NTNU
Det skapende universitet

Lydutbredelse i havområder med avstandsavhengig oseanografi

Håvar Slåttrem Olsen

Master i elektronikk
Oppgaven levert: Juli 2008
Hovedveileder: Ulf R Kristiansen, IET
Biveileder(e): Karl Thomas Hjelmervik, Forsvarets forskningsinstitutt
Trond Jenserud, Forsvarets forskningsinstitutt

Norges teknisk-naturvitenskapelige universitet
Institutt for elektronikk og telekommunikasjon

Oppgavetekst

Oppgaven skal fokusere på beregningsmodeller for lydforplantning i områder med avstandavhengig oseanografi. Spesielt er det interessant å sammenligne resultat beregnet med strålegangsprogrammene LYBIN og RAY5.
Kandidaten skal ta utgangspunkt i et eller flere sett måledata fra Forsvarets forskningsinstitutt (FFI), der lydbaner og transmisjonstap skal beregnes med begge metodene.
I tillegg kan det tas utgangspunkt i analytisk beskrevne "benchmark" situasjoner.
Mer detaljert skal oppgaven inneholde følgende :

- En litteraturundersøkelse på området lydutbredelse i områder med avstandsavhengig oseanografi.
- Kjøre Lybin og Ray5 i avstandsuavhengig test case med referanseløsning (Oases). Forslag: flat bunn, 200m bunndyp, 10km rekkevidde, frekvens på 1000Hz, fluid halvrom (1800m/s og 1800kg/m3).
- Det skal finnes en relevant metode for romlig glatting av datasett.
- Det skal undersøkes hvordan RAY5 og LYBIN må settes opp slik at relevant sammenligning av resultat kan utføres.
- LYBIN inkluderer oseanografisk variabilitet ved å blokkoppdele beregningsområdet. Det skal undersøkes effekten av antall blokker som brukes.
- Videre skal det undersøkes hvilken effekt andre metoder for å inkludere avstandavhengighet i LYBIN har å si på transmisjonstap. Det tenkes her på gjennomsnittlige hastighetsprofiler, eller midlete transmisjonstap basert på mange hastighetsprofiler.

Oppgaven gitt: 05. februar 2008
Hovedveileder: Ulf R Kristiansen, IET

LYDUTBREDELSE I HAVOMRÅDER MED AVSTANDSAVHENGING OSEANOGRAFI

Håvar Slåttrem Olsen

15. Juli, 2008

Masteroppgave

Institutt for elektronikk og
telekommunikasjon
Norges Teknisk-Naturvitenskapelige Universitet

Hovedveileder: **Ulf Kristiansen**
Ekstern veileder: **Trond Jenserud**
Ekstern veileder: **Karl Thomas Hjelmervik**

Sammendrag

Normalt går man ut fra at lydhastigheten i havet ikke er avhengig av avstand. Det er vanlig å anta en lydhastighetsprofil som kun avhenger av dybden og la den være gyldig i et større område slik at dette blir grunnlag for eventuelle beregninger. Dette er selvfølgelig en forenkling. I noen tilfeller kan sågar lydhastigheten varierere granske kraftig som funksjon av avstand. En litteraturstudie er gjort som underbygger dette faktum. Studien har tatt for seg noen av de typer avstandsavhegige fenomener som er observert og hvordan disse kan påvirke lydutbredelsen. Ikke sjelden er påvirkningen vist å være ganske kraftig.

Den raske strålegangsmodellen Lybin som er utviklet og brukes av forsvaret, er tidligere testet opp mot en annen strålegangsmodell, Ray5, for å se hvordan hvordan den klarer seg for strålegangsberegninger i et havområde med avstandsavhengig oseanografi. Sammenligningen den gangen viste at ytterligere undersøkelser var nødvendige. En del tiltak ble gjort for å forsøke å få sammenlignbare resultater fra de to modellene. Ett tiltak skilte seg ut og førte til at man fikk god sammenheng mellom resultatene fra modellene. Dette var en glattemetode som ble brukt på de avstandsavhengige lyddataene beregningene ble gjort på grunnlag av.

Innhold

Kapittel 1

Innledning

Denne oppgaven er basert på en tidligere prosjektoppgave der målet var å sammenligne forsvarets raske og kommersielle programvare for strålegangsberegning for lyd under vann, Lybin med en i teorien mer nøyaktig metode, Ray5. Sammenligningen ble gjort på grunnlag av reelle, målte lydhastighetsdata. Disse lydhastighetsdataene var avstandsavhengige og mye av målet med oppgaven var å sjekke hvor godt Lybin ville klare å beregne strålegangen i et havområde med avstandsavhengig lydhastighet i forhold til den mer nøyaktige Ray5-metoden. I de fleste tilfeller regner man med at lydhastigheten kun varierer med dybden, men Lybin har også muligheten til å ta hensyn til avstandsavhengighet. Resultatene fra sammenligningene i prosjektoppgaven ga ikke helt klare svar. Ting kunne tyde på at programmene ikke hadde fått så like forutsetninger som mulig for å kunne sammenlignes på en god måte for de avstandsavhengige dataene. Følgelig ble det besluttet å jobbe videre med problemstillingen.

For å sørge for at programmene hadde så like arbeidvilkår som mulig ble bestemt å gjøre en del tiltak:

1. Et forsøk på å forbedre metoden som ble brukt for å beregne transmisjonstap fra Ray5-strålegangsberegninger skulle gjøres. Denne metoden var basert på en ganske enkel telling av strålekrysninger med en mottakerdybde. Alle stråler ble vektet likt uansett hvor langt de hadde gått, og en trolig forbedring av funsksjonen ville være å ta hensyn til nettopp ganglengde.

2. Det skulle gjøres en sammenligning av beregnet transmisjonstap både fra Ray5 og Lybin mot en referansemodell, Oases, for en kun dybdeavhengig hastighetsprofil. Dette for å sjekke hvor like resultater Ray5

og Lybin ga, og om de ga resultater som var sammenlignbare med Oases.

3. En metode for å glatte hastighetsdataene skulle finnes. Det var nemlig mistanke om at hastighetstabellene i datasettet ikke var helt glatte. Dette vil spesielt forstyrre strålegangsberegningen i Ray5.

4. Det skulle sjekkes hvilken effekt blokkindeling i Lybin hadde. I Lybin realiseres avstandsavhengighet ved å dele beregningsområdet inn i avstandsblokker. Hver slik avstandsblokk har tildelt en dybdeavhengig lydhastighetsprofil. Målet var å finne ut hvor mange slike blokker det egentlig var nødvendig å bruke for hastighetsprofilene fra datasettet.

5. En metode for å gjøre beregninger i Lybin på grunnlag av en gjennomsnittlig hastighetsprofil skulle testes. Grunnen til at dette skulle gjøres, var at man ville sjekke om resultater som var like sammenlignbare med Ray5 kunne oppnås ved å helt enkelt bruke en gjennomsnittlig hastighetsprofil

I tillegg ble det bestemt at en litteraturstudie på området avstandsavhengig oseanografi skulle gjennomføres. Dette ville være viktig både for å se hvordan andre har angrepet problemstillingen tidligere, men også for å understreke at avstandsavhengig oseanografi er noe det ofte kan være nødvendig å ta hensyn til.

Kapittel 2

Litteraturstudie

Det er gjort en del undersøkelser rundt problemstillingen avstandsavhengig oseanografi og påvirkning på lydutbredelse. Grovt sett kan en si at det arbeidet som som er gjort søker å finne påvirkningen fra ett eller begge av de to obeserverte fenomenene interne bølger og solitoner. I det videre skal noen begrep gås gjennom før det blir tatt en nærmere kikk på en del arbeid som er utført for se på hvordan avstandsavhengig oseanografi virker på lydutredelse under vann.

2.1 Viktige begrep

Termoklinen er en definert som det en grenseregion i havdypet der vanntemperaturen endrer seg hurtig. I havet er det gjerne et topplag som er varmet opp av sollys og blandet av vind og bølger der temperaturen er ganske jevn. Temperaturgradienten er altså ganske svak her. Under dette laget, i en dybde på 100-400 meter begynner termoklinen. Her er temperaturgradienten ganske sterk, og vanntemperaturen kan falle så mye som 20 grader Celsius på 150 meter. Under dette ligger dypere havlag der temperaturen er ganske stabil og lav (mellom 0 og 3 grader Celsius). I havområder der overflatetemperaturen er ekstremt lav, slik som i polare områder, vil ikke temperaturgradienten i termoklinen være spesielt sterk.

Interne bølger er et fenomen som gir seg utslag i bølger internt i vannlagene i havet. Bølgene vises normalt altså ikke på overflaten men holder seg innenfor et dybdeintervall. Ofte er det tidevannsstrømmer som skaper interne bølger.

Solitoner er en form for interne bølger men skiller seg fra disse ved at de

gjerne bare utgjør én periode av en en bølge og forplanter seg innenfor et havdyp.

2.2 Tidlige studier

Etter krigen og fram mot begynnelsen av sekstitallet kom noen toneangivende studier av avstandsavhengige variasjoner i sjøtemperaturen og hvordan dette kunne påvirke lydutbredelse under vann. Hvorfor dette skjedde på denne tiden kan man jo spekulere i, men å anta at det hadde en sammenheng med ubåtkrigføring og en gryende tilgang på større regnekraft (les datamaskiner) til å utføre nødvendige simuleringer er nok ikke så dumt. Vel, antagelser kan egentlig spares til annen litteratur. Disse tidlige arbeidene så i alle fall i første rekke på dette med interne bølger.

I en forskningsartikkel publisert i 1961 så Owen S. Lee nærmere på temperaturvariasjoner i sjøvannet i et grunt havområde med dybde på ca 60 meter utenfor kysten av San Diego [1]. Måleapparater som registerte den varierende dybden til en isoterm på et bestemt sted ble utplassert, og data ble samlet inn fra disse. Hensikten med dette var å finne ut om det var mulig å finne periodisitet i temperaturvariasjonene, altså interne bølger. Periodisitet ble for det første funnet i form av variasjoner i dybden til isotermen med perioder på 5 til 15 minutter. I tillegg ser det ut som Lee har observert solitoner, selv om dette ordet ikke blir brukt. Det som blir presentert er enkeltstående perioder av bølger som beveger seg forbi måleapparatene. Disse har tilsvarende utbredelse i tid som de nevnte interne bølgene. Videre ble det funnet variasjoner med periodetid som var sammenlignbar med tidevannsyklusen. Disse interne bølgene hadde dog ikke samme fase som, eller en konstant faseforskjell i forhold til den observerte tidevannssyklusen. Uansett hadde Lee funnet at det var en del periodiske variasjoner i temperaturen i havdybden.

Samme år publiserte Lee en annen artikkel [2] der vinklingen var på hvordan interne bølger ville påvirke lydutbredelse i havet. Bølgene han så på her var av samme type som de han hadde observert utenfor kysten av San Diego. En todimensjonal havmodell med 60 meters dyp ble laget. Med utganngspunkt i denne modellen ble det gjort strålegangs- og intensitetsberegninger for en sinusformet intern bølge i termoklinen. Denne bølgen hadde en periode med tilnærmet lik størrelse som de observerte bølgene. Det ble også gjort beregninger for samme modell uten en intern bølge i termoklinen for å kunne gjøre

sammenligninger. En konklusjon som kunne trekkes fra undersøkelsene var at den interne bølgen hadde signifikant innvirkning på beregnet strålegang. Videre kunne en se at den interne bølgen førte til mer irregulære intensitetstap.

Ira Dyer tok litt senere i [3] for seg hvordan lyd forplanter seg gjennom mange forskjellige veier gjennom vannet. Han så statistisk på hvordan multiple gangveier for lyden og spredning ville påvirke et lydsignal under vann. Det ble vist at å ta hensyn til at lyden tar mange forskjellige veier fra kilden til mottakeren gir et høyere bidrag til å øke standardavviket for mottatt signal enn å ta hensyn til spredning. Dyer laget også en modell for bakgrunnsstøy med ved å utvide modellen for én tone. En modellering av bakgrunnsstøy som flere toner ga han muligheten til å teste hvordan støy påvirket signalet. Det viste seg at signal og støy sammen kan gi mye høyere standardavvik i for mottatt signal. Det vil altså kunne skje kraftigere både positiv og negativ interferens som i en gitt situasjon kan utnyttes til å øke muligheten til å motta et signal.

De tre artiklene fra Lee og Dyer legger et slags grunnlag for denne litteraturstudien. Rapportene dokumenterer avstandsavhengige fenomener under vann, som interne bølger og solitoner. I tillegg gjorde Lee et tidlig forsøk på finne ut hvordan interne bølger påvirker lydutbredelsen under vann. Dyers artikkel er et godt eksempel på hvordan et statistisk utgangspunkt kan være veldig nyttig for å se på variasjoner i lydutbredelsen under vann.

2.3 Interne bølger og solitiners påvirkning på undervanns lydutbredelse

DeFerrari så i [4] på måleresultater over 48 timer med en 406 Hz kilde på et gitt, grunt dyp og en mottaker 700 NM unna og under den dype lydkanalen. Han registrerte et veldig fluktuerende transmisjonstap med varisjoner på inntil 40 dB og en mer saktevarierende faseendring. Dette ble forsøkt forklart ved hjelp av tre mulige modeller for interne bølger. Den første forklaringsmodellen antok interne bølger med mye større bølgelengde enn avstanden mellom kilde og mottaker. Den andre modellen gikk ut fra interne bølger med mindre bølgelengde enn avstanden, men større enn bølgelengden til lydbølgen fra kilden. Den siste og tredje modellen antok interne bølger med bølgelengde i samme størrelsesorden som lydbølgen, altså 3-4 meter lange interne bølger. Disse tre modellene ble brukt i siumuleringer for å se om han klarte å gjen-

skape de observerte transmisjonstap og faseendringer. Det viste seg at den første modellen ikke ga resultat i nærheten av det som var observert. Den andre modellen for interne bølger ga delvis lignende resultater. For å få like kraftige fluktuasjoner i transmisjonstapet som fra målingene måtte interne bølger, med bølgelengde i samme størrelsesorden som lydbølgen fra kilden, introduseres i simuleringene.

I [5] prøvde Porter et al. å forklare sakte varierende fasendringer og sterkt varierende mottatt amplitude for lyd som har propagert langt i havet. De antok at variasjonen i amplitude skyldes at lydbølgefronter med forskellig løpeavstand og dermed fase kom fram til mottakeren samtidig og i varierende grad forsterket eller utlignet hverandre. Ut fra denne tanken lagde de en strålegangsmodell der et spektrum av interne bølger kunne integreres. I modellen ble fasevariasjon over tid beregnet ut fra at en stråle ville få en viss faseendring hver gang den krysset en intern bølge. Håpet var at de fra modellen kunne få en god prediksjon av hvordan mottatt amplitude og fase fra en kilde langt unna ville variere over tid ved mottakeren. Simuleringene de gjorde med modellen, ga resultater som i alle fall for fasen stemte godt med observasjoner gjort av samme team i [6].

Mens DeFerrari kun så på én intern bølge og Porter et al. så på et spektrum av bølger, men kun i et veldig tynt lag, gikk Flatte og Tappert litt annerledes til verks for å prøve å finne ut ut hvordan interne bølger påvirker lydtransmisjon i havet [7]. De valgte seg et havdyp på 4 km med et lydkanal på 1 km for sin modell. For å beregne lydfeltet i modellen ble en parabolic equation (PE) metode brukt. Det dybde- og avstansavhengige lydhastighetsfeltet som denne metoden skulle bruke, var skapt ved å gå ut fra en ganske typisk Munk-profil. Det som ble gjort videre var å legge et tilfeldig spektrum av interne bølger til denne hastighetsprofilen, og ikke kun én intern bølge. På denne måten fikk de realisert et lydhastighetsfelt påvirket av interne bølger slik at det i tillegg til å være dybdeavhengig også var avstands- og tidsavhengig. De første simuleringene i dette feltet ble gjort med utgangspunkt i en strålegangsberegning på grunnlag av den kun dybdeavhengige Munk-profilen. For ståler med de fire forskjellige utgangsvinklene $-10\,^\circ$,$-5\,^\circ$,$0\,^\circ$ og $5\,^\circ$ fra kilden ble stråleplottet brukt til å finne hvor disse strålene krysset de fire avstandene 20, 60, 100 og 250 km. I disse punktene ble det plassert tenkte hydrofoner i modellen og PE-metoden ble deretter kjørt for finne transmisjonstapet over 128 timer i hydrofonpunktene ved lydutbredelse gjennom det avstandsavhengige feltet. Over 128 timer ble det observert signifikante svingninger i transmisjonstapet. Beregninger ble gjort på nytt med simulering av 700 meters vertikale hydrofonarrayer som mottakere i stedet for enkle hydrofoner,

og dette ga enda kraftigere variasjoner i transmisjonstapet. Variasjoner på 5-30 dB ble observert og dette var sammenlignbart med observerte verdier fra felteksperimenter. Kildefrekvensen som ble brukt var 100 Hz.

I [8] så Baxter og Orr først på en virkelig, observert intern bølge i nitti meters havdyp. Lydhastigheten i et havområde med en slik registrert bølge kan ikke beskrives av en enkel matematisk funksjon. Derimot kan den ganske enkelt beskrives av en endelig oppløst matrise med lydhastigheter. Dette valgte Baxter og Orr å gjøre. For å beregne strålegang i et slikt tilfelle gikk de ut fra den eikonale ligningen [1]. Denne ble løst stegvis for hver stråle ved hjelp av en fjerdeordens Runge-Kutta metode. For at en slik stegvis numerisk metode skulle fungere, måtte mellompunktsverdier den etterspurte interpoleres ut fra lydhastighetmatrisen.

Før den numeriske metoden ble sluppet løs på den målte hastighetsprofilen måtte den verifiseres. Dette ble gjort ved å beregne strålegangsplot med Lees metode ut fra de matematiske uttrykkene for hastighetsprofilene han brukte i [2], og sammenligne med stråleberegninger fra den numeriske metoden på grunnlag av de samme profilene, men nå realisert i en lydhastighetsmatrise. Forskjellene viste seg å være ubetytelige mellom metodene. Det samme ble gjort for hastighetsprofiler som kan løses eksakt, og den numeriske metoden viste seg her å være mer nøyaktig enn Lees metode.

Baxter og Orr lagde også funksjonalitet for å beregne intensitet i sin strålegangsmodell. Dette ble gjort ut fra det forholdsvis enkle prinsippet om at utstrålt energi fra kilden vil holde seg konstant og lik mellom hvert par av nabostråler. Forutsetningen er selvfølgelig at stråler blir sendt ut fra kilden i like vinkelintervaller. Prinsippet ble utnyttet til å finne intensiteten i et avgrenset, lite område ut fra avstanden mellom to nabostråler som går gjennom området. Resultatene fra simuleringene i det målte lydhastighetsfeltet med intern bølge viste kraftig innvirkning på strålegangen i forhold til en avstandsuavhengig lydhastighetsprofil. Profilen det ble sammenlignet med er en tilnærming til lydhastigheten i det samme området uten påtrykk fra den interne bølgen. Det ble fra beregningene observert en forskjell i intensitet mellom de to hastighetsprofilene på så mye som inntil 20 dB.

[1] *Eikonal equation* på engelsk. Det er vanskelig å finne gode norske oversettelser. I [9] er ligningen gitt på side 136.

2.4 Modal kobling

Zhou, Zhang og Rogers beskriver i [10] observasjoner av det de kaller abnorme variasjoner i transmisjonstapet ved målinger av lydutbredelse i det grunne Gulehavet. Endringer i transmisjonstapet på inntil 20 dB ble noen ganger observert for en signalfrekvens på rundt 600 Hz, og andre ganger for andre frekvenser. Det ble mistenkt at dette skyldtes resonanser i den modale koblingen mellom lydbølgen fra kilden og solitoner og interne bølgepakker som var observert i området. For å sjekke dette ble det gjort simuleringer av lydutbredelsen i en modell med justerbar kildefrekvens, solitonfrekvens og bølgepakkelengde. Simuleringen viste at alle de justerbare parametrene kunne føre til resonanser i den modale koblingen. Dette kunne igjen gi store innvirkninger på transmisjonstapet.

Chin-Bing et al. [11] fortsatte undersøkelsene til Zhou et al. ti år etterpå. Utgangspunktet ble tatt en i modell av samme grunne havområde som tar hensyn til bunntopografi og tidevansstrømmer. Simuleringer gjort med modellen viser at det blir generert solitoner i havområdet. Dette blir bekreftet av satelittbilder av området som viser tilsvarende solitoner. Videre simuleringer med lydutbredelse i modellen viser at det for en resonansefrekvens er en modal kobling med solitonene som gjør at enkeltmoder blir veldig dempet. Transmisjonstapet for signal sendt på denne resonansfrekvensen viser seg å være kraftigere enn for andre frekvense.

I [12] har Tielburger et al. også sett på dette med modal kobling i grunne havområder i en statistisk modell for lydhastighetene i et havområde. Både interne bølger og solitoner er inkludert i denne modellen. Det er gått grundig til verks og undersøkt med både flat og skrånende bunn og med lydforplantning over forskjellig avstander. Ut fra den stastistiske modellen er det så forsøkt å si noe om hvordan modal kobling mellom lydbølgen fra kilde og interne bølger og solitoner påvirker lydforplantningen og variasjonen i mottatt lydintensitet. Konklusjonen er at variasjonen i mottatt intensitet, som i artikkelen blir tallgitt ved en blinkeindex[2], stiger expontentielt med avstanden. I klartekst betyr dette at sterkt varierende mottatt signalstyrke kan forventes ved lydforplantning over store avstander i grunt vann med interne bølger.

[2]Oversatt fra det engelske uttrykket *scintillation indeks*.

Kapittel 3

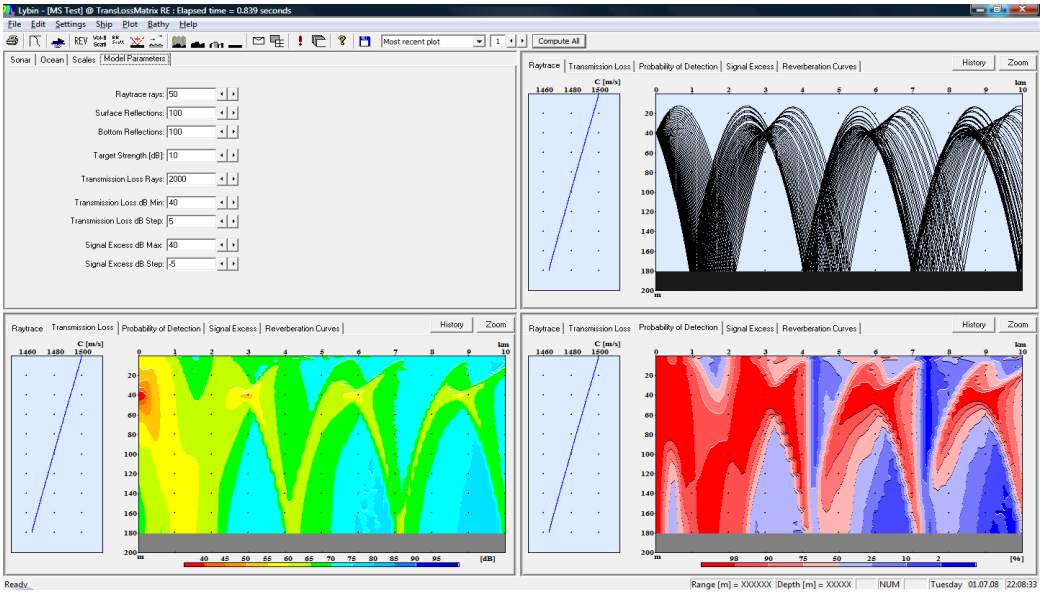
Programvare og avstandsavhengig lyddata

Det er brukt tre forskjellige programmer for å produsere resultater for strålegangen og transmisjonstapet ut fra målte lydhastighetsprofiler. Programmene som er brukt er Lybin, Ray5 og Oases. I de etterfølgende delkapitler vil programmene bli presentert. Til sist i kapitlet vil bakgrunnen til og særtrekkene ved lydhastighetsprofilene bli gjennomgått.

3.1 Lybin

Det er nyttig å kunne estimere hvor god mulighet man har til å i forskjellige retninger og avstander detektere objekter som befinner seg under vann ved hjelp avstrålt eller reflektert lyd. Det klassiske eksempelet på dette er en potensiell neddykket ubåt som man prøver å oppdage fra sin splitter nye fregatt ved hjelp av aktiv eller passiv sonar. Når man søker etter en slik ubåt er det viktig å vite hvilke baner lyden har fulgt til og fra ubåten og i hvilke områder man i det hele tatt har mulighet for for å finne den. Lybin er et slikt verktøy med en stålegangsmodell i bunn som raskt kan estimere hvor godt en sonar kan yte.

Opprinnelig ble Lybin laget av Svein Mjølsnes ved Forsvarets Logistikkorganisasjon. Nå er det Forsvarets forskningsinstitutt som videreutvikler og raffinerer programvaren. Lybin framstår i nåværende utgave som et brukervennlig og fleksibelt program for sonarytelsesmåler. Figur 3.1 viser et skjermbilde fra hovedprogramvinduet i Lybin.



Figur 3.1: Skjermbilde fra Lybin.

3.1.1 Virkemåte

Lybin baserer seg på en løsning av bølgeligningen som gir ut strålebaner i områder med konstant vertikal lydhastighetsgradient, og ingen slik horisontal gradient. For disse begrensningene viser det seg at lydstråler vil følge sirkelbuede baner. Løsningen bygger på noen tilnærminger som forutsetter rimelig høye frekvenser og tar ikke vare på faseinformasjon. Framgangsmåten for å komme fram til en slik løsning gjennomgås av Hovem i kapittel 6 i [13]. Løsningen han har kommet fram til for en slik sirkulær lydstrålebane i et område med konstant lydhastighetsgradient er da, uttrykt ved radiusen R,

$$R = \frac{1}{|g|} \frac{c(z)}{cos\theta(z)}. \tag{3.1}$$

Her er $c(z)$ lydhastigheten som funksjon av dybden z, g den konstante dybdegradienten til lydhastigheten og $\theta(z)$ er vinkelen lydstrålen har med det vannrette plan i en gitt dybde. Med utgangspunkt i dette konstante forholdet er det lett å beregne lydbanen for stråler med en gitt utgangsvinkel i forhold til det horisontale plan. Problemet er at slike enkle lydhastighetsprofiler ikke har noen noen rot i virkelighetens verden. Likevel er metoden nyttig. Ved å dele opp havet i flere lag etter dybden, kan man tilnærme en reell kontinuerlig dybdeavhengig lydhastighetsprofil ved hjelp av lineære profiler i hvert lag. Lybin gjør en slik lagdeling og kan på denne måten beregne strålegangen lagvis veldig kjapt. I utgangspunktet er ikke Lybin laget med tanke på

avstandsavhengighet for lydhastigheten. Dette er dog mulig ved å dele inn havområdet man jobber med i avstandsblokker, og for hver slik avstandsblokk definere en lagvis lydhastighetsprofil.

Lybin har et rent grafisk grensesnitt som er vist i figur 3.1. Vi ser fire plott i figuren, men totalt får man ut seks plott:

Lydhastighetsprofilen Viser et konvensjonelt plott av lydhastighetsprofilen man bruker, med lydhastigheten bortover den horisontale aksen som funksjon av dybden nedover den lodrette aksen.

Strålegang Viser et plott av hvordan området man simulerer i ser ut med tanke på avstand, dybde og bunntopografi, og selvfølgelig hvordan et angitt antall stråler propagerer fra kilden gjennom området.

Transmisjonstap Plott i dB av transmisjonstapet i hele området.

Deteksjonssansynlighet Plott som viser sannsynlighet for deteksjon av objekter med en gitt målstyrke for hele området.

Signaloverskudd Plott i dB for hele området der 0 dB er grensen for om det er mulig å oppdage et mål.

Gjenklangsgrafer Fire kurver som viser gjenklang i dB fra bunn, overflate, volumet og støy over den totale avstanden.

Vi har gått gjennom hvordan strålegangen beregnes, men ikke hvordan man finner transmisjonstapet og de andre plottene som er avledet ut fra det. Det vil her kun bli gitt en kort forklaring av hovedprinsippet bak beregningen avtransmisjonstapet. Dette blir faktisk funnet ut fra et strålegangsplot. Hele beregningsområdet er delt inn i et egendefinert oppløst grid. Dette vil i praksis si at brukeren angir et visst antall celler i dybden og et visst antall kolonner i lengden i Lybin. Transmisjonstapet beregnes så ved hjelp av en algoritme som sjekker energibidraget fra hver hver stråle som krysser gjennom en celle. Strålene blir sendt ut fra kilden i en vifte på 180 °, og bidraget fra hver stråle blir korrigert ut fra en del innvirkende faktorer som sonardirektivitet og tilbakelagt avstand.

3.1.2 Innstillingsmuligheter

Vi har vært inne på noen av dem, men Lybin har en rekke innstillingsmuligheter og parametere som må settes slik at man kan gjøre simuleringer som stemmer mest mulig med situasjonen man ønsker å teste. De fleste listes opp nedenfor, sortert etter kategori

Sensorparametre Sensordybde, tilt, styrken til sidelober, frekvens, direktivitetsindex, nivå, strålevidden til mottakeren og strålevidden til kilden.

Pulsparametre Pulslengde i milisekund og båndbredde i Hz.

Modellparametre Rekkevidde, dybde, antall rekkeviddeceller, antall dybdeceller, antall stråler for beregning av transmisjonstap og antall stråler for strålegangsplot.

Målparameter Målstyrke.

Miljøparametre Vindhastighet, lydhastighetsprofil(er), bunntype og bunntapsvifte.

I det grafiske grensesnittet kan man sette mange av verdiene manuelt og gjøre beregninger. Men dette blir tungvint å holde på med om man for eksempel vil generere mer avanserte lydhastighetsprofiler eller bunntyper. Til å avhjelpe dette fins LybinCom som gir tilgang til Lybin gjennom Matlab. Man kan lage et script som genererer lydprofiler og setter de forskjellige parametrene. Alt dette kan lagres i en .xml fil som Lybin kan lese og derfra laste inn alle variable for simulering. LybinCom gir også muligheten til å kjøre Lybin-algoritmene direkte i Matlab og lagre matriser for transmisjonstap, signaloverskudd, deteksjonssannsynligher og vektorer for gjenklang. Disse kan da plottes og behandles i Matlab. Strålegangsplottet får man derimot ikke tilgang til gjennom Matlab. Et eksempel på et Matlab-script som belyser mange av mulighetene med LybinCom kan sees i B.1.1.

Oppsummert kan vi si at Lybin er en strålegangsmodell som har sin store fordel i at den er meget rask og gir ut intuitive plot. I tillegg er Lybin veldig fleksibelt ved at man har muligheten til å få tilgang til modellen gjennom Matlab-script og .xml-filer med med modelldata. Ulempen er at den benytter seg av en forenklet løsning av bølgeligningen. Egentlig har ikke dette så mye å si for nøyaktigheten av strålegangsberegningene da usikkerheten for nøyaktigheten av lydhastighetsprofilene stort sett har en større innvirkning på beregningene. Løsningen er derimot inkoherent, og dermed kan man ikke ta hensyn til fase ved beregning av transmisjonstapet.

3.2 Ray5

I motsetning til Lybin er ikke Ray5 et kommersielt ferdig program. Ray5 er en Matlab-kode utviklet av Trond Jenserud ved Forsvarets forskningsinstitutt. Koden kan utføre strålegangsberegning for lyd i et område geometrisk

avgrenset av bunn, overflate og horisontal avstand.

Der Lybin bruker en forenklet løsning av bølgeligningen, bruker Ray5 en eksakt løsning av bølgelegningen for å beregne strålegangen. I kapittel 3.2 og delkapittel 3.2.1 i [14] vises det hvordan man kan gå ut fra Helmholz-ligningen i kartesiske koordinater,

$$\nabla^2 p + \frac{\omega^2}{c^2(\vec{x})} p = \delta(\vec{x} - \vec{x_s}), \tag{3.2}$$

der $\vec{x} = (x, y, z)$, og komme fram til de fire straleligningene:

$$\frac{dr}{ds} = c\xi(s) \tag{3.3}$$

$$\frac{d\xi}{ds} = \frac{1}{c^2}\frac{dc}{dr} \tag{3.4}$$

$$\frac{dz}{ds} = c\varsigma(s) \tag{3.5}$$

$$\frac{d\varsigma}{ds} = \frac{1}{c^2}\frac{dc}{dz}. \tag{3.6}$$

Her er $\xi = \frac{\cos\theta}{c(0)}$, $\xi = \frac{\cos\theta}{c(0)}$, mens r og z er strålekoordinatene og c er lydhastigheten. I delkapittel 3.6.1 i [14] er det vist hvordan 3.3-3.6 kan løses ved hjelp av en numerisk metode. Ray5 benytter en slik strategi og løser ligningene stegvis med en adaptiv andreordens Runge-Kutta metode. En gjennomgang av virkemåten til Runge-Kutta metoder kan finnes i delkapittel 3.6.1 i [15]. Den numeriske metoden må hele tiden få tilgang til lydhastigheten $c(r, z)$ og dens første- og andreordens partiellderiverte $c_z(r, z)$, $c_r(r, z)$, $c_{zz}(r, z)$, $c_{rz}(r, z)$ og $c_{rr}(r, z)$ i et punkt for å løse de fire ligningene 3.3-3.6 og beregne videre en liten bit av banen til en lydstråle. I tillegg må et startpunkt og en startvinkel være kjent for en stråle. I klartekst betyr dette at Ray5 trenger disse verdiene tilgjengelig for hele området man skal beregne strålegang i. Den numeriske metoden innfører nødvendigvis en liten feil, men at metoden er adaptiv betyr at feilen gjøres vilkårlig liten ved å tilpasse steglengden.

Det er ikke intensjonen å gå gjennom i detalj hvordan Ray5 fungerer. For å få et grovt overblikk er det greit å se på hva de viktigste funksjonene i programmet gjør:

ray_ init.m I denne funksjonen setter man alle parametre og forutsetninger for strålegangsberegning med Ray5. De viktigste er kildeavstand og -dybde, mottakerrekkevidde, strålevifte (utgangsvinkler i grader i forhold

til det horisontale plan), maksimal beregningsrekevidde, batymetri og overflatetopografi (mulighetene er for eksempel bølget eller flat havoverflate). En del andre styringsverdier for plotting og den numeriske metoden settes også i *ray_ init.m*. Når funksjonen blir kallet returnerer den stuctures med alle variablene.

ssp.m Oppgaven til denne funksjonen er å ta inn en avstands- og dybdekoordinat og returnere lydhastigheten og dens første- og andreordens partiellderiverte i dette punktet. Siden den numeriske metoden av og til etterspør verdier like utenfor det definerte beregningsområdet når den skal håndtere et topp- eller bunntreff, må *ssp.m* kunne levere verdier herfra også.

ode.m Dette er den numeriske algoritmen som løser ligningene 3.3 - 3.3 på et gitt sted for en lydstråle. Funksjonen tilpasser steglengden mellom startpunktet og det nye estimerte punnktet for å ikke overstige en feilmargin angitt i *ray_ init.m*.

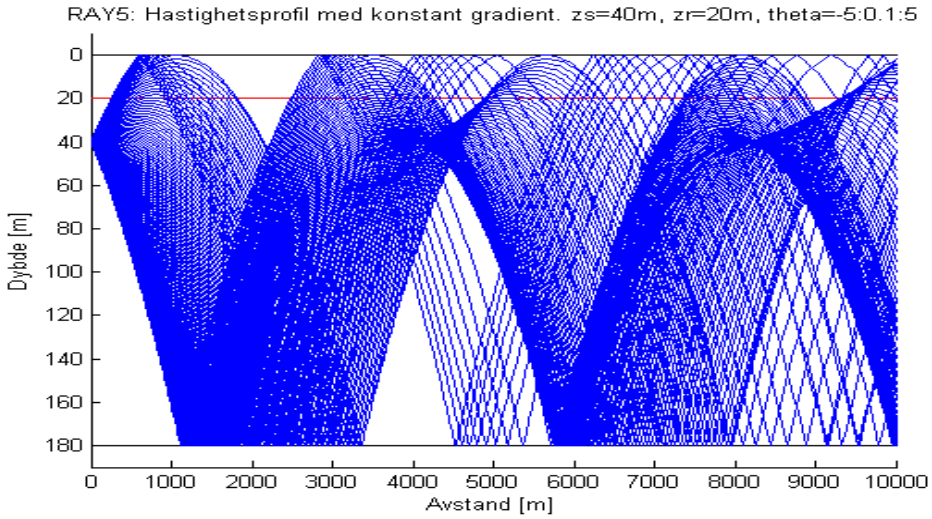
ray_ plot.m Funksjonen tar seg kort og godt av all plotting i Ray5. Det er for eksempel mulig å angi i i *ray_ init.m* at hastighetsprofilen skal plottes i bakgrunnen av stråleplottet, og at strålene skal plottes stegvis etterhvert som den numeriske metoden skrider fram.

ray5.m Dette er hovedfunksjonen som styrer gangen i hele Ray5. Den sørger for stegvis beregning av strålebanene. Hvert treff med topp og bunn blir tatt hånd om som refleksjoner. Koordinatene for disse blir også lagret og det samme blir strålekrysninger med mottakerdybden. Funksjonen sørger også for å lagre hvert beregnede strålekoordinat for plotting av strålegangen. I tillegg til de ovenstående funksjonen har *ray5.m* mange underfunksjoner den kaller. Disse vil ikke bli gjennomgått.

De fem Matlab-funksjonene ovenfor er vedlagt i B.2.1-B.2.5. I figur 3.2 er et eksempel på strålegangsplottet fra Ray5 vist for en veldig enkel lydhastighetsprofil med konstant gradient i dybden og ingen avstandsavhengighet. Det er forøvrig denne profilen som er realisert i de vedlagte funksjonene B.2.1 og B.2.2.

3.3 Oases

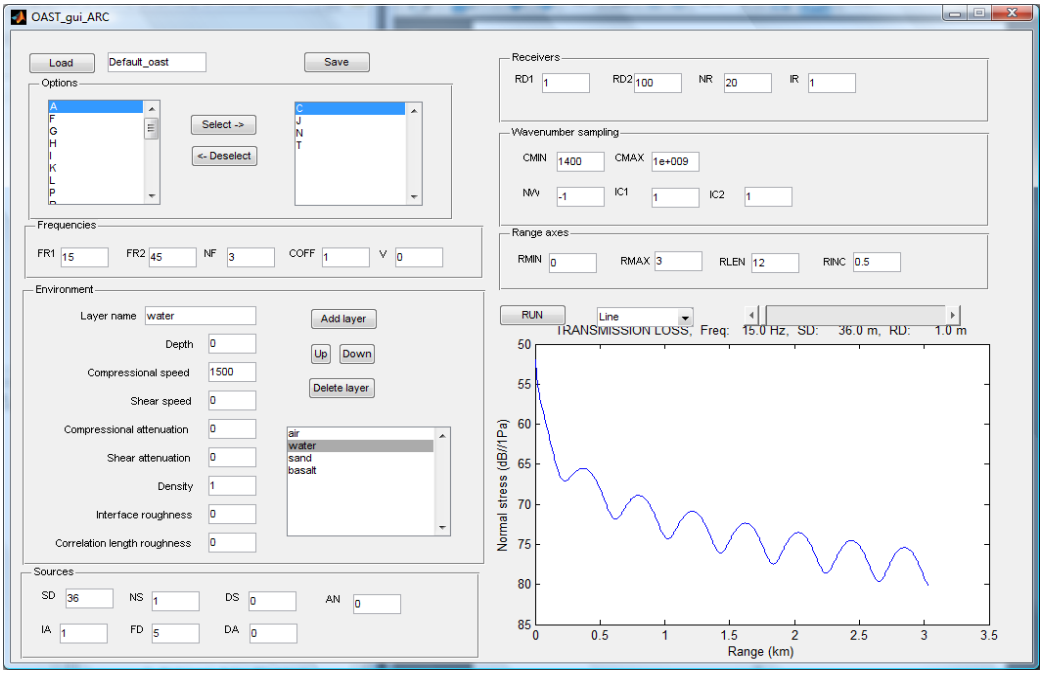
Oases er ikke en strålegangsmodell slik som Lybin og Ray5. Modellen er laget for å beregne lydutbredelse i avgrensede områder med avstandsuavhengig oseanografi. Den er mye brukt i akustiske og seismiske simuleringer og er basert



Figur 3.2: Ray5 strålegangsberegning for lydhastighetsprofil med konstant gradient.

på SAFARI som er en forutgående modell distribuert av SACLANTCEN. Det vil ikke bli gjennomgått hvordan Oases virker, men den kan regnes som en modell som kan gi referanseløsninger for lydfeltet. I dette tilfellet er det særlig muligheten til å produsere en referanseløsning for transmisjonstapet som er aktuell.

Ved NTNU har det blitt laget et grafisk brukergrensesnitt for blant annet transmisjonstapsmodulen til Oases (se figur 3.3). Dette gir muligheten til å enkelt kunne legge inn en ønsket modell og få beregnet transmisjonstapet. Det er mange parametre som er mulig å justere i Oases. I denne omgang er det noen parametre som er mer relevante enn andre. Modellen man lager må ha en viss horisontal avstand og dybde, flat bunn og bestå av ett eller flere horisontale lag. Lagene kan ha en en rekke parametre. Det vil ikke bli aktuelt å sette andre typer lag enn vannlag og for disse må lydhastigheten settes sammen med tettheten. For kilden, eller eventuelt kildene, må ønsket frekvens settes og selvfølgelig må posisjonen bestemmes. I tillegg kan én eller flere mottakere velges og disse må settes i ønsket dybde. Linjemottaker er et valg for mottakertype. Ved å sette disse parametrene etter ønske, skal det være mulig å produsere transmisjonstapsplott for sammenligning mot Ray5 og Lybin.

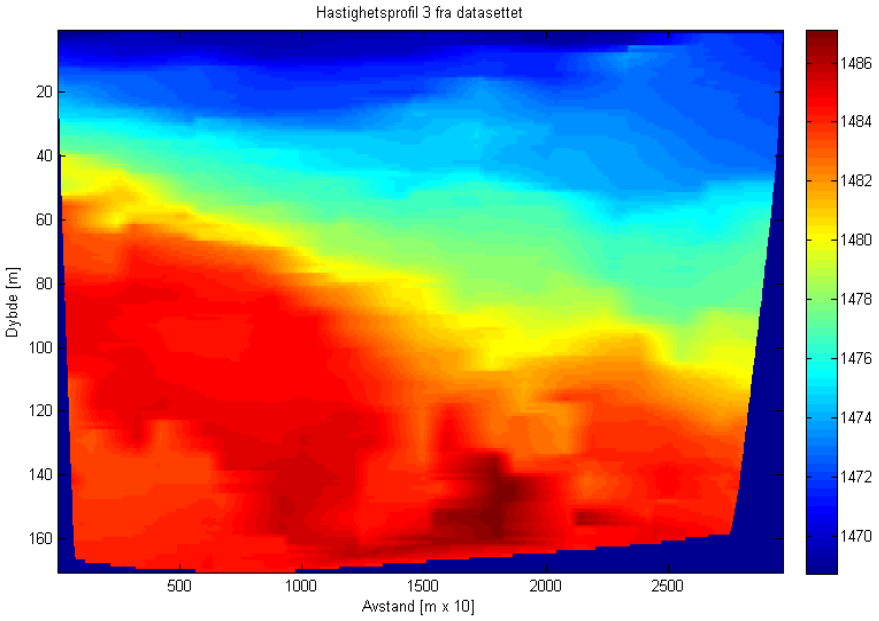


Figur 3.3: Oases GUI for transmisjonstapsmodulen.

3.4 Datasett med lydhastighetsmålinger

For å teste hvordan Lybin håndterer avstandsavhengig lydhastighet skulle det brukes tabeller med ekte målinger av lydhastigheten. Disse målingene er gjort mellom 2. og 5. mars 2007 utenfor vestlandskysten. Mer presist mellom lengdegrad 4,0 og 4,5 og mellom breddegrad 60,00 og 60,17. De seks linjene målingene ble gjort langs hadde en innbyrdes avstand på ca 3,7 km. Målingene ble utført ved å kjøre en båt etter rette øst-vest linjer samtidig som en sensor ble hevet og senket mellom overflate og en bunnposisjon. Sensoren målte lydhastigheten kontinuerlig samtidig som dybden til sensoren og posisjonen til båten ble registrert.

Resultatet fra målingene var et datasett bestående av seks matriser med lydhastigheter. Hver matrise inneholdt ca 3000 kolonner og ca 180 rader. Dette motsvarer målinger over en avstand på 30 km i en dybde på 180 meter. Altså er den horisontale oppløsningen i hastighetsmatrisene 10 meter og den vertikale oppløsningen 1 meter. Det er viktig å merke seg at hver matrise kun er basert på maksimalt 20 målinger i havets fulle dybde. Matrisene er fylt ut ved hjelp av interpolering ut fra måledataene. I figur 3.4 er hastighetsprofil 3 plottet. En ting man kan se mot sidene og mot bunnen av denne



Figur 3.4: Lydhastighetsprofil 3 fra datasettet

figuren er noen mørke blå felter. Dette er tomme felter i hastighetsmatrisen og kommer trolig av de er utenfor området det ble gjort målinger i.

Kapittel 4

Oppsett for sammenligningene og tidligere resultater

I en forutgående prosjektoppgave av undertegnede ble det gjort en del sammenligninger mellom Lybin og Ray5 for hastighetsprofilene i datasettet. Programmene ble sammenlignet både med hensyn på strålegang og transmisjonstap. Dette er mye av basisen for det som er gjort i denne oppgaven og hvorfor det er gjort. Derfor er det nødvendig å gå gjennom hvordan disse sammenligningene ble gjort og hvilke resultater de ga.

4.1 Tilpasninger og oppsett av Ray5 for hastighetprofilene fra datasettet

Ray5 er i utgangspunktet ikke laget for å kunne håndtere lydhastighetsprofiler av den typen vi har fra det innsamlede datasettet. Før stråleberegninger kunne utføres måtte noen justeringer gjøres. Først var det viktig å ta hensyn til at hastighetsmatrisene ikke er helt fullstendige. En veldig enkel metode ble tatt i bruk for å sørge for at alle feltene i den gjeldende hastighetsmatrise fikk en verdi. Alle tomme felter fikk rett og slett samme verdi som det nærmeste feltet med en verdi. En annen ting å huske på er at det må lages matriser for de første- og andreordens partiellderiverte av lydhastigheten. Marie Darrieus, en fransk sommerstudent ved NTNU i 2007, jobbet en del med Ray5 og prøvde også å få programmet til å fungere med hastighetsprofiler fra det målte datasettet. Hun kom ikke helt i mål med dette, men fikk blant annet laget funksjonen *interpolation.m* som tar inn en lydhastighetsmatrise og gir ut matriser med dens første og andreordens partiellderiverte. Koden til denne funksjonen finnes i vedlegg B.2.6. Den stegvise numeriske metoden i Ray5 avhenger som nevnt av å i hvert beregningssteg få verdiene for lydhastigheten

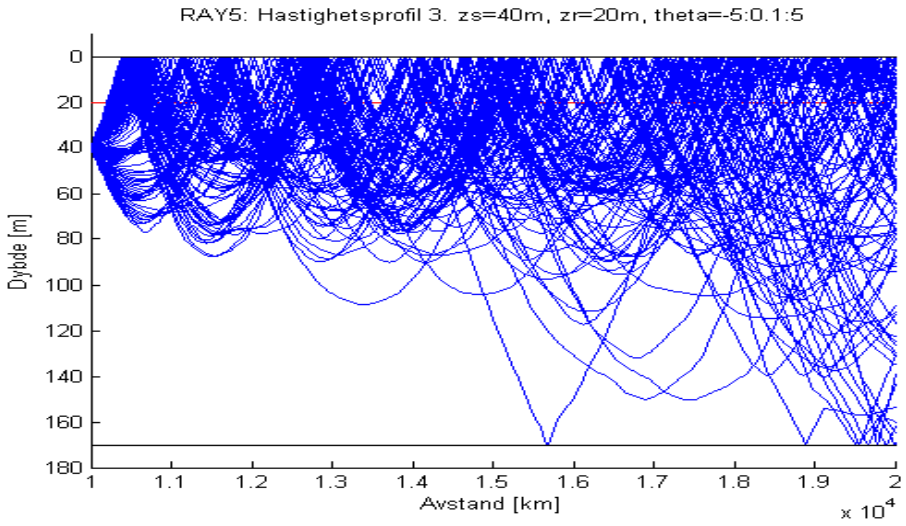
og dens første- og andreordens partiellderivierte i et nøyaktig punkt i beregningsområdet. Disse verdiene er funksjonen *ssp.m* ansvarlig for å gi ut. Når utgangspunktet er diskrete verdier spredt i en ca 200x3000 elementer stor lydhastighetsmatrise betyr det at verdier må interpoleres ut for lydhastighetene som ligger mellom disse punktene. Dette blir løst ved å la *ssp.m* interpolere med den innebygde Matlab-funksjonen *interp2* i hastighetsmatrisen for hver gang *ssp.m* blir kalt fra den numeriske metoden. I tillegg vil den numeriske metoden noen ganger spørre etter verdier like utenfor hastighetsmatrisen. (Dette skyldes en shooting-metode som brukes for å finne treffpunkt med havbunn og og overflate, og deretter starte strålen på nytt som om en refleksjon skulle ha skjedd.) I disse tilfellene gis topp- eller bunnverdien ut fra lydhastighetsmatrisen. *ssp.m* tilpasset for datasettet kan sees i vedlegg B.2.8.

Når de omtalte tilpasningene av Ray5 var gjort som beskrevet over, var det viktig å bestemme seg for en del fornuftige forutsetninger for hvordan Ray5 skulle kjøres med hastighetsprofilene i datasettet. Tabell 4.1 oppsummerer de valgte verdiene for de styrende parameterene. Valget av antall stråler

Antall stråler	100
Vinkelintervall	$-5\,^\circ$ til $5\,^\circ$
Kildedybde	20 m
Mottakerdybde	40 m
Kildeavstand	10000 m
Maksimal avstand	20000 m
Havdybde	Dybden i gjeldende hastighetsprofil
Bunntopografi	Flat

Tabell 4.1: Tabell med verdier til styrende parametre for stråleberegning med Ray5 for datasett 3.

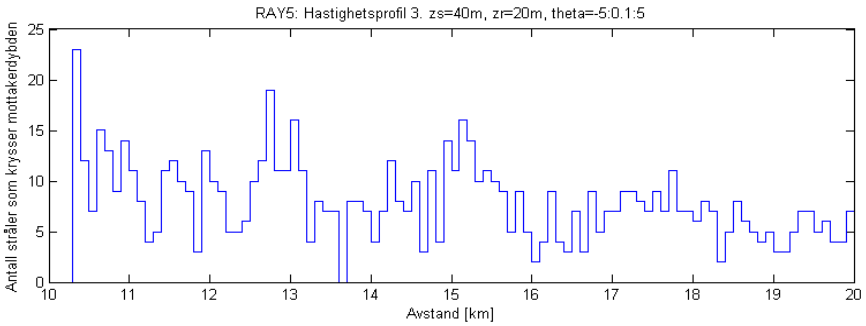
til 100 ble gjort på grunn av at dette ga en beregningstid for strålegangen på fire til fem timer. Det skulle tross alt gjøres en del simuleringer og testinger, så dette ble sett på som et greit kompromiss mellom beregningstid og tilstrekkelig tetthet av stråler for å kunne beregne transmisjonstap. Verdiene for vinkelintervall for strålene, kildedybde og mottakerdybde ble satt som de ble gjort for for å prøve å få ganske mange strålekryssinger med mottakerdybden hele veien. Dette ser ut til å ha lyktes, men det er nok mange andre verdier for disse parametrene som kunne fungert like bra. Verdiene for kildeavstand og maksimal avstand er satt slik for at beregningen skal skje i den midterste delen av den aktuelle hastighetsprofilen. Grunnen til dette var at hastighetsprofilene i datasettet manglet data i en del punkter, spesielt



Figur 4.1: Ray5 strålegangsplot for hastighetsprofil 3.

ut mot endene. Disse punktene ble gitt verdier på en ganske enkel måte, så ved å gjøre beregninger mot midten av datasettet ville man i stor grad unngå stråleberegning på grunnlag av kunstig satte hastighetsverdier. Den neste parameteren, dybden, ble satt til å være lik den maksimale dybden i den målte hastighetsprofilen. Bunntopografien ble også satt til å være flat. Disse to siste valgene ble gjort slik siden det var strålegang med avstandsavhengig lydhastighet som skulle studeres. Dermed var det greiest å ha en enkel havdybde og bunnbeskrivelse å forholde seg til, selv om havet var litt dypere enn dypeste hastighetsmåling, og bunnen nok ikke var helt flat i området målingene ble gjort. I vedlegg B.2.9 er funksjonen *ray_init.m* vist når den er satt opp for hastighetsprofil 3 i datasettet. Figur 4.1 viser det resulterende strålegangsplottet.

I Lybin fins det en ganske god metode for å beregne transmisjonstapet som forklart i avsnitt 3.1.1. I Ray5 er det foreløpig ingen innebygd metode for slik beregning. Derimot har Marie Darrieus laget en enkel metode for transmisjonstapsberegning. Metoden tar utgangspunkt i at Ray5 lagrer koordinatene hver gang en stråle krysser den brukerdefinerte mottakerdybden. Ved så å dele inn beregningsområdet horisontalt i et antall blokker og telle hvor mange strålekrysninger som skjer med mottakerdybden per blokk, så kan man finne et anslag for hvor sterkt signalet er i hver slik avstandsblokk på mottakerdybden. For å få sammenlignbare resultater fra Lybin, kunne transmisjonstap fra samme dybde hentes ut og summeres i tilsvarende store avs-



Figur 4.2: Ray5 transmisjonstap for hastighetsprofil 3.

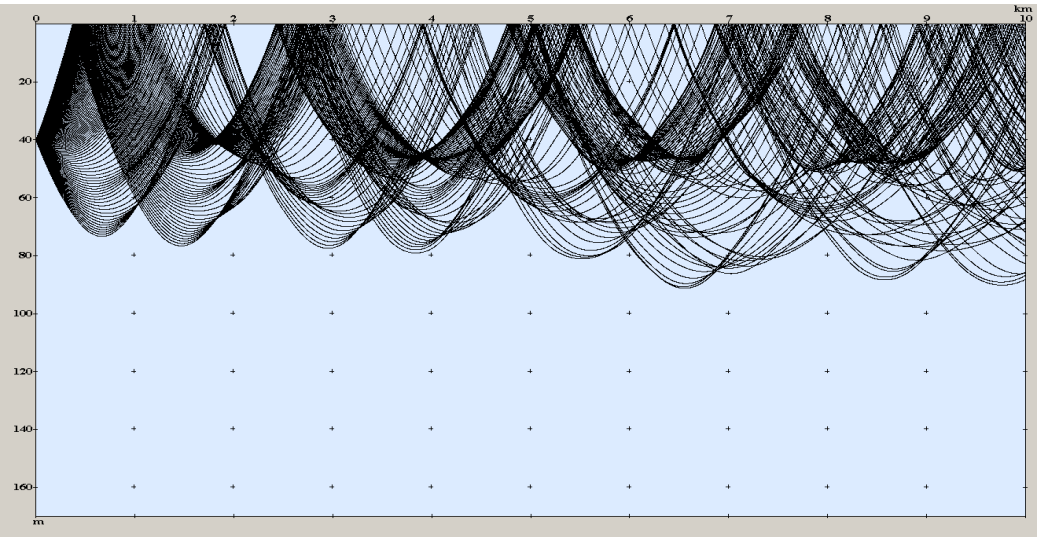
tandsblokker. I vedlegg B.2.10 er funksjonen som beregner transmisjonstapet, *plot_rayrange4.m* vist. Figur 4.2 viser transmisjonstapsplottet beregnet på grunnlag av hastighetsprofil 3.

4.2 Oppsett av Lybin for beregning med hastighetsprofilene fra datasettet

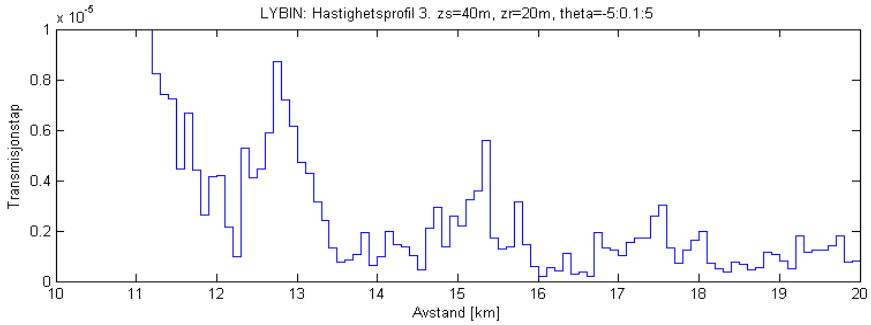
For å få Lybin til å håndtere avstandsavhengige lydhastighetsprofiler må det gjøres på den måten at havområdet deles inn i horisontale soner. Hver slik horisontal sone må tildeles en lydhastighetsprofil som funksjon av dybden. Man står fritt til å velge yttergrensene til hver sone og dermed størrelsen av gyldighetsområdet til hver vertikale lydhastighetsprofil. En kan for eksempel tenke seg nødvendigheten av å ha svært smale soner i områder der lydhastigheten endrer seg fort med avstand, mens sonene kan være brede der det motsatte er tilfelle.

I praksis ble Lybin satt opp til å plukke 25 lodrette vektorer fra en lydhastighetsmatrise fra datasettet. For enkelhets skyld ble disse plukket i lik avstand innenfor det samme horisontale området mellom 10 og 20 km som Ray5 ble satt opp for å beregne strålegang i. Hver slik lodrette lydhastighetsvektor ble så satt til å gjelde i hvert sitt 400 m lange område. I Lybin er det nemlig mulig å definere lydhastighetsprofiler i slike blokker bortover. Koden for hvordan dette ble gjort for hastighetsprofil 3 i datasettet er vist i vedlegg B.1.2 og et plott av strålegangen er vist i figur 4.3.

Lybin beregner transmisjonstapet cellevis for hele beregningsområdet som forklart i 3.1.1. Transmisjonstapet fra Ray5 estimeres som strålekrysninger innenfor avstandsblokker langs en mottakerdybde. Et sammenlignbart plot



Figur 4.3: Lybin strålegangsplot for hastighetsprofil 3.



Figur 4.4: Lybin transmisjonstap for hastighetsprofil 3.

fås fra Lybin ved å hente ut transmisjonstapet fra samme dybde. På samme måte blir dette summert innenfor blokker før plotting. Vedlegg B.1.3 viser Matlab-scriptet som produserer transmisjonstapsplottet fra Lybin, for hastighetsprofil 3. Selve plottet er vist i figur 4.4.

4.3 Oppsummering av tidligere sammenligninger mellom Ray5 og Lybin

Sammenligninger av strålegang og transmisjonstap mellom Lybin og Ray5 ble gjort for alle de målte hastighetsprofilene, og en del konklusjoner kunne deretter trekkes. Det generelle trekket som ble lagt merke til for alle hastighet-

sprofilene fra datasettet, var at Ray5 så ut til å gi mye mer lokale variasjoner for strålene. Dette var i grunnen ikke så overraskende og det ble delvis konkludert med at datasettet kanskje ikke var så glatt som man først hadde trodd. En teori er at Ray5-metoden, som hadde tilgang på langt mer detaljerte versjoner av hastighetprofilene enn Lybin, i større grad fanget opp ujevnheter i disse. Dette kunne dermed sende strålene på enn del mer varierende veier enn hva tilfellet var for Lybin. Et ekstremt tilfelle kan sees i figur 4.1 i en avstand på rundt 19 kilometer og dybde på omtrent 155 meter. Her får en stråle en merkelig brå knekk.

I tillegg ble det gjort noen tanker om blokkinndelingen i Lybin. Selv om hver hastighetsprofil kun var basert på maksimalt 20 fulldybde hastighetsmålinger, ble man i ettertid usikker på om det var rett å bruke kun 25 avstandsblokker i Lybin. For den ene hastighetsprofilen ble det sågar testet med 100 blokker, uten at dette ga noen særlig endring i de resulterende plottene.

Det viste seg at det var tydelige likhetstrekk i transmisjonstapsplottene fra de to programmene for et par av hastighetsprofilene, deriblant for profil nummer 3 som man kan se av figur 4.2 og 4.4. For andre igjen var det nesten vanskelig å se likheter i det hele tatt. For den ene hastighetsprofilen ble det i tillegg produsert transmisjonstapsplott på grunnlag av Ray5 med 500 stråler og med 100 avstandsblokker i Lybin. Disse plottene viste faktisk en del likhetstrekk. Det ble dermed nærliggende å tenke at kanskje 100 stråler i Ray5 var for få for å få et godt estimat av transmisjonstapet, samtidig som 25 avstandsblokker i Lybin kanskje var en for grov inndeling. Som for strålegangsberegningene ble det også antatt at lokale variasjoner i hastighetsprofilene slo sterkere ut i Ray5. Et annet spørsmål som kan reises er om metoden for å beregne transmisjonstapet fra Ray5 kan være for enkel. Alle strålekryssinger med mottakerdybden ble vektet likt, selv om strålene hadde forskjellig reiselengde. På grunn av geometrisk spredning vil en strålekrysning fra en stråle med lang reiselengde tilsvare et mindre energibidrag enn en krysning fra en stråle med kortere reiselengde.

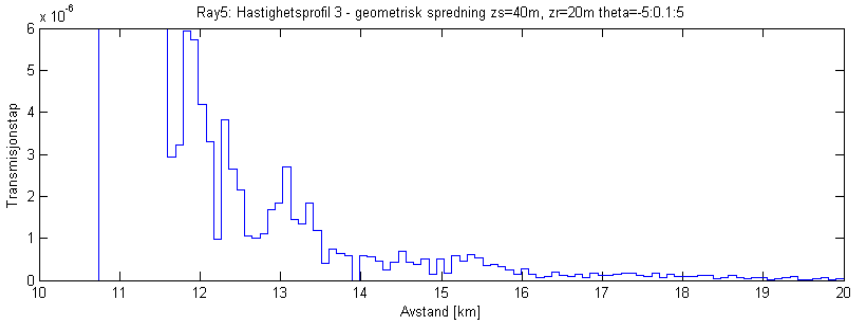
Kapittel 5

Forbedringer og nye resultater

For å fortsette tråden fra forrige kapittel kan vi si at noen viktige erfaringer hadde blitt gjort etter sammenligningene mellom Ray5 og Lybin. Disse ga motivasjon til en del forbedringer og videre testing. I dette kapitlet vil det nye som er gjort bli gjennomgått. Først skal et forsøk på en forbedring av metoden for transmisjonstapsberegninger fra Ray5 gjennomgås og testresultater for dette skal vises. Transmisjonstapsberegninger fra Lybin og Ray5 har også blitt testet opp mot referansemodellen Oases for en avstansuavhengig hastighetsprofil. Hvordan dette er gjort og resultater fra testen vil presenter som neste punkt. Tredje delkapittel inneholder en gjennomgang av en metode for å glatte hastighetsprofilene og resultatene dette gir for Ray5 og Lybin. Videre skal en undersøkelse av effekten av antall avstandsblokker i Lybin gjennomgås. Til sist vil det bli presentert tresultater for en gjennomsnittlig hastighetsprofil basert. Siden hastighetsprofil 3 fra datasettet viste så tydelig disse rare knekkene i strålegangen, er det denne som er brukt videre i alle undersøkelsene.

5.1 Geometrisk spredning i Ray5

Én mulighet for å bygge inn tr.tap i Ray5 er å gjøre omtrent som Baxter og Orr gjorde i [8] og finne de to nabostrålene fra kilden som passerer nærmest et punkt på mottakerdybden. Ved å sende ut stråler i faste vinkelintervall fra kilden og anta at utstrålt energi holder seg konstant mellom to stråler fra kilden, kan man finne et estimat for transmisjonstapet ved å se på avstanden mellom de to nabostrålene når de krysser mottakerdypet. En slik metode er under uttesting av Trond Jenserud som har laget Ray5, og vil bli en mulighet i programmet etterhvert. Problemet er at metoden skyter lydstråler flere ganger for å treffe et krysningspunkt med mottakerdybden



Figur 5.1: Ray5 transmisjonstap for hastighetsprofil 3 med geometrisk spredning.

(en egenstråle) for den ønskede avstanden. Dette tar utrolig mye tid slik som Ray5 er satt opp for beregning med datasettet. Særlig når det skal gjøres for for eksempel 100 punkter langs mottakerdybden for å skape et transmisjonstapsplott for denne dybden. Av denne grunnen ble det valgt å prøve med en annen teknikk som gikk ut på å fikse litt på Marie Darrieus metode som er beskrevet i 4.1. Ray5 lagrer nemlig også total ganglengde, s for hver stråle for hvert beregningssteg. Dette gjør at det er mulig å vekte hvert bidrag i transmisjonstapsberegninga med $\frac{1}{s}$. Det er kjent at som følge av geometrisk spredning vil lydtrykket dempes slik at det avhenger inverst av avstanden. Håpet er at en slik vekting skal gjøre transmisjonstapsberegningene fra Ray5 mer sammenlignbare med motsvarende beregninger fra Lybin.

I vedlegg B.2.11 er det vist hvordan funksjonen *plot_rayrange5.m* er bygd opp. Dette er i all hovedsak den samme funksjonen som *plot_rayrange4.m*, med det unntaket at hver eneste strålekrysning er vektet med $\frac{1}{s}$. Et resulterende plot er vist i figur 5.1 for datasett 3. Dette plottet motsvarer figur 4.2 som er laget ved hjelp den gamle metoden i funksjonen *plot_rayrange4.m*. Resultatet var ikke akkurat veldig lovende for den nye metoden. Det nye plottet har en god del mindre likhetstrekk med transmisjonstapet fra Lybin i figur 4.4. Metoden ble likevel testet videre for en enkel avstandsuavhengig hastighetsprofil. Mer om dette kommer i påfølgende avsnitt.

5.2 Sammenligning med Oases

Siden de første sammenligingene mellom Lybin og Ray5 ikke ga helt sikre svar rundt gyldigheten av transmisjonstapet som ble beregnet, ble det besluttet å sammenligne med en annen, kvalitetssikret modell. Modellen som ble

valgt var Oases. For en avstandsuavhengig hastighetsprofil skulle altså transmisjonstapet fra Lybin, Ray5 og Oases sammenlignes.

5.2.1 Den avstandsuavhengige hastighetsprofilen

Profilen som skulle brukes var en såkalt n^2 = lineær hastighetsprofil. Dette betyr at brytningsindeksen^2 varierer lineært med dybden. Uttrykket for lydhastigheten blir som følger:

$$c(z) = \frac{c_0}{(\sqrt{(1+bz)})} \tag{5.1}$$

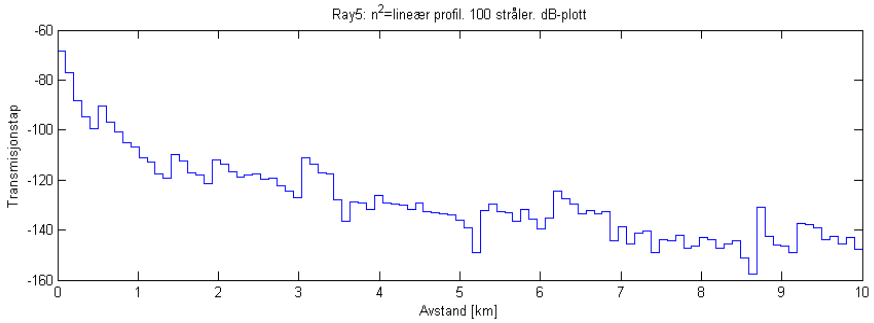
der $c_0 = c(z = 0)$ er lydhastigheten ved toppen og b er en konstant som bestemmes av c_0 og lydhastigheten ved bunnen, $c(z = D)$.

5.2.2 Oppsett for sammenligningene

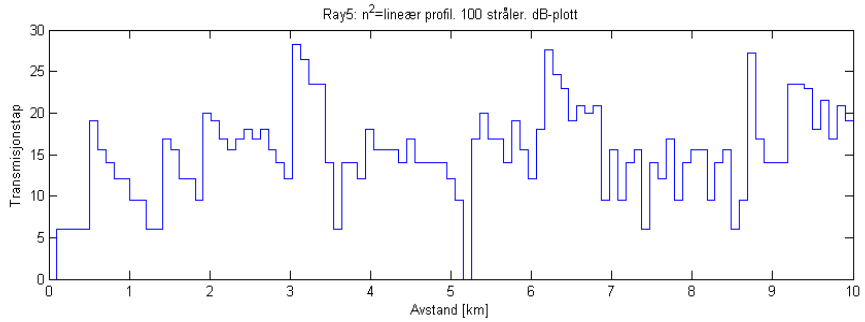
Total avstand og dybde transmisjontapet skulle beregnes i var henholdsvis 10 km og 200 m og kildefrekvensensen skulle være 100 Hz. c_0 ble valgt til 1500 m/s mens $c(z = D)$ ble satt til 1460 m/s. Kildedybden ble i dette tilfellet satt til det samme som mottakerdybden på 40 m. Hvordan hastighetsprofilen ble realisert for Ray5 i *ssp.m* kan sees i vedlegg B.2.15. Transmisjonstapsplott for Ray5 på grunnlag av 100 stråler er vist i figur 5.2 for den nye metoden (forklart i avsnitt 5.1) og i figur 5.3 for den gamle metoden (som forklart i avsnitt 4.1). I tillegg er det laget et plott på grunnlag av 1000 stråler i Ray5. Dette er vist i figur 5.4 for ny type transmisjonstapsberegning og i figur 5.5 for den gamle typen. I vedlegg B.1.4 er det vist hvordan hastighetsprofilen ble implementert i Matlab-scriptet for Lybin. Det resulterende transmisjonstapsplottet er gitt i figur 5.6. Alle plottene av transmisjonstap fra Lybin og Ray5 måtte nå gjøres i dB for å være sammenlignbare med Oases. For å få et transmisjonstapsplott ut fra Oases måtte det gjøres noen valg. Etter tips fra dem som hadde brukt Oases GUIen mye ble det bestemt at det ikke burde brukes mer enn 15 lag i modellen. Alle disse 15 lagene var realisert som vannlag med lik utstrekning i dybden, og hastigheten i hvert lag var satt lik hastigheten fra n^2-profilen ved dybden midt laget. Håpet var at dette skulle være et godt estimat av ett lag med en lineær n^2-profil. I figur 5.7 er det resulterende transmisjonstapsplottet for Oases vist.

5.2.3 Betraktninger rundt sammenligningene

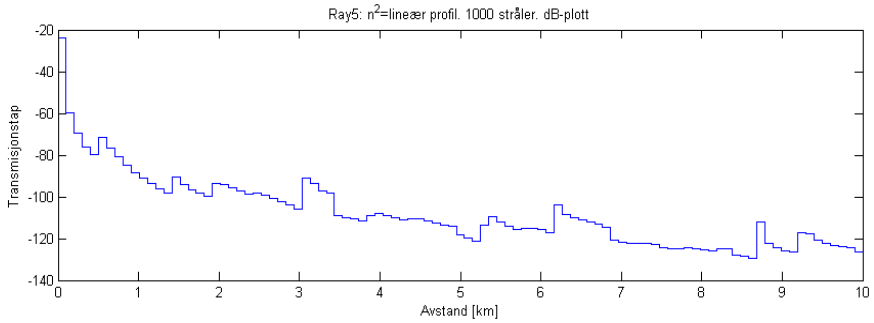
Etter å ha studert figurene litt er det noen trekk som kommer fram. Den nye metoden for å beregne transmisjonstap fra Ray5 ser ut til å gi plott som



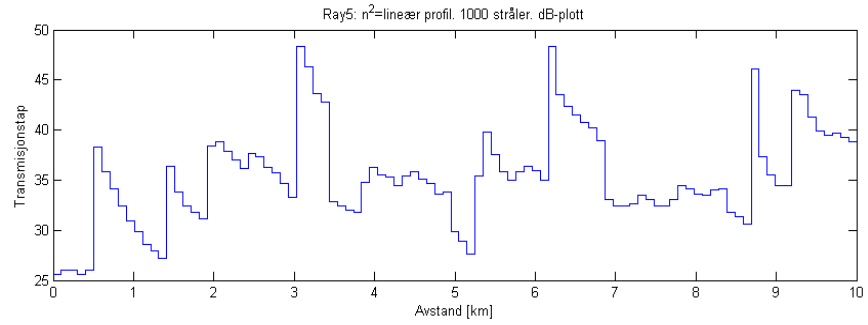
Figur 5.2: Ray5 transmisjonstap for n^2-profil på grunnlag av 100 stråler.



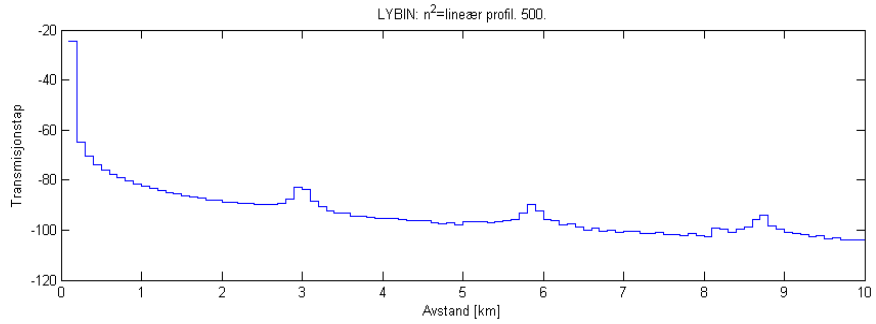
Figur 5.3: Ray5 transmisjonstap for n^2-profil på grunnlag av 100 stråler (gammel metode).



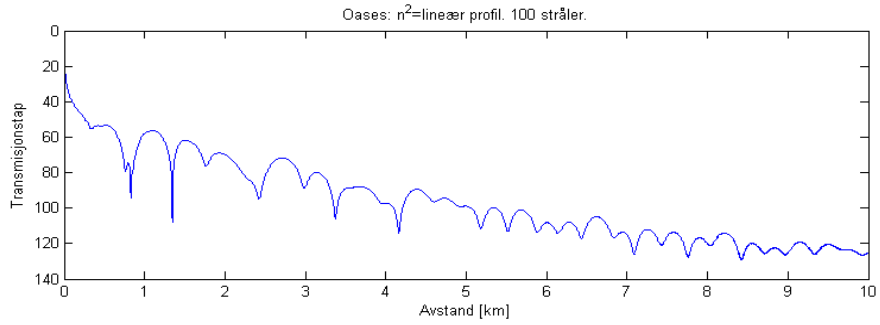
Figur 5.4: Ray5 transmisjonstap for n^2-profil på grunnlag av 1000 stråler.



Figur 5.5: Ray5 transmisjonstap for n^2-profil på grunnlag av 1000 stråler (gammel metode).



Figur 5.6: Lybin transmisjonstap for n^2-profil.



Figur 5.7: Oases transmisjonstap for n^2-profil.

er tildels sammenlignbare med transmisjonstapsplottet fra Oases, når det er brukt 1000 stråler. Transmisjonstapsplottet fra Lybin er også sammenlignbart med Oases. Dette kom ikke som en vedlig stor overraskelse, siden en gradvis avtagende kurve med avstanden er forventet når man beregner transmisjonstapet. Ingen av plottene ser ut til å falle like mye i verdi, men tendensene mellom plottene er like og kanskje så like som man kan forvente når man sammenligner mellom tre så forskjellige modeller. Transmisjonstapsplotting fra Ray5 på den gamle måten fungerer helt klart ikke i denne testen. Strålekrysninger langt unna kilde blir tillagt altfor høy vekt, og en dB-plotting blir meningsløs.

5.3 Glatting av lydhastighetsprofil

Ett av problemene som ble påpekt ved tidligere simuleringer med datasettene var muligheten for at hastighetsprofilene ikke var så glatte som først antatt. De er som nevnt basert på et visst antall målinger i havområdets fulle dybde over en avstand på ca 30 km. Resten av hastighetsprofilene er fylt ut ved hjelp av interpolering. For å få bukt med eventuelle ujevnheter som kunne oppstått i både måle- og interpoleringsprosessen var det naturlig å tenke på å finne en metode for å glatte datasettet. Den første metoden som ble foreslått var å finne en polynom- eller spline-metode som kunne brukes til å beskrive hastighetsprofilen fra datasettet. Dette ble etterhvert forlatt på grunn av problemer med å finne gode metoder i Matlab for å implementere dette for funksjoner av to variabler. Det var i tillegg ikke lisens å oppdrive ved NTNU for spline-verktøykassen i Matlab. Dette var litt synd, for det virker å være en interessant løsning å se på videre. Trolig vil den kunne gi meget kjapp kjøring av Ray5 siden man slipper å kontinuerlig interpolere i lydhastighetsmatrisen.

5.3.1 Kagawas metode

Metoden som til slutt ble valgt til glatting ble tatt fra en artikkel av Kagawa et al. [16], der den ble brukt til å glatte lydtrykksverdier i bruk av TLM-metoden. Prinsippet er ganske enkelt og fungerer som et todimensjonalt gaussisk filter. En algoritme går gjennom hele den aktuelle lydhastighetsmatrisen og erstatter hver enkelt verdi med en ny gjennomsnittsverdi. Denne nye gjennomsnittsverdien er basert på den opprinnelige verdien i punktet og de åtte naboverdiene. Verdien i midtpunktet blir vektet med 1, de to vannrette og de to loddrette naboverdiene blir vektet med $e^{-\mu}$, mens de fire diagonale naboverdiene blir vektet med $e^{-2\mu}$. Hvis vi tenker oss det oprinnelige

punktet $C_{i,j}$ i en lydhastighetsmatrise, der i tilsvarer avstandskoordinaten og j tilsvarer dybdekoordinaten, så kan den nye gjennomsnittsverdien $\bar{C}_{i,j}$ beregnes ved hjelp av

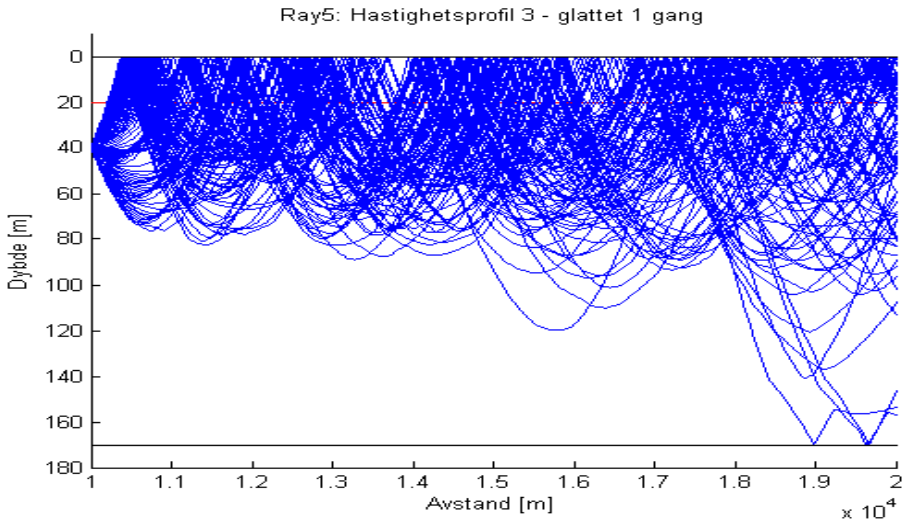
$$\begin{aligned}\bar{C} =&\{C_{i,j} + e^{-\mu}(C_{i-1,j} + C_{i+1,j} + C_{i,j-1} + C_{i,j+1}) + e^{-2\mu}(C_{i-1,j-1} \\ &+ C_{i+1,j-1} + C_{i-1,j+1} + C_{i+1,j+1})\}/(1 + 4e^{-\mu} + 4e^{-2\mu}).\end{aligned} \tag{5.2}$$

Dette er en ganske enkel metode å implementere i Matlab, og vedlegg B.2.12 viser hvordan det er gjort i en funksjon som tar inn en lydhastighetsmatrise, fyller de tomme feltene og glatter den et gitt antall ganger. Det er selvfølgelig en litt annen framgangsmåte for verdiene som ligger langs kantene av matrisen og dermed ikke har åtte naboverdier.

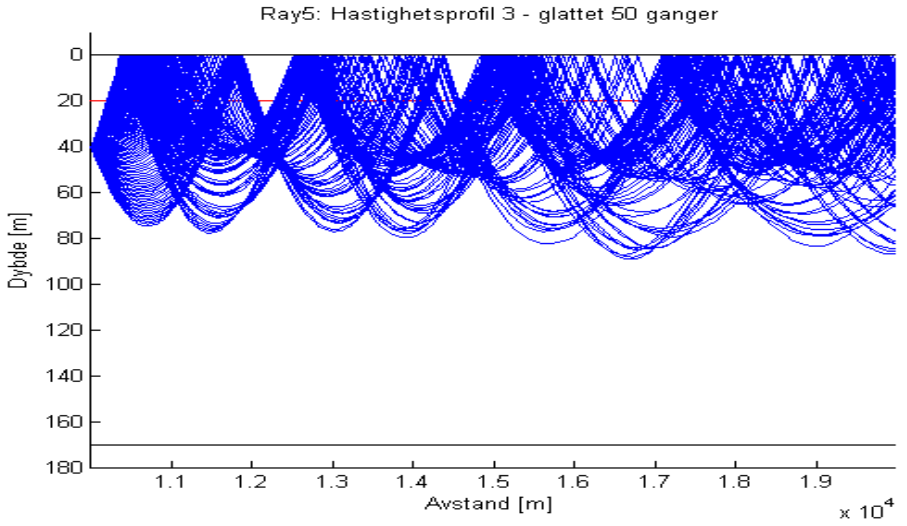
5.3.2 Resultater med Kagawas metode

I simuleringene som ble gjort var alltid $\mu = 1$. Som en første test ble hastighetsprofil 3 fra datasettet glattet én gang, og Ray5 kjørt på grunnlag av denne glattede profilen. De andre forutsetningene for beregningen var akkurat som forklart i 4.1. Funksjonen *Ray5.m* med mulighet for glatting er vist i vedlegg B.2.13. Resulterende strålegangsplott er vist i figur 5.8. Dette plottet viste seg å nesten ikke være forskjellig fra strålegangsplottet som var basert på den uglattede profilen. For å virkelig teste om glattemetoden gjorde noe som helst utslag ble deretter det samme forsøkt for hastighetsprofilen glattet 50 ganger. I figur 5.9 kan resultatet betraktes. Det kom som en gledelig overraskelse at dette nye plottet var veldig likt strålegangsplottet fra Lybin i figur 4.3. Etter dette resultatet ble det nødvendig å se på hvordan strålegangsplottet fra Lybin ble seende ut for hastighetsprofil 3 glattet 50 ganger. I vedlegg B.1.5 er det vist hvordan dette ble gjort og plottet kan kan sees i figur 5.10. Det er en forskjell mellom dette plottet og det i figur 4.3, men den er ikke veldig stor. Det nyeste plottet ser mer ordnet ut og strålene virker å løpe i mer samlede bunter og i lik innbyrdes avstand. Det gir altså litt effekt å glatte, men ikke like mye som i Ray5. Dette kan forklares med at Lybin for det første ser et slags glattet datasett på grunn av blokkdelingen. For det andre er det slik at Lybin foretar en glatting av de loddrette hastighetsprofilene over sju bølgelengder av kildefrekvensen. For en kildefrekvens på 1000 Hz og lydhastighet på ca 1500 m/s tilsvarer sju bølgelengder av kildefrekvensen $7\frac{1500m/s}{1000Hz} = 10.5m$. Kanskje er det slik at disse to glattende bidragene i Lybin er grunnen til at sammenligningen med Ray5 med 50 glattinger av hastighetsprofilen fra datasettet blir så god.

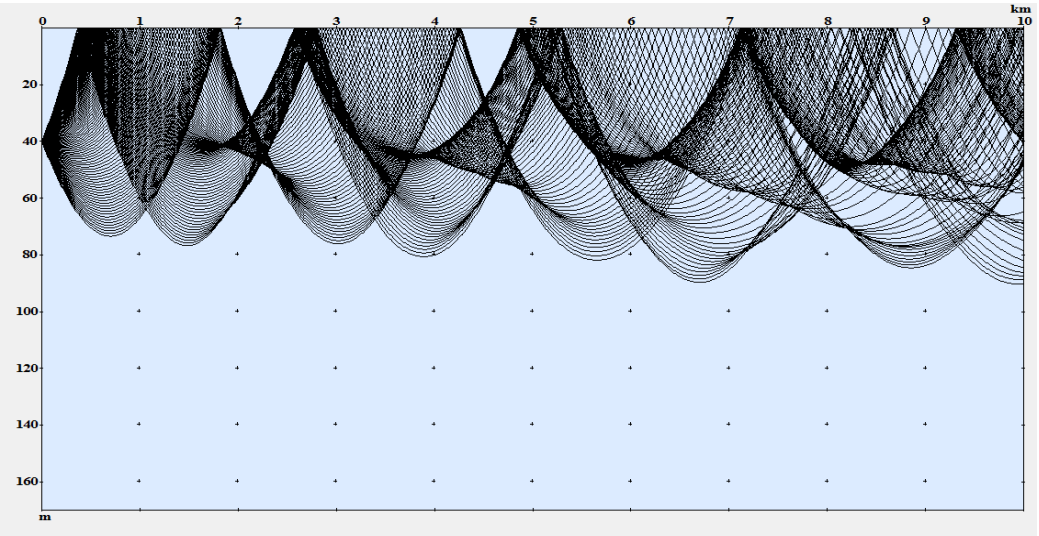
Transmisjonstapsplottene for 50 glattinger i Ray5 og Lybin måtte også



Figur 5.8: Ray5 strålegangsplot for hastighetsprofil 3 glattet 1 gang.



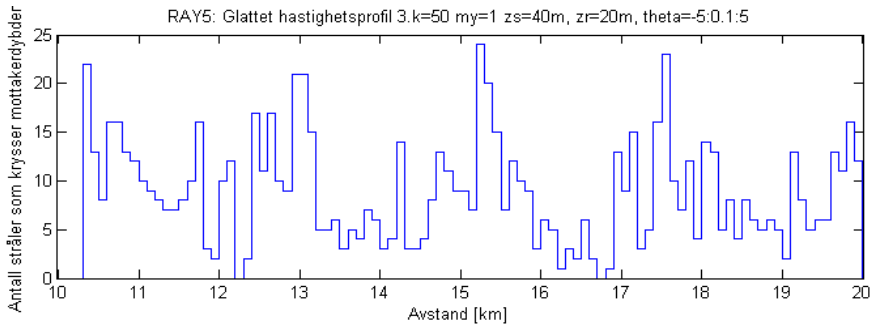
Figur 5.9: Ray5 strålegangsplot for hastighetsprofil 3 glattet 50 ganger.



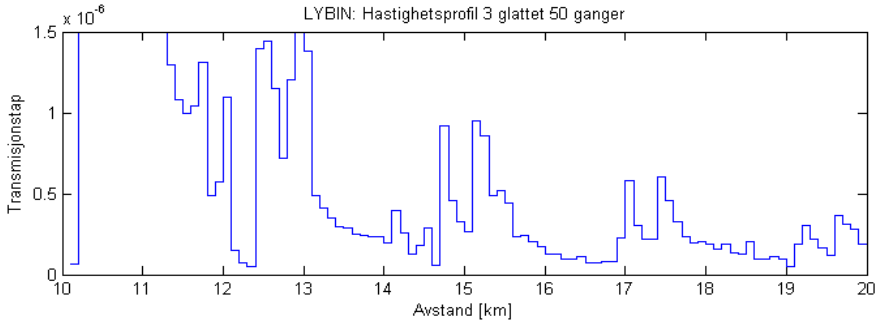
Figur 5.10: Lybin strålegangsplot for hastighetsprofil 3 glattet 50 ganger.

sjekkes. Disse er vist i Figur 5.11 og 5.12. Den gamle metoden for beregning av transmisjonstap er brukt, siden den viste seg å gi bedre sammenligninger mellom plott fra Ray5 når det ikke skulle plottes i dB. Historien gjentok seg her. En kan se at plottet fra glattet profil i Ray var mer likt transmisjonstapsplottet fra Lybin for uglattet profil (figur 4.4). Den største likheten sees likevel mellom de to nye figurene som begge er basert på et glattet datasett. Det har vist seg å være vanskelig å få sammenlignbare resultater fra Lybin og Ray5 når beregninger skal gjøres på grunnlag av hastighetstprofilene fra datasettet. Etter resultatene med glattingen kan det se ut som en metode for å gi de to programmene like forutsetninger er funnet.

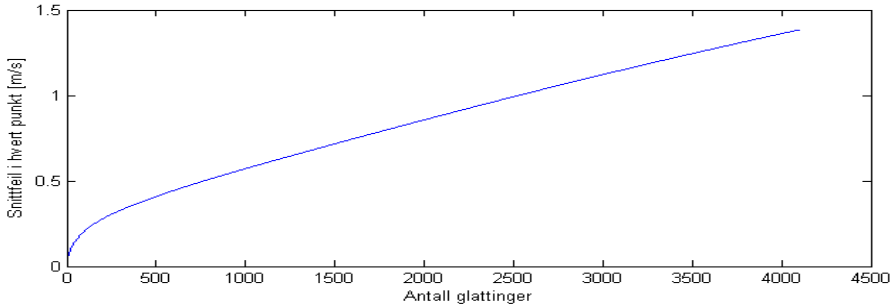
Noen videre undersøkelser ble også gjort. Det reiste seg et spørsmål om glattemetoden ville konvergere etterhvert. Altså om man ville komme fram til en endelig hastighetsmatrise etter en del glattinger. For å teste dette ble glattemetoden kjørt flere ganger for hastighetsprofil 3. Samtidig ble det beregnet gjennomsnittlig endring i hastighetene i de glattede hastighetsprofilene i forhold den opprinnelige for hver kjøring. På denne måten kunne endringen i hastighetsprofilen som funksjon av antall glattinger plottes i figur 5.13 og i et mindre utsnitt i figur 5.14. Fra den første fiuguren ser man tydelig at glattingen ikke kommer til å konvergere. Fra det mindre utsnittet i figur 5.14 kan det leses ut at gjennomsnittsfeilen i hvert punkt i hastighetsmatrisen glattet 50 ganger er ca 0.15 m/s. Ikke veldig mye altså. Scriptet som lager figurene er vist i vedlegg B.2.14. For ytterligere undersøkelse er stråle-



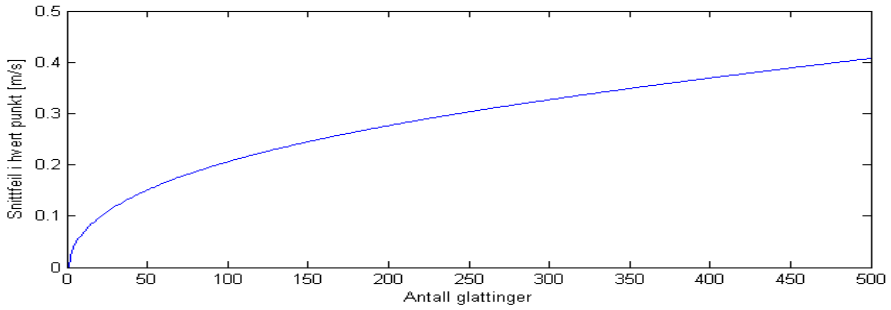
Figur 5.11: Ray5 transmisjonstapsplott for hastighetsprofil 3 glattet 50 ganger.



Figur 5.12: Lybin transmisjonstapsplott for hastighetsprofil 3 glattet 50 ganger.



Figur 5.13: Konvergensplott for glatting av hastighetsprofil 3.



Figur 5.14: Konvergensplott for glatting av hastighetsprofil 3 - mindre utsnitt.

gangsplott fra Ray5 for økende antall glattinger er vist vedlegg A.1 Disse figurene har også hastighetsprofilen plottet i bakgrunnen slik at man kan se hvordan den utvikler seg. Det kan virke som om 50 glattinger er tilstrekkelig få slik at hastighetsprofilen ikke har rukket å bli særlig utsmørt.

5.4 Variabel blokkoppdeling i Lybin

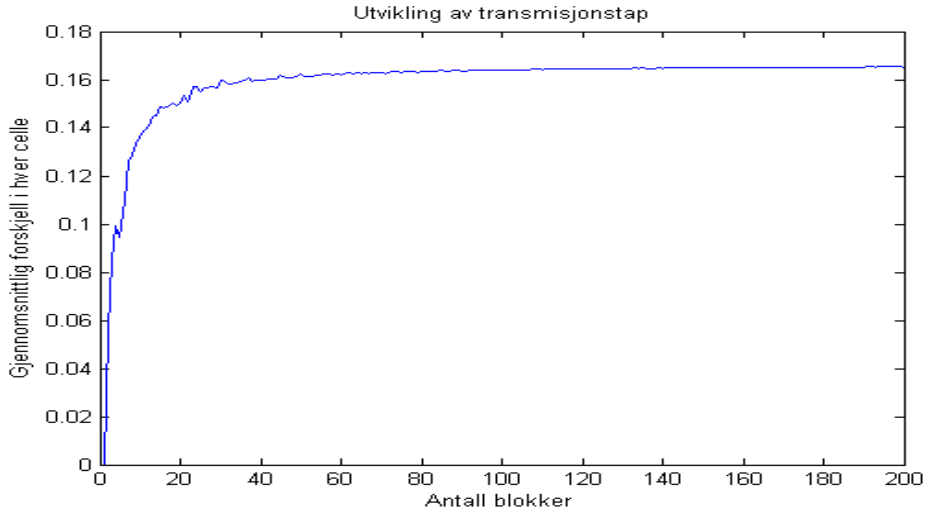
I det tidligere arbeidet med lydhastighetsdatasettet fra Nordsjøen ble det i all hovedsak brukt 25 like store horisontale blokker til å beskrive lydhastigheten i havet over en avstand på 10 km. Denne inndelingen, som høres grov ut, ble gjort på grunnlag av den egentlige oppløsningen i målingene av lydhastigheten. Som nevnt i delkapittel 3.4 var den reelle oppløsningen på ca 20 målinger i havets fulle dybde over 30 km. Selv om den horisontale blokkinndelingen var nesten fire ganger finere enn målegrunnlaget, er det godt mulig at den ikke var fin nok for å gi en god sammenligning med resultatene fra Ray5. En glatting av datasettet ville delvis kunne avhjelpe dette problemet. Det er også veldig interessant å se grundigere på hvordan inndelingen i blokker påvirker resultatet fra Lybin. For å sjekke dette ble det gjort

en konvergenstest for innvirkningen av blokkindeling på trasnmisjonstapet for hastighetsprofil 3 fra datasettet. Både en glattet og en uglattet versjon av datasettet ble testet. Dette ble gjort ved å kjøre Lybin med et økende antall blokker og etterhvert sjekke hvordan den gjennomsnittlige forskjellen i transmisjonstap utviklet seg. I detalj ble framgangsmåten slik i Matlab:

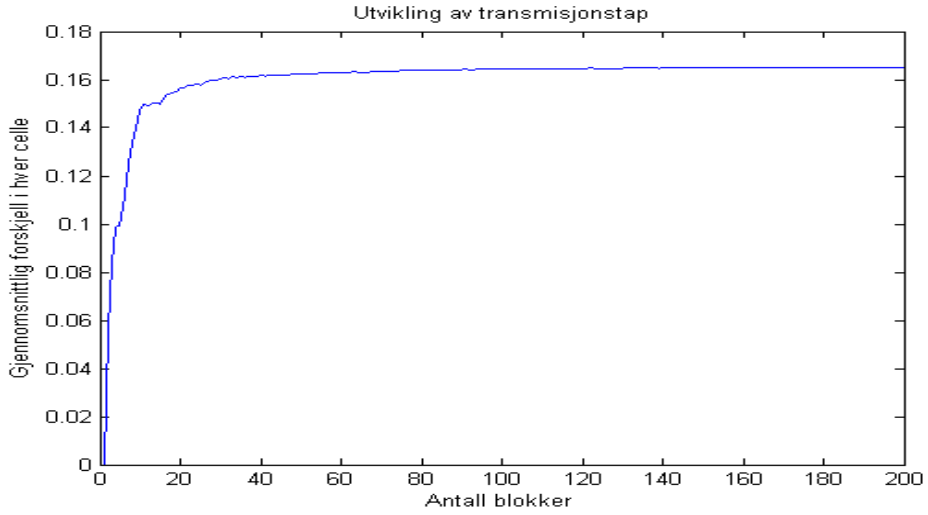
1. Bestemme seg for et antall ganger man ønsker å kjøre Lybin.
2. Sette antall avstandsblokker hastighetsprofilen er delt inn i til én.
3. Beregne transmisjonstapsmatrise i Lybin med gitt blokkinndeling.
4. Beregne den elementvise differansematrisen i absoluttverdier mellom sist utregnede transmisjonstapsmatrise og transmisjonstapsmatrisen for én blokk.
5. Lagre gjennomsnittsverdien til elementene i resultatmatrisen fra steg 4 i en differansevektor.
6. OM antall ganger Lybin skal kjøres *ikke er* nådd: Øk antall blokker hastighetsprofilen er delt inn i med én og gå til steg 3
   ELLER om antall ganger Lybin skal kjøres *er* nådd: Avslutt beregningen

Differansevektorens k-te element vil her være et tall på hvor stor gjennomsnittlig forskjell det er mellom transmisjonstapet beregnet på grunnlag av en inndeling av hastighetsprofilen i én blokk, og en inndeling i k blokker. I figur 5.15 er differansevektoren plottet for inntil 200 blokker. Det samme er plottet i figur 5.16, men her er hastighetsprofilen glattet 50 ganger med Kagawas metode først. Matlab-scriptet som er brukt for å produsere disse plottene er vist i vedlegg B.1.6.

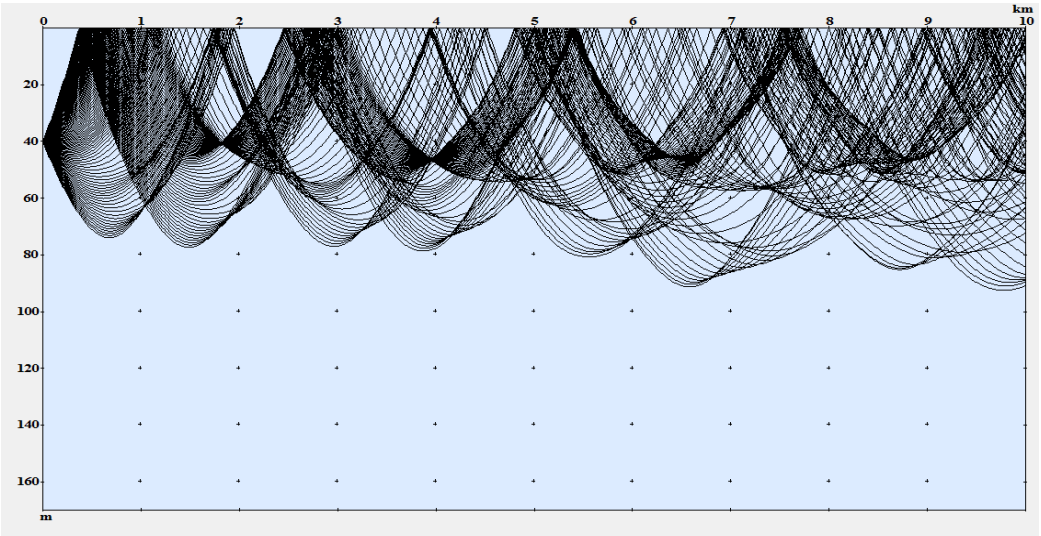
I figur 5.15 og 5.16 er verdien på den vertikale aksen forholdsverdier. De viser altså den relative endringen i det gjennomsnittlige transmisjonstapet, i forhold til om transmisjonstapet var beregnet på grunnlag av hastighetsprofilen inndelt i én blokk. Begge plottene viser konvergens mot en verdi på ca 0.165, altså ca 16,5% gjennomsnittlig forskjell fra om transmisjonstapet var beregnet med én blokk. Ved en inndeling i 25 blokker er verdien for den gjennomsnittlige endringen 0,1549 for det uglattede datasettet og 0,1578 for for det glattede datasettet. Transmisjonstapet til det glattede datasettet ser også ut til konvergere litt fortere enn transmisjonstapet til det uglattede for økende antall blokker. Dette er ikke så overraskende siden punktprøving, eller snarere kolonneprøving, fra en glattet versjon av en matrise bør gi mindre



Figur 5.15: Konvergensplott for hastighetsprofil 3 for økende antall avstandsblokker.



Figur 5.16: Konvergensplott for hastighetsprofil 3 glattet 50 ganger med Kagawas metoder for økende antall avstandsblokker.



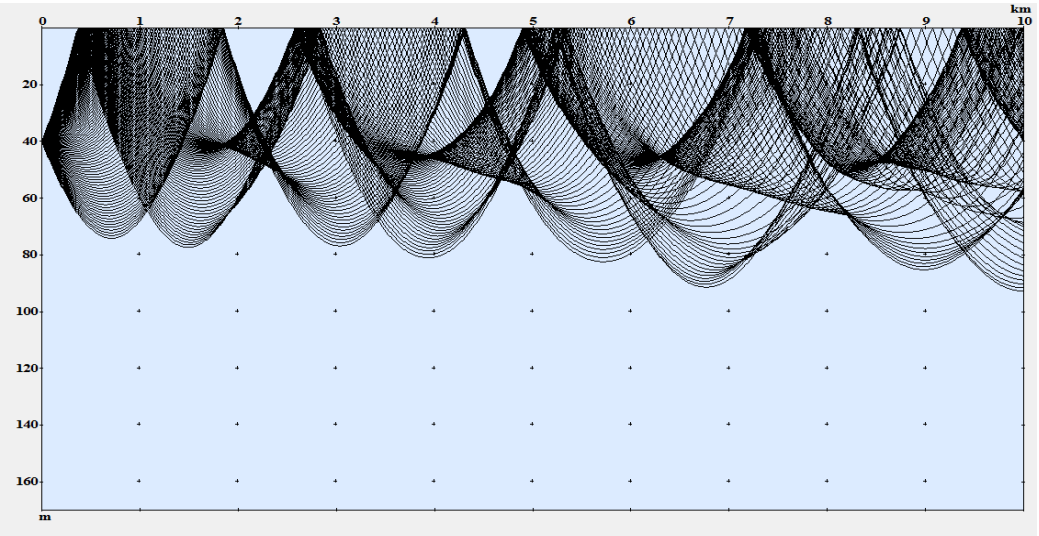
Figur 5.17: Lybin strålegangsplott for hastighetsprofil 3 delt inn i 200 blokker.

variasjon mellom kolonnene.

Det ser ut som transmisjonstapet er rimelig nært konvergens for en inndeling av hastighetsprofilen i 25 avstandsblokker. Kanskje er en slik inndeling fin nok for hastighetsprofilene i datasettet likevel. Dette var jo det som først ble antatt. For å prøve å komme nærmere et svar på dette ble strålegangen til profil 3, delt inn i 200 avstandsblokker og plottet i Lybin for den uglattede og den glattede hastighetsprofil 3. I Figur 5.17 og 5.18 er disse vist. Det er vanskelig å se noen forskjeller for både den uglattede og glattede versjonen av hastighetsprofilen når de sammenlignes mot deres søsterstrålegangsplott i figur 4.3 og 5.10, som er basert på en en inndeling i 25 blokker. Det kan derfor trolig konkluderes med at for dette datasettet var det faktisk en inndeling i 25 blokker nok for å få gode strålegangsplot. Konvergensundersøkelsene underbygger også dette.

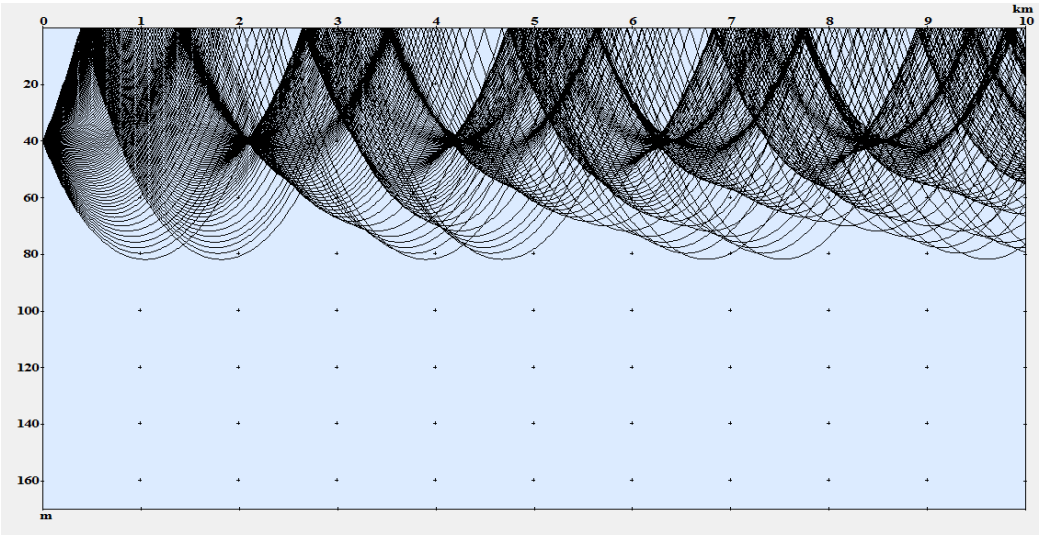
5.5 Gjennomsnittlig hastighetsprofil

En del av oppgaven gikk også ut på å sjekke hvilken effekt det har å inkludere avstandsavhengighet i form av en gjennomsnittlig hastighetsprofil. Spesielt skulle det undersøkes hvordan dette påvirket transmisjonstapet. Dette er en ganske interessant problemstilling. Om man får omtrent tilsvarende resutalteter for en slik gjennomsnittlig profil som for den reelle hastighetsprofilen så kan i teorien masse tilpasningsarbeid spares.

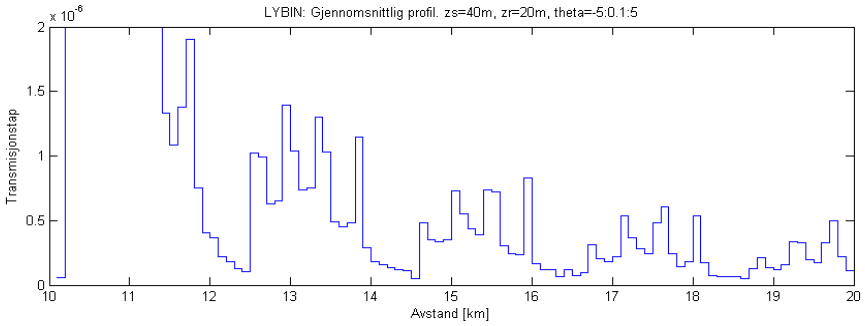


Figur 5.18: Lybin strålegangsplott for hastighetsprofil 3 glattet 50 ganger og delt inn i 200 blokker.

Problemstillingen ble løst ved å ta utgangspunkt i hastighetsprofil 3 slik som den er satt opp for Lybin i 4.2. Denne hastighetsprofilen ble så gjort om til en gjennomsnittsprofil ved å ta gjennomsnittet av hver enkelt rad i profilen. Disse gjennomsnittsverdiene ble brukt som en loddrett hastighetsprofil som ble satt gjeldende i hele beregningsområdet. Matlab-scriptet som viser hvordan dette er løst er vedlagt i B.1.7. I figur 5.19 er strålegangsplottet for den gjennomsnittlige profilen vist, mens transmisjonstapsplottet fins i figur 5.20. For begge disse plottene er det likhetstrekk med tilsvarende plott for både uglattet og glattet versjon av hastighetsprofil tre. For transmisjonstapet er det sågar veldig klare likheter, men forskjellene er viktige. Strukturer som kjennes igjen virker som de er forskjøvet stedvis litt til høyre og venstre i figuren. Dette betyr at hastighetsprofilen tross alt er en del endret og at dette får godt merkbare innvirkninger på beregningen av transmisjonstapet.



Figur 5.19: Lybin strålegangsplot for gjennomsnittlig profil.



Figur 5.20: Lybin transmisjonstap for gjennomsnittlig profil.

Kapittel 6

Konklusjoner og avslutning

6.1 Konklusjoner fra litteraturstudien

Avstandsavhengig oseanografi og dens påvirkning på lydutbredelse er noe som har vært studert lenge. Veldig ofte er det fenomenene interne bølger og solitoner det har blitt sett på. Forskjellige angrepsvinkler for å vise hvordan de påvirker lydutbredelsen er prøvd i forskjellige studier. Noen baserer seg på strålegangsmodeller for å forklare hvordan interaksjon med interne bølger og solitoner fører til tidsvarierende lydfelt. Andre igjen ser på modal kobling mellom kildefrekvensen og bølgelengden til de interne bølgene for å forklare sterke variasjoner transmisjonstapet for enkelte frekvenser.

Effekten av avstandsavhengig oseanografi på lydutbredelse er uansett noe det har blitt forsket mye på og skrevet utallige artikler om. Dette beviser at det i alle fall er noe man må vurdere om man skal ta hensyn til når man skal beregne lydutbredelse under vann.

6.2 Oppsummering av nye resulater

Forsøket med å prøve å inkludere geometrisk spredning i transmisjonstapsberegningene fra Ray5 så i første omgang ikke ut som noen stor suksess. Sammenlignbarheten med transmisjonstapet fra Lybin ble heller en del dårligere.

I sammenligning med referansemodellen Oases for en avstandsuavhengig hastighetsprofil klarte derimot den nye beregningsmetoden for transmisjonsjonstapet fra Ray5 seg ganske godt. Transmisjonstapet Lybin viste også likheter med med resultatet fra referansemodellen. Den gamle metoden for transmisjonstapsberegning fra Ray5 produsert resultater som bare lignet på seg selv. Selv

om det var likheter mellom transmisjonstapsplottene fra Lybin og Ray5og plottet fra referansemodellen, hersker det litt usikkerhet rundt om referansemodellen virkelig hadde nok lag i dybden og om den var satt opp helt optimalt for å gi en god sammenligning med Lybin. Det var nemlig ikke overbevisende likheter som ble observert.

Kagawas glattemetode så ut til å være medisinen den avstandsavhengige hastighetsprofilen trengte for at Ray5 og Lybin skulle kunne gi ut rimelig like resultater. Ved 50 glattinger av hastighetsprofilen med Kagawas metode ble resultatene man hadde håpet på oppådd. Strålegangsplottene fra Ray5 og Lybin ble veldig like, og transmisjonstapsplottene ble også ganske like. Trolig har hastighetsprofilene i datasettet, som mistenkt, ikke vært helt glatte. Videre undersøkelser viste at glattingen ikke ville konvergere om den ble kjørt lenge. 50 glattinger så ut til å være et godt kompromiss mellom å få glattet dataene tilstrekkelig og å få en veldig utsmørt hastighetsprofil.

For blokkdelingen sin del viste det seg at det ikke er nødvendig med en inndeling i veldig mange avstandsblokker i Lybin for å få få et godt resultat ut fra hastighetsprofilene fra datasettet. Resulterende transmisjonstap fra Lybin konvergerte etterhvert for økende blokkindeling. Mye tydet på at 25 avstandsblokker som var det opprinnelig valget var nok for å få tilfredsstillende resultater.

Til sist ble det gjort en rask test for å se om en gjennomsnittlig hastighetsprofil vil gi gode resultater i Lybin. Siden hastighetsprofilene i datasettet ikke har voldsomme horisontale variasjoner, hadde resultatene likhetstrekk med resultatene for de avstandsavhengige profilene. Likevel var det en del forskjeller. Hovedtrekkene i både transmisjonstapsplott og strålegangsplott så ut til å være forskjøvet litt sideveis.

6.3 Takk

Jeg ønsker å takke min hovedveileder, professor Ulf Kristiansen for all hjelp med oppgaven. Også Trond Jenserud og Karl Thomas Hjelmervik ved FFI fortjener takk for god veiledning og hjelp med Ray5 og Lybin. I tillegg er jeg glad for at post doc. Shefeng Yan ved NTNU viste meg hvordan Oases og Oases-GUIen skulle brukes.

Bibliografi

[1] Owen S. Lee. Observations on internal waves in shallow water. *Limnology and Oceanography*, 6(3):312–321, 1961.

[2] Owen S. Lee. Effect of an internal waves on sound in the ocean. *The Journal of the Acoustical Society of America*, 33(5):677–681, 1961.

[3] Ira Dyer. Statistiscs of sound propagation in the ocean. *The Journal of the Acoustical Society of America*, 48(1):337–345, 1970.

[4] Harry A. DeFerrari. Effects of horizontally varying internal wavefields on multipath interference for propagation through the deep sound channel. *The Journal of the Acoustical Society of America*, 56(1):40–46, 1974.

[5] R. C. Spindel R. P. Porter and R. J. Jaffee. Acoustic-internal wave interaction at long ranges in the ocean. *The Journal of the Acoustical Society of America*, 56(5):1426–1436, 1974.

[6] R. P. Porter R. C. Spindel and R. J. Jaffee. Long-range sound fluctuations with drifting hydrophones. *The Journal of the Acoustical Society of America*, 56(2):440–446, 1974.

[7] Stanley M. Flatt´e and Frederick D. Tappert. Calculation of the effect of internal waves on oceanic sound transmission. *The Journal of the Acoustical Society of America*, 58(6):1151–1159, 1976.

[8] Lincoln Baxter II and Marshall H. Orr. Fluctuations in sound transmission through internal waves assosiated with the thermocline: A computer model for acoustic transmission through sound velocity fields calculated from thermistor chain, ctd, xbt, and acoustic backscattering. *The Journal of the Acoustical Society of America*, 71(1):61–66, 1982.

[9] Alan B. Coppens James V. Sanders Lawrence E. Kinsler, Austin R. Frey. *Fundamentals of Acoustics, Fourth Edition.* John Wiley and sons, Inc., 2000.

[10] Xue-zhen Zhang Ji-xun Zhou and Peter H. Rogers. Resonant interaction of sound wave with internal solitons in the coastal zone. *The Journal of the Acoustical Society of America*, 90(4):2042–2054, 1991.

[11] D. B. King K. G. Lamb M. Teixeira S. A. Chin-Bing, A. Warn-Varnas and J. A. Hawkins. Analysis of coupled oceanographic and acoustic soliton simulations in the yellow sea: a search for soliton induced resonances. *Mathematics and Computers in Simulation*, 62(1), 2003.

[12] Steven Finette Dirk Tielbürger and Stephen Wolf. Acoustic propagation through an internal wave field in a shallow water waveguide. *The Journal of the Acoustical Society of America*, 101(2):789–808, 1996.

[13] Jens M. Hovem. *Marine Acoustics - The Physics of Sound in Underwater Environments.* Applied Research Laboratories - The University of Texas at Austin and Norwegian University of Science and Technology in Trondheim, 2005.

[14] Michael B. Porter Finn B. Jensen, William A. Kuperman and Henrik Schmidt. *Computational Ocean Acoustics.* AIP Press, 1994.

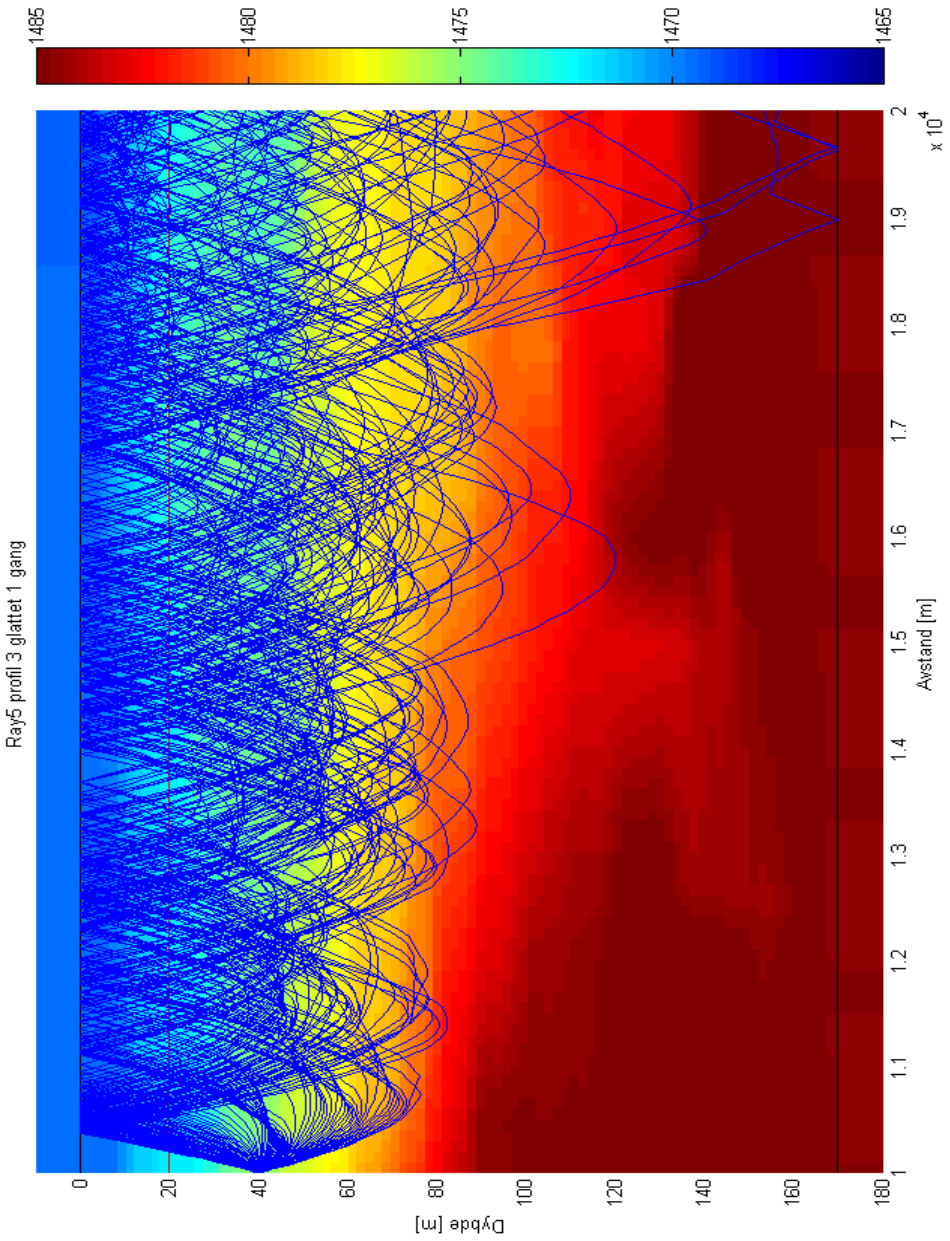
[15] Richard L. Burden and J. Douglas Faires. *Numerical Analysis.* Thomson Brooks/Cole, 2005.

[16] B. Fujii Y. Kagawa, T. Tsuchiya and K. Fujioka. Discrete huygens' model approach to sound wave propagation. *Journal of Sound and Vibration*, 218(3):419–444, 1998.

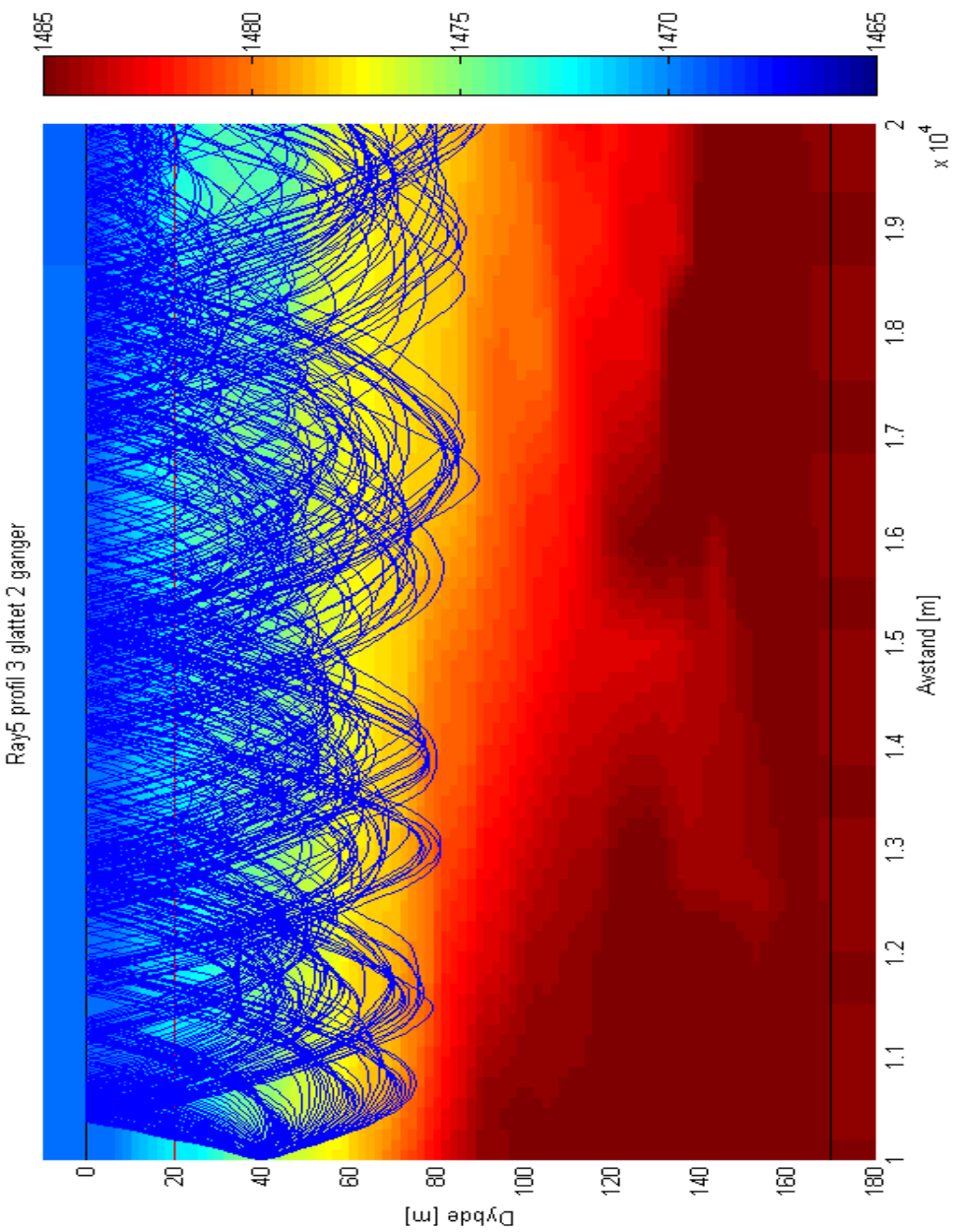
Tillegg A

figurer

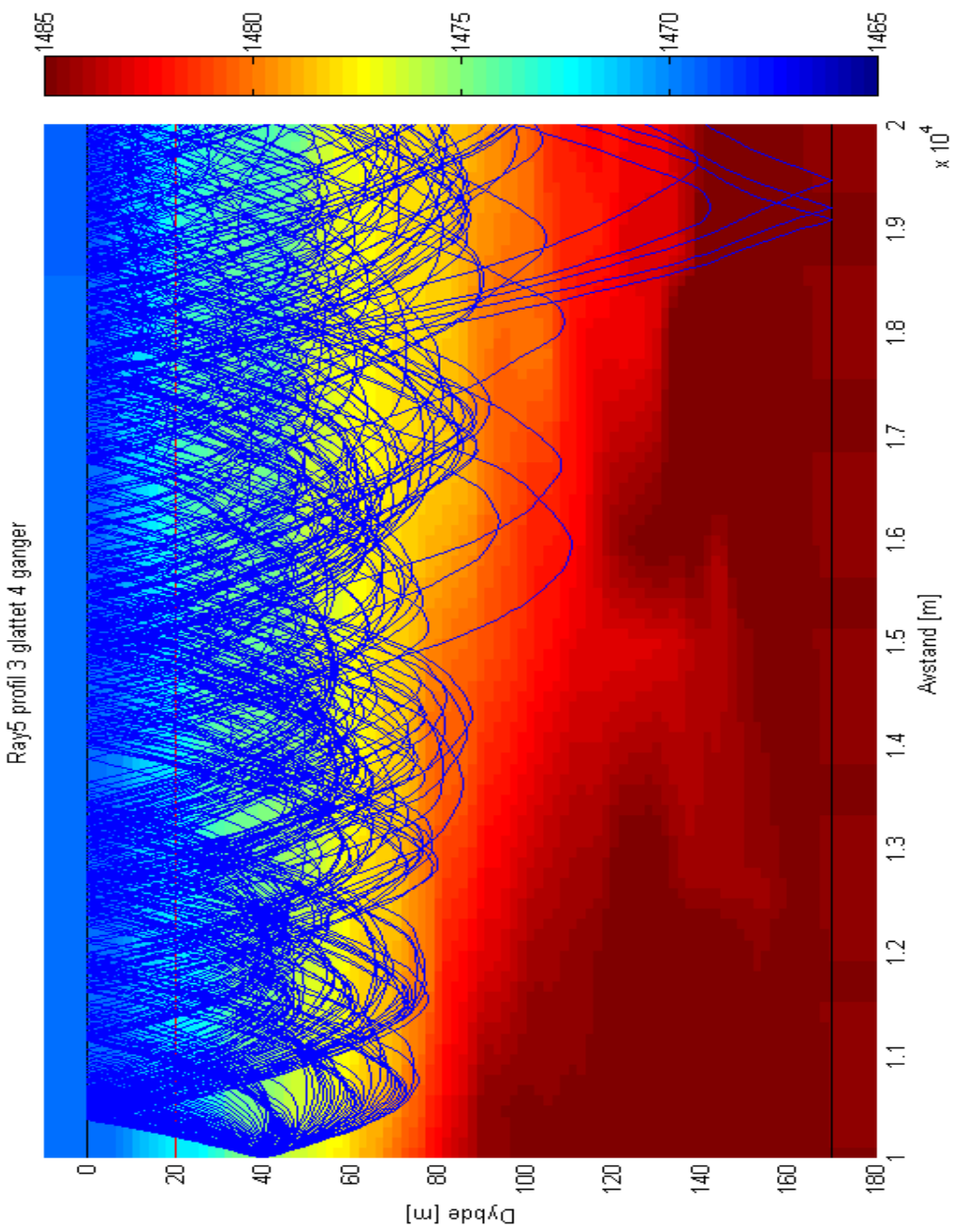
A.1 Ray5: glattede strålegangsplott med hastighetsprofilen i bakgrunnen



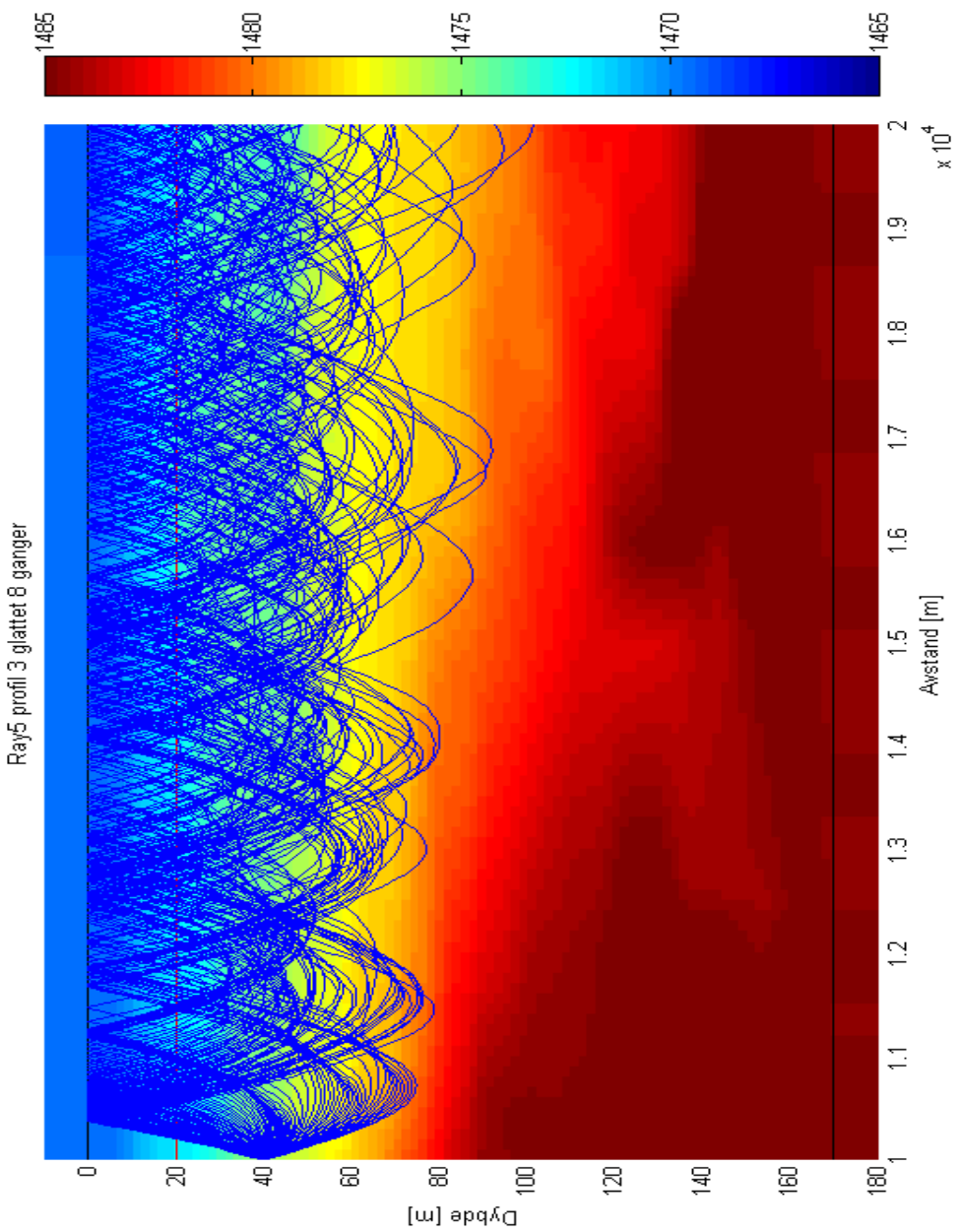
Figur A.1: Ray5 strålegangsberegning for datasett 3 glattet 1 gang



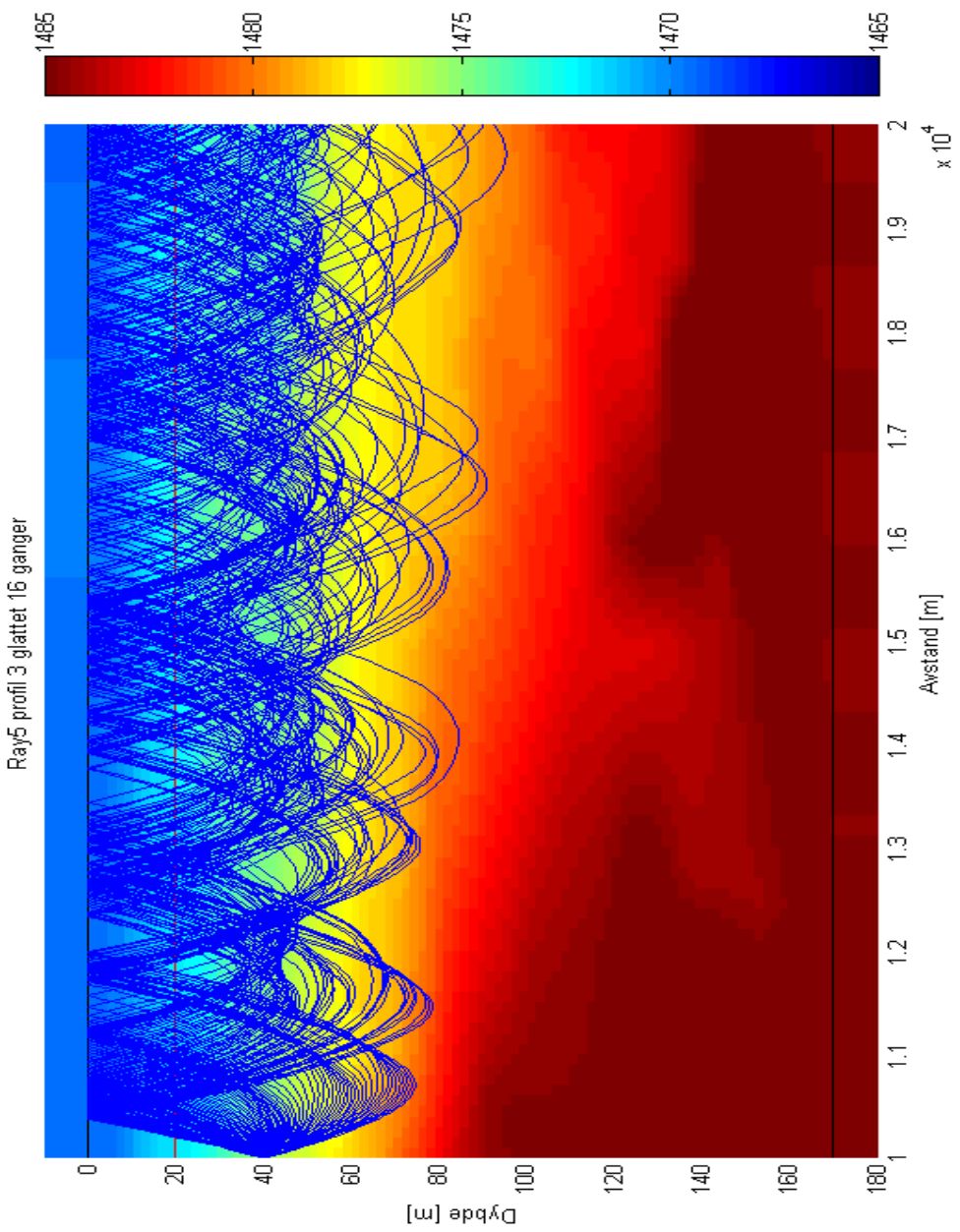
Figur A.2: Ray5 strålegangsberegning for datasett 3 glattet 2 ganger



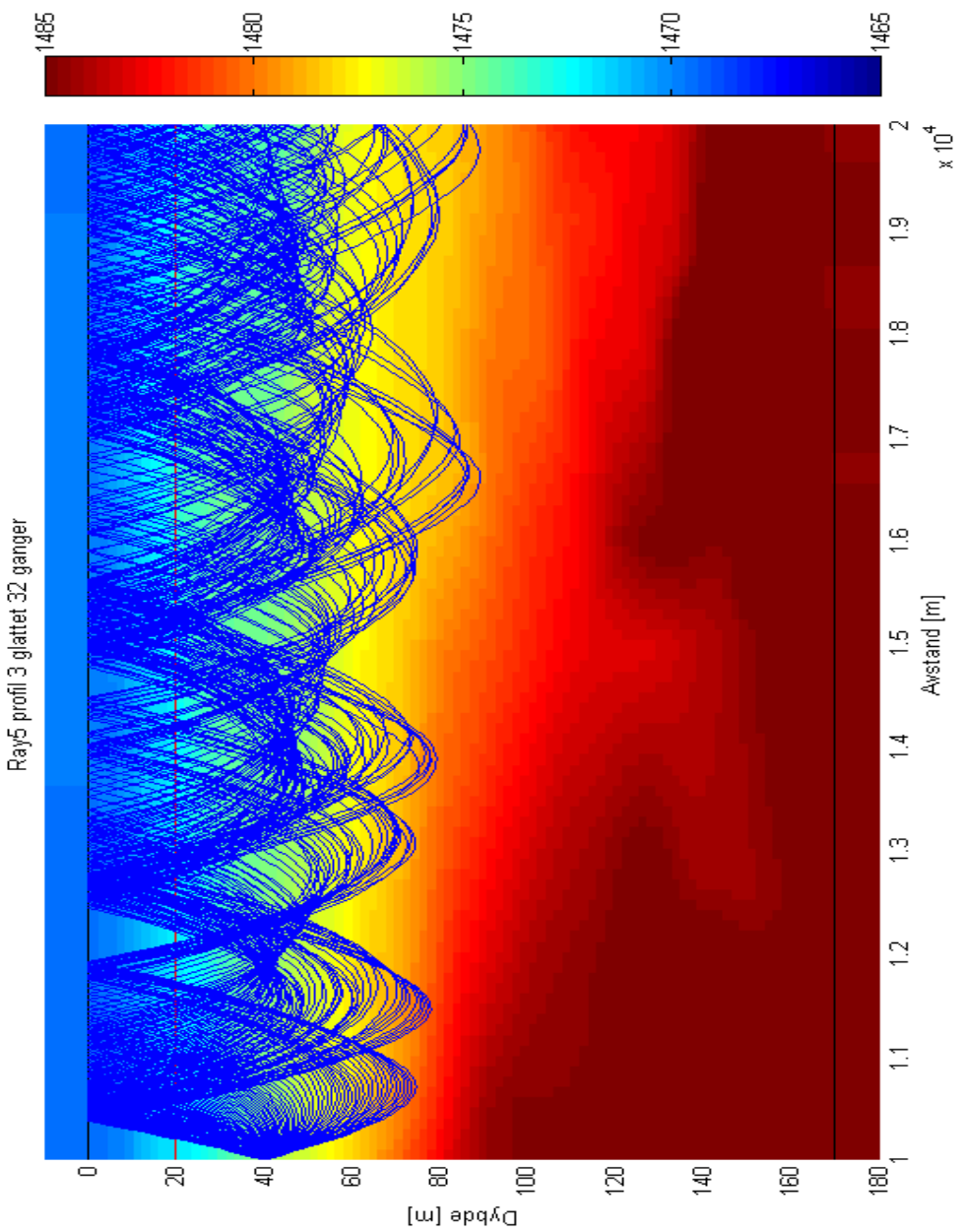
Figur A.3: Ray5 strålegangsberegning for datasett 3 glattet 4 ganger



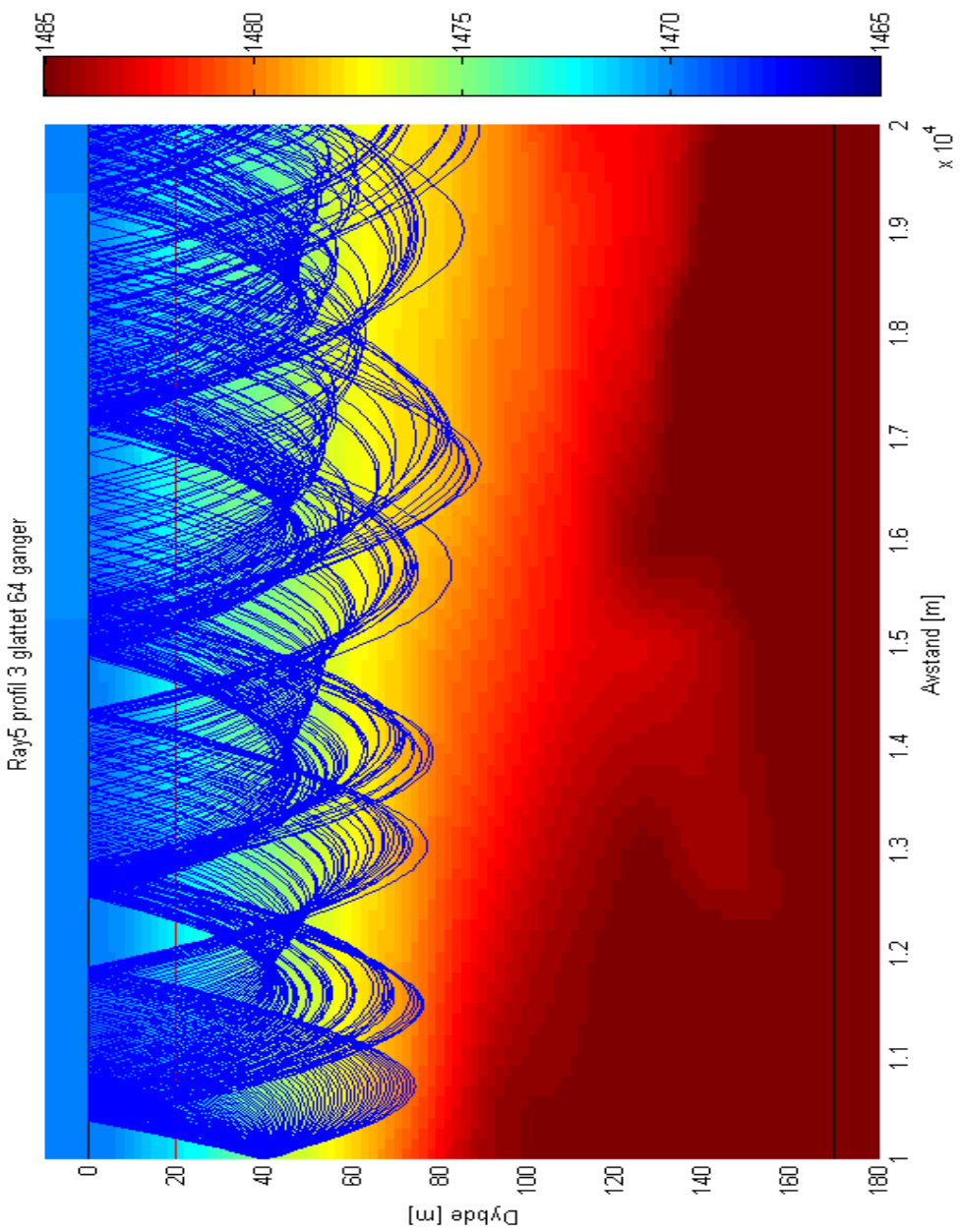
Figur A.4: Ray5 strålegangsberegning for datasett 3 glattet 8 ganger



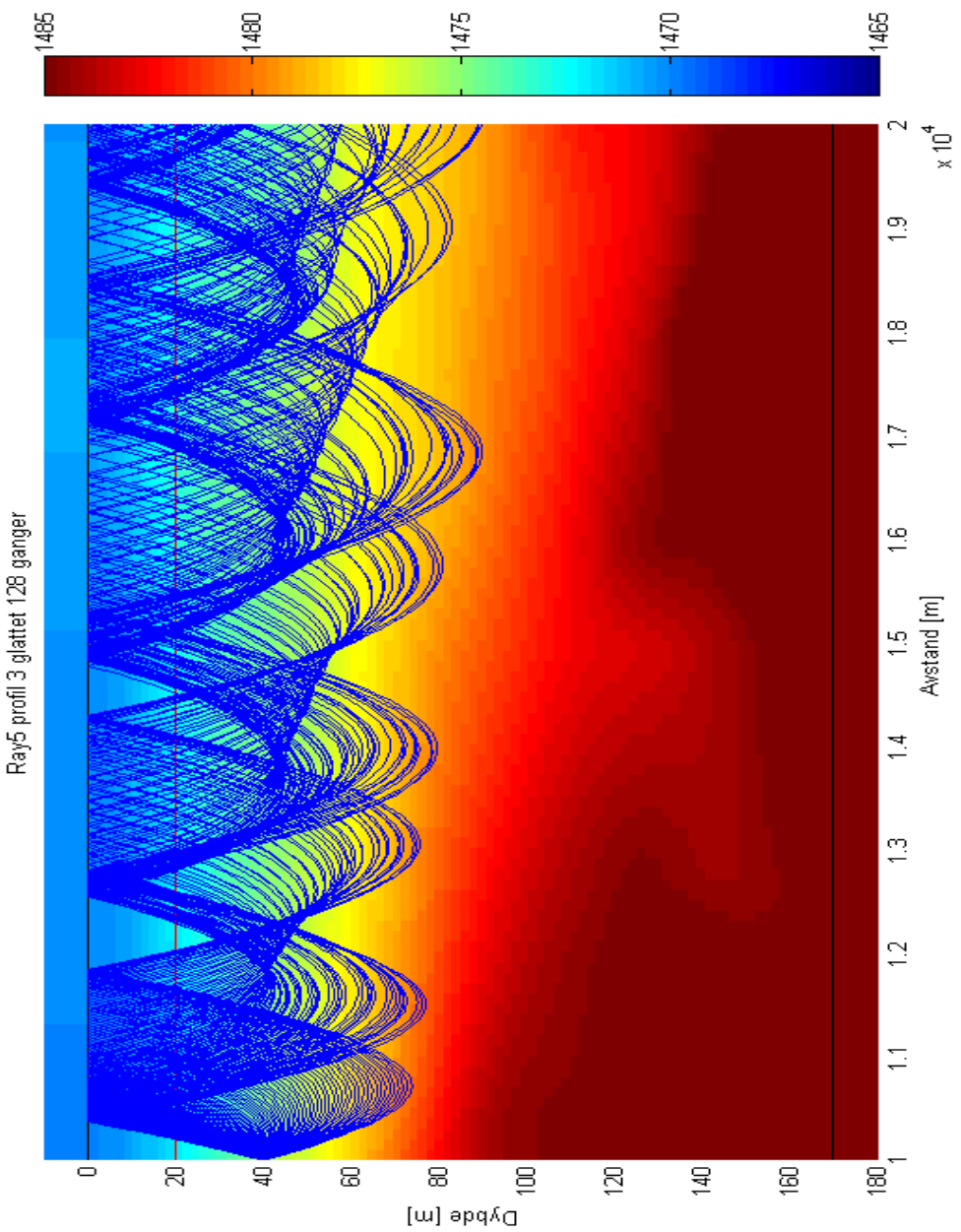
Figur A.5: Ray5 strålegangsberegning for datasett 3 glattet 16 ganger



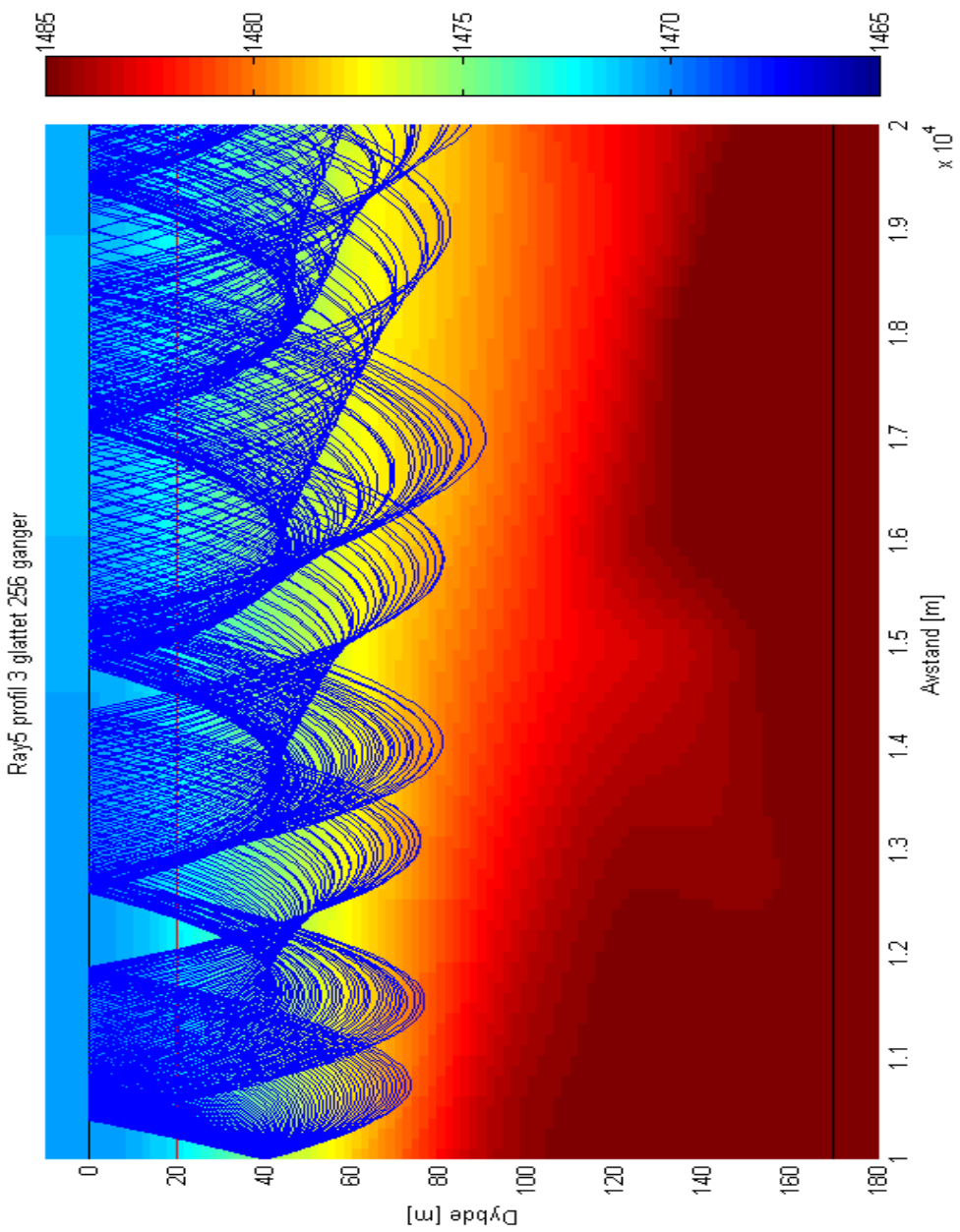
Figur A.6: Ray5 strålegangsberegning for datasett 3 glattet 32 ganger



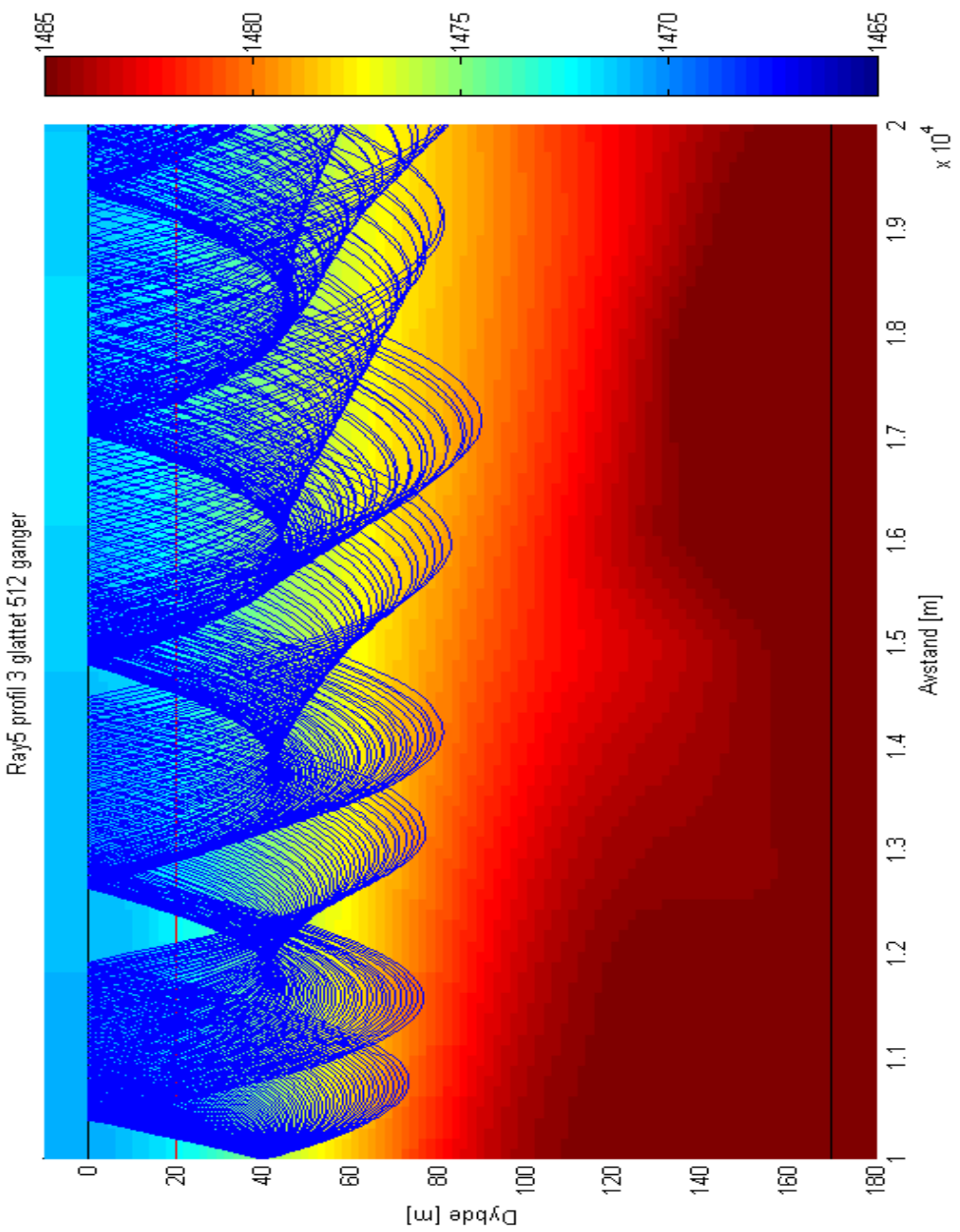
Figur A.7: Ray5 strålegangsberegning for datasett 3 glattet 64 ganger



Figur A.8: Ray5 strålegangsberegning for datasett 3 glattet 128 ganger



Figur A.9: Ray5 strålegangsberegning for datasett 3 glattet 256 ganger



Figur A.10: Ray5 strålegangsberegning for datasett 3 glattet 512 ganger

Tillegg B

Matlab-kode

B.1 Matlab-kode for Lybin

B.1.1 Eksempelscript for Lybin

```
% initialiserer LYBIN
lb=actxserver('LybinCom.LybinModelComBin');

% Initialiserer inputt-metoder
env = lb.invoke('IEnvironment'); % Miljø (bunnprofile,
bunntype, vindhastighet etc)
mod = lb.invoke('IModelData'); % Modellparametre (antall
stråler, cellestørrelser etc)
sensor = lb.invoke('ISensor'); % Sonarparametre (strålebredde,
kildestyrke etc)
pulse = lb.invoke('IPuls'); % pulsegenskaper (båndbredde,
pulslengde)
platform = lb.invoke('IPlatform'); % Platformegenskaper
(egenstøy)
ocean = lb.invoke('IOcean'); % Havegenskaper (Ph, støynivå)

% Setter inn sensordata
sensor.Depth = 40;
sensor.TiltReceiver = 0;
sensor.TiltTransmitter = 0;
sensor.SideLobeReceiver = 12;
sensor.SideLobeTransmitter = 12;
sensor.DetectionThreshold = 13;
sensor.Frequency = 1000;
sensor.DirectivityIndex = 25;
sensor.SourceLevel = 210;
sensor.BeamWidthReceiver = 180; % Vertikal strålebredde
```

```
23 sensor.BeamWidthTransmitter = 180; % Vertikal strålebredde
24
25 % Pulsegenskaper
26 pulse.Length = 1000; % i millisekund
27 pulse.FMBandWidth = 1000; % i Hz
28
29 % Modellegenskaper
30 R = 20000;          % Rekkevidde i modellen
31 Z = 600;                % Dybde
32 R_cells = 100;  % Rekkeviddeceller
33 Z_cells = 50;       % Dybdeceller
34 mod.SetRangeScaleAndRangeCells(R, R_cells); % Første inputt er
   avstand og andre er antall rekkeviddeceller. De må settes
   sammen og ratioen må være et heltall.
35 mod.SetRangeScaleAndRangeSteps(R, R_cells*10); % rekkevidde og
   steglengde. La steglengden typisk være en tidel av
   cellestørrelsen
36 mod.TRLRays = 1000; % Antall stråler. Bruk typisk mer enn 10
   ganger antall rekkeviddeceller
37 mod.SetDepthScaleAndDepthCells(Z, Z_cells);% Første inputt er
   dyp og andre er antall dybdeceller. De må settes sammen og
   ratioen må være et heltall.
38 mod.SetDepthScaleAndDepthSteps(Z,Z_cells*10); % dybde og
   steglengde. La steglengden typisk være en tidel av
   cellestørrelsen
39
40 % Målstyrke
41 ocean.TargetStrength = 10; % Målstyrken
42
43 % Miljøparametre
44 env.WindSpeedMeasurment = [0,5000,10; 5000,10000,6];
45 % Rekkeviddeavhengig vindhastighet. Første kolonne er fra.
46 % Andre er til. Tredje er vindhastighet. Så eksemplet har
47 % 10m/s vind fra 0 til 5km, og 6m/s fra 5km til 10km.
48 SetFirstSoundSpeedProfile(env, 0, 10000, [  0, 7, 35, 1750;
49 620, 8, 34, 1750])
50 % Setter første lydhastighetsprofil. Gjelder i avstanden
51 % 0 til 10000 meter. I siste input som er en matrise angir første
52 % kolonne dybde, andre temperatur, tredje salinitet og fjerde
53 % lydhastighet.
54 AddSoundSpeedProfile(env, 10000, 15000, [  0, 7, 35, 1750;
55 620, 8, 34, 1400])
56 % Setter neste lydhastighetsprofil på samme måte som første.
57 % Ser at denne gjelder fra 10000 til 15000 meter
58 AddSoundSpeedProfile(env, 15000, 20000, [    0, 7, 35, 1750;
59 620, 8, 34, 1750])
60 % Setter en tredje lydhastighetsprofil. Denne gjelder fra
61 % 15000 til 20000 meter.
62 env.BottomType = [0,5000,3;5000,10000,4]; % Bunntype. Første
```

```
% kolonne er fra, andre er til, tredje er bunntype. Lave
% verdier = hard bunn, høye verdier = myk bunn. Dette avgjør
% både bunnens refleksjonsevne og tilbakespredningsfaktor.
prof = [0,200;5000,400; 10000,600];
env.bottomProfile = prof; % Bunnprofil. første kolonne er
% rekkevidde. Andre kolonne er dyp.
env.AddBottomLossFan(0,0,[0,40;10,50; 90, 60]);
% Bunntapsverdier med tap (i dB) mot vinkel (i grader).
mod.UseMeasuredBottomLoss = 1; % Skrur på bunntapstabell.

% Kjører LYBIN
lb.DoCalculation

% Henter data
data.trl.forward = lb.GetResultsBin(0); % Fra sender og
% ut i volumet
data.trl.backward = lb.GetResultsBin(1); % Fra volumet tilbake
% til mottager
data.sig = lb.GetResultsBin(2);
data.pod = lb.GetResultsBin(3);
data.rev.Tot_rev = lb.GetResultsBin(4);
data.rev.Surf_rev = lb.GetResultsBin(5);
data.rev.Vol_rev = lb.GetResultsBin(6);
data.rev.Bot_rev = lb.GetResultsBin(7);
data.rev.Noise = lb.GetResultsBin(8);

% Plotteparametre
r = R/R_cells * [.5:(R_cells-.5)];
z = Z/Z_cells * [.5:(Z_cells-.5)];

% Plotter data
figure
contourf(r/1000,z,10*log10(data.trl.forward), 30,'linestyle',
'none')
set(gca, 'ydir', 'reverse')
xlabel('Range [km]')
ylabel('Depth [m]')
title('Transmission loss')
hold on
fill([prof(:,1);0;0]/1000, [prof(:,2);Z;Z], 'k');

figure
contourf(r/1000,z,10*log10(data.sig), 30,'linestyle', 'none')
set(gca, 'ydir', 'reverse')
xlabel('Range [km]')
ylabel('Depth [m]')
title('Signal excess')
hold on
fill([prof(:,1);0;0]/1000, [prof(:,2);Z;Z], 'k');
```

```
111
112 figure
113 contourf(r/1000,z,data.pod, 30,'linestyle', 'none')
114 set(gca, 'ydir', 'reverse')
115 xlabel('Range [km]')
116 ylabel('Depth [m]')
117 title('Propability of detection')
118 hold on
119 fill([prof(:,1);0;0]/1000, [prof(:,2);Z;Z], 'k');
120
121 figure
122 plot(r/1000, 10*log10(data.rev.Tot_rev))
123 hold on
124 plot(r/1000, 10*log10(data.rev.Bot_rev), 'r')
125 plot(r/1000, 10*log10(data.rev.Surf_rev), 'g')
126 plot(r/1000, 10*log10(data.rev.Vol_rev), 'c')
127 title('Reverberation')
128 legend('Total', 'Bottom', 'Surface', 'Volume')
129 xlabel('Range [km]')
130 ylabel('Rev [dB]')
131
132 % Kjører "vanlig" med samme inputt
133 s = lb.modelData; % Henter ut inputtdata i XML-format
134 fid = fopen('lybin.xml', 'w');
135 fprintf(fid, '%s', s); % Legger XML-data i XML-fil
136 dos('lybin') % Kjører vanlig LYBIN med XML-data
```

B.1.2 Script for hastighetsprofil 3 med 25 avstandsblokker

```
1  % initialiserer LYBIN
2  lb=actxserver('LybinCom.LybinModelComBin');
3
4  % Initialiserer inputt-metoder
5  env = lb.invoke('IEnvironment');
6  mod = lb.invoke('IModelData');
7  sensor = lb.invoke('ISensor');
8  pulse = lb.invoke('IPuls');
9  platform = lb.invoke('IPlatform');
10 ocean = lb.invoke('IOcean');
11
12 % Setter inn sensordata
13 sensor.Depth = 40;
14 sensor.TiltReceiver = 0;
15 sensor.TiltTransmitter = 0;
16 sensor.SideLobeReceiver = 12;
17 sensor.SideLobeTransmitter = 12;
18 sensor.DetectionThreshold = 13;
19 sensor.Frequency = 1000;
```

```
20 sensor.DirectivityIndex = 25;
21 sensor.SourceLevel = 210;
22 sensor.BeamWidthReceiver = 180;
23 sensor.BeamWidthTransmitter = 10;
24
25 % Pulsegenskaper
26 pulse.Length = 1000; % i millisekund
27 pulse.FMBandWidth = 1000; % i Hz
28
29 % Modellegenskaper
30 R = 10000;
31 Z = 170;
32 R_cells = 500;
33 Z_cells = 34;
34 mod.SetRangeScaleAndRangeSteps(R, R_cells*10);
35 mod.TRLRays = 500;
36 mod.SetDepthScaleAndDepthCells(Z, Z_cells);
37 mod.SetDepthScaleAndDepthSteps(Z,Z_cells*10);
38
39 % Målstyrke
40 ocean.TargetStrength = 10;
41
42 % Laster inn lydhastighetstabell
43 load sound.mat;
44 M=sound.L_3;
45 n=size(M,1); % depth
46 m=size(M,2); % range
47
48 for i=1:m
49     first_num = find(¬isnan(M(:,i)),1,'first');
50     M(1:first_num-1,i)=M(first_num,i);
51     last_num = find(¬isnan(M(:,i)),1,'last');
52     M(last_num+1:n,i)=M(last_num,i);
53 end
54
55 for i=1:n
56     first_num = find(¬isnan(M(i,:)),1,'first');
57     M(i,1:first_num-1)=M(i,first_num);
58     last_num = find(¬isnan(M(i,:)),1,'last');
59     M(i,last_num+1:m)=M(i,last_num);
60 end
61
62 start=1000;
63 num_profiles=25;
64 stop=2000;
65
66 % Avstandsvektor
67 % Legger til en grense . 25 intervaller vil f.eks kreve 26 grenser
68 range = round(linspace(start,stop,num_profiles+1));
```

```
% Dybdevektor
depth = round(linspace(1,170,34));
% Lager 25 vertikale hastighetprofiler som er kolonner plukket
% fra hastighetsmatrisen M i henhold til avstands- og dybdevektoren.
spdprfl_1=zeros(length(depth),4);
spdprfl_1(:,1)=depth; % Legger inn dybdevektor
spdprfl_1(:,2)=8; % Temperatur som holdes konstant
spdprfl_1(:,3)=34; % Salinitet som holdes konstant
spdprfl_1(:,4)=M(depth,range(1)); % Henter en lydhast. kolonne
spdprfl_2=spdprfl_1;spdprfl_2(:,4)=M(depth,range(2));
spdprfl_3=spdprfl_1;spdprfl_3(:,4)=M(depth,range(3));
spdprfl_4=spdprfl_1;spdprfl_4(:,4)=M(depth,range(4));
spdprfl_5=spdprfl_1;spdprfl_5(:,4)=M(depth,range(5));
spdprfl_6=spdprfl_1;spdprfl_6(:,4)=M(depth,range(6));
spdprfl_7=spdprfl_1;spdprfl_7(:,4)=M(depth,range(7));
spdprfl_8=spdprfl_1;spdprfl_8(:,4)=M(depth,range(8));
spdprfl_9=spdprfl_1;spdprfl_9(:,4)=M(depth,range(9));
spdprfl_10=spdprfl_1;spdprfl_10(:,4)=M(depth,range(10));
spdprfl_11=spdprfl_1;spdprfl_11(:,4)=M(depth,range(11));
spdprfl_12=spdprfl_1;spdprfl_12(:,4)=M(depth,range(12));
spdprfl_13=spdprfl_1;spdprfl_13(:,4)=M(depth,range(13));
spdprfl_14=spdprfl_1;spdprfl_14(:,4)=M(depth,range(14));
spdprfl_15=spdprfl_1;spdprfl_15(:,4)=M(depth,range(15));
spdprfl_16=spdprfl_1;spdprfl_16(:,4)=M(depth,range(16));
spdprfl_17=spdprfl_1;spdprfl_17(:,4)=M(depth,range(17));
spdprfl_18=spdprfl_1;spdprfl_18(:,4)=M(depth,range(18));
spdprfl_19=spdprfl_1;spdprfl_19(:,4)=M(depth,range(19));
spdprfl_20=spdprfl_1;spdprfl_20(:,4)=M(depth,range(20));
spdprfl_21=spdprfl_1;spdprfl_21(:,4)=M(depth,range(21));
spdprfl_22=spdprfl_1;spdprfl_22(:,4)=M(depth,range(22));
spdprfl_23=spdprfl_1;spdprfl_23(:,4)=M(depth,range(23));
spdprfl_24=spdprfl_1;spdprfl_24(:,4)=M(depth,range(24));
spdprfl_25=spdprfl_1;spdprfl_25(:,4)=M(depth,range(25));

% Skalerer opp til virkelig avstand og legger utvalget fra 1 - 10 km
new_range=10*(range-1000)

% Miljøparametre
env.WindSpeedMeasurment = [0,10000,0];
% Legger inn de 25 hastighetsprofilene i respektive avtandsblokker
SetFirstSoundSpeedProfile(env,new_range(1), new_range(2),spdprfl_1)
AddSoundSpeedProfile(env, new_range(2), new_range(3),spdprfl_2)
AddSoundSpeedProfile(env, new_range(3), new_range(4),spdprfl_3)
AddSoundSpeedProfile(env, new_range(4), new_range(5),spdprfl_4)
AddSoundSpeedProfile(env, new_range(5), new_range(6),spdprfl_5)
AddSoundSpeedProfile(env, new_range(6), new_range(7),spdprfl_6)
AddSoundSpeedProfile(env, new_range(7), new_range(8),spdprfl_7)
AddSoundSpeedProfile(env, new_range(8), new_range(9),spdprfl_8)
AddSoundSpeedProfile(env, new_range(9), new_range(10),spdprfl_9)
```

```
AddSoundSpeedProfile(env, new_range(10), new_range(11),spdprfl_10)
AddSoundSpeedProfile(env, new_range(11), new_range(12),spdprfl_11)
AddSoundSpeedProfile(env, new_range(12), new_range(13),spdprfl_12)
AddSoundSpeedProfile(env, new_range(13), new_range(14),spdprfl_13)
AddSoundSpeedProfile(env, new_range(14), new_range(15),spdprfl_14)
AddSoundSpeedProfile(env, new_range(15), new_range(16),spdprfl_15)
AddSoundSpeedProfile(env, new_range(16), new_range(17),spdprfl_16)
AddSoundSpeedProfile(env, new_range(17), new_range(18),spdprfl_17)
AddSoundSpeedProfile(env, new_range(18), new_range(19),spdprfl_18)
AddSoundSpeedProfile(env, new_range(19), new_range(20),spdprfl_19)
AddSoundSpeedProfile(env, new_range(20), new_range(21),spdprfl_20)
AddSoundSpeedProfile(env, new_range(21), new_range(22),spdprfl_21)
AddSoundSpeedProfile(env, new_range(22), new_range(23),spdprfl_22)
AddSoundSpeedProfile(env, new_range(23), new_range(24),spdprfl_23)
AddSoundSpeedProfile(env, new_range(24), new_range(25),spdprfl_24)
AddSoundSpeedProfile(env, new_range(25), new_range(26),spdprfl_25)

env.BottomType = [0,5000,1;5000,10000,1];
prof = [0,170;10000,170;];
env.bottomProfile = prof;
env.AddBottomLossFan(0,0,[0,0;10,0; 90, 0]);
mod.UseMeasuredBottomLoss = 1;

% Kjører LYBIN
lb.DoCalculation

% Henter data
data.trl.forward = lb.GetResultsBin(0);
data.trl.backward = lb.GetResultsBin(1);
data.sig = lb.GetResultsBin(2);
data.pod = lb.GetResultsBin(3);
data.rev.Tot_rev = lb.GetResultsBin(4);
data.rev.Surf_rev = lb.GetResultsBin(5);
data.rev.Vol_rev = lb.GetResultsBin(6);
data.rev.Bot_rev = lb.GetResultsBin(7);
data.rev.Noise = lb.GetResultsBin(8);

% Kjører "vanlig" med samme inputt
s = lb.modelData; % Henter ut inputtdata i XML-format
fid = fopen('datasett3.xml', 'w');
fprintf(fid, '%s', s); % Legger XML-data i XML-fil
% dos('lybin') % Kjører vanlig LYBIN med XML-data
```

B.1.3 Script for plotting av transmisjonstap i Lybin

```
s_depth=round(20/170*34)
lengde=length(data.trl.backward(s_depth,:));
spaces=100;
```

```
space_size=lengde/spaces;
vektor=zeros(1,spaces);
for i=0:(spaces-1)
    for j=1:space_size
        vektor(i+1)=vektor(i+1)+ data.trl.forward(s_depth,(i*space_size+j));
    end
end
vektor;

figure(4)
x_plot=(10.1:0.1:20);
stairs(x_plot,vektor)

xlabel('Avstand [km]')
ylabel('Transmisjonstap')
title('LYBIN: Hastighetsprofil 3. zs=40m, zr=20m, theta=-5:0.1:5')
```

B.1.4 Script for n^2 hastighetsprofil

```
% initialiserer LYBIN
lb=actxserver('LybinCom.LybinModelComBin');

% Initialiserer inputt-metoder
env = lb.invoke('IEnvironment');
mod = lb.invoke('IModelData');
sensor = lb.invoke('ISensor');
pulse = lb.invoke('IPuls');
platform = lb.invoke('IPlatform');
ocean = lb.invoke('IOcean');

% Setter inn sensordata
sensor.Depth = 40;
sensor.TiltReceiver = 0;
sensor.TiltTransmitter = 0;
sensor.SideLobeReceiver = 12;
sensor.SideLobeTransmitter = 12;
sensor.DetectionThreshold = 13;
sensor.Frequency = 1000;
sensor.DirectivityIndex = 25;
sensor.SourceLevel = 210;
sensor.BeamWidthReceiver = 180; % Vertikal strålebredde
sensor.BeamWidthTransmitter = 20; % Vertikal strålebredde

% Pulsegenskaper
pulse.Length = 1000; % i millisekund
pulse.FMBandWidth = 1000; % i Hz

% Modellegenskaper
```

```
30 R = 10000;
31 Z = 200;
32 R_cells = 500;
33 Z_cells = 200;
34 mod.SetRangeScaleAndRangeCells(R, R_cells);
35 mod.SetRangeScaleAndRangeSteps(R, R_cells*10);
36 mod.TRLRays = 1000;
37 mod.SetDepthScaleAndDepthCells(Z, Z_cells);
38 mod.SetDepthScaleAndDepthSteps(Z,Z_cells*10);
39
40 % Målstyrke
41 ocean.TargetStrength = 10; % Målstyrken
42
43 % Miljøparametre
44 env.WindSpeedMeasurment = [0,10000,0];
45
46 c0=1500; % lydahstighet på toppen
47 c1=1460; % lydhastighet på bunnen
48 D=200;   % dybden
49 b=((c0/c1)^2-1)/D;
50 % jalla=sqrt(1+b*z)
51 resol=201;
52 profil=zeros(resol,4);
53
54 depth=linspace(0,200,resol);
55 c_of_z = c0./(sqrt(1+b*depth)); %lager lydhastighetsvektor
56
57 %lager profilen
58 profil(:,1)=depth;
59 profil(:,2)=8;
60 profil(:,3)=34;
61 profil(:,4)=c_of_z;
62
63 SetFirstSoundSpeedProfile(env, 0, 10000, profil) %setter inn profilen
64 env.BottomType = [0,1500,1;1500,10000,1];
65 prof = [0,180;10000,180;];
66 env.bottomProfile = prof;
67 env.AddBottomLossFan(0,0,[0,0;10,0; 90, 0]);
68 mod.UseMeasuredBottomLoss = 1; %
69
70 % Kjører LYBIN
71 lb.DoCalculation
72
73 % Henter data
74 data.trl.forward = lb.GetResultsBin(0);
75 data.trl.backward = lb.GetResultsBin(1);
76 data.sig = lb.GetResultsBin(2);
77 data.pod = lb.GetResultsBin(3);
78 data.rev.Tot_rev = lb.GetResultsBin(4);
```

```
data.rev.Surf_rev = lb.GetResultsBin(5);
data.rev.Vol_rev = lb.GetResultsBin(6);
data.rev.Bot_rev = lb.GetResultsBin(7);
data.rev.Noise = lb.GetResultsBin(8);

% Plotteparametre
r = R/R_cells * [.5:(R_cells-.5)];
z = Z/Z_cells * [.5:(Z_cells-.5)];

% Plotter data
figure
contourf(r/1000,z,10*log10(data.trl.forward), 30,'linestyle', 'none')
set(gca, 'ydir', 'reverse')
xlabel('Range [km]')
ylabel('Depth [m]')
title('Transmission loss')
hold on
fill([prof(:,1);0;0]/1000, [prof(:,2);Z;Z], 'k');


% Kjører "vanlig" med samme inputt
s = lb.modelData; % Henter ut inputtdata i XML-format
fid = fopen('oasesref.xml', 'w');
fprintf(fid, '%s', s); % Legger XML-data i XML-fil
% dos('lybin') % Kjører vanlig LYBIN med XML-data
```

B.1.5 Script for glatting i Lybin

```
% initialiserer LYBIN
lb=actxserver('LybinCom.LybinModelComBin');

% Initialiserer inputt-metoder
env = lb.invoke('IEnvironment');
mod = lb.invoke('IModelData');
sensor = lb.invoke('ISensor');
pulse = lb.invoke('IPuls');
platform = lb.invoke('IPlatform');
ocean = lb.invoke('IOcean');

% Setter inn sensordata
sensor.Depth = 40;
sensor.TiltReceiver = 0;
sensor.TiltTransmitter = 0;
sensor.SideLobeReceiver = 43;
sensor.SideLobeTransmitter = 43;
sensor.DetectionThreshold = 13;
sensor.Frequency = 1000;
sensor.DirectivityIndex = 25;
```

```
21 sensor.SourceLevel = 210;
22 sensor.BeamWidthReceiver = 10;
23 sensor.BeamWidthTransmitter = 10;
24
25 % Pulsegenskaper
26 pulse.Length = 1000; % i millisekund
27 pulse.FMBandWidth = 1000; % i Hz
28
29 % Modellegenskaper
30 R = 10000;
31 Z = 170;
32 R_cells = 100;
33 Z_cells = 34;
34 mod.SetRangeScaleAndRangeCells(R, R_cells);
35 mod.SetRangeScaleAndRangeSteps(R, R_cells*10);
36 mod.TRLRays = 1000;
37 mod.SetDepthScaleAndDepthCells(Z, Z_cells);
38 mod.SetDepthScaleAndDepthSteps(Z,Z_cells*10);
39
40 % Målstyrke
41 ocean.TargetStrength = 10; % Målstyrken
42
43 % Laster inn lydhastighetstabell
44 load sound.mat;
45 M_grid=sound.L_3;
46 % Fyller hastighetsprofilen med veridier
47 % og glatter 50 ganger med Kagawa.
48 M=smooth(M_grid,50);
49
50 start=1000;
51 num_profiles=25;
52 stop=2000;
53
54 range = round(linspace(start,stop,num_profiles+1));
55 depth = round(linspace(1,170,34));
56
57 spdprfl_1=zeros(length(depth),4);spdprfl_1(:,1)=depth;spdprfl_1(:,2)=8;spdprfl_1(:,3)=
58 spdprfl_2=spdprfl_1;spdprfl_2(:,4)=M(depth,range(2));
59 spdprfl_3=spdprfl_1;spdprfl_3(:,4)=M(depth,range(3));
60 spdprfl_4=spdprfl_1;spdprfl_4(:,4)=M(depth,range(4));
61 spdprfl_5=spdprfl_1;spdprfl_5(:,4)=M(depth,range(5));
62 spdprfl_6=spdprfl_1;spdprfl_6(:,4)=M(depth,range(6));
63 spdprfl_7=spdprfl_1;spdprfl_7(:,4)=M(depth,range(7));
64 spdprfl_8=spdprfl_1;spdprfl_8(:,4)=M(depth,range(8));
65 spdprfl_9=spdprfl_1;spdprfl_9(:,4)=M(depth,range(9));
66 spdprfl_10=spdprfl_1;spdprfl_10(:,4)=M(depth,range(10));
67 spdprfl_11=spdprfl_1;spdprfl_11(:,4)=M(depth,range(11));
68 spdprfl_12=spdprfl_1;spdprfl_12(:,4)=M(depth,range(12));
69 spdprfl_13=spdprfl_1;spdprfl_13(:,4)=M(depth,range(13));
```

```
spdprfl_14=spdprfl_1;spdprfl_14(:,4)=M(depth,range(14));
spdprfl_15=spdprfl_1;spdprfl_15(:,4)=M(depth,range(15));
spdprfl_16=spdprfl_1;spdprfl_16(:,4)=M(depth,range(16));
spdprfl_17=spdprfl_1;spdprfl_17(:,4)=M(depth,range(17));
spdprfl_18=spdprfl_1;spdprfl_18(:,4)=M(depth,range(18));
spdprfl_19=spdprfl_1;spdprfl_19(:,4)=M(depth,range(19));
spdprfl_20=spdprfl_1;spdprfl_20(:,4)=M(depth,range(20));
spdprfl_21=spdprfl_1;spdprfl_21(:,4)=M(depth,range(21));
spdprfl_22=spdprfl_1;spdprfl_22(:,4)=M(depth,range(22));
spdprfl_23=spdprfl_1;spdprfl_23(:,4)=M(depth,range(23));
spdprfl_24=spdprfl_1;spdprfl_24(:,4)=M(depth,range(24));
spdprfl_25=spdprfl_1;spdprfl_25(:,4)=M(depth,range(25));

%skalerer opp til virkelig avstand og legger utvalget fra 1 - 10 km
new_range=10*(range-1000);
% Miljøparametre
env.WindSpeedMeasurment = [0,10000,0];
SetFirstSoundSpeedProfile(env,new_range(1), new_range(2),spdprfl_1)
AddSoundSpeedProfile(env, new_range(2), new_range(3),spdprfl_2)
AddSoundSpeedProfile(env, new_range(3), new_range(4),spdprfl_3)
AddSoundSpeedProfile(env, new_range(4), new_range(5),spdprfl_4)
AddSoundSpeedProfile(env, new_range(5), new_range(6),spdprfl_5)
AddSoundSpeedProfile(env, new_range(6), new_range(7),spdprfl_6)
AddSoundSpeedProfile(env, new_range(7), new_range(8),spdprfl_7)
AddSoundSpeedProfile(env, new_range(8), new_range(9),spdprfl_8)
AddSoundSpeedProfile(env, new_range(9), new_range(10),spdprfl_9)
AddSoundSpeedProfile(env, new_range(10), new_range(11),spdprfl_10)
AddSoundSpeedProfile(env, new_range(11), new_range(12),spdprfl_11)
AddSoundSpeedProfile(env, new_range(12), new_range(13),spdprfl_12)
AddSoundSpeedProfile(env, new_range(13), new_range(14),spdprfl_13)
AddSoundSpeedProfile(env, new_range(14), new_range(15),spdprfl_14)
AddSoundSpeedProfile(env, new_range(15), new_range(16),spdprfl_15)
AddSoundSpeedProfile(env, new_range(16), new_range(17),spdprfl_16)
AddSoundSpeedProfile(env, new_range(17), new_range(18),spdprfl_17)
AddSoundSpeedProfile(env, new_range(18), new_range(19),spdprfl_18)
AddSoundSpeedProfile(env, new_range(19), new_range(20),spdprfl_19)
AddSoundSpeedProfile(env, new_range(20), new_range(21),spdprfl_20)
AddSoundSpeedProfile(env, new_range(21), new_range(22),spdprfl_21)
AddSoundSpeedProfile(env, new_range(22), new_range(23),spdprfl_22)
AddSoundSpeedProfile(env, new_range(23), new_range(24),spdprfl_23)
AddSoundSpeedProfile(env, new_range(24), new_range(25),spdprfl_24)
AddSoundSpeedProfile(env, new_range(25), new_range(26),spdprfl_25)

env.BottomType = [0,5000,1;5000,10000,1];
prof = [0,170;10000,170;];
env.bottomProfile = prof;.
env.AddBottomLossFan(0,0,[0,0;10,0; 90, 0]);
mod.UseMeasuredBottomLoss = 1;

```

```
% Kjører LYBIN
lb.DoCalculation

% Henter data
data.trl.forward = lb.GetResultsBin(0);
data.trl.backward = lb.GetResultsBin(1);
data.sig = lb.GetResultsBin(2);
data.pod = lb.GetResultsBin(3);
data.rev.Tot_rev = lb.GetResultsBin(4);
data.rev.Surf_rev = lb.GetResultsBin(5);
data.rev.Vol_rev = lb.GetResultsBin(6);
data.rev.Bot_rev = lb.GetResultsBin(7);
data.rev.Noise = lb.GetResultsBin(8);

% Kjører "vanlig" med samme inputt
s = lb.modelData; % Henter ut inputtdata i XML-format
fid = fopen('datasett3.xml', 'w');
fprintf(fid, '%s', s); % Legger XML-data i XML-fil
% dos('lybin') % Kjører vanlig LYBIN med XML-data
```

B.1.6 Script for konvergensplott som funksjon av antall blokker

```
% initialiserer LYBIN
lb=actxserver('LybinCom.LybinModelComBin');

% Initialiserer inputt-metoder
env = lb.invoke('IEnvironment');
mod = lb.invoke('IModelData');  etc)
sensor = lb.invoke('ISensor');
pulse = lb.invoke('IPuls');
platform = lb.invoke('IPlatform');
ocean = lb.invoke('IOcean');

% Setter inn sensordata
sensor.Depth = 40;
sensor.TiltReceiver = 0;
sensor.TiltTransmitter = 0;
sensor.SideLobeReceiver = 43;
sensor.SideLobeTransmitter = 43;
sensor.DetectionThreshold = 13;
sensor.Frequency = 1000;
sensor.DirectivityIndex = 25;
sensor.SourceLevel = 210;
sensor.BeamWidthReceiver = 10;
sensor.BeamWidthTransmitter = 10;

```

```
% Pulsegenskaper
pulse.Length = 1000; % i millisekund
pulse.FMBandWidth = 1000; % i Hz

% Modellegenskaper
R = 10000;
Z = 170;
R_cells = 100;
Z_cells = 34;
mod.SetRangeScaleAndRangeCells(R, R_cells);
mod.SetRangeScaleAndRangeSteps(R, R_cells*10);
mod.TRLRays = 1000;
mod.SetDepthScaleAndDepthCells(Z, Z_cells)
mod.SetDepthScaleAndDepthSteps(Z,Z_cells*10);

% Målstyrke
ocean.TargetStrength = 10;

% Laster inn lydhastighetstabell
load sound.mat;
M=sound.L_3;
n=size(M,1); % depth
m=size(M,2); % range

for i=1:m
    first_num = find(¬isnan(M(:,i)),1,'first');
    M(1:first_num-1,i)=M(first_num,i);
    last_num = find(¬isnan(M(:,i)),1,'last');
    M(last_num+1:n,i)=M(last_num,i);
end

for i=1:n
    first_num = find(¬isnan(M(i,:)),1,'first');
    M(i,1:first_num-1)=M(i,first_num);
    last_num = find(¬isnan(M(i,:)),1,'last');
    M(i,last_num+1:m)=M(i,last_num);
end

% Glatter profilen 50 ganger med Kagawas metode
glatt_M=smooth(M,50);
M=glatt_M;

runs=200
error10=zeros(1,runs-10);
norm_error10=zeros(1,runs-10);
error1=zeros(1,runs);
norm_error1=zeros(1,runs);
trl=zeros(Z_cells,R_cells,runs);
for k=1:runs;
```

```
74      start=1000;
75      num_profiles=k;
76      stop=2000;
77      % Avstandsvektor med grensene for hver blokk
78      % Legger til en grense. 25 intervaller vil f.eks kreve 26 grenser
79      range = round(linspace(start,stop,num_profiles+1));
80      % Dybdevektor. Trenger ikke ha flere elmenter enn antall dybdeceller
81      depth = round(linspace(1,170,34));
82      % Lager k antall hastihetsprofiler (for hver blokk)
83      spdprfl=zeros(length(depth),4,num_profiles);
84      for i=1:num_profiles
85          spdprfl(:,1,i)=depth;
86          spdprfl(:,2,i)=8;
87          spdprfl(:,3,i)=34;
88          spdprfl(:,4,i)=M(depth,range(i));
89      end
90
91      % Skalerer opp til virkelig avstand og legger utvalget fra 1-10 km
92      new_range=10*(range-1000);
93
94      % Miljøparametre
95      env.WindSpeedMeasurment = [0,10000,0];
96      % Legger hastighetsprofilene inn i modellen
97      for j=1:num_profiles
98          if j==1
99              SetFirstSoundSpeedProfile(env,new_range(1),
    new_range(2),spdprfl(:,:,1))
100         else
101             AddSoundSpeedProfile(env, new_range(j),
    new_range(j+1),spdprfl(:,:,j))
102         end
103     end
104     env.BottomType = [0,5000,1;5000,10000,1];
105     prof = [0,170;10000,170;];
106     env.bottomProfile = prof;
107     env.AddBottomLossFan(0,0,[0,0;10,0; 90, 0]);
108     mod.UseMeasuredBottomLoss = 1;
109
110     % Kjører LYBIN
111     lb.DoCalculation
112
113     % Henter data
114     trl(:,:,k) = lb.GetResultsBin(0);
115
116     %Beregner differansevektor
117     diff1=abs(trl(:,:,k)-trl(:,:,1));
118     diff1_sum=sum(sum(diff1));
119     error1(1,k)=diff1_sum/(R_cells*Z_cells);
120
```

```
121     %Normalisert differansevektor
122     trl_k_sum=sum(sum(trl(:,:,k)));
123     norm_error1(1,k)=diff1_sum/trl_k_sum;
124 end
125 
126 figure
127 plot(error1)
128 xlabel('Antall blokker')
129 ylabel('Gjennomsnittlig forskjell i hver celle')
130 title('Utvikling av transmisjonstap')
131 
132 figure
133 plot(norm_error1)
134 xlabel('Antall blokker')
135 ylabel('Gjennomsnittlig forskjell i hver celle')
136 title('Utvikling av transmisjonstap')
```

B.1.7 Script for gjennomsnittlig hastighetsprofil

```
1  % initialiserer LYBIN
2  lb=actxserver('LybinCom.LybinModelComBin');
3  
4  % Initialiserer inputt-metoder
5  env = lb.invoke('IEnvironment');
6  mod = lb.invoke('IModelData');
7  sensor = lb.invoke('ISensor');
8  pulse = lb.invoke('IPuls');
9  platform = lb.invoke('IPlatform');
10 ocean = lb.invoke('IOcean');
11 
12 % Setter inn sensordata
13 sensor.Depth = 40;
14 sensor.TiltReceiver = 0;
15 sensor.TiltTransmitter = 0;
16 sensor.SideLobeReceiver = 43;
17 sensor.SideLobeTransmitter = 43;
18 sensor.DetectionThreshold = 13;
19 sensor.Frequency = 1000;
20 sensor.DirectivityIndex = 25;
21 sensor.SourceLevel = 210;
22 sensor.BeamWidthReceiver = 10;
23 sensor.BeamWidthTransmitter = 10;
24 
25 % Pulsegenskaper
26 pulse.Length = 1000;
27 pulse.FMBandWidth = 1000;
28 
29 % Modellegenskaper
```

```
R = 10000;
Z = 170;
R_cells = 100;
Z_cells = 34;
mod.SetRangeScaleAndRangeCells(R, R_cells);
mod.SetRangeScaleAndRangeSteps(R, R_cells*10);
mod.TRLRays = 1000;
mod.SetDepthScaleAndDepthCells(Z, Z_cells);
mod.SetDepthScaleAndDepthSteps(Z,Z_cells*10);
% Målstyrke
ocean.TargetStrength = 10;

% Laster inn lydhastighetstabell
load sound.mat;
M=sound.L_3;
n=size(M,1); % depth
m=size(M,2); % range

for i=1:m
    first_num = find(¬isnan(M(:,i)),1,'first');
    M(1:first_num-1,i)=M(first_num,i);
    last_num = find(¬isnan(M(:,i)),1,'last');
    M(last_num+1:n,i)=M(last_num,i);
end

for i=1:n
    first_num = find(¬isnan(M(i,:)),1,'first');
    M(i,1:first_num-1)=M(i,first_num);
    last_num = find(¬isnan(M(i,:)),1,'last');
    M(i,last_num+1:m)=M(i,last_num);
end

% lager gjennomsnittsvektor
snitt_M=zeros(1,Z);
for i=1:n
    snitt_M(i)=sum(M(i,:))/m;
end

depth=(1:1:Z);
% lager gjennomsnittsprofil
spdprfl_1=zeros(Z,4);
spdprfl_1(:,1)=depth;
spdprfl_1(:,2)=8;
spdprfl_1(:,3)=34;
spdprfl_1(:,4)=snitt_M(depth)

% Miljøparametre
env.WindSpeedMeasurment = [0,10000,0]; % Rekkeviddeavhengig vindhastighet. Første kolo
SetFirstSoundSpeedProfile(env,0,Z,spdprfl_1)
```

```
env.BottomType = [0,5000,1;5000,10000,1]; % Bunntype. Første kolonne er fra, an
prof = [0,170;10000,170;];
env.bottomProfile = prof; % Bunnprofil. første kolonne er rekkevidde. Andre kol
env.AddBottomLossFan(0,0,[0,0;10,0; 90, 0]); % Bunntapsverdier med tap (i dB) m
mod.UseMeasuredBottomLoss = 1; % Skrur på bunntapstabell.

% Kjører LYBIN
lb.DoCalculation

% Henter data
data.trl.forward = lb.GetResultsBin(0);
data.trl.backward = lb.GetResultsBin(1);
data.sig = lb.GetResultsBin(2);
data.pod = lb.GetResultsBin(3);
data.rev.Tot_rev = lb.GetResultsBin(4);
data.rev.Surf_rev = lb.GetResultsBin(5);
data.rev.Vol_rev = lb.GetResultsBin(6);
data.rev.Bot_rev = lb.GetResultsBin(7);
data.rev.Noise = lb.GetResultsBin(8);


% Kjører "vanlig" med samme inputt
s = lb.modelData; % Henter ut inputtdata i XML-format
fid = fopen('datasett3snitt.xml', 'w');
fprintf(fid, '%s', s); % Legger XML-data i XML-fil
% dos('lybin') % Kjører vanlig LYBIN med XML-data
```

B.2 Ray5-kode

B.2.1 Funksjonen *ray_init.m* for konstant hastighetsgradient

```
function [g od p plott] = ray_init
%Set up parameters.
%
% % Initial conditions
rs=0; %source range
zs=40; %source depth
zr=20; %receiver depth
rr=10000; %receiver range
theta_fan=deg2rad(-5:0.1:5); %take-off angles (ray-fan) in rad
rmax=10000; %mar range
bathy=[0 180;10000 180]; %bathymetry
surf=[0 0;10000 0]; %surface
%ode-solver
h=2; %Step-size (Intital)
```

```
atol=0; rtol=1.0E-5; %Error
%
Neqs=7; %# equations
%
%Parameter study
pno= 'no'; %parameter ('no', 'rtol', )
par=[1.E-3 1.E-4 1.E-6]; %parameter value
pcolor={'b','r','g','c','k'};
%
%Plotting
plott.a=0; %init. plot ssp as background
plott.b=2; %plot rays (1=points, 2=as continuous line)
plott.c=0; % plot receiver range crossings
plott.d=0; %plot integration points along rays (slow!!!)
%
%Store in structs
%..geometry, environment in struct g
g.rs=rs; g.zs=zs; g.zr=zr; g.rr=rr; g.rmax=rmax;
g.bz=bathy(:,2); g.br=bathy(:,1); %bottom depth
g.sz=surf(:,2); g.sr=surf(:,1); %surface roughness
g.theta_fan=theta_fan; %ray fan
%..ode params in struct od
od.h=h; od.atol=atol; od.rtol=rtol; od.Neqs=Neqs;
%..parameter study
if strcmp(pno,'no'), par=[1]; end
p.par=par; p.pcolor=pcolor; p.pno=pno;
%h.g=g; h.od=od; h.plott=plott;
return
```

B.2.2 Funksjonen *ssp.m* for konstant hastighetsgradient

```
function [c,cr,cz,crr,crz,czz] = ssp(r,z)
%Provide sound velocity (c) and its derivatives
%wrt. r (dcdr) and z (dcdz). SSP given analytically
%or as a table in which case interpolation necessary
type=9;
switch type
case 9
c = 1490 - 0.1*z; % hastighetsprofil med konstant gradient
cz = -0.1;
cr = 0;
czz = 0;
crz = 0;
crr = 0;
end
return
```

B.2.3 Funksjonen *ode.m*

```
function [x1,y1,h1] = ode(odeFcn,y,x,h,method)
%  Integrate a first order system of Ordinary Differential equations.
%
%    dy (x)/dx = f (x,y , . . ,y )  ,i=1,..,n    (*)
%      i          i    1        N
%
%  Given values for the variables y(1:n) known at x, use
%  Euler/Runge Kutta method to advance solution to x+h.
%  The function  dydx=F(x,y) which returns derivatives dydx at x
%  (right hand side of (*)) must be supplied.
% Call
%  [x1,y1] = ode(@F,y,x,h);
% where F is the function that evaluates the right hand side
%  function dydx = F(x,y)
% and h is stepsize
%  stepcontrol: 1=fixed, 2=adaptive
%
% Methods:
% ode1 - Euler, fixed stepsize
% ode2 - Runge-Kutta 2 with adaptive stepsize
if (nargin==5)
    [x1,y1] = ode1(odeFcn,y,x,h);
else
    [x1,y1,h1] = ode2(odeFcn,y,x,h);
end
return


function [x1,y1] = ode1(odeFcn,y,x,h)
%Euler
h=abs(h); %h<0 means force stepsize, used in rk2
f=feval(odeFcn,x,y);
y1=y+h*f;
x1=x+h;
return


function [x,y,h] = ode2(odeFcn,y,x,h,xend)
%-------------------------------------------------------------------
% Ivars rk2. Adaptiv Second-order Runge Kutta Method.
%
%  Integrate a first order system of Ordinary Differential equations.
%
%    dy (x)/dx = f (x,y , . . ,y )  ,i=1,..,n    (*)
%      i          i    1        N
```

```
46 %
47 %  Given values for the variables y(1:n) known at x, use
48 %  Runge Kutta method to advance solution over to x+h.
49 %  The subroutine  equa(x,y,dydx) which returns derivatives dydx at x
50 %  (right hand side of (*)) must be supplied.
51 %
52 %  Routine uses stepsize h given as input. (h must have an initial value
53 %  h = 0.5/(NX*NY is a good start)
54 %  If step accepted (error sufficiently small) stepsize is increased.
55 %  On return h contains suggestion for the new stepsize.
56 %  If step is not accepted, stepsize is reduced untill step accepted
57 %  or stepsize becomes too small, in which case the program stops.
58 %  Will not integrate longer than xend (h is adjusted to obtain this)
59 %
60 %  Check for negative stepsize which may occur when  a picture is written
61 %
62 %  Variables:
63 %     n - # equations
64 %     x,y - solution y(x) is known at x.
65 %     h - stepsize to be used. On output suggestion for size of next step
66 %         if h<0 use fixed stepsize abs(h), if h>0 adaptive stepsize
67 %     xend - end of integration interval
68 %     atol,rtol - error control
69 %     x, at input the current time, at output the time after the integration
70 % TODO
71 %  -xend må håndteres (nå bare fortsetter den med h=1.E-6). Gi flag ut.
72 %-------------------------------------------------------------------------
73 global od
74 %Initialize
75 if (nargin==4) %dont use xend (set large)
76     xend=1.0E30;
77 end
78 %atol=0;
79 %rtol=1E-6;
80 atol=od.atol; rtol=od.rtol;
81 %
82 uround=1.0E-10;
83 nstp=0;
84 nrej=0;
85 nf=0; %count function evaluations
86 isw=1;
87 %
88 %Fixed stepsize
89 if h<0 %force fixed stepsize
90     h=abs(h);
91     k1 = feval(odeFcn,x,y);
92     y2 = y+h*0.5*k1;
93     k2 = feval(odeFcn,x+0.5*h,y2);
94     %..advance solution
```

```
95      x = x+h;         %x-value after step
96      y = y + h*k2;    %y-value after step
97      return
98  end
99  %
100 %Adaptive stepsize
101 %
102 %Take one trial step with stepsize h
103 k1 = feval(odeFcn,x,y);
104 nf=nf+1;
105 y2 = y+h*0.5*k1;
106 k2 = feval(odeFcn,x+0.5*h,y2);
107 nf = nf+1;
108 %
109 %Estimate error
110 [err,hjust] = error_estimate(k1,k2,y,h,atol,rtol);
111 %
112 %Reduce stepsize untill error is below spec.
113 while err > 1.0
114     %Step rejected
115     %%disp(sprintf('step rejected: h,hjust,x %f %f %f',h,hjust,x))
116     if (hjust < 1.0e-10)
117         disp('stepsize too small');
118         pause;
119     end
120     %..reduce stepsize
121     h= hjust*0.9;
122     isw=0;
123     nrej=nrej+1;
124     %..recompute k2 with new stepsize
125     y2 = y+h*0.5*k1;
126     k2 = feval(odeFcn,x+0.5*h,y2);
127     nf = nf+1;
128     %..estimate error
129     [err,hjust] = error_estimate(k1,k2,y,h,atol,rtol);
130 end
131 %
132 %Now error is below spec. (or stepsize too small), accept step
133
134 %
135 %Step accepted
136 %%disp(sprintf('step accepted: h,x %f %f',h,x));
137 isw=1;
138 %..advance solution
139 x = x+h;         %x-value after step
140 y = y + h*k2;    %y-value after step
141 %
142 %..suggest new value of stepsize (h), but do not go beyond endpoint
143 if (abs(x-xend) > uround) %endpoint not reached yet
```

```
144     h = max(h,min(hjust*0.95,4.0*h)); %suggest new value of stepsize
145     %     Reduce h such that xend is reached exactly.
146     if (h ≥ (xend-x))
147         h = xend-x;
148     end
149     %     If this is the first step h may become negative, in that case
150     %     force h small positive
151     if (h < 0.0)
152         h = 1.0d-6;         %h should not be negative
153         %write(*,*) '** h was negative, set to 1.0d-6'
154     end
155     nstp = nstp+1;
156 else %x=xend, endpoint reached
157     disp('x=xend');
158     return
159 end
160 return
161
162
163 function [err,hjust] = error_estimate(k1,k2,y,h,atol,rtol);
164     %Estimate error. Error term is h(k1-k2)
165     %..Find max relative error h|k1-k2|/(atol+rtol*max|y|) and index
166     [err,ikk]=max(abs(k1-k2)/(atol+rtol*max([abs(y); 1.0e-10])));
167     err = err*h;
168     hjust = sqrt(1.0/max(1.0E-6,err))*h; %an estimate of stepsize
    based on the error?
169 return
```

B.2.4 Funksjonen *ray_plot.m*

```
1  function ray_plot(mode,varargin)
2  %
3  switch mode
4      case 'init'
5          [g,plott] = deal(varargin{:}); %deal input arguments
6          plot_init(g,plott);     %run function
7          %varargout = {ch};         %assign outputarguments
8
9      case 'ssp'
10         [ch] = deal(varargin{:}); %deal input arguments
11         ch = noise_setup(ch);     %run function
12         %varargout = {ch};         %assign outputarguments
13
14     case 'ray'
15         [ipar,event,theta_fan,ray1,ray2,plott,p,g] = deal(varargin{:});
16         plot_ray(ipar,event,theta_fan,ray1,ray2,plott,p,g);
17         %varargout = {y};
```

```
18
19      case 'rTheta'
20          [g,event,plott] = deal(varargin{:});
21          plot_rTheta(g,event,plott);
22          %varargout = {y};
23
24  end
25
26
27  function plot_init(g,plott)
28  %Plot geometry and possibly ssp as background
29  %
30  figure(5); clf;
31  xlim=[0,g.rmax]; ylim=[min(g.sz)-10,max(g.bz)+10];
32  a=axes('XLim',xlim,'YLim',ylim); hold on;%Set axis limits
33  %get(a); %list all axis properties
34  set(a,'Ydir','reverse');
35  cmin=1490; cmax=1530;
36  caxis([cmin cmax]);
37  %
38  %plot ssp as background
39  if (plott.a==1),
40      zv=linspace(ylim(1),ylim(2),100);
41      rv=linspace(xlim(1),xlim(2),200);
42      for ir=1:length(rv)
43          for iz=1:length(zv)
44              [c,dcdz,dcdr]=ssp(rv(ir),zv(iz));
45              C(iz,ir)=c;
46          end
47      end
48      imagesc(rv,zv,C);
49      colorbar; hold on
50  end
51  %colormap cool %
52  %
53  %Plot geometry
54  plot(g.br,g.bz,'k'); %bottom
55  plot(g.sr,g.sz,'k'); %surface
56  plot(xlim,[g.zr g.zr],'r')
57  %plot([50000 50000],ylim,'r')
58  return
59
60
61  function plot_ray(ipar,event,theta_fan,ray1,ray2,plott,p,g)
62  %
63  pcolor=p.pcolor;
64  rc1=event.rc1; rc2=event.rc2;
65  turn1=event.turn1; turn2=event.turn2;
66  %
```

```
for itheta=1:length(theta_fan)
    k=ray2(itheta,1);
    rv=ray1(itheta,1:k,1); zv=ray1(itheta,1:k,2);
    %plot(rv,zv,'.')
    if (plott.b==1), plot(rv,zv,'.b','MarkerSize',2); end
    if (plott.b==2), plot(rv,zv,pcolor{ipar}); end
    %plot Receiver depth crossing
    rc=rc1(itheta,1:rc2(itheta));
    if (plott.c),
        if ¬isempty(rc), plot(rc,g.zr,'ks','MarkerSize',5); end
    end
    %plot Turning Points
    rt=turn1(itheta,1:turn2(itheta),1);
    zt=turn1(itheta,1:turn2(itheta),2);
    %if ¬isempty(rt), plot(rt,zt,'g+'); end
end
return


function plot_ssp

figure(3); clf;
a2=axes('XLim',[1480 1500],'YLim',[0 320]);
title('Sound speed profile'); hold on;%Set axis limits
set(a2,'Ydir','reverse');
for z=0:1:300
    [c,dcdz,dcdr]=ssp(0,z);
    plot(c,z);
end
return


function plot_rTheta(g,event,plott)
figure(2); clf;
xlim=[0,g.rmax]; ylim=[-20,20];
a2=axes('XLim',xlim,'YLim',ylim);
title('r-theta'); hold on;%Set axis limits
rc1=event.rc1; rc2=event.rc2;

for itheta=1:length(g.theta_fan)
    rc=rc1(itheta,1:rc2(itheta));
    theta=rad2deg(g.theta_fan(itheta));
    if ¬isempty(rc), plot(rc,theta,'+'); end
end
return
```

B.2.5 Funksjonen *ray5.m*

```
%Direct integration raytrace
%Dynamic raytracing
%
%Integrates the following equations with the pathlength as the free %parameter,
%
% %Eikonal equations
% dy(1) = c*ksi; %dr/ds = c*ksi(s);
% dy(2) = c*eta; %dz/ds = c*eta(s);
% dy(3) = -dcdr/c^2; %dksi/ds = - (1/c^2)*dc/dr
% dy(4) = -dcdz/c^2; %deta/ds = - (1/c^2)*dc/dz
% %Travel time
% dy(5) = 1/c;
% %Dynamic ray tracing
% dy(6) = c*p; %dq/ds=c*p
% dy(7) = -(crr*eta^2 - 2*crz*ksi*eta + czz*ksi^2)*p;
% dp/ds= -(crr*eta^2 - 2*crz*ksi*eta + czz*ksi^2)*q
%
%where
% y(1) = r   (range)
% y(2) = z   (depth)
% y(3) = ksi = cos(theta)/c;
% y(4) = eta = sin(theta)/c;
% y(5) = tau  (travel time)
% x    = s    (pathlength)
%
%All parameters along a ray is stored in ray1(theta,k,p)
%where theta is takeoff angle, k is index of point and p
%is parameter no. 1-8;
% 1=r,2=z,3=ksi,4=eta,5=tau,6=q,7=p,8=s(pathlength)
%Receiver range crossings(range,pathlength,angles) are
%stored in rc1,rc2
%
%Todo
% - implementer bakgrunnsstrømning (adveksjon)
% - bruk av global er litt rotete, kanskje nødvendig ifm ode-solver

function ray5

global g od p plott ray1 ray2
clc
%%profile on
%
%Initialize
[g od p plott] = ray_init;
%
%Allocate storage
kpoints=800;
ray1=zeros(length(g.theta_fan),kpoints,od.Neqs+2);
ray2=zeros(length(g.theta_fan),3);
```

```
%
%Set up plot
ray_plot('init',g,plott)
%
%Loop through a parameter (parameterstudy)
for ipar=1:length(p.par)
    tic
    modify_parameter(ipar) %modify parameter for parameterstudy
    %
    %Loop through ray-fan
    for itheta=1:length(g.theta_fan)
        theta=g.theta_fan(itheta);
        disp(sprintf('theta= %7.2f deg',rad2deg(theta)));
        singleray(theta,g.rs,g.zs,g.rmax,od.h,itheta);
        %[event1] = singleray(theta,g.rs,g.zs,g.rmax,od.h,itheta); %For shooting
    end
    %
    %Find where (r) ray crosses receiver depth
    [event] = check_events(ray1,ray2,g.theta_fan);
    toc
    %
    %plot
    ray_plot('ray',ipar,event,g.theta_fan,ray1,ray2,plott,p,g);
end %ipar loop
%
%r-theta plot
%ray_plot('rTheta',g,event,plott)
%
%Print ray table
table(event,g.theta_fan)
%
%%profile off
%%profview
%
%Check that preallocated storage (ray1) as sufficient (else slow)
[npoints,nray] = max(ray2(:,1)); % max # integration points along ray
disp(sprintf('Max # integration points is %u along ray no.
%u at angle %6.2f deg',npoints,nray,rad2deg(g.theta_fan(nray))));

if (npoints > kpoints), disp(sprintf('***Warning: program slow
due to ray1 allocated too small, kpoints should be gt %u',npoints)); end
save('eventinfo','event')
return


function table(event,theta_fan)
%Print table of ray history
rc1=event.rc1; sc1=event.sc1; bc1=event.bc1;
rc2=event.rc2; sc2=event.sc2; bc2=event.bc2;
```

```
97  nTheta=length(rc2);
98  disp(' ')
99  disp('————————————————————————————————————————————————————')
100 disp('ray  angle  receiver   surface  bottom   turning')
101 disp('no.         crossings  hits     hits     points ')
102 disp('————————————————————————————————————————————————————')
103 for iTheta=1:nTheta
104     disp(sprintf('%3u %6.1f %6u %8u %8u',iTheta,
    rad2deg(theta_fan(iTheta)),rc2(iTheta),sc2(iTheta),bc2(iTheta)));
105 end
106 disp('————————————————————————————————————————————————————')
107 disp(' ');
108 for i=1:nTheta
109     nrc=rc2(i); nsc=sc2(i); nbc=bc2(i);
110     rr=rc1(i,1,1:nrc); rr=rr(:); %rc range
111     sr=sc1(i,1,1:nsc); sr=sr(:);
112     br=bc1(i,1,1:nbc); br=br(:);
113     disp(sprintf('ray %u ————————————————————————',i));
114     disp('rec. crossing ranges');
115     disp(rr')
116     disp('surface hits ranges');
117     disp(sr')
118     disp('bottom hits ranges');
119     disp(br')
120 end
121 return
122
123
124 function modify_parameter(ipar)
125 %Parameter study, modify paramemeter
126 global g od p plott
127 %
128 pno = p.pno; %parameter
129 pval= p.par(ipar); %parameter value
130 switch p.pno,
131     case 'no', disp('no parameter study');
132     case 'rtol', od.rtol=pval; %rtol
133
134     otherwise, error('Unknown parameter');
135 end;
136 disp(sprintf('Parameter: %s set to %g',pno,pval));
137 return
138
139
140 function F=score_ray(theta)
141 %For shooting. Compute 'miss' distance F for given ray angle.
142 global g od
143 [event1]=singleray(theta,g.rs,g.zs,g.rmax,od.h,1);
144 range=event1.rc1(event1.rc2);
```

```
145 F=range-g.rr;
146 disp([rad2deg(theta) range])
147 return
148
149
150 function [event1]=singleray(theta,rs,zs,rmax,h,itheta)
151 %Launch single ray. Handle interfaces.
152 %Interface hits are stored in ray1 (as param no.9, as a flag ihit
153 %indicating 0=no interface,1=surface,2=bottom)
154 %May return crossing with receiver depth.
155 global ray1 ray2 plott
156 %
157 %Initial conditions for ray itheta
158 c0=ssp(rs,zs); %Sound-speed at source
159 ksi=cos(theta)/c0;
160 eta=sin(theta)/c0;
161 q=0;
162 p=1/c0;
163 tau=0;
164 y0=[rs zs ksi eta tau q p]'; %Initial conditions
165 %
166 x=0; r=0; z=0; %x=s
167 k=0; %point no. along ray
168 y=y0;
169 while r<rmax
170     %
171     %advance solution one step from x (=s) to x+h=x1
172     [x1,y1,h1] = ode(@F,y,x,h);
173     %
174     %Check for events (interface, receiver depth, turning point, ..)
175     [x,y,ihit] = check_interfaces(x,y,x1,y1,itheta,h);
176     %record y for ray
177     r=y(1);
178     z=y(2);
179     k=k+1;
180     ray1(itheta,k,:)=[y; x; ihit]; %save all params. along ray
181     %Test, plot while integrationg along ray (slow!!!)
182     if (plott.d),
183         drawnow
184         plot(r,z,'r.','MarkerSize',2);
185     end
186     %Next step with suggested stepsize
187     h=h1;
188 end
189 %Record params for each ray (angle)
190 ray2(itheta,1)=k; %record #points stored for each angle
191 ray2(itheta,2)=x1; %Path length (NOTE. Ray not stopped exactly at a range
192 ray2(itheta,3)=y1(5); %Travel time
193 %Find where (r) ray crosses receiver depth (and all other events)
```

```
194 %for current ray. Compute only if asked for. Necessary for shooting.
195 if nargout==1, [event1] = check_events(ray1,ray2,theta); end
196 return
197
198
199 function [xi,yi,ihit] = check_interfaces(x,y,x1,y1,itheta,h)
200 %Check for interfaces (surface/bottom hits).
201 %Handle reflections at surface/bottom.
202 %Return initial conditions (xi,yi) for next step and flag ihit
203 %showing 0=no interface, 1=surface, 2=bottom.
204 %
205 %Extract parameters
206 global g
207 r1=y1(1); z1=y1(2);
208 [D]=interpL(r1,g.br,g.bz); %Bottom depth at current range (r1)
209 [S]=interpL(r1,g.sr,g.sz); %Surface height at current range (r1)
210 %..Surface reflection
211 if z1<S
212     %handle reflection
213     [xi,yi] = surface_reflection(x,y,x1,y1,h);
214     %record event
215     ihit=1;
216     %..Bottom reflection
217 elseif z1>D
218     %handle reflection
219     [xi,yi] = bottom_reflection(x,y,x1,y1,h);
220     %record event
221     ihit=2;
222 else %no boundary crossings
223     xi=x1; yi=y1;
224     ihit=0;
225 end
226 return
227
228
229 function [event] = check_events(ray1,ray2,theta_fan)
230 %Check for events (receiver depth crossings ..),
231 global g
232 %
233 %..Receiver depth crossing
234 [rc1,rc2]=RecCross(ray1,ray2,theta_fan,g.zr);
235 event.rc1=rc1; event.rc2=rc2;
236 %
237 %..Interface (surface/bottom) hits (recorded in ray1)
238 [sc1,sc2,bc1,bc2]=interfaceHits(ray1,ray2,theta_fan,g.zr);
239 event.sc1=sc1; event.sc2=sc2; event.bc1=bc1; event.bc2=bc2;
240 %
241 %..Turning point
242 [event.turn1,event.turn2]=RecTurn(ray1,ray2,theta_fan);
```

```
243 %
244 %..Ray Splitting Point
245 % if
246 %     ss
247 % end
248 %..Caustic
249 % if ..
250 %     ss
251 % end
252 %..Marker point
253 % if
254 %     s
255 % end
256 %..Ray Termination
257 % if
258 %     ss
259 % end
260 return
261
262 %
263 %RAY EQUATIONS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
264
265 function dy = F(x,y)
266 %Eikonal equation
267 %Get sound speed and its derivatives wrt. r and z
268 global od
269 global last %save last pars used by integrator
270 r=y(1); z=y(2); ksi=y(3); eta=y(4); tau=y(5); q=y(6); p=y(7);
271 [c,dcdr,dcdz,crr,crz,czz] = ssp(r,z);
272 last.c=c; last.r=r; last.z=z; %save last c,r,z used by integrator
273 dy=zeros(od.Neqs,1); %column vector%
274 %   1 2  3   4   5  6 7
275 %y=[r z ksi eta tau q p]
276 %
277 %Eikonal equations
278 dy(1) = c*ksi; %dr/ds = c*ksi(s);
279 dy(2) = c*eta; %dz/ds = c*eta(s);
280 dy(3) = -dcdr/c^2; %dksi/ds = - (1/c^2)*dc/dr
281 dy(4) = -dcdz/c^2; %deta/ds = - (1/c^2)*dc/dz
282 %Travel time
283 dy(5) = 1/c;
284 %Dynamic ray tracing
285 dy(6) = c*p; %dq/ds=c*p
286 dy(7) = -(crr*eta^2 - 2*crz*ksi*eta + czz*ksi^2)*p; %dp/ds=
     -(crr*eta^2 - 2*crz*ksi*eta + czz*ksi^2)*q
287 return
288
289 %
290 %BOUNDARIES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
291
292
293 function [xr,yr] = surface_reflection(x,y,x1,y1,h)
294 %Find surface by shooting. Adjust stepsize h to land exactly on surface.
295 %Restart raytrace from surface with correct take-off angle
296 global xc yc g last
297 xc=x; yc=y;
298 %
299 %Find h1 to land exactly on bottom by shooting
300 h1 = fzero(@score_s,[0 h]);
301 %
302 %Take step to land exactly on surface
303 [x1,y1] = ode(@F,y,x,-h1); %force stepsize to h1
304 %
305 %Restart raytrace from surface with correct take-off angle
306 r1=y1(1); z1=y1(2); ksi1=y1(3); eta1=y1(4); tau1=y1(5);
    q1=y1(6); p1=y1(7);
307 % %Use this to handle rough surace
308 c = ssp(r1,z1);%sound speeed at surface
309 % %c = last.c; %sound speed at last eval. of f
310 %..angle of ray
311 slope_ray=eta1/ksi1; %slope of ray at bottom = tan(Theta)=eta/ksi
312 Theta_ray=atan(slope_ray); %Angle of ray
313 % %Theta_ray=acos(c*ksi1); %Angle of ray
314 %..angle of surface
315 [D,slope_sur,Theta_sur,rbk,zbk]=interpL(r1,g.sr,g.sz); %depth at
    current range (r1)
316 %..Reflected angle
317 Theta_refl=2*Theta_sur-Theta_ray; %Reflected angle
318 ksi2=cos(Theta_refl)/c;
319 eta2=sin(Theta_refl)/c;
320 tau2=tau1; %Should be a correction
321 yr=[r1 z1 ksi2 eta2 tau2 q1 p1]'; xr=x1;%Initial conditions for
    reflected ray
322 return
323
324
325 function F1 = score_s(h1)
326 %Integrate equations from x,y using stepsize h1
327 global xc yc g
328 [x1,y1] = ode(@F,yc,xc,-h1); %force stepsize to h1
329 %Discrepancy
330 r1=y1(1); z1=y1(2);
331 [D]=interpL(r1,g.sr,g.sz); %Surface at range r1
332 %D=0; %Surface height at r1
333 F1=z1-D; %discrepancy
334 return
335
336
```

```
337 function [xr,yr] = bottom_reflection(x,y,x1,y1,h)
338 %Find bottom by shooting. Adjust stepsize h to land exactly on bottom.
339 %Restart raytrace from surface with correct take-off angle
340 global xc yc g last
341 xc=x; yc=y;
342 %
343 %Find h1 to land exactly on bottom by shooting
344 h1 = fzero(@score_b,[0 h]);
345 %
346 %Take step
347 [x1,y1] = ode(@F,y,x,-h1); %force stepsize to h1
348 %
349 %Restart raytrace from bottom with correct take-off angle
350 r1=y1(1); z1=y1(2); ksi1=y1(3); eta1=y1(4); tau1=y1(5); q1=y1(6); p1=y1(7);
351 c = ssp(r1,z1);%sound speeed at bottom
352 %c = last.c; %sound speed at last eval. of f
353 %..angle of ray
354 slope_ray=eta1/ksi1; %slope of ray at bottom = tan(Theta)=eta/ksi
355 Theta_ray=atan(slope_ray); %Angle of ray
356 %Theta_ray=acos(c*ksi1); %Angle of ray
357 %..angle of bottom
358 [D,slope_bot,Theta_bot,rbk,zbk]=interpL(r1,g.br,g.bz); %depth at
    current range (r1)
359 %..Reflected angle
360 Theta_refl=2*Theta_bot-Theta_ray; %Reflected angle
361 ksi2=cos(Theta_refl)/c;
362 eta2=sin(Theta_refl)/c;
363 %Reflection of p and q not impl. yet
364 yr=[r1 z1 ksi2 eta2 tau1 q1 p1]'; xr=x1;%Initial conditions for
    reflected ray
365 return
366
367
368 function F1 = score_b(h1)
369 %Integrate equations from x,y using stepsize h1
370 global xc yc g
371 [x1,y1] = ode(@F,yc,xc,-h1); %force stepsize to h1
372 %Discrepancy
373 r1=y1(1); z1=y1(2);
374 [D]=interpL(r1,g.br,g.bz); %Bottom Depth at range r1
375 F1=z1-D; %discrepancy
376 return
377
378 %YMSE%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
379
380 function [y0,dyk,Theta_bot,xk,yk] = interpB(x0,x,y)
381 % Find indices of subintervals, x(k) ≤ u < x(k+1),
382 % or u < x(1) or u ≥ x(m-1).
383 m=length(x);
```

```
%as in interp1 (
[ignore,k] = histc(x0,x);
k(x0<x(1) | ¬isfinite(x0)) = 1; %if x0<x(1), k=1
k(x0≥x(m)) = m−1;               %if x0≥x(m), k=m−1
%k = max(find(x≤x0))
%Coord. for point with index k
xk=x(k);
yk=y(k);
%Linear interpolation
dyk=(y(k+1)−y(k))/(x(k+1)−x(k)); %slope of bottom (tan(Theta)(from k to k+1)
Theta_bot=atan(dyk); %bottom slope angle
dy = dyk*(x0−x(k));
y0 = y(k)+dy;
return


function [y0,dyk,Theta_bot,xk,yk] = interpL(x0,x,y)
%Find bottom depth/surface height y(x0) at current range x0,
%given a table of bottom depths/surface heights y(k) vs range x(k).
%(Find y(x0) given y(x(k)) by linear interpolation.)
%
% Find indices of subintervals, x(k) ≤ u < x(k+1),
% or u < x(1) or u ≥ x(m−1).
m=length(x);
%as in interp1 (
[ignore,k] = histc(x0,x);
k(x0<x(1) | ¬isfinite(x0)) = 1; %if x0<x(1), k=1
k(x0≥x(m)) = m−1;               %if x0≥x(m), k=m−1
%k = max(find(x≤x0))
%Coord. for point with index k
xk=x(k);
yk=y(k);
%Linear interpolation
dyk=(y(k+1)−y(k))/(x(k+1)−x(k)); %slope of bottom
(tan(Theta)(from k to k+1)
Theta_bot=atan(dyk); %bottom slope angle
dy = dyk*(x0−x(k));
y0 = y(k)+dy;
return


function [x,y] = BotCross(x1,y1,a1,x2,y2,a2)
%Determine coordinates (x,y) where ray intersect bottom.
% x1,y1,a1: coord. of bottom at index k, and slope from k to k+1
% x2,y2,a2: coord. of ray at index k, and slope from k to k+1
% a1= bottom slope =tan(Theta1)
% a2= ray slope
x=(1/(a1−a2))*(a1*x1−y1−a2*x2+y2);
y=y1+a1*(x−x1);
```

```
return


function [turn1,turn2]=RecTurn(ray1,ray2,theta_fan)
%Determine coordinates for Turning points
% when drdz=0 (changes sign), drdz=ksi/eta
turn1=zeros(length(theta_fan),20,2);
turn2=zeros(length(theta_fan),1);
for itheta=1:length(theta_fan)
    %Find index before Turning point
    k=ray2(itheta); %#points on ray for angle itheta
    drdz1=ray1(itheta,1:k,3)./ray1(itheta,1:k,4); %drdz=ksi/eta
    %locate sign change by left shift 1, multiply and test for negative
    drdz2=[drdz1(2:k) drdz1(k)]; %same, left shifted 1
    a=drdz1.*drdz2; %
    ai=find(a<0);
    ar=ray1(itheta,ai,1);
    az=ray1(itheta,ai,2);
    %plot(ar,az,'*')
    %Locate exact by interpolation (cubic spline or Hermite)
    if ¬isempty(ai)
        turn1(itheta,1:length(ai),1)=ar;
        turn1(itheta,1:length(ai),2)=az;
        turn2(itheta)=length(ai);
    end
end
return


function [rc1,rc2] = RecCross(ray1,ray2,theta_fan,zr)
%Determine coordinates (x,y) where ray intersect receiver depth.
%Receiver crossings in: rc1(theta,i,:),
%  i=1: ranges,
%  i=2: depths (not nec.)
%  i=3: pathlengths
%  i=4: angles (rad)
rc1=zeros(length(theta_fan),4,20);
rc2=zeros(length(theta_fan),1);
for itheta=1:length(theta_fan)
    %Find index before crossing
    k=ray2(itheta);
    a1=ray1(itheta,1:k,2)-zr;
    a2=[a1(2:k) a1(k)];
    a=a1.*a2;
    ai=find(a<0);
    ar=ray1(itheta,ai,1);
    az=ray1(itheta,ai,2);
    %plot(ar,az,'o')
```

```
481     %Locate crossing(s) by linear interpolation
482     if ¬isempty(ai)
483         z0=zr; %receiver depth
484         %Range of crossing
485         r1=ray1(itheta,ai,1);   %r1 (before crossing)
486         r2=ray1(itheta,ai+1,1); %r2 (after crossing)
487         z1=ray1(itheta,ai,2);   %z1
488         z2=ray1(itheta,ai+1,2); %z2
489         r0=r1+((z0-z1)./(z2-z1)).*(r2-r1);
490         rc1(itheta,1,1:length(r0))=r0; %range(s) for crossing(s)
491         rc1(itheta,2,1:length(r0))=zr; %depth  for crossing(s)
492         rc2(itheta)=length(r0); %# crossings
493         %Pathlength of crossing
494         s1=ray1(itheta,ai,8);   %s to point before crossing (r1,z1)
495         ds=sqrt((r0-r1).^2+(z0-z1).^2); %ds to crossing (r0,z0)
496         s0=s1+ds;
497         rc1(itheta,3,1:length(s0))=s0; %range(s) for crossing(s)
498         %Angle of crossings
499         %ksi1=ray1(itheta,ai,3); %before crossing
500         %eta1=ray1(itheta,ai,4);
501         %Theta_cross=atan(eta1/ksi1); %
502         slopes=(z2-z1)./(r2-r1);
503         Theta_cross=atan(slopes);
504         rc1(itheta,4,1:length(slopes))=Theta_cross; %range(s)
    for crossing(s)
505     end
506     %plot(r0,z0,'ro')
507 end
508 return
509
510 function [sc1,sc2,bc1,bc2]=interfaceHits(ray1,ray2,theta_fan,zr);
511 %Interface hits
512 sc1=zeros(length(theta_fan),4,20);
513 sc2=zeros(length(theta_fan),1);
514 bc1=zeros(length(theta_fan),4,20);
515 bc2=zeros(length(theta_fan),1);
516 %
517 for itheta=1:length(theta_fan)
518     idx_s = find(ray1(itheta,:,9)==1); %Index for surface hits
519     idx_b = find(ray1(itheta,:,9)==2); %Index for bottom hits
520     if ¬isempty(idx_s)
521         ns=length(idx_s); %# surf. hits
522         rs=ray1(itheta,idx_s,1); %r
523         zs=ray1(itheta,idx_s,2); %z
524         ss=ray1(itheta,idx_s,8); %s
525         ksis=ray1(itheta,idx_s,3);
526         etas=ray1(itheta,idx_s,4);
527         as=atan(etas./ksis); %angle
528         %plot(rs,zs,'*');
```

```
529         sc1(itheta,:,1:ns)=[rs; zs; ss; as];
530         sc2(itheta)=ns; %# crossings
531     end
532     if ¬isempty(idx_b)
533         nb=length(idx_b); %# surf. hits
534         rb=ray1(itheta,idx_b,1);
535         zb=ray1(itheta,idx_b,2);
536         sb=ray1(itheta,idx_b,8); %s
537         ksib=ray1(itheta,idx_b,3);
538         etab=ray1(itheta,idx_b,4);
539         ab=atan(etab./ksib); %angle
540         %plot(rb,zb,'*');
541         bc1(itheta,:,1:nb)=[rb; zb; sb; ab];
542         bc2(itheta)=nb; %# crossings
543     end
544
545 end
546 return
```

B.2.6 Funksjonen *interpolation.m*

```
1  function [sound_speed,cr_grid,cz_grid,crr_grid,crz_grid,
   czz_grid] = interpolation(M)
2
3  % definition of size
4  n=size(M,1); % depth
5  m=size(M,2); % range
6
7  sound_speed = M;
8
9  % To find the outputs, I will assume that a partial derivate
10 % is like a slope in a particular direction.
11 % First of all, we should calcul the derivates for the datapoints
12 % of the file sound.mat.
13
14 % We have to be careful with the fact that in the file sound.mat
15 % the rows represent the depth and the columns represent the
16 % range. Usually this is the contrary!
17
18 % calculation of cr=dc/dr
19 cr_grid = zeros(n,m); % same size than the matrix M
20 for i=2:(n-1)
21     for j=2:(m-1)
22         cr_grid(i,j) = (M(i,(j+1))-M(i,(j-1)))/20;
23     end
24 end
25
26 for j=2:(m-1)
```

```
    cr_grid(1,j) = (M(1,(j+1))-M(1,(j-1)))/20;
    cr_grid(n,j) = (M(n,(j+1))-M(n,(j-1)))/20;
end

for i=2:(n-1)
    cr_grid(i,1) = (M(i,2)-M(i,1))/10;
    cr_grid(i,m) = (M(i,m)-M(i,(m-1)))/10;
end

cr_grid(1,1) = (M(1,2)-M(1,1))/10;
cr_grid(1,m) = (M(1,m)-M(1,(m-1)))/10;
cr_grid(n,1) = (M(n,2)-M(n,1))/10;
cr_grid(n,m) = (M(n,m)-M(n,(m-1)))/10;

for i=1:n
    for j=1:m
        if isnan(cr_grid(i,j))==1
            cr_grid(i,j)=0;
        end
    end
end

% calculation of cz=dc/dz
cz_grid = zeros(n,m); % same size than the matrix M
for i=2:(n-1)
    for j=2:(m-1)
        cz_grid(i,j) = (M((i+1),j)-M((i-1),j))/2;
    end
end

for j=2:(m-1)
    cz_grid(1,j) = M(2,j)-M(1,j);
    cz_grid(n,j) = M(n,j)-M((n-1),j);
end

for i=2:(n-1)
    cz_grid(i,1) = (M((i+1),1)-M((i-1),1))/2;
    cz_grid(i,m) = (M((i+1),m)-M((i-1),m))/2;
end

cz_grid(1,1) = M(2,1)-M(1,1);
cz_grid(1,m) = M(2,m)-M(1,m);
cz_grid(n,1) = M(n,1)-M((n-1),1);
cz_grid(n,m) = M(n,m)-M((n-1),m);

for i=1:n
    for j=1:m
        if isnan(cz_grid(i,j))==1
            cz_grid(i,j)=0;
```

```
        end
    end
end

% calculation of crr=d2c/dr2
crr_grid = zeros(n,m); % same size than the matrix M

for i=2:(n-1)
    for j=2:(m-1)
        crr_grid(i,j) = (cr_grid(i,(j+1))-cr_grid(i,(j-1)))/20;
    end
end

for j=2:(m-1)
    crr_grid(1,j) = (cr_grid(1,(j+1))-cr_grid(1,(j-1)))/20;
    crr_grid(n,j) = (cr_grid(n,(j+1))-cr_grid(n,(j-1)))/20;
end

for i=2:(n-1)
    crr_grid(i,1) = (cr_grid(i,2)-cr_grid(i,1))/10;
    crr_grid(i,m) = (cr_grid(i,m)-cr_grid(i,(m-1)))/10;
end

crr_grid(1,1) = (cr_grid(1,2)-cr_grid(1,1))/10;
crr_grid(1,m) = (cr_grid(1,m)-cr_grid(1,(m-1)))/10;
crr_grid(n,1) = (cr_grid(n,2)-cr_grid(n,1))/10;
crr_grid(n,m) = (cr_grid(n,m)-cr_grid(n,(m-1)))/10;

for i=1:n
    for j=1:m
        if isnan(crr_grid(i,j))==1
            crr_grid(i,j)=0;
        end
    end
end

% calculation of crz=d2c/drdz
crz_grid = zeros(n,m); % same size than the matrix M
for i=2:(n-1)
    for j=2:(m-1)
       crz_grid(i,j) = (cr_grid((i+1),j)-cr_grid((i-1),j))/2;
    end
end

for j=2:(m-1)
    crz_grid(1,j) = cr_grid(2,j)-cr_grid(1,j);
    crz_grid(n,j) = cr_grid(n,j)-cr_grid((n-1),j);
end

```

```
125 for i=2:(n-1)
126     crz_grid(i,1) = (cr_grid((i+1),1)-cr_grid((i-1),1))/2;
127     crz_grid(i,m) = (cr_grid((i+1),m)-cr_grid((i-1),m))/2;
128 end
129
130 crz_grid(1,1) = cr_grid(2,1)-cr_grid(1,1);
131 crz_grid(1,m) = cr_grid(2,m)-cr_grid(1,m);
132 crz_grid(n,1) = cr_grid(n,1)-cr_grid((n-1),1);
133 crz_grid(n,m) = cr_grid(n,m)-cr_grid((n-1),m);
134
135 for i=1:n
136     for j=1:m
137         if isnan(crz_grid(i,j))==1
138             crz_grid(i,j)=0;
139         end
140     end
141 end
142
143 % calculation of czz=d2c/dz2
144 czz_grid = zeros(n,m); % same size than the matrix M
145 for i=2:(n-1)
146     for j=2:(m-1)
147         czz_grid(i,j) = (cz_grid((i+1),j)-cz_grid((i-1),j))/2;
148     end
149 end
150
151 for j=2:(m-1)
152     czz_grid(1,j) = cz_grid(2,j)-cz_grid(1,j);
153     czz_grid(n,j) = cz_grid(n,j)-cz_grid((n-1),j);
154 end
155
156 for i=2:(n-1)
157     czz_grid(i,1) = (cz_grid((i+1),1)-cz_grid((i-1),1))/2;
158     czz_grid(i,m) = (cz_grid((i+1),m)-cz_grid((i-1),m))/2;
159 end
160
161 czz_grid(1,1) = cz_grid(2,1)-cz_grid(1,1);
162 czz_grid(1,m) = cz_grid(2,m)-cz_grid(1,m);
163 czz_grid(n,1) = cz_grid(n,1)-cz_grid((n-1),1);
164 czz_grid(n,m) = cz_grid(n,m)-cz_grid((n-1),m);
165
166 for i=1:n
167     for j=1:m
168         if isnan(czz_grid(i,j))==1
169             czz_grid(i,j)=0;
170         end
171     end
172 end
```

B.2.7 Datasettversjonen av *ray5.m* (kun den endrede delen)

```
function ray5

global g od p plott ray1 ray2 MYRANGE MYDEPTH sound_speed cr_grid cz_grid
crr_grid crz_grid czz_grid

        %laster inn lydhastighetsmatrisen fra datasettet
        load sound.mat;
        M_grid=sound.L_3;
        n=size(M_grid,1); % depth
        m=size(M_grid,2); % range
        % Fyller matrisen der den ikke har verdier
        for i=1:m
            first_num = find(¬isnan(M_grid(:,i)),1,'first');
            M_grid(1:first_num-1,i)=M_grid(first_num,i);
            last_num = find(¬isnan(M_grid(:,i)),1,'last');
            M_grid(last_num+1:n,i)=M_grid(last_num,i);
        end

        for i=1:n
            first_num = find(¬isnan(M_grid(i,:)),1,'first');
            M_grid(i,1:first_num-1)=M_grid(i,first_num);
            last_num = find(¬isnan(M_grid(i,:)),1,'last');
            M_grid(i,last_num+1:m)=M_grid(i,last_num);
        end

        range_max = 10*size(M_grid,2);
        depth_max = size(M_grid,1);

        myrange = 10:10:range_max;
        mydepth = 1:1:depth_max;

        %Setter verdier for den virkelige størrelsen til
        %beregningsområdet slik at interpoleringen i ssp.m
        %skalerer rett og gir ut verdier for rett avstand og
        %dybde
        [MYRANGE,MYDEPTH] = meshgrid(myrange,mydepth);
        %deriverer og lagrer matrisene med de deriverte og
        %lydhastigheten i globale variabler som ssp får
        %tilgang til
        [sound_speed,cr_grid,cz_grid,crr_grid,crz_grid,
czz_grid] =  interpolation(M_grid);

clc
%%profile on
```

```
%
%Initialize
[g od p plott] = ray_init;
%
%Allocate storage
kpoints=800;
ray1=zeros(length(g.theta_fan),kpoints,od.Neqs+2);
ray2=zeros(length(g.theta_fan),3);
%
%Set up plot
ray_plot('init',g,plott)
.
.
.
```

B.2.8 Datasettversjon av funksjonen *ssp.m*

```
function [c,cr,cz,crr,crz,czz] = ssp(r,z)
global MYRANGE MYDEPTH sound_speed cr_grid cz_grid
crr_grid crz_grid czz_grid

%Provide sound velocity (c) and its derivatives wrt.
% r (dcdr) and z (dcdz)
%SSP given analytically or as a table in which case
%interpolation necessary

type=9;
switch type
    case 9 % profil for sound.mat file
        r
        z
        %Gir ut topp og bunnverdier i matrisen om verdier
        %utenfor matrisen blir etterspurt
        if z<1
            z=1;
        elseif z>186
            z=186
        end

        %interpolerer ut verdier fra matrisene
        c = interp2(MYRANGE,MYDEPTH,sound_speed,r,z,'cubic');
        cz = interp2(MYRANGE,MYDEPTH,cz_grid,r,z,'cubic');
        cr = interp2(MYRANGE,MYDEPTH,cr_grid,r,z,'cubic');
        czz = interp2(MYRANGE,MYDEPTH,czz_grid,r,z,'cubic');
        crz = interp2(MYRANGE,MYDEPTH,crz_grid,r,z,'cubic');
        crr = interp2(MYRANGE,MYDEPTH,crr_grid,r,z,'cubic');
end
return
```

B.2.9 Funksjonen *ray_ init.m* for hastighetsprofil 5

```
function [g od p plott] = ray_init
%Set up parameters.

% % Initial conditions for the sound.mat file
rs=10000; %source range
zs=40; %source depth
zr=20; %receiver depth
rr=20000; %receiver range
theta_fan=deg2rad(-5:0.1:5); %take-off angles (ray-fan) in rad
rmax=20000; %mar range
bathy=[0 170;2968 170]; %bathymetry
surf=[0 0;2968 0]; %surface

%ode-solver
h=2; %Step-size (Intital)
atol=0; rtol=1.0E-5; %Error
%
Neqs=7; %# equations
%
%Parameter study
pno= 'no'; %parameter ('no', 'rtol', )
par=[1.E-3 1.E-4 1.E-6]; %parameter value
pcolor={'b','r','g','c','k'};
%
%Plotting
plott.a=0; %init. plot ssp as background
plott.b=2; %plot rays (1=points, 2=as continuous line)
plott.c=0; % plot receiver range crossings
plott.d=0; %plot integration points along rays (slow!!!)
%
%Store in structs
%..geometry, environment in struct g
g.rs=rs; g.zs=zs; g.zr=zr; g.rr=rr; g.rmax=rmax;
g.bz=bathy(:,2); g.br=bathy(:,1); %bottom depth
g.sz=surf(:,2); g.sr=surf(:,1); %surface roughness
g.theta_fan=theta_fan; %ray fan
%..ode params in struct od
od.h=h; od.atol=atol; od.rtol=rtol; od.Neqs=Neqs;
%..parameter study
if strcmp(pno,'no'), par=[1]; end
p.par=par; p.pcolor=pcolor; p.pno=pno;
%h.g=g; h.od=od; h.plott=plott;
return
```

B.2.10 Funksjonen *plot_rayrange4.m*

```
% function plot_rayrange4(path)
path=100
global g od p plott ray1 ray2

%Initialize
[g od p plott] = ray_init;

figure(6);
% hold on;
load 'eventinfo.mat'
event.rc1;

% for each ray, we need #times when the ray crosses the receiver depth
% for each ray, we know how many times the ray crosses the receiver depth
% and where it cross (range)
% the function RecCross gives us the range for each crossing

% we should
% - divide the space [0,rmax]in several spaces
% - find the number of crossings in each space
% - create 1 vector containing the #crossings per space

rmax = 20000;
number_spaces = floor(rmax/path);
ray = zeros(length(g.theta_fan),20);
N = zeros(number_spaces,1);

for irange = 1:(number_spaces-1)
    n_irange = 0;
    for i = 1:20 %itererer 20 ganger per stråler per space
        ray(:,i) = event.rc1(:,1,i);   % lagrer i-te raycrossing
                                       % for hver stråle
        [number,column,v_i] = find(ray(:,i));
        v_i;
        if isempty (v_i)==0
          for j = 1:length(v_i)
              if ((irange-1)*path <= v_i(j)) && (v_i(j) < irange*path)
                  n_irange = n_irange+1; %teller hvor mange
                                         %kryssinger som er
                                         %funnet i dette spacet
              end
          end
        end
    end
    N(irange,1)= n_irange; %lagrer kryssinger for aktuelt space i
```

```
                                %vektoren N
end

x_plot=(.1:.1:20);
stairs(x_plot,N)
xlabel('Avstand [km]')
ylabel('Antall stråler som krysser mottakerdybden')
title('RAY5: Hastighetsprofil 3. my=1 zs=40m, zr=20m, theta=-5:0.1:5')

return
```

B.2.11 Funksjonen *plot_rayrange5.m*

```
plot_rayrange5(path)
path=100
global g od p plott ray1 ray2

%Initialize
[g od p plott] = ray_init;

figure;
% hold on;
load 'eventinfo.mat'
event.rc1;

% for each ray, we need #times when the ray crosses the receiver depth
% for each ray, we know how many times the ray crosses the receiver depth
% and where it cross (range)
% the function RecCross gives us the range for each crossing

% we should
% - divide the space [0,rmax]in several spaces
% - find the number of crossings in each space
% - create 1 vector containing the #crossings per space

rmax = 20000;
number_spaces = floor(rmax/path);
ray = zeros(length(g.theta_fan),20);
N = zeros(number_spaces,1);

for irange = 1:(number_spaces-1)
    n_irange = 0;
    for i = 1:20 %itererer 20 ganger per stråler per space
        ray(:,i) = event.rc1(:,1,i);   % lagrer i-te raycrossing
                                       % for hver stråle
        [number,column,v_i] = find(ray(:,i));
        v_i;
```

```
        if isempty (v_i)==0
          for j = 1:length(v_i)
              if ((irange-1)*path ≤ v_i(j)) && (v_i(j) < irange*path)
                  ds=event.rc1(j,3,i); %henter ut gangveien til
                                       %funnet stråle
                  if ds ≠ 0
                  n_irange = n_irange+(1/(ds^2)); %teller hvor mange
                  end                             %kryssinger som er
                                                  %funnet i dette spacet
              end
          end
        end
    end
    N(irange,1)= n_irange; %lagrer kryssinger for aktuelt space i
                           %vektoren N
end

x_plot=(.1:.1:20);
stairs(x_plot,20*log10(N/max(N)))
xlabel('Avstand [km]')
ylabel('Transmisjonstap')
title('RAY5: Hastighetsprofil 3 - geom. spr. zs=40m, zr=20m, theta=-5:0.1:5')

return
```

B.2.12 Funksjonen *smooth.m*

```
function glatt_M=smooth(M_grid,K)

% Har flyttet over utfyllingen av tomme felter i datasettet
% fra ray5.m til denne funksjonen. ray5.m ser ryddigere ut da.
n=size(M_grid,1) % depth
m=size(M_grid,2) % range
for i=1:m
    first_num = find(¬isnan(M_grid(:,i)),1,'first');
    M_grid(1:first_num-1,i)=M_grid(first_num,i);
    last_num = find(¬isnan(M_grid(:,i)),1,'last');
    M_grid(last_num+1:n,i)=M_grid(last_num,i);
end

for i=1:n
    first_num = find(¬isnan(M_grid(i,:)),1,'first');
    M_grid(i,1:first_num-1)=M_grid(i,first_num);
    last_num = find(¬isnan(M_grid(i,:)),1,'last');
    M_grid(i,last_num+1:m)=M_grid(i,last_num);
end

% Glatting med Kagawa's metode
```

```
22 M=M_grid;
23
24 dim=size(M)
25 z_dim=dim(1)
26 x_dim=dim(2)
27 my=1
28
29 glatt_M=zeros(z_dim,x_dim);
30
31 for k=1:K
32     %glatter indre del av mattrisen
33     for i=2:z_dim-1
34         for j=2:x_dim-1
35             glatt_M(i,j)=(M(i,j)+exp(-my)*(M(i,j-1)+M(i,j+1)+M(i-1,j)
   +M(i+1,j))+exp(-2*my)*(M(i-1,j-1)+M(i-1,j+1)+M(i+1,j-1)+M(i+1,j+1)))
   /(1+4*exp(-my)+4*exp(-2*my));
36         end
37     end
38
39     %glatter sidekanter
40     %hjørner først. disse har tre nabopunkter med verdier
41     glatt_M(1,1)=(M(1,1)+exp(-my)*((2*M(1,2))+(2*M(2,1)))
   +exp(-2*my)*(4*M(2,2)))/(1+4*exp(-my)+4*exp(-2*my));
42     glatt_M(1,x_dim)=(M(1,x_dim)+exp(-my)*((2*M(1,x_dim-1))
   +(2*M(2,x_dim)))+exp(-2*my)*(4*M(2,x_dim-1)))/(1+4*exp(-my)+4*exp(-2*my));
43     glatt_M(z_dim,1)=(M(z_dim,1)+exp(-my)*((2*M(z_dim-1,1))+
   (2*M(z_dim,2)))+exp(-2*my)*(4*M(z_dim-1,2)))/(1+4*exp(-my)+4*exp(-2*my));
44     glatt_M(z_dim,x_dim)=(M(z_dim,x_dim)+exp(-my)*((2*M(z_dim,x_dim-1))+
   (2*M(z_dim-1,x_dim)))+exp(-2*my)*(4*M(z_dim-1,x_dim-1)))
   /(1+4*exp(-my)+4*exp(-2*my));
45
46     %sidekanter foruten hjørner
47     %øvre
48     for j=2:x_dim-1
49         glatt_M(1,j)=(M(1,j)+exp(-my)*(M(1,j-1)+M(1,j+1)+2*M(2,j))+
   exp(-2*my)*(2*M(2,j-1)+2*M(2,j+1)))/(1+4*exp(-my)+4*exp(-2*my));
50     end
51     %nedre
52     for j=2:x_dim-1
53         glatt_M(z_dim,j)=(M(z_dim,j)+exp(-my)*(M(z_dim,j-1)+M(z_dim,j+1)
   +2*M(z_dim-1,j))+exp(-2*my)*(2*M(z_dim-1,j-1)+2*M(z_dim-1,j+1)))
   /(1+4*exp(-my)+4*exp(-2*my));
54     end
55     %venstre
56     for i=2:z_dim-1
57         glatt_M(i,1)=(M(i,1)+exp(-my)*(M(i-1,1)+M(i+1,1)+2*M(i,2))
   +exp(-2*my)*(2*M(i-1,2)+2*M(i+1,2)))/(1+4*exp(-my)+4*exp(-2*my));
58     end
59     %Høyre
```

```
60      for i=2:z_dim-1
61          glatt_M(i,x_dim)=(M(i,x_dim)+exp(-my)*(M(i-1,x_dim)+M(i+1,x_dim)
    +2*M(i,x_dim-1))+exp(-2*my)*(2*M(i-1,x_dim-1)+2*M(i+1,x_dim-1)))
    /(1+4*exp(-my)+4*exp(-2*my));
62      end
63
64      M=glatt_M;
65      k
66  end
67  return
```

B.2.13 Datasettversjonen av *ray5.m* med mulighet for glatting (kun den endrede delen)

```
1   function ray5
2
3   global g od p plott ray1 ray2 MYRANGE MYDEPTH sound_speed cr_grid cz_grid
    crr_grid crz_grid czz_grid
4
5           %laster inn lydhastighetsmatrisen fra datasettet
6           load sound.mat;
7           M_grid=sound.L_3;
8
9                   %Utvider matrisen og glatter den et visst antall ganger med Kag
10                  glatt_M=smooth(M_grid,512);
11
12          range_max = 10*size(M_grid,2);
13          depth_max = size(M_grid,1);
14
15          myrange = 10:10:range_max;
16          mydepth = 1:1:depth_max;
17
18          %Setter verdier for den virkelige størrelsen til
19          %beregningsområdet slik at interpoleringen i ssp.m
20          %skalerer rett og gir ut verdier for rett avstand og
21          %dybde
22          [MYRANGE,MYDEPTH] = meshgrid(myrange,mydepth);
23          %deriverer og lagrer matrisene med de deriverte og
24          %lydhastigheten i globale variabler som ssp får
25          %tilgang til
26          [sound_speed,cr_grid,cz_grid,crr_grid,crz_grid,
    czz_grid] =  interpolation(M_grid);
27
28  clc
29  %%profile on
30  %
31  %Initialize
```

```
[g od p plott] = ray_init;
%
%Allocate storage
kpoints=800;
ray1=zeros(length(g.theta_fan),kpoints,od.Neqs+2);
ray2=zeros(length(g.theta_fan),3);
%
%Set up plot
ray_plot('init',g,plott)
.
.
.
```

B.2.14 Script for konvergensplott for glattemetoden

```

runs=4096 %antall glattinger
load sound.mat
M=sound.L_3;

M1=M;
error=zeros(1,runs);

for k=1:runs
    % glatter profilen 1 gang
    M=smooth(M,1)
    % regner ut differanse mot uglattet matrise
    diff2=abs(M1-M);
    colsum=sum(diff2);
    error(1,k)=sum(colsum)/(n*m);
    k=k+1
end
error

figure
plot(error)
```

B.2.15 Funksjonen *ssp.m* for sammenligning med Oases

```
function [c,cr,cz,crr,crz,czz] = ssp(r,z)

%Provide sound velocity (c) and its derivatives wrt. r (dcdr) and z (dcdz)
%SSP given analytically or as a table in which case interpolation necessary

type=9;
switch type
```

```
8       case 9 % profil for sound.mat file
9           %
10          % Grunnen til at vi ønsker denne spesielle (n^2) formen på
11          % lydhastighetsprofilen er at referansemodellen har en eksakt
12          % løsning i dette tilfellet, dvs. ikke noe tøys ved at
13          % lydhastighetsprofilen skal interpoleres.
14          % Analytisk løsning eksisterer for to lydhastighetsprofiler;
15          % konstant lydhastighet og n^2 lineær, dvs at brytningsindeksen^2
16          % varierer lineært med dypet, altså
17          % n^2=(c0/c)^2=a+b*z. Lydhastigheten er altså gitt ved:
18          % c(z)=c0/(sqrt(1+b*z)). b bestemmer gradienten. Det kan være praktisk
19          % å bestemme b fra verdien av lydhastigheten ved topp (c(z=0)=c0)
20          % og bunn (c(z=D)=c1). Da får vi at b=((c0/c1)^2-1)/D
21          %
22          % Trond
23
24          c0=1500;            % Lydhastighet ved overflaten
25          c1=1460;            % Lydhastighet ved bunnen
26          D=200;              % Total dybde
27          b=((c0/c1)^2-1)/D;
28
29          c = c0/(sqrt(1+b*z));
30          cz = -0.5*b*c0*((1+b*z)^(-3/2));
31          cr = 0;
32          czz = 0.75*(b^2)*c0*((1+b*z)^(-5/2));
33          crz = 0;
34          crr = 0;
35  end
36  return
```