overlappende blokker

$$x_m[n] = x[n+mS], \begin{cases} 0 \le n \le L-1 \\ 0 \le m \le K-1 \end{cases} \tag{3.5}$$

U er et uttrykk for normalisering på grunn av energitapet ved bruk av vindusfunksjonen.

$$U = \frac{1}{L}\sum_{n=0}^{L-1}\|w[n]\|^2 \tag{3.6}$$

Følgende parametre er benyttet under beregning av plottene.

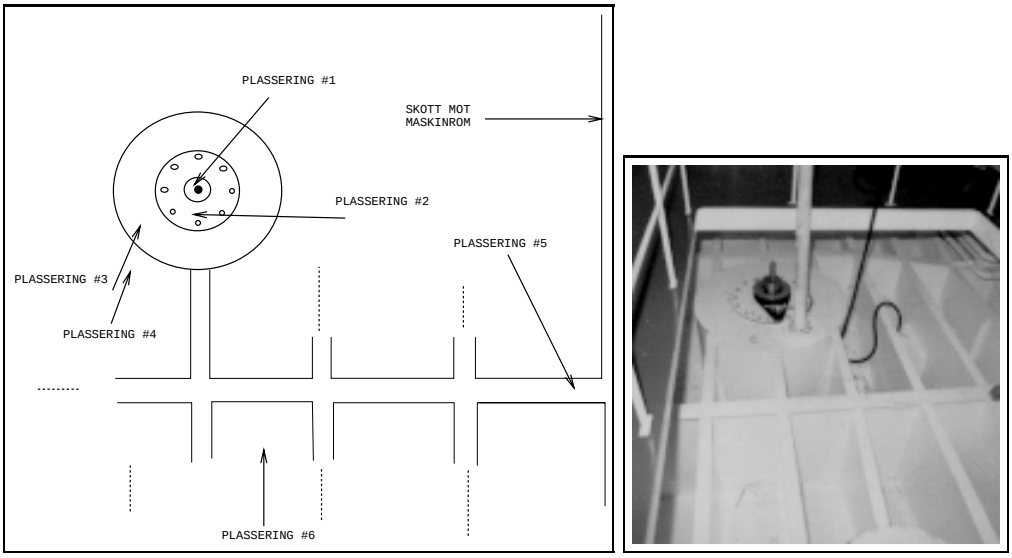
- Blokk-lengde L på 8K (8x1024)
- Hanning vindu
- Datalengde N på 120x1K (120x1024)
- Blokkoverlapp på 50%, det vil si at 29 periodogramestimater midles.

Frekvensoppløsning er gitt ved

$$f = \frac{1}{L\Delta t} = \frac{1}{8K\frac{1}{48K}} = 6Hz \tag{3.7}$$

der Δt er samplingsavstanden. Her har man en avveining mellom oppløsning og varians. Maksimum oppløsning fåes ved å velge $L = N-1$, mens ved å velge $K = \frac{N}{L}$ stor (eller L liten) oppnås størst reduksjon av variansen.

Et viktig punkt å nevne er forholdet stasjonæritet/ikke-stasjonæritet. Welchs metode er definert for stasjonære signaler, mens signalene fra sonaren og referanseføleren er ikke-stasjonære. Dette løses ved å definere de aktuelle signalene som stasjonære innenfor et tidsrom. Uten å sette en tydelig grense på dette intervallet av stasjonæritet, kan man definere at signalene er stasjonære innenfor den aktuelle datalengden i effekttetthetsspekteret. Matlabprogrammet er vist i appendiks A på side 83.



Figur 3.2: Måleplasseringer i svingerrom

Koherens mellom signaler

For å estimere koherensen mellom signalene er det brukt matlab's funksjon cohere.m. Koherensen mellom to signaler x(n) og y(n) er, fra [Kay 88] side 454.

$$C_{xy}(f) = \frac{|P_{xy}(f)|^2}{P_{xx}(f)P_{yy}(f)} \tag{3.8}$$

Kvotienten er et reelt tall mellom 0 og 1 som måler koherensen, samsvaret, mellom x(n) og y(n) i frekvensen f. Det ideelle er $C_{xy}(f) = 1$, mens hvis x og y er ukorrelerte blir $C_{xy}(f) = 0$. Matlabprogrammet er vist i appendiks A på side 83.

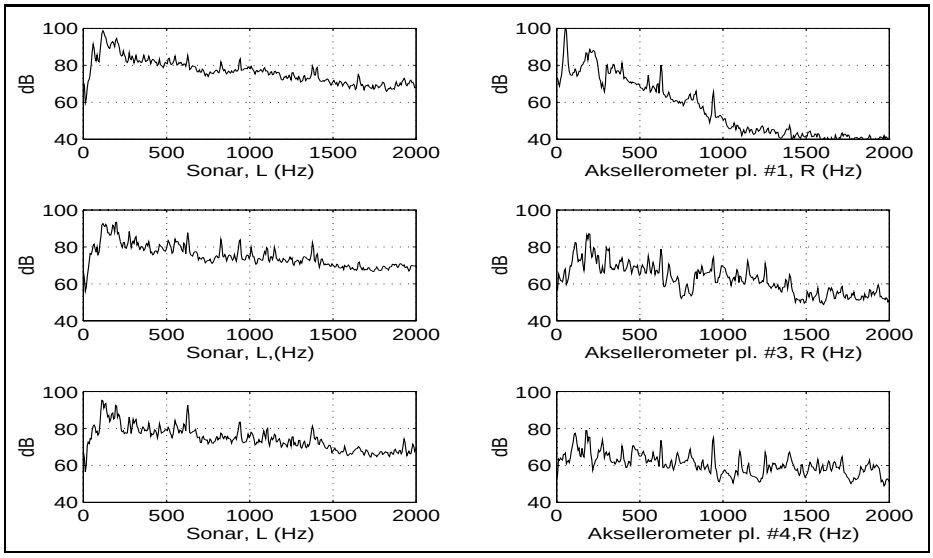
3.2.2 Målepunkter for referanseføleren

Målepunktene i maskinrom forklares underveis, og det vil i de aktuelle tilfeller være nyttig å se tilbake på figur 1.1 på side 2.

Målepunktene i svingerrom er vist på figur 3.2. Målinger ble utført på 6 punkter, men i overføringen fra DAT-tape til fil oppstod det en feil på måling av plassering #2, slik at denne målingen er ubrukelig.

Trunken som i figuren er representert ved den største sirkelen er 1,5 m. i diameter og avstanden fra svingertopp og skott mot maskinrom er 3 m. Nedenfor følger en nøkkel til plasseringene i svingerrom, se også figur 3.2.

1. På toppen av svinger.
2. På toppen av svingerdreier.
3. Plate på toppen av trunk.
4. Skrog nær trunk.
5. Bjelke nær skott mot maskinrom.
6. På skrog.



Figur 3.3: Plott av ETS i svingerrom rundt trunk

3.2.3 Målinger gjort i svingerrom rundt svingertrunk

Disse målingene omfatter akselleromeeterplasseringene #1, 3 og 4 vist på figur 3.2. Tanken bak disse plasseringene er at det er ønskelig med signaler uten tidsforskyvning. Plott av ETS og koherens er vist i henholdsvis figur 3.3 og 3.4.

Sonarsignal og aksellerometer plassering #1

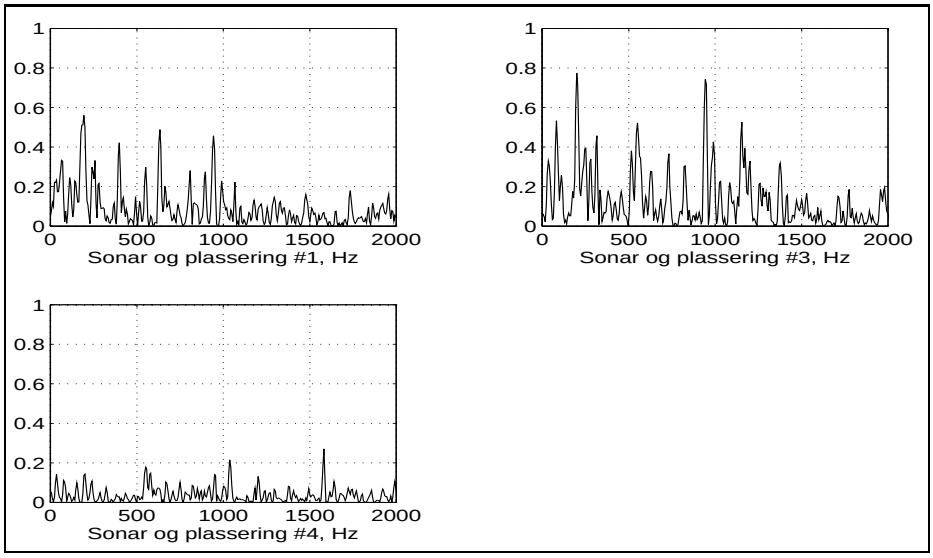
Opptaket av plassering #1 på toppen av svingeren ble gjort sammen med sonarsignalet. ETS for disse målingen er vist i de to øverste plottene i figur 3.3. Fra plottene antydes samsvar på frekvensene 630 og 940Hz innenfor det interessante frekvensområdet 500-1000Hz. Dette ble avlest ved hjelp av frekvensanalysatoren, og kan være vanskelig å lese nøyaktig ut av plottene. Frekvensene viser seg også som to topper som når en verdi opp mot 0,5 (over 0,5 anses som godt samsvar) i koherensplottet i figur 3.4.

Sonarsignal og aksellerometer plassering #3

Plott fra målingene fra sonarsignalet og plassering #3 i figur 3.3 viser også samsvar på frekvensene 630 og 940Hz. I koherensestimatet i figur 3.4 er det godt samsvar kun ved 940Hz, i frekvensområdet 500-1000Hz. Det ble notert at signalene fra plassering #3 var ustabile.

Sonarsignal og aksellerometer plassering #4

I plott fra sonarsignalet og plassering #4 i figur 3.3 ser man også klare topper ved frekvensene 630 og 940Hz, men her viser koherensplottet i figur 3.4 dårlig samsvar mellom sonar og aksellerometer.



Figur 3.4: Plott av koherensen mellom sonar og referanseføler i svingerrom

3.2.4 Målinger gjort i maskinrom

Plott av ETS er vist i figur 3.5 på side 32 og plott av koherens-funksjonen er vist i figur 3.6 på side 32. Motivasjonen for disse plasseringene var å se om det kunne finnes plasseringer med sterkere og mer stabile frekvenskomponenter enn i foregående undersøkelse.

Aksellerometer gearboks og aksellerometer plassering #4

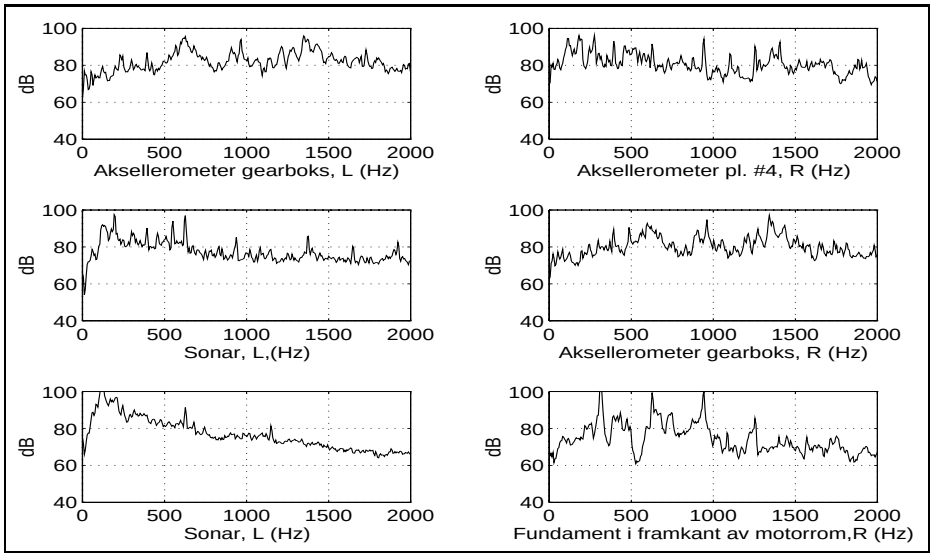
Målingen av aksellerometer på gearboks i øverste plott i figur 3.5 inneholder mye støy over et bredt bånd. Det som er verdt å legge merke til i plottet er forskjellen rundt 940Hz i forhold til plottet av aksellerometer på plassering #4. Opptaket fra gearboks har en frekvenslinje på 965Hz, målt ved hjelp av frekvensanalysatoren, kontra tidligere målt 940Hz.

Sonarsignal og aksellerometer gearboks

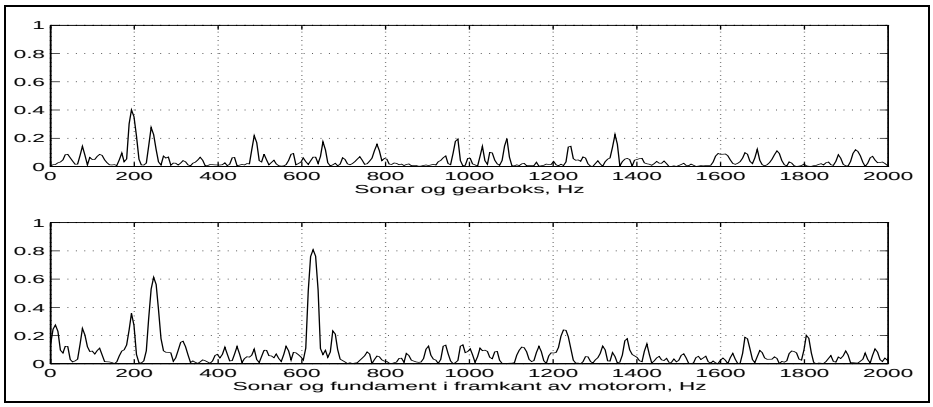
I den neste målingen av sonar og aksellerometer på gearboks i de midterste plottene i figur 3.5 gjenspeiles forholdet mellom frekvensene 940 og 965 tydelig. Det ble under opptakene observert at frekvenslinjene 940 og 965Hz fluktuerte i takt. Man kan derfor slutte at de har den samme årsaken. Det vil kommes tilbake til dette i konklusjonen i dette kapittelet, men foreløbig ser det ut som om at aksellerometer på gearboks ikke egner seg som referansesignal. Koherensestimatet i figur 3.6 understreker det dårlige samsvaret med verdier som jevnt over ligger lavt.

Sonarsignal og aksellerometer i framkant av motorrom

Opptakene fra framkant av motorrom virker derimot mer lovende. I plottet i figur 3.5 ser man tydelige frekvenser, og det var på dette stadium av arbeidet at det ble lagt merke til avhengigheten mellom 312.5, 625, 937.5 og 1250Hz. Med andre ord harmoniske av den førsteharmoniske 312.5Hz. I koherensestimatet i figur 3.6 ser man at bare 637.5 Hz viser et tilfredstillende samsvar.



Figur 3.5: Plott av ETS i maskinrom



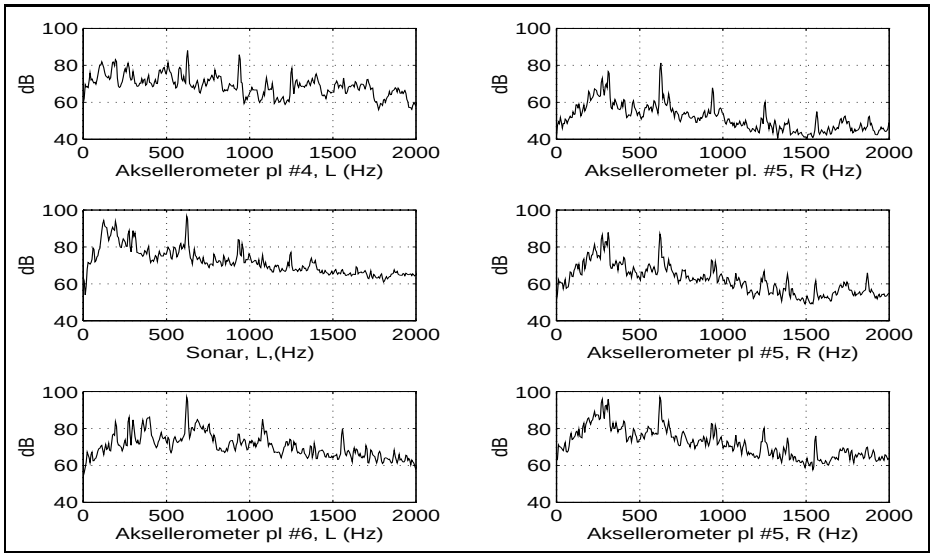
Figur 3.6: Plott av korrelasjonen mellom sonar og referanseføler i maskinrom

3.2.5 Målinger i svingerrom, bjelke og skrog

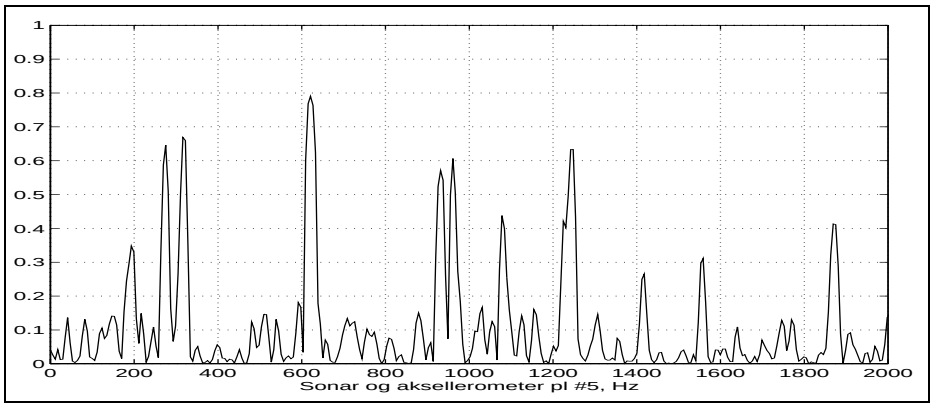
ETS fra disse målingene er vist i figur 3.7, koherensestimatet er vist i figur 3.8. Det ble nå returnert til svingerrom for å prøve å lokalisere steder der man kunne forvente gode signaler ved forplantning av vibrasjonene fra gearboks mot sonar.

Aksellerometer plassering #4 og aksellerometer plassering #5

I sammenligning mellom plassering #4 og #5 i de to øverste plottene i figur 3.7 ser man at den sistnevnte har tydelige frekvenskomponenter fra 312.5Hz og harmoniske av denne. Disse kommer ikke så tydelig fram i plottet fra plassering #4. Årsaken til dette kan ligge i at aksellerometer-plassering #5 er på en bjelke som har en mer direkte kontakt/vei til støyårsaken i motorrom. Plassering #4 har kontakt med flere bjelker, som kan gi et mer komplisert signal.



Figur 3.7: Plott av ETS i svingerrom, bjelke og skrog



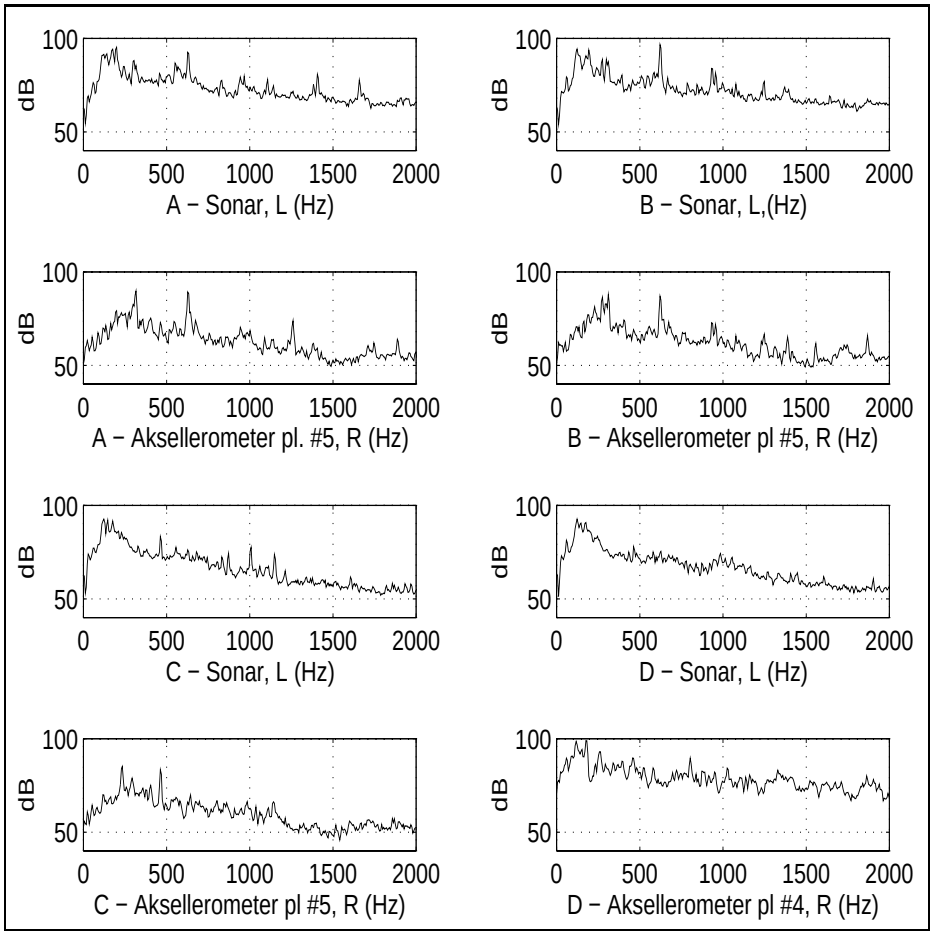
Figur 3.8: Plott av koherensen i svingerrom mellom sonar og plassering 5

Sonarsignal og aksellerometer plassering #5

Ser man på frekvensbildet i plott av plassering #5 og sonaren sammen i figur 3.7, ser man et tydelig samsvar i frekvensen 312.5Hz og harmoniske til denne. Plott av koherensestimatet i figur 3.8 viser at de overharmoniske frekvensene fra 312.5Hz alle ligger over verdien 0,6. Dette virker lovende for en plassering av referansesignalet.

Aksellerometer plassering #5 og aksellerometer plassering #6

Sammenligner man plottene av plassering #5 og plassering #6 i figur 3.7 får man et tilnærmet likt frekvensspekter. Skroget (plassering #6) er en av hovedveiene for transisjon av vibrasjonen fra gearbox, men man kan stille spørsmål ved støypåvirkning på grunn av den nære kontakten til sjøen ved denne plasseringen. Den bredbåndede støyen fra flow noise vil kunne påvirke akselleromeeteret.



Figur 3.9: Plott av ETS i svingerrom med aktiv sonar

3.2.6 Måling ved kjøring av aktiv sonar

ETS fra disse målingene er vist i figur 3.9, koherensestimatet er vist i figur 3.10. De neste målinger ble gjort med den parametriske sonaren i aktiv modus. 2 av målingene er tatt opp når motoren gikk med 600RPM[2], i stedet for 800RPM, som hittil har vært tilfelle. Plottene er merket fra A til D, målingene er tatt opp under følgende forhold:

A: Ricker puls 500Hz, 100% amplitude, 1 sek. reprate, 800RPM.

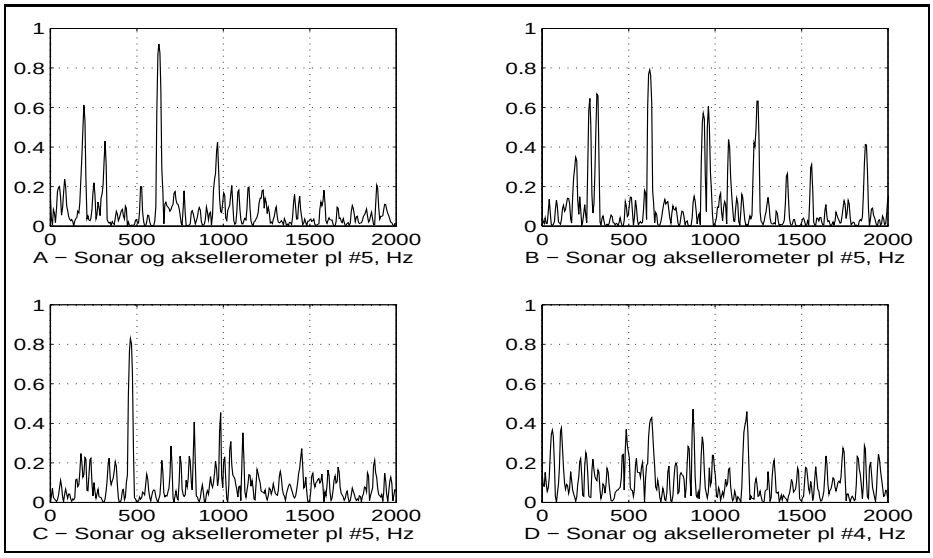
B: Ricker puls 1kHz, 100% amplitude, 2 sek. reprate, 800RPM.

C: Ricker puls 1kHz, 100% amplitude, 2 sek. reprate, 600RPM.

D: Ricker puls 1kHz, 100% amplitude, 2 sek. reprate, 600RPM.

Ricker er en kort puls med bredt frekvensbånd, 100% amplitude betyr maksimal utgangseffekt og med reprate menes repetisjonsintervall for utsending av pulser.

[2]Rotations Per Minute



Figur 3.10: Plott av koherensen i svingerom med aktiv sonar

Måling A og B: sonarsignal og aksellerometer plassering #5

I de to første plottene i figur 3.9 av målingene fra kjøring med TOPAS, (plott A og B), ser man ingen merkbar forandring i frekvensspekteret under 1000Hz i forhold til tidligere målinger. Koherensestimatene i figur 3.10 viser tilfredstillende samsvar ved frekvensene 625 og 945Hz.

Måling C: sonarsignal og aksellerometer plassering #5

Når omdreiningene ble justert ned fra 800 til 600 RPM, plott C i figur 3.9, ble det en merkbar forandring i frekvensspekteret. I plottet kan man se en frekvenslinje like under 500Hz. Hvis man går ut ifra at disse omtalte frekvenslinjene 312.5, 625Hz... skyldes gearboksen, så kan man sette opp følgende ligning;

$$\frac{600RPM}{800RPM} * 625Hz = 466Hz \tag{3.9}$$

Dette stemmer med plottet. Plott av koherensestimatet i figur 3.10 viser tydelig skifte i frekvensen (Plott A og B versus C). Man har nå at frekvensen 233Hz og overharmoniske av denne påvirker svingeren til den parametriske sonaren.

Måling D: sonarsignal og aksellerometer plassering #4

Den siste målingen, plott D i figur 3.9 er tatt med som sammenligning til målingene av plassering #5. Man har ikke de samme klare frekvenskomponentene i frekvensområdet 500-1000Hz. Koherensestimatet i figur 3.10 viser også klart dårligere samsvar.

3.3 Konklusjon og videre arbeid

Med tanke på en plassering av referansesignalet, er det 3 steder som skiller seg positivt ut. Det er plassering #5, #6 og i framkant av motorrom. Av disse vil jeg vektlegge plassering

#5 som den beste. Her er det gode transmisjonsforhold av vibrasjonen fra motorrom, og et egnet sted for en fast montering av akselleromeˆter. En plassering av akselleromeˆteret på skroget (plassering #6) kan medføre uønsket støy fra sjøen. Tanken om å plassere referanseakselleromeˆteren ved vibrasjonens utgangspunkt, viste seg ikke å være gunstig. Som nevnt i avsnitt 3.2.4 viser det seg tydelige frekvenser ved 965Hz, til forskjell fra frekvensene rundt 940Hz observert tidligere. Det er vanskelig å gi en forklaring på dette, men det er verdt å legge merke til det faktum at gearboksens høye temperatur vil kunne gi dårlig kontakt for akselleromeˆteren. En mulig årsak kan være at strukturen i skipet virker som et filter, slik at det kun er de omtalte støykomponentene som når transduseren.

Plassering #5 viste også en evne til å oppta gode signaler ved kjøring av 600RPM. Det er viktig, da referansesignalet skal kunne gi et godt bilde av støyen under alle forhold.

Jeg har vært i kontakt med leverandøren av skipets maskin, og fått vite at hovedtannhjulet i gearboksen inneholder 83 tenner. Med en omdreiningshastighet på 800 RPM gir dette en forventet frekvens på 146 Hz. Dette stemmer ikke med den viste fundamentale frekvensen 325 Hz. Det anbefales videre undersøkelser for å få klarlagt dette forhold, gjennomføringen av en slik undersøkelse anses å ligge utenfor oppgaven. For denne oppgaves problem er de eksisterende resultater tilfredstillende, og vil kunne brukes i videre behandling.

Kapittel 4

Konvergensanalyse

Avsnitt i dette kapittelet

I forrige kapittel, måling av referanseplasseringer, ble det gjort opptak av signalene for støykansellereren, og det ble konkludert med hvilken plassering av referanseføleren som ut fra de foretatte målinger var best. Disse opptakene, og da spesielt den som gir best resultat, vil stå sentralt i filtersimuleringer med de aktuelle algoritmene. Men før den side av saken behandles, vil det i dette kapittelet gjøres simuleringer som gir en mer direkte sammenligning mellom de aktuelle algoritmene. De reelle signalene tatt opp på H. U. Sverdrup, se kapittel 3, egner seg da dårlig. På grunn av disses ikke-stasjonæritet, er det vanskelig å se konvergensegenskapene for den midlere kvadratiske feil. Plott av midlere kvadratiske feil kalles i litteraturen også innlæringskurver [1]. Derfor vil det i dette kapittelet benyttes stasjonære signaler, som er invariante ovenfor tids-skifting.

[1] engelsk: learning curves

> In practice it is almost impossible to characterize the performance of an adaptive filter for a nonstationary environment exept by experimentation. Thus, for analytical purposes the stationary environment is frequently assumed. The manner in which the filter adapts to a stationary environment from an arbitrary initial condition is fortunately very informative with respect to its performance in a nonstationary environment, at least on an intiutive level [Honig & Messerschmitt 84].

Neste avsnitt vil inneholde en oversikt over oppbygningen av disse syntetiske stasjonære signalene. Deretter, for å utdype det man ser i plottene, er det inkludert en beskrivelse av konvergensteori. To forhold som i den sammenheng knytter teori og simulering sammen forklares; konvergenstid og feiljustering. Så følger en beskrivelse av simuleringene med algoritmene utledet i kapittel 2, deretter avsluttes det med oppsummering og konklusjon for det videre arbeidet.

4.1 Signaler i simuleringene

Ved valg av stasjonære signaler er det lagt vekt på å vise algoritmenes egenskaper på en oversiktlig måte. Valget falt på en kombinasjon av hvit støy og sinussignaler, fordelt etter følgende nøkkel, jamfør figur 2.1 på side 8.

- **Signalkilde**. Som ønsket signal er det brukt tilnærmet hvit støy generert ved matlab-funksjonen randn.m. Dette er en funksjon som genererer verdier på bakgrunn av en gaussisk fordeling [Kay 88] side 55. Hvit støy gir hver frekvenskomponent samme effekt over hele frekvensbåndet. Betegnelsen hvit kommer fra analogi til Newtons hvitt lys.
- **Støykilde**. Som referansesignal (støy) er det brukt ett eller flere sinus signaler. Disse vil skille seg godt ut fra hvit støy. To modeller er brukt, den ene består av et sinussignal med frekvens 300Hz, den andre består av tre sinuser på 200, 400 og 600 Hz.
- Det **støybefengte signalet** blir da summasjonen av de to ovenstående. Dette betyr at man har meget godt samsvar mellom referansesignalet og støyen i primærsignalet.

4.2 Teori konvergens/innlæringskurver

Hvilke måter har man for å sammenligne algoritmer med hverandre. Man kan;

1. Se hvordan midlere kvadratfeil (MSE[2])forandrer seg med tiden (antall iterasjoner).
2. Se hvordan filterkoeffisientene forandrer seg med tiden.

MSE har i dette aktuelle tilfellet kun praktisk betydning når man jobber med stasjonære signaler. Derfor vil det bli tatt i bruk den første metoden i dette kapittelet, mens den andre metoden blir brukt i neste kapittel, da det der vil arbeides med ikke-stasjonære signaler.

MSE har vært mye brukt i publikasjoner om adaptiv støykansellering, og går også under navnet innlæringskurver. Plott av MSE med hensyn på iterasjoner gir to egenskaper til filteret; konvergenstid og feiljustering.

[2]Mean Square Error

- Konvergenstiden er det antall sampler/iterasjoner som skal prosesseres før algoritmens feil har nådd et minimum $\varepsilon = y - y'$. Konvergenstiden avhenger, i tillegg til inngangssignalene, av visse stillbare parametre.
- Feiljustering er den relative avvikelse fra Wiener-løsningen, et mål for den feil som oppstår fordi algoritmen aldri faller helt til ro. [Haykin 91] side 166.

Egenskapene kan man se i plottene ved to faser; først en "konvergens-fase", så en "stabilt nivå-fase".

For å finne midlere kvadratfeil, er det benyttet to metoder;

1. Midle kvadratet av feilen over et vindu, typisk 10-20 iterasjoner langt.
2. Kjøre et antall simuleringer og midle kvadratfeilen over ensemblet for den aktuelle iterasjonstiden.

Begge metodene er i teorien brukbar, men problemer oppstår i metode 1 når feilen endrer seg mye over det definerte vinduet, slik det gjerne gjør i raskere adaptive algoritmer. Derfor brukes den første metoden i beregning av midlere kvadratfeil for LMS algoritmen, og den andre metoden i simulering av RLS algoritmen. Den andre metoden begrenses bare til de tilfeller der den er nødvendig, fordi det er en relativt regnekrevende metode.

Når det i neste kapittel simuleres med de virkelige signalene, vil en annen egenskap til den adaptive algoritmen være aktuell. Det er sporingsevne, som er evnen til å følge variasjonene i et ikke-stasjonært signal. Her skal algoritmen hurtig glemme fortiden, og beskrive signalets nuværende egenskaper.

4.3 Simulering av LMS algoritmen med transversalt filter

Den første simuleringen som ble gjort tok naturlig nok utgangspunkt i minste midlere kvadrats metode. Teorien bak denne metoden ble gjennomgått i avsnitt 2.1.1, og algoritmen er gitt i tabell 2.1. På bakgrunn av denne algoritmen er det laget et matlab program som er basisen for simuleringene, programlistingen er vist i appendiks B.1.

I LMS algoritmen er det to variabler som må bestemmes på forhånd:

- Filterorden, som varieres mellom 3 og 10.
- Konvergeringsfaktor, som varieres mellom 0.1, 0.01 og 0.001.

Før de utførte simuleringene kommenteres, kan det være nyttig å ta et blikk på teorien for den midlere kvadratiske feils oppførsel.

4.3.1 Beskrivelse av transient oppførsel for midlere kvadratisk feil

De følgende 4 punkter har alle sin basis i ligningen utledet i [Haykin 91] side 327. Denne ligningen er et uttrykk for tidsutviklingen av MSE for LMS algoritmen.

$$\xi = \sum_{i=1}^{N} \gamma_i \mathbf{c}_i^n + \frac{\xi_{MIN}}{1 - \sum_{i=1}^{N} \frac{\mu\lambda_i}{2-\mu\lambda_i}} \tag{4.1}$$

der $\mathbf{c}_i$ er egenverdiene i $N * N$ matrisen $\mathbf{B}$ definert ved;

$$b_{ij} = \begin{cases} (1-\mu\lambda_i)^2 + \mu^2\lambda_i^2 & i = j \\ \mu^2\lambda_i\lambda_j & i \neq j \end{cases} \tag{4.2}$$

λ_i er egenverdiene til autokorrelasjonsmatrisen $\mathbf{R}$ av inngangssignal vektoren $\mathbf{x}_k$, γ_i er en konstant og μ er konvergeringsfaktoren i algoritmen. Ut fra ligningen kan man se følgende.

1. Transiente komponenter av MSE framviser ikke svingninger. Den transiente komponenten av ξ er

$$\sum_{i=1}^{N} \gamma_i \mathbf{c}_i^n \tag{4.3}$$

   Fordi matrisen $\mathbf{B}$ er en positiv-definit matrise, er $\mathbf{c}_i$ reelle positive tall. Ensemble-midlet MSE består derfor av eksponentialer, ikke ensemble-midlet MSE har i tillegg støy, som kan dempes ved riktig bruk av konvergeringsfaktoren.

2. MSE konvergerer til en stabil verdi lik ξ_∞ hvis, og bare hvis, konvergeringsfaktoren μ oppfyller følgende krav;

$$0 < \mu < \frac{2}{\lambda_{MAX}} \tag{4.4}$$

   der λ_{MAX} er den største egenverdien i $\mathbf{R}$, og at også følgende krav er oppfylt:

$$\sum_{i=1}^{N} \frac{\mu\lambda_i}{2-\mu\lambda_i} < 1 \tag{4.5}$$

   der $\lambda_i, i = 1, 2, \ldots, N$ er egenverdiene til matrisen R, og N er filterorden.

   Dette kan forenkles til følgende uttrykk;

$$0 \leq \mu \leq \frac{2}{\delta_x^2} \tag{4.6}$$

   der δ_x^2 er et uttrykk for summen av midlede kvadratverdier av inngangsvektoren $\mathbf{x}_k$. Dette kalles også total inngangseffekt. Konvergeringsfaktoren μ bestemmer LMS algoritmens konvergeringsegenskaper. En stor verdi for μ vil få algoritmen til å konvergere raskt mot middelkvadratfeilens minimumspunkt, men dette vil også medføre en større feiljustering, se punkt 4.

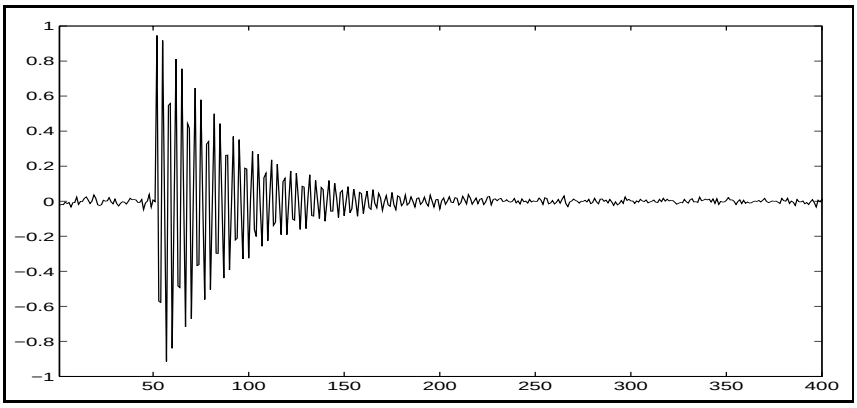
3. Man kan se at MSE har en endelig verdi ved å sette $c_i \leq 1$ og $n = \infty$ i ligning 4.1. Da får man;

$$\xi_\infty = \frac{\xi_{MIN}}{1 - \sum_{i=1}^{N} \frac{\mu\lambda_i}{2-\mu\lambda_i}} \tag{4.7}$$

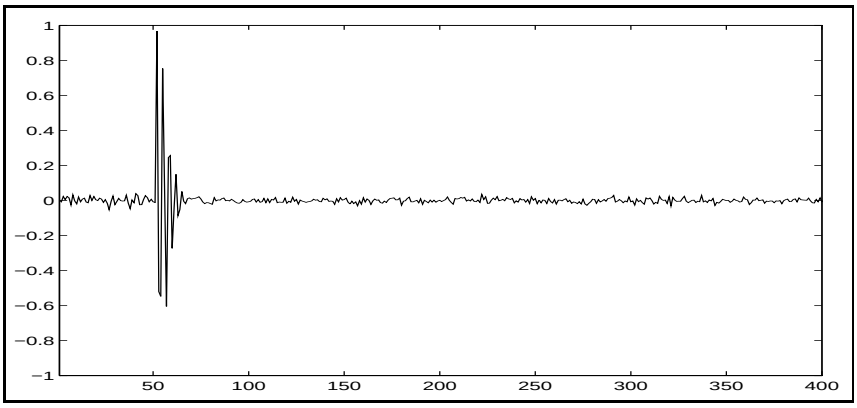
4. Feiljusteringen er definert som

$$M = \frac{\xi - \xi_{MIN}}{\xi_{MIN}} \approx \mu N \delta_x^2 \tag{4.8}$$

   hvor μ er konvergeringsfaktoren, N er filterordenen og δ_x^2 er effekten av inngangssignalvektoren. M gir et mål for adapsjonskosten. En verdi M=10 vil her bety at den adaptive algoritmen produserer en ξ etter adapsjonen som er 10% større enn ξ_{MIN}. Utfra ligningen ser man at M øker lineært med filterlengden og er direkte proporsjonal med konvergeringsfaktoren.
   Fordelen ved å bruke stor μ er rask konvergens, dette utnyttes best ved ikke-stasjonære signaler. Da må man akseptere en større feiljustering.



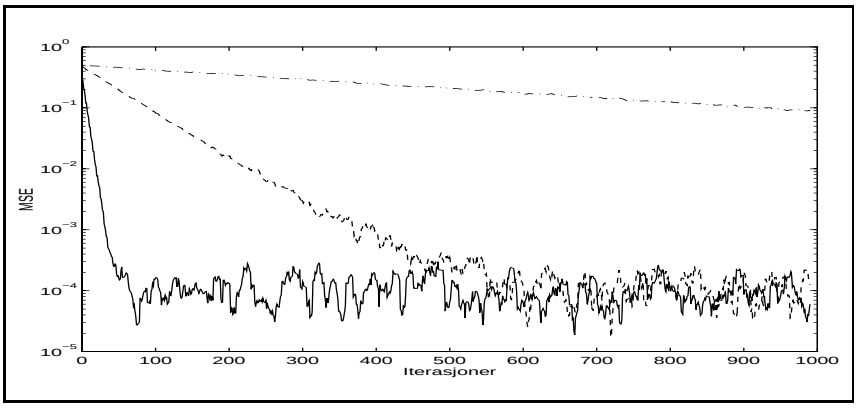
Figur 4.1: Plott av signalet for LMS ut fra filteret før og etter adapsjon, sett i forhold til iterasjonstid, N=10, konvergeringsfaktor=0.01.



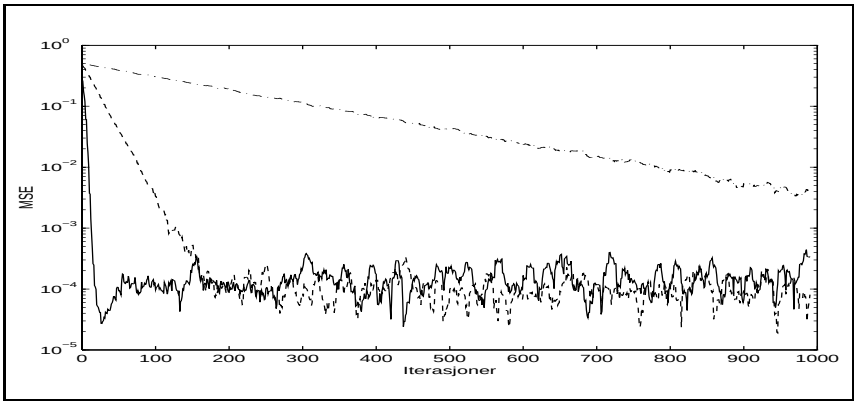
Figur 4.2: Plott av signalet etter filtrering for LMS, sett i forhold til iterasjonstid, N=10, konvergeringsfaktor=0.1.

4.3.2 Kommentarer til simuleringer med LMS algoritmen

3 faktorer påvirker LMS algoritmens filtrering: Konvergeringsfaktoren μ, filterorden N og egenverdiene av inngangsvektorens autokorrelasjonsmatrise $\mathbf{R}$. Forholdet mellom disse og feiljustering har man fra ligning 4.8. Man kan med de foreløbige kunnskaper forutse konvergensforløp og feiljustering. Konvergensen til MSE er i stor grad avhengig av konvergeringsfaktoren μ. Er μ liten, så er adapsjonen sakte og feiljusteringen liten, se ligning 4.8, fordi algoritmen bruker mer data på å estimere gradient vektoren. Ved liten konvergeringsfaktor har man bedre oppløsning og man kan komme nærmere minimumet til prestasjonsflaten. Svingningene rundt minimumspunktet blir mindre enn ved større konvergeringsfaktor. Er i motsetning μ stor, får man raskere adapsjon, men desto større feiljustering. Filterorden N's påvirkning på konvergenshastigheten har man til nå ingen kjennskap til, men man kan dedusere at et filter med flere filterkoeffisienter vil kunne øke adapsjonshastigheten. I følge ligning 4.8 kan man si at med liten N får man mindre feiljustering, og med større N høyere feiljustering.



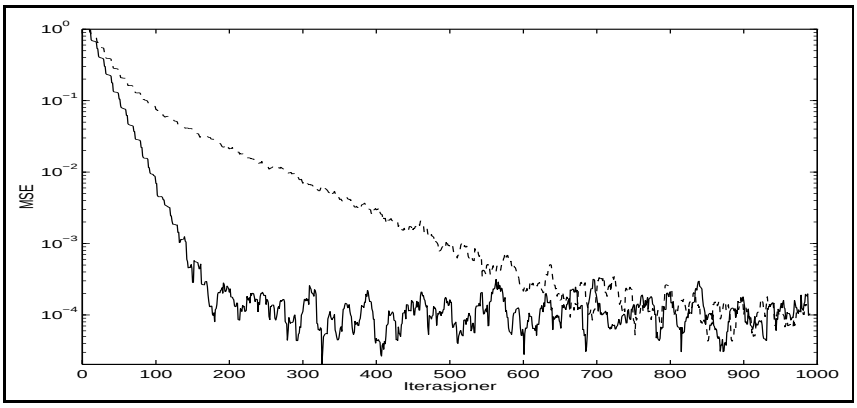
Figur 4.3: Plott midlere kvadratfeil for LMS, N=3. Konvergeringsfaktor er 0.001(-.), 0.01(- -) og 0.1(-).



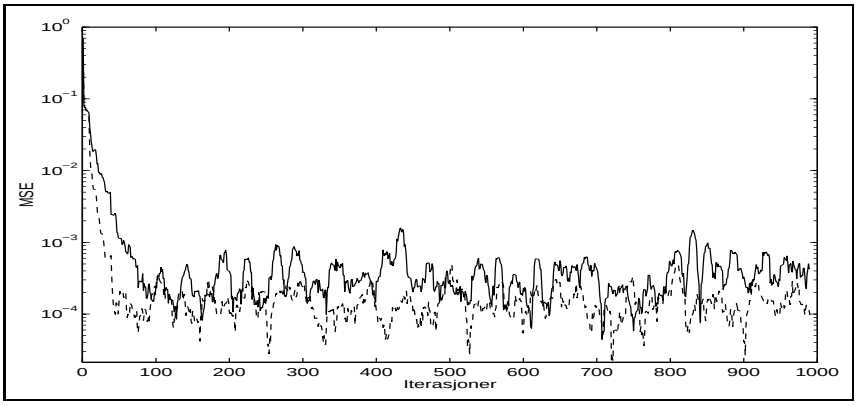
Figur 4.4: Plott av midlere kvadratfeil for LMS, N=10. Konvergeringsfaktor er 0.001(-.), 0.01(- -) og 0.1(-).

I det påfølgende blir simuleringene forklart, der målet er å filtrere sinuskomponenter. De to første plottene i figur 4.1 og 4.2 på side 41 viser signalet ut fra det adaptive filteret. Begge simuleringene er kjørt med en filterorden lik 10, men det er brukt forskjellige konvergeringsfaktorer. Man får her et godt inntrykk av hvordan konvergeringsfaktoren påvirker prosessen. I simuleringen i figur 4.1 med konvergeringsfaktor = 0.01 tar det omtrent 150 iterasjoner før signalet har stabilisert seg til et minimum. Dette skjer mye raskere i figur 4.2 med konvergeringsfaktor = 0.1, der utgangen av filteret har nådd en stabil minimumsverdi allerede etter 10-20 iterasjoner.

Det som er mest nærliggende for å teste de adaptive algoritmenes konvergens i forhold til hverandre i et stasjonært miljø, er å se på midlere kvadratfeil. I figur 4.3 er filterorden på $N = 3$, mens konvergeringsfaktoren varieres. Man ser tydelig hvordan MSE avtar, men med forskjellig rate. Adapsjon med konvergeringsfaktor = 0.001, vist med strek-prikk-linje i plottet, avtar meget sakte, og når ikke sin stabile minimumsverdi i løpet av simuleringens tids(iterasjons)-periode. Konvergeringsfaktor = 0.01, vist med stiplet linje i plottet, medfø-



Figur 4.5: Plott av MSE for LMS med N=3(- -) og N=10(-), konvergeringsfaktor = 0.01

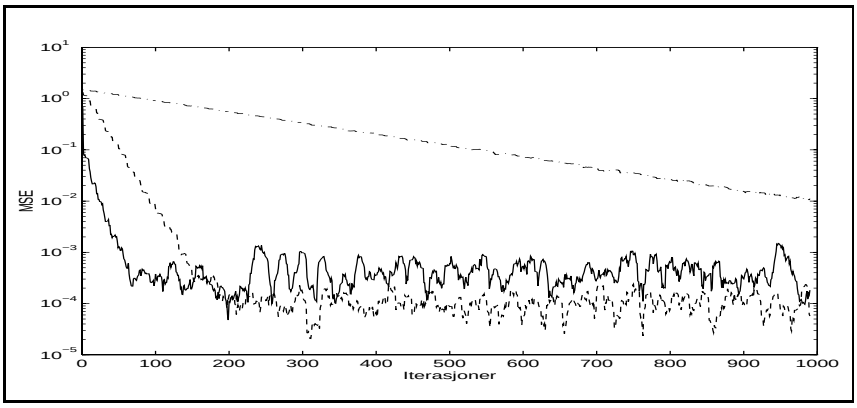


Figur 4.6: Plott av MSE for LMS med N=3 -) og N=10(- -), konvergeringsfaktor = 0.1

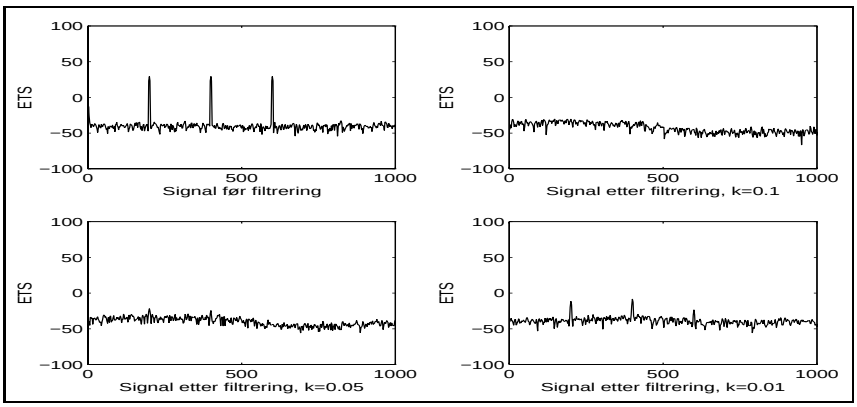
rer en raskere adapsjon ned til en stabil minimumsverdi i løpet av omtrent 400 iterasjoner, mens konvergeringsfaktor = 0.1, vist med hel linje, medfører en meget rask adapsjon, slik at allerede etter omtrent 50 iterasjoner er den midlere kvadratfeil på et minimumsnivå. Det er vanskelig å trekke noen slutninger ut ifra plottet i figur 4.3 om feiljusteringen, omtalt i avsnitt 4.3.1.

Den neste simuleringen har samme utgangspunkt som det forrige, med de samme variasjonene i konvergeringsfaktoren (vist med de samme linjetypene). Derimot er filterorden forandret til $N = 10$. Dette gir den effekt, som man kan se i figur 4.4, at midlere kvadratfeil konvergerer raskere mot minimumspunktet, midlere kvadratfeil med konvergeringsfaktor = 0.01 når sitt minimumspunkt allerede etter omtrent 150 iterasjoner, mot 400 i forrige simulering. Dette skulle, i henhold til avsnitt 4.3.1 punkt 4, gi en øket feiljustering. Det er vanskelig å trekke en slutning om dette ut ifra plottet i figur 4.4.

I figurene 4.5 og 4.6 er det gjort simuleringer for å se nærmere på filterorden N's påvirkning på konvergensen. I figur 4.5 er det simulert med konvergeringsfaktor 0.01. Resultatet er en raskere konvergens med filterorden lik 10. Dette er resultatet også i figur 4.6, der konvergeringsfaktoren er 0.1. Sammenligner man plottene, kan man se at påvirkningen fra



Figur 4.7: Plott midlere kvadratfeil for LMS, N=10, simulering med tre støykomponenter. Konvergeringsfaktor er 0.001(-.), 0.01 - -) og 0.1(-).



Figur 4.8: Plott av ETS med tre sinus støysignaler

filterorden er av mindre betydning ved høyere konvergeringsfaktor.

I figur 4.7 er det tatt med en simulering der støy (referanse)-signalet er bygget opp av flere sinuser, se avsnitt 4.1. Dette skyldes en tilnærmelse mot simulering med reelle signaler i neste kapittel. Man ser, som tidligere, at konvergeringsraten samstemmer med konvergeringsfaktoren. Men i tillegg ser man nå hvordan feiljusteringen blir påvirket i henhold til ligning 4.8 i avsnitt 4.3.1. En bekreftelse av denne feiljusteringen får man ved å beregne gjennomsnittlig MSE over iterasjonene 400-1000, etter at MSE er konvergert. Da får man, for $N = 10$, at;

- $\mu = 0.1$ gir en gjennomsnittlig MSE verdi $3.9232 * 10^{-4}$
- $\mu = 0.01$ gir en gjennomsnittlig MSE verdi $1.1574 * 10^{-4}$

I figur 4.8 er det vist plott av effekttetthetsspekteret (ETS) før og etter filtrering, etter at adapsjonen er gjennomført. Man ser i figuren hvordan konvergeringsfaktoren påvirker filtreringen. Man må her ha en konvergeringsfaktor i området 0.1 - 0.05 for å fjerne støykomponentene helt.

4.4 Simulering av RLS algoritmen med transversalt filter

Etter å ha gjennomført en simulering med minste midlere kvadrat metoden, er det nå klart for å ta fatt på en ny simulering med basis i den rekursive minste kvadratiske metode. Teorien for denne metoden er beskrevet i avsnitt 2.1.2, og den sentrale algoritmen er gitt i tabell 2.2. Ut fra denne algoritmen er det skrevet et matlab program gitt i appendiks B.2.

Sammenlignet med simuleringene i det forrige kapittelet, kan man si at de aktuelle variablene for rekursiv minste kvadratiske metode har ubetydelig påvirkning på resultatet av filtreringen. Man kan på grunn av signalenes stasjonæritet med hell sette den eksponensielle vektfaktoren λ til 1. Disse simuleringene er derfor representert med færre plott. Plottene vil bli kommentert i avsnitt 4.4.2, men først skal konvergensteorien for den rekursive minste kvadratiske metode vises.

4.4.1 Konvergensteori for RLS algoritmen

Denne teorien er hentet fra [Haykin 91] side 489-491.
For å vise konvergensegenskapene til RLS algoritmen, går man fram ved først å omskrive ligning 2.6.

$$\varepsilon_k = y_k - \mathbf{x}_k^T \mathbf{a}_{k-1} \tag{4.9}$$

til

$$\varepsilon_k = \mathbf{x}_k^T (\mathbf{a}_{opt} - \mathbf{a}_{k-1}) + \varepsilon_{opt,k} \tag{4.10}$$

Dette kan gjøres fordi man i en situasjon med optimal Wiener-filtrering kan skrive den ønskede responsen y_k som

$$y_k = \mathbf{x}_k^T \mathbf{a}_{opt} + \varepsilon_{opt,k} \tag{4.11}$$

der $\mathbf{a}_{opt}$ er den optimale Wiener filterkoeffisienten og $\varepsilon_{opt,k}$ er den korresponderende feilverdien. Videre får man middelkvadratfeilen

$$E[\varepsilon_k^2] = E[(\mathbf{x}_k^T (\mathbf{a}_{opt} - \mathbf{a}_{k-1}) + \varepsilon_{opt,k})^2] \tag{4.12}$$

Ortogonalitetsprinsippet sier at

$$E[\varepsilon_{opt,k} \mathbf{x}_k] = 0 \tag{4.13}$$

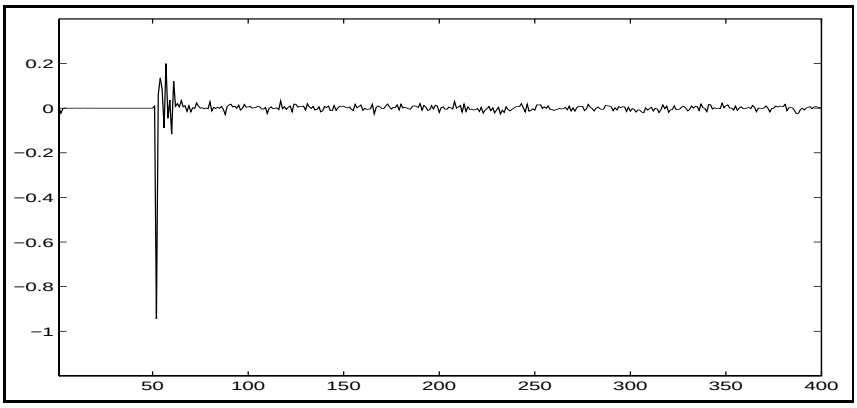
og siden $\mathbf{x}_k^T (\mathbf{a}_{opt} - \mathbf{a}_{k-1})$ og $\varepsilon_{opt,k}$ er ortogonale, kan man forenklet skrive

$$E[\varepsilon_k^2] = E[(\mathbf{a}_{opt} - \mathbf{a}_{k-1})^T \mathbf{x}_k \mathbf{x}_k^T (\mathbf{a}_{opt} - \mathbf{a}_{k-1})] + \varepsilon_{MIN} \tag{4.14}$$

der ε_{MIN} er minimum middelkvadratfeil.
Ved å ta i bruk funksjonen trace[3] og operasjoner rundt denne, får man følgende utregning [Haykin 91] side 490.

$$\begin{aligned}
E[\varepsilon_k^2] &= tr[E[(\mathbf{a}_{opt} - \mathbf{a}_{k-1})^T \mathbf{x}_k \mathbf{x}_k^T (\mathbf{a}_{opt} - \mathbf{a}_{k-1})]] + \varepsilon_{MIN} \\
&= E[tr[(\mathbf{a}_{opt} - \mathbf{a}_{k-1})^T \mathbf{x}_k \mathbf{x}_k^T (\mathbf{a}_{opt} - \mathbf{a}_{k-1})]] + \varepsilon_{MIN} \\
&= E[tr[(\mathbf{a}_{opt} - \mathbf{a}_{k-1})(\mathbf{a}_{opt} - \mathbf{a}_{k-1})^T \mathbf{x}_k \mathbf{x}_k^T]] + \varepsilon_{MIN} \\
&= tr[E[(\mathbf{a}_{opt} - \mathbf{a}_{k-1})(\mathbf{a}_{opt} - \mathbf{a}_{k-1})^T \mathbf{x}_k \mathbf{x}_k^T]] + \varepsilon_{MIN} \\
&= tr[E[(\mathbf{a}_{opt} - \mathbf{a}_{k-1})(\mathbf{a}_{opt} - \mathbf{a}_{k-1})] E[^T \mathbf{x}_k \mathbf{x}_k^T]] + \varepsilon_{MIN}
\end{aligned} \tag{4.15}$$

[3] Sum av matrisens diagonale elementer



Figur 4.9: Plott av signalet for RLS før og etter adapsjon, N=10

Ved å sette

$$\mathbf{R} = E[\mathbf{x}_k \mathbf{x}_k^T] \tag{4.16}$$

og

$$\mathbf{P}_{k-1} = \frac{1}{\varepsilon_{MIN}} E[(\mathbf{a}_{opt} - \mathbf{a}_k)(\mathbf{a}_{opt}\mathbf{a}_k^T)] \tag{4.17}$$

inn i ligning 4.15 får man

$$E[\varepsilon^2] = \varepsilon_{MIN} tr[\mathbf{P}_{k-1}\mathbf{R}] + \varepsilon_{MIN} \tag{4.18}$$

ved store k (iterasjoner) har man

$$\mathbf{P}_{k-1} = \frac{1}{k-1}\mathbf{R}^{-1} \tag{4.19}$$

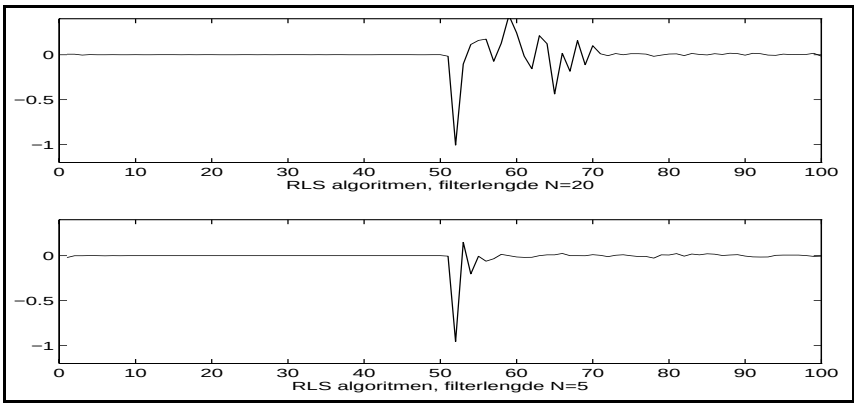
slik at man får et uttrykk for algoritmens konvergens

$$E[\varepsilon^2] = \frac{\varepsilon_{MIN}}{k-1} tr[\mathbf{R}^{-1}\mathbf{R}] + \varepsilon_{MIN} = \varepsilon_{MIN}(1 + \frac{M}{k-1}) \tag{4.20}$$

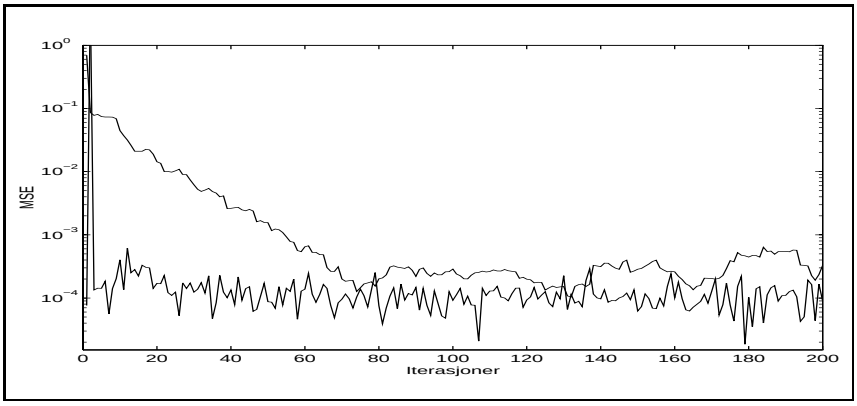
Ut fra dette uttrykket kan man si følgende;

- RLS konvergerer etter omtrent $2M$ iterasjoner, der M er filterorden.
- Når antall iterasjoner nærmer seg uendelig, nærmer MSE seg en verdi tilnærmet variansen δ_x^2 av inngangssignalet ε_{MIN}. Siden minimum MSE også er lik δ_x^2, følger det at RLS i teorien har null feiljustering i et stasjonært miljø.

Noen forutsetninger må være oppfylt for at RLS algoritmen skal ha konvergensegenskaper i henhold til de ovenstående punkter. For det første må man ha høyt signal/støy forhold, for det andre må man ha glemmefaktor $\lambda = 1$ for å oppnå null feiljustering.



Figur 4.10: Plott av signalet for RLS, N=5 og 20



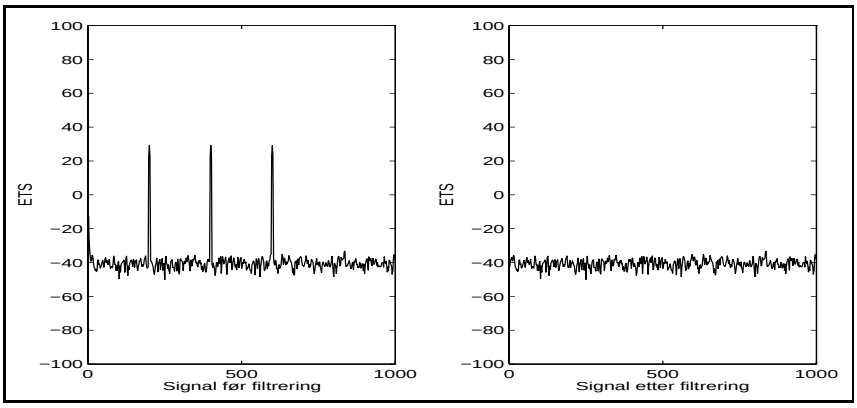
Figur 4.11: Plott av midlere kvadratfeil for RLS og LMS, N=10, konvergeringsfaktor=0.1.

4.4.2 Kommentarer til simuleringer med RLS algoritmen

Når man nå kommenterer disse simuleringene, kan man ha to forhold i tankene. For det første kan man gjøre sammenligninger med LMS algoritmen i forrige avsnitt, for det andre kan man forholde seg til de i forrige avsnitts nevnte teoretiske konvergensegenskaper til RLS algoritmen.

I figur 4.9 er det vist plott av signalet ut av filteret. Som man ser klarer filteret raskt å dempe støypåvirkningen. Denne simuleringen ble utført med en filterorden $N = 10$ foruten som tidligere nevnt vektfaktor $\lambda = 1$. Det er ønskelig å se litt nærmere på adapsjonshastighetens forhold til N, derfor er det kjørt to simuleringer vist i figur 4.10, der filterorden er henholdsivs $N = 20$ og $N = 5$. Man ser i figuren at RLS konvergerer raskere enn $2N$ iterasjoner, faktisk opp mot N iterasjoner. Dette skyldes at signal/støy-forholdet i dette aktuelle tilfellet er meget godt.

For å få et bedre bilde av RLS algoritmens konvergensegenskaper i forhold til LMS algoritmen, er det tatt med et plott av begge disse. De har begge filterorden $N = 10$, og LMS algoritmen har konvergeringsfaktor $\mu = 0.1$. Simuleringene med RLS algoritmen viser tydelig det samme hendelsesforløpet som ved LMS algoritmen. MSE starter med en



Figur 4.12: Plott av ETS for RLS, N=10

høy verdi, for deretter å falle raskt mot et minimum. Med likheten følger det også klare forskjeller. RLS algoritmen konvergerer mye raskere enn LMS algoritmen. Allerede etter noen få sampler har MSE nådd sin minimumsverdi. Med tanke på at algoritmen har en filterlengde $N = 10$, så konvergerer RLS mye raskere enn teorien i forrige avsnitt skulle tilsi.

Videre i følge avsnitt 4.4.1 var konvergenshastigheten til RLS algoritmen avhengig av godt signal/støyforhold. I eksempelet med syntetiske signaler synes dette å være oppfylt. Plottet i figur 4.11 viser også at RLS algoritmen har en marginalt bedre feiljustering enn LMS algoritmen. Dette stemmer også godt med konvergensteorien fra forrige avsnitt.

Det siste plottet i figur 4.12 viser effekttetthetsspekteret etter adapsjon for RLS algoritmen. Plottet viser at algoritmen fint klarer å filtrere bort støykomponentene.

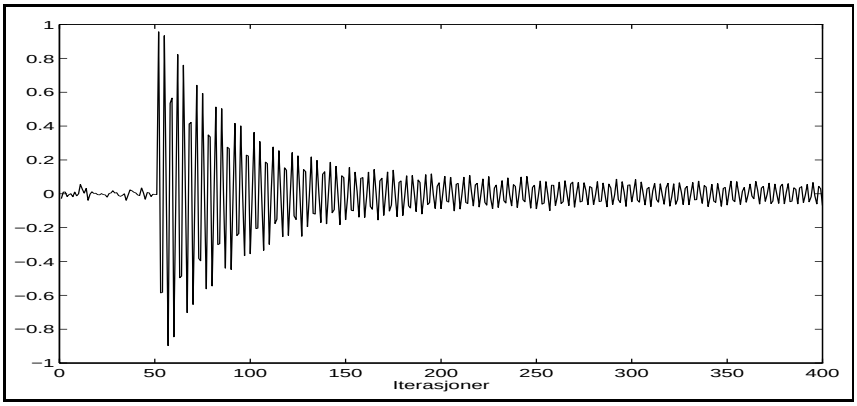
4.5 Simulering av GAL algoritmen

Hittil er det simulert med to algoritmer med sin basis i det transversale filteret, beskrevet i avsnitt 2.1. Nå vil gitter-filteret, beskrevet i avsnitt 2.2, bli behandlet. Den metoden som fremsettes er den minste midlere kvadratiske metode omarbeidet for gitter-filteret; GAL algoritmen. Teorien for denne metoden er vist i avsnitt 2.2.2, med den resulterende algoritmen gitt i tabell 2.3. Matlab programmet for GAL algoritmen er vist i appendiks B.3. Som i de foregående metodene vil algoritmen uttestes med forskjellig filterorden og konvergeringsfaktor.

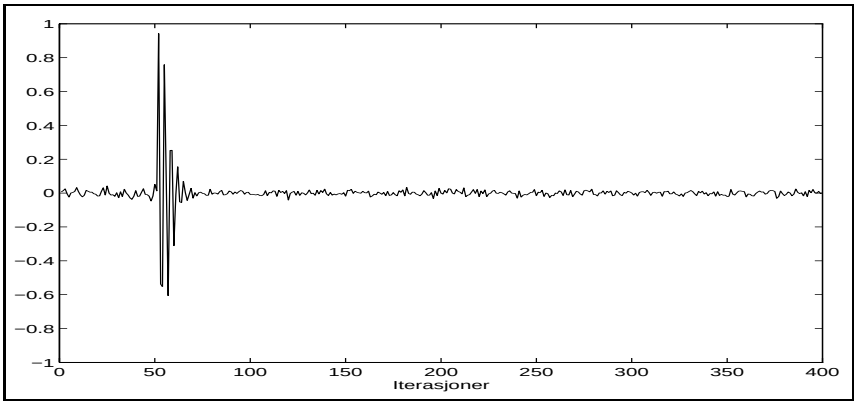
4.5.1 Konvergensteori for GAL algoritmen

Konvergensteorien for GAL er betydelig mer avansert enn for de foregående metodene, på grunn av en sterk ulineær karakter ved adapsjonen. Det vil derfor ikke gåes dypere inn i denne delen av materien, men følgende utsagn vil kunne gi en pekepinn på algoritmens konvergeringsegenskaper:

> The successive orthogonalization provided by the lattice offers advantages in adaptive convergence rate which cannot be achieved with tapped-delay-lines. [Griffiths 78]



Figur 4.13: Plott av signalet for GAL N=10, konvergeringsfaktor=0.01



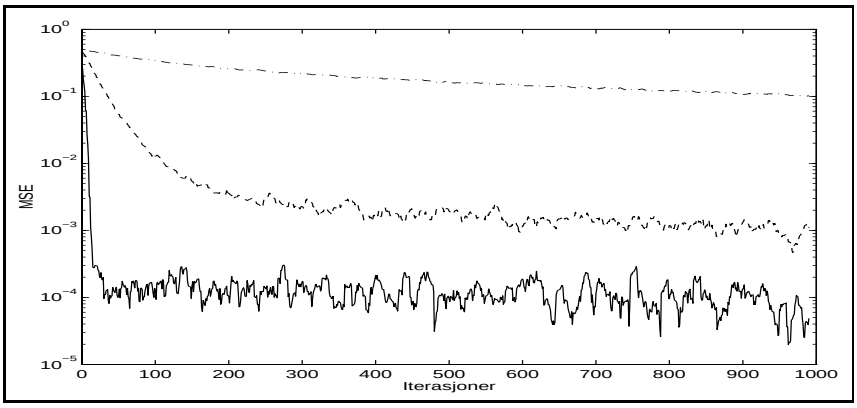
Figur 4.14: Plott av signalet for GAL N=10, konvergeringsfaktor=0.1

> The primary benefit of using a lattice filter joint process estimator is that the successive orders of backward prediction error are orthogonal after convergence of the lattice coefficients, and therefore the joint process coefficients adapt more quickly since the interaction of the coefficient adaptions are eliminated. [Honig & Messerschmitt 84]
>
> ... the convergence behavior of the GAL algorithm is somewhat more rapid than that of the LMS algorithm. [Haykin 91]

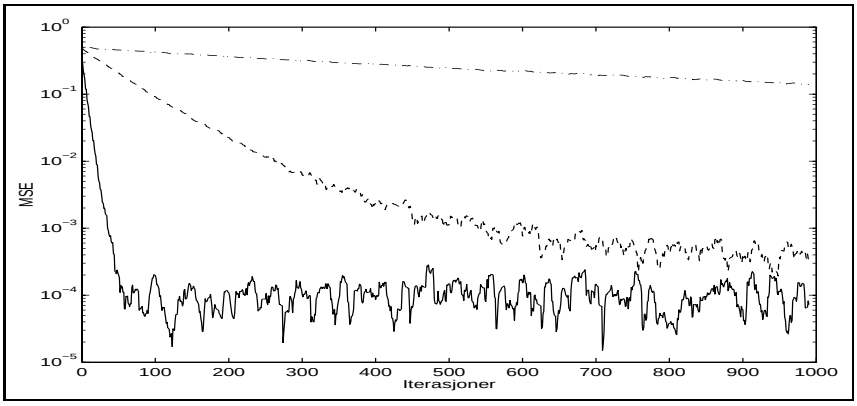
For å sammenfatte disse sitatene kan man si at gitter filterets ortogonalisering gir raskere konvergens enn ved transversalt filter.

4.5.2 Kommentarer til simuleringer med GAL algoritmen

Som fortalt tidligere, er det utført simuleringer med GAL algoritmen med de samme variabelsettingene som i simuleringene med LMS algoritmen. Dermed kan man gjøre direkte sammenligninger mellom disse to algoritmene, for å underbygge påstanden nevnt i forrige avsnitt 4.5.1, om at GAL algoritmen konvergerer raskere enn LMS.



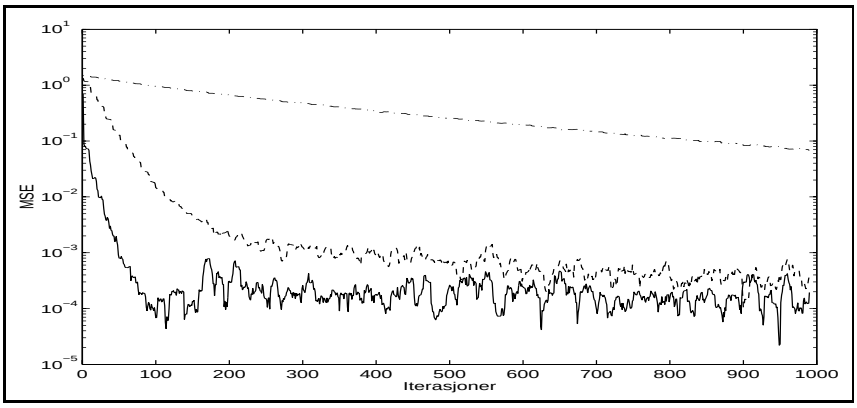
Figur 4.15: Plott av MSE for GAL N=10, konvergeringsfaktor=0.1 (-), 0.01 (- -), 0.001 (-.).



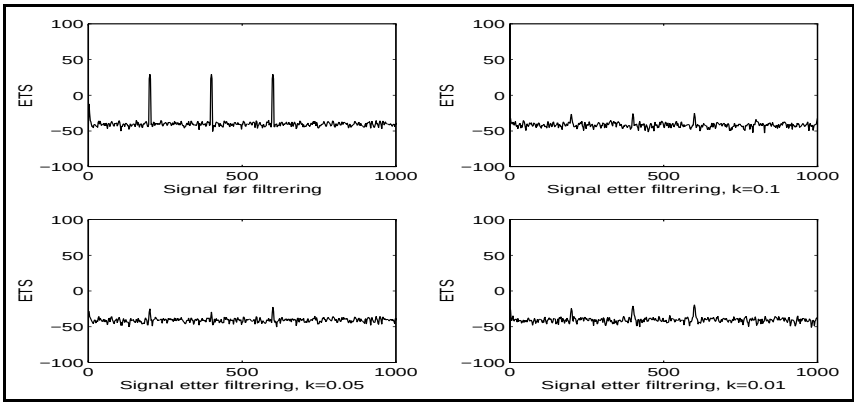
Figur 4.16: Plott av MSE for GAL N=3, konvergeringsfaktor=0.1 (-), 0.01 (- -), 0.001 (-.).

De to første plottene i figur 4.13 og 4.14 viser signalet ut av systemet etter gjennomført filtrering med filterorden $N = 10$. Man ser tydelig den transiente oppførsel i begge figurer, og at algoritmen etterhvert stabiliserer seg , henholdsvis etter ca. 120 iterasjoner med konvergeringsfaktor lik 0.01 og 20 iterasjoner med konvergeringsfaktor lik 0.1. Sammenligner man disse med simuleringene med LMS algoritmen i figurene 4.1 og 4.2, vil man vanskelig kunne bekrefte påstanden om at GAL konvergerer raskere enn LMS. Spesielt i plottet av simulering med konvergeringsfaktor 0.01 er det vanskelig å se dette forholdet. Dessuten har GAL algoritmen en betydelig større feiljustering i figur 4.13.

Hvorfor konvergerer ikke GAL algoritmen så raskt som forventet? En effekt ved gitter filteret er følgende [Haykin 91]; hvis inngangsdata inn til gitter-prediktoren er støyende (sterk hvit støy komponent) blir prediksjonsfeilen ut av filteret stor. Dette medfører at parameteren $\xi(n)$ blir stor, og videre at gitter-filterets konvergeringsfaktor (oppdatering av Γ) blir ekvivalent mindre, se tabell 2.3. Ved undersøkelse av refleksjonskoeffisientene under simuleringene, viser det seg at denne verdien er liten og ligger jevnt i området 10^{-3}. I dette aktuelle simuleringstilfellet er det ingen hvit støy komponent i signalet inn til gitter-prediktoren. En mer fruktbar forklaring på forholdet mellom LMS og GAL algoritmene kan



Figur 4.17: Plott av MSE GAL for tre støysignaler, N=10, konvergeringsfaktor=0.1 (-), 0.01 (- -), 0.001 (-.).

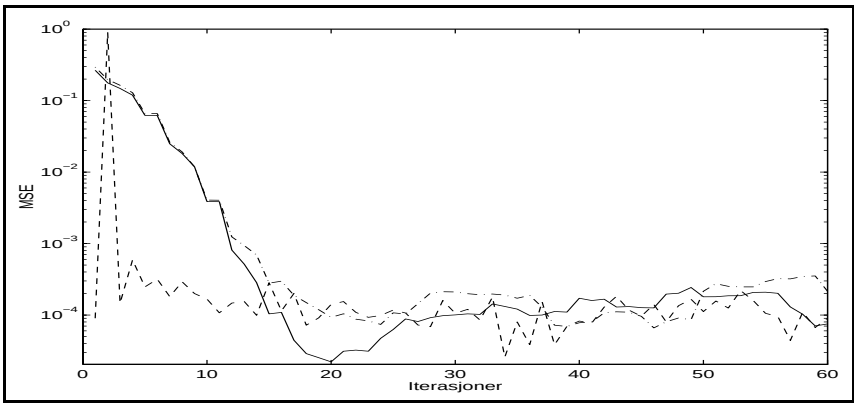


Figur 4.18: Plott av ETS for GAL, N=10

finnes ved å se på inngangssignalene. Støyen i referansesignalet består av en sinuskomponent. Dette signalet krever lite arbeid av filteret for å justere fasen og amplituden slik at utsignalet fra filteret blir et estimat av støyen i primærsignalet. Det er derfor mer riktig å si at det er LMS algoritmen som har optimal ytelse, enn å si at GAL algoritmen er mindre rask enn forventet.

Figur 4.15 og 4.16 gir en oversikt over midlere kvadratfeil for henholdsvis $N = 3$ og $N = 10$, og varierende konvergeringsfaktor. Som antatt konvergerer midlere kvadratiske feil raskere ved økende konvergeringsfaktor og økende filterorden N, men sammenlignet med LMS algoritmen, representert ved figurene 4.3 og 4.4 på side 42, har midlere kvadratiske feil en betydelig større feiljustering for konvergeringsfaktor mindre enn 0.1.

Simuleringen med en modell med tre støysignaler gir et forventet resultat, med tanke på forholdet mellom konvergeringsfaktorene. Derimot, i sammenligning med GAL simuleringen i figur 4.15, har man motstridende resultater. Konvergeringsfaktor 0.1 gir nå et dårligere resultat når det gjelder både adapsjonshastighet og feiljustering, mens 0.01 og 0.001 gir et bedre resultat. Det er vanskelig å si noe ut over dette uten en mer omfattende analyse.



Figur 4.19: Plott av alle algoritmene, N=10, konvergeringsfaktor=0.1. LMS(-), GAL(.-), RLS(- -)

Dette anses å ligge utenfor oppgavens definerte område.

Figur 4.18 viser effekttetthetsspekteret etter adapsjon for filterorden lik 10 og varierende konvergeringsfaktor. Det vises tydelig at GAL algoritmen ikke har klart å filtrere bort støysignalene fullstendig. Dette kan sees i sammenheng med generelt høy feiljustering for GAL algoritmen.

4.6 Konklusjon på simulering med stasjonære data

I dette kapittelet er det tatt utgangspunkt i teorien for adaptiv filtrering og det er implementert tre filtre i matlab for uttesting med påtrykk av syntetiske signaler. Teorien har gitt en pekepinn på forholdet mellom disse algoritmene, og en slik analyse har vært viktig for å kunne gi en bekreftelse på hvordan algoritmene er i forhold til hverandre.

To forhold ble satt opp i introduksjonen i avsnitt 2.0.2. Det er forholdet LMS vs RLS og forholdet transversal filter vs gitter-filter. Når det gjelder det først forholdet, var spørsmålet; konvergerer RLS så mye raskere enn LMS? Svaret på dette er utvilsomt ja. Figur 4.11 gir et godt forhold mellom disse algoritmene. Generelt sett kan man si at resultatet av simuleringene med disse to algoritmene stemmer med teorien.

Forholdet mellom transversalt filter og gitter-filter er representert ved LMS og GAL algoritmene. Teorien ga et utgangspunkt for å tro at GAL ville konvergere en god del raskere. Dette er ikke bekreftet i de utførte simuleringer. I følge simuleringene har LMS og GAL algoritmene bortimot lik konvergensrate, og LMS har mindre feiljustering enn GAL algoritmen. Årsaken til dette ligger i at de påtrykte støysignalene er enkle sinuskomponenter. Dette resulterer i en utjevning i ytelsesforholdet mellom transversalt filter og gitter-filter.

Avslutningsvis er det med et plott i figur 4.19 av simuleringer med alle tre algoritmer, alle med optimale variable. Plottet gir et godt bilde av hva som er sagt hittil i dette avsnittet.

I egenskap av konvergensanalyse med stasjonære data gir ikke dette kapittelet et helhetlig bilde av forholdet mellom algoritmene. Andre faktorer, som algoritmenes kompleksitet, stabilitet og de reelle signalenes beskaffenhet må, og vil også vurderes senere i oppgaven. Men nå har man en basis for å gå videre til neste kapittel, som omhandler analyse av algoritmene ved påtrykk av reelle signaler.

Kapittel 5

Analyse av simulering med reelle signaler

Avsnitt i dette kapittelet

I kapittel 2 ble teorigrunnlaget for adaptive algoritmer gjennomgått. 3 algoritmer ble valgt ut til å representere det store utvalget:

- Minste midlere kvadrats metode. (LMS)
- Rekursiv minste kvadrats metode. (RLS)
- Gradient adaptiv gitter metode. (GAL)

Disse algoritmene ble gjenstand for analyse av konvergensegenskapene i et stasjonært miljø i kapittel 4. Dette resulterte i en god oversikt over forholdet mellom disse algoritmene.

I dette kapittelet vil det i simuleringene bli tatt utgangspunkt i opptakene av sonarsignaler gjort ombord på FFI-U's forskningsskip H. U. Sverdrup. Disse opptakene ble studert nærmere i kapittel 3, der det ble konkludert med en passende plassering for referansesignalet til støykansellereren. Det ble i tillegg gjort opptak av sonarsignalene mens sonaren opererte i aktiv modus. Dette gir en mulighet til å se hvordan den adaptive prosessen blir påvirket av pulsutsendingene. Det ble i kapittel 3 konkludert med at følgende frekvenskomponenter utgjør støybidraget; 312.5, 625, 937.5 og 1250Hz.

5.1 Algoritmer brukt i adaptiv støykansellering

De tre aktuelle algoritmene er gitt i tabellene 2.1, 2.2 og 2.3. Algoritmene ble utledet med tanke på et stasjonært miljø, det ble derfor ikke gjort store forandringer på disse ved simulering med syntetiske signaler. Derfor forklares nå de forandringene som må gjøres ved reelle, ikke-stasjonære signaler, inkludert initialisering av variabler.

5.1.1 LMS algoritmen

Det er ikke stort man trenger å forandre på ved LMS algoritmen, men det man må gjøre er å dempe oppdateringen av filterkoeffisientene som funksjon av inngangssignalet. Størrelsen på filterkoeffisientoppdateringen kan holdes omtrent på samme gjennomsnittlige størrelsesnivå hvis filterkoeffisientene er normalisert ved et estimat av inngangseffekten $\hat{\sigma}_x^2 = \mathbf{x}_k^T \mathbf{x}_k$. Dette er nødvendig i simulering med reelle ikke-stasjonære signaler, i motsetning til i simulering med syntetiske stasjonære signaler, fordi signalet x_k hele tiden varierer i amplitude. I [Bellanger 87] er det vist at man kan bruke følgende oppdatering for konvergeringsfaktoren.

$$\mu = \frac{2\mu}{\alpha + \hat{\sigma}_x^2} \tag{5.1}$$

der α er et lite positivt tall som sikrer at oppdateringen blir stor når $\hat{\sigma}_x^2$ blir liten. Den praktiske oppdateringen av estimatet av effekten kan skrives som

$$\hat{\sigma}_{k+1}^2 = (1 - \beta)\hat{\sigma}_k^2 + N\beta x_k^T x_k \tag{5.2}$$

der $\beta = 1 - \frac{1}{N}$ og N er filterorden. Filterkoeffisientene oppdateres dermed på følgende vis.

$$\mathbf{a}_{k+1} = \mathbf{a}_k + \frac{2\mu}{\alpha + \hat{\sigma}_{k+1}^2} \varepsilon_k \mathbf{x}_k \tag{5.3}$$

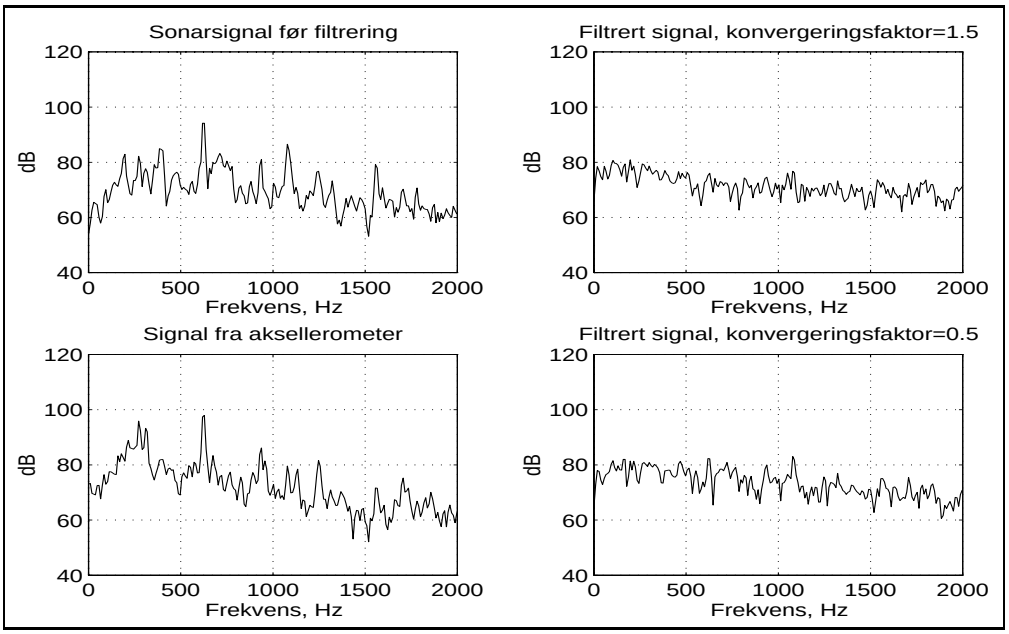
Ellers er algoritmen lik som i tabell 2.1.

En teori man kan fremsette er at det er behov for at filterkoeffisientene reagerer raskere på forandringer i signalene. Dette kan en til en viss grad styre ved å bruke høyere verdier for konvergeringsfaktoren μ. Som det ble vist i avsnitt 4.3 har man en avveining mellom rask konvergens og liten feiljustering, slik at man får større feiljustering på MSE av utsignalet ved å velge høyere μ.

5.1.2 RLS algoritmen

Heller ikke ved RLS algoritmen er det stort som må forandres. Det er nødvendig å initialisere autokorrelasjonsmatrisen $\mathbf{P}$ i RLS algoritmen. Det er to måter å gjøre det på. Enten kan man lese inn nok verdier til inngangsvektoren $\mathbf{x}_k$, regne ut autokorrelasjonsmatrisen og sette $\mathbf{P}$ lik dette, eller så kan man sette $\mathbf{P}$ lik identitetsmatrisen ganget med en variabel med stor verdi. Det er i oppgaven brukt siste metode, fordi man da vil kunne utføre adapsjonen allerede fra første iterasjon.

Den såkalte glemmefaktoren, λ, hadde en verdi lik 1 under simulering i det stasjonære miljøet. Nå vil det være nødvendig å vektlegge tidligere signaler mindre, derfor vil det bli simulert med $\lambda < 1$, og dermed gis algoritmen endelig minne.
RLS algoritmen er ellers lik som i tabell 2.2.



Figur 5.1: Plott av ETS for LMS, N=10

5.1.3 GAL algoritmen

For å overføre GAL algoritmen til bruk for et ikke-stasjonært miljø, må man gjøre visse forandringer. For det første går det ut på at man modifiserer uttrykket for ζ i ligning 2.62 side 24, som beskriver et estimat av den totale energi i forlengs og baklengs prediksjonsfeil. Den vil nå bli skrevet på formen [Griffiths 78] [Haykin 91];

$$\zeta_k(n-1) = \beta\zeta_{k-1}(n-1) + (1-\beta)(|f_k(n-1)|^2 + |b_{k-1}(n-1)|^2) \tag{5.4}$$

der man bruker et en-pol lavpass filter, der $0 < \beta < 1$ kontrollerer båndbredden på filteret og den resulterende effekttidsmidlingen. β gir GAL algoritmen endelig minne, som hjelper den å håndtere statistiske variasjoner når den opererer i et ikke-stasjonært miljø.

Den andre forandringen gjelder oppdatering av filterkoeffisientene, der man, som i LMS algoritmen, normaliserer konvergeringsfaktoren med et uttrykk

$$\sigma_{k+1}^2 = \beta\sigma_k^2 + (1-\beta)(\mathbf{b}_k\mathbf{b}_k^T) \tag{5.5}$$

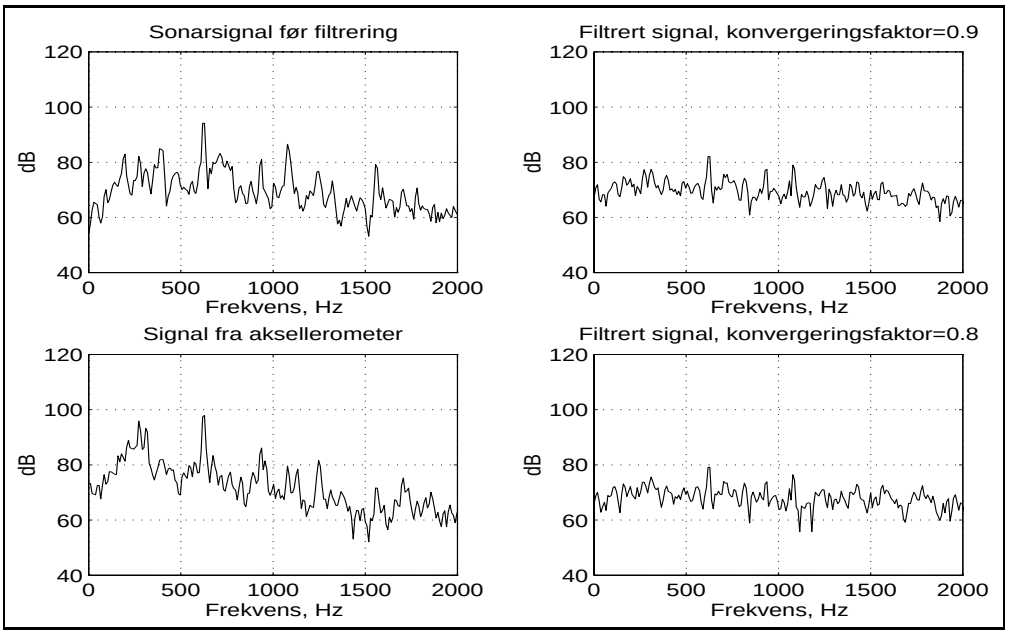
Ligningen for oppdatering av filterkoeffisientene blir

$$\mathbf{a}_{k+1} = \mathbf{a}_k + \frac{2\mu}{\sigma_{k+1}^2}\varepsilon_k\mathbf{b}_k \tag{5.6}$$

Ellers er algoritmen som vist i tabell 2.3.

5.2 Kommentar til simulering med reelle signaler

Hvordan kan man gjøre vurderinger av simuleringenes resultater? Vil det adaptive støykanselleringssystemet filtrere støysignalene, og vil prosessen i så tilfelle kunne forringe det ønskede signal? Et fullverdig svar kan ikke gis uten en test av den adaptive støykansellereren



Figur 5.2: Plott av ETS for RLS, N=10

på den parametriske sonaren. En slik test vil relativt raskt kunne avdekke problemer ved den adaptive algoritmen. Dette vil også kunne bidra til å motivere til videre teoretiske undersøkelser av algoritmer og filtertyper.

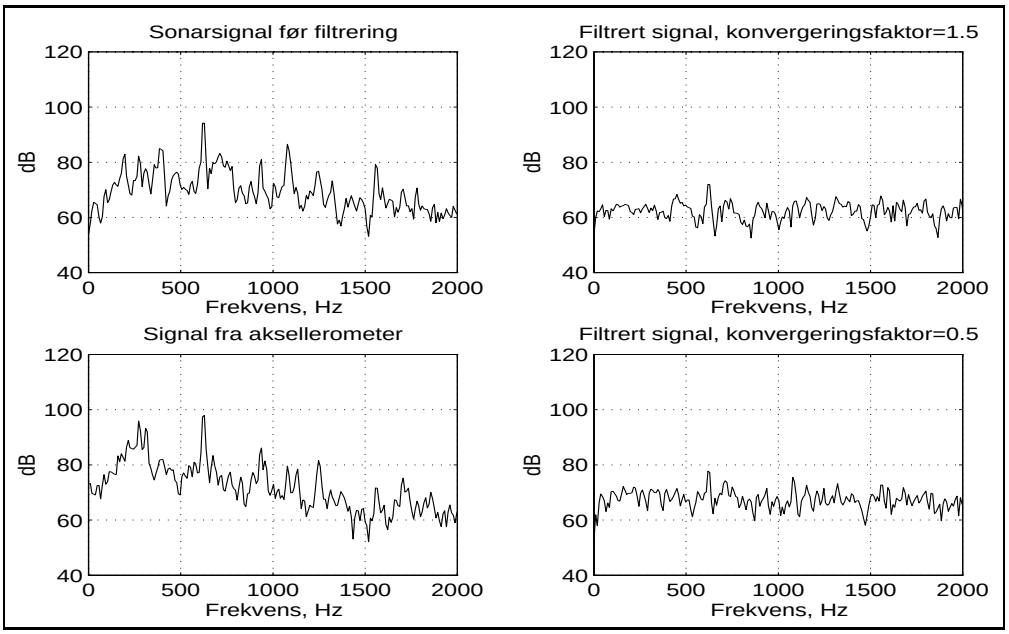
Hva kan man så i dette kapittelet trekke ut av simuleringer av den adaptive prosessen med reelle signaler? Plott av effekttetthetsspekteret av signalene ut av støykanselleringssystemet vil kunne gi en pekepinn på om de forskjellige algoritmene klarer å filtrere støykomponentene. Selv om signalene per definisjon er ikke-stasjonære, kan man utføre plott av ETS, fordi man definerer signalene av interesse (støykomponentene) som stasjonære innenfor den aktuelle datalengden. Signalene av interesse ligger i frekvensområdet 500 til 1000Hz, og det vil derfor konsentreres om å se hvordan disse støykomponentene filtreres.

Det vil bli gjort en undersøkelse av filterkoeffisientene for et begrenset iterasjonsområde, for om mulig å sammenligne konvergensegenskapene til algoritmene. Til slutt vil sonarsignalet ut av det adaptive støykanselleringssystemet bli vurdert når den parametriske sonaren kjører i aktiv modus. Dette vil gi en viss bakgrunn for å gi svar på hvordan den adaptive filtreringsprosessen vil påvirke sonarsignalet i praksis.

Plott av ETS, sonar i passiv modus

For en beskrivelse av ETS, se avsnitt 3.2.1 side 27. I figurene 5.1, 5.2 og 5.3 er det vist plott av ETS for henholdsvis de tre algoritmene LMS, RLS og GAL. Signalene som er vist er primærsignalet, referansesignalet og to forskjellige signaler ut av filteret med forskjellig konvergeringsfaktor. Alle simuleringene er gjort med filterlengde $N = 10$.

I figur 5.1 kan man se hvordan signalet oppfører seg etter filtrering med LMS algoritmen med konvergeringsfaktor 0.5 og konvergeringsfaktor 1.5. Det vil med ikke-stasjonære signaler være behov for at algoritmen skal ha rask konvergens, fordi signalene varierer i amplitude i stor grad. Det er tydelig på utsignalets beskaffenhet i figur 5.1 at filteret fint



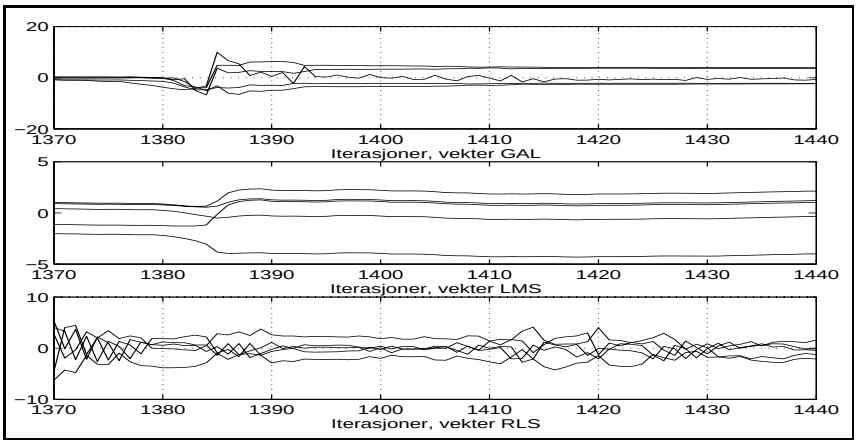
Figur 5.3: Plott av ETS for GAL, N=10

klarer å dempe støykomponentene. Det virker som at man får et bedre resultat ved konvergeringsfaktor lik 1.5, men dette vil nok også gi en noe større feiljustering, se avsnitt 4.3 side 39.

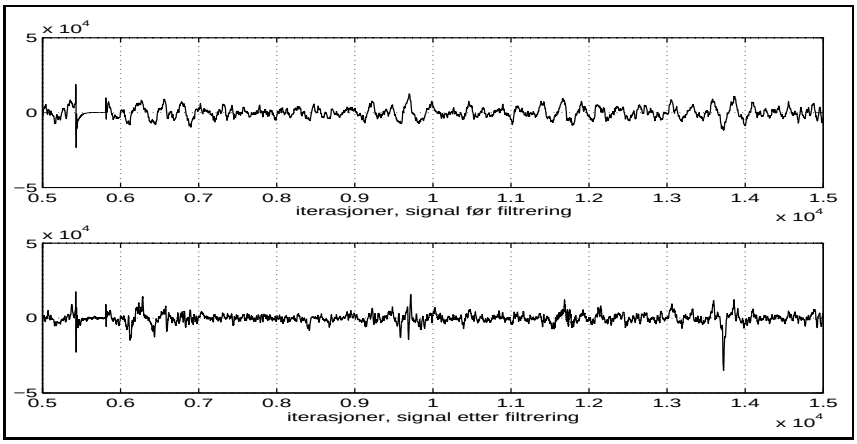
ETS fra kjøring med RLS algoritmen er vist i figur 5.2. Variabelen λ i RLS algoritmen er et utrykk for algoritmens hukommelse. I stasjonære miljøer benytter man $\lambda = 1$, mens her må man redusere effekten av eldre signaler. Derfor er det kjørt simuleringer med $\lambda = 0.8$ og 0.9. Det ble også gjort forsøk med lavere λ verdi, men dette førte til at feilsignalet divergerte. Plottene i figur 5.2 viser at støykomponentene ikke er filtrert bort. På bakgrunn av de gode resultatene med RLS algoritmen ved simulering med stasjonære signaler, kunne man forventet et bedre resultat. Følgende bekrefter dette dårlige resultatet.

> With both algorithms tuned to minimize the misadjustment at the filter output by a proper optimization of their forgetting rate, the LMS algorithm is found to have a superior tracking performance compared to the RLS algorithm. [Haykin 91]

I figur 5.3 er det vist ETS fra simuleringene med GAL algoritmen. Det er simulert med konvergeringsfaktor lik 0.5 og 1.5. Plottene viser at støykomponentene ikke er helt filtrert bort. Til gjengjeld har signalet med konvergeringsfaktor 1.5 et generelt lavere støygulv enn i de andre simuleringene. GAL algoritmen vil fungere best hvis inngangssignalet x_k er korrelert med seg selv. Er ikke dette tilfellet, vil refleksjonskoeffisientene bli tilnærmet lik null, og filteret reduseres til et transversalt filter. Et signal som har høy korrelasjon med seg selv er en sinus, et signal som er sterkt ukorrelert er hvit støy. Det aktuelle signalet x_k er en mellomting mellom disse, og dette kan derfor være årsaken til at GAL ikke fjerner støykomponentene helt.



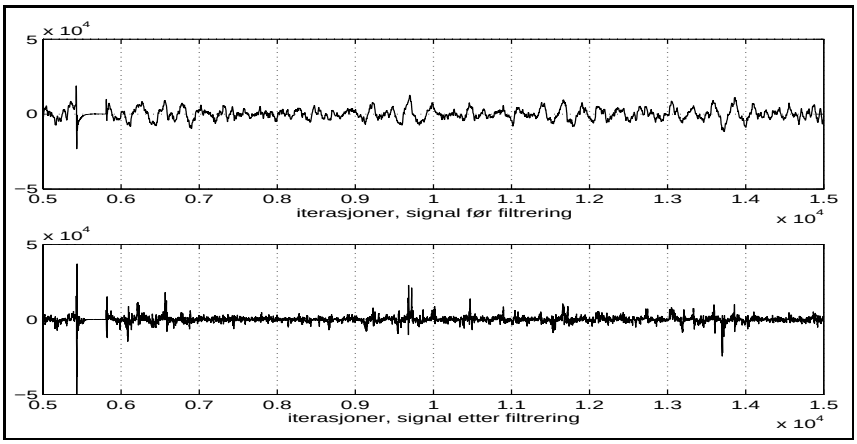
Figur 5.4: Plott av filterkoeffisienter for alle algoritmer



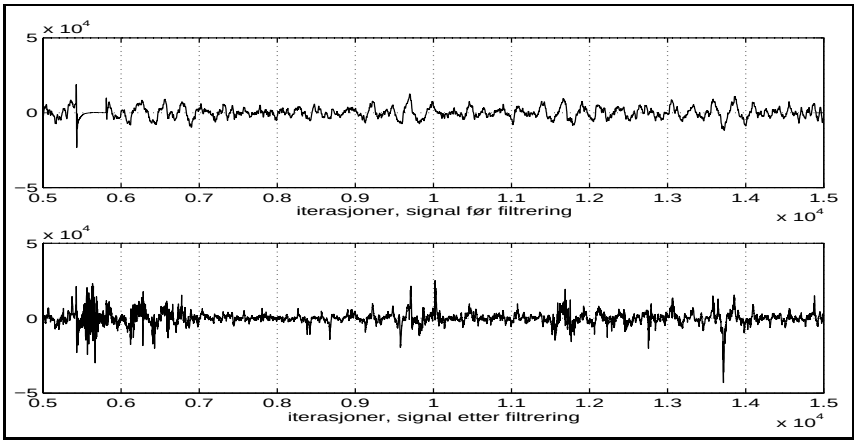
Figur 5.5: Plott av signalet for LMS med aktiv puls, N=10, konvergeringsfaktor=0.5

Plott av filterkoeffisientene, sonar i passiv modus.

Tidligere i oppgaven er det forklart at filterkoeffisientene kan brukes på samme måte som midlere kvadratfeil for å se på konvergensegenskapen. Det ble da nevnt at analyse med hensyn på filterkoeffisientene kunne være aktuelt med ikke-stasjonære signaler. I figuren 5.4 side 58 ser man plott av filterkoeffisientene for henholdsvis LMS, RLS og GAL algoritmen, og det er tatt et utsnitt fra ett iterasjonsområde. For oversiktens skyld er det simulert med filterorden $N = 5$. Det er imidlertid vanskelig å se noe fornuftig i forhold til konvergensegenskaper ut fra plottene. Man ser i figur 5.4 hvordan filterkoeffisientene for LMS algoritmen og GAL algoritmen reagerer og justerer seg inn på en ny verdi. RLS algoritmens filterkoeffisienter gir ikke et slikt forløp.



Figur 5.6: Plott av signalet for RLS med aktiv puls, N=10, glemmefaktor λ=0.85



Figur 5.7: Plott av signalet for GAL med aktiv puls, N=10, konvergeringsfaktor=0.5

Plott av sonarsignalet, sonar i aktiv modus

I kapittel 3 er det vist hvordan signalene fra sonar og referanseføler ble tatt opp. De siste opptakene ble gjort med sonaren i aktiv modus. Pulser ble sendt ut med forskjellig repetisjonsrate og amplitude. Det signalet som nå skal behandles, hadde følgende karakteristika,

- Ricker puls, 500 Hz
- 1 sekund repetisjonsrate
- 100 % amplitude

I figurene 5.5, 5.6 og 5.7 er det vist signalene fra henholdsvis LMS, RLS og GAL algoritmene. Plottene viser fra toppen; primærsignalet før filtrering, og utsignalet etter filtrering. Det ble gjort simuleringer for å vurdere om pulsutsendelsen påvirket den adaptive prosessen negativt. Disse forsøkene viste at dette ikke skjer i noen av simuleringene. Man kan derfor

holde den adaptive prosessen igang hele tiden. Plott fra simuleringene er vist i appendiks D.

I figur 5.5 er det vist plott av signalet etter filtrering med LMS algoritmen, filterorden er lik 10 og konvergeringsfaktor er lik 0.5. Man ser at enkelte signaler har kommet tydeligere frem. Det kan tyde på at dette er ønsket signal, og i så måte ser disse resultatene lovende ut. Den høyfrekvente komponenten som er overlagret signalet, skyldes algoritmens fluktuering rundt prestasjonsflatens minimumspunkt.

I plott av RLS algoritmen i figur 5.6 ser man noe av det samme som med LMS algoritmen. Verdt å merke seg er mer høyfrekvent støy i signalet. Plottet av signalet fra GAL algoritmen i figur 5.7 er det derimot flere topper. GAL algoritmen har større problemer med å gi et godt bilde av det ønskede signalet.

Dybden varierte under kjøringene, men man kan regne ut dybden hvis man identifiserer det første ekkoet fra bunnen ved formelen [Johnson & Dudgeon 93]

$$r = \frac{tc}{2} \tag{5.7}$$

det t er tiden fra pulsutsendelse til mottak av reflektert puls og c er lydhastigheten i vann.

I figurene ser man at pulsen sendes ut ved tid lik iterasjon 5400. Det første returnerte ekkoet er ved iterasjon 6300. Ved et samplingsintervall lik $\frac{1}{48000}$ sekunder har man at tid t fra pulsutsendelsen til mottak er 0.019 sekunder. Siden lydhastigheten i vann er 1498 m/s, er dybden fra sonar til bunn lik 28 meter. En verifisering av dette er ikke mulig, men det kan tyde på at dybden er riktig siden opptakene ble gjort innaskjærs ved Bergensområdet.

5.3 Konklusjon av simulering med reelle signaler

Med bakgrunn i analysen av algoritmenes konvergensegenskaper, vist i kapittel 4, er det i dette kapittelet søkt å finne svar på om algoritmene er i stand til å filtrere sonarsignalets støykomponenter, samt å sammenligne algoritmene ved reelle ikke-stasjonære signaler.

Det er hentet utsagn fra litteraturen som gir et visst innblikk i algoritmenes egenskaper i et ikke-stasjonært miljø. Disse utsagnene har blitt bekreftet i simuleringene utført med reelle sonarsignaler. Resultatene fra simuleringene kan oppsummeres slik;

- Plottene av ETS i figurene 5.1, 5.2 og 5.3 viser at kun LMS algoritmen klarer å kansellere støykomponentene helt i frekvensområdet 500-1000Hz. Støykomponentene har i simuleringene med RLS og GAL algoritmen ikke blitt fjernet helt.
- Plott av filterkoeffisientene i figuren 5.4 gir lite informasjon om konvergeringsegenskapene.
- Plott av sonarsignalet etter filtrering i figurene 5.5, 5.6 og 5.7 viser at det ønskede signalet blir tydeligere. (Vises som topper i signalet). Den adaptive prosessen har redusert signaltopper som skyldes støykomponentene, slik at det returnerte bunnekkoet blir tydeligere. LMS og RLS algoritmen viser her bedre resultater enn GAL algoritmen.

Man kan dermed ut ifra disse simuleringene konkludere med at LMS algoritmen er den best egnede algoritmen for den adaptive støykansellereren. Det må poengteres at det fortsatt gjenstår tester av støykansellereren på den parametriske sonaren for å sikre at det ønskede signalet ikke er forringet av den adaptive prosessen. Andre faktorer som også er med på å bestemme algoritmevalget vil bli diskutert i neste kapittel.

Kapittel 6

Valg av algoritme

Avsnitt i dette kapittelet

I avsnitt 2.0.2 ble det gjort et valg av tre algoritmer. Disse representerte det brede spekteret av algoritmer og filterstrukturer, som etterhvert er blitt utviklet. Simuleringene av de adaptive algoritmene i kapittel 4 og 5 ga en god innføring i algoritmenes egenskaper. Sammen med egenskaper som kompleksitet, robusthet og numeriske egenskaper, forhold diskutert tidligere i oppgaven, vil dette danne grunnlaget for valg av en algoritme for den adaptive støykansellereren.

6.1 Kriterieundersøkelse for valg av algoritme

De forskjellige kriteriene for valg av algoritme for den adaptive støykansellereren kan forklares på følende måte;

- **Konvergens.** Den tid det tar for algoritmen å minimalisere feilsignalet.
- **Sporingsevne.** Algoritmens evne til å følge signaler som varierer i tid (ikke-stasjonære signaler).
- **Feiljustering.** Den relative avvikelsen fra minstefeilen.
- **Numeriske egenskaper.** Hvordan algoritmen kontrollerer avrundingsfeil som skjer i adapsjonsprosessen.
- **Robusthet.** Hvordan algoritmens følsomhet er ovenfor inngangssignalene.

- **Kompleksitet.** Hvor stort regnearbeide algoritmen representerer.

De fleste av disse punktene er nevnt i løpet av oppgaven. En viss grad av vektlegging av punktene kan sies å forekomme. Det er vanskelig å sette et tall på vektleggingen, men nedenfor vil det i hvert tilfelle antydes en vektlegging i forhold til den aktuelle applikasjonen.

Konvergens

Det er en viktig egenskap for algoritmen å konvergere raskt. Analysen av algoritmenes konvergensegenskaper i kapittel 4 viste at RLS algoritmen konvergerte raskere enn LMS og GAL algoritmen. I den aktuelle støykanselleringsapplikasjonen er det imidlertid ikke et krav til rask konvergens, slik at alle algoritmene fint tilfredstiller dette kravet.

Sporingsevne

Sporingsevne kan oversettes som konvergens i et ikke-stasjonært miljø, og vektlegges av den grunn mer enn konvergens. Det ble i kapittel 5 gitt henvisninger fra litteraturen som viser at RLS algoritmen mister sin konvergensfordel når den arbeider med ikke-stasjonære signaler. Det viste seg vanskelig å få en direkte sammenligning av algoritmenes konvergenshastighet ved ikke-stasjonære signaler. De simuleringene som ble utført viste at LMS algoritmen filtrerte bort støykomponentene best. Både RLS og GAL algoritmene hadde problemer med å fjerne støykomponentene.

Feiljustering

I teorien skal RLS algoritmen ha null feiljustering, men dette holder ikke ved bruk av eksponensielt avtagende vindu, som er nødvendig ved ikke-stasjonære signaler. LMS algoritmen har en feiljustering gitt av ligning 4.8, der det er tydelig at det skjer en avveining mellom rask konvergens og liten feiljustering. GAL algoritmen skal i teorien ha minimal feiljustering ved stasjonære signaler, på grunn av ortogonaliseringen av baklengs prediksjonsfeil. Ved ikke-stasjonære signaler kan man tolke den høyfrekvente komponenten i sonarsignalet etter filtrering i figurene 5.5, 5.6 og 5.7 som et uttrykk for feiljustering. Alle algoritmene har en slik høyfrekvent komponent. Siden det er vanskelig å gi et klart uttrykk for feiljustering, vil ikke dette punktet få større betydning.

Numeriske egenskaper

Dette punktet er hittil kun kort nevnt, men det er likevel viktig å være bevisst problemer som kan oppstå grunnet algoritmenes numeriske egenskaper. Et klart problem er at det oppstår avrundingsfeil ved de aritmetiske operasjonene i det digitale systemet. LMS algoritmen kan godt brukes med heltallsaritmetikk, men har også sine tilfeller der avrundingsfeil kan oppstå [Lebeda 90]. Ved LMS algoritmen kan det også oppstå problemer basert på at det digitale systemet har endelig ordlengde for å representere vektene [Ifeachor & Jervis 93].

RLS algoritmen er også sensitiv i forhold til avrundingsfeil i det digitale systemet. Dette kan resultere i negativ definit $\mathbf{P}$ matrise (autokorrelasjonsmatrisen til inngangssignalet x_k) og dermed ustabilitet [Ifeachor & Jervis 93]. I avsnitt 2.1.2 side 17 ble det nevnt en løsning på dette problemet; $\mathbf{UDU}^T$ faktorisering av $\mathbf{P}$.

Det er ikke funnet noen henvisninger til GAL algoritmens numeriske egenskaper i litteraturen, men man kan gå ut ifra at avrundingsfeil vil også her kunne ha en negativ effekt på algoritmens optimale egenskaper.

Robusthet

Har man spesielle inngangssignal til LMS algoritmen (for eksempel smalbåndede signaler) kan filterkoeffisientene flytte fra en optimal verdi, vokse, og til slutt overstige den tillate ordlengden. Dette kan løses ved å introdusere en lekkasjefaktor i oppdateringen av filterkoeffisientene. [Lebeda 90], [Ifeachor & Jervis 93].

RLS algoritmen vil divergere hvis man på inngangen påtrykker en lang sekvens nuller. Da vil inngangssignalets autokorrelasjonsmatrise $\mathbf{P}$ vokse eksponensielt som følge av divisjon med γ, se tabell 2.2 [Ifeachor & Jervis 93].

I avsnitt 4.5.2 ble det nevnt et tilfelle der GAL algoritmens egenskaper drastisk synker. Hvis inngangsignalet x_k er ukorrelert med seg selv, vil gitterfilteret i praksis reduseres til et transversalt filter.

Robusthet er ikke av de mest vektlagte punktene, men man må allikevel være oppmerksom på de aktuelle tilfellene der de nevnte problemer kan oppstå.

Kompleksitet

Generelt kan man si at det alltid er å foretrekke en algoritme med lav kompleksitet. Dette medfører at man har flere valgmuligheter i implementasjonen av systemet og i valg av samplingsfrekvens. Følgende gir en oversikt over de tre algoritmenes kompleksitet;

- **Minste midlere kvadrat metoden.** $2N + 1$
- **Rekursiv minste kvadrat metoden.** $4N^2 + 5N + 2$
- **Gradient adaptiv gitter.** $14N$

LMS har lav kompleksitet, mens RLS har kvadratisk kompleksitet. GAL har kompleksitet i mellom de to nevnte. RLS algoritmen har helt klart en ulempe ved sin høye orden.

6.2 Valg basert på simuleringer og kriterieundersøkelse

For det videre arbeid velges LMS algoritmen. Gjennom oppgavens simuleringer har algoritmen vist seg å være stabil med tilstrekkelig gode konvergensegenskaper. Kriterieundersøkelsen har vist at LMS algoritmen ikke har utpregede negative egenskaper eller problemer. Spesielt viste simuleringene med reelle signaler i kapittel 5 at algoritmen er best egnet for det adaptive støykanselleringssystemet.

RLS algoritmen konvergerte raskt med stasjonære signaler, men denne fordelen synes å være svekket i det ikke-stasjonære miljøet, sannsynligvis på grunn av ikke-optimal koherens mellom primærsignalet og referansesignal, og ikke optimal bruk av vektfaktor λ. Den kvadratiske kompleksiteten og ustabiliteten gjør også til at denne algoritmen ikke er det beste valget.

GAL algoritmen hadde i utgangspunkt interessante muligheter, men dette viste seg å ikke slå til hverken i det stasjonære eller ikke-stasjonære miljøet. I tillegg syntes det som om algoritmen lider av relativt høy feiljustering i forhold til de andre algoritmene, se figurene 5.7 på side 59 og 4.13 på side 49.

I oppgaven er LMS algoritmen simulert med en rekke forskjellige parametere. Simuleringene har vist at gode resultater oppnås ved bruk av filterorden $N = 10$ og konvergeringsfaktor $\mu = 0.5$. Ved en senere implementering vil det være hensiktsmessig å gjennomføre tester med forskjellige parametere, for dermed å kontrollere sonarens evne til å prosessere de bunn-reflekterte ekkoene, gitt forskjellige parameterverdier.

6.3 Konklusjon

I dette kapittelet er algoritmenes egenskaper vurdert på bakgrunn av kunnskap fra litteraturen og fra simuleringene. LMS algoritmen ble valgt for den adaptive støykansellereren. I det neste kapittelet vil det bli gitt et forslag til implementering av dette systemet.

Kapittel 7

Forslag til implementering av støykansellereren

Avsnitt i dette kapittelet

I dette kapittelet er det satt fokus på et forslag til implementering av systemet for adaptiv støykansellering med grunnlag i LMS algoritmen for transversalt filter. Det blir gitt en beskrivelse av den sentrale regneenheten i systemet, den digitale signalprosessoren (DSP). En evaluering av aktuelle prosessorer danner basis for et forslag til valg av en slik i det adaptive støykanselleringssystemet. En del aspekter ved implementeringen av hele systemet diskuteres, spesielt konverteringen fra analogt til digitalt signal og vise versa.

Det må understrekes at dette kapittelet på ingen måte er en tilstrekkelig beskrivelse av implementasjonen. En grundig gjennomgang av algoritmen versus maskinvareimplementasjonen må gjøres for å sikre at sanntidsapplikasjonen arbeider innenfor tidsintervallet bestemt av samplingsfrekvensen.

7.1 Evaluering av DSP

Selv om det i denne oppgaven kun vil konsentreres om implementering av systemet basert på digitale signalprosessorer, er ikke dette den eneste mulige løsningen. Man kan ha løsninger basert på blant annet array av prosessorer, standard mikroprosessorer eller mikroprogrammerte spesialbrikker. Siden de digitale signalprosessorenes bruksområde er klart i samsvar med denne oppgaves problem, er det derfor ingen grunn til å gå nærmere inn på andre implementasjonsløsninger.

Dette avsnittet vil bestå av en beskrivelse av hva digitale signalprosessorer er og hvorfor de passer godt i dette aktuelle tilfellet. Det vil inneholde en oversikt over aktuelle prosessorer, valgt ut og sammenlignet på bakgrunn av et sett kriterier. Et forslag til en prosessor tilpasset denne applikasjonen vil bli gitt. Prosessoren vil dermed danne basis i arbeidet med å gi et forslag til implementeringen av det adaptive støykanselleringssystemet.

I tillegg er det ønskelig at dette avsnittet skal kunne stå for seg selv som en generell innføring og oversikt over de eksisterende digitale signalprosessorer, og dermed kunne være en retningsgiver for implementasjon av andre applikasjoner med digitale signalprosessorer ved Forsvarets Forskningsinstitutt.

7.1.1 Digitale signalprosessorer - hva og hvorfor

Digitale signalprosessorer er beregnet for å utføre operasjoner i stor hastighet, spesielt repeterende beregninger med en uendelig strøm av inngangsdata [Ifeachor & Jervis 93]. Et sentralt begrep ved digitale signalprosessorer er eksekvering av sanntids algoritmer. Sanntid vil her si "så fort som mulig" innenfor et bestemt tidsavsnitt. Prosesseringen av slike algoritmer kan deles i to grupper;

- prosessering ett sample ad gangen, for eksempel ved filtrering, og
- blokk prosessering, for eksempel ved FFT[1]

Hva ved de digitale signalprosessorene gjør sanntids digital signalprosessering mulig? Følgende punkter virker optimaliserende ved arkitekturen:

- Flere buss-strukturer med separat minneområde for data og programinstruksjoner.
- I/O-porten gir mulighet for å sende data til og fra eksterne enheter, slik som analog-digital og digital-analog konvertere.
- Hurtige aritmetiske enheter for logiske og aritmetiske operasjoner.

En slik arkitektur passer godt for signalbehandlingsalgoritmer, som typisk inneholder mange repetive aritmetiske operasjoner, i tillegg til en stor strøm av data gjennom prosessoren.

Et fenomen som kjennetegner de digitale signalprosessorene er den utstrakte bruken av parallellisme, som innbefatter optimalisering av både arkitekturen og instruksjonssettet. Følgende teknikker innenfor sfæren parallellisme blir benyttet i disse prosessorene:

- **Harvard arkitektur** betyr at programinstruksjoner og data ligger i separate minneområder, slik at henting av den neste instruksjonen overlapper med eksekveringen av den nåværende instruksjonen.

[1] Fast Fourier Transform

- **Pipelining**. To eller flere operasjoner overlapper hverandre.
- **Maskinvare multiplikasjons akkumulator (MAC)**. For å redusere tiden på de tidskrevende operasjoner multiplikasjon og addisjon.
- **Spesielle instruksjoner** spesielt beregnet på signalbehandling. Fører til mer kompakt kode og øker eksekveringshastigheten av signalprosesseringsalgoritmer.
- **Replikasjon**. Man benytter seg av to eller flere basisenheter, for eksempel ALU[2] og multiplikator, som gjerne er satt til å arbeide samtidig.
- **Minne/cache på brikken**. For å redusere buss-flaskehalser på systemet ved ekstern minne/cache.

Digitale signalprosessorer kan deles i to hovedtyper; heltalls og flyttallsprosessorer. 16/24-bits heltallsprosessorer er fortsatt et godt alternativ i kosteffektive løsninger, mens 32-bits flyttallsprosessorer tar stadig en større del av markedet. Mens 16/24-bits heltallsprosessorer ser ut til å fortsatt holde stand på områder som telekommunikasjon, mobiltelefoner og disk-drivere, har 32-bits flyttallsprosessorer sitt generelle bruksområde i applikasjoner der hovedkravet er ytelse.

7.1.2 Aktuelle digitale signalprosessorer

For å kunne velge en riktig tilpasset digital signalprosessor, er det nødvendig å fremlegge et sett utvelgeseskriterier. Disse kriteriene er satt sammen i lys av det teoretiske problem denne oppgaven behandler. Etter å ha kommet fram til kriteriene, kan man sette opp resultatene fra kriterieundersøkelsen i en oversikt basert på et bredt utvalg av digitale signalprosessorer. Dette vil kunne gi en lettere pekepinn på det videre valget, og sammen med utdypende kommentarer vil dette danne grunnlaget for utvelgelsen av en digital signalprosessor som kan gjennomføre oppgaven på en god måte.

Som man husker fra avsnitt 6 har den valgte LMS algoritmen operasjoner tilsvarende $2N + 1$ multiplikasjoner og $2N + 1$ addisjoner basert på en filtrerings- og en adaptiv prosess. Algoritmen består av en løkke som repeterer disse operasjonene så lenge systemet er operativt. Det er nødvendig at prosessoren har lagringsmuligheter for N filterkoeffisienter og N inngangsverdier. Med dette i minne kan man sette opp kriteriene:

- **Intern RAM[3] og ROM[4]**. Siden ekstern RAM og ROM kan skape en I/O flaskehals, er det ønskelig at prosessoren har mulighet for internt minne. Dette vil imidlertid kun være et teoretisk problem ved det aktuelle systemet, fordi kravene til lagring er små, (N filterkoeffisienter og N inngangssignaler ved filterorden $N = 10$).
- **Grensesnitt**. Det implementerte systemet vil bestå av både analog-digital og digital-analog konvertere. Derfor må prosessoren ha et I/O grensesnitt som muliggjør slik kommunikasjon.
- **Repeterende operasjoner**. Systemet vil bestå av en uendelig løkke innenfor sonarens brukstid. Derfor bør prosessoren ha effektive metoder for å utføre repeterende instruksjoner. En slik metode er løkke uten overhead (zero overhead loop).

[2] Arithmetic Logic Unit
[3] Random Access Memory
[4] Read Only Memory

Type DSP	Intern RAM/ROM words	Sykel tid	zero overhead loop	Effekt mW	Flyt-tall
ADSP-2101	2k/2k	50	Ja	350	Nei
ADSP-2171	4k/8k	30	Ja	312	Nei
ADSP-21020	Nei	30	Ja	1360	Ja
ADSP-2106x	512kb	25	Nei	3600	Ja
AT&T DSP1617	4kb/24kb	33	Nei	108	Nei
AT&T DSP32C	8kb/1kb	N/A	Nei	180	Ja
DSP 56002	1k/1k	30	Ja	-	Nei
DSP 56L002	1k/1k	50	Ja	-	Nei
DSP 96002	2kb/1kb	50	Ja	-	Ja
TMS 320C25	1k/4k	100	Nei	550	Nei
TMS 320C30	2k/4k	60	Ja	1000	Ja
TMS 320C40	2k/4k	50	Ja	1500	Ja
TMS 320C53	4k/16k	50	Ja	500	Nei

Tabell 7.1: Kort oversikt over DSP'er og deres forhold til de oppsatte kriterier.

- **Effektforbruk**. Det tas utgangspunkt i at dette er et prosjekt for FFI, der lavt effektforbruk ofte er et krav i applikasjoner. Selv om det i dette systemet ikke er påtrengende med et slikt krav, er det naturlig å tro at denne oppgaven vil kunne danne basis for implementasjoner der dette kravet er aktuelt.

- **Flyttall**. Den fremsatte LMS algoritmen fungerer best i flyttalls regning [Lebeda 90]. Ved begrenset regnenøyaktighet bør algoritmen forbedres ved å innføre vekt-lekkasje.

Tabell 7.1 viser en slik oversikt, basert på et bredt utvalg av digitale signalprosessorer fra leverandørene Analog Devices, AT&T, Motorola og Texas Instruments. Oversikten viser de digitale signalprosessorene i forhold til de oppsatte kriteriene. Kriteriet for grensesnitt er ikke tatt med, fordi alle de nevnte prosessorene har slike funksjoner.

Denne oversikten gir ikke et fullgodt bilde av prosessorenes egenskaper mot adaptiv filtrering, men en gjennomgang av oversikten gjør at man kan velge ut enkelte prosessorer for nøyere gjennomgang, gjerne på en bredere og mer generell basis enn denne oppgaves problem. Informasjonen er i hovedsak tatt fra [EDN 94] og fra innsamlet data fra leverandører.

ADSP-2106x

Dette er en relativt ny 32-bits flyttallsprosessor fra Analog Devices, som også går under navnet SHARC. Denne prosessoren har en egenskap som skiller seg ut; i en rekke applikasjoner eksekverer den med bare internt minne. Dette minnet består av 521 kbyte SRAM, organisert i to banker. Andre merkbare egenskaper er omfattende I/O grensesnitt, bestående av 6 kommunikasjonsporter og 2 serielle porter. Kommunikasjonsportene åpner for multiprosessor implementasjon, mens de serielle portene er grensesnitt mot eksternt minne, konvertere osv. Prosessoren er moderne, med en rekke optimaliserende egenskaper, god internt-minne organisering, omfattende grensesnitt og lav klokkesykeltid.

AT&T DSP32C

Dette er en CMOS versjon av en av de første 32-bits flyttallsprosessorene. Den har en enkel arkitektur, med en høyhastighetsbuss som kan klare opptil 4 instruksjoner/operasjoner per sykel. Prosessoren har utstrakt bruk av pipelining og internt minne. Prosessoren har ikke funksjon for løkke uten overhead. På grunn av omfattende serieproduksjon for kommunikasjonsmarkedet er AT&T's prosessorer kosteffektive.

DSP 56L002

Dette er en laveffektsprosessor basert på Motorolas DSP 56002, en 24-bits heltallsprosessor. Med tanke på at dette er en heltallsprosessor, er ikke denne i utgangspunktet interessant, men med en lavere driftspenning 3.3V er dette en aktuell løsning i applikasjoner med behov for lavere effekt-konsumering, slik som håndportable batteridrevne systemer. 24-bits ordstørrelse kan være en fordel i forhold til 16-bit i applikasjoner med store datamengder. Prosessoren består dessuten av 2 serielle porter, men har svakheter ved minneorganiseringen.

DSP 96002

Dette er en 3. generasjons 32-bits flyttallsprosessor fra Motorola, som har hatt stor utbredelse i militære applikasjoner. Den har egenskaper som relativt lavt effektforbruk, internt data og programminne (2kbyte) og to preprogrammerbare ROM's. Grensesnittet er derimot noe uklart, selv om prosessoren står oppført med 1 seriell port. Generelt sett er denne prosessoren godt egnet innenfor felt som grafikk, bilde og numerisk prosessering.

TMS 320C25

Denne 2. generasjons heltallsprosessoren fra Texas Instruments er tatt med på tross av manglende flyttall og løkke uten overhead. Den ble introdusert i 1986 og er en CMOS versjon av TMS 32020. Årsaken til at denne er tatt med, er dens store popularitet og dens omfattende instruksjonssett, som også omfatter instruksjoner for adaptiv filtrering. Prosessoren har tilstrekkelig med internt RAM og ROM, men har litt uklare løsninger for grensesnitt. Den har heller ikke spesielle funksjoner for løkke uten overhead, bare mulighet for å repetere en enkel instruksjon.

TMS 320C30

Dette er en 3. generasjons 32-bits flyttallsprosessor fra Texas Instruments, som har vært populær i mange systemer. Prosessoren har en kompleks arkitektur, bestående av flere busser, minner og registre. Denne er valgt foran den 4. generasjons prosessoren TMS 320C4x, som har klare ytelsesforbedringer over TMS 320C30, først og fremst på grunn av dens optimalisering mot parallell prosessering, men også på grunn av den utstrakte bruken av TMS 320C30 i mange systemer. Fra oversikten i tabell 7.1 ser man at prosessoren tilfredstiller de oppsatte krav. Prosessoren har i tillegg 2 serielle I/O porter.

TMS 320C53

I denne 5. generasjons 16-bits heltallsprosessoren har Texas Instruments fått fram en laveffekts, lavkostnads digital signalprosessor med gode ytelser. Den har sin basis i TMS

320C2x, men har mer effektive operasjoner og høyere gjennomstrømning (throughput). Ytelsesforbedringene synes å ligge på det dobbelte av tidligere TMS 320C2x, og prosessoren har gode muligheter for økt internt minne. Dette er en rask prosessor med lavt effektforbruk grunnet effektive power down moder.

7.1.3 Valg av digital signalprosessor

Man kan stille spørsmålet - klarer disse prosessorene å gjennomføre adapsjonen og filtreringen innenfor tidsrammen? Uten å gå om en nøyaktig analyse av tiden for hver operasjon kan man foreta en beregning som svarer bekreftende på spørsmålet.

Det tas utgangspunkt i LMS algoritmen med $2N + 1$ addisjoner og multiplikasjoner og filterorden $N = 10$. Det aktuelle frekvensområdet ved sonaren, 500-5000Hz, tillater, ved samplingsteoremet, en samplingsfrekvens ned mot 10 kHz. Fra tabell 7.1 ser man at sykeltiden varierer fra 25-100 ns. Hvis hver aritmetiske operasjon tar en klokkesykel, vil man i verste tilfelle (sykeltid = 100ns) oppleve at hver gjennomgang av LNS algoritmen tar

$$(2 * 10 + 2) * 100 = 2.2\mu s. \tag{7.1}$$

Med samplingsfrekvens lik 10kHz hentes et nytt sample inn hvert 100μ s. Man ser imidlertid at man kunne fått problemer med kombinasjon av oversampling og RLS algoritmen.

Hvilken DSP vil fungere best i det aktuelle støykanselleringssystemet. Det er vist at alle prosessorene klarer å utføre hver algoritme-iterasjon godt innenfor samplingsintervallet, selv med oversampling. Fra tidligere har man at LMS algoritmen fungerer best med flyttalls aritmetikk, derfor utelukkes heltallsprosessorene i det videre arbeidet. Av de gjenstående aktuelle prosessorene ADSP-2106x, AT&T DSP32C, DSP 96002 og TMS 320C30 skiller Texas Instruments prosessor TMS 320C30 seg ut ved at den har instruksjoner som er optimaliserende mot adaptiv filtrering og mot LMS algoritmen i særdeleshet. Prosessoren er moderne, og har vært i omfattende bruk i mange systemer. Det vil derfor være en prosessor med et instruksjonssett mange kjenner. Blokkskjema av prosessoren er vist i appendiks F.

7.2 Implementasjonsløkken

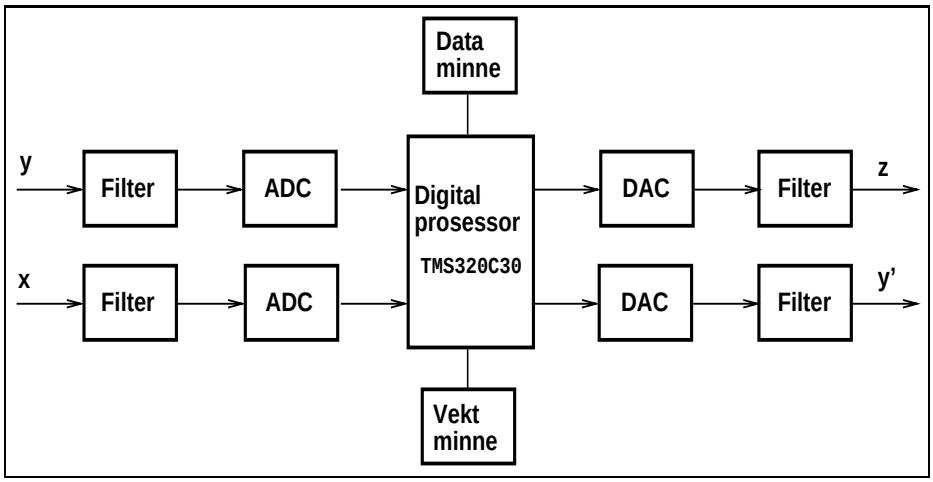
Nå vil systemet bli vurdert helhetlig. De følgende steg beskriver en løkke som må gjennomføres innenfor tidsintervallet ett sample. Det refereres til LMS algoritmen i tabell 2.1 side 14 og figuren 2.1 side 8.

1. **Hent et inngangssample fra inngangene.** I dette steget leses inngangssamplet fra referanseføleren fra porten mot den analoge-digitale konverteren. For at dette skal skje har prosessoren mottatt et avbrudds-signal fra konverteren. Samtidig vil det bli lest inn et sample fra primærsignalet.

2. **Lagre inngangssamplene.** Inngangssamplet må lagres for videre bruk. Det mest naturlige er å bruke en RAM-blokk til dette formålet, der et adresseregister holder greie på samplenes posisjoner.

3. **Beregn filterutgang.** Konvolusjonssummen av N filterkoeffisienter og N inngangssampler blir utført ved bruk av en serie multiplikasjons/akkumulasjonsoperasjoner.

4. **Beregn feil.** Feilen er også det ønskede signal, dette blir funnet ved å subtrahere filterutgangen fra primærsignalet.



Figur 7.1: Implementering av det adaptive støykanselleringssystemet med TMS320C30 digital signalprosessor.

5. **Send resultatet til utgang.** Feilen i forrige steg er lik utgangen av det adaptive filteret. Dette ønskede signal blir sendt til utgangsporten og videre til den digital-analoge konverteren.

6. **Beregn oppdatering.** Filterkoeffisientene må nå oppdateres, med produktet av feilen, inngangssignalet og en konvergeringsfaktor. Denne operasjonen må gjennomføres for hver filterkoeffisient, og er dermed den dominerende del av løkkeprosessen.

7. **Ventetilstand**[5]**.** Når løkkeprosessen er ferdig, kan prosessen gå i en ledig tilstand[6], der den venter på neste inngangssample. I denne tiden kan bakgrunnsprosesser med lav prioritet kjøres, for eksempel et estimat av midlere kvadratfeil.

7.3 Implementasjon av systemet

I kapittel 7.1 ble det gjennomført en analyse av aktuelle digitale signalprosessorer. Flere av disse kunne utvilsomt implementeres for denne applikasjonen, men allikevel var det naturlig å konkludere med at prosessoren TMS320C30 fra Texas Instruments var riktig for dette adaptive systemet.

Det adaptive støykanselleringssystemet består av flere deler enn den digitale signalprosessoren. I figur 7.1 er det vist hvordan en slik implementering kan se ut. Det analoge inngangsfilteret skal redusere båndet til inngangssignalet for å redusere aliasing, den analog-digitale konverteren (ADC) konverterer signalet fra analog til digital form. Efter prosessorens arbeid konverterer den digital-analoge konverteren (DAC) signalet til analog form, og utgangsfilteret glatter signalet og fjerner uønsket høyfrekvent støy. Det vil nå gåes gjennom disse overgangene litt mer i detalj, slik at det blir mer forståelig hvilke problemer man kan stå ovenfor.

[5] engelsk: wait state
[6] engelsk: idle state

Sampling i ADC

Samplingsteoremet er grunnlaget for at samplingen skal gi et signal som blir fullt ut beskrevet. Teoremet sier at hvis den høyeste frekvenskomponenten i et signal er f_{MAX}, må signalet samples med en rate minst lik $2f_{MAX}$. Sampling med lavere samplingsfrekvens gir aliasing i det ønskede frekvensbåndet. Prosessen kan forklares nærmere på følgende måte.

Ved sampling får man det samme spekteret som ved det analoge signalet, men i tillegg får man de samme spektralkomponentene repetert ved multipler av samplingsfrekvensen F_s. Disse høyere ordens komponentene kalles speilfrekvenser, og hvis samplingsfrekvensen F_s ikke er tilstrekkelig høy, vil det, som nevnt tidligere, oppstå aliaser i det ønskede frekvensbånd.

For å redusere effekten av aliasing kan man enten bruke anti-aliasing filtre, som båndbegrenser inngangssignalet, eller bruke høyere samplingsfrekvenser. Et anti-aliasing filter vil ideelt redusere komponenter over fold-over frekvensen (høyeste frekvenskomponent), i praksis vil filteret ha cut off og stopband frekvenser, i tillegg til amplitude forvrengning i pass-båndet [Ifeachor & Jervis 93].

Ved implementering av dette systemet må det tas hensyn til konverterens bit-oppløsning for å beregne den nødvendige dempningen over Nyquist frekvensen slik at signalet ikke blir detektert av konverteren. En formel som beskriver dempningen A_{MIN} er [Ifeachor & Jervis 93]

$$A_{MIN} = 20log(\sqrt{1.5} * 2^{B+1}) \tag{7.2}$$

der B er antall bit i AD konverteren. Generelt sett er det en trend i å oversample signalet, det vil si bruke høy samplingsfrekvens. Dette medfører igjen bruk av raskere AD konvertere. Dermed kan man bruke enklere anti-aliasing filtre. Dette vil også medføre høyere signal/støy forhold.

Signalgjenvinning i DAC

Etter operasjonen i den digitale signalprosessoren går signalet igjennom først en digital-analog konverter, deretter et lavpass filter for å glatte utgangssignalet. Konverteren mottar digitale data parallelt og produserer et analogt utgangssignal. Utgangen fra konverteren har uønskede høyfrekvenskomponenter sentrert i multipler av samplingsfrekvensen. Disse komponentene kan forringe det analoge signalet. Filteret glatter utgangssignalet og fjerner dermed uønskede høyfrekvenskomponenter.

Skal man bruke klassiske AD og DA konvertere med høy oppløsning, er de generelt trege. Dette medfører at man i et sanntids problem med rask digital signalprosessor får en flaskehals mot konverteren. Nyere teknologi har gitt såkalte sigma-delta konvertere, som kjennetegnes ved oversampling, raskhet og anti-aliasing filtrering. Disse gir ikke flaskehalser i systemet.

7.4 Programmering av den digitale signalprosessoren

Tidligere var det vanlig å programmere prosessorene i maskinspråk. I den senere tid er det gjort omfattende arbeid med å lage høynivå kompilatorer for prosessorene. Dette sees på som en stor fordel fordi det forenkler programmeringen. Til prosessoren TMS320C30 er det tilgjengelig både C og C++ kompilatorer. I appendiks E.1 er det vist et LMS C-program. Det er allikevel flere som foretrekker maskinspråk fordi dette ofte gir mer effektiv kode,

derfor er det også listet i appendiks E.2 et maskinspråk LMS-program. Disse programmene er uten I/O definisjoner, og vil derfor kun representere et utgangspunkt og hjelp i programmeringen av LMS algoritmen for det adaptive støykanselleringssystemet.

7.5 Konklusjon

Som nevnt i forrige kapittel, er ikke LMS algoritmen spesielt kompleks. Derfor var det enkelt å vise at prosessoren fint greide å utføre prosesseringen innenfor det avsatte samplingsintervallet. En mulig flaskehals kan man, som nevnt tidligere i dette kapittelet, se i overgangene mellom konverterene. Dette kan imidlertid løses ved å bruke moderne AD og DA konvertere.

For implementering av systemet må gangen i det adaptive støykanselleringssystemet analyseres mer nøye. Man må i tillegg ha mer omfattende kjennskap til de forskjellige delene i maskinvaren. Dette er nødvendig for en optimal støykanselleringsoperasjon, selv om man kan konkludere med at det ikke synes å oppstå noen direkte problemer med implementeringen av det adaptive støykanselleringssystemet.

[Denne siden er blank med hensikt]

Kapittel 8

Oppsummering, konklusjon og videre arbeid

Avsnitt i dette kapittelet

Denne oppgaven har hatt følgende problemstilling: Forsvarets Forskningsinstitutt's forskningsskip H. U. Sverdrup har en parametrisk sonar for seismikk og topografiske undersøkelser. Sonarens mottakende frekvensområde 500-5000Hz er overlagret med egenstøy fra gearboksen i området 500-1000Hz. Siden dette frekvensområdet er spesielt interessant, er det ønskelig å redusere denne påvirkningen. I arbeidet med å løse problemet ble følgende gjennomgått:

3 adaptive støykanselleringsalgoritmer ble valgt ut på bakgrunn av et sett kriterier; LMS, RLS og GAL. Utledningen av disse algoritmene ble vist, fordi dette ga bakgrunn for å forstå forskjellene i egenskaper. Algoritmene, gitt i sin helhet i tabellene 2.1, 2.2 og 2.3, representerte basisen for simuleringene i senere kapitler.

For å finne den beste referanseplasseringen for støykansellereren, ble det gjort en undersøkelse ombord på H.U. Sverdrup. Denne undersøkelsen konkluderte med en plassering av referanseføleren som ga størst samsvar mellom støy i primærsignal og referansesignal. Opptakene dannet også basis for senere simuleringer.

For å få et godt sammenligningsgrunnlag mellom algoritmene, ble det gjort simuleringer med syntetiske, stasjonære signaler. Teori ble gjennomgått for å gi en forståelse av algoritmenes konvergerende egenskaper. Resultatet ble et grunnlag for senere utvelgelse av en algoritme.

Opptakene fra rapporten om referanseplasseringer ble brukt til å simulere en virkelig adaptiv støykansellerer. Mindre modifikasjoner på de eksisterende algoritmer måtte gjøres for å oppnå tilfredsstillende resultat. Algoritmenes egenskaper i et ikke-stasjonært miljø ble vurdert både på bakgrunn av simuleringene, og på bakgrunn av foreliggende litteratur.

På bakgrunn av oppgavens simuleringer og andre egenskaper ved algoritmene, ble en algoritme valgt ut. Denne algoritmen ble grunnlaget for forslaget til implementering av det adaptive støykanselleringssystemet.

En kort gjennomgang av aspekter ved implementeringen av støykanselleringssystemet ble vist. Egenskaper ved de digitale signalprosessorene ble gjennomgått. Aktuelle prosessorer for denne oppgaves problem ble diskutert, og man kom frem til en konklusjon for valg av prosessor til implementeringen. Aspekter ved maskinvareimplementeringen som har innvirkning på algoritmenes effektivitet ble diskutert.

8.1 Konklusjon

Fra litteraturen kan man få inntrykk av at de moderne, raske algoritmene og gitter filterstrukturen er uovertrufne i alle situasjoner. Dette er ikke nødvendigvis tilfelle. I denne oppgaven er det vist at de forventede gode konvergensegenskapene ved RLS algoritmen og GAL algoritmen ikke oppfylles, gitt signaler med ikke-stasjonær karakter og støysignal bestående av spektrale komponenter. Konvergensegenskapene mellom algoritmene synes å jevne seg ut gitt ikke-stasjonære signaler.

I kapittel 4 ble det vist at RLS hadde meget gode konvergensegenskaper vis a vis LMS algoritmen ved stasjonære signaler. Man kunne forvente at også GAL algoritmen skulle hatt rask konvergens fordi gitter-filteret ortogonaliserer baklengs prediksjonsfeil. Sinuskomponentene i støysignalet er korrelerte (ikke ortogonale), og dette skulle ha gitt gitter-filteret en fordel. Dette ble ikke bekreftet i oppgavens simuleringer. En årsak til dette kan være at LMS algoritmen har optimal ytelse gitt testsignalene, og derfor er forskjellen mellom LMS og GAL algoritmenes konvergensrate utjevnet.

I kapittel 5 ble det vist at RLS algoritmen mister mye av sine gode konvergensegenskaper ved ikke-stasjonære signaler. Årsaken til dette ligger ved glemmefaktoren λ, som gir RLS algoritmen vektet hukommelse. og den ikke-optimale korrelasjonen mellom referansesignalet og støyen i primærsignalet. Det er i plott av effektetthetsspekteret vist at alle algoritmene har filtrert støykomponentene. Simuleringene med aktiv sonar har vist at den adaptive filtreringen tydeliggjør de returnerte bunnekkoene i sonarsignalet. Dette er et meget positivt resultat med tanke på et forbedret signal/støy forhold i sonarsignalet. Alle algoritmene ga et slikt resultat, men LMS algoritmen tydeliggjorde de ønskede signalene best.

Siden konvergensegenskapene til algoritmene er tilnærmet like ved ikke-stasjonære signaler, spiller andre faktorer en vesentlig større rolle i valget av algoritme for det adaptive støykanselleringssystemet. Egenskaper som kompleksitet og numerisk stabilitet blir derfor vektlagt i større grad. På bakgrunn av en slik helhetsvurdering ble LMS algoritmen valgt for den adaptive støykanselleringsapplikasjonen, fordi den er numerisk stabil og har lav kompleksitet.

Et forslag til implementeringen av et slikt system er vist i kapittel 7. På grunn av LMS algoritmens enkle kompleksitet er det blitt større rom for valg av maskinvarekomponenter i systemet. Det vises i avsnitt 7.1.3 at systemet fungerer godt med de gitte forutsetningene; LMS algoritmen, TMS320C30 signalprosessor og sampling ved Nyquist frekvensen.

Denne oppgaven har analysert effekten av et adaptivt støykanselleringssystem på den parametriske sonaren. Det positive resultatet gir bakgrunn for å tro at et slikt system vil bli implementert. Avsnittet som gir et forslag til implementering vil da danne basis for konstruksjonen av det adaptive støykanselleringssystemet.

8.2 Videre arbeid

Det er tidligere nevnt at en fullverdig verifisering av støykanselleringens positive effekt på sonarsignalet, kun kan gjøres ved å implementere støykanselleringssystemet i kombinasjon med sonaren. Dette vil medføre at man kan kontrollere det filtrerte signalet opp mot det ikke filtrerte, ved å se på for eksempel seismisk bunngjennomtrengning. En slik implementering har jeg gitt retningslinjer for i denne oppgaven. Enkelte aspekter må vurderes nøyere, men alt i alt er det i denne oppgaven gjort et arbeid som forhåpentligvis vil forenkle implementeringsarbeidet av den adaptive støykanselleringen.

Acta est fabula, plaudite.

[Denne siden er blank med hensikt]

Referanser

[Bellanger 87] M. G. Bellanger.
Adaptive Digital Filters and Signal Analysis.
Marcel Dekker, 1987.

[Cioffi & Kailath 84] J. M. Cioffi, T Kailath.
Fast, Recursive-Least-Squares Transversal Filters for Adaptive Filtering.
IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-32, No. 2, April 1984.

[Friedlander 82] B. Friedlander.
Lattice Filters for Adaptive Processing.
Proceedings of the IEEE. Vol. 70, No.8, August 1982, pp. 829-867.

[Griffiths 78] L. J. Griffiths.
An Adaptive Lattice Structur for Noise-Cancelling Applications.
IEEE International Conference on Acoustics, Speech and Signal Processing, New York, 1978, pp. 87-90.

[Haykin 91] S. Haykin.
Adaptive Filter Theory.
2nd ed., Englewood Cliffs, NJ.: Prentice Hall, 1991.

[Haykin 84] S. Haykin.
Introduction to Adaptive Filters.
Macmillan Publishing Company, 1984.

[Honig & Messerschmitt 84] M. L. Honig, D. G. Messerschmitt.
Adaptive Filters. Structures, Algorithms, and Applications.
Kluwer Academic Publishers, 1984.

[Hovem 80] J. M. Hovem.
Optimale ytelser ved adaptiv kansellering av støy på sonarer og ekkolodd.
Rapport ELAB, 1980.

[Ifeachor & Jervis 93] E. C. Ifeachor, B. W. Jervis.
Digital Signal Processing. A Practical Approach.
Addison-Wesley, 1993.

[Ingebretsen 86] T. W. Ingebretsen.
Adaptiv FIR-filter benyttet i støykansellering.
FFI/NOTAT-86/2004.

[Johnson & Dudgeon 93] D. H. Johnson, D. E. Dudgeon.
Array Signal Processing - Concepts an Techniques. Prentice Hall 1993.

[Kay 88] S. M. Kay.
Modern Spectral Estimation.
Prentice Hall, 1988.

[Kjølås 82] K. O. Kjølås.
Adaptiv støykansellering.
Rapport ELAB, 1982.

[Knudsen 78] T. Knudsen.
Adaptiv støyreduksjon.
ELAB, arbeidsnotat prosjekt No. 449500.21, juni 1978.

[Larssen 94] J. - E. Larssen.
Adaptiv støykansellering på ekkolodd.
Diplomoppgave akustikk NTH, høst 1994.

[Lebeda 90] M. Lebeda.
Adaptiv Signalbehandling - En Introduktion.
TFL RR 1990-1, mai 1990.

[Treichler 87] J. R. Treichler, C. R. Johnson jr., M. G. Larimore.
Theory and Design of Adaptive Filters.
Wiley-Interscience, 1987.

[Tveit 90] Elling Tveit.
Måling av støy ombord i H. U. Sverdrup.
Rapport FFI-U, 7. november 1990.

[Urick 83] R. J. Urick.
Principles of Underwater Sound.
McGraw-Hill, 1983.

[Wagner 90] M. L. Wagner.
Evaluation of Real Time Adaptive Noise Cancelling Algorithms Using a Digital Signal Processor Chip.
IEEE Proceedings, Vol 137, No. 2, Mars 1990, pp. 144-150.

[Widrow et. al. 75] B. Widrow et. al.
Adaptive Noise Cancelling: Principles and Applications.
Proceedings of the IEEE, Vol. 63, No. 12, december 1975, pp. 1692-1716.

[Widrow et. al. 76] B. Widrow et. al.
Stationary and Nonstationary Learning Characteristics of the LMS Adaptive Filter.
Proceedings of the IEEE, Vol. 64, No. 8, august 1976, pp. 1151-1162.

[EDN 94] EDN: The Design Magazine of the Electronics Industry
Special Issue: 1994 DSP Chip Direcory Juni 9, 1994.

[Denne siden er blank med hensikt]

Tillegg A

Matlab-programmer for plott av effekt-tetthets-spekter og koherens

Det er for plott av effekttetthetsspekteret brukt en funksjon i matlab **psd.m** som er basert på en klassisk metode utviklet av Welch. De reelle signalene er ikke-stasjonære, men det er antatt stasjonæritet for signalene av interesse innenfor datalengden. For plott av koherensen mellom signalene er det brukt en funksjon i matlab **cohere.m**.

```
function ets_koh(filnavn)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Plott av ETS og koheren                  %
%                                                %
%       Input:  filnavn                          %
%       Plott blir generert i funksjonen         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



xmin=0;                                          % Her settes grenser for
xmax=3000;                                       % plottene.
ymin=40;
ymax=100;

a=aifcread(filnavn);                             % Innlesning fra fil,
                                                 % legges i vektor
aleft=a(1:120*1024,1);                           % Et utsnitt tas ut,
aright=a(1:120*1024,2);                          % Kanalene deles i to

%%%%%%%%%% PSD-Welch metoden %%%%%%%%%%%%%%%%%%
Fs=48000;                                          % Samplingsfrekvensen
nfft=8192;                                         % Anntall fft
window=hanning(nfft);                              % vindusfunksjon
noverlap=nfft/2;                                   % samplingsoverlap 50%
[Pxl,f]=psd(aleft,nfft,Fs,window,noverlap,[0]);   % psd kanal left
[Pxr,f]=psd(aright,nfft,Fs,window,noverlap,[0]);  % psd kanal right
```

```
subplot(2,1,1)
plot(f,10*log10(Pxl));
axis([xmin xmax ymin ymax])
title('Effekt-tetthetsspekter kanal left')
xlabel('Frekvens, Hz')
ylabel('Effekt-spekter, dB')
grid
subplot(2,1,2)
plot(f,10*log10(Pxr));
axis([xmin xmax ymin ymax])
title('Effekt-tetthetsspekter kanal right')
xlabel('Frekvens, Hz')
ylabel('Effekt-spekter, dB')
grid

%%%%%%%%%% Korrelasjonen %%%%%%%%%%%%%%%%%%
[Cxy,f]=cohere(aleft,aright,8192,Fs,4096);
plot(f,Cxy)
title('koherense estimat')
xlabel('Frekvens, Hz')
axis([xmin 2000 0 1])
grid
```

Tillegg B

Matlab-programmer for simulering med syntetiske signaler

B.1 LMS algoritmen

```
utsig = function lms(N,konv)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       LMS algoritmen                          %
%       Input:  Filterorden                     %
%               Konvergeringsfaktor             %
%       Output: Filtrert signal                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%% VARIABLER %%%%%%%%%%%%%%%%%%
t=(0:.001:1)';                % Sampler/iterasjoner
NR=length(t)-1;               % Iterasjonslengden
buf=zeros(1,NR);              % Buffer for utgangsfeilen
a_k=zeros(N,1);               % Filtervekt-vektor
x_vekt=zeros(N,1);            % Inngangsvektor


%%%%%%%%% GENERERING AV SIGNALER %%%%%%%%%%
s=0.01*randn(size(t));
n=sin(2*pi*300*t);
sn=n+s;

%%%%%%%%%%%%%%% HOVEDLOEKKE %%%%%%%%%%%%%%%%%%
for k=1:NR
for j=N:-1:2                  % Skift operasjon i inngangsvektoren
x_vekt(j)=x_vekt(j-1);
end
x_vekt(1) = n(k);             % Nytt sampel inn i vektoren
y_sig=sn(k);                  % Nytt sampel fra primaersignalet
```

```
y_merket=a_k'*x_vekt;        % Filteroperasjonen
feil=y_sig-y_merket;         % Feilen finnes
a_k=a_k+konv*feil.*x_vekt;   % Oppdatering av ny filterkoeffisient
buf(1,k)=feil;
end
utsig=buf;
```

B.2 RLS algoritmen

```
utsig = function rls(N,la)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       RLS algoritmen                          %
%       Input:  Filterorden                     %
%               Glemmefaktor                    %
%       Output: Filtrert signal                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%% VARIABLER I SIMULERINGEN %%%%%%%%%%
t=(0:.001:1)';                  % Sampler/iterasjoner
NR=length(t)-1;                 % Iterasjonslengden
sigma=333;                      % Variable for def. av P

%%%%%%%%%%%%% DEFINERING AV BUFFERE %%%%%%%%%%
buf=zeros(1,NR+1);              % Buffer for utgangsfeilen
w=zeros(N,1);
x=zeros(N,1);

%%%%%%%%%%%%%% GENERERING AV SIGNALER %%%%%%%%%%
s=0.01*randn(size(t));
n=sin(2*pi*300*t);
sn=n+s;

%%%%%%%%%%  INITIERING AV AUTOKORRELASJONSMATRISEN P %%%%%%%%%%%%%
P=sigma*eye(N);      % En stor variabel * identitetsmatrisen

%%%%%%%%%%%%%%%%%%%% HOVEDLOEKKEN %%%%%%%%%%%%%%%%%%%%

for k=1:NR
for j=N:-1:2                  % Skift operasjon i inngangsvektoren
x(j)=x(j-1);
end
x(1) = n(k);                  % Nytt sampel inn i vektoren
y=sn(k);                      % Nytt sampel fra primaersignalet
pii=x'*P;
k1=la+pii*x;
k_vekt=(P*x)/k1;
```

```
sne=y-w'*x;                    % Feilen finnes
w=w+k_vekt*sne;                % Oppdatering av filterkoeffisienter
P = P./la - k_vekt*pii;        % Beregner neste matrise
buf(1,k)=sne;
end
utsig=buf;
```

B.3 GAL algoritmen

```
utsig = function gal(N,konv)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       GAL algoritmen                               %
%       Input:  Filterorden                          %
%               Konvergeringsfaktor                  %
%       Output: Filtrert signal                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%% VARIABLER
t=(0:.001:1)';
NR=length(t)-1;
beta=1;                            % Veiing av xi
c=0.001;                           % Initiell xi verdi

%%%%%%%%%%% INITIALISERING
buf=zeros(1,NR+1);
a_k=zeros(N,1);                    % Filterkoeffisientene
f=zeros(1,N);                      % Forlengs feil
b=zeros(1,N);                      % Baklengs feil
xi=zeros(1,N-1);                   % Normalisering av gamma
for k2=1:N-1 xi(1,k2)=c; end
gamma=zeros(1,N-1);                % Refleksjonskoeffisienter

%%%%%%%% SIGNALER INN
s=0.01*randn(size(t));
n=sin(2*pi*300*t);
sn=n+s;

%%%%%%%%%% HOVEDLOEKKE
for k = 1:NR
b_forrige = b;                     % Skift b
b(1,1) = n(k);     %%%%%%%% NYE SIGNALER
f(1,1) = n(k);
y_sig=sn(k);
%%%%%%%%% OPPDATERING AV FILTERETS REFLEKSJONSKOEFFISIENTER
for k1 = 1:N-1
    f(1,k1+1) = f(1,k1) - (gamma(1,k1) * b_forrige(1,k1));
```

```
    b(1,k1+1) = b_forrige(1,k1) - (gamma(1,k1) * f(1,k1));
    xi(1,k1) = (beta*xi(1,k1))+(2-beta)*(f(1,k1)^2 + b_forrige(1,k1)^2);
    gamma(1,k1) = gamma(1,k1) + (1/(xi(1,k1))) * ((f(1,k1)*b(1,k1+1))
    + (b_forrige(1,k1)*f(1,k1+1)));
end
%%%%%%%%% ADAPTIV FILTRERING
y_merket=sum(a_k'.*b);                          % Filterutgang
feil=y_sig-y_merket;                            % Utgangsfeil
a_k=a_k+konv*(feil.*b');                        % Filterkoeffisienter
buf(1,k)=feil;
end
utsig=buf;
```

Tillegg C

Matlab-programmer for simulering med reelle data

C.1 LMS algoritmen

```
utsig = function lms_reell(N,konv,filnavn,Innleslengde)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       LMS algoritmen for reelle data      %
%       Input:  Filterorden                 %
%               Konvergeringsfaktor         %
%               Filnavn                     %
%               Innelest datalengde         %
%       Output: Filtrert signal             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%% Simulering av lms-algoritmen %%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%% Innlesning
a=aifcread(filnavn);                          % lokal rutine aifcread.m
aleft=a(1:Innleslengde,1);                    % Et utsnitt tas ut,
aright=a(1:Innleslengde,2);                   % deles i to kanaler

%%%%%%%%%%%% Variabler
NR=length(aleft);
buf=zeros(1,Innleslengde);                    % Buffer for utgangsfeil
a_k=zeros(N,1);                               % Filterkoeffisienter
x_vekt=zeros(N,1);                            % Inngangsvektor

%%%%%%%%%%%%%% HOVEDLOEKKE
for k=1:NR
for j=N:-1:2                                  % Skift register for
x_vekt(j)=x_vekt(j-1);                        % inngangsvektor
end
```

```
x_vekt(1) = aleft(k);                          % Les inn ny innsignal
y=aright(k);                                   % Nytt prim\ae rsignal
y_merket=a_k'*x_vekt;
feil=y-y_merket;                               % Beregn utgangsfeil
xi=((1-beta)*xi)+(N*beta*(x_vekt'*x_vekt));    % Beregn normalisering
midler=(2*konv)./(0.1+xi);
a_k=a_k+((midler.*feil).*x_vekt);              % Oppdater filterkoeffisient
buf(1,k)=feil;
end
utsig=buf
```

C.2 RLS algoritmen

```
%%%%%%%%%%%%%%%% rls-algoritmen %%%%%%%%%%%%%%%%%%%%5

utsig = function rls_reell(N,la,filnavn,Innleslengde)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       RLS algoritmen for reelle data      %
%       Input:  Filterorden                 %
%               Glemmefaktor                %
%               Filnavn                     %
%               Innelest datalengde         %
%       Output: Filtrert signal             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%% Innlesning av signaler
a=aifcread(filnavn);                           % Se lokal aifcread.m
aleft=a(1:Innleslengde,1);                     % Et utsnitt tas ut,
aright=a(1:Innleslengde,2);                    % deles i to kanaler
NR=size(aleft,1);

%%%%%%%%%%%%%% Variabler i simuleringen
sigma=333333;                                  % Variable for def. av P
buf=zeros(1,NR+1);                             % Buffer for utgangsfeil
w=zeros(N,1);                                  % filterkoeffisienter
x=zeros(N,1);                                  % inngangsvektor

% Initiering av autokorrelasjonsmatrisen P
P=sigma*eye(N);      % En stor variabel * identitetsmatrisen

%%%%%%%%%%%%%%%%%%% HOVEDLOEKKE
for k=N+1:NR
for j=N:-1:2                                   % skiftregister for inngangsvektoren
x(j)=x(j-1);
end
x(1) = aleft(k);                               % Nye signaler inn
```

```
y=aright(k);
pii=x'*P;
k1=la+pii*x;
k_vekt=(P*x)/k1;
sne=y-w'*x;                                % Beregn utgangsfeil
w=w+k_vekt*sne;                            % Oppdater filterkoeffisienter
P = P./la - k_vekt*pii;                    % Oppdater autokorrelasjonsmatrisen
buf(1,k)=sne;                              % Buffer for utgangsfeil
end
utsig=buf;
```

C.3 GAL algoritmen

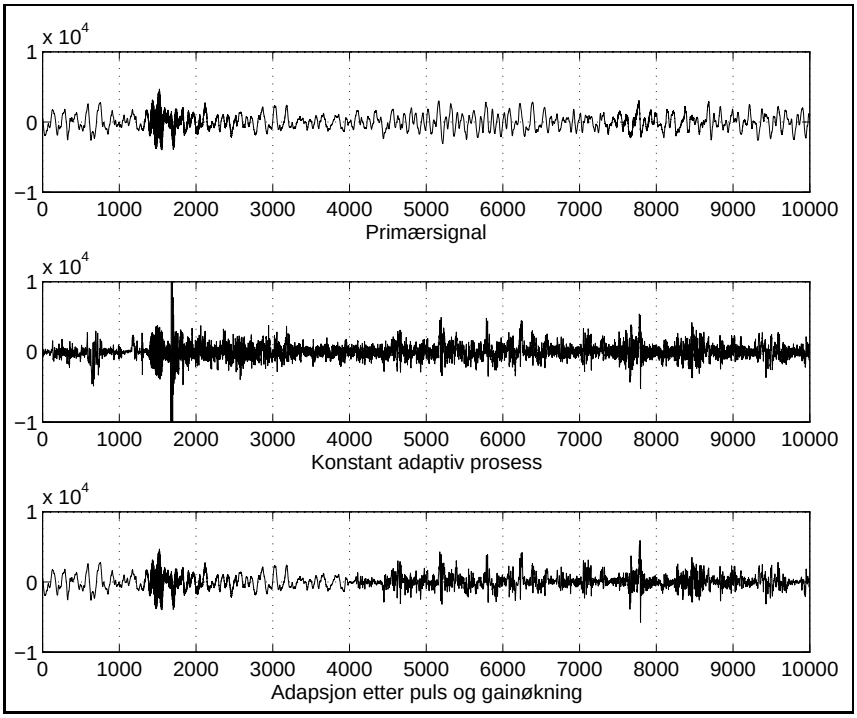
```
utsig = function gal_reell(N,konv,filnavn,Innleslengde)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       RLS algoritmen for reelle data      %
%       Input:  Filterorden                 %
%               Konvergeringsfaktor         %
%               Filnavn                     %
%               Innelest datalengde         %
%       Output: Filtrert signal             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%% INNLESNING
a=aifcread(filnavn);
aleft=a(1:Innleslengde,1);                 % Et utsnitt tas ut,
aright=a(1:Innleslengde,2);                % deles i to kanaler


%%%%%%%%%%%%% VARIABLER
NR=length(aleft);
ga=0;                                      % Filterkoeff normalisering
beta=0.1;                                  % Veiing av xi og ga.
c=0.0001;                                  % Initialverdi for xi

%%%%%%%%%%% INITIALISERING
buf=zeros(1,NR);                           % Buffer for utgagnsfeil
a_k=zeros(N,1);                            % Filterkoeffisient
f=zeros(1,N);                              % Forlengs feil
b=zeros(1,N);                              % baklengs feil
xi=zeros(1,N-1);                           % Normalisert refleksjonskoeff
for k2=1:N-1 xi(1,k2)=c; end
gamma=zeros(1,N-1);                        % refleksjonskoeff

%%%%%%%%%% HOVEDLOEKKE
```

```
for k = 1:NR
b_forrige = b;                                  % Skifting
b(1,1) = aleft(k);                              % nye signaler
f(1,1) = aleft(k);
y_sig=aright(k);
%%%%%%%%% OPPDATERING AV REFLEKSJONSKOEFFISIENTER FILTER
for k1 = 1:N-1
    f(1,k1+1) = f(1,k1) - (gamma(1,k1) * b_forrige(1,k1));
    b(1,k1+1) = b_forrige(1,k1) - (gamma(1,k1) * f(1,k1));
    xi(1,k1) = (beta*xi(1,k1))+((1-beta)*(f(1,k1)^2 + b_forrige(1,k1)^2));
    gamma(1,k1) = gamma(1,k1) + (1/(xi(1,k1))) * ((f(1,k1) * b(1,k1+1))
    + (f(1,k1+1) * b_forrige(1,k1)));
end
%%%%%%%%% OPPDATERING AV ADAPTIVT FILTER
y_merket=sum(a_k'.*b);
feil=y_sig-y_merket;
ga = (beta*ga) + ((1-beta)*(b*b'));
a_k=a_k+konv*((feil.*b')./(ga));
buf(1,k)=feil;
end
utsig=buf;
```

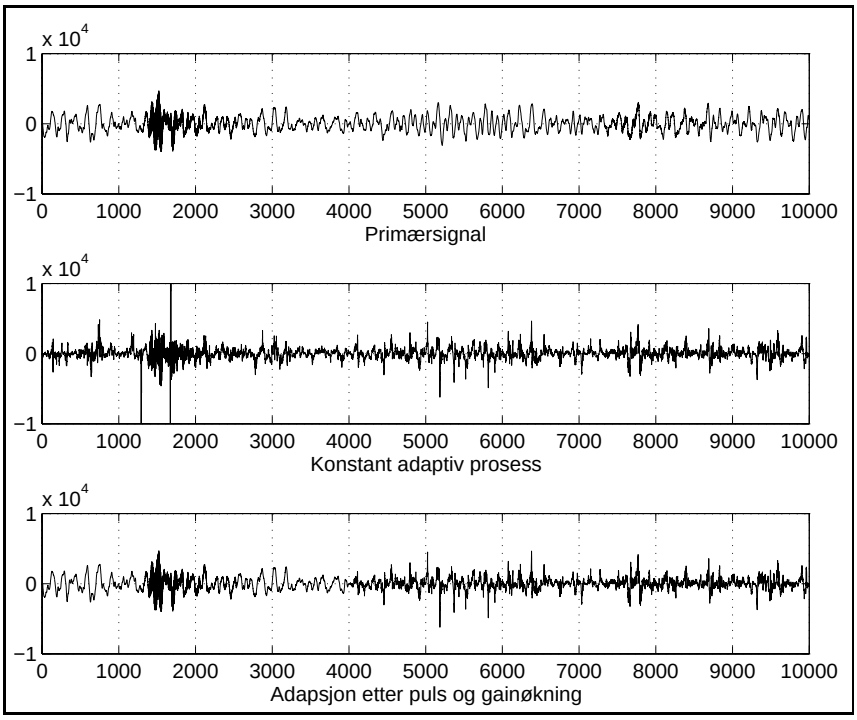
Tillegg D

Plott av sonarsignaler

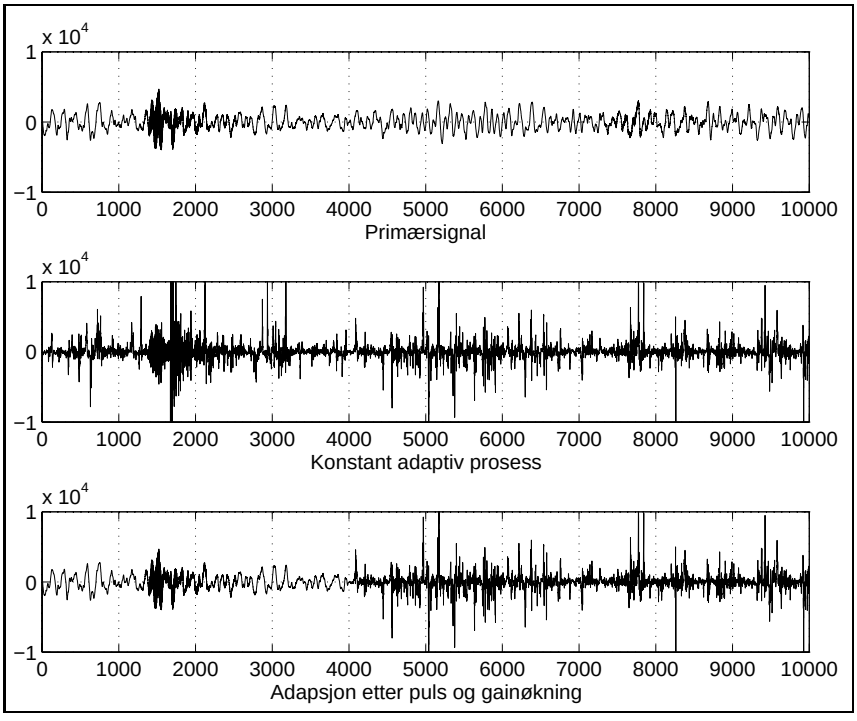
Dette er plott av sonarsignaler med sonar i aktiv modus. Det som vises er før filtrering, etter filtrering med konstant adaptiv filtrering og etter filtrering med adaptiv filtrering ikke ved pulsutsendelse.



Figur D.1: LMS algoritmen, filterorden 10 og konvergeringsfaktor 0.5



Figur D.2: RLS algoritmen, filterorden 10 og glemmefaktor 0.9



Figur D.3: GAL algoritmen, filterorden 10 og konvergeringsfaktor 0.5

Tillegg E

Program for implementeringen av støykansellereren

Her vil det bli gitt to programmer som vil kunne forenkle implementeringen av den adaptive støykanselleringen. Først blir det gitt et C-program av LMS algoritmen. Deretter et assembler program for den digitale signalprosessoren TMS 32030. Programmene er ikke fullverdige. Allikevel vil de kunne være til hjelp i implementeringen av det adaptive støykanselleringssystemet.

E.1 LMS C-program

```
/*.........................................*/
/*    LMS algoritmen                       */
/*                                         */
/*    Input:    x[]  referansevektor       */
/*              dk   primærsignal          */
/*              w[]  filterkoeffisientvektor */
/*                                         */
/*    Output:   ek   feilverdi             */
/*              yk   filterutgang          */
/*                                         */
/*.........................................*/

double  lms()
{
     int      i;
     double   uek,yk;

     yk=0;
     for(i=0;i<N,++i){       /* digital filtrering */
         yk=yk+w[i]*x[i];
     }
     ek=dk-yk;               /* Beregn utgangsfeil */

     uek=2*mu*ek;            /* Oppdater filterkoeffisientene */
```

```
     for(i=0;i<N;i++){
         w[i]=w[i]+uek*x[i];
     }
     return(ek);
}
```

E.2 LMS maskinprogram for TMS 320C30

Dette maskinprogrammet er generert ved hjelp av et program fra Texas Instruments. Det er ikke tatt hensyn til I/O konfigurering.

```
*****************************************************************
*
*    TMS320C30 adaptive filter Using Transversal Structure
*    and Normalized LMS Algorithm ,Looped Code
*
*    d(n) --------------------------|
*                                   |
*                                   |
*                                   |+
*                                (SUM)------> e(n)
*                                   |-
*                  ------           |
*    x(n) ---------| AF |-----------|--------> y(n)
*                  ------
*
*    Algorithm:
*
*               9
*       y(n) = SUM w(k)*x(n-k)  k=0,1,2,...,9
*              k=0
*
*       e(n) = d(n) - y(n)
*
*       var(k) = (1.-r) * var(K-1) + r * x(n) * x(n)
*
*       w(k) = w(k) + u*e(n)*x(n-k)/var(k)  k=0,1,2,..9
*
*       where filter order = 10, r = 0.9 and mu = 0.5.
*
*       Note:  This source program is the generic version, I/O
*              configuration has not been set up. User has to modify
*              the main routine for specific application.
*
*       Initial condition:
*              1) BK = N, where N is filter order
*              2) AR0 and AR1 should pointer to x[0] and w[0]
*              3) R6 and R7 should contain input x(n) and d(n) signals
```

```
*              4) memory u should contain valve of 0.5
*              5) memory r should contain valve of 0.9
*              6) memory r_1 should contain valve of 0.1 (=1.0-0.9)
*              7) memory var should be initialized to 1.000000
*
*              Chen, Chein-Chung  April, 1989
*****************************************************************
order       .set    10                  ; filter order
;
;   Perform adaptive filter
;
            .text
begin       .set    $
            STF     R6,*AR0             ; insert x(n) to buffer;

;   estimate the power of the input signal
;
            MPYF    R6,R6               ; R6 = x**2
            MPYF    @r_1,R6             ; R6 = (1-r) * x**2
            LDF     @r,R3
            MPYF    @var,R3             ; R3 = r * var(n-1)
;
;   Compute filter output y(n)
;
            LDF     0.0,R2              ; R2 = 0.0
            MPYF3   *AR0++(1)%,*AR1++(1)%,R1
        ||  ADDF    R6,R3
            STF     R3,@var             ; restore var(n)
            RPTS    order-2
            MPYF3   *AR0++(1)%,*AR1++(1)%,R1
        ||  ADDF3   R1,R2,R2            ; y(n) = w[].x[]
            ADDF    R1,R2               ; include last result
;
;   Compute error signal e(n)
;
            SUBF    R2,R7               ; e(n) = d(n) - y(n)
;
;   Update weights w(n)
;
            PUSHF   R3                  ; compute  1/var(n)
            POP     R2                  ; var(n) = a * 2**e
            ASH     -24,R2
            NEGI    R2
            SUBI    1,R2                ; Now we have -e-1, the exponent of x[0].
            ASH     24,R2
            PUSH    R2
            POPF    R2                  ; Now R2 = x[0] = 1.0 * 2**(-e-1).
;
```

```
            MPYF    R2,R3,R0            ; R0 = v * x[0]
            SUBRF   2.0,R0              ; R0 = 2.0 - v * x[0]
            MPYF    R0,R2               ; R2 = x[1] = x[0] * (2.0 - v * x[0])

            MPYF    R2,R3,R0            ; R0 = v * x[1]
            SUBRF   2.0,R0              ; R0 = 2.0 - v * x[1]
            MPYF    R0,R2               ; R2 = x[2] = x[1] * (2.0 - v * x[1])

            MPYF    R2,R3,R0            ; R0 = v * x[2]
            SUBRF   2.0,R0              ; R0 = 2.0 - v * x[2]
            MPYF    R0,R2               ; R2 = x[3] = x[2] * (2.0 - v * x[2])

            MPYF    R2,R3,R0            ; R0 = v * x[3]
            SUBRF   2.0,R0              ; R0 = 2.0 - v * x[3]
            MPYF    R0,R2               ; R2 = x[4] = x[3] * (2.0 - v * x[3])

            RND     R2                  ; This minimizes error in the LSBs.
;
            SUBRF   1.0,R0              ; R0 = 1.0 - v * x[4] = 0.0..01... => 0
            MPYF    R2,R0               ; R0 = x[4] * (1.0 - v * x[4])
            ADDF    R0,R2               ; R0 = x[5] = (x[4]*(1.0-(v*x[4])))+x[4]

            RND     R2,R0               ; Round since this is follow by a MPYF.
;
            MPYF    @u,R7               ; R7 = e(n) * u
            MPYF    R0,R7               ; R7 = err = e(n) * u / var(n)
            MPYF3   *AR0++(1)%,R7,R1    ; R1 = err * x(n)
            LDI     order-3,RC          ; initialize repeat counter
            RPTB    WEIGHT              ; do i = 0, N-3
            MPYF3   *AR0++(1)%,R7,R1    ; R1 = err * x(n-i-1)
        ||  ADDF3   *AR1,R1,R2          ; R2 = wi(n) + err * x(n-i)
WEIGHT      STF     R2,*AR1++(1)%       ; update wi(n+1)
            MPYF3   *AR0,R7,R1          ; for i = N - 2
        ||  ADDF3   *AR1,R1,R2
            STF     R2,*AR1++(1)%       ; update wi(n+1)
            ADDF3   *AR1,R1,R2
            STF     R2,*AR1++(1)%       ; update last w
;
;   Define constants
;
xn          .usect  "buffer",order
wn          .usect  "coeffs",order
u           .usect  "vars",1
var         .usect  "vars",1
r           .usect  "vars",1
r_1         .usect  "vars",1
            .end
```

Tillegg F

Blokkdiagram av TMS320C3x

I figur F er det vist et funksjonelt blokkdiagram som viser forholdet mellom de forskjellige komponentene i den digitale signalprosessoren TMS320C3x.