Maximum Likelihood basert estimering av frekvens og fase til sinussignal.

RAPPORT

Jakob Vahlin & Andreas Horpedal Bugge

Fakultet for informasjonsteknologi og elektroteknikk
Norges teknisk-naturvitenskapelige universitet

Sammendrag

Frekvensen ω og fasen ϕ til et sinussignal påført støy har ved hjelp av to metoder blitt estimert. For å bedømme kvaliteten på estimatene $\hat{\omega}$ og $\hat{\phi}$ har resultatene blitt sammenlignet med CRLB for de to parameterne samt avvik fra faktisk verdi. Metode a baserer seg på FFT-basert ML-estimering og det er beregnet estimater for ulike FFT-lengder ved ulike SNR. Metode b baserer seg på en kortere FFT-lengde finjustert med et numerisk søk.

Sammenlikningen av variansen til $\hat{\omega}$ og $\hat{\phi}$ med CRLB, viser at metode a gir svært gode estimatorer for de lengre FFT'ene. Resultatene fra metode b viser at en slik finjustering kan være et godt alternativ dersom man er villig til å ofre estimatorens ytelse mot mer effektiv beregning.

Innhold

1 Innledning

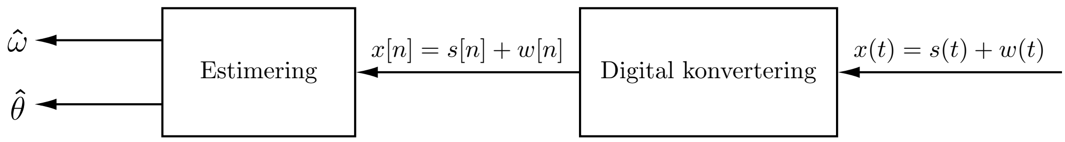
Sinusformede signaler har en sentral signifikans i mange teknologiske anvendelser. I mange tilfeller, f.eks i ulike kommunikasjonssystemer, mottas et kompleks signal med kjent sinusform som av ulike årsaker er påført støy. Det totale signalet $x(t)$ som mottas kan uttrykkes ved

$$x(t) = Ae^{i(\omega_0 t+\phi)} + w(t) \tag{1}$$

hvor $w(t)$ er kompleks hvitt gaussisk støy. Ofte er en eller flere av parametrene A, ω, ϕ ukjente. For å hente ut den ønskede informasjonen fra singalet må de følgelig *estimeres*. I denne rapporten vil sinusformede signaler på formen gitt i (1) med kjent amplitude A bli betraktet og vinkelen ω og fasen ϕ signalet vil bli forsøkt estimert. Som estimator vil *Maximum Likelihood Estimator* (MLE) bli brukt. Estimatene $\hat{\omega}$ og $\hat{\phi}$ sin varians, hhv $\sigma^2_{\hat{\omega}}$ og $\sigma^2_{\hat{\phi}}$ vil bli sammenliknet med sine respektive nedre grenser, gitt av *Cramer Raos Lower Bound* (CRLB) for ulike verdier av signalets "Signal-til-støy forhold" (SNR) for å evaluere til hvilken grad MLE estimatoren er *effektiv*. Estimatene $\hat{\omega}$ og $\hat{\phi}$ vil bli beregnet digitalt og følgelig er det det samplede signalet som betraktes. Dette er gitt ved

$$\begin{aligned} x[n] &= Ae^{i(\omega_0 nT+\phi)} + w[n] \\ &= s[n] + w[n] \end{aligned} \tag{2}$$

hvor n representerer sample nr n og T er samplingsperioden. Et blokkskjema for som oppsummerer estimeringsprossesen rapporten tar for seg er vist Figur 1.



Figur 1: Blokkskjema representasjon av estimeringsprossesen.

I rapporten vil nødvendig teori om MLE og CRLB, både genrerelt og spesifikt signaler på formen (2), bli presentert i kapittel 2. Oppgaven introdusert ovenfor vil blir presentert i større detalj i kapittel 3. Videre vil den digitale implementeringen gjennomgås i kapittel 4 før resultatene og en diskusjon av dette blir gjort i kapittel 5 og kapittel 6.

2 Teori

I dette kapittelet presenteres nødvendig teori relatert til *Maximum Likelihood Estimators* og Cramer Rao Lower Bound for å forstå arbeidet gjort i prosjektet.

Cramer-Rao Lower Bound

Cramer-Rao Lower Bound (CRLB) er definert som den nedre grensen variansen til en forventningsrett estimator kan oppnå [1]. Gitt at observasjonene $\boldsymbol{x}$ er distribuert i henhold til sannsyneligtetthetsfunksjonen p($\boldsymbol{x}$;θ), samtidig som p($\boldsymbol{x}$;θ) tilfredsstiller

$$E[\frac{\partial \ \log \ p(\boldsymbol{x};\theta)}{\partial\theta}] = 0 \tag{3}$$

vil CRLB til den forventningsrette estimatoren $\hat{\theta}$ være definert som

$$var(\hat{\theta}) \geq \frac{1}{-E[\frac{\partial^2 \ \log \ p(\boldsymbol{x};\theta}{\partial\theta^2}]} = \frac{1}{E[(\frac{\partial \ \log \ p(\boldsymbol{x;\theta})}{\partial\theta})^2]} \tag{4}$$

I (4) beregnes forventningsverdien med hensyn til p($\boldsymbol{x}$;θ), og de partiellderiverte med hensyn til den sanne parameteren θ. Ettersom CRLB beskriver den minimale variansen en forventningsrett estimator kan oppnå, brukes det gjerne som et mål på maksimal ytelse til en estimator. Estimatorer som oppnår CRLB kalles effektive [2].

I dette tilfellet hvor frekvens, ω, og fase, ϕ, fra (1) er ukjent og skal estimeres, er CRLB for de respektive estimatene $\hat{\omega}$ og $\hat{\phi}$ gitt av

$$\mathrm{Var}(\hat{\omega}) \geq \frac{12\sigma^2}{A^2T^2N(N^2-1)} \tag{5}$$

$$\mathrm{Var}(\hat{\phi}) \geq \frac{12\sigma^2(n_0^2N + 2n_0P + Q)}{A^2N^2(N^2-1)} \tag{6}$$

Her er σ^2=Var($w_r[n]$)=Var($w_i[n]$), altså variansen til den reelle og komplekse delen til det samplede støysignalet $w[n]$, N er antall sampler, A er amplitude, T er samplingsperiode, n_0 er indeksen til første sample, mens P og Q er gitt av likningene (7) og (8) [3].

$$P = \frac{N(N-1)}{2} \tag{7}$$

$$Q = \frac{N(N-1)(2N-1)}{6} \tag{8}$$

Fra (5) og (6) ser man at for en fiksert og kjent N, n_0, T og A, avhenger CRLB for $\hat{\omega}$ og $\hat{\phi}$ kun av σ^2. σ^2 kan beregnes ved uttrykket for signal-støy-forhold (SNR), som i dette tilfellet er gitt av (9) [3].

$$\mathrm{SNR_{dB}} = \frac{A^2}{2\sigma^2} \tag{9}$$

Her er SNR gitt i dB. Ved å løse (9) med hensyn på σ^2 og gjøre om SNR fra dB er σ^2 gitt av

$$\sigma^2 = \frac{A^2}{2 \cdot 10^{\frac{SNR_{dB}}{10}}} \tag{10}$$

Fra (9) ser man at σ^2 minsker jo større verdien SNR er, som igjen fører til at verdien av CRLB av både $\hat{\omega}$ og $\hat{\phi}$ blir lavere. Sagt på en annen måte, betyr dette at den potensielle ytelsen til estimatorene vil være bedre for signaler der parameterne en ønsker å estimere dominerer, sammenlignet med signaler der støy er den domminerende faktoren.

Maximum Likelihood Estimator

Maximum likelihood estimering basere seg på maksimering av rimelighetsfunksjonen. Rimelighetsfunksjonen beskriver sannsyneligheten for å observere $\boldsymbol{x}$ for ulike verdier av parameteren vi ønsker å estimere, θ.

$$L(\theta|\boldsymbol{x}) = p(\boldsymbol{x};\theta) \tag{11}$$

Her er $\mathrm{L}(\theta|\boldsymbol{x})$ er logaritmen til reimelighetsfunskjonen. Ideen bak MLE er så å finne den verdien av θ som gir størst sannsynlighet for å gjøre en observasjon $\boldsymbol{x}$ [2]. MLE kan altså defineres som

$$\theta_{MLE} = \arg\max_{\theta} L(\theta|\boldsymbol{x}) \tag{12}$$

I dette tilfellet, hvor frekvensen ω og ϕ, fra fasen (1) er parameterne som skal estimeres, gir MLE at frekvensestimatet $\hat{\omega}$ er gitt av

$$\hat{\omega} = \arg\max_{\omega_0} |F(\omega_0)| \tag{13}$$

der $\mathrm{F}(\omega_0)$ er

$$F(\omega_0) = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-i\omega_0 nT} \tag{14}$$

Ettersom $\mathrm{F}(\omega_0)$ er den diskre Fourier-transformasjonen av observasjonene, $x[n]$, gir (13) at $\hat{\omega}$ tilsvarer den frekvensen som maksimerer den diskre Fourier-transformasjonen i (14). Denne frekvensen som maksimerer (14), kan finnes ved å beregne Fast-Fourier transformasjonen (FFT) av det samplede signalet $x[n]$. Med FFTen kjent vil MLE for $\hat{\omega}$ være uttrykt ved

$$\hat{\omega} = \frac{2\pi m'}{MT} \tag{15}$$

hvor M er lengden på FFT'en og m' er indeksen som maksimerer FFT'en. Med andre ord

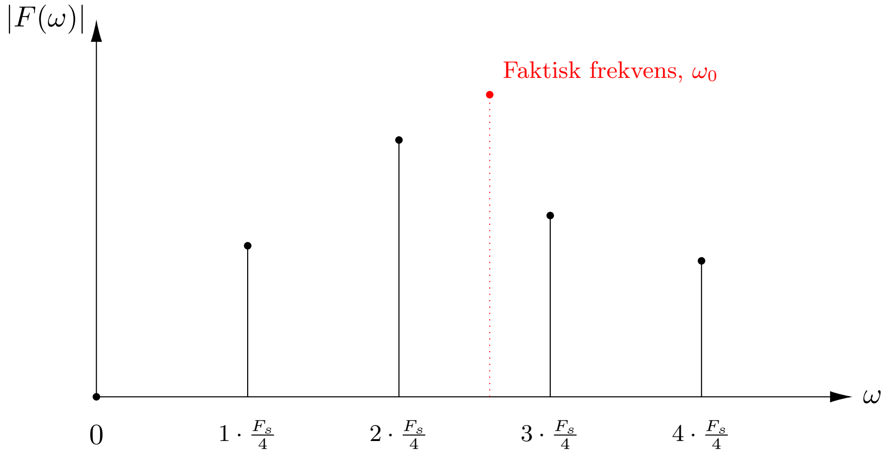
$$m' = \arg\max_{m} \mathrm{FFT_M}\{\mathbf{x}\} \tag{16}$$

Med $\hat{\omega}$ kjent kan fasen, $\hat{\phi}$, enkelt beregnes ved ligning (17) [3].

$$\hat{\phi} = \angle\left\{e^{-i\hat{\omega}n_0T}\cdot F(\hat{\omega})\right\} \tag{17}$$

Som man ser fra ligning (17), beregnes $\hat{\phi}$ basert på estimatet gjort av frekvensen $\hat{\omega}$. Av den grunn vil et dårlig estimat av $\hat{\omega}$ føre til et dårlig estimat av $\hat{\phi}$. For FFT-basert ML estimering avhenger kvaliten til estimatene av, som nevnt over, SNR i tillegg til lengden av FFT'en.

En diskret fouriertransform med lengde N deler frekvensspektert sitt inn i N forskjellige frekvenser, hver med en avstand $\frac{F_S}{N}$ til neste "frekvensnabo". Følgelig er det ikke gitt at den faktiske frekvensen til signalet blir inkludert i spekteret til transformen. Dette er illustrert i Figur 2. Med det vil en større N gi et frekvensspekter som inkluderer flere frekvenser.



Figur 2: Frekvensspekter til en DFT med lengde 4. Som marker vil ikke en diskret Fourier transform plukke ut *akkurat* den frekvensen som gir maksimum i transformen. Hvor godt den treffer avhenger av lengden på transformen.

På samme måte vil også lengden, M, til FFT'en påvirke hvor nøyaktig det er mulig å identifisere indeksen m' fra ligning (16), der en større M vil gi større presisjon.

3 Oppgaven

Estimeringen av det komplekse sinussignalet beskrevet i kapittel 1 vil bli gjortpå to forskjellige måter, a) og b) som begge er basert på teorien beskrevet i kapittel 2. I dette kapittelet vil disse bli grundigere gjennomgått.

Metode a)

Som nevnt i kapittel 2 er en ML-estimator til frekvensen i signalet beskrevet i (1), den frekvensen som maksimerer absoluttveriden til signalets Fouriertransform. I metode a) estimeres frekvensen i henhold til likningene (15) og (16). Fasen vil bli beregnet ved å evaluere (17) med den esitmerte frekvensen.

Hvor godt estimatet blir vil avhenge av hvor mye støy som er tilført signalet, representert ved signalets støyforhold (SNR). For å undersøke hvordan ulike grader av støy påvirker estimatet vil estimatoren bli beregnet med SNR verdier (i decibel) fra -10 til 60, i 10dB steg. Hvor godt estimatet blir avhenger ikke bare av signalets støyforhold, men også av FFTens lengde, som diskutert i kapittel 2. Av den grunn vil FFTer med lengde $M = 2^k, k \in \{10, 12, 14, 16, 18, 20\}$ bli brukt for å undersøke hvordan FFT lengden påvirker estimatet.

Metode b)

Til tross for at en radix-2 FFT er basert på en effektiv algoritme tar det likevel lang tid å bergene store FFTer. Følgelig er estimater basert på lange FFTer, som f.eks 2^{20}, ikke gjennomførbar i praktiske applikasjoner. En mer kostandseffektiv måte er å ta utgangspunkt i en kortere FFT og finjustere dette estimatet ved å numerisk søke etter den frekvensen som generer et signal som igjen generer en FFT som er likest FFTen til det opprinnelige estimatet. I metode b) vil det bli tatt utgangspunkt i estimatet basert på en FFT med lengde 2^{10}, som i utgangspunktet vil gi relativt stor feil på grunn av sin korte lengde. Videre vil et "rent" sinussignal, uten støy, genereres og det vil bli numerisk søkt etter den frekvensen som minimerer det gjennomsnittlige kvadratiske avviket av de to FFTene. Matematisk kan det nye estimatet $\hat{\omega}_{new}$ utrykkes som

$$\hat{\omega}_{\text{new}} = \arg\min_{\omega} \left\{ \Big(|F_{\text{original}}(\hat{\omega}_{\text{original}})| - |F_{\text{new}}(\omega)| \Big)^2 \right\} \tag{18}$$

hvor $F_{\text{original}}(\omega)$ og $F_{\text{new}}(\omega)$ er transformene til hhv det opprinnelig estimatet basert på signal med støy og det nye "rene" signalet.

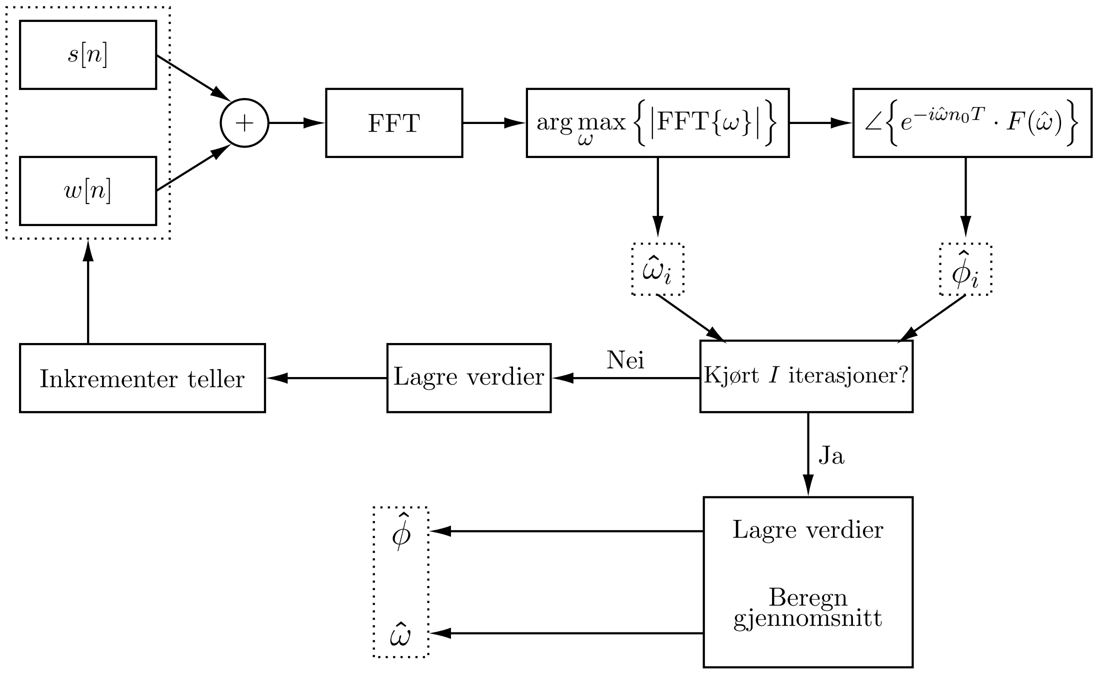
4 Implementering

I implementeringen testes ytelsen til ML-estimatoren ved å generere et kompleks sinussigal tilført støy på formen gitt ved (1). I simuleringene brukes en amplitude $A = 1$, en vinkelfrekvens $\omega_0 = 2\pi f_0 = 2\pi \cdot 10^5$ og en fase $\phi = \frac{\pi}{8}$ radianer. Det blir generert kompleks hvit støy hvor real og imaginærdelen til støyen er gaussisk fordelt, $\mathrm{Re}\{w(t)\} = \mathrm{Im}\{w(t)\} \sim N(0, \sigma^2)$, hvor σ^2 beregnes fra det spesifiserte støyforholdet ved (10). Signalet blir samplet til formen gitt i (2) med en samplingsperiode $T = 10^{-6}$. Dette innfrir Nyqvists samplingsteorem. Første sample, n_0, blir gjort ved $n_0 = -256$.

Både signalet, med og uten støy, samt estimeringen av signalets fase og vinkel blir implementert i Python. Python bibliotekene numpy, cmath og statistics blir brukt til å gjøre matematiske beregninger. I tilegg benyttes funksjonen scipy.optimize.minimize() til å gjennomføre et numerisk søk i metode b). Koden er inkludert i Vedlegg C, Vedlegg D og Vedlegg E. Den interesserte leser, som ønsker å kjøre kode selv anbefales å sjekke dokumentasjon og kildekode på GitHub [4].

Metode a)

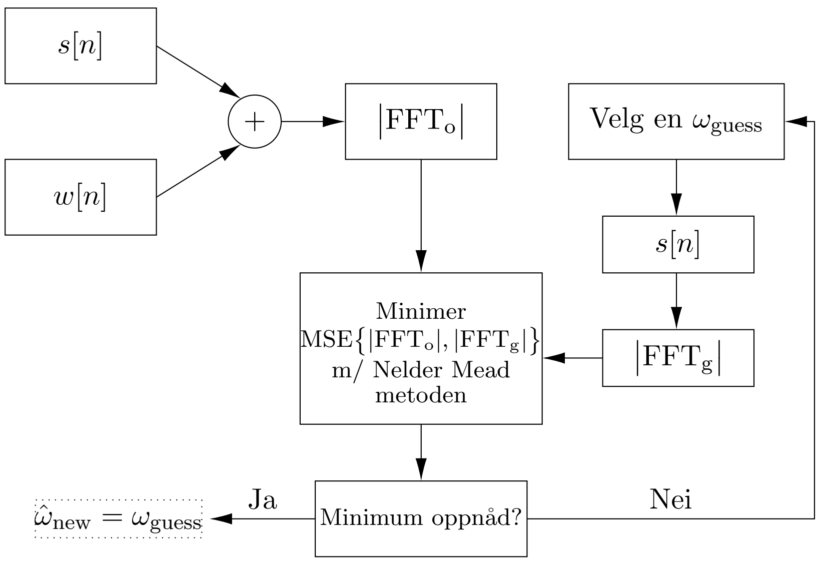
Et flytdiagram av kodeimplementasjonen som gjennomfører *Maximum Likelihood* estimatet av signalet ved metode a) beskrevet i kapittel 3 er vist i Figur 3. Det endelige estimatet for fasen og vinklene er basert på et gjennomsnitt av 1000 iterasjoner. For hver iterasjon genereres det samme deterministiske signalet $s[n]$, mens $w[n]$ er stokastisk og vil følgelig ha ulik verdi for hver iterasjon, i henhold til sin gaussiske sannsynlighetsfordeling.



Figur 3: Flyskjema for koden som gjennomfører ML estimatet.

Metode b)

Et flytdiagram av kodeimplemenasjon som gjennomfører finjusteringen av det opprinnelige estimatet basert på en FFT med lengde 2^{10} er vist i Figur 4. Først genereres et kompleks sinussignal påført støy med frekvens $\omega = \omega_0$. Signalets diskrete fourier transform med lengde 2^{10}, $\mathrm{FFT_o}$, blir beregnet og tatt absoluttverdien av. I tillegg genereres et rent kompleks sinussignal, *uten* støy, med frekvens ω_{guess}. Hensikten er å finne den frekvensen som gir en fouriertransform, $\mathrm{FFT_g}$ som er likest mulig fouriertransformen det opprinnelige estimatet er basert på. Matematisk blir dette gjort ved å finne den frekvensen, ω_{guess}, som minimerer det gjennomsnittlige kvadratiske avvik mellom $\mathrm{FFT_o}$ og $\mathrm{FFT_g}$. Minimeringen blir gjort ved å numerisk søke etter frekvensen som minimerer gjennomsnittlig kvadratisk avvik. Til dette blir *Nelder-Mead* algoritmen brukt. Deltajene ved denne algoritmen er utenfor omfanget til denne rapporten og er følgelig utelatt. Når Nelder-Mead algoritmen har funnet den frekvensen som minimerer avviket veøges denne frekvensensom det nye estimatet.



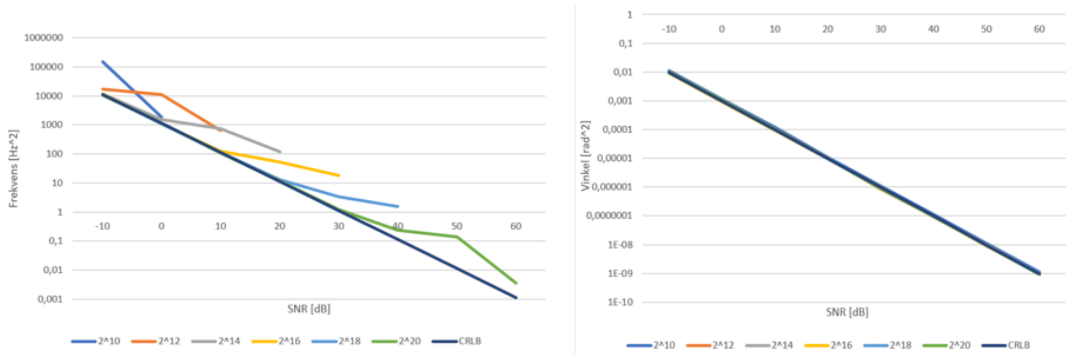
Figur 4: Flytskjema for koden som finjusterer estimatet. Detaljene ved Nelder Mead algoritmen er utelatt.

5 Resultater

Metode a)

Resultatene fra den FFT-baserte ML-estimeringen er vist i tabell 2 og tabell 3 i vedlegg A og B. Resultatene er basert på implementasjonen gjort i kapittel 4, og de endelige resultatene er basert på et gjennomsnitt av 1000 estimater. Tabell 2 viser den gjennomsnittlige estimerte frekvensen, gjennomsnittlig frekvensavvik, beregnet varians og CRLB for de forskjellige FFT-lengdene med ulikt SNR. Tabell 3 viser de samme parameterne beregnet for fasen $\hat{\phi}$.

Figur 5 viser til venstre variansen til $\hat{\omega}$ plottet for ulike FFT-lengder og SNR sammen med CRLB, mens det til høyre plottes det samme for $\hat{\phi}$. Fra Tabell 2 er det tydelig at den beregnete variansen til $\hat{\omega}$ er 0 for de kortere FFT-lengdene med høyt SNR. De er følgelig utelatt fra plottet i figuren, ettersom de ikke bidrar med noe informasjon. Dette skyldes antakelig at avstanden mellom hver indeks i FFT'en er så stor, samtidig som variansen til støyen er liten nok, at indeksen m' som maksimerer FFT'en blir den samme for hver iterasjon. Med det blir også den estimerte frekvensen den samme, og variansen til $\hat{\omega}$ som er et mål på spredning blir 0.

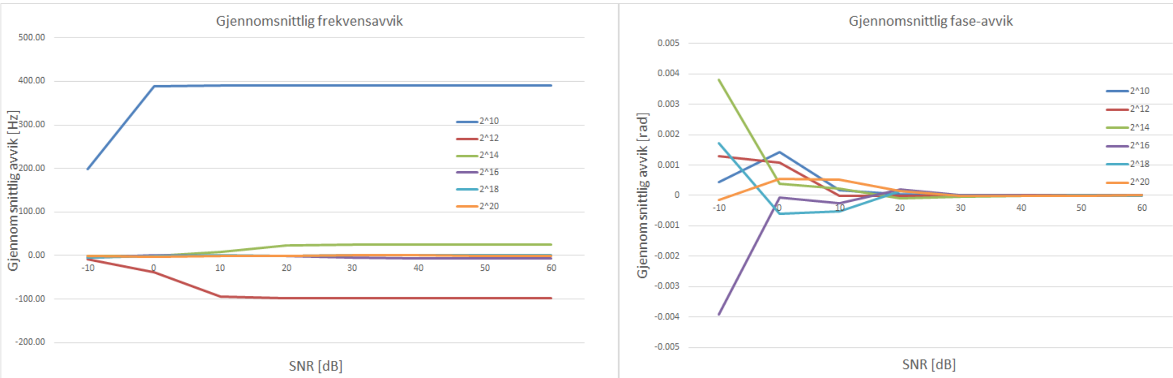


Figur 5: Til venstre: Varians til $\hat{\omega}$ og CRLB plottet for ulik FFT-lengde og SNR. Til Høyre: Varians til $\hat{\phi}$ og CRLB plottet for ulik FFT-lengde og SNR.

Ser man på resultatene for FFT-lengde 2^{20}, som gir de mest presise beregningene, er den estimerte variansen for både $\hat{\omega}$ og $\hat{\phi}$ hele tiden i nærheten av CRLB, og synker i likhet med CRLB for høyere SNR. For enkelte SNR er den beregnede variansen til og med lavere enn CRLB. Dette er fordi realiseringen av ML estimatoren ikke er forventningsrett, men har *bias*. Denne biasen skyldes, som diskutert i kapittel 2, kvantisering av frekvensspekteret estimatoren er basert på, som resulterer i at estimatet gir samme frekvens flere ganger. I et tilfelle med kontinuerlig spekter ville estimatet gi variende frekvenser. Av den grunn kan variansen til en kvantifisert estimator bli lavere enn CRLB, som er basert på et kontinuerlig tilfelle. Hadde en FFT med "uendelig" lengde blitt brukt, ville biasen blitt null.

Figur 6 viser avviket mellom de estimerte parameterene $\hat{\omega}$ og $\hat{\phi}$ og faktiske parameterne ω og ϕ. Fra plottene ser man at avviket er størst for beregningene gjort med kort FFT, og mindre for beregningene gjort med lang FFT. Dette er forventet ettersom lengden på FFT'en, som skrevet før, påvirker presisjonen i beregningene.

Fra Tabell 2 og Tabell 3 samt plottene i Figur 6 framgår det at til tross for at frekvensestimatets avvik og varians er hhv konstant og 0 ved høye SNR verider, er det faseestimatets avvik og varians hhv variende og ulikt 0 for tilsvarende SNR verdier. Dette skyldes at når $\hat{\phi}$ beregnes fra (17) benyttes et nytt datasett for hvert estimat. Ved bruk av korte FFT lengder vil ulike datasett ved høye SNR verdier gi samme frekvensestimat. Men fordi ulike datasett gir ulike tallverdier i sine fouriertransformer, vil estimatet $\hat{\phi}$ variere fra datasett til datasett, som i sin tur gir en varians i faseestimatet.



Figur 6: Simlueringsresultater for samtlige FFT lengder og samtlige SNR verider. Til venstre: Ulike FFT lengders gjennomsnittlige frekvensavvik fra ω_0 ved en gitt SNR. Til høyre: g samtlige SNR verider. Til venstre: Ulike FFT lengders gjennomsnittlige faseavvik fra ϕ ved en gitt SNR.

Metode b)

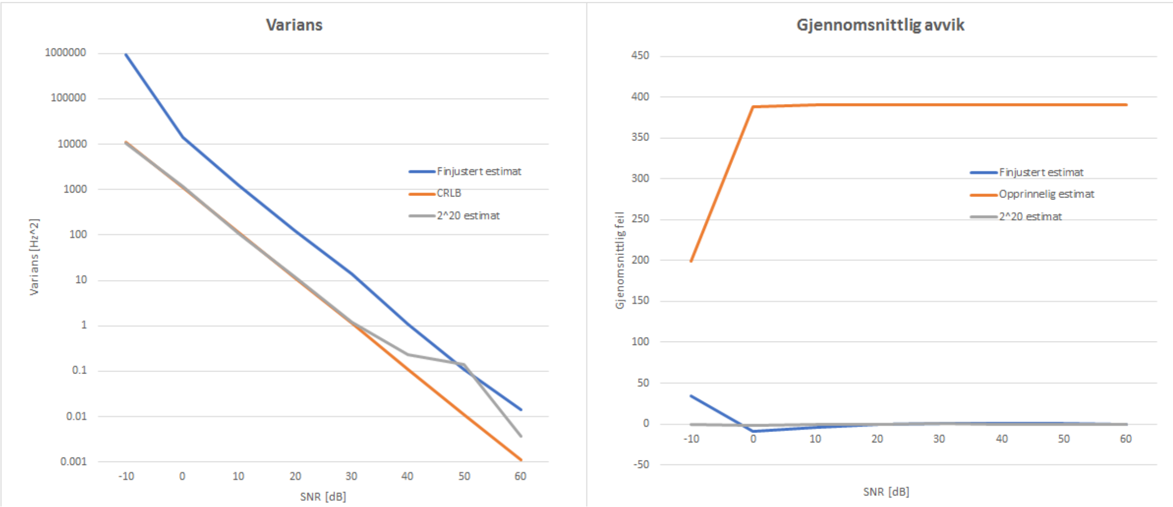
Resultatene fra 100 iterasjoner med finjustering ved ulike støysignal verdier er vist i Tabell 1. Fra tabellen framkommer det tydelig at estimatoren yter svært godt. Sammenliknet med ML estimatene basert på FFT med lengde 2^{10} i Tabell 2 er det tydelig at det gjennomsnittlige feilestimatet er redusert fra 400 Hz til rundt 30 Hz etter at Nelder-Mead algortimen har søkt etter beste frekvens. En interessant observasjon fra tabell Tabell 1 er at estimatoren er forventningsrett med 2 desimalers presisjon når signal støyforholdet er 60 dB.

Tabell 1: Resultatene etter å ha finjustert et estimat basert på en FFT med lengde 2^{10}. Verdiene er basert på gjennomsnittet av 100 finjusteringer av 100 genererte signal på formen i (1).

SNR	**Gjennomsnittlig frekvens**	**Gjennomsnittlig error**	**Varians**
-10 dB	99 966.23 Hz	34.30 Hz	927 190.70 Hz^2
0 dB	100 009.07 Hz	-9.07 Hz	14 034.01 Hz^2
10 dB	100 003.81 Hz	-3.81 Hz	1 205.85 Hz^2
20 dB	100 000.08 Hz	0.08 Hz	123.72 Hz^2
30 dB	99 999.57 Hz	0.43 Hz	13.70 Hz^2
40 dB	99 999.90 Hz	0.10 Hz	1.11 Hz^2
50 dB	99 999.93 Hz	0.07 Hz	0.11 Hz^2
60 dB	100 000.00 Hz	0.00 Hz	0.01 Hz^2

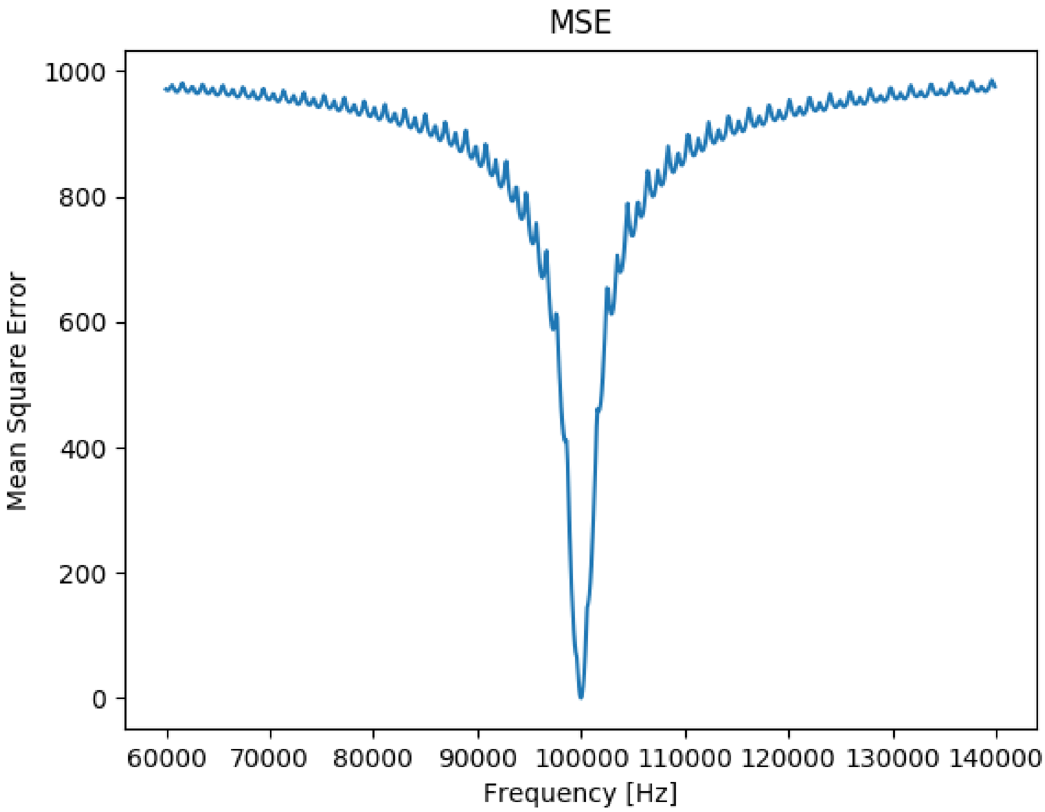
Variansdataene fra Tabell 1 er plottet logaritmisk sammen med CRLB og variansen til 1000 estimat basert på en FFT med lengde 2^{20} i grafen til venstre i Figur 7. Å sammenlikne det finjusterte estimatet mot et estimat basert på en lang FFT (i dette tilfellet 2^{20}) er interessant fordi det er nettopp en slik langt FFT basert estimat finjusteringen er ment å erstatte for mer effektivt estimering. Fra Figur 7 er det tydelig at finjusteringens varians er større enn for 2^{20}-estimatet, med unntak av ved SNR = 50 dB. Det er viktig å poengtere at variansen til finjusteringsestimatet er basert på 100 iterasjoner, mot 2^{20}-estimatets 1000. Flere iterasjoner vil gi datasett med lavere varians.

Grafen til høyre i Figur 7 viser det finjusterte estimatets, det opprinnelige 2^{10}-estimatets og 2^{20}-estimatets gjennomsnittlige avvik fra den faktiske frekvensen. Fra denne er det tydelig at finjusteringen oppnår et signifikant bedre estimat enn det opprinnelige 2^{10}-estimatet. Sammenliknet med 2^{20}-estimatet er det kun ved SNR verdiene -10 og 0 dB finjusteringen er synlig dårligere. Igjen må faktumet at finjusteringens gjennomsnitt er basert på 10% så mange datapunkter som 2^{20}-estimatet trekkes fram som en mulig svakhet ved sammenlikningen.



Figur 7: Finjustering av et estimat basert på en FFT med lengde 2^{10}. Til venstre: Variansen til 100 finjusteringer plottet mot CRLB og variansen til 1000 estimat basert på en 2^{20} lang FFT. Til høyre: Gjennomsnittlig avvik over 100 iterasjoner fra den faktiske frekvensen ω_0 for finjustering, samt det opprinnelige estimatet basert på 2^{10} lang FFT og en 2^{20} lang FFT begge basert på 1000 iterasjoner.

Det gjennomsnittlige kvadratiske avviket mellom den opprinnelig FFTen og FFTen basert på $s[n]$ for ulike frekvenser er vist i Figur 8. Fra grafen er det tydelig at det gjennomsnittlige kvadratiske avviket har sitt minimum ved signalets faktiske frekvens $f_0 = 100$ kHz. Dette viser tydelig at frekvensen som gir en FFT som er mest lik den opprinnelige FFTen er den faktiske frekvensen til signalet som ga opphav til den opprinnelige FFTen. Dette er i tråd med det som var målet i metode b).



Figur 8: *Mean Square Error* mellom opprinnelig FFT og FFTen til et "rent" signal for ulike frekvenser og SNR = 60 dB.

6 Diskusjon

For å vurdere ytelsen til en estimator er det naturlig å betrakte til hvilken grad estimatoren er forvetningsrett ved å studere estimatorens bias, samt å betrakte til hvilken grad estimatoren er effektiv, ved å studere estimatorens varians. En god estimator har lav varians, og liten eller ingen bias. Fra resultatene presentert i kapittel 5 er det en klar trend; de to overnevnte egenskapene forbedres ved **økt lengde på FFTen** og **økt signal støyforhold** (SNR). Sistnevnte er åpenbart mtp på variansen til estimatet. Jo lavere SNR desto større varians får støyen, og desto flere frekvenskomponenter i frekvensspekteret kan "bli nådd" i estimatet. Det er tydelig at for lave SNR verdier kreves FFTer av en hvis lengde for få variansen til en verdi som kvalitativt kan sammenliknes med CRLB. Er FFT lengden stor nok til at variansen blir ulik null, spiller den en signifikant mindre rolle enn støyforholdet i sin påvirkning av estimatets varians.

Lengden til FFTen spiller hovedsaklig inn på estimatets avvik fra den faktiske frekvensen. Resultatene viser en klar trend til at feilen minker signifikant når lengden til FFTen økes. Dette stemmer godt overens med resultatene fra teorien beskrevet i kapittel 2. Når FFTen er kort, er det relativt få frekvenser å velge mellom i frekvensspekter, som leder til et større avvik fra den faktiske frekvensen.

Det er likevel verdt å påpeke estimatoren yter relativt godt, selv ved den korteste FFT lengden med avvik på 0.4% i forhold til faktisk frekvens. For FFTer med lengde 2^{20} ble avviket målt til å være 0.00001% av den faktiske frekvensen. Ved slike verdien kan det argumenteres for at estimatoren for alle praktiske formål er forventningsrett. Hvorvidt dette er godt nok kommer ann på kravene stilt til en eventuell praktisk anvendelse av estimatoren.

Basert på sammenlikningen av CRLB for ω og ϕ med estimert varians for $\hat{\omega}$ og $\hat{\phi}$, er det tydelig at FFT-basert ML-estimering fungerer godt gitt at lengden på FFT'en er lang nok slik at beregningene blir presise, samtidig er det en fordel at parameterne en ønsker å estimere er dominerende fremfor at støy er det. Særlig er det tydelig at slik estimering fungerer godt for $\hat{\phi}$, ved gjennomsnitt av 1000 iterasjoner, ettersom denne estimatoren baserer seg gjennomsnittlig estimert frekvens, en verdi som ligger veldig nær ω_0 for alle målingene.

Resultatene fra finjusteringen av estimatet i metode b) viser at å finjustere et estimat basert på en FFT med lengde 2^{10} er et tilstrekkelig alternativ, dersom kjøretiden er en kritisk faktor i den praktiske anvendelsen. For å vurdere til hvilken grad det er tilstrekkelig hadde det vært naturlig å sammenlikne kjøretiden til et finjustert estimat med kjøretiden til et estimat basert på en FFT med lengde 2^{20}. En kvalitativt analyse av dette er ikke blitt gjort. Det er likevel verdt å merke seg at kompleksiteten til en radix-2 FFT er $N \log N$. Følgelig er FFT estimatene med lengde 2^{20} omtrent 20 (må dobbeltregnes) ganger mer kompleks enn FFTer med lengde 2^{10}. Kompleksiteten til Nelder-Mean algortimen brukt er ikke kjent, men det er altså godt med tid til å finjustere. Fra simuleringene kom det tydelig fram at beregningen av et estimat ved finjustering tok **signifikant** mindre tid (i størrelsesorden flere sekunder) enn estimatet basert på en FFT med lengde 2^{20}.

Resultatene av finjusteringen sammenliknet med resultatene fra et 2^{20} lang FFT estimat antyder at sistenevnte yter bedre, både sammenliknet med CRLB og i avviket, spesielt for lavere SNR. Det er dog verdt å merke seg at det var bare ved finjusteringen at det lykkes å få til en helt forventningsrett estimator, riktignok ved SNR = 60 dB som er et signal med meget god kvalitet. Hvorvidt en slik “trade-off” med ytelse mot kjøretid er verdt det kommer igjen ann på kravene til en eventuell praktisk implementering.

7 Konklusjon

I arbeidet gjennomgått i denne rapporten er teorien og implementeringen av en *Maximum Likelihood Estimator* til å estimere frekvens og fase i et kompleks sinussignal påført hvit gaussisk støy blitt diskutert. ML-estimatoren er blitt realisert ved å finne frekvensen som maksimierer FFten til signalet. Estimatoren yter svært godt, spesielt ved lengre FFT lengder hvor den er tilnærmet forventningsrett og dens varians tilnærment lik grensen gitt ved Cramer Rao Lower Bound. Faseestimatene, som er beregnet ved hjelp av frekvensestimatet, yter svært godt, med små avvik fra den faktiske fasen, spesielt for høyere SNR verider. Sammenliknet med CRLB kan det konkluderes med at faseestimatet er effektivt, ettersom variansen er så god som det er det i praksis er mulig å få til.

Det har også blitt gjennomført finjusteringer av frekvensestimatet basert på "korte" 2^{10} lange estimat, i et forsøk på å redusere beregningstiden til estimatoren. Dette blir gjort ved å numerisk søke etter frekvensen som gjør at et "rent" signal produserer en FFT som er likest mulig FFTen til det opprinnelige estimatet. Resultatene fra disse simuleringene viser at kjøretiden reduseres betydelig, men at en slik estimator ikke yter like godt sammenliknet med estimatoren basert på en 2^{20} lang FFT som finjusteringen er ment å erstatte. Estimatoren yter fortsatt svært godt, mye bedre enn det opprinnelig 2^{10}-estimatet. Hvorvidt en slik finjustering, som vinner kjøretid på bekostning av ytelse, er brukbar kommer ann på kravene satt av den praktiske anvendelsen.

Referanser

[1] *Cramer-Rao bound - Wikipedia*. Wikipedia. URL: https://en.wikipedia.org/wiki/Cram%C3%A9r%E2%80%93Rao_bound.

[2] Magne H. Johnsen Tor A. Myrvoll Stefan Werner. *Estimation, Detection and Classification Theory*. NTNU.

[3] *TTT4175 Estimation, Detection and Classification Project Descriptions: Estimation Theory*. NTNU.

[4] *Kildekode brukt til simuerlinger*. J. Vahlin A. H. Bugge. URL: https://github.com/jakvah/TTT4275-EstimationProject.

A Simuleringsresultater frekvens

Tabell 2: Simuleringsresultater for $\hat{\omega}$. Gjennomsnittene er basert på 1000 frekvensestimat.

FFT Lengde	**SNR** [dB]	**Gjennomsnittlig frekvens [Hz]**	**Gjennomsnittlig feil [Hz]**	**Varians** [Hz^2]	**CRLB** [Hz^2]
1024	-10	99 800.78	199.22	150 434.25	11 257.48
	0	99 611.33	388.67	1 905.44	1 125.75
	10	99 609.38	390.63	0.00	112.57
	20	99 609.38	390.63	0.00	11.26
	30	99 609.38	390.63	0.00	1.13
	40	99 609.38	390.63	0.00	0.11
	50	99 609.38	390.63	0.00	0.01
	60	99 609.38	390.63	0.00	$1.13 \cdot 10^{-3}$
4096	-10	100 007.81	-7.81	16 859.70	11 257.48
	0	100 039.06	-39.06	10 882.77	1 125.75
	10	100 094.97	-94.97	649.09	112.57
	20	100 097.66	-97.66	0.00	11.26
	30	100 097.66	-97.66	0.00	1.13
	40	100 097.66	-97.66	0.00	0.11
	50	100 097.66	-97.66	0.00	0.01
	60	100 097.66	-97.66	0.00	$1.13 \cdot 10^{-3}$
16384	-10	100 001.22	-1.22	11 744.92	11 257.48
	0	100 001.53	-1.53	1 500.46	1 125.75
	10	99 992.13	7.87	736.70	112.57
	20	99 977.60	22.40	119.00	11.26
	30	99 975.59	24.41	0.00	1.13
	40	99 975.59	24.41	0.00	0.11
	50	99 975.59	24.41	0.00	0.01
	60	99 975.59	24.41	0.00	$1.13 \cdot 10^{-3}$
65536	-10	100 005.43	-5.43	11 905.84	11 257.48
	0	99 998.60	1.40	1 118.22	1 125.75
	10	100 000.52	-0.52	125.40	112.57
	20	100 000.85	-0.85	52.59	11.26
	30	100 004.81	-4.81	18.13	1.13
	40	100 006.10	-6.10	0.00	0.11
	50	100 006.10	-6.10	0.00	0.01
	60	100 006.10	-6.10	0.00	$1.13 \cdot 10^{-3}$
262144	-10	100 004.81	-4.81	11 281.95	11 257.48
	0	100 000.60	-0.60	1 084.14	1 125.75
	10	99 999.61	0.39	113.82	112.57
	20	100 000.13	-0.13	13.22	11.26
	30	99 999.83	0.17	3.34	1.13
	40	99 998.94	1.06	1.57	0.11
	50	99 998.47	1.53	0.00	0.01
	60	99 998.47	1.53	0.00	$1.13 \cdot 10^{-3}$
1048576	-10	100 000.83	-0.83	10 648.19	11 257.48
	0	100 002.39	-2.39	1 175.49	1 125.75
	10	100 000.39	-0.39	107.59	112.57
	20	100 000.11	-0.11	11.77	11.26
	30	99 999.97	0.03	1.20	1.13
	40	99 999.99	0.01	0.23	0.11
	50	100 000.20	-0.20	0.14	0.01
	60	100 000.38	-0.38	$3.63 \cdot 10^{-3}$	$1.13 \cdot 10^{-3}$

B Simuleringsresulater fase

Tabell 3: Simuleringsresultater for $\hat{\phi}$. Gjennomsnittene er basert på 1000 faseestimat.

FFT Lengde	**SNR [dB]**	**Gjennomsnittlig fase [rad]**	**Gjennomsnittlig feil [rad]**	**Varians** [rad^2]	**CRLB** [rad^2]
1024	-10	0.39	4.50E-04	1.10E-02	9.75E-03
	0	0.39	1.42E-03	1.14E-03	9.75E-04
	10	0.39	1.83E-04	1.19E-04	9.75E-05
	20	0.39	3.65E-05	1.10E-05	9.75E-06
	30	0.39	7.98E-06	1.08E-06	9.75E-07
	40	0.39	6.38E-06	1.11E-07	9.75E-08
	50	0.39	3.38E-06	1.10E-08	9.75E-09
	60	0.39	2.12E-06	1.11E-09	9.75E-10
4096	-10	0.39	1.30E-03	1.02E-02	9.75E-03
	0	0.39	1.08E-03	1.00E-03	9.75E-04
	10	0.39	-2.64E-05	1.01E-04	9.75E-05
	20	0.39	-8.07E-06	1.02E-05	9.75E-06
	30	0.39	-6.88E-06	8.73E-07	9.75E-07
	40	0.39	7.29E-06	9.37E-08	9.75E-08
	50	0.39	1.84E-07	9.85E-09	9.75E-09
	60	0.39	-7.56E-07	9.76E-10	9.75E-10
16384	-10	0.39	3.81E-03	9.75E-03	9.75E-03
	0	0.39	3.94E-04	9.59E-04	9.75E-04
	10	0.39	2.16E-04	1.02E-04	9.75E-05
	20	0.39	-9.33E-05	9.76E-06	9.75E-06
	30	0.39	-3.18E-05	1.05E-06	9.75E-07
	40	0.39	-1.80E-05	9.23E-08	9.75E-08
	50	0.39	-3.86E-06	1.00E-08	9.75E-09
	60	0.39	-6.62E-07	9.96E-10	9.75E-10
65536	-10	0.40	-3.91E-03	9.14E-03	9.75E-03
	0	0.39	-7.80E-05	9.21E-04	9.75E-04
	10	0.39	-2.64E-04	9.43E-05	9.75E-05
	20	0.39	1.88E-04	9.84E-06	9.75E-06
	30	0.39	1.80E-06	9.21E-07	9.75E-07
	40	0.39	1.52E-05	9.27E-08	9.75E-08
	50	0.39	-3.80E-06	9.27E-09	9.75E-09
	60	0.39	1.19E-06	9.59E-10	9.75E-10
262144	-10	0.39	1.73E-03	9.44E-03	9.75E-03
	0	0.39	-5.90E-04	1.01E-03	9.75E-04
	10	0.39	-5.10E-04	9.94E-05	9.75E-05
	20	0.39	1.43E-04	1.01E-05	9.75E-06
	30	0.39	-1.94E-05	9.97E-07	9.75E-07
	40	0.39	-8.03E-06	9.96E-08	9.75E-08
	50	0.39	2.87E-06	9.45E-09	9.75E-09
	60	0.39	-1.29E-06	9.92E-10	9.75E-10
1048576	-10	0.39	-1.57E-04	1.00E-02	9.75E-03
	0	0.39	5.48E-04	1.05E-03	9.75E-04
	10	0.39	5.21E-04	1.02E-04	9.75E-05
	20	0.39	1.38E-04	1.00E-05	9.75E-06
	30	0.39	-1.26E-05	8.94E-07	9.75E-07
	40	0.39	-1.57E-05	9.41E-08	9.75E-08
	50	0.39	-1.95E-06	9.69E-09	9.75E-09
	60	0.39	4.41E-07	9.21E-10	9.75E-10

C Kode for estimering av parametre

Se dokumentasjon på `https://github.com/jakvah/TTT4275-EstimationProject` for kildekode, samt hvordan kjøre programmet.

```python
# For instructions on how to run, see:
# https://github.com/jakvah/TTT427-EstimationProject

import numpy as np
import matplotlib.pyplot as plt
import buggesmatteland as bml
import cmath
import statistics as st
import scipy.optimize
import sys

# ---------- Specifications from CMD line ---------- #
attemptCounter = 0

try:
    SNR_db = int(sys.argv[1])
except IndexError:
    print("No arguments given. Running with default.")
    SNR_db = -10
    k = 10
    fft_length = 2**k
    ITERATIONS = 100
    attemptCounter += 1

try:
    k = int(sys.argv[2])
    fft_length = 2**k
except IndexError:
    if attemptCounter == 1:
        print("No length or iteration arguments given. Running with default.")
        k = 10
        fft_length = 2**k
        ITERATIONS = 100
        attemptCounter += 1

try:
    ITERATIONS = int(sys.argv[3])
except IndexError:
    if attemptCounter == 2:
        print("No iteration argument given. Running with default.")
        ITERATIONS = 100
```

```

print()
# ---------- Specifications ---------- #
A = 1.0

SNR_linear = 10.0**(SNR_db/10)
SIGMA_SQUARED = (A**2)/(2*SNR_linear)

T = 10**(-6)
N = 513
n_0 = -256
f_0 = 10**5
omega_0 = 2*np.pi*f_0
theta = np.pi/8

# ---------- CRLB Helpers ---------- #
P = (N*(N-1)) / 2
Q = (N*(N-1)*(2*N-1)) / 6

# ---------- CRLB ---------- #

CRLB_OMEGA = (12*(SIGMA_SQUARED)) / ((A**2)*(T**2)*N*((N**2)-1)) # In Radians^2
CRLB_THETA = 12*(SIGMA_SQUARED)*((n_0**2)*N + 2*n_0*P + Q) / ((A**2)*(N**2)*((N**2)-1))

# ---------- GLOBAL FFT TIL 1B) ---------- #

# Generate a signal with some sweet sweet noise
# White complex Gaussian noise
gwReal = np.random.normal(0, np.sqrt(SIGMA_SQUARED), size=N)
gwImag = np.random.normal(0, np.sqrt(SIGMA_SQUARED), size=N)*1j

gw = []
for i in range(N):
    gw.append(gwReal[i] + gwImag[i])

# Exponential signal
gs = []
for n in range(N):
    gs.append(A*np.exp(np.complex(0,1)*((omega_0)*(n + n_0)*T + theta)))

# Total signal
gx = []
for i in range(N):
    gx.append(gs[i] + gw[i])

gFFT = np.fft.fft(gx,2**10)
gf = bml.findDominantFrequency(np.absolute(gFFT),T,2**10)
```

```

def iterate():
    # ---------- Signals ---------- #

    # White complex Gaussian noise
    wReal = np.random.normal(0, np.sqrt(SIGMA_SQUARED), size=N)
    wImag = np.random.normal(0, np.sqrt(SIGMA_SQUARED), size=N)*1j

    w = []
    for i in range(N):
        w.append(wReal[i] + wImag[i])

    # Exponential signal
    s = []
    for n in range(N):
        s.append(A*np.exp(np.complex(0,1)*((omega_0)*(n + n_0)*T + theta)))

    # Total signal
    x = []
    for i in range(N):
        x.append(s[i] + w[i])

    # Fourier transform
    FT_x = np.fft.fft(x,fft_length)

    # Finding most dominant in total signal
    f_2,i = bml.findDominantFrequency(np.absolute(FT_x),T,fft_length)

    t = np.angle((np.exp(-(np.complex(0,1)*2*np.pi*f_2*n_0*T)))*FT_x[i])

    return f_2,t

def functionToBeMinimized(f_variable):
    f_var_sliced = f_variable[0]

    # -------- Exponential signal without noise -------- #
    s = []
    for n in range(N):
        s.append(A*np.exp(np.complex(0,1)*((2*np.pi*f_var_sliced)*(n + n_0)*T + theta)))

    fftGuess = np.fft.fft(s,2**10)

    if f_var_sliced == 100000:
        plt.figure(1)
        plt.subplot(211)
        plt.title("With noise")
        plt.plot(np.absolute(gFFT))
```

```python

        plt.subplot(212)
        plt.title("Without noise")
        plt.plot(np.absolute(fftGuess))

        plt.savefig("compare.png")

    return bml.meanSquareError(np.absolute(fftGuess),np.absolute(gFFT))


def main():

    print("Running ",ITERATIONS, "iterations with:")
    print("SNR [dB]:",SNR_db)
    print("FFT length:",fft_length,"(2^" + str(k) + ")")
    print("Frequency:",f_0/1000,"kHz")
    print("The CRLB for the Omega estimator is:", CRLB_OMEGA/(4*np.pi**2),"Hz^2")
    print("The CRLB for the Theta estimator is:",CRLB_THETA)
    print()

    print(" *--------------- RESULTS ---------------*")

    error_theta=[]
    error_f=[]
    freqs = []
    thetas = []

    for i in range(ITERATIONS):
        f,t = iterate()
        err_f = f_0 - f
        err_theta=theta-t
        freqs.append(f) # In hertz
        error_f.append(err_f)
        thetas.append(t)
        error_theta.append(err_theta)


    print("Mean freq:",st.mean(freqs))
    errmean=st.mean(error_f)
    print("Mean freq error is: ", errmean/1000, "kHz")
    thetamean = st.mean(thetas)

    errvar_f=st.variance(error_f, errmean)
    print("The variance of the freq error is: ",errvar_f, "Hz^2")

    mean_error=st.mean(error_theta)

```

```
    print("Mean theta is:", thetamean)
    print("Mean theta error is: error", st.mean(error_theta))
    print("The variance of the phase  is: ", st.variance(error_theta, mean_error))

    print()
    print("Doing part b)")
    print()

    result = scipy.optimize.minimize(
        functionToBeMinimized,100000,method = "Nelder-Mead"
    )

    # Plotting MSE
    mse = []
    t = [1,2]
    for f in range(60000,140000,100):
        t[0] = f
        mse.append(functionToBeMinimized(t))

    plt.figure(2)
    plt.title("MSE")
    plt.xlabel("Frequency [Hz]")
    plt.ylabel("Mean Square Error")
    plt.plot(np.arange(60000,140000,100),mse)
    plt.savefig("mse.png")

    print("The guess with noise and FFT length 2^10:",gf[0], "Hz")
    print("The guess after finetuning:",result.x[0])




main()
```

D Kode for automatisk simulering

Brukt til å simulere signaler med alle kombinasjoner av FFT lengder og SNR. Se `https://github.com/jakvah/TTT4275-EstimationProject` for kildekode og dokumentasjon.

```
import numpy as np
import matplotlib.pyplot as plt
import buggesmatteland as bml
import math
import statistics as st
import xlsxwriter
import sys

# ---------- These are supposed to change ---------- #
dBs = [-10,0,10,20,30,40,50,60]
lengthPowers = [10,12,14,16,18,20]

ITERATIONS = 10

# ---------- Constants ---------- #
A = 1.0
T = 10**(-6)
N = 513
n_0 = -256
f_0 = 10**5
omega_0 = 2*np.pi*f_0
theta = np.pi/8

FILENAME = "innafor.xlsx"


# ---------- CRLB Helpers ---------- #
P = (N*(N-1)) / 2
Q = (N*(N-1)*(2*N-1)) / 6

# ---------- CRLB ---------- #

def iterate(SIGMA_SQUARED,fft_length):
    # ---------- Signals ---------- #

    # White complex Gaussian noise
    wReal = np.random.normal(0, np.sqrt(SIGMA_SQUARED), size=N)
    wImag = np.random.normal(0, np.sqrt(SIGMA_SQUARED), size=N)*1j

    w = []
    for i in range(N):
```

```
        w.append(wReal[i] + wImag[i])

    # Exponential signal
    s = []
    for n in range(N):
        s.append(A*np.exp(np.complex(0,1)*((omega_0)*(n + n_0)*T + theta)))

    # Total signal
    x = []
    for i in range(N):
        x.append(s[i] + w[i])

    # Fourier transform
    FT_x = np.fft.fft(x,fft_length)


    # Finding most dominant in total signal
    f_2,i = bml.findDominantFrequency(np.absolute(FT_x),T,fft_length)

    t = np.angle((np.exp(-(np.complex(0,1)*2*np.pi*f_2*n_0*T)))*FT_x[i])

    return f_2,t

def main():
    print("Simulating datasets with:")
    print("SNRs: ",dBs)
    print("FFTs with radix-2 powers:",lengthPowers)
    print("Each with",ITERATIONS, "iterations. This might take some time ...")
    print()

    wb = xlsxwriter.Workbook(FILENAME)
    ws = wb.add_worksheet()

    # Colum names
    ws.write(0, 0, "FFT Length")
    ws.write(0, 1, "SNR [dB]")
    ws.write(0, 2, "Mean estimated frequency [Hz]")
    ws.write(0, 3, "Mean frequency error [Hz]")
    ws.write(0, 4, "Frequency variance [Hz^2]")
    ws.write(0, 5, "Frequency CRLB [Hz^2]")
    ws.write(0, 6, "Mean estimated phase [rad]")
    ws.write(0, 7, "Mean phase error [rad]")
    ws.write(0, 8, "Phase variance [rad^2]")
    ws.write(0, 9, "Phase CRLB [rad^2]")

    lengthIterationIndex = 0
    for p in lengthPowers:
```

```
        fft_length = 2**p
        dataIterationIndex = 0

        ws.write(1 + lengthIterationIndex*len(dBs), 0, str(fft_length))
        print("Computing FFTs of length", fft_length)
        if p == 18:
                print("Now also showing SNR progress:")

        for SNR_db in dBs:
            ws.write(1 + dataIterationIndex +lengthIterationIndex*len(dBs), 1, SNR_db)


            SIGMA_SQUARED = bml.sigmaSquaredFromdB(SNR_db,A)
            CRLB_OMEGA = (12*(SIGMA_SQUARED)) / ((A**2)*(T**2)*N*((N**2)-1))
            CRLB_THETA = 12*(SIGMA_SQUARED)*((n_0**2)*N + 2*n_0*P + Q) /
                        ((A**2)*(N**2)*((N**2)-1))

            freqError=[]
            freq = []
            thetas = []
            phaseError = []
            for i in range(ITERATIONS):
                f,t = iterate(SIGMA_SQUARED,fft_length)

                errf = f_0 - f
                freq.append(f)
                freqError.append(errf)

                errp = theta - t
                thetas.append(t)
                phaseError.append(errp)

            freqmean = st.mean(freq)
            freqErrMean=st.mean(freqError)
            freqErrVar=st.variance(freqError, freqErrMean)

            phaseMean = st.mean(thetas)
            phaseErrMean = st.mean(phaseError)
            phaseErrVar = st.variance(phaseError,phaseErrMean)

            ws.write(1 + dataIterationIndex +lengthIterationIndex*len(dBs),
            2, freqmean)
            ws.write(1 + dataIterationIndex +lengthIterationIndex*len(dBs),
            3, freqErrMean)
            ws.write(1 + dataIterationIndex +lengthIterationIndex*len(dBs),
            4, freqErrVar)
            ws.write(1 + dataIterationIndex +lengthIterationIndex*len(dBs),
```

```
                5, (CRLB_OMEGA/(4*np.pi**2)))

                ws.write(1 + dataIterationIndex +lengthIterationIndex*len(dBs),
                6, phaseMean)
                ws.write(1 + dataIterationIndex +lengthIterationIndex*len(dBs),
                7, phaseErrMean)
                ws.write(1 + dataIterationIndex +lengthIterationIndex*len(dBs),
                8, phaseErrVar)
                ws.write(1 + dataIterationIndex +lengthIterationIndex*len(dBs),
                9, CRLB_THETA)

                dataIterationIndex += 1
                if p > 16:
                    print("Done with SNR:",SNR_db, "dB")
            lengthIterationIndex += 1

    wb.close()
    print("Added iterations data to",FILENAME)
main()
```

E Kode for matematiske beregninger

```python
import numpy as np
import matplotlib.pyplot as plt
import sys

def meanSquareError(list1,list2):
    if len(list1) != len(list2):
        print("List lengths don't match. Exiting")
        sys.exit()

    total = 0
    for i in range(len(list1)):
        total += (list1[i] - list2[i])**2

    return total/len(list1)

# Returns most dominant frequency of FFT in hertz
# fft must be numpy array, absolute value
def findDominantFrequency(fft,samplingPeriod,fftLength):
    maxVal = max(fft)
    maxIndex = np.where(fft == maxVal)
    maxIndex = maxIndex[0][0]

    f = maxIndex * (1/(samplingPeriod*fftLength))
    return f, maxIndex

def sigmaSquaredFromdB(SNR_db,A):
    SNR_linear = 10.0**(SNR_db/10)
    SIGMA_SQUARED = (A**2)/(2*SNR_linear)
    return SIGMA_SQUARED
```