Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Matematikk og fysikk	Vårsemesteret, 2022 Åpen
Forfatter: Sivert Hopland	x Sivert Hopland Sivert Hopland Forfatter
Veiledere: Intern: Eva Rauls Ekstern: Svein Haugen	
Tittel på bacheloroppgaven: Beamforming anvendt på tauet antenne Engelsk tittel: Beamforming applied to towed antenna	
Studiepoeng: 20	
Emneord: • Hydroakustikk • Tauet antenne • Digital signalbehandling • Beamforming	Sidetall: 44 + vedlegg/annet: 21 Bergen, 14/05-2022

Innhold

1.0 Motivasjon og definering av problemstilling

I denne oppgaven ønsket jeg å fordype meg innenfor digital signalbehandling, ettersom dette er et område med utrolig mange bruksområder. Det viktige for meg gjennom hele oppgaven er å holde en rød tråd fra de matematiske byggesteinene til den ferdige modellen man kommer til å bruke for å trekke slutninger.

For å gjøre dette kommer jeg først til å gjøre kort rede for signal-prosesseringens historie, og hva slags prosjekt det er jeg skal jobbe med og skrive om. Deretter kommer jeg til å forklare sentrale fysiske og matematiske prinsipper som spiller inn i denne oppgaven, og videre følger en detaljert avhandling om selve signalbehandlingen som blir utført i forbindelse med prosjektet. Til slutt kommer jeg til å presentere resultatene som oppnås, og drøfte gyldigheten og brukbarheten til disse ut fra et praktisk perspektiv.

Når det kommer til selve problemstillingen i oppgaven, kommer den til å dreie seg om en veldig spesifikk del av signalbehandlingen, noe som heter beamforming. Min oppgave kommer hovedsakelig til å være utviklingen av, og implementeringen av en beamformer som skal bidra til å peke ut retningen de ulike signalsignaturene kommer fra i forhold til en antenne. Helt konkret blir da problemstillingen min:

«Kode og implementere en enkel beamformer til bruk av en ny generasjon tauet antenne.»

1.1 Introduksjon og historie

Digital signalprosessering (DSP), er en sentral del av dagens teknologiske verden, og finnes i mange kjente databehandlingsapplikasjoner som RADAR, SONAR, telekommunikasjon, digital bildebehandling og mange andre. I kjernen dreier DSP seg om all behandling av data fra et signal blir plukket opp, til vi får et ferdig produkt. Dette produktet vil være en digitalisert versjon av det opprinnelige signalet. Dermed åpner det for muligheten til å analysere signalene med en datamaskin som gjør videre behandling mye lettere å gjennomføre. Sentrale teknikker innen DSP innebærer oppfanging, filtrering, manipulering, lagring og reprodusering av signaler. (IEEE Signal Processing Soceiety, 1998)

Røttene til DSP kan spores tilbake til 1700-tallet, da matematikere begynte å utvikle algoritmer som gjorde det lettere å analysere kompliserte systemer. Dette innebar framveksten av den reelle og komplekse analysen, samt numeriske metoder som gjorde det mulig å genere bedre og bedre tilnærminger til modeller som ikke har eksakte løsninger. (Stillwell)

Moderne signalprosessering begynte for fullt i 1948 med grunnleggelsen av IEEE Signal Processing Society. Opprinnelig bestod gruppen av radio ingeniører og signalprosesseringens-teknikkene som var i bruk, var få og langt imellom. Dette skulle imidlertid endre seg fortløpende da framveksten av moderne datamaskiner begynte på slutten av 50-tallet. Etter hvert gikk teknikkene fra å være et begrenset antall analoge innretninger, til å bli fullstendig dominert av digitale algoritmer. Digitalisering åpnet opp øynene til vitenskapsmenn og ingeniører fra en rekke fagfelt om nytten DSP kunne ha. Derfor fulgte 1960 og 1970 årene med seg mange nye teknikker og forbedringer som følge av den samlede tankekraften til mange aktører fra forskjellige felt. (IEEE Signal Processing Soceiety, 1998)

Teknikkene som brukes i de ulike feltene, er i bunn og grunn veldig like, men skiller seg fra hverandre med tanke på hvilke antakelser man kan gjøre for å forenkle utregningene. I tillegg er det også store forskjeller mellom anvendelsene med tanke på hvordan tilfeldig støy og andre signalforurensende kilder påvirker signalbehandlingen. Derfor snevrer vi inn diskusjonen til ett fagfelt, nemlig hydroakustikk. Nærmere bestemt havner dette prosjektet innenfor feltet SONAR, som er en forkortelse for Sound Navigation And Ranging. SONAR er en fellesbetegnelse på innretninger laget for å detektere og sende ut lydsignaler i vann.

2.0 Om prosjektet

Som nevnt i problemstillingen dreier prosjektet seg om å fremstille en ny generasjon av en såkalt tauet antenne. Bedriften jeg jobber for, som skal bygge antennen, heter Naxys. Dette er en liten bedrift som spesialiserer seg innenfor akustiske systemer til bruk under vann, deriblant innenfor oljesektor, miljøvern og lokasjons anvendelser. Dette prosjektet faller inn under sistnevnte, og hensikten er å kunne peke ut retningen de ulike lydsignaturene kommer fra i forhold til antennen. Dersom omstendighetene tillater det kan det også være mulig å identifisere hvilken kilde som kan ha laget signalet. Målet for ferdigstilling av antennen er høsten 2022, og tiden fram til da blir brukt til ferdigstilling og testing av komponenter.

SONAR feltet deles ofte inn i to deler, den ene kalles aktiv SONAR, hvor innretningen genererer lydsignaler og måler tiden det tar før den reflekterte lyden igjen treffer systemet. Det er dette som brukes for eksempel i ekkolodd på fritidsbåter, for å finne ut hvor dypt det er, eller for å finne ut om det er fisk i området under båten. Den andre kalles passiv SONAR, hvor innretningen kun skal lytte etter lydsignaler i vannet. Antennen vi jobber med, er altså et eksempel på passiv SONAR ettersom den kun skal plukke opp lydsignaler fra vannet. Den store fordelen med passiv SONAR er at i enkelte situasjoner er det hensiktsmessig at det du ønsker å lytte til, ikke vet at du er der. Et godt eksempel på dette finner vi i biologiverdenen. Det har seg nemlig slik at hvaler som kommuniserer med hverandre på ultralyd, gjerne skifter frekvens dersom de hører lyd som kommer fra en aktiv SONAR, så dersom biologene ønsker å analysere hvalsangen, så må de lytte passivt.

2.1 Utstyr

Selve antennen består hovedsakelig av tre deler. Den første delen heter lead-in, og er festet i båten. I denne ligger det strømkabler og optiske kabler hvor dataene antennen plukker opp, sendes til fartøyet. Neste del er noe som kalles en VIM, som er en forkortelse for vibrasjonsisolasjonsmodul. Hensikten med en VIM er at den minimerer forurensende støy fra bevegelsene og maskineriet til fartøyet som skal trekke antennen, et kritisk steg for å sikre best mulig utgangspunkt for innsamling og behandling av data. For å oppnå dette, er VIM'en laget av et elastisk materiale som overfører veldig lite mekanisk energi til antennen. Til sammen utgjør VIM'en og lead-in det som heter tauekabelen i illustrasjonen nedenfor, som til sammen har oppgaven å gi tilstrekkelig avstand, dybde og demping for at antennen skal fungere optimalt. Sist, men

ikke minst, har vi selve antennen som skal ta opp og sende videre lydsignaler den fanger opp fra vannet.

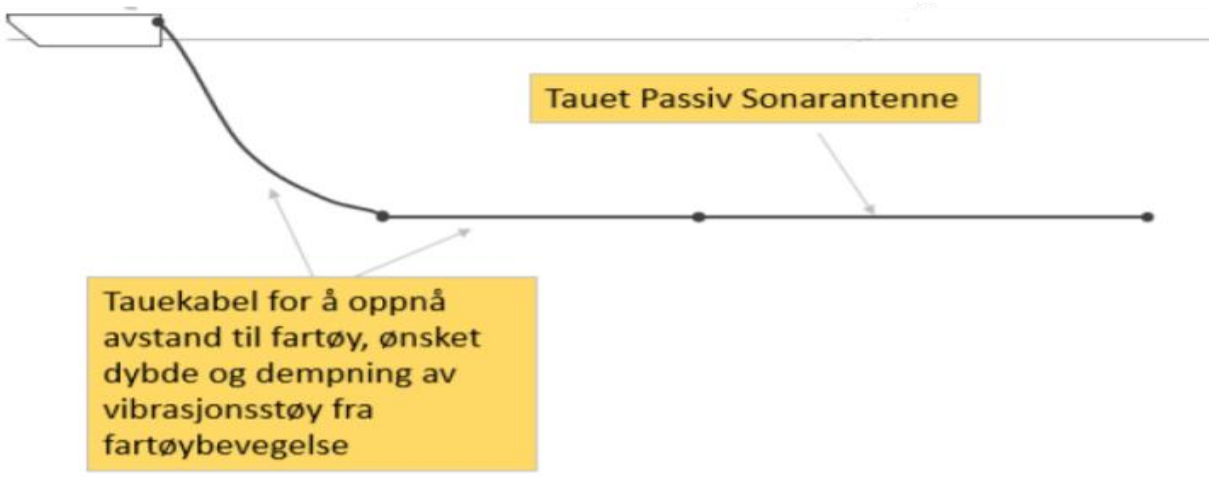

Fig 2.0: Antenne illustrasjon

Det vi skal produsere, er teknisk sett bare antenneseksjoner hvor en eller flere seksjoner kan settes sammen til en fullverdig slepeantenne. Hver seksjon er delt inn i ni deler med en sentral ADC-node i hver, hvor hver node digitaliserer data fra åtte hydrofoner. Sammen med hver hydrofon hører det også til en preamp som skal forsterke signalet fra hydrofonen, slik at det blir mer støytolerant og kraftig nok for videre bearbeiding. ADC står for «Analog to Digital Converter», og det er disse som gjør dataene om til et diskret format som kan behandles videre med datamaskin. Det blir mer om dette i seksjon 4.1 av oppgaven. Hver antenneseksjon inneholder også tre NAS-noder som plasseres bak, foran og i midten. NAS står for «Non Acoustic Sensors» og disse nodene inneholder kompass, trykksensor og temperatursensor. Dette er nyttig for å bestemme dybden og orienteringen til antennen, samt for å kunne bestemme lydhastigheten i vannet nøyaktig, noe som er viktig for best mulige resultater når vi skal analysere dataene. Dataene fra NAS-nodene og ADC-nodene flettes sammen og sendes videre til en master-node som sitter på starten av lead-in kabelen. Dette er «hjernen» til antennen hvor dataene lagres på en harddisk, og det er her dataene hentes ut for å analyseres.

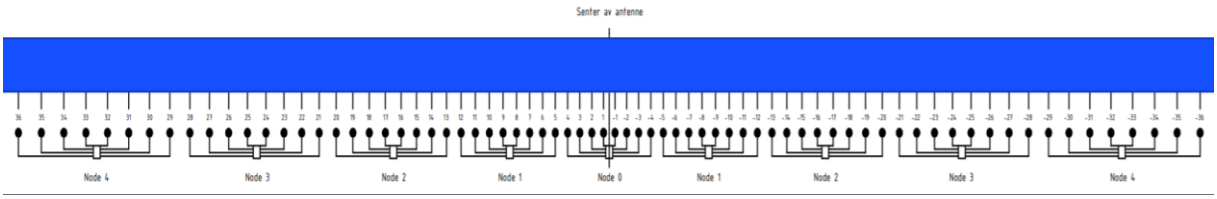

Fig 2.1: Seksjonsinndeling

Strukturen i antenneseksjonene er sånn at det er et sentralt tau hvor alt av elektronikk og hydrofoner ligger langs, mens det hele er omsluttet av et plasthylster. For å minimere bevegelse og friksjon gjennom vannet, gjøres diameteren til antennen så liten som mulig, og forskjellige utgaver lages med et tverrsnitt på 30 eller 50 mm. Som nevnt skal antennen slepes på en gitt dybde, men det er ikke så lett når ledningene og hydrofonene har høyere tetthet enn vann, og vil derfor nødvendigvis synke? Dette problemet løses ved å fylle hulrommet i slangen med olje, og siden olje har lavere tetthet enn vannet, så kan vi fylle slangen med ønsket mengde slik at den holder den dybden i vannet som er ønskelig. Dette kalles elektronikk i olje teknologi. Alle de elektriske komponentene utenom ledningene ligger inni 3D-printede kapsler for å ikke bli ødelagte.

Dette antennedesignet er første generasjon av digital antenne selskapet produserer. Antennen er digital i den forstand at signalene digitaliseres i nærheten av hydrofonene. Dermed må det analoge signalet fra hver hydrofon kun bevege seg den korte avstanden til nærmeste ADC-node, og derfra sendes de digitale signalene gjennom samme kabel til master-noden. Denne strukturen minsker antall ledninger det må være plass til i antennen, som gjør at den kan gjøres tynnere enn tidligere generasjoner, samtidig som vekten blir mindre. Summen av disse fordelene er at lengre antenner kan realiseres enn tidligere; en positiv forbedring jeg kommer tilbake til verdien av i seksjon 5.1. Det at antennen er delt inn i deler, er også fordelaktig fordi elektronikken som er tilstede i antennen, blir mer håndterbar. Dette minsker faren for at ledninger blir ødelagte, og om noe først er galt, er det lettere å lokalisere feilen.

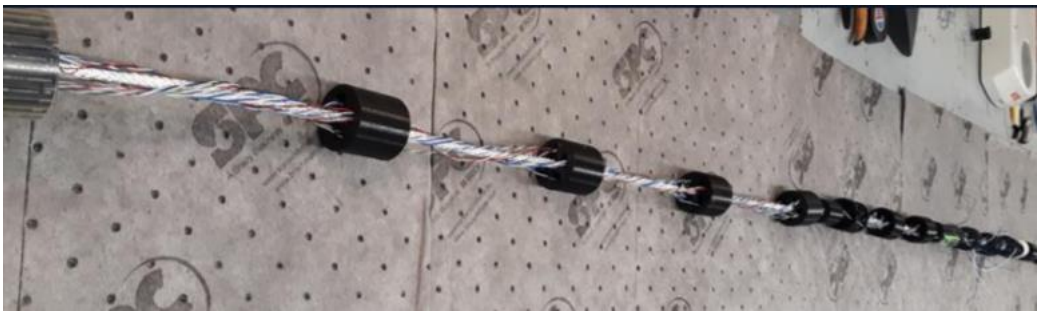

Fig 2.2: Antennens bestanddeler

Hydrofonene som skal brukes, er også nylig utviklet. De består hovedsakelig av en sylinder laget av piezoelektrisk keramikk, som måler lydtrykket i vannet. Forskjellen på disse hydrofonene og tidligere varianter er at de er mye mindre, noe som er viktig for at antennen skal ha et så lite tverrsnitt som mulig. Piezoelektrisitet er en egenskap noen materialer besitter som gjør at det danner seg en elektrisk ladning når materialet blir deformert. Denne ladningen fører til at det oppstår et inhomogent elektrisk felt som resulterer i en spenning. For hydrofonene er det det akustiske trykket i vannet som deformerer keramikken, og genererer et kontinuerlig elektrisk signal som sendes videre til ADC-noden. Sensitiviteten til hydrofonene gis som oftest i desibels, ifølge relasjonen:

1) $G = 20 \log_{10}(M_v)$

Hvor G [dB] er sensitiviteten, og M_v [V/Pa] er hydrofonens lineære spenningssensitivitet, definert som endringen i utgangsspenningen pr micropascal endring i lydtrykket til vesken rundt hydrofonen. (National Physics Laboratory, 2018) Signalet hydrofonene sender videre, er omtrent en sann kopi av det originale signalet ned til sensitiviteten til hydrofonen.

3.0 Nyttig fysikk og matematikk

Signalene som skal plukkes opp av antennen, er akustiske. Lyd som brer seg i vann, er veldig lik som motparten i luft, og i begge medier kan lyd beskrives som bølger som brer seg gjennom mediene med en gitt hastighet. Den største forskjellen er at lyden brer seg mye fortere i vann, nemlig mellom 1450 og 1570 m/s, altså omtrent 4.4 ganger raskere enn i luft. Dette skyldes at partiklene i vann, er mye tettere enn i luft, og har derfor lettere for å kollidere med hverandre, utveksle mekanisk energi og dermed sende lydbølgen videre. Den store rekkevidden farten kan ha, skyldes flere faktorer. Desto høyere saltnivået er, desto fortere vil lyden bre seg. Lyden går også ca. 4.5 m/s fortere per 1°C temperaturen øker ved konstant trykk, og lyden beveger seg også ca. 1.7 m/s fortere per 10 atm trykkøkning. (Duxbury) Alle disse faktorene måles av antennen for at vi mer nøyaktig kan bestemme lydhastigheten, og på bakgrunn av den kunne trekke konklusjoner om retningen og bevegelsen til en lydkilde basert på signalene vi fanger opp.

Det matematiske maskineriet man som regel treffer på når man har med bølger å gjøre, er en sentral differensiallikning som heter bølgelikningen:

2) $\nabla^2 p = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}$

I bølgelikningen er p(**r**,t) lydtrykket som funksjon av posisjonsvektoren **r** og tiden t. I formelen ser vi også c som symboliserer lydens hastighet i mediet vi studerer, altså vann. (Torre, 2014) Bølgelikningen representerer en lineær, homogen andre grads partiell differensiallikning. Den er lineær fordi den har egenskapen at summen av to lineært uavhengige løsninger også er en løsning. Homogeniteten kommer av at alle leddene som inneholder p eller deriverte av p kan flyttes til en side uten noen konstantledd eller funksjoner til overs.

Når man diskuterer bølger i fysikk sammenheng, er det to begrep som er verdt å merke seg. «Near field» og «far field», som referer til områder hvor lydbølgenes karakteristikk endrer seg nok til at vi kan behandle dem ulikt i matematisk forstand. En lydbølge som brer seg uten hindringer fra en kilde, er formet som en kule. Hver bølgefront er altså krummet, men når avstanden er stor nok, kan vi anta at frontene er planbølger. Grensen som er typisk å sette, er ved ≈1 bølgelengde. (Siemens Digital Industries Software, 2020) Hvis vi ser på typiske bølgelengder for lyd, finner vi blant annet at typisk menneskelig hørselsspenn, altså fra 20 – 20000 Hz, tilsvarer bølgelengder fra 17 m – 17 mm. Desto lavere frekvens, desto lengre blir bølgelengden.

For antennen vår som skal opereres i havet, kan vi altså trygt anta at vi er utenfor en bølgelengdes rekkevidde for de aller fleste lydkilder. Planbølger er faktisk en løsning av bølgelikningen, og attpåtil en av løsningene med lettest form:

3) p(**r**,t) = A$e^{i(\omega t - \boldsymbol{kr})}$

I formelen er A amplituden av signalet og ω er vinkelfrekvensen. Symbolet **k** er en vektor kalt bølgetallet, som peker langs retningen lydbølgene brer seg i. Absoluttverdien til **k** blir ofte bare kalt k, og oppfyller følgende likheter, hvor λ er bølgelengden:

4) $k = \frac{2\pi}{\lambda} = \frac{\omega}{c}$

For å gjøre analysen lettere så velger vi å legge koordinatsystemet slik at antennen ligger langs x-aksen med origo i den ene enden. Da kommer x > 0 for alle punktene på antennen, og i tillegg oppfyller punktene på antennen **r** = [x,0,0] og $|\mathbf{r}|$ = x. Hvis vi ser på eksponenten i formel 3, finner vi skalarproduktet **kr**, og om vi benytter definisjonen av skalarproduktet, ser vi at vi kan erstatte dette med $|k|*|x|*\cos\Phi$, hvor Φ representerer vinkelen mellom antennen og bevegelsesretningen til lydsignalet. Da ender vi opp med følgende uttrykk:

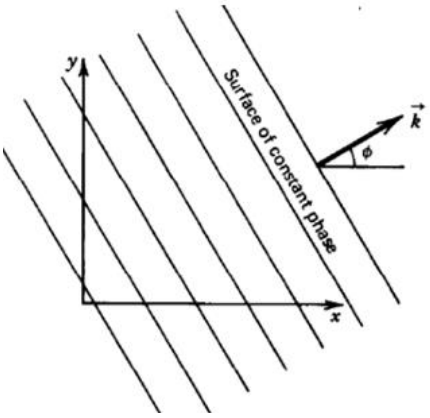

Fig 3.0: Koordinatsystemet

5) p(x,t) = A$e^{i(\omega t - kx\cos\Phi)}$

Det siste vi trenger av matematikk, er litt teori om Fourier-analyse, et veldig sentralt konsept innenfor signalprosesseringens verden. Den sentrale ideen er at alle periodiske funksjoner kan skrives som en uendelig sum av sinus og cosinus funksjoner. Et ofte brukt triks er å bruke Euler's formel, gjengitt i formel 6, for å skrive om rekken til en uendelig rekke av komplekse eksponentialfunksjoner. Da ender vi opp med et utrykk med som i formel 7:

6) $e^{i\Phi} = \cos\Phi + i\sin\Phi$

7) $x(t) = \sum_{k=-\infty}^{\infty} c_k e^{i\omega k t}$

Koeffisientene c_k er da komplekse tall. Disse koeffisientene inneholder nyttig informasjon om et ukjent signal, og det er derfor disse man ønsker å finne fram til gjennom en Fourier analyse. Fremgangsmåten for å finne koeffisientene er en såkalt Fourier transformasjon:

8) $c_k = X(\omega) = \frac{1}{P}\int_0^P x(t)e^{-i\omega kt}\,dt$

Her er P den typiske perioden for signalet. Formel 8 kan tolkes som at vi vil vite hvor korrelert x(t) er med de ulike eksponentialfunksjonene med ulik vinkelfrekvens, og desto mer korrelert disse er, desto høyere verdi vil X(ω) ha.

Fourier analysen jeg gjorde rede for i forrige avsnitt, gjelder kun for kontinuerlige signaler, men når man jobber med DSP, har man bruk for diskret Fourier analyse. Som nevnt i seksjon 2.1 skal nemlig ADC nodene diskretisere det signalet som kommer fra hydrofonene. Resultatet er at signalet blir omgjort til et diskret-tid signal. I tillegg til denne diskretiseringen måles også lydtrykket bare på utvalgte steder langs antennen, nemlig der hydrofonene er. Derfor er dataene også diskrete i rommet. Disse formene for diskretisering er i praksis veldig like, og jeg gjør derfor kun rede for hvordan Fourier analysen gjøres i et diskret tidsdomene, og vil forklare nærmere hvordan dette gjøres for romdimensjonen der dette er relevant.

Enkelt fortalt blir hydrofondataene diskretisert ved at de måles ved gitte mellomrom, kalt sampling. Hvor mange ganger signalet måles i løpet av et sekund, kalles for samplingfrekvensen, ofte forkortet F_S, og tiden mellom to påfølgende målinger kalles for T. Nå vil sammenhengen mellom det kontinuerlige og diskret signalet være beskrevet av: t = nT, hvor n representerer det n'te samplet, og det er vanlig å skrive dette skiftet som x(t) → x[n].

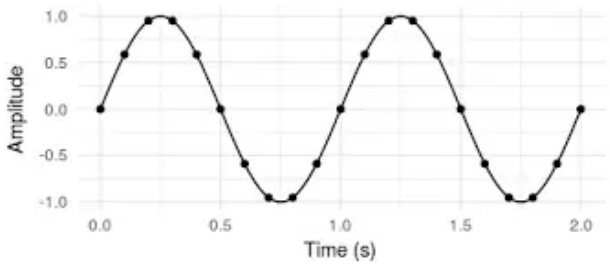

Fig 3.1: Sinussignal med Fs = 10 Hz

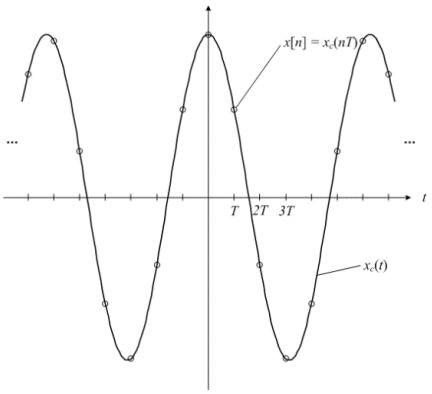

Fig 3.2: Sammenhengen mellom x(t) og x[n]

Hvis vi nå foretar dette skiftet av variabler ved å bytte ut t med nT i formel 8, samt anta at x(t) ≈ x[n] og dt ≈ T, som fører til at P = NT. Formel 4 kan også omskrives for å vise at $\omega = 2\pi f = \frac{2\pi}{P} = \frac{2\pi}{NT}$. Basert på dette kan utlede den diskrete Fourier transformasjonen fra formel 8:

9) $c_k = \frac{1}{N}\sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi kn}{N}} = \frac{1}{N}\,\mathrm{DFT}(x[n]) = \frac{X[k]}{N}$

En siste nødvendig del fra Fourier-analysen er den inverse Fourier transformasjonen. Den trenger vi for å komme tilbake igjen til det originale signalet vi hadde før vi gikk igjennom Fourier-analysen, og ser sånn ut:

10) $x[n] = \mathrm{IDFT}(X[k]) = \frac{1}{N}\sum_{k=0}^{N-1} X[k] e^{i\frac{2\pi kn}{N}}$

Avslutningsvis vil jeg forsøke og enkelt forklare hva det egentlig er vi oppnår gjennom denne typen analyse. Først og fremst er Fourier transformasjoner mer generelle enn Fourier rekker, og kan anvendes på ikke periodiske funksjoner. Vi bruker Fourier analyse i tidsdomenet som et eksempel og starter da med et vilkårlig signal. Signalet vil være en kombinasjon av sinus signaler med ulike frekvenser. Hvis vi nå utfører en Fourier transformasjon av signalet får vi en ny funksjon, enkelt kalt signalets Fourier transformasjonen, som er en funksjon av vinkelfrekvensen. Hvis vi nå plotter Fourier transformasjonen mot vinkelfrekvens, kommer grafen til å få lokale maksimumer hvor toppene er lokalisert presist ved frekvensene til delsignalene av det opprinnelige signalet. Dette kalles populært for frekvens domenet til signalet, eller invers domenet som det også kalles generelt. Man kan nemlig på samme måte veksle mellom andre domener, for eksempel posisjon, og deres invers domener, for å finne fram til bestanddelene til et signal. Den store fordelen med Fourier transformasjoner er imidlertid at mange problemer faktisk lar seg løse lettere i inversdomenet, og når vi finner løsningen kan vi utføre en invers transformasjon for å få løsningen i det det opprinnelige domenet. Det er dette prinsippet beamformeren baserer seg på, som vi kommer tilbake til i seksjon 4.2.

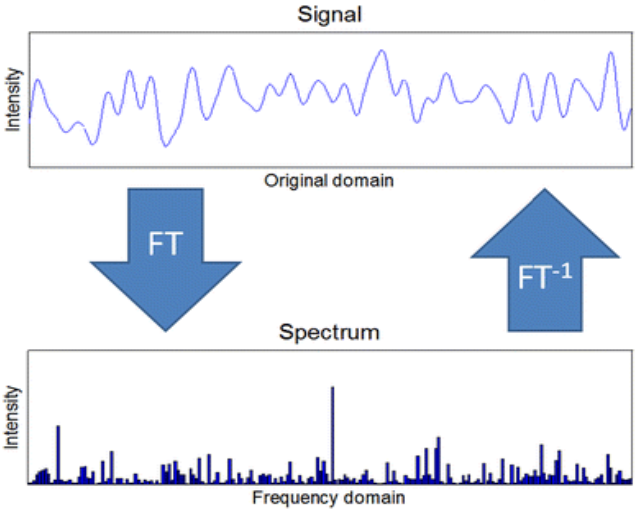

Fig 3.3: Eksempel på Fourier transformasjon

4.0 Strukturen til signalbehandlingen

Så kommer vi til hjertet av prosjektet, nemlig signalbehandlingen. Det har seg nemlig slik at utstyret har sine begrensninger uansett hvor bra det er laget, og da er det godt at vi har DSP. Det bringer oss et steg nærmere en god forståelse av hva slags signal det er vi har med å gjøre. I denne seksjonen skal jeg gå gjennom de tekniske detaljene bak teknikkene som skal anvendes i prosjektet, og hvorfor de fungerer. Denne delen av prosjektet er i bunn og grunn todelt. Den første delen handler om å samle inn og raffinere rådataene, mens den andre går ut på å analysere dataene. Begge disse delene er noe en ser i stort sett alle former for moderne signalbehandling, og de skal derfor få hver sin seksjon. I tillegg blir det en seksjon til som omhandler dataene som skal analyseres i forbindelse med oppgaven.

For å utføre digital signalbehandling, trenger man et passende verktøy. For denne oppgaven har jeg valgt å benytte kodespråket Python. Python er et brukervennlig programmeringsspråk med mange innebygde funksjoner og moduler. Derfor er Python brukt til alt fra å lese dataene fra antennen, plotte disse dataene, analysere dataene og selvsagt beamformingen. De ulike delene av koden kommer til å bli beskrevet der det er relevant, og hele skriptet kan finnes i Appendix A.

4.1 Sampling og filtrering

Først av alt må vi snakke om hvordan signalet går fra trykkbølger i vannet til digitale strukturer som vi kan jobbe med. Som nevnt må signalet digitaliseres. Dette gjøres ved at det analoge signalet samples. Da er det naturlig å tro at vi mister litt informasjon, men det viser seg faktisk at sampling ikke er så ille som det høres ut. Dette er beskrevet i et sentralt teorem innenfor DSP kalt «The Nyquist-Shannon sampling theorem.» Helt konkret sier det at dersom et signal ikke inneholder frekvenser over $0.5*F_S$, så kan det originale signalet bli reprodusert nøyaktig fra det samplede signalet. Frekvensen $0.5*F_S$ kalles Nyquist frekvensen. En annen måte å beskrive dette på er at dersom vi har et kontinuerlig signal bygd opp av flere signaler med ulike frekvenser, alle mindre enn en frekvens B, så er $F_S = 2B$ tilstrekkelig samplingfrekvens for å rekonstruere signalet. Frekvensen 2B kalles Nyquist raten. Under har jeg lagt ved et bilde som illustrerer sammenhengen mellom de ulike frekvensene som inngår i teoremet. «Bandlimited channel» referer til at signalet ikke inneholder frekvenser over en bestemt frekvens, nemlig B.

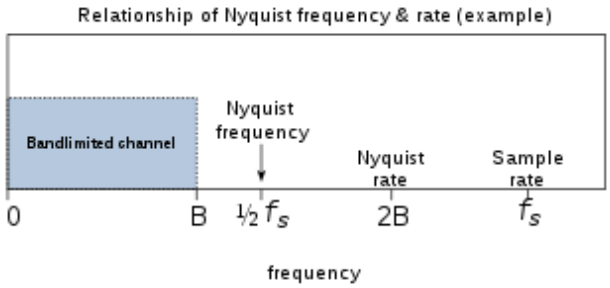

Fig 4.0: Sammenheng mellom sentrale begreper

Sampling teoremet er så sentralt i DSP at det fortjener litt videre forklaring. Et fullstendig bevis av teoremet er utenfor hensikten for denne oppgaven, så jeg nøyer meg med å illustrere hvorfor teoremet stemmer med å se på Claude Shannons opprinnelige bevis fra 1948. I det som følger antar vi at $F_S = 2B$. Vi vil vise at dette medfører at et analogt signal kan reproduseres fullstendig fra det samplede signalet. Ettersom $|f| = \left|\frac{\omega}{2\pi}\right| < B$ for alle frekvensene x(t) inneholder, medfører det at Fourier transformasjon til x(t), X(ω), er null utenfor dette intervallet. I det kontinuerlige domenet vil en invers Fourier transformasjon normalt gå fra -∞ til ∞, men siden X(ω) er null utenfor dette intervallet, kan vi endre integrasjonsgrensene for den inverse Fourier-transformasjonen:

11) $$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{i\omega t}\,d\omega = \frac{1}{2\pi}\int_{-2\pi B}^{2\pi B} X(\omega)e^{i\omega t}\,d\omega$$

Hvis vi nå beveger oss inn i det diskre domenet og lar $t = nT = \frac{n}{2B}$, ender vi opp med følgende representasjon:

12) $$x[\frac{n}{2B}] = \frac{1}{2\pi}\int_{-2\pi B}^{2\pi B} X(\omega)e^{i\omega\frac{n}{2B}}\,d\omega$$

På venstre side av denne likningen har vi verdien av x(t) ved sampling punktene. Hvis vi sammenligner høyresiden med høyresiden av formel 8 ser vi at X(ω) har samme rolle som x(t), og at det er positiv eksponent istedenfor. Dette er altså essensielt den n'te Fourierkoeffisienten i rekkeutvidelsen til X(ω), hvis man regner med en fundamentalperiode [-B, B]. Det eneste som mangler er å gange med 1/P = 1/2B. Dette betyr at verdiene til samplingpunktene avgjør Fourierrekken til X(ω), som entydig beskriver X(ω). Og X(ω) igjen avgjør x(t) entydig gjennom en invers Fourier transformasjon. Vi vet også at vi har all informasjonen vi trenger ettersom vi har samplingpunkter fra hele intervallet hvor X(ω) er forskjellig fra null, som følge av antakelsen. Samplingpunktene leder altså entydig tilbake til det opprinnelige signalet så lenge $F_S = 2B$. (Shannon, 1949) Det er lett å se at samme resultat følger også for $F_S > 2B$

ettersom X(ω) fortsatt er null for |f| > B. Med denne argumentasjonen får vi i alle fall en forståelse av hvor sampling teoremet kommer fra, uten at det går an å kalle det et fullverdig bevis etter dagens standard.

Et spørsmål som er naturlig å stille i forbindelse med sampling teoremet er: «Hva skjer om signalet inneholder frekvenser over Nyquist frekvensen?» I ekte situasjoner er det nemlig umulig å utelukke så høye frekvenser verken fra signalet, eller tilfeldig støy. Når disse frekvensene blir samplet, skjer det et fenomen som på fagspråket heter aliasing, som kan beskrives som at kilden med denne frekvensen prøver å skjule identiteten sin ved å utgi seg for å ha frekvens lavere enn 0.5*F_S. Resultatet blir at dataene vi sitter igjen med, består av en kombinasjon av signaler over og under denne grensen, uten noen måte å finne ut hva som er hva. Under ser vi et vanlig sinussignal samplet med to forskjellige samplingfrekvenser. I bildet til venstre er samplingfrekvensen over Nyquist raten, Fs = 10f, og til høyre er samplingfrekvensen under Nyquist raten, Fs = 1.1f. (Efstathiou)

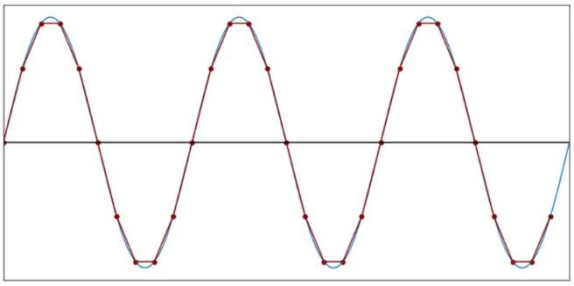

Fig 4.1: Fs = 10f

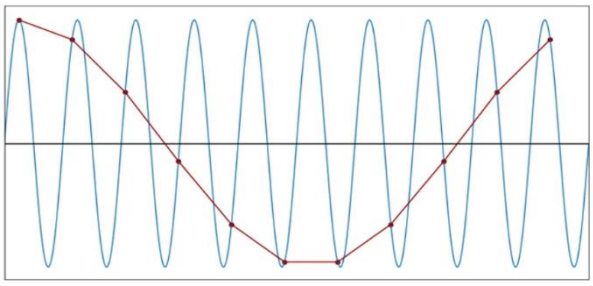

Fig 4.2: Fs = 1.1f

Vi ser tydelig at dersom F_S blir for lav vil en lineær interpolering mellom samplingpunktene ikke lenger representere det opprinnelige signalet. Heldigvis finnes det en metode som kan redusere aliasing, noe som kalles et anti-alias filter. Ideelt sett ønsker man å benytte et skarpt filter som fjerner alle frekvenser over 0.5* F_S, og beholder resten. Dette kan gjøres ved hjelp av et analogt filter før signalene blir diskretisert, men effekten av slike filtre er veldig begrenset og gir ikke så gode resultater. Derfor benytter vi en annen tilnærming i tillegg for å hjelpe problemet, en teknikk som kalles oversampling. Som navnet tilsier, velger vi en veldig høy samplingfrekvens slik at vi plukker opp tilstrekkelig mange frekvenser for en gitt situasjon. Deretter går disse diskrete dataene gjennom et diskret-tid anti-aliasing filter som er innebygget i ADC-nodene. Dette reduserer kravene betraktelig for det analoge filteret,

ettersom signalet nå passerer to filtre istedenfor et og vi oppnår et digitalt signal som mye nærmere representerer det originale.

De sammenflettede dataene blir lagret på harddisken i et spesielt filformat som kalles WAV-format. Dette er et veldig populært filformat fordi det er et såkalt «lossless audio format», altså lagres signalene i sin helhet. Dette er forskjellig fra for eksempel MP3 formatet hvor dataene komprimeres. Resultatet er at du mister en del informasjon, men dette veies opp med at filen ender opp med å bli mindre. WAV-filene derimot kan gjøres så store som ønskelig, og gjør at en kan implementere høy sampling frekvens, som er nyttig for vår del siden vi vil benytte oss av et oversampling-filter.

For å måle kvaliteten på dataene vi analyser er det vanlig å bruke et forhold som kalles «Signal to Noise Ratio», vanligvis forkortet til SNR. Definisjonen av SNR oppgis, som regel, som forholdet mellom intensiteten av det ønskede signalet og uønsket støy. Dersom alle lydintensitetene oppgis i dB, kan vi uttrykke formelen for SNR på en lettere måte:

13) $SNR = S\text{-}N = S_L\text{-}T_L\text{-}N$

Grunnen til at vi kan subtrahere, er at måleenheten dB er en relativ enhet, og regnes ut via en logaritme, som vi så i formel 1 for hydrofonsensitiviteten. Og for logaritmer er det slik at logaritmen av et forhold er lik differansen mellom logaritmene. I formelen har vi følgende komponenter: S er signalet som måles av en hydrofon, S_L står for «signal level» og er lydnivået på stedet lyden oppstår, T_L står for «transmission loss» og er hvor mye lydnivået reduseres som følge av avstanden mellom kilden og hydrofonen og N er lydnivået av støyen. Støy kommer fra mange faktorer, deriblant tilfeldig bakgrunnsstøy, ofte kalt hvit støy, eller fra elektriske kretser i form av elektrisk støy. I tillegg vil frekvenser som blir aliaset, også oppføre seg som støy i det aktuelle frekvensområdet, ettersom frekvensene høyere enn $0.5*F_S$ blir representert som frekvenser under $0.5*F_S$.

Ut ifra formelen er det lett å se at SNR = 0 dB tilsier at lydnivået er likt, mens alt over 0 dB tilsier at det ønskede signalet er sterkere enn bakgrunnsstøyen. For å oppnå best mulige resultater, er det derfor ønskelig å ha så høy SNR verdi som mulig. I bunn og grunn er det lite man kan gjøre fysisk for å forbedre SNR. De eneste mulighetene er å flytte seg nærmere kilden, eller benytte bedre utstyr. Det er imidlertid en metode som ikke krever noen fysiske endringer overhodet, nemlig beamforming, som diskuteres i neste seksjon.

4.2 Beamforming

Beamforming er en fellesbetegnelse for applikasjoner hvor det benyttes flere sendere eller sensorer for å sende eller motta signaler, med den hensikt å oppnå en form for selektivitet. Når det gjelder sending av signaler, handler dette om å sende ut signalene fra de ulike senderne på en slik måte at signalene kombineres konstruktivt i retningen signalet skal, og destruktivt i andre retninger. Denne oppgaven handler om beamforming av signaler som mottas. I slike applikasjoner er hensikten å kombinere signalene på en sånn måte at vi oppnår mer informasjon enn om vi kun hadde brukt en sensor.

Ideen om å bruke flere sensorer for å øke kvaliteten av signalet er langt fra ny. For eksempel ble en primitiv utgave benyttet allerede under første verdenskrig for å lytte etter fiendtlige fly. Systemet de brukte bestod av 12 horn som var plassert i hjørnene av to heksagoner. Vinklene til heksagonene kunne styres for å kjapt rette hornene i retning av innkommende fly. Lyden hornene plukket opp, ble kombinert i et rørsystem som ledet rett til øret til en soldat. Det er i ettertid beregnet at dette systemet kan ha økt lyttekapasiteten opp mot ti ganger. På 70-tallet begynte noen å eksperimentere med mikrofoner og oppdaget at selv simple beamformingsalgoritmer basert på gjennomsnitt gjorde signalene bedre enn de individuelle mikrofonene klarte. Derifra eksploderte interessen for beamforming i DSP-grupper, og det ble en periode med hurtig utvikling av algoritmer, og flere og flere anvendelser ble etablert. I dag er det et mangfold av ulike typer beamforming, og det er vanskelig å finne systemer for behandling av signaler som ikke implementerer noen form for beamforming.

I de 100 årene beamforming applikasjoner har vært i bruk har det blitt laget mange beamformere. Noen av disse er enkle og effektive å utføre på bekostning av resultatene, mens andre er veldig omfattende å utføre, men gir bedre resultater. Det de gjør annerledes er hvordan de kombinerer signalene fra de ulike sensorene, samt hvilke antakelser som blir gjort. Noen beamformere er til og med adaptive, og justerer seg selv etter hvert som de mottar signalene, for økt presisjon. Hvilken type beamformer man bruker velges basert på ønsket presisjon veid opp mot hvor rask prosesseringshastighet som kreves.

Felles for alle typene av beamforming er at de basert på kjente egenskaper om sensorene og deres geometriske lokasjoner i forhold til hverandre, kombinerer signalene på en hoherent måte, og dermed oppnår mer informasjon om

signalene som analyseres. Dette er mulig fordi at i de aller fleste tilfeller, med unntak av bølgefronter parallelle med lineære sensor konfigurasjoner, så vil de ulike sensorene registrere et tidsforskjøvet signal i forhold til hverandre. På bakgrunn av dette kan dataene sammenlignes og kombineres, og vi ender opp med informasjon som ellers ville vært utenkelig med bare en sensor. For å illustrere dette har jeg lagt ved et bilde. RF står for «radio frequency». Dette er dermed ikke et akustisk signal, men i praksis er effekten lik. Signalet må fysisk bevege seg lengre for å nå sensorene lengre unna, og siden farten lokalt er konstant, fører dette automatisk til at det blir en tidsforskyvning.

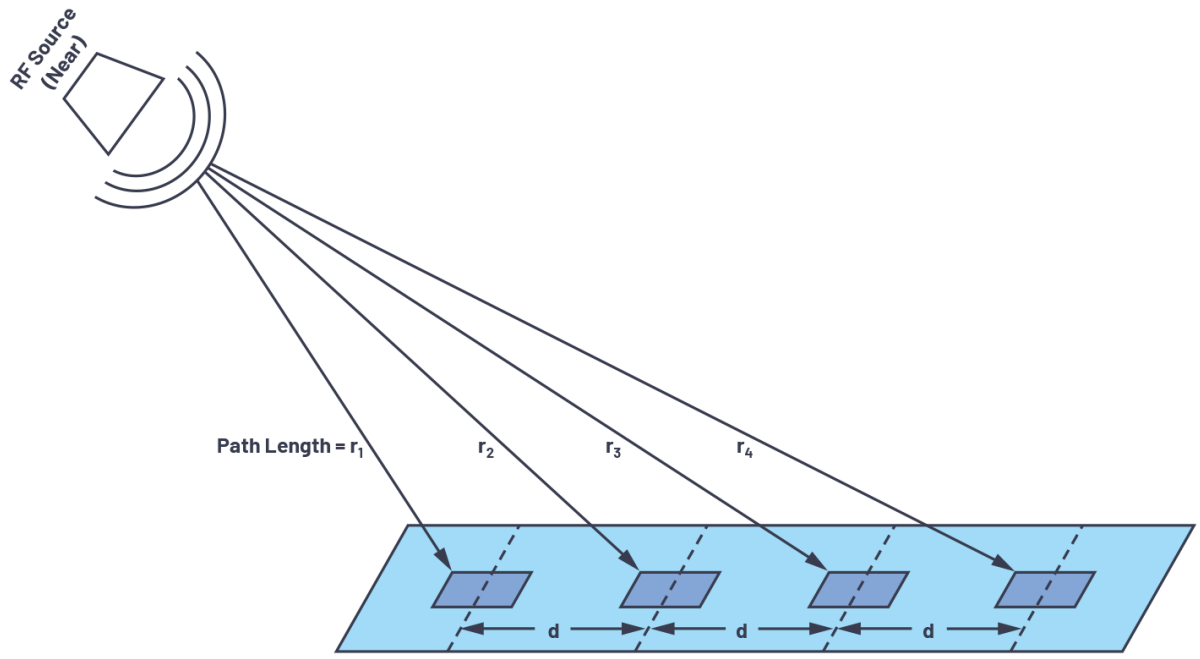

Fig 4.3: Tidsforskjøvet signal

Godene av beamforming er mange, men jeg begrenser meg til å fortelle kort om de fordelene som vi kan dra mest nytte av i dette prosjektet:

1) En av de mest ettertraktede fordelene er at beamforming lar deg lokalisere kilden. For vår del dreier det seg da om å bestemme vinkelen mellom antenneaksen og kilden. I bestemte tilfeller kan det også være mulig å hente ytterlig informasjon. For eksempel om kilden beveger seg i forhold til oss kan det være mulig å avgjøre kursen fartøyet har ved å se på hvordan, og hvor raskt signalet endrer seg. I tillegg kan det være mulig å avgjøre hastigheten til kilden. Dette er kun i situasjoner der bevegelsen er tilstrekkelig rask eller sensorene tilstrekkelig følsom, slik at det lar seg gjøre å detektere et dopplerskift i lydsignalene.

2) Som nevnt i seksjon 4.1 øker beamforming SNR verdien. Dette er et resultat av at beamformingsalgoritmer er laget for å koherent legge sammen signalet fra alle sensorene. Støyen derimot er antatt å være tilfeldig, og kan derfor ikke legges sammen koherent med samme operasjoner. Dette blir altså en dobbel positiv effekt i og med at signalet forsterkes samtidig som støyen forblir på samme nivå. Hvor mye bedre en bestemt sensor konfigurasjon er enn en enkelt sensor, er et typisk spørsmål. Svaret er noe som på fagspråket kalles «array gain». Det er definert som SNR verdien for det beamformede signalet delt på SNR verdien på det opprinnelige signalet.

3) Når vi har funnet ut hvilken retning lydkilden ligger er det mulig å manøvrere båten slik at antennes konfigurasjon tillater oss å få mer informasjon ut av signalet. Dette kan for eksempel være å flytte båten nærmere kilden hvor signalet er sterkere.

Som nevnt i problemstillingen for oppgaven, har målet vært å lage en beamformer som kan brukes for å analysere data fra antennen. For å oppnå dette, begynte vi med å se på et generelt lydsignal som fanges opp av antennen. Dette signalet husker vi at kan uttrykkes som formel 5 i seksjon 3.0. Herfra ønsker vi å finne en måte å manipulere dette uttrykket, og forhåpentligvis oppdage sammenhenger vi kan utnytte oss av i en beamformer. Med Fourier analyse i bakhodet gjør vi følgende omskriving av formel 5:

14) $$\mathrm{A}e^{i(\omega t-\mathrm{kxcos}\Phi)} = \left(Ae^{-i\mathrm{kxcos}\Phi}\right)e^{i\omega t}$$

Her gjenkjenner vi utrykket i parentesen som en Fourier koeffisient. Nærmere bestemt koeffisienten som tilhører tid-frekvensen ω, altså det jeg har kalt for c_k i formel 7. Dette er faktisk det samme resultatet som vi hadde fått om vi hadde utført en Fourier transformasjon av dataene fra en hydrofon langs antennen, så vi sparer litt tid med å gjøre denne observasjonen.

Neste steg er å gjøre denne koeffisienten diskret. Så langt har vi bare sett på diskretisering av tid, men samme framgangsmåte kan brukes for å diskretisere rommet også. Vi kan nemlig utnytte at antennen inneholder mange hydrofoner med uniform avstand mellom, slik at vi kan gjøre følgende skifte av variabler: x → m*d, hvor m korresponderer til den m'te hydrofonen i antennen og d er den uniforme avstanden mellom hydrofonene. Da får vi følgende utrykk, som vi gir navnet p[m]:

15) $p[m] = Ae^{-i\mathrm{kmdcos}\Phi}$

For å gjøre neste steg litt lettere å visualisere, introduserer vi to nye «dummy» variabler og gjør følgende omskrivning i eksponenten i formel 15:

16) $kdcos\Phi = 2\pi(l+uN)/N$

Her er l og u «dummy» variablene, og N antall hydrofoner. Denne omskrivingen er også en form for diskretisering. Denne diskretiseringen er for å sikre at inversdomenet til rommet, som frekvensdomenet er for tid, også er diskret. Dette er nødvendig for å sikre at bruken av Fourier transformasjoner i den ferdige beamformeren, kan utføres begge veier.

For at de to uttrykkene på hver side av formel 16 skal være like, ser vi at høyresiden også må ha en form for periodisitet. Denne periodisiteten er gjemt i uN-leddet. Vi kan se for oss at l-verdiene ligger langs en ring som N diskrete punkter. Når vi legger til uN til en vilkårlig l-verdi, vil vi bare bevege oss u ganger rundt ringen, og ende opp på samme l-verdi. Altså representer l + uN samme punkt som l, så lenge u er et helt tall. Derfor kan vi for øyeblikket se bort fra dette leddet og fortsette med følgende uttrykk:

17) $p[m] = Ae^{-im\,2\pi(l+uN)/N} = Ae^{-i\frac{2\pi ml}{N}}$

Nok en gang med Fourier analyse i bakhodet, gjenkjenner vi dette som et ledd av følgende rekkeutvidelse av p[m].

18) $p[m] = \frac{AN}{N} * \sum_{n=0}^{N-1} \delta[n-l]e^{-i\frac{2\pi mn}{N}}$

Dette har nøyaktig samme form som formel 9, altså en diskret Fourier transformasjon. Det er dette vi har jobbet oss mot, et utrykk vi vet hvordan vi kan jobbe med. Helt konkret ser vi at p[m] er den diskrete Fourier transformasjonen av $\delta[n-l]$. Dette er en vanlig funksjon som dukker opp ofte i fysikken, og har navnet deltafunksjonen. Kjennetegnet til deltafunksjonen er at funksjonsverdien er null alle steder bortsett fra hvor n = l, da er verdien 1 for diskrete signaler. Det er dette faktumet vi vil dra nytte av i beamformeren. Dersom vi tar en invers Fourier transformasjon av p[m], er det altså deltafunksjonen vi sitter igjen med:

19) $IDFT(p[m]) = AN\,\delta[n-l]$

Hvis vi går tilbake til formel 16 hvor vi introduserte l, ser vi at den har en direkte sammenheng med vinkelen mellom antenneaksen og bølgenummer

vektoren, altså retningen lydbølgene kommer fra. Dermed har vi kommet på sporet av en algoritme som kan gi oss denne vinkelen, som er det vi er ute etter om vi ønsker å finne ut hvor lydkilden er lokalisert! Neste steg er å løse formel 16 for vinkelen, da ender vi opp med følgende uttrykk for retningsvinkelen:

20) $$\Phi = \text{acos}(\frac{2\pi(\text{l}+\text{uN})}{kdN}) = \text{acos}(\frac{\lambda(\text{l}+\text{uN})}{dN})$$

Siste omskrivning får vi fra sammenhengen mellom bølgetallet og bølgelengden i formel 4. Hvis vi tenker tilbake på det vi nettopp fant ut om deltafunksjonens maksimum ved verdien l, impliserer dette at denne vinkelen er forbundet med et maksimum. Det er nemlig vinkelen som tilsvar Fourier koeffisienten i formel 14 som er størst. Denne koeffisienten er en av mange, og det er som sagt koeffisientene vi får av å ta en DFT av tidsdataene fra en hydrofon. Nå når vi kjenner de relevante sammenhengene er det på tide å introdusere selve beamformeren som skal ta oss fra tidsdataene til retningen lydkilden er lokalisert, samt frekvensen lydsignalet har.

Dataene vi analyserer er i form av en 8192x144 matrise hvor tidsdataene for hydrofonene er kolonnene. Første steg av beamformeren er å ta en kolonnevis DFT, eller FFT. FFT står for Fast Fourier Transform, og er en algoritme med samme egenskaper som DFT, men effektivisert for å være raskere å utføre for en datamaskin. Resultatet av denne FFT'en er en matrise med samme dimensjoner som den opprinnelige, bare at kolonnene nå inneholder Fourier koeffisienter på samme form som p[m]. For et enkelt signal som svinger med en frekvens, kommer en av disse koeffisientene til å være større enn de andre for den frekvensen, og for alle 144 hydrofonene. Neste steg er å ta en IDFT, eller rettere sagt den effektiviserte motparten IFFT, for å finne vinkelen som gir størst amplitude for hver frekvens. Vi gjør imidlertid et lite triks. Som sagt vil vi ta Fourier transformasjonene kolonnevis, og siden vi vil at IFFT'en også skal tas frekvens-vis, trenger vi at Fourier koeffisientene som tilsvarer samme frekvens er i samme kolonne. Disse koeffisienten opptrer i samme rad, så om vi transponerer FFT-matrisen, ender vi opp med en 144x8192 matrise hvor kolonnene tilsvarer like frekvenser. Etter IFFT'en ender vi opp med en ny 144x8192 matrise som også inneholder en rad med høyere amplitude enn resten. Dette kommer til å være den l'te raden, som altså korresponderer med retningsvinkelen. I tillegg til at denne raden generelt har høyere amplituder, har kolonnen som kom av den dominante frekvensen, fortsatt høyere amplituder, og elementet som er felles er derfor maksimumsverdien til matrisen. Dette er verdien som tilsvarer både rett vinkel og frekvens.

Det er altså etter IFFT'en at «dummy» variablene l og u kommer inn i bildet. Vi kan si at l og u er indeks-variabler, hvor l angir en indeks mellom 0 og 143, og u kan være hvilket som helst helt tall, i og med at Fourier rekken er uendelig lang. Dette er sant så lenge vi regner med komplekse vinkler, men ettersom vi lever i den reelle verden, hvor alle vinkler er reelle, begrenser vi oss til den vanlige definisjonsmengden for cosinus. Da må uttrykket vi tar invers cosinus av i formel 20, være mellom -1 og 1:

$$21) \qquad -1 \leq \frac{\lambda(l+uN)}{dN} \leq 1$$

I dette uttrykket er alle komponentene utenom l og u konstant, men l er jo en indeks mellom 0 og 143 som referer til IFFT matrisen, og det er derfor kun u som må begrenses for å sikre en reell vinkel. For å finne ut hva disse begrensningene innebærer, må vi løse denne ulikheten for u. Med å gange med dN, dele på bølgelengden, trekke fra l på begge sider og dele på N igjen, ender vi opp med følgende begrensning:

$$22) \qquad -\frac{d}{\lambda} - \frac{l}{N} \leq u \leq \frac{d}{\lambda} - \frac{l}{N}$$

For å sikre at vinkelen i tillegg er unik, må vi legge en ytterlig begrensning på u, nemlig at differansen mellom den største og minste u-verdien må være mindre enn eller lik 1. Denne begrensningen skjønner vi gjennom analogien jeg brukte tidligere mellom variablene l og u, og en ring. Dersom vi antar at differansen mellom u_{max} og u_{min} er større enn 1, så kommer l + uN til å representere samme verdi for flere ulike verdier av u. Og siden l + uN inngår i formel 20, må verdien være unik for at vinkelen skal være unik. Når vi implementerer denne begrensningen på u ender vi opp med følgende resultat:

$$23) \qquad u_{max} - u_{min} \leq 1 \rightarrow \left(\frac{d}{\lambda} - \frac{l}{N}\right) - \left(-\frac{d}{\lambda} - \frac{l}{N}\right) \leq 1 \rightarrow d \leq \frac{\lambda}{2}$$

Men d er jo mellomrommet mellom hydrofonene i antennen. Altså har ved å prøve å sikre oss unike og reelle vinkler, snublet over et kriterium for mellomrommet mellom hydrofonene. Dette er enda et eksempel på aliasing, som jeg snakket om i seksjon 4.1, bare for rommet istedenfor tiden. Neste steg er å plugge resultatet i formel 23 tilbake i formel 22. I tillegg velger vi verdien for l som gir størst og lavest u, henholdsvis l = N for laveste verdi og l = 0 for høyeste verdi. Da får vi at u må være mellom -1.5 og 0.5, men siden u må være et heltall er u begrenset til å være enten -1 eller 0. Dette gir oss to muligheter for utrykket for vinkelen, og hvilke indekser som tilsvarer hvilket utrykk:

24) $u = 0: 0 \leq \frac{d}{\lambda} - \frac{l}{N} \rightarrow l \leq N\frac{d}{\lambda} \leq \frac{N}{2}$ og $\Phi = \text{acos}(\frac{\lambda l}{dN})$

25) $u = -1: -\frac{d}{\lambda} - \frac{l}{N} \leq -1 \rightarrow l \geq (-\frac{d}{\lambda} + 1)N \geq \frac{N}{2}$ og $\Phi = \text{acos}(\frac{\lambda(l-N)}{dN})$

IFFT-matrisen vi sitter igjen med etter at signalet har passert gjennom beamformeren, kalles enkelt og greit det beamformede signalet. For å sette pris på effekten av denne beamformeren har jeg valgt å plotte det beamformede signalet mot frekvens og mot vinkel. Når vi skal oversette fra indeksverdiene til vinklene, må vi ta hensyn til informasjonen i formel 24 og 25. Plottet vi ender opp med er tredimensjonalt, men for å gjøre det lettere å visualisere, benytter jeg et colormap, samt snur perspektivet sånn at vi ser plottet fra en vinkel. Nå er den tredimensjonale karakteren erstattet med fargekoding hvor blå farger tilsvarer lavere amplitude, mens rød tilsvarer høy amplitude. Frekvensene er langs ene aksen, og vinklene på den andre, og signaler som skiller seg ut i dataene opptrer som fargede punkter i plottet. Resultatene av beamformingsanalysen for oppgaven ligger i seksjon 5.0.

4.3 Data

Som nevnt innledningsvis, skal ikke antennen være ferdig før høsten 2022, og mens denne oppgaven blir skrevet er det derfor mangel på reelle data fra sjø-tester. Siden dette er en første generasjons innretning, er det heller ikke muligheter for å benytte data fra tidligere generasjoner. Heldigvis har vi en redning, syntetiske data.

Det er nemlig mulig å lage WAV-filer fra bunnen av, med forutbestemte egenskaper. Dette kan gjøres med de fleste kodespråk, og Python har en egen modul som dreier seg om lesing og generering av WAV-filer. For WAV-filene fra antennen kommer det til å være slik at en får ut en matrise som beskrevet i forrige avsnitt, 8192x144, når vi leser av filen. For å genere syntetiske signaler begynner jeg derfor med å lage en matrise på samme form, med ønskede hydrofondata som kolonnene. Denne matrisen kommer til å inneholde verdier som er av type float32, altså 32-bit floating point number. Dette er et vanlig tallformat for desimaltall som det er mulig for datamaskiner å bearbeide. WAV modulen jeg bruker, støtter imidlertid ikke dette formatet, og det må derfor oversettes til et byteformat. Dette gjør jeg ved å gå veien om tallformatet, int32, altså 32-bit integer number, og derfra til et 24-bit bytearray. Dette bytearrayet kan oversettes direkte til en 24-bit wavefil. Denne prosessen er nødvendig ettersom selskapet har valgt å bruke dette 24-bit formatet på wave-filene for antenneprosjektet. Grunnen til dette er at med utstyret som brukes, er dette det formatet som gir høyest presisjon, samtidig som det ikke krever for

mye prosesserings kraft. Denne størrelsen muliggjør også for databehandling i realtime, noe som er veldig attraktivt for denne typen innretninger.

Når de syntetiske dataene er lagret som en wave-fil, kan de åpnes igjen ved å gjøre stegene i forrige avsnitt i motsatt rekkefølge. Filen leses av, bytearrayet oversettes til en int32 matrise, som igjen oversettes til det opprinnelige float32 formatet. Denne metoden krever noen avrundinger, men disse påvirker verdiene i så liten grad at de ikke er av praktisk betydning.

De testsignalene jeg har valgt å teste ut beamformeren på, er et rent cosinussignal, et signal som er en kombinasjon av to cosinussignaler med samme frekvens fra forskjellige retninger, og til slutt en kombinasjon av to cosinussignaler fra forskjellige retninger, og med ulik frekvens. I tillegg til disse signalene introduserer jeg hvit støy. Denne simulerer jeg ved hjelp av en normalfordelingsfunksjon som angir en støyverdi for hvert punkt i signalmatrisen. Fremgangsmåten for å generere alle disse signalene er den samme. Først velger jeg hvilke egenskaper jeg vil at signalet skal, altså frekvens og hvilken retning det skal komme fra. Frekvensen er det bare å definere i Python, og retningen avgjør vi ved hjelp av formel 24 og 25, ved å velge oss hvilken indeks vi ønsker det skal være høyest amplitude. Når retningsvinkelen er valgt, bruker vi denne informasjonen til å introdusere et faseskift som simulerer tidsforsinkelsen vi hadde observert mellom hydrofonene med ekte data. Nå er vi klare til å definere signalene på formen:

26) $s = \cos(2\pi fn\text{-phase}) = \cos(2\pi fn - kx\cos\Phi)$

Hvor f er frekvensen til signalet i det diskrete domenet, n er en indeksvariabel for tidssamplene og phase er fasen som er individuell for hver hydrofon. Fasen i formel 26 er den samme som i formel 5. Dette kommer av at formel 26 representerer den reelle løsningen av bølgelikningen, mens p(x,t) er en av to komplekse løsninger, som kombinert gir denne reelle løsningen. Dette beskriver jeg nærmere i seksjon 5.0.0. Et slikt cosinussignal genereres per hydrofon og legges til en liste, og denne listen blir deretter konvertert til en matrise med riktige dimensjoner, og da har vi et ferdig syntetisk signal. Under følger skjermbilder fra kodingen av denne prosessen, og hva de ulike delene er:

```
##Få signalet på rett form

def cossignal(freq,phase):
    l = []
    for i in range(nchannels):
        s = np.cos(2*np.pi*freq*n-phase[i], dtype='float32')
        l.append(s)
        A = np.array(l)
        At = np.transpose(A)
    return At

##Convertere dataene til wave-fil

def convertInt32to24bitsBytearray(s):
    b = bytearray([int('%d'%(((s[i, j] & (0x0000ff << (8*k))) >> (8*k)))) \
                   for i in range(s.shape[0]) for j in range(s.shape[1])
                   for k in range(3)])
    return b

def convertFloat32ToInt32(s):
    return np.round(s * 0x7FFFFF).astype('int32') & 0xFFFFFF
```

Fig 4.4: Viktige funksjoner

Dette er alle funksjonene jeg trenger for å genere signalene. Funksjonen cossignal returnerer en 8192x144 matrise med samplede cosinussignaler på formen spesifisert i formel 26 i kolonnene. De to andre er for å konvertere dataene til riktig format sånn at de kan skrives om til wave-filer.

```
BytesPerSample = 3                          ##Antall byte per sample
NumSamples = 8192                           ##Anatall tidssampler
Fs = 12207                                  ##Sampler/sekund
nchannels = 144                             ##Antall hydrofoner
c = 1500                                    ##Lydhastigheten i vann
d = 0.5                                     ##Avstanden mellom hydrofonene
n = np.arange(0,NumSamples,1)               ##Sample indeks liste
Listnchannels = np.arange(0,nchannels,1) ##Hydrofon/IFFT indeks liste
```

Fig 4.5: Generelle parametere

Her definerer jeg de parameterne som skal være felles for alle signalene. Alle er oppgitt i SI-enheter. De to siste er lister som inneholder henholdsvis hydrofonindekser og sampleindekser. Siden Python er et digitalt verktøy, arbeider det med diskrete data og lister som disse er derfor nødvendig for å fortelle Python hvilke og hvor mange punkter den skal utføre en beregning for. For vår del blir disse listene brukt for å definere signalene og faseskiftene som inngår i analysen. Under ser vi koden som definerer de tre testsignalene.

```
##Enkelt cosinussignal

f0 = 1/20           ##Diskret tid frekvens
F0 = f0*Fs          ##Kontinuerlig tid frekvens
wavelength = c/F0   ##Bølgelengden til signalet

l = 10                                                  ##IFFT indeksen hvor vi ønsker maks amplitude
phi = np.arccos(wavelength*l/(nchannels*d))             ##Retningsvinkelen
phase = (2*np.pi/wavelength)*np.cos(phi)*d*Listnchannels ##Tidsforsinkelse mellom hydrofonene

y = cossignal(f0,phase)               ##Rent signal
wn = np.random.normal(0,0.05,y.shape) ##Tilfeldig støy

y = 0.75*y + wn   ##Signal + støy

y_n = convertFloat32ToInt32(y)
data = convertInt32to24bitsBytearray(y_n)

wavefile = wave.Wave_write('Testsignal_1.wav')
wavefile.setnchannels(nchannels)
wavefile.setnframes(NumSamples)
wavefile.setsampwidth(BytesPerSample)
wavefile.setframerate(Fs)
wavefile.writeframes(data)
wavefile.close()
```

Fig 4.6: Genere testsignal 1, F0 = 610.35 Hz og Φ = 70°

```
##To cosinussignal fra forskjellig retning

f0 = 1/20           ##Diskret tid frekvens
F0 = f0*Fs          ##Kontinuerlig tid frekvens
wavelength = c/F0   ##Bølgelengden til signalet

l1 = 17                                                 ##IFFT indeksen hvor vi ønsker maks amplitude
phi1 = np.arccos(wavelength*l1/(nchannels*d))           ##Retningsvinkelen
phase1 = (2*np.pi/wavelength)*np.cos(phi1)*d*Listnchannels ##Tidsforsinkelse mellom hydrofonene
y1_1 = cossignal(f0,phase1)

l2 = 116                                                ##IFFT indeksen hvor vi ønsker maks amplitude
phi2 = np.arccos(wavelength*(l2-nchannels)/(nchannels*d)) ##Retningsvinkelen
phase2 = (2*np.pi/wavelength)*np.cos(phi2)*d*Listnchannels ##Tidsforsinkelse mellom hydrofonene
y1_2 = cossignal(f0,phase2)

y1 = 0.5*(y1_1 + y1_2)                ##Kombinert signal
wn1 = np.random.normal(0,0.05,y1.shape) ##Tilfeldig støy
y1 = 0.75*y1 + wn1  ##Signal + støy

y1_n = convertFloat32ToInt32(y1)
data1 = convertInt32to24bitsBytearray(y1_n)

wavefile = wave.Wave_write('Testsignal_2.wav')
wavefile.setnchannels(nchannels)
wavefile.setnframes(NumSamples)
wavefile.setsampwidth(BytesPerSample)
wavefile.setframerate(Fs)
wavefile.writeframes(data1)
wavefile.close()
```

Fig 4.7: Genere testsignal 2, F0 = 610.35 Hz for begge, Φ1 = 55°og Φ2 = 163°

```
##To cosinussignal fra forskjellig retning med forskjellig frekvens

f1 = 1/10           ##Diskret tid frekvens
F1 = f1*Fs          ##Kontinuerlig tid frekvens
wavelength1 = c/F1  ##Bølgelengden til signalet
l1 = 40                                              ##IFFT indeksen hvor vi ønsker maks amplitude
phi1 = np.arccos(wavelength1*l1/(nchannels*d))       ##Retningsvinkelen
phase1 = (2*np.pi/wavelength1)*np.cos(phi1)*d*Listnchannels   ##Tidsforsinkelse mellom hydrofonene
y2_1 = cossignal(f1,phase1)

f2 = 1/15            ##Diskret tid frekvens
F2 = f2*Fs           ##Kontinuerlig tid frekvens
wavelength2 = c/F2   ##Bølgelengden til signalet
l2 = 132                                             ##IFFT indeksen hvor vi ønsker maks amplitude
phi2 = np.arccos(wavelength2*(l2-nchannels)/(nchannels*d))  ##Retningsvinkelen
phase2 = (2*np.pi/wavelength2)*np.cos(phi2)*d*Listnchannels ##Tidsforsinkelse mellom hydrofonene
y2_2 = cossignal(f2,phase2)

y2 = 0.5*(y2_1 + y2_2)                  ##Kombinert signal
wn2 = np.random.normal(0,0.05,y2.shape) ##Tilfeldig støy
y2 = 0.75*y2 + wn2  ##Signal + støy

y2_n = convertFloat32ToInt32(y2)
data2 = convertInt32to24bitsBytearray(y2_n)

wavefile = wave.Wave_write('Testsignal_3.wav')
wavefile.setnchannels(nchannels)
wavefile.setnframes(NumSamples)
wavefile.setsampwidth(BytesPerSample)
wavefile.setframerate(Fs)
wavefile.writeframes(data2)
wavefile.close()
```

Fig 4.8: Genere testsignal 3, F1 = 1221 Hz og Φ1 = 47°, F2 = 814 Hz og Φ2 = 108°

Som nevnt er fremgangsmåten for å genere signalene den samme. Først velger jeg frekvensen jeg vil signalet skal ha, i det diskrete domenet. Denne frekvensen er forholdet mellom den kontinuerlige frekvensen og sampling-frekvensen. Det er derfor vi kan finne frekvensen og bølgelengden til det kontinuerlige signalet. Deretter velger vi en IFFT indeks hvor vi vil ha maks amplitude, som vi husker fra formel 24 og 25 gir oss retningsvinkelen Φ. I beregningene av Φ har jeg brukt formel 24 for alle l ≤ N/2, mens formel 25 er brukt når l > N/2. Retningsvinkelen Φ er med på å bestemme tidsforskyvningen som oppstår mellom de ulike hydrofonene, som vi finner ved hjelp av phase-linjen. Grunnen til at vi skriver fasen som $\frac{2\pi}{\lambda}$cos(Φ)dn istedenfor kxcos(Φ), som i formel 26, er at disse skrivemåtene er ekvivalente, og at vi da kan bruke parameterne vi allerede har definert. Ekvivalensen følger av formel 4 og diskretiseringen x = nd. Nå har vi alt vi trenger for å bruke cossignal funksjonen som returnerer den ferdige matrisen. For testsignal 2 og 3 som består av to signaler, tar jeg gjennomsnittet av disse for å unngå at amplituden til det kombinerte signalet aldri blir større enn en. Nå er det klart for å genere støykomponenten i signalet, og legge dette til det opprinnelige. Her må jeg også vekte det opprinnelige signalet for å unngå at amplituden blir større enn en. Grunnen til at dette er viktig er at det er en forutsetning for cosinussignaler med reelle argumenter, som er det vi ønsker å simulere. Alt som gjenstår, er å lagre signalene som wave-filer sånn at de kan åpnes igjen i en annen Python notebook.

5.0 Resultater

I denne seksjonen presenterer jeg resultatene som oppnås ved å analysere de syntetiske dataene fra seksjon 4.3. I denne analysen har jeg valgt å ta med både et plot av rådataene og av det beamformede signalet, for å se forskjellen denne analysen faktisk har å si når det kommer til å kunne tolke signalet. Siden antennen inneholder 144 hydrofoner, og dataene blir samplet 12207 ganger per sekund, er det lett å skjønne at det blir mye data å holde styr på. Plottefunksjonen jeg har kodet, håndterer dette med å lage et plott for hver ADC node i antennen. I vårt tilfelle er det derfor 18 plot med åtte hydrofoner i hvert plot. Signalene vi har generert, er 8192 sampler lange og tilsvarer derfor bare et lydsignal på 0.67 sekunder. Det er likevel mye data som skal plottes på et lite område, noe som resulterer i at det blir ganske uoversiktlig. Derfor kommer jeg bare til å legge ved et skjermbilde som tilsvarer noen få perioder fra noen av de 18 plottene for å vise hvordan signalene typisk kan se ut. Resten av plottene vil se nesten like ut ettersom de bare er faseforskjøvet i forhold til hverandre. Før vi ser på dette, vil jeg raskt presentere og forklare litt om koden som bygger opp til de ulike plottene. Dette innebærer funksjoner og generelle parametere som er felles for alle tre testsignalene.

```
##Conversion from file

def convert24bitsBytearrayToInt32(b, shape):
    s = np.array([[np.int32(b[3*(shape[1] * i + j)]) << 8 \
            | np.int32(b[3*(shape[1] * i + j) + 1]) << 16 \
            | np.int32(b[3*(shape[1] * i + j) + 2]) << 24 \
            for j in range(shape[1])] \
           for i in range(shape[0])], dtype='int32')
    return s

def convertInt32toFloat32(s):
    return (s/0x7FFFFF00).astype('float32')
```

Fig 5.0: Funksjoner for å overføre dataene fra wave-fil

Dette er funksjonene som oversetter dataene fra en wave-fil til en matrise vi kan analysere med Python. Som nevnt i seksjon 4.3, gjør den bare stegene for å genere wave-filen i revers.

```
##Plotte rådata
class ScrollableWindow(QtWidgets.QMainWindow):
    def __init__(self, fig):
        self.qapp = QtWidgets.QApplication([])

        QtWidgets.QMainWindow.__init__(self)
        self.widget = QtWidgets.QWidget()
        self.setCentralWidget(self.widget)
        self.widget.setLayout(QtWidgets.QVBoxLayout())
        self.widget.layout().setContentsMargins(0,0,0,0)
        self.widget.layout().setSpacing(0)

        self.fig = fig
        self.canvas = FigureCanvas(self.fig)
        self.canvas.draw()
        self.scroll = QtWidgets.QScrollArea(self.widget)
        self.scroll.setWidget(self.canvas)

        self.nav = NavigationToolbar(self.canvas, self.widget)
        self.widget.layout().addWidget(self.nav)
        self.widget.layout().addWidget(self.scroll)

        self.show()
        exit(self.qapp.exec_())
```

Fig 5.1: Lage eksternt plottevindu

```
def plotting2(wavdata,delay):
    l=[]
    for i in range(wavdata.shape[1]):
        l.append(wavdata[:,i])
    m=[]
    for i in range(0,len(l),8):
        m.append(l[i:i+8])
    fig, chg = plt.subplots(round(len(m)/2),2, figsize = (30,20), constrained_layout = True,sharex=True,sharey=True)
    fig.suptitle('Plot of antenna-channels',fontsize = 16)
    fig.supxlabel('Sample index')
    fig.supylabel('Hydrophone levels(dB?)')
    for i in range(round(len(m)/2)):
        chg[i,0].set_facecolor('black')
        chg[i,0].set_title('Group ' + str(i+1) + '; Hydrophone nr: ' + str(i*8+1) + '-' + str((i+1)*8))
        chg[i,0].grid(color='white',linewidth = 0.5)
        chg[i,0].set_xlim(0,len(m[0][0]) + 11)
        k = int(i+(round(len(m))/2+1))
        chg[i,1].set_facecolor('black')
        chg[i,1].set_title('Group ' + str(k) + '; Hydrophone nr: ' + str((k-1)*8+1) + '-' + str(k*8))
        chg[i,1].grid(color='white',linewidth = 0.5)
        chg[i,1].set_xlim(0,len(m[0][0]) + 11)
    dom = np.arange(0,len(m[0][0]))
    for i in range(0,round(len(m)/2)):
        for k in range(len(m[i])):
            chg[i,0].plot(dom[(delay*k):],m[i][k][(delay*k):], label = 'Hyd' + str((k + 8*i)+1))
            chg[i,0].legend(loc="upper right")
            j = i + round(len(m)/2)
            chg[i,1].plot(dom[(delay*k):],m[j][k][(delay*k):],label = 'Hyd' + str((round(len(m)/2))*8 + k + 8*i + 1))
            chg[i,1].legend(loc="upper right")
    ScrollableWindow(fig)
```

Fig 5.2: Plottefunksjonen

I fig 5.1 blir det definert et eksternt vindu Python skal plotte i istedenfor å plotte internt i notebooken. Dette gir oss mer frihet til å bevege og zoome inn på plottene, som var kritisk ettersom det er så mye data. I tillegg trengte jeg et vindu det gikk an å scrolle i for at plottene ikke skulle bli for små til å kunne ses. Fig 5.2 viser funksjonen som generer rådataplottet. Den er laget for å kunne plotte data fra en matrise ved å lage en liste av kolonnene, og deretter

gruppere disse i grupper på åtte slik at vi ender opp med en liste av lister. Disse gruppene blir deretter plottet i hvert sitt vindu og vi ender opp med et 9x2 grid av plot hvor hvert plot inneholder data for åtte hydrofoner. Aksene i plottet er henholdsvis amplitude på y-aksen og sampling indeks på x-aksen. I tillegg har jeg gjort det slik at hvert plot har en overskrift som forteller hvilke hydrofoner som er i plottet, og en oversikt over hvilken farge som tilsvarer hvilken hydrofon. Når antennen skal testes, er denne funksjonaliteten nyttig for å lett kunne se hvilken hydrofon det eventuelt er noe galt med. Delay argumentet i funksjonen er et manuelt faseskift som kan introduseres for å skille mellom grafer som ligger nesten oppå hverandre. Dette argumentet er satt til null i analysen fordi det naturlige faseskiftet introdusert av tidsforskyvningen, er stort nok til at vi tydelig kan se separate grafer.

```
##Beamformer

def beamformer(sign):
    Y = np.fft.fft(sign,axis=0)
    Y_t = Y.T
    return np.absolute(np.fft.ifft(Y_t,axis=0))
```

Fig 5.3: Beamformer funksjonen

Beamformingsalgoritmen er kodet inn i Python ved hjelp av de innebygde funksjonene for FFT og IFFT. Argumentet sign er signalet vi ønsker å beamforme. Axis 0 er spesifisert for at Fourier transformasjonene skal tas kolonnevis som spesifisert i seksjon 4.2. Absoluttverdien tas fordi vi til syvende og sist ønsker at beamforming plottet skal vise amplituden i desibels. Da må vi bruke formel 1, og siden det da blir tatt en logaritme, kan vi ikke ha noen negative verdier. Som nevnt kommer beamformingsmatrisen til å ha sitt maksimumspunkt for et element i matrisen, samt litt høyere verdier langs den ene raden og kolonnen. Det viste seg at det standard plotteverktøyet til Python, matplotlib, ikke hadde oppløsning nok til å skille ut dette punktet om vi brukte desibelverdiene. Heldigvis er amplitudene til punktene rundt maksimumspunktet litt nærmere maksimumsverdien før vi tar logaritmen. Derfor ser vi resultatet mye tydeligere om vi bruker beamformingsmatrisen direkte uten å regne om til desibel. Derfor landet vi på å gjøre dette istedenfor å leite etter et alternatibeavt plotteverktøy.

```
##Generere MAP og freq

Freqs = np.arange(0,NumSamples,1)*sampleRate/NumSamples
ONE = np.ones(144)

Ymat = np.outer(ONE,Freqs)

l1 = np.arange(0,nchannels/2+1,1)
l2 = np.arange(nchannels/2+1,nchannels,1)-nchannels
l = list(np.append(l1,l2))

with np.errstate(divide='ignore', invalid='ignore'):
    wavelengths = []
    for x in Freqs:
        wavelengths.append(c/(x*nchannels*d))

MAP = np.outer(l,wavelengths)
Xmat = np.real(np.emath.arccos(MAP))*180/np.pi
```

Fig 5.4: Matrisene som inngår i beamforming plottet

For å kunne plotte tredimensjonalt, trenger man en regel som linker en z verdi til alle punkter (x,y) i definisjonsmengden. Som regel bruker dette å være en funksjon f(x,y), men det finnes andre måter også. Siden vi alt har z verdiene i form en 144x8192 matrise, kan vi for eksempel definere to matriser til av samme størrelse, slik at vi har en 1:1 korrespondanse mellom alle tre matrisene. Da vil Python kunne plotte dataene som punkter (x,y,z) på en tredimensjonal overflate. Som sagt, ønsker vi å plotte det beamformede signalet mot vinkel og frekvens. X-verdiene ønsker vi at skal være de ulike retningsvinklene signalet kan ha. For å lage en matrise som kan representere dette, tenker vi tilbake på formel 24 og 25. Listen l1 i bildet over følger formel 24 sin l opp til og med N/2, mens liste l2 følger formel 25 sin l = l – N helt opp til N-1. Listen l er disse listene satt sammen og er en komplett liste av alle unike l-verdier. Resten av formel 24 og 25 er å gange l med bølgelengde og dele på Nd. Jeg slår to fluer i en smekk når jeg definerer listen jeg har kalt wavelengths. Her kjenner vi igjen c som lydhastigheten, og deler c på alle frekvensene i frekvenslisten Freqs, som tilsvarer bølgelengde. I tillegg har jeg delt på Nd, slik at wavelengths er en liste hvor alle elementene er på formen $\frac{c}{fNd}$. Sammen med listen l har vi all informasjonen i formel 24 og 25. Python lar oss nå generere et elementvis produkt av disse listene i form av et ytre produkt, som da leverer matrisen MAP. Siste steg for å få vinklene, er å ta invers cosinus av MAP. Her benytter jeg meg for sikkerhets skyld av en invers cosinus funksjon som tolererer argumenter med absoluttverdi større enn 1, som da kan levere komplekse vinkler. I tillegg bruker jeg np.real for å kun beholde den reelle

delen der dette skulle bli relevant. I tillegg må vi gjøre om fra radianer til grader og står da igjen med vinkelmatrisen, som jeg har gitt navnet Xmat. Frekvensmatrisen, som vi velger å ha som våre y-verdier, lager vi med å definere en liste av frekvenser mellom null og samplingfrekvensen med jevne mellomrom. Deretter tar vi et ytre produkt mellom denne listen og en liste med ettall, sånn at vi ender opp med en matrise hvor alle radene er denne frekvenslisten, som vi kaller Ymat.

```
NumSamples = 8192     ##Anatall tidssampler
sampleRate = 12207    ##Sampler/sekund
nchannels = 144       ##Antall hydrofoner
c = 1500              ##Lydfarten i vann [m/s]
d = 0.5               ##Felles avstand mellom hydrofonene
```

Fig 5.5: Generelle parametere

Dette er de parameterne som må defineres for å kunne lage Xmat og Ymat som beskrevet over, og i tillegg er de nødvendig for at Python skal klare å lese wave-filene riktig og oversette til riktig dataformat.

```
##Beamform plot

fig = plt.figure(figsize = (14,9), dpi=100)
ax = plt.axes(projection = '3d')

surf = ax.plot_surface(Xmat,Ymat,z,cmap='jet')
ax.view_init(30,-90)

c = fig.colorbar(surf)

ax.set(xlim=(0,180),ylim=(0, sampleRate),xlabel='Angle (deg)',ylabel='Frequencies (Hz)')

plt.show()
```

Fig 5.6: Kode for beamforming plottet

Dette er koden for å plotte de beamformede dataene, altså z, mot Xmat og Ymat.

5.0.0 Enkelt cosinussignal

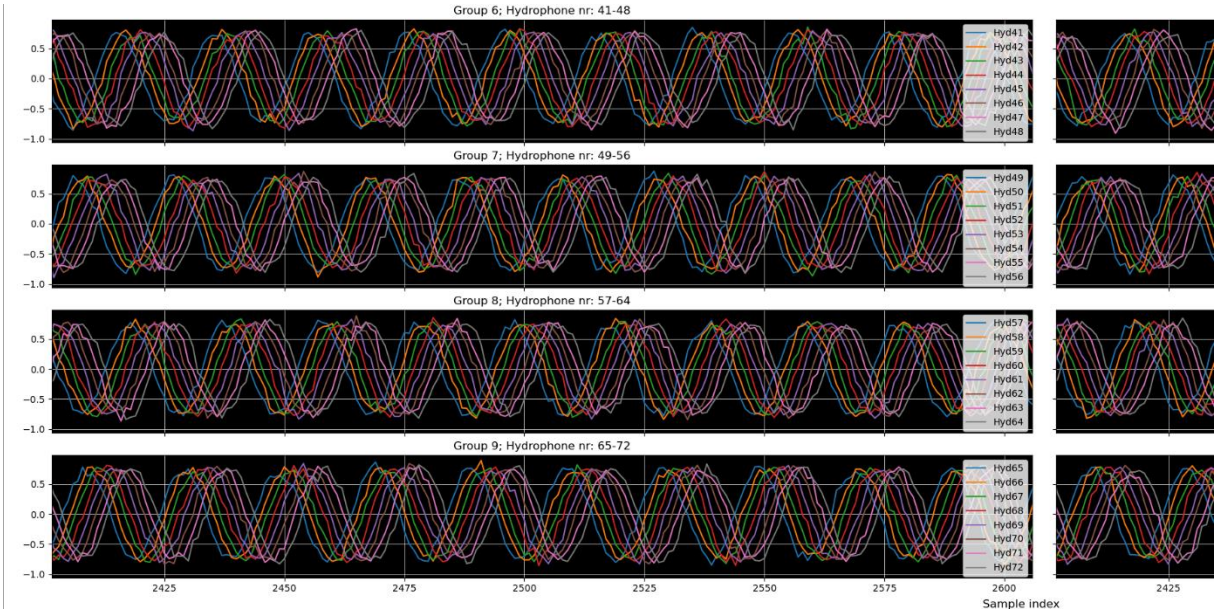

Fig 5.7: Rådata testsignal 1

Her ser vi ni bølgelengder av testsignal 1 som tilsvarer sampling indeksene fra 2400 til 2600. Her er det forholdsvis lett å se at det er en frekvens som skiller seg ut, siden dette tross alt bare er ett signal. Hvis vi ser nøye, ser vi også hvordan støyen påvirker signalet ved at det er små variasjoner bort fra en ellers jevn graf. Basert på dette plottet, er det så å vanskelig å se for seg hvordan man kan bestemme retningen til signalet kun basert på disse dataene, men som vi har sett i gjennom denne oppgaven, skal det altså la seg gjøre.

```
##Beamformer

z = beamformer(y)
maxpoint = np.where(z == np.max(z))
index = list(zip(maxpoint[0],maxpoint[1]))
print("Maximumsverdien tilhører element: ", (index[0],index[1]))
```

```
Maximumsverdien tilhører element:  ((10, 410), (134, 7782))
```

Fig 5.8: Indeksen til maksimumspunktet

```
print(Xmat[10,410])
print(Ymat[10,410])
```

```
70.0621157932827
610.946044921875
```

Fig 5.9: Indeks regnet om til vinkel og frekvens

For å verifisere at beamformeren fungerer, lokaliserte jeg matriseelementet til det beamformede signalet som er størst. Det viser seg at Python finner 2 punkter med samme amplitude. Dette er faktisk en konsekvens som dukker opp fordi vi jobber med et reelt signal. Dette er en side av matematikken som jeg har unnlatt å nevne så langt. Fram til nå har vi bare jobbet med formel 5 som løsning av bølgelikningen for antennen, men siden bølgelikningen er en andregrads differensiallikning, medfører det at den alltid har to lineært uavhengige løsninger. For løsningen i formel 5, som vi gav navnet p(x,t), er denne andre løsningen gitt ved den komplekskonjugerte av p(x,t). Hvis vi kaller denne løsningen q(x,t), har vi altså følgende sammenheng:

27) $q(x,t) = p^*(x,t) = Ae^{-i(\omega t - \mathrm{kxcos}\Phi)}$

Den relle løsningen vi tok utgangspunkt i for de syntetiske signalene, i formel 26, er gitt ved summen av disse to komplekse løsningene.

28) $p(x,t) + q(x,t) = 2\cos(\omega t - \mathrm{kxcos}\Phi)$

For å forstå hvorfor dette fører til at det dukker opp to maksimumspunkter i beamformingsmatrisen, må vi en ny tur innom diskretetisering. Tidligere har vi kun sett eksplisitt hvordan vi kan diskretisre p(x,t) i rommet, som også vil være likt for q(x,t), bare med motsatt fortegn. På samme måte kan vi også diskretisere disse løsningene i tidsdomenet ved å la t = nT. I tillegg må vi diskretisere frekvensen slik at vi kan gjøre de diskrete Fourier transformasjonene begge veier. Vi kan gjøre dette på en liknende måte som for diskretisering av inversdomenet til rommet i fomel 16:

29) $T\omega = 2\pi(m+vM)/M$

I formel 29 er m og v «dummy variabler», som l og u var i formel 16, og som vi fant ut for l og u, er også m og v indeksvariabler. M er det totale antallet tidssampler vi har. Indeksvariabelen m kan da ha verdier mellom 0 og 8191, mens v kan være hvilket som helst heltall. Hvis vi imidlertid vil at verdien eksponenten skal være minst mulig og samtidig positiv for både p(x,t) og q(x,t), krever det at v må være enten 0 eller -1. Nå kan vi skrive opp diskret tid representasjonen av formel 14 for p(x,t) og q(x,t):

30) $p(x,t) = e^{i(\omega t - \mathrm{kxcos}\Phi)} = Ae^{-i\mathrm{kxcos}\Phi}e^{i2\pi\frac{(m+vM)}{M}n}$

31) $q(x,t) = e^{-i(\omega t - \mathrm{kxcos}\Phi)} = Ae^{i\mathrm{kxcos}\Phi}e^{-i2\pi\frac{(m+vM)}{M}n}$

Positiv eksponent ser vi krever v = 0 i formel 30, og v = -1 i formel 31 slik at vi ender opp med:

32) $p(x,t) = Ae^{-i\mathrm{kxcos}\Phi}e^{i2\pi\frac{\mathrm{m}}{M}n}$

33) $q(x,t) = Ae^{i\mathrm{kxcos}\Phi}e^{i2\pi\frac{(\mathrm{M}-\mathrm{m})}{M}n}$

Dette betyr at en delta ved indeks m for p(x,t) alltid impliserer en delta ved indeks M - m for q. Det samme kan vises ved lignende argumentasjon for delta ved indeks l og N - l for diskret rom representasjonen av p og q. For testsignal 1 ser vi at de to makspunktene er (10,410) og (134,7782). Vi ser at dette tilfredstiller det vi fant ut over ettersom l = 10 gir N – l = 144-10 = 134, og m = 410 gir M - m = 8192 – 410 = 7882.

Siden både p og q ligger gjemt i den reelle løsningen vi valgte, vil det alltid følge med to deltaer i beamformingsmatrisen. Grunnen til at denne symmetrien ikke har vært diskutert før, er at q egentlig ikke bringer noen ny informasjon om den reelle løsningen. Amplituden til p og q har jo samme absoluttverdi, og eksponentene er like bare med motsatt fortegn. Det er grunnen til at vi kan forholde oss til kun en av de komplekse løsningene når vi skal estimere frekvensen og retningsvinkelen.

Hvis vi nå undersøker Xmat og Ymat elementet med indeks (10, 410), kjenner vi igjen vinkelen og frekvensen vi ga dette signalet da det ble definert. Det er ingen informasjon om disse verdiene lagret i denne notebooken utenom i signalet vi overførte fra wave-filen, noe som tyder på at beamformeren klarer oppgaven den ble laget for å utføre.

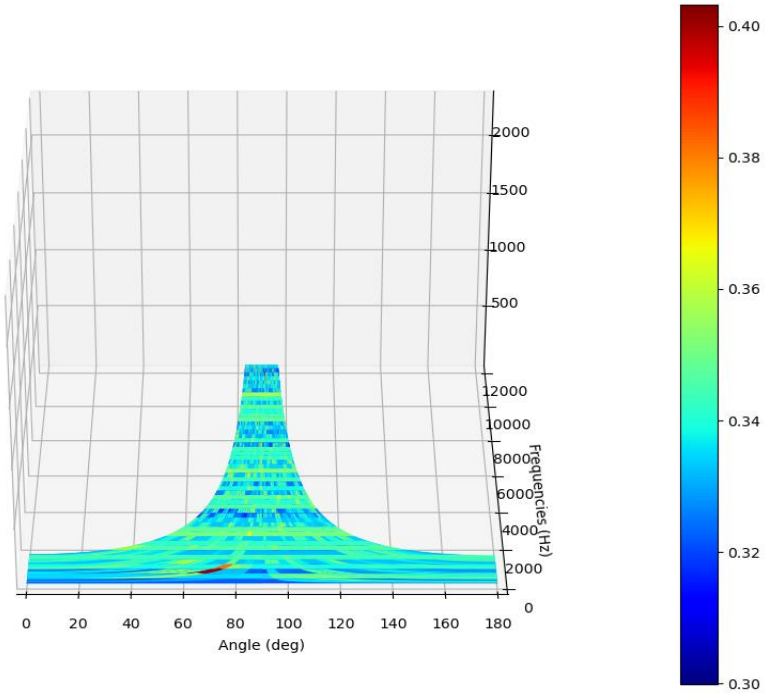

Fig 5.10: Beamforming plot av testsignal 1

Her ser vi et typisk beamformingsplot med riktig deteksjon ved 70°. Her har matplotlib automatisk valgt en fargeskala som skal passe dataene, og har valgt et ganske kort intervall å fordele fargene på. Grunnen til dette er at de fleste verdiene i beamformingsmatrisen er veldig nærme null, mens kun verdiene langs den ene raden og kolonnen er høyere. For å fikse dette forsøkte vi å endre fargeskalaen for å gi litt mer kontrast til mellom de høye verdiene og de lave, men som følge av den lave oppløsningen i matplotlib ente dette opp med at plottet ikke viste noen deteksjon i det hele tatt. Derfor legger jeg ved et beamformingsplot av testsignal 1 fra MATLAB. Dette er slik plottet skal se ut om man tilpasser fargeskalaen bedre til maksimums og minimumsverdiene. I MATLAB er plottet vist ovenfra slik at det i prinsippet er blitt et todimensjonalt plot. I matplotlib var det ikke noen måte å se plottet fra denne vinkelen uten at det ødela kvaliteten på plottet, som er grunnen til at plottene er sett fra forskjellig vinkel. Dette er et gjennomgående problem for testsignal 1 og 2, men vi har valgt å beholde disse plottene ettersom de fortsatt er leselige, og viser riktig deteksjon.

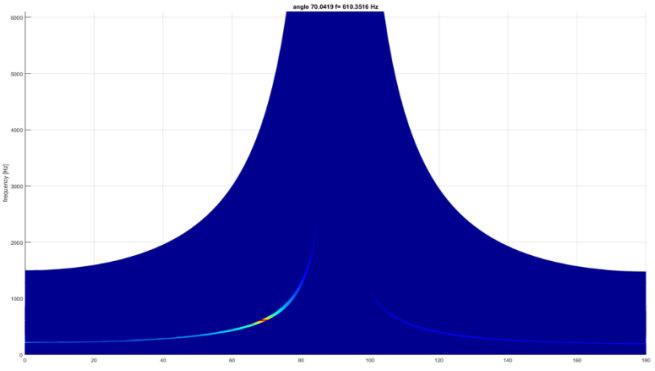

Fig 5.11: MATLAB plot av testsignal 1

5.0.1 To cosinussignaler med samme frekvens fra ulike retninger

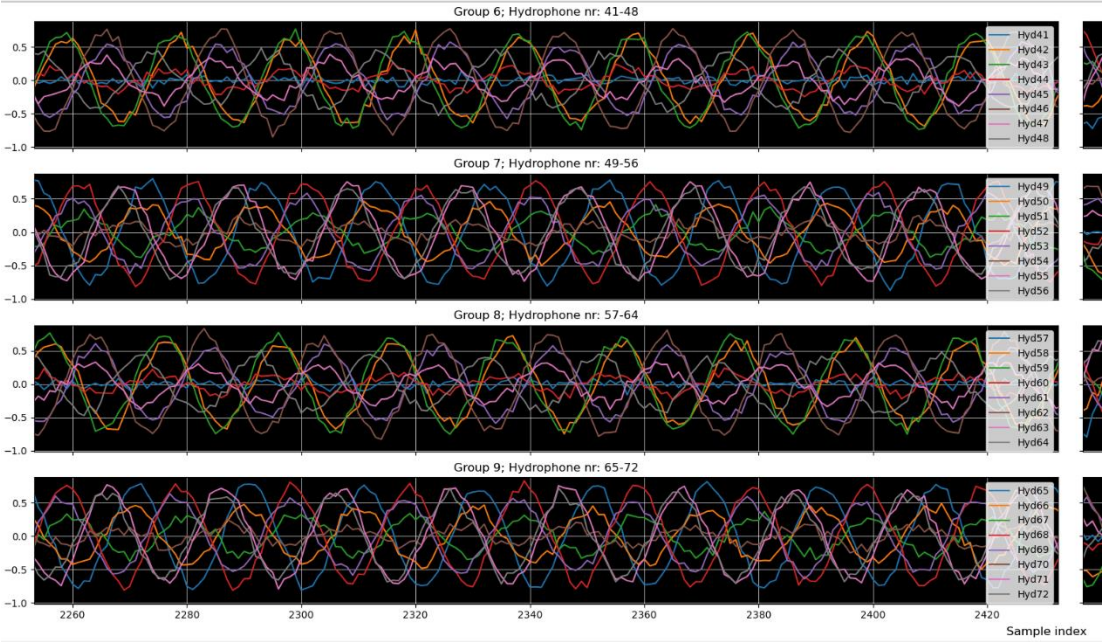

Fig 5.12: Rådata testsignal 2

Her ser vi noen bølgelengder av testsignal 2 som tilsvarer sampling indeksene fra 2250 til 2430. Her er det ikke like lett å se at det er en typisk frekvens som det var for testsignal 1. Vi legger merke til amplituden ikke er konstant ettersom de to testsignalene interferer med hverandre, noe som kan arte seg både konstruktivt og destruktivt. I dette plottet, som for testsignal 1, ser vi at støyen påvirker signalet med at det er små variasjoner bort fra et ellers tydelig signal. Det er umulig å se på disse signalene at det er snakk om to signaler fra forskjellige retninger, men beamformeren klarer det fint.

```
##Beamformer

z = beamformer(y)
maxval = np.sort(z.flat)[-1]
secondmaxval = np.sort(z.flat)[-2]
thirdmaxval = np.sort(z.flat)[-3]

maxpoint = np.where(z == maxval)
index1 = list(zip(maxpoint[0],maxpoint[1]))

secondmaxpoint = np.where(z == secondmaxval)
index2 = list(zip(secondmaxpoint[0],secondmaxpoint[1]))

thirdmaxpoint = np.where(z == thirdmaxval)
index3 = list(zip(thirdmaxpoint[0],thirdmaxpoint[1]))

print("Maximumsverdien tilhører element: ", index1[0])
print("Nest høyeste verdi tilhører element: ", index2[0])
print("Tredje høyeste verdi tilhører element: ", index3[0])
```

```
Maximumsverdien tilhører element:  (116, 410)
Nest høyeste verdi tilhører element:  (28, 7782)
Tredje høyeste verdi tilhører element:  (17, 410)
```

Fig 5.13: Indeksen til maksimumspunktet

```
print(Xmat[116,410])
print(Ymat[116,410])

print(Xmat[17,410])
print(Ymat[17,410])
```

```
162.7082011067378
610.946044921875
54.57041177013335
610.946044921875
```

Fig 5.14: Indekser regnet om til vinkel og frekvens

For å verifisere at beamformeren fungerer må jeg denne gangen lokalisere de to matriseelementene som er størst, ettersom vi forventer et lokalt maksimum for hvert delsignal. Det viser seg imidlertid at den nest høyeste verdien nok en gang kommer av symmetrien vi diskuterte i seksjon 5.0.0, og kan derfor ses bort ifra. Hvis vi hadde sjekket det fjerde høyeste punktet i matrisen, ville vi sett at dette er den symmetriske motparten av (17,410). Vi ser derfor nærmere på det høyeste og tredje høyeste punktet. Hvis vi undersøker Xmat og Ymat elementet med indeks (116, 410), kjenner vi igjen vinkelen og frekvensen vi ga dette delsignalet da det ble definert. Det samme gjelder for Xmat og Ymat verdiene for punktet (17,410). Igjen er det ingen informasjon om disse verdiene lagret i denne notebooken utenom i signalet vi overførte fra wave-filen, noe som tyder på at beamformeren klarer oppgaven den ble laget for å utføre.

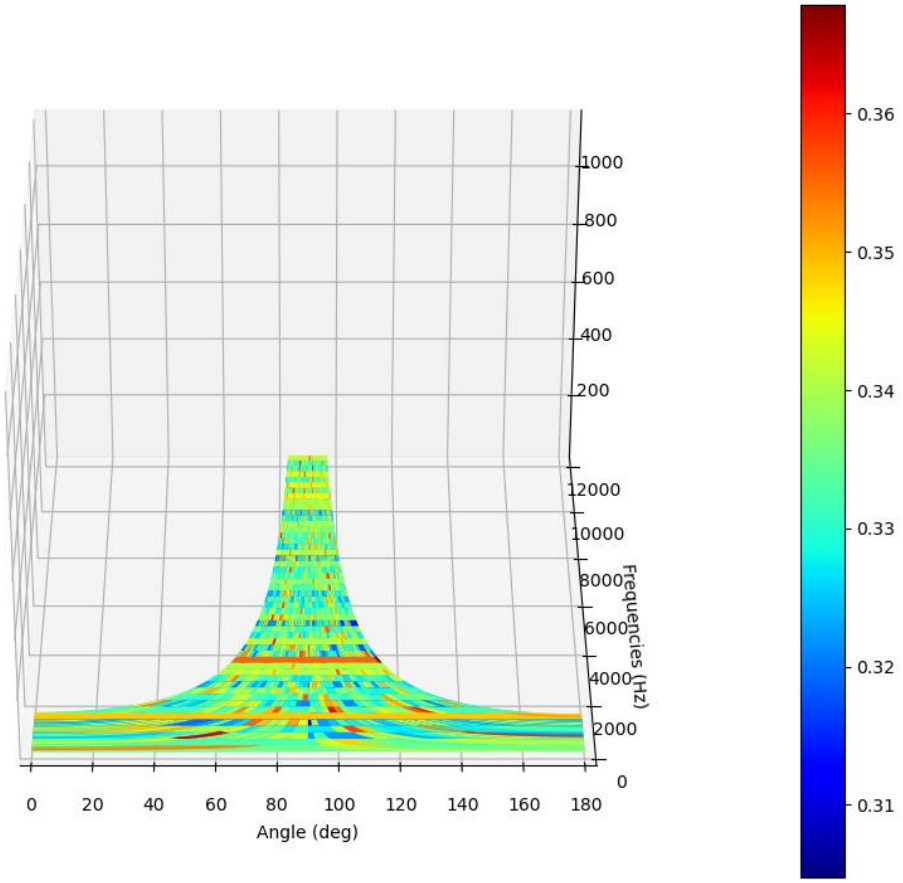

Fig 5.15: Beamforming plot av testsignal 2

Her er ikke beamforming plottet like lett å tolke som for testsignal 1, men vi kan se at plottet kun er mørkerødt rett under 60° merket, som tilsvarer delsignalet som har retningsvinkel 55°, og rundt 160°, som tilsvarer delsignalet med retningsvinkel 163°. Vi ser her at punktene lengst ut mot kantene, har en tendens til å bli smurt ut mer enn punktene nærmere midten av plottet. Grunnen til at dette plottet er så mye mindre oversiktlig er at matplotlib har valgt et enda kortere intervall for fargeskalaen. Derfor gir mindre variasjoner i amplituden større endringer i plottet.

5.0.2 To cosinussignaler med ulike frekvenser og fra ulike retninger

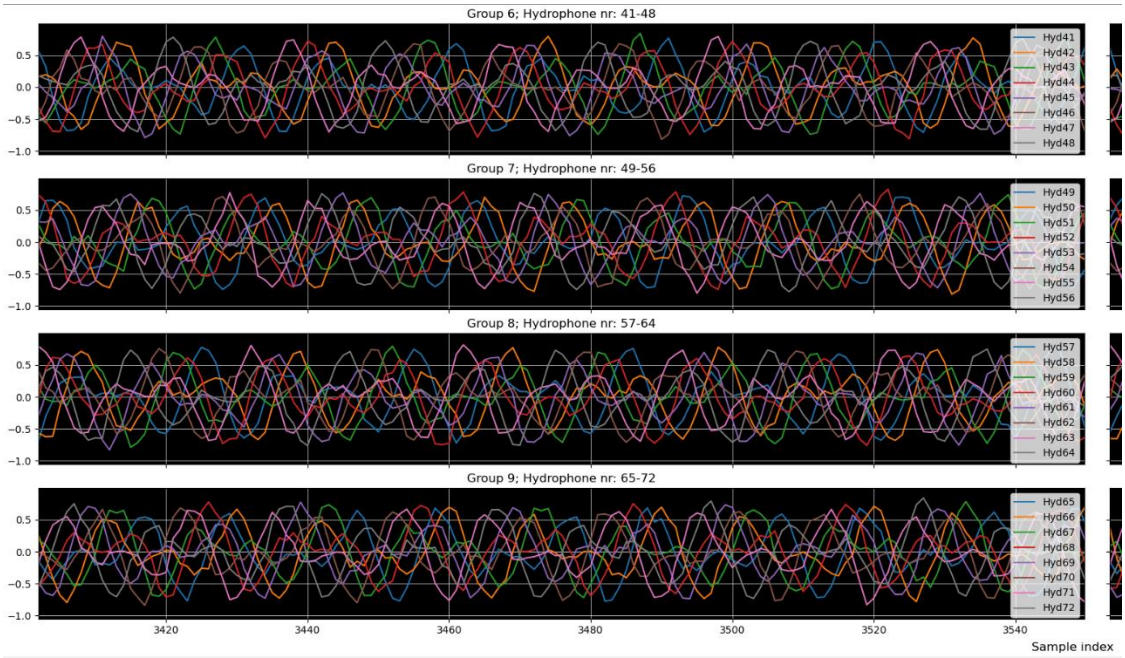

Fig 5.16: Rådata testsignal 3

Her ser vi et utsnitt av testsignal 3 som tilsvarer sampling indeksene fra 3400 til 3560. Som for testsignal 2, ser vi at amplituden varierer som følge av interferens. Nå når frekvensen i tillegg er ulik for de to signalene, er det nesten umulig å se noen form for periodisitet i det kombinerte signalet. Som for de to andre signalene, ser vi fortsatt effekten støy har på et slikt lydsignal. Til tross for hvor kaotisk rådataene ser ut, klarer likevel beamformeren å gjøre mening ut av kaoset?

```
##Beamformer

z = beamformer(y)
maxval = np.sort(z.flat)[-1]
secondmaxval = np.sort(z.flat)[-2]
thirdmaxval = np.sort(z.flat)[-3]
fourthmaxval = np.sort(z.flat)[-4]

maxpoint = np.where(z == maxval)
index1 = list(zip(maxpoint[0],maxpoint[1]))

secondmaxpoint = np.where(z == secondmaxval)
index2 = list(zip(secondmaxpoint[0],secondmaxpoint[1]))

thirdmaxpoint = np.where(z == thirdmaxval)
index3 = list(zip(thirdmaxpoint[0],thirdmaxpoint[1]))

fourthmaxpoint = np.where(z == fourthmaxval)
index4 = list(zip(fourthmaxpoint[0],fourthmaxpoint[1]))

print("Maximumsverdien tilhører element: ", index1[0])
print("Nest høyeste verdi tilhører element: ", index2[0])
print("Tredje høyeste verdi tilhører element: ", index3[0])
print("Fjerde høyeste verdi tilhører element: ", index4[0])
```

```
Maximumsverdien tilhører element:  (132, 546)
Nest høyeste verdi tilhører element:  (12, 7646)
Tredje høyeste verdi tilhører element:  (104, 7373)
Fjerde høyeste verdi tilhører element:  (40, 819)
```

Fig 5.17: Indeksen til maksimumspunktet

```
print(Xmat[40,819])
print(Ymat[40,819])

print(Xmat[132,546])
print(Ymat[132,546])
```

```
46.93441128969786
1220.4019775390625
107.89513430068399
813.601318359375
```

Fig 5.18: Indeks regnet om til vinkel og frekvens

For å verifisere at beamformeren fungerer må jeg også denne gangen lokalisere de to matriseelementetene som er størst. Det viser seg imidlertid at den nest høyeste og tredje høyeste verdien kommer av symmetrien vi diskuterte i seksjon 5.0.0, og kan derfor ses bort ifra. Vi ser derfor nærmere på det høyeste og fjerde høyeste punktet. Hvis vi undersøker Xmat og Ymat elementet med indeks (40, 819), kjenner vi igjen vinkelen og frekvensen vi ga dette signalet da det ble definert. Det samme gjelder for Xmat og Ymat verdiene for elementet med indeks (132, 546). Det er ingen informasjon om disse verdiene lagret i denne notebooken utenom i signalet vi overførte fra wave-filen, noe som tyder på at beamformeren klarer oppgaven den ble laget for å utføre.

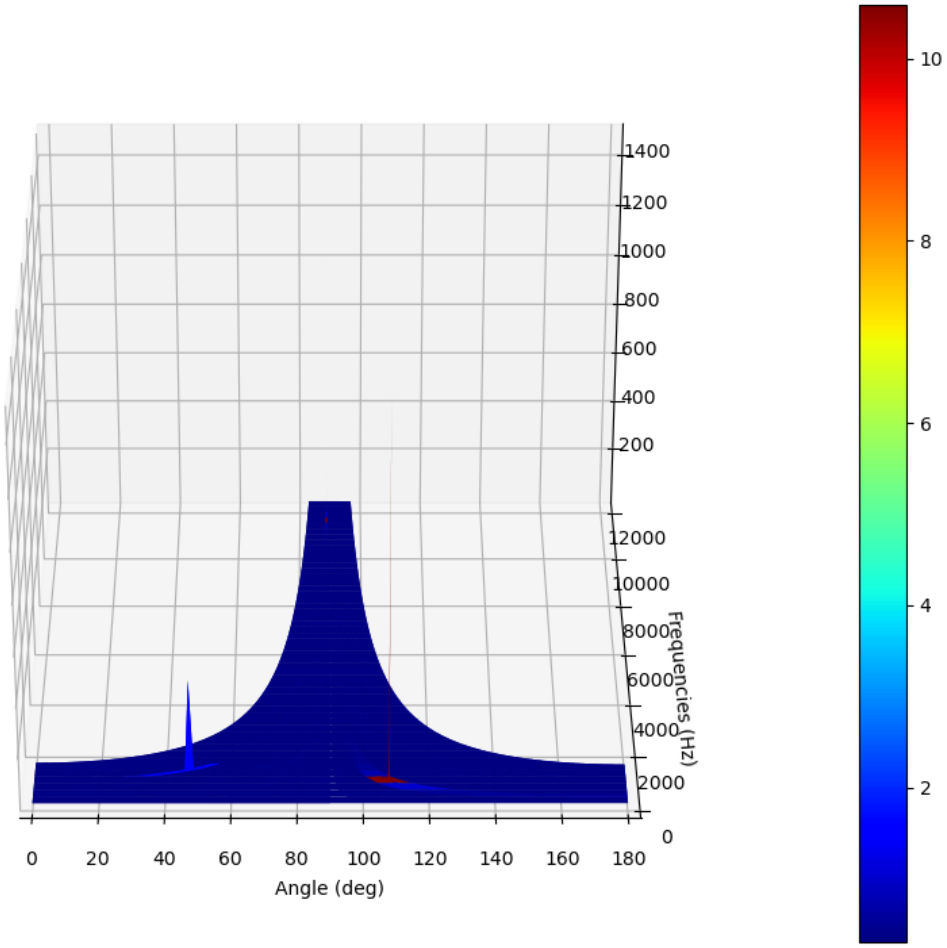

Fig 5.19: Beamforming plot

Her ser vi tydelig at plottet viser deteksjon for vinklene 47° og 108°. Dette minner mye mer om MATLAB utgaven av plottet til testsignal 1, som er det vi ville likt for alle plotene våre. Grunnen til dette har nok en gang med fargeskalaen å gjøre. Her spenner den nemlig et mye større intervall enn de to andre testsignalene, som tillater plottet å vise mye skarpere kontraster.

5.1 Drøfting

I denne oppgaven har jeg valgt å se bort fra mange egenskaper med lyd som ville bidratt til å gjøre analysen vanskeligere. Noen av disse egenskapene er for eksempel demping, refleksjon og refraksjon. Derfor tenkte jeg kort å gå gjennom hva hver av disse innebærer, og hvorfor de kompliserer analysen:

Demping er en egenskap som gjør at lydstyrken blir lavere desto lengre vekk fra kilden en kommer. Denne effekten påvirker også frekvensen og bølgelengden til signalet, hvor frekvensen blir lavere og bølgelengden lengre mens en beveger seg lengre unna. Resultatet er at vi kan gå glipp av et signal dersom

kilden er langt nok unna til at hydrofonene ikke merker forskjell på signaturene i signalet og tilfeldig bakgrunnsstøy. Dette kan forbedres enten ved å bevege seg nærmere signalet, eller gjøre antennen lengre. Her ser vi faktisk enda en fordel med at vi har antenneseksjoner som kan settes sammen, og at antennen er tynnere enn før. Vi kan nemlig bygge en antenne veldig lange ved å bare koble mange seksjoner sammen, uten at dette går på bekostning av plass eller vekt. Grunnen til at lange antenner er hensiktsmessig i denne sammenhengen, henger sammen med observasjonen vi gjorde oss av frekvenser og bølgelengder for menneskelig hørselsspenn, nemlig at bølgelengden ble 17 m allerede ved 20 Hz. Antennen jeg har fokusert på i oppgaven, består av to seksjoner og er 100 m lang. Altså kan vi observere nesten seks bølgelengder langs antennen. Etter hvert som frekvensen blir lavere, og bølgelengden lengre når vi til slutt et punkt hvor bølgelengden er lengre enn antennen. Og da får vi et problem. Selv med 144 hydrofoner klarer vi ikke å sample over en hel bølgelengde, og vi får ikke et troverdig bilde av signalet. Det er gjort mange forsøk innenfor DSP på å gjenskape signaler med sampler som tilsvarer mindre enn en bølgelengde. Et kjent eksempel er Burg algoritmen, som bruker minste kvadraters metode til å ekstrapolere signalet forover og bakover i tid. Effekten av algoritmen blir dårligere og dårligere desto mindre av signalet vi har sampler fra. Den sikreste måten er altså at antennen er lengre enn bølgelengden av signalet, og derfor er det nyttig at vi kan bygge den lengre etter behov.

Refleksjon handler om hvordan lyd oppfører seg i møte med faste gjenstander, som steiner eller fartøy. Lydbølgene blir da reflektert av gjenstanden og skifter dermed retning. Hvis gjenstanden er liten, kan lydbølgene skilles og rekombineres på andre siden. Da er det mulig at signalet interferer med seg selv, som fører til at informasjon kan gå tapt. Et annet fenomen som kan oppstå i grunt vann, eller i trange passasjer, er at lydbølgene blir reflektert flere ganger, og det kan virke som det er flere kopier av samme signal som kommer fra litt ulike retninger. Refraksjon oppstår når lydbølgene beveger seg fra et medium til et annet og skifter retning i prosessen. Dette er en følge av en naturlov som kalles Snell's lov, best kjent i forbindelse med bøying av lysbølger. Refraksjon kan også oppstå når lydbølgene passerer mellom lag i vannet med forskjellig temperatur, salinitet eller trykk. Effekten av både refleksjon og refraksjon er at lydsignalene vi observerer ikke nødvendigvis har beveget seg i en rett linje hele veien fra kilden.

En annen svakhet med antennen jeg har unnlatt å nevne, kommer av at antennen er endimensjonal. Vi har derfor kun muligheten til å vurdere

tidsforsinkelser i signalene i en retning, nemlig forover og bakover. Opp og ned er ikke noe problem ettersom antennen slepes så nærme overflaten at kilden praktisk talt enten er ved samme dybde, eller dypere enn antennen. Det vi ikke kan skille, er hvilken side av antennen kilden er på. Signaler som kommer inn fra samme vinkel på hver side av antennen, gir samme tidsforsinkelser mellom hydrofonene, og vi har ingen informasjon som kan skille de to sidene fra hverandre. Det er imidlertid en løsning tilgjengelig på dette problemet. Når slepefartøyet svinger, kommer antennen til å følge svingen ettersom den er elastisk. Nå er ikke lenger hydrofonene på en rett linje, og tidsforsinkelser fra hver side vil være annerledes. Med hjelp av kompassdataene fra NAS-nodene kan vi finne omtrentlige koordinater for antennen, og dermed hydrofonene. Grunnen til at jeg ikke har nevnt dette før, er at matematikken fort blir veldig komplisert når en i real-time skal parametrisere antennens krumning, finne hydrofonenes koordinater og kjøre disse dataene gjennom enda en beamformer for å få den økte sikkerheten i retningen til lydkilden. For ikke å snakke om at denne beamformeren også må være mer komplisert enn den jeg har jobbet med i oppgaven, i og med at den må ta hensyn hydrofonenes koordinater i to dimensjoner.

6.0 Konklusjon og veien videre

I denne oppgaven har vi sett hvor gode resultater selv en enkel beamformer kan bringe med seg. Beamformeren ble realisert gjennom bare å studere formelen for planbølger som treffer antennen, og manipulere denne for å finne en skjult sammenheng med retningsvinkelen til et signal, og videre hvordan denne er relatert til en deltafunksjon gjennom en Fourier analyse. Ettersom denne primitive beamformeren presterte så godt selv på signaler forurenset med støy, er det lett å skjønne hvorfor beamforming har blitt så utbredt innen DSP.

Arbeidet som er lagt ned i denne oppgaven er bare starten på et omfattende arbeid innenfor signalbehandlingen som skal benyttes i antenneprosjektet. Veien videre kommer først og fremst til å dreie seg om å implementere sikkerhetsmarginer som tar hensyn til de fysiske prinsippene jeg har valgt å utelate. I tillegg så kommer det til å bli utviklet en ny, mer robust beamformingsalgoritme som kan håndtere generelle problemstillinger som kan oppstå, ikke bare tilfellet jeg har sett på hvor antennen er rett og sammensatt av to seksjoner. Det kommer også til å bli utviklet mer omfattende programvare for plotting og analysering av data slik at det er lettere å manøvrere enn det jeg har kodet.

En annen ide vi skal satse på fremover er antenner med varierende avstand mellom hydrofonene. Poenget er at vi ønsker å lage et tett hydrofongrid langs antennen, men med en vri. Ikke alle disse gridpunktene kommer til å bli fylt med hydrofoner. Istedenfor kommer vi til å behandle disse punktene som om det er hydrofoner der, bare at de registrer null signal. Fordelen vi ønsker å oppnå med å benytte et tettere hydrofongrid er at aliasing i rommet for gridspacing d > $\frac{\lambda}{2}$, ref. seksjon 4.2, blir et mindre problem. Etter hvert som d blir mindre, vil nemlig frekvensene som fører til denne typen aliasing bli forskjøvet til høyere og høyere verdier, mye høyere enn de hadde vært om vi hadde hatt samme antall sensorer med uniform avstand. Dette er et eksempel på noe som heter «sparse array» på fagspråket, og er en teknikk som har fattet interesse i DSP miljøet de siste par tiårene. De fleste studiene gir veldig positive tilbakemeldinger på bruken av sparse arrays. Det finnes mange ulike typer som implementerer ulike fordelinger av sensorer i forhold til hull, samt hvilken type beamformer de bruker for å analysere dataene. Felles for alle er imidlertid at de kan oppnå nesten like god, eller like god presisjon som en antenne av

samme lengde med uniform avstand mellom sensorene. Dermed bidrar bruken av et sparse array også til å redusere kostnadene for å bygge antennen, samt driftskostnader i form av strømbehov og prosesseringstid, uten å gå på bekostning av presisjonen. (Patwari, 2021)

En annen utfordring som skal arbeides videre med, er relatert til noe som kalles «tracking». Dette går ut på å estimere frekvens og retningsvinkel som endrer seg, som derfor tilhører en lydkilde som beveger seg. For å gjøre dette, baserer vi oss av en estimeringsalgoritme kalt Kalman filtrering. Siden vi ikke kjenner bevegelsen til kilden, kan vi ikke estimere frekvensen og vinkelen ved hjelp av vanlig statisk analyse. Istedenfor antar vi at endringene skjer langsomt, og at støyen forbundet med hver måling, er konstant. Under disse antakelsene har vi et system som utvikler seg forutsigbart, og Kalman filteret vil være en iterativ prosess hvor man estimerer frekvensen og vinkelen for neste øyeblikk basert på verdiene vi har nå. Med denne prosessen kan vi spore endringen i disse verdiene over tid, og på bakgrunn av dette kunne avgjøre posisjonen og hastigheten til kilden. (Bang, 2018) En problemstilling som oppstår med denne tilnærmingen er hva som skjer hvis antakelsen om konstant støy ikke er holdbar? Da kommer ikke modellen til å levere gode resultater. Derfor er dette også noe som skal undersøkes nærmere for å finne teknikker og antakelser som takler varierende støy.

Før de siste bestemmelsene gjøres innenfor det digitale må imidlertid de første prototypene ferdigstilles mekanisk. Med prototyper her mener jeg alt fra individuelle komponenter til en hel antenneseksjon. Alle komponentene skal gjennom en rekke tester, enten alene, eller montert sammen med andre komponenter. Grunnen til at det gjøres på denne måten, er at det er lettere å lokalisere eventuelle feil og mangler før hele antennen er ferdig montert. I første omgang er det snakk om småskala tester som utføres i selskapets lokaler eller i nærområdet. Disse testene har to hensikter:

- Den første er feilsøking. Før antennen kan tas i bruk må vi være sikre på at alle komponenter virker som de skal, og at antennen oppfyller alle kravene den må. Dette innebærer for eksempel om den tåler de omgivelsene den kommer til å brukes i, eller om den er godt nok laget til å ikke lide store informasjonstap på grunn av egne støykilder. Mens disse testene blir gjennomført, kommer det også til å være et kunstig lydsignal i vannet som antennen skal måle. Kravet er da at antennen kan loggføre

kontinuerlig under hele testens lengde. Hvis dette ikke er tilfellet, tyder det på at det er en feil som må utbedres.

- Den andre hensikten er muligheten for å teste nye ideer og løsninger. Dette kan blant annet innebære andre deler eller orienteringen til komponentene. Dataene kan sammenlignes, og siden vi vet hvordan resultatet skal være, siden vi tester de ulike løsningene med samme syntetiske signal, så kan vi avgjøre hvilken løsning som er best. Her kan vi også finne ut hvor presise dataene våre er i et kontrollert miljø. Dette er viktig informasjon å ha når det skal tas avgjørelser om signalbehandlingen.

Etter disse innledende testene er det duket for fullskala tester. Nå monteres antenneseksjonene sammen til antenner av gitt lengde basert på trekkfartøy. Deretter skal antennene slepes på samme måte som de kommer til å bli i en reell situasjon, og vi får en endelig bekreftelse på om alle elementene fungerer som de skal. Her kan vi virkelig få testet egenskapene til antennene, og hvor mye ekstra informasjon vi klarer å få ut gjennom den digitale signalbehandlingen som gjennomføres.

Appendix A Koding

A.0 Generere testsignalene (Generating the wave files.py):

```
#!/usr/bin/env python
# coding: utf-8
# In[1]:

import numpy as np
import wave
import matplotlib.pyplot as plt
import cmath as cm

# Viktige funksjoner:
# In[2]:
##Få signalet på rett form
def cossignal(freq,phase):
    l = []
    for i in range(nchannels):
        s = np.cos(2*np.pi*freq*n-phase[i], dtype='float32')
        l.append(s)
        A = np.array(l)
        At = np.transpose(A)
    return At

##Convertere dataene til wave-fil

def convertInt32to24bitsBytearray(s):
    b = bytearray([int('%d'%(((s[i, j] & (0x0000ff << (8*k))) >> (8*k))))
    for i in range(s.shape[0]) for j in range(s.shape[1])
                for k in range(3)])
    return b

def convertFloat32ToInt32(s):
    return np.round(s * 0x7FFFFF).astype('int32') & 0xFFFFFF

# Generelle parametre:

# In[3]:

BytesPerSample = 3                  ##Antall byte per sample
NumSamples = 8192                   ##Anatall tidssampler
Fs = 12207                          ##Sampler/sekund
nchannels = 144                     ##Antall hydrofoner
c = 1500                            ##Lydhastigheten i vann
d = 0.5                             ##Avstanden mellom hydrofonene
```

```
n = np.arange(0,NumSamples,1)          ##Sample indeks liste
Listnchannels = np.arange(0,nchannels,1) ##Hydrofon/IFFT indeks liste

# Testsignal 1:
# In[4]:

##Enkelt cosinussignal
f0 = 1/20          ##Diskret tid frekvens
F0 = f0*Fs         ##Kontinuerlig tid frekvens
wavelength = c/F0   ##Bølgelengden til signalet
l = 10                                  ##IFFT indeksen hvor vi ønsker maks amplitude
phi = np.arccos(wavelength*l/(nchannels*d))          ##Retningsvinkelen
phase = (2*np.pi/wavelength)*np.cos(phi)*d*Listnchannels ##Tidsforsinkelse mellom hydrofonene

y = cossignal(f0,phase)              ##Rent signal
wn = np.random.normal(0,0.05,y.shape) ##Tilfeldig støy
y = 0.75*y + wn   ##Signal + støy

y_n = convertFloat32ToInt32(y)
data = convertInt32to24bitsBytearray(y_n)

wavefile = wave.Wave_write('Testsignal_1.wav')
wavefile.setnchannels(nchannels)
wavefile.setnframes(NumSamples)
wavefile.setsampwidth(BytesPerSample)
wavefile.setframerate(Fs)
wavefile.writeframes(data)
wavefile.close()

# In[5]:

print('Retningsvinkelen til testsignal 1 er: ', round(phi*180/np.pi),' grader')
print('Frekvensen er: ', F0, ' Hz')

# Testsignal 2:

# In[6]:

##To cosinussignal fra forskjellig retning

f0 = 1/20          ##Diskret tid frekvens
F0 = f0*Fs         ##Kontinuerlig tid frekvens
wavelength = c/F0   ##Bølgelengden til signalet
l1 = 17                                  ##IFFT indeksen hvor vi ønsker maks amplitude
phi1 = np.arccos(wavelength*l1/(nchannels*d))          ##Retningsvinkelen
phase1 = (2*np.pi/wavelength)*np.cos(phi1)*d*Listnchannels ##Tidsforsinkelse mellom
hydrofonene
y1_1 = cossignal(f0,phase1)
```

```
l2 = 116                                    ##IFFT indeksen hvor vi ønsker maks amplitude
phi2 = np.arccos(wavelength*(l2-nchannels)/(nchannels*d))  ##Retningsvinkelen
phase2 = (2*np.pi/wavelength)*np.cos(phi2)*d*Listnchannels ##Tidsforsinkelse mellom
hydrofonene
y1_2 = cossignal(f0,phase2)

y1 = 0.5*(y1_1 + y1_2)              ##Kombinert signal
wn1 = np.random.normal(0,0.05,y1.shape) ##Tilfeldig støy
y1 = 0.75*y1 + wn1  ##Signal + støy

y1_n = convertFloat32ToInt32(y1)
data1 = convertInt32to24bitsBytearray(y1_n)

wavefile = wave.Wave_write('Testsignal_2.wav')
wavefile.setnchannels(nchannels)
wavefile.setnframes(NumSamples)
wavefile.setsampwidth(BytesPerSample)
wavefile.setframerate(Fs)
wavefile.writeframes(data1)
wavefile.close()

# In[7]:

print('Retningsvinkelen til testsignal 2.1 er: ', round(phi1*180/np.pi),' grader')
print('Retningsvinkelen til testsignal 2.2 er: ', round(phi2*180/np.pi),' grader')
print('Frekvensen er lik for begge og har verdi: ', F0, ' Hz')

# Testsignal 3:

# In[8]:

##To cosinussignal fra forskjellig retning med forskjellig frekvens

f1 = 1/10          ##Diskret tid frekvens
F1 = f1*Fs         ##Kontinuerlig tid frekvens
wavelength1 = c/F1  ##Bølgelengden til signalet
l1 = 40                                     ##IFFT indeksen hvor vi ønsker maks amplitude
phi1 = np.arccos(wavelength1*l1/(nchannels*d))          ##Retningsvinkelen
phase1 = (2*np.pi/wavelength1)*np.cos(phi1)*d*Listnchannels   ##Tidsforsinkelse mellom
hydrofonene
y2_1 = cossignal(f1,phase1)

f2 = 1/15          ##Diskret tid frekvens
F2 = f2*Fs         ##Kontinuerlig tid frekvens
wavelength2 = c/F2   ##Bølgelengden til signalet
l2 = 132                                    ##IFFT indeksen hvor vi ønsker maks amplitude
phi2 = np.arccos(wavelength2*(l2-nchannels)/(nchannels*d))  ##Retningsvinkelen
```

```
phase2 = (2*np.pi/wavelength2)*np.cos(phi2)*d*Listnchannels ##Tidsforsinkelse mellom
hydrofonene
y2_2 = cossignal(f2,phase2)

y2 = 0.5*(y2_1 + y2_2)              ##Kombinert signal
wn2 = np.random.normal(0,0.05,y2.shape) ##Tilfeldig støy
y2 = 0.75*y2 + wn2  ##Signal + støy

y2_n = convertFloat32ToInt32(y2)
data2 = convertInt32to24bitsBytearray(y2_n)

wavefile = wave.Wave_write('Testsignal_3.wav')
wavefile.setnchannels(nchannels)
wavefile.setnframes(NumSamples)
wavefile.setsampwidth(BytesPerSample)
wavefile.setframerate(Fs)
wavefile.writeframes(data2)
wavefile.close()

# In[9]:

print('Retningsvinkelen til testsignal 3.1 er: ', round(phi1*180/np.pi),' grader')
print('Frekvensen til testsignal 3.1 er: ', round(F1), ' Hz')
print('Retningsvinkelen til testsignal 3.2 er: ', round(phi2*180/np.pi),' grader')
print('Frekvensen til testsignal 3.2 er: ', round(F2), ' Hz')
# In[ ]:
```

A.1 Analyse av testsignal 1 (Testsignal 1 analyse.py):

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

import numpy as np
import wave
import matplotlib
matplotlib.use('QT5Agg')
import matplotlib.pyplot as plt
from IPython import get_ipython
from PyQt5 import QtWidgets
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as NavigationToolbar
# Viktige funksjoner:

# In[2]:

##Conversion from file
```

```
def convert24bitsBytearrayToInt32(b, shape):
   s = np.array([[np.int32(b[3*(shape[1] * i + j)]) << 8          | np.int32(b[3*(shape[1] * i + j) + 1]) <<
16          | np.int32(b[3*(shape[1] * i + j) + 2]) << 24          for j in range(shape[1])]          for i in
range(shape[0])], dtype='int32')
   return s

def convertInt32toFloat32(s):
   return (s/0x7FFFFF00).astype('float32')

##Plotte rådata
class ScrollableWindow(QtWidgets.QMainWindow):
   def __init__(self, fig):
      self.qapp = QtWidgets.QApplication([])

      QtWidgets.QMainWindow.__init__(self)
      self.widget = QtWidgets.QWidget()
      self.setCentralWidget(self.widget)
      self.widget.setLayout(QtWidgets.QVBoxLayout())
      self.widget.layout().setContentsMargins(0,0,0,0)
      self.widget.layout().setSpacing(0)

      self.fig = fig
      self.canvas = FigureCanvas(self.fig)
      self.canvas.draw()
      self.scroll = QtWidgets.QScrollArea(self.widget)
      self.scroll.setWidget(self.canvas)

      self.nav = NavigationToolbar(self.canvas, self.widget)
      self.widget.layout().addWidget(self.nav)
      self.widget.layout().addWidget(self.scroll)

      self.show()
      exit(self.qapp.exec_())

def plotting2(wavdata,delay):
   l=[]
   for i in range(wavdata.shape[1]):
      l.append(wavdata[:,i])
   m=[]
   for i in range(0,len(l),8):
      m.append(l[i:i+8])
   fig, chg = plt.subplots(round(len(m)/2),2, figsize = (30,20), constrained_layout =
True,sharex=True,sharey=True)
   fig.suptitle('Plot of antenna-channels',fontsize = 16)
   fig.supxlabel('Sample index')
   fig.supylabel('Hydrophone levels(dB?)')
   for i in range(round(len(m)/2)):
```

```
        chg[i,0].set_facecolor('black')
        chg[i,0].set_title('Group ' + str(i+1) + '; Hydrophone nr: ' + str(i*8+1) + '-' + str((i+1)*8))
        chg[i,0].grid(color='white',linewidth = 0.5)
        chg[i,0].set_xlim(0,len(m[0][0]) + 11)
        k = int(i+(round(len(m))/2+1))
        chg[i,1].set_facecolor('black')
        chg[i,1].set_title('Group ' + str(k) + '; Hydrophone nr: ' + str((k-1)*8+1) + '-' + str(k*8))
        chg[i,1].grid(color='white',linewidth = 0.5)
        chg[i,1].set_xlim(0,len(m[0][0]) + 11)
    dom = np.arange(0,len(m[0][0]))
    for i in range(0,round(len(m)/2)):
        for k in range(len(m[i])):
            chg[i,0].plot(dom[(delay*k):],m[i][k][(delay*k):], label = 'Hyd' + str((k + 8*i)+1))
            chg[i,0].legend(loc="upper right")
            j = i + round(len(m)/2)
            chg[i,1].plot(dom[(delay*k):],m[j][k][(delay*k):],label = 'Hyd' + str((round(len(m)/2))*8 + k +
8*i + 1))
            chg[i,1].legend(loc="upper right")
    ScrollableWindow(fig)

##Beamformer

def beamformer(sign):
    Y = np.fft.fft(sign,axis=0)
    Y_t = Y.T
    return np.absolute(np.fft.ifft(Y_t,axis=0))

# Generelle parametre:

# In[3]:

NumSamples = 8192    ##Anatall tidssampler
sampleRate = 12207   ##Sampler/sekund
nchannels = 144      ##Antall hydrofoner
c = 1500             ##Lydfarten i vann [m/s]
d = 0.5              ##Felles avstand mellom hydrofonene

# In[4]:

##Overføre fra fil

wavefile = wave.Wave_read('Testsignal_1.wav')
data = wavefile.readframes(NumSamples)
y_n = convert24bitsBytearrayToInt32(data, (NumSamples, nchannels))
y = convertInt32toFloat32(y_n)

# In[ ]:
```

```
##Plotte rådata

plotting2(y,0)

# In[26]:

##Beamformer

z = beamformer(y)
maxpoint = np.where(z == np.max(z))
index = list(zip(maxpoint[0],maxpoint[1]))
print("Maximumsverdien tilhører element: ", (index[0],index[1]))

# In[25]:

##Generere MAP og freq

Freqs = np.arange(0,NumSamples,1)*sampleRate/NumSamples
ONE = np.ones(144)
Ymat = np.outer(ONE,Freqs)

l1 = np.arange(0,nchannels/2+1,1)
l2 = np.arange(nchannels/2+1,nchannels,1)-nchannels
l = list(np.append(l1,l2))

with np.errstate(divide='ignore', invalid='ignore'):
   wavelengths = []
   for x in Freqs:
      wavelengths.append(c/(x*nchannels*d))
MAP = np.outer(l,wavelengths)
Xmat = np.real(np.emath.arccos(MAP))*180/np.pi

print(Xmat[10,410])
print(Ymat[10,410])

# In[ ]:

##Beamform plot
fig = plt.figure(figsize = (14,9), dpi=100)
ax = plt.axes(projection = '3d')
surf = ax.plot_surface(Xmat,Ymat,z,cmap='jet')
ax.view_init(30,-90)
c = fig.colorbar(surf)
ax.set(xlim=(0,180),ylim=(0, sampleRate),xlabel='Angle (deg)',ylabel='Frequencies (Hz)')
plt.show()
# In[ ]:
```

A.2 Analyse av testsignal 2 (Testsignal 2 analyse.py):

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

import numpy as np
import wave
import matplotlib
matplotlib.use('QT5Agg')
import matplotlib.pyplot as plt
from IPython import get_ipython
from PyQt5 import QtWidgets
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as NavigationToolbar

# Viktige funksjoner:

# In[2]:

##Conversion from file

def convert24bitsBytearrayToInt32(b, shape):
    s = np.array([[np.int32(b[3*(shape[1] * i + j)]) << 8          | np.int32(b[3*(shape[1] * i + j) + 1]) <<
16          | np.int32(b[3*(shape[1] * i + j) + 2]) << 24          for j in range(shape[1])]          for i in
range(shape[0])], dtype='int32')
    return s

def convertInt32toFloat32(s):
    return (s/0x7FFFFF00).astype('float32')

##Plotte rådata
class ScrollableWindow(QtWidgets.QMainWindow):
    def __init__(self, fig):
        self.qapp = QtWidgets.QApplication([])

        QtWidgets.QMainWindow.__init__(self)
        self.widget = QtWidgets.QWidget()
        self.setCentralWidget(self.widget)
        self.widget.setLayout(QtWidgets.QVBoxLayout())
        self.widget.layout().setContentsMargins(0,0,0,0)
        self.widget.layout().setSpacing(0)

        self.fig = fig
        self.canvas = FigureCanvas(self.fig)
        self.canvas.draw()
```

```
        self.scroll = QtWidgets.QScrollArea(self.widget)
        self.scroll.setWidget(self.canvas)

        self.nav = NavigationToolbar(self.canvas, self.widget)
        self.widget.layout().addWidget(self.nav)
        self.widget.layout().addWidget(self.scroll)

        self.show()
        exit(self.qapp.exec_())

def plotting2(wavdata,delay):
    l=[]
    for i in range(wavdata.shape[1]):
        l.append(wavdata[:,i])
    m=[]
    for i in range(0,len(l),8):
        m.append(l[i:i+8])
    fig, chg = plt.subplots(round(len(m)/2),2, figsize = (30,20), constrained_layout =
True,sharex=True,sharey=True)
    fig.suptitle('Plot of antenna-channels',fontsize = 16)
    fig.supxlabel('Sample index')
    fig.supylabel('Hydrophone levels(dB?)')

    for i in range(round(len(m)/2)):
        chg[i,0].set_facecolor('black')
        chg[i,0].set_title('Group ' + str(i+1) + '; Hydrophone nr: ' + str(i*8+1) + '-' + str((i+1)*8))
        chg[i,0].grid(color='white',linewidth = 0.5)
        chg[i,0].set_xlim(0,len(m[0][0]) + 11)

        k = int(i+(round(len(m))/2+1))
        chg[i,1].set_facecolor('black')
        chg[i,1].set_title('Group ' + str(k) + '; Hydrophone nr: ' + str((k-1)*8+1) + '-' + str(k*8))
        chg[i,1].grid(color='white',linewidth = 0.5)
        chg[i,1].set_xlim(0,len(m[0][0]) + 11)

    dom = np.arange(0,len(m[0][0]))
    for i in range(0,round(len(m)/2)):
        for k in range(len(m[i])):
            chg[i,0].plot(dom[(delay*k):],m[i][k][(delay*k):], label = 'Hyd' + str((k + 8*i)+1))
            chg[i,0].legend(loc="upper right")

            j = i + round(len(m)/2)
            chg[i,1].plot(dom[(delay*k):],m[j][k][(delay*k):],label = 'Hyd' + str((round(len(m)/2))*8 + k +
8*i + 1))
            chg[i,1].legend(loc="upper right")

    ScrollableWindow(fig)
```

```
##Beamformer

def beamformer(sign):
   Y = np.fft.fft(sign,axis=0)
   Y_t = Y.T
   return np.absolute(np.fft.ifft(Y_t,axis=0))

# Generelle parametre:

# In[3]:

NumSamples = 8192    ##Anatall tidssampler
sampleRate = 12207   ##Sampler/sekund
nchannels = 144      ##Antall hydrofoner
c = 1500             ##Lydfarten i vann [m/s]
d = 0.5              ##Felles avstand mellom hydrofonene

# In[4]:

##Overføre fra fil

wavefile = wave.Wave_read('Testsignal_2.wav')
data = wavefile.readframes(NumSamples)
y_n = convert24bitsBytearrayToInt32(data, (NumSamples, nchannels))
y = convertInt32toFloat32(y_n)

# In[ ]:

##Plotte rådata

plotting2(y,0)

# In[5]:

##Beamformer

z = beamformer(y)
maxval = np.sort(z.flat)[-1]
secondmaxval = np.sort(z.flat)[-2]
thirdmaxval = np.sort(z.flat)[-3]

maxpoint = np.where(z == maxval)
index1 = list(zip(maxpoint[0],maxpoint[1]))

secondmaxpoint = np.where(z == secondmaxval)
index2 = list(zip(secondmaxpoint[0],secondmaxpoint[1]))

thirdmaxpoint = np.where(z == thirdmaxval)
```

```
index3 = list(zip(thirdmaxpoint[0],thirdmaxpoint[1]))

print("Maximumsverdien tilhører element: ", index1[0])
print("Nest høyeste verdi tilhører element: ", index2[0])
print("Tredje høyeste verdi tilhører element: ", index3[0])

# In[6]:

##Generere MAP og freq

Freqs = np.arange(0,NumSamples,1)*sampleRate/NumSamples
ONE = np.ones(144)

Ymat = np.outer(ONE,Freqs)

l1 = np.arange(0,nchannels/2+1,1)
l2 = np.arange(nchannels/2+1,nchannels,1)-nchannels
l = list(np.append(l1,l2))

with np.errstate(divide='ignore', invalid='ignore'):
   wavelengths = []
   for x in Freqs:
      wavelengths.append(c/(x*nchannels*d))

MAP = np.outer(l,wavelengths)
Xmat = np.real(np.emath.arccos(MAP))*180/np.pi

print(Xmat[116,410])
print(Ymat[116,410])
print(Xmat[17,410])
print(Ymat[17,410])

# In[ ]:

##Beamform plot

fig = plt.figure(figsize = (14,9), dpi=100)
ax = plt.axes(projection = '3d')
surf = ax.plot_surface(Xmat,Ymat,z,cmap='jet')
ax.view_init(30,-90)
c = fig.colorbar(surf)
ax.set(xlim=(0,180),ylim=(0, sampleRate),xlabel='Angle (deg)',ylabel='Frequencies (Hz)')
plt.show()
# In[ ]:
```

A.3 Analyse av testsignal 3 (Testsignal 3 analyse.py):

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

import numpy as np
import wave
import matplotlib
matplotlib.use('QT5Agg')
import matplotlib.pyplot as plt
from IPython import get_ipython
from PyQt5 import QtWidgets
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as NavigationToolbar

# Viktige funksjoner:

# In[2]:

##Conversion from file

def convert24bitsBytearrayToInt32(b, shape):
    s = np.array([[np.int32(b[3*(shape[1] * i + j)]) << 8           | np.int32(b[3*(shape[1] * i + j) + 1]) <<
16          | np.int32(b[3*(shape[1] * i + j) + 2]) << 24          for j in range(shape[1])]          for i in
range(shape[0])], dtype='int32')
    return s

def convertInt32toFloat32(s):
    return (s/0x7FFFFF00).astype('float32')

##Plotte rådata
class ScrollableWindow(QtWidgets.QMainWindow):
    def __init__(self, fig):
        self.qapp = QtWidgets.QApplication([])

        QtWidgets.QMainWindow.__init__(self)
        self.widget = QtWidgets.QWidget()
        self.setCentralWidget(self.widget)
        self.widget.setLayout(QtWidgets.QVBoxLayout())
        self.widget.layout().setContentsMargins(0,0,0,0)
        self.widget.layout().setSpacing(0)

        self.fig = fig
        self.canvas = FigureCanvas(self.fig)
        self.canvas.draw()
```

```
        self.scroll = QtWidgets.QScrollArea(self.widget)
        self.scroll.setWidget(self.canvas)

        self.nav = NavigationToolbar(self.canvas, self.widget)
        self.widget.layout().addWidget(self.nav)
        self.widget.layout().addWidget(self.scroll)

        self.show()
        exit(self.qapp.exec_())

def plotting2(wavdata,delay):
    l=[]
    for i in range(wavdata.shape[1]):
        l.append(wavdata[:,i])
    m=[]
    for i in range(0,len(l),8):
        m.append(l[i:i+8])
    fig, chg = plt.subplots(round(len(m)/2),2, figsize = (30,20), constrained_layout =
True,sharex=True,sharey=True)
    fig.suptitle('Plot of antenna-channels',fontsize = 16)
    fig.supxlabel('Sample index')
    fig.supylabel('Hydrophone levels(dB?)')

    for i in range(round(len(m)/2)):
        chg[i,0].set_facecolor('black')
        chg[i,0].set_title('Group ' + str(i+1) + '; Hydrophone nr: ' + str(i*8+1) + '-' + str((i+1)*8))
        chg[i,0].grid(color='white',linewidth = 0.5)
        chg[i,0].set_xlim(0,len(m[0][0]) + 11)

        k = int(i+(round(len(m))/2+1))
        chg[i,1].set_facecolor('black')
        chg[i,1].set_title('Group ' + str(k) + '; Hydrophone nr: ' + str((k-1)*8+1) + '-' + str(k*8))
        chg[i,1].grid(color='white',linewidth = 0.5)
        chg[i,1].set_xlim(0,len(m[0][0]) + 11)

    dom = np.arange(0,len(m[0][0]))
    for i in range(0,round(len(m)/2)):
        for k in range(len(m[i])):
            chg[i,0].plot(dom[(delay*k):],m[i][k][(delay*k):], label = 'Hyd' + str((k + 8*i)+1))
            chg[i,0].legend(loc="upper right")

            j = i + round(len(m)/2)
            chg[i,1].plot(dom[(delay*k):],m[j][k][(delay*k):],label = 'Hyd' + str((round(len(m)/2))*8 + k +
8*i + 1))
            chg[i,1].legend(loc="upper right")

    ScrollableWindow(fig)
```

```
##Beamformer

def beamformer(sign):
   Y = np.fft.fft(sign,axis=0)
   Y_t = Y.T
   return np.absolute(np.fft.ifft(Y_t,axis=0))

# Generelle parametre:

# In[3]:

NumSamples = 8192    ##Anatall tidssampler
sampleRate = 12207   ##Sampler/sekund
nchannels = 144      ##Antall hydrofoner
c = 1500             ##Lydfarten i vann [m/s]
d = 0.5              ##Felles avstand mellom hydrofonene

# In[4]:

##Overføre fra fil

wavefile = wave.Wave_read('Testsignal_3.wav')
data = wavefile.readframes(NumSamples)
y_n = convert24bitsBytearrayToInt32(data, (NumSamples, nchannels))
y = convertInt32toFloat32(y_n)

# In[ ]:

##Plotte rådata

plotting2(y,0)

# In[5]:

##Beamformer

z = beamformer(y)
maxval = np.sort(z.flat)[-1]
secondmaxval = np.sort(z.flat)[-2]
thirdmaxval = np.sort(z.flat)[-3]
fourthmaxval = np.sort(z.flat)[-4]

maxpoint = np.where(z == maxval)
index1 = list(zip(maxpoint[0],maxpoint[1]))

secondmaxpoint = np.where(z == secondmaxval)
index2 = list(zip(secondmaxpoint[0],secondmaxpoint[1]))
```

```
thirdmaxpoint = np.where(z == thirdmaxval)
index3 = list(zip(thirdmaxpoint[0],thirdmaxpoint[1]))

fourthmaxpoint = np.where(z == fourthmaxval)
index4 = list(zip(fourthmaxpoint[0],fourthmaxpoint[1]))

print("Maximumsverdien tilhører element: ", index1[0])
print("Nest høyeste verdi tilhører element: ", index2[0])
print("Tredje høyeste verdi tilhører element: ", index3[0])
print("Fjerde høyeste verdi tilhører element: ", index4[0])

# In[6]:

##Generere MAP og freq

Freqs = np.arange(0,NumSamples,1)*sampleRate/NumSamples
ONE = np.ones(144)

Ymat = np.outer(ONE,Freqs)

l1 = np.arange(0,nchannels/2+1,1)
l2 = np.arange(nchannels/2+1,nchannels,1)-nchannels
l = list(np.append(l1,l2))

with np.errstate(divide='ignore', invalid='ignore'):
   wavelengths = []
   for x in Freqs:
      wavelengths.append(c/(x*nchannels*d))

MAP = np.outer(l,wavelengths)
Xmat = np.real(np.emath.arccos(MAP))*180/np.pi

print(Xmat[40,819])
print(Ymat[40,819])
print(Xmat[132,546])
print(Ymat[132,546])

# In[ ]:

##Beamform plot
fig = plt.figure(figsize = (14,9), dpi=100)
ax = plt.axes(projection = '3d')
surf = ax.plot_surface(Xmat,Ymat,z,cmap='jet')
ax.view_init(30,-90)
c = fig.colorbar(surf)
ax.set(xlim=(0,180),ylim=(0, sampleRate),xlabel='Angle (deg)',ylabel='Frequencies (Hz)')
plt.show()
# In[ ]:
```

Appendix B Formler

1) $G = 20 \log_{10}(M_v)$
2) $\nabla^2 p = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}$
3) $p(\mathbf{r},t) = Ae^{i(\omega t - \boldsymbol{kr})}$
4) $k = \frac{2\pi}{\lambda} = \frac{\omega}{c}$
5) $p(x,t) = Ae^{i(\omega t - kx\cos\Phi)}$
6) $e^{i\Phi} = \cos\Phi + i\sin\Phi$
7) $x(t) = \sum_{k=-\infty}^{\infty} c_k e^{i\omega kt}$
8) $X(\omega) = \frac{1}{P}\int_0^P x(t)e^{-i\omega kt}\,dt$
9) $X[k] = DFT(x[n]) = \sum_{n=0}^{N-1} x[n]e^{-i\frac{2\pi kn}{N}}$
10) $x[n] = IDFT(X[k]) = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{i\frac{2\pi kn}{N}}$
11) $x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{i\omega t}\,d\omega = \frac{1}{2\pi}\int_{-2\pi B}^{2\pi B} X(\omega)e^{i\omega t}\,d\omega$
12) $x[\frac{n}{2B}] = \frac{1}{2\pi}\int_{-2\pi B}^{2\pi B} X(\omega)e^{i\omega\frac{n}{2B}}\,d\omega$
13) $SNR = S\text{-}N = S_L\text{-}T_L\text{-}N$
14) $Ae^{i(\omega t - kx\cos\Phi)} = \left(Ae^{-ikx\cos\Phi}\right)e^{i\omega t}$
15) $p[m] = Ae^{-ikmd\cos\Phi}$
16) $kd\cos\Phi = 2\pi(l+uN)/N$
17) $p[m] = Ae^{-im\,2\pi(l+uN)/N} = Ae^{-im2\pi l/N}$
18) $p[m] = \frac{AN}{N} * \sum_{n=0}^{N-1} \delta[n-l]e^{-i\frac{2\pi mn}{N}}$
19) $IDFT(p[m]) = AN\,\delta[n-l]$
20) $\Phi = \arccos(\frac{2\pi(l+uN)}{kdN}) = \arccos(\frac{\lambda(l+uN)}{dN})$
21) $-1 \le \frac{\lambda(l+uN)}{dN} \le 1$
22) $-\frac{d}{\lambda} - \frac{l}{N} \le u \le \frac{d}{\lambda} - \frac{l}{N}$
23) $u_{max} - u_{min} \le 1 \rightarrow (\frac{d}{\lambda} - \frac{l}{N}) - (-\frac{d}{\lambda} - \frac{l}{N}) \le 1 \rightarrow d \le \frac{\lambda}{2}$
24) $u = 0: 0 \le \frac{d}{\lambda} - \frac{l}{N} \rightarrow l \le N\frac{d}{\lambda} \le \frac{N}{2}$ og $\Phi = \arccos(\frac{\lambda l}{dN})$
25) $u = -1: -\frac{d}{\lambda} - \frac{l}{N} \le -1 \rightarrow l \ge (-\frac{d}{\lambda} + 1)N \ge \frac{N}{2}$ og $\Phi = \arccos(\frac{\lambda(l-N)}{dN})$
26) $s = \cos(2\pi fn - phase)$
27) $q(x,t) = p^*(x,t) = Ae^{-i(\omega t - kx\cos\Phi)}$
28) $p(x,t) + q(x,t) = 2\cos(\omega t - kx\cos\Phi)$

29) $T\omega = 2\pi(m+vM)/M$

30) $p(x,t) = e^{i(\omega t - kx\cos\Phi)} = Ae^{-ikx\cos\Phi}e^{i2\pi\frac{(m+vM)}{M}n}$

31) $q(x,t) = e^{-i(\omega t - kx\cos\Phi)} = Ae^{ikx\cos\Phi}e^{-i2\pi\frac{(m+vM)}{M}n}$

32) $p(x,t) = Ae^{-ikx\cos\Phi}e^{i2\pi\frac{m}{M}n}$

33) $q(x,t) = Ae^{ikx\cos\Phi}e^{i2\pi\frac{(M-m)}{M}n}$

Appendix C Bibliografi

Bang, Y. K. (2018, November 5). *IntechOpen.* Hentet fra https://www.intechopen.com/chapters/63164

Duxbury, A. C. (u.d.). *Britannica.* Hentet fra www.britannica.com: https://www.britannica.com/science/seawater/Acoustic-properties

Efstathiou, P. C. (u.d.). *Signal sampling - Nyquist-Shannon theorem.* Hentet fra http://195.134.76.37/applets/AppletNyquist/Appl_Nyquist2.html#:~:text=The%20answer%20is%20given%20by,of%20its%20highest%20frequency%20component.

IEEE Signal Processing Soceiety. (1998). Hentet fra https://signalprocessingsociety.org/uploads/history/history.pdf

National Physics Laboratory. (2018). *National Physics Laboratory.* Hentet fra http://resource.npl.co.uk/acoustics/techguides/concepts/sen.html#:~:text=Hydrophone%20sensitihttp://resource.npl.co.uk/acoustics/techguides/concepts/sen.html#:~:text=Hydrophone%20sensitivity%20is%20the%20ratio%20of%20its%20output,linear%20value%20of%20the

Patwari, A. (2021, September 2). *IntechOpen.* Hentet fra https://www.intechopen.com/chapters/78401

Shannon, C. E. (1949, Januar). Communication in the Precence of Noise. *Proceedings of the Institute of Radio Engineers*, ss. 10-21. Hentet fra https://pure.mpg.de/rest/items/item_2383164_3/component/file_2383163/content

Siemens Digital Industries Software. (2020, 7 29). Hentet fra https://community.sw.siemens.com/s/article/sound-fields-free-versus-diffuse-field-near-versus-far-field

Stillwell, J. C. (u.d.). *History of Analysis.* Hentet fra Britannica: https://www.britannica.com/science/analysis-mathematics/Rebuilding-the-foundations

Szabo, T. L. (2014). *ScienceDirect.* Hentet fra https://www.sciencedirect.com/topics/engineering/hydrophones

Torre, C. G. (2014, August). Hentet fra The wave equation in 3 dimensions: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1009&context=foundation_wave