



PROSJEKTRAPPORT 2

Lyd Analyse - Pitch Perfect

ING2501 MM2

AV

Abay, Yonatan Monken

Aleksandersen, Tor Johan Drange

Bertilsson, Agnes Liv

Bjørdal, Olav Schei

Vesteng, Karen

KLASSE: VING 78

Rapport levert: 5. november 2025

Sammendrag

Denne rapporten presenterer utviklingen og analysen av et sanntids lydanalysesystem basert på Fourier-transformasjonen og dens diskrete varianter. Hovedmålet var å undersøke hvordan teoretiske prinsipper fra signalbehandling kan anvendes praktisk for å identifisere og visualisere frekvensinnholdet i lydsignaler. Systemet, kalt *Pitch Perfect*, ble implementert i Python ved bruk av Fast Fourier Transform (FFT) og Short-Time Fourier Transform (STFT) for å analysere og representere hvordan et lydsignals spekter endres over tid.

Teoridelen beskriver Fourier-transformasjonens egenskaper, den diskrete Fourier-transformasjonen (DFT), samt optimalisering ved hjelp av FFT. Videre diskuteres vindusfunksjoner, parabolisk toppinterpolasjon og RMS-beregning som grunnlag for nøyaktig og stabil signalanalyse. Metoden bygger på innhenting av lyd i sanntid, der signalet deles i vinduer, vindusvektes, transformeres og analyseres kontinuerlig.

Resultatene viser at systemet nøyaktig kan identifisere og visualisere dominerende frekvenser i et lydsignal, og at parabolisk interpolasjon forbedrer frekvensestimaterne betydelig. Bruk av Hanning-vindu reduserte *spectral leakage*, mens RMS-gating effektivt filtrerte bort bakgrunnsstøy. Analysen bekrefter at balansen mellom tids- og frekvensoppløsning er avgjørende for systemets presisjon og respons.

Prosjektet demonstrerer hvordan matematiske metoder for signalbehandling kan brukes til å bygge praktiske analyseverktøy for lyd. Resultatene samsvarer godt med teori, og implementasjonen danner et solid grunnlag for videre utvikling innen digital signalanalyse og musikalsk notegjenkjenning.

Innhold

1	Innledning	1
2	Teori	2
2.1	Fourier-transformasjonen	2
2.2	Diskret Fourier Transform (DFT)	3
2.3	Fast Fourier Transform (FFT)	5
2.4	Short-Time Fourier Transform (STFT)	8
2.4.1	Ramme lengde	8
2.4.2	Hop-størrelse	8
2.5	Vindusfunksjoner (Windowing)	9
2.5.1	Hanning-vinduet	10
2.5.2	Parabolsk toppinterpolasjon (Quadratic interpolation)	12
2.6	Root Mean Square (RMS)	12
3	Metode	13
3.1	Oppbygging av koden	13
3.2	Initialverdier	13
3.3	Sanntids innhenting av lyd	13
3.4	Fourier-analyse av lydsignalet	14
3.5	Støygating og glatting	14
3.6	Beregning av tone og note	15
4	Signalanalyse med Python	17
4.1	Pitch Perfect	17
4.2	Resultater	17
5	Diskusjon	18
6	Feilkilder	19
7	Konklusjon	20
	Referanser	21
	Vedlegg	22

1 Innledning

Lydanalyse er et viktig område innen moderne signalbehandling. Målet er å kunne beskrive, forstå og behandle signaler som varierer over tid, slik som lyd. Et lydsignal kan beskrives som en funksjon av tid, men mange av de mest interessante egenskapene, tonehøyde, klang og overtoner kommer tydeligere frem når signalet analyseres i frekvensdomenet. For å gjøre dette brukes Fourier-transformasjonen, et matematisk verktøy som gjør det mulig å representere et tidsdomene-signal som en sum av sinusformede bølger med ulike frekvenser, styrker og faser.

I dette prosjektet undersøkes hvordan Fourier-transformasjonen og dens diskrete varianter kan brukes til praktisk lydanalyse. Den diskrete Fourier-transformasjonen (DFT) og den raske Fourier-transformasjonen (FFT) implementeres for å beregne og visualisere frekvensinnholdet i digitale lydsignaler. Gjennom dette får man en forståelse av hvordan matematiske metoder kan brukes til å trekke ut informasjon fra reelle signaler.

Videre introduseres Short-Time Fourier Transform (STFT), som gjør det mulig å analysere hvordan frekvensinnholdet i et signal endrer seg over tid. Denne metoden gir en mer detaljert fremstilling av komplekse signaler, som for eksempel tale og musikk, der både tonehøyde og intensitet varierer kontinuerlig.

Prosjektets mål er å utvikle et system for sanntidsanalyse og visualisering av lyd ved hjelp av Fourier-baserte metoder. Arbeidet kombinerer teoretisk forståelse av Fourier-transformasjonen med praktisk programmering i Python, og viser hvordan matematiske prinsipper kan brukes til å løse konkrete problemer innen lydteknologi og signalbehandling.

2 Teori

Målet i dette prosjektet er å utvikle et system som analyserer lydsignaler i sanntid ved hjelp av Fourier-metoder. For å gjøre dette trenger vi en grunnleggende forståelse av Fourier-transformasjonen og hvordan diskrete varianter (DFT/FFT) anvendes på digitale lydsignaler. Videre beskriver vi Short-Time Fourier Transform (STFT) for tids-frekvensanalyse, og diskuterer praktiske hensyn som vinduvalg, frekvensoppløsning, lekkasje og latens.

Kapittelet gir først en kort teori for FT/DFT/FFT og deres sammenhenger. Deretter presenterer vi STFT-formuleringen og sentrale designvalg (vindustype, rammelengde N , hop-size H). Til slutt omtaler vi metoder for støyhåndtering og presisjonsheving (RMS-gating, båndbegrensning og parabolisk toppinterpolasjon) som danner grunnlaget for analysen brukt i prosjektet.

2.1 Fourier-transformasjonen

Fourier-transformasjonen er et matematisk verktøy som brukes til å analysere signaler i frekvensdomenet. Den gir en måte å representere et tidsavhengig signal som en sum av sinusformede bølger med forskjellige frekvenser og amplituder. Den kontinuerlige Fourier-transformasjonen (FT) er definert som:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft} dt = \mathcal{F}\{x(t)\} \quad (1)$$

Hvor $X(f)$ er frekvensspekteret til tidsdomenet signalet $x(t)$, og f er frekvensen. Dette gir sammenhengen:

$$X(f) = \mathcal{F}\{x(t)\} \Leftrightarrow x(t) = \mathcal{F}^{-1}\{X(f)\}$$

Noen nyttige egenskaper ved Fourier-transformasjonen inkluderer linearitet, tids- og frekvensforskyvning, samt konvolusjonsteoremet. Disse egenskapene gjør det lettere å analysere og manipulere signaler i frekvensdomenet:

Linearitet:

$$\mathcal{F}\{ax_1(t) + bx_2(t)\} = aX_1(f) + bX_2(f)$$

Tidsforskyvning:

$$\mathcal{F}\{x(t - t_0)\} = X(f)e^{-j2\pi ft_0}$$

Frekvensforskyvning:

$$\mathcal{F}\{x(t)e^{j2\pi f_0 t}\} = X(f - f_0)$$

Konvolusjonsteoremet:

$$\mathcal{F}\{x_1(t) * x_2(t)\} = X_1(f) \cdot X_2(f)$$

Parsevals teorem:

$$\int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |X(f)|^2 df$$

Hver av sammenhengene beskriver en grunnleggende regel som gjør det enklere å forstå og beregne hvordan et signal endrer seg når det forskyves, skales eller kombineres med

andre signaler. For eksempel viser linearitet at transformasjonen bevares når signaler legges sammen, tidsforskyvning viser hvordan et signal som flyttes i tid gir en faseendring i frekvensdomenet, mens frekvensforskyvning beskriver hvordan modulasjon i tidsdomenet flytter signalet i frekvens. Konvolusjonsteoremet brukes til å analysere hvordan systemer filtrerer signaler, fordi konvolusjon i tid tilsvarer multiplikasjon i frekvens. Til slutt beskriver Parsevals teorem sammenhengen mellom energi i tids- og frekvensdomenet, og brukes til å beregne eller sammenligne signalenergi på tvers av domener. **kan også være en ide å legge inn forklaring etter hvert element istedenfor å ha det i et samlet avsnitt. 'Personal preference'**

2.2 Diskret Fourier Transform (DFT)

Den diskrete Fourier-transformasjonen (DFT) er en numerisk metode for å beregne Fourier-transformasjonen av et diskret signal. Det at et signal er diskret betyr at det er representert som en sekvens av prøver tatt på bestemte tidspunkter. DFT brukes ofte i digital signalbehandling for å analysere frekvensinnholdet i digitale signaler. DFT er definert som:

$$X[k] = \sum_{n=0}^{N-1} x[n]W^{kn}, \quad k = 0, 1, \dots, N-1, \quad W = e^{-j2\pi/N} \quad (2)$$

Hvor N er antall prøver i signalet $x[n]$. DFT gir et diskret frekvensspekter $X[k]$ som representerer signalets frekvensinnhold. DFT kan også uttrykkes i matriseform som:

$$\mathbf{X} = \mathbf{A} \cdot \mathbf{x}[\mathbf{n}] \quad (3)$$

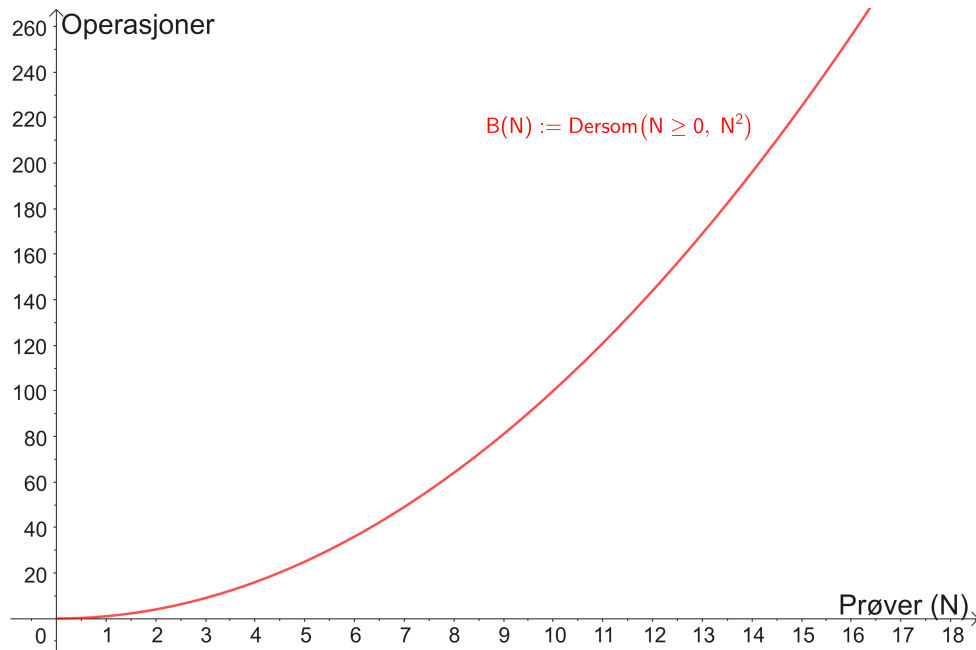
Hvor \mathbf{x} er en vektor av tidsdomenet signalprøver, \mathbf{X} er frekvensdomenet representasjonen, og \mathbf{A} er en $N \times N$ matrise med elementene:

$$W_{kn} = W^{kn} = e^{-j2\pi kn/N}$$

For å vurdere hvor effektiv en DFT-beregning er, ser vi på tidskompleksiteten, altså antall (elementære) operasjoner som funksjon av antall prøver N . DFT har en tidskompleksitet på $O(N^2)$, noe som kan være ineffektivt for lange signaler. Dette kan vi se ved at for hver av de N frekvenskomponentene må vi utføre en sum over N tidsprøver, noe som gir totalt $N \times N = N^2$ operasjoner. Altså:

$$B(N) = (k_{max} + 1) \cdot (n_{max} + 1) = N \cdot N = N^2$$

En slik kompleksiteten har den følgende øknings om vist i figuren under:



Figur 1: Beregningstid for DFT som funksjon av antall prøver N .

2.3 Fast Fourier Transform (FFT)

Som beskrevet tidligere har DFT en tidskompleksitet på $O(N^2)$, noe som kan være ineffektivt for lange signaler siden det tar lengre tid. For å løse dette problemet ble Fast Fourier Transform (FFT) utviklet. FFT er en familie av effektive algoritmer for å beregne DFT med betydelig redusert tidskompleksitet. Den reduserer tidskompleksiteten fra $O(N^2)$ til $O(N \log N)$ ved å utnytte symmetrier i DFT og dele opp beregningene i mindre deler. FFT er spesielt nyttig for lange signaler, der den betydelige reduksjonen i beregningstid kan være avgjørende for sanntidsapplikasjoner slik som lyd- og bildebehandling. Blant de mest kjente FFT-algoritmene er Cooley-Tukey-algoritmen (Radix-2), som vi skal bruke som et eksempel. Notasjonen vi bruker er formelen for DFT (2):

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad W_N = e^{-j2\pi/N}$$

Her bruker vi W_N for å indikere N -punkt DFT. Denne kan deles opp i to summer, en for partallsindekserte elementer og en for oddetallsindekserte elementer:

$$X[k] = \sum_{m=0}^{N/2-1} x[2m] W_N^{k(2m)} + \sum_{m=0}^{N/2-1} x[2m+1] W_N^{k(2m+1)}$$

Ved å faktorisere ut W_N^k fra den andre summen får vi:

$$X[k] = \sum_{m=0}^{N/2-1} x[2m] W_N^{2km} + W_N^k \sum_{m=0}^{N/2-1} x[2m+1] W_N^{2km}$$

Legg merke til at:

$$W_N^{2km} = e^{-j2\pi(2km)/N} = e^{-j2\pi km/(N/2)} = W_{N/2}^{km}$$

Som tilsvarer DFT av lengde $N/2$. Vi kan derfor definere to nye DFT-er:

$$E[k] = \sum_{m=0}^{N/2-1} x[2m]W_{N/2}^{km}, \quad O[k] = \sum_{m=0}^{N/2-1} x[2m+1]W_{N/2}^{km}$$

Dermed kan vi skrive:

$$X[k] = E[k] + W_N^k O[k], \quad k = 0, 1, \dots, N/2 - 1$$

For $k = 0 \dots N/2 - 1$ får vi også (pga periodisiteten til DFT):

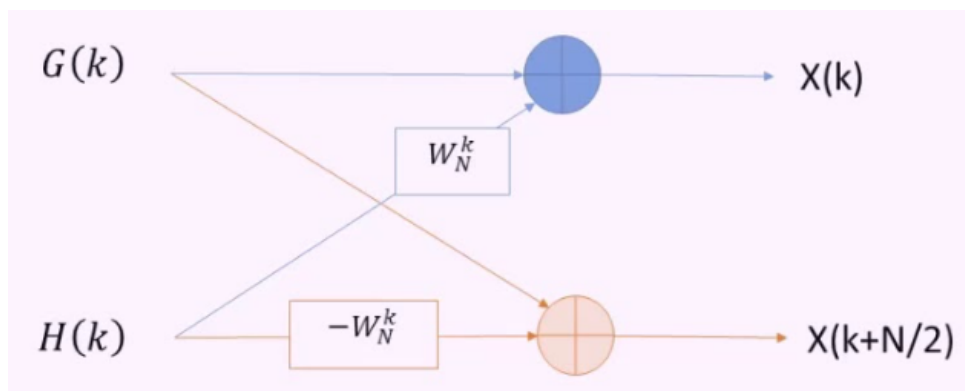
$$X[k + N/2] = E[k] - W_N^k O[k], \quad k = 0, 1, \dots, N/2 - 1$$

Grunnen til at vi får minus-tegnet i den andre ligningen er at:

$$W_N^{k+N/2} = W_N^k \cdot W_N^{N/2} = W_N^k \cdot e^{-j\pi} = -W_N^k$$

Vi har da butterfly-operasjonen:

$$X[k] = E[k] + W_N^k O[k], \quad X[k + N/2] = E[k] - W_N^k O[k], \quad k = 0, 1, \dots, N/2 - 1$$



Figur 2: Butterfly-operasjonen i FFT-algoritmen $G(k) = E(k)$ og $H(k) = O(k)$.

Dette betyr at vi kan beregne en N -punkt DFT ved å beregne to $N/2$ -punkt DFT-er, noe som reduserer antall nødvendige beregninger betydelig. Ved å gjenta denne prosessen rekursivt til vi når DFT-er av lengde 2, oppnår vi den effektive FFT-algoritmen med tidskompleksitet $O(N \log N)$. Dette kan vi se ved å la $B(N)$ være antall grunnoperasjoner for en N -punkts FFT. Da har vi:

$$B(N) = aB(N/b) + cN, \quad B(1) = c_0$$

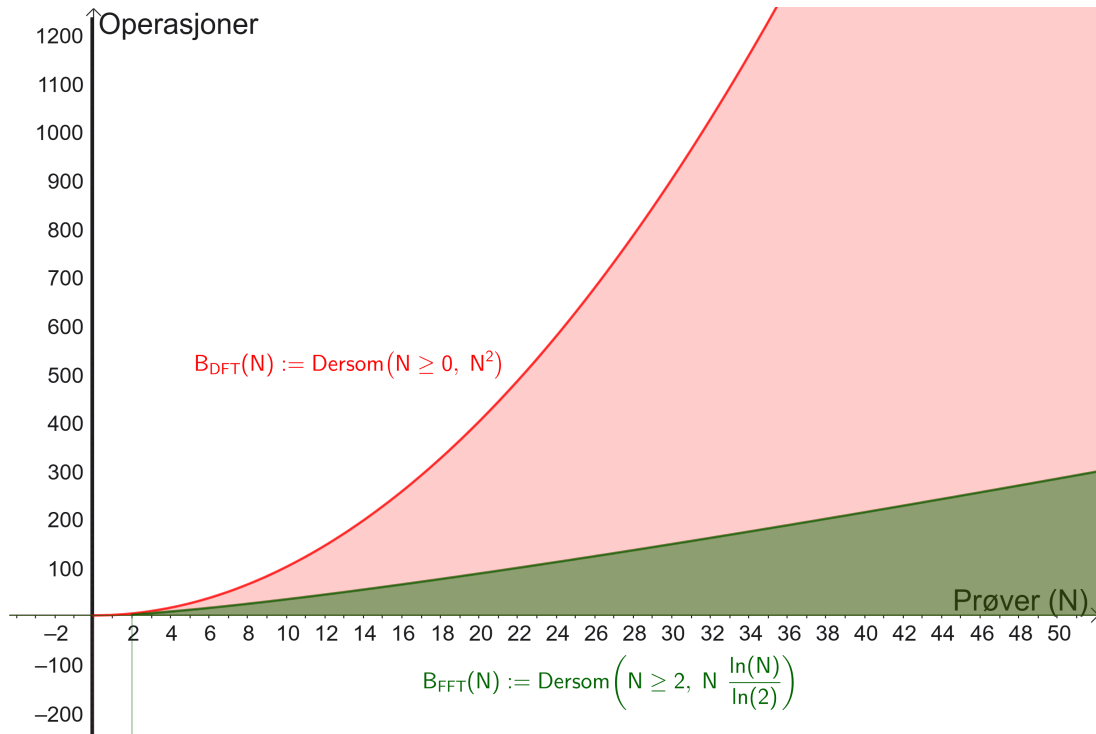
hvor a er antall delproblemer (her 2), b er faktoren vi deler problemet med (her 2), og cN representerer den lineære tiden som kreves for å kombinere resultatene, c er altså en konstant. Dette gir:

$$B(N) = 2B(N/2) + cN$$

Siden $N = 2^m$ har rekursjonstreet $m = \log_2 N$ nivåer. Hvert nivå koster lineær tid i N (ca. $N/2$ *butterflies*), vi får derfor totalt:

$$B(N) = cN \log_2 N = O(N \log_2 N).$$

Dette gjør FFT til en svært effektiv metode for å beregne DFT, spesielt for store datasett. Dette illustreres i figur 1.



Figur 3: Operasjoner for DFT vs FFT som funksjon av antall prøver N .

2.4 Short-Time Fourier Transform (STFT)

STFT er en utvidelse av Fourier-transformasjonen som gjør det mulig å analysere tidsvarierende signaler. STFT deler signalet i overlappende rammer og beregner en DFT for hver ramme. Slik får vi frekvensinnholdet som funksjon av tid. Matematisk defineres STFT som vist under:

$$\text{STFT}\{x[n]\}(m, k) = \sum_{n=-\infty}^{\infty} x[n + mH]w[n]e^{-j2\pi kn/N}$$

hvor $w[n]$ er vindusfunksjonen, m er tidsindeksen for rammen, H er hop-størrelsen, og k er frekvensindeksen. Ved å bruke STFT kan vi visualisere tids-frekvensrepresentasjonen av signalet, noe som er nyttig i mange applikasjoner som talegjenkjenning, musikkprosessering og mer.

2.4.1 Rammelengde

Rammelengden (N) refererer til antall prøver i hver ramme som analyseres av STFT. Valget av rammelengde påvirker både tids- og frekvensoppløsningen. Generelt gjelder følgende forhold:

Tidsoppløsning: Jo kortere rammelengde, desto bedre tidsoppløsning. Noe som gjør at applikasjonen kan fange raske endringer i signalet og oppleves mer responsivt, men dette kan føre til redusert frekvensoppløsning.

Frekvensoppløsning: Jo lengre rammelengde, desto bedre frekvensoppløsning. Noe som gir mer presis frekvensanalyse, men dette kan føre til redusert tidsoppløsning.

$$\Delta f = \frac{f_s}{N} \quad [\text{Hz}]$$

hvor f_s er samplingsfrekvensen.

2.4.2 Hop-størrelse

I STFT er hop-størrelse (H) avstanden mellom påfølgende rammer i tid. Det bestemmer hvor mye hver ramme overlapper med den forrige. En mindre hop-størrelse gir mer overlapping, noe som kan forbedre tidsoppløsningen, men øker beregningskostnaden. En større hop-størrelse reduserer overlappingen, noe som kan føre til tap av informasjon om raske endringer i signalet. Valget av hop-størrelse avhenger av applikasjonen og ønsket balanse mellom tids- og frekvensoppløsning. En vanlig praksis er å bruke en hop-størrelse som er en brøkdel av rammelengden, for eksempel:

$$H = \frac{N}{4} \quad \text{eller} \quad H = \frac{N}{2}$$

Noe som gir 75% eller 50% overlapping mellom rammer. Dette gir en god balanse mellom tids- og frekvensoppløsning for mange applikasjoner. Ved disse valgene vil tidsoppløsningen være:

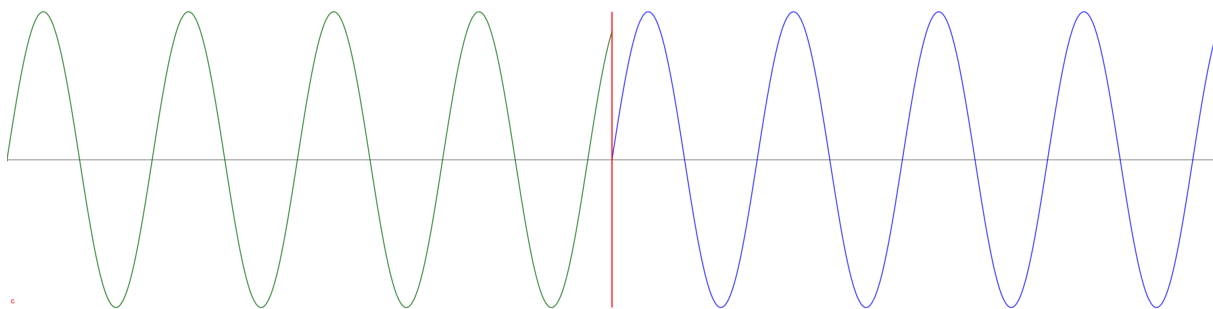
$$\Delta t = \frac{H}{f_s} \quad [\text{s}]$$

2.5 Vindusfunksjoner (Windowing)

Fourier-transform analyserer et tidskontinuerlig signal på formen

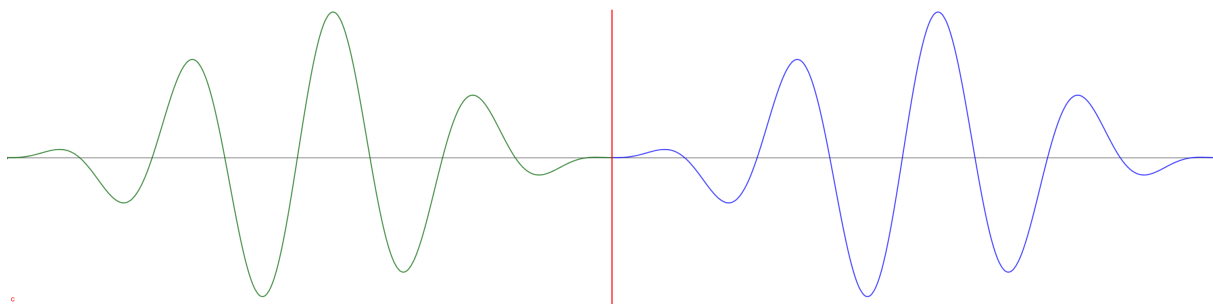
$$x(t) : \mathbb{R} \rightarrow \mathbb{R}$$

I praksis er lydsignaler tidsvarierende og ikke uendelig lange, og derfor deles de inn i kortere tidssegmenter (rammer). Hver ramme behandles som om den gjentas periodisk, slik at vi kan analysere hvilke frekvenser som dominerer innenfor et gitt tidsrom. Slik kan man finne hvilke frekvenser som er til stede i et gitt tidsintervall Δt [s]. Når signalet deles i rammer oppstår skarpe overganger ved kantene. Disse diskontinuitetene fører til at energien i spekteret “lekker” utover flere frekvenser, noe som kalles *spectral leakage*. Dette skjer fordi brå avslutning i tidsdomenet (diskontinuiteter) innebærer et bredt frekvensinnhold, slik at energien i spekteret lekker utover flere frekvenskomponenter. Et eksempel på dette vises i figuren under, der et sinusoidalt signal blir avkuttet brått ved rammens start og slutt:



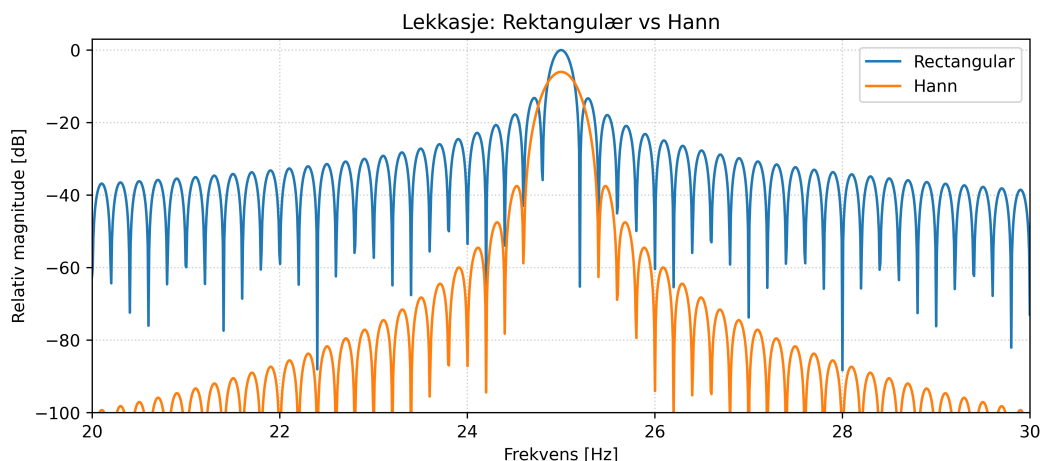
Figur 4: Eksempel på spectral leakage i et sinusoidalt signal ved bruk av rektangulært vindu

For å redusere lekkasje benytter man vindusfunksjoner, som gradvis demper signalet mot null ved rammekantene. Dette gjør overgangen mykere og reduserer diskontinuiteter i tidsdomenet (Se figur 5).



Figur 5: Eksempel på Hanning-vindu som jevner ut rammekantene sammenlignet med rektangulært vindu

Figur 6 viser at Hanning-vinduet gir betydelig lavere sidelober i frekvensdomenet enn det rektangulære vinduet. Dette betyr at mindre energi lekker over i nabofrekvenser, og frekvensanalysen blir mer presis.



Figur 6: Sammenligning av Hanning-vinduet og rektangulært vindu i frekvensdomenet

Frekvensdomenet plottet i figur 6 er basert på følgende tidsfunksjon:

$$x(t) = \sin(25t), \quad t \in [0, 5]$$

Slik kan vi oppsummere effekten av vindusfunksjonene:

Hanning-vinduet: Reduserer lekkasje betydelig sammenlignet med det rektangulære vinduet. Dette skyldes at vinduet taperer rammen mot null ved endene, slik at diskontinuiteter i tidsdomenet blir mindre. Uten gain-korreksjon får man lavere toppamplitude (coherent gain $\approx 0,5$), altså blir den målte toppen omtrent halvert. Hovedloben blir bredere ($\approx 4 \Delta f$ mot $\approx 2 \Delta f$ for rektangel), mens sidelobene er betydelig lavere, noe som reduserer lekkasje til andre frekvenser.

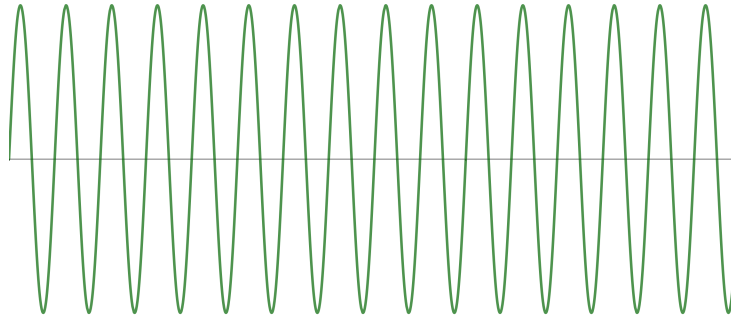
Rektangulært vindu: Har skarpe overganger ved rammeendene, som gir sterke diskontinuiteter i tidsdomenet og dermed betydelig spectral leakage i frekvensdomenet. Energien fra en tone fordeles over flere frekvensbinner, noe som kan maskere svake nabotoner og forvrengne tolkningen. Hovedloben er smalere enn for Hanning-vinduet ($\approx 2 \Delta f$ mot $\approx 4 \Delta f$), men sidelobene er høyere og avtar saktere, noe som øker lekkasjen.

2.5.1 Hanning-vinduet

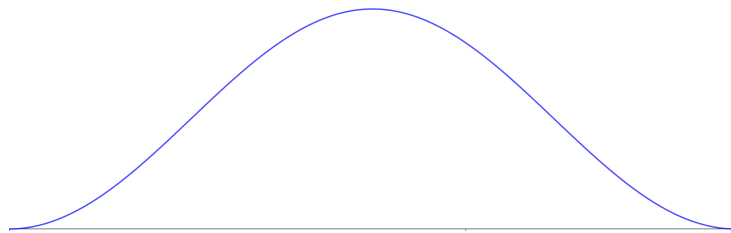
Blant vindusfunksjoner er Hanning-vinduet en av de mest brukte. Den er definert som:

$$w[n] = \frac{1}{2} \left(1 - \cos \left(\frac{2\pi n}{N-1} \right) \right) \quad (4)$$

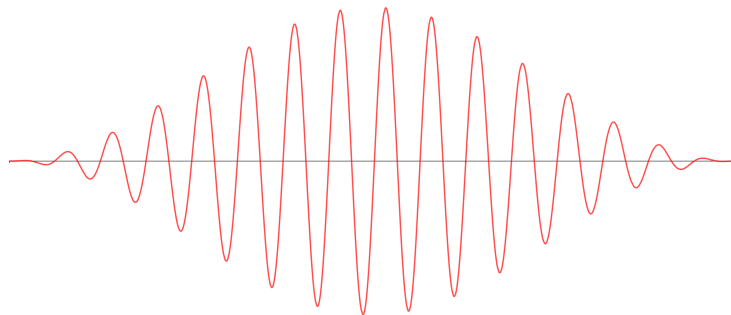
Der N er antall prøver i vinduet. Hanning-vinduet gir en god balanse mellom frekvensoppløsning og lekkasjereduksjon. I koden multipliseres hvert signalutdrag med et Hanning-vindu før FFT-beregningen, slik at de beregnede frekvenskomponentene bedre representerer de faktiske tonene i lyden vi analyserer. Grafene under viser hvordan et signal $x(t)$ (figur 7) vil se ut etter et Hanning-vindu transformasjonsfunksjon w (figur 8). Resultatet er en funksjon $f \cdot w$ (figur 9):



Figur 7: Originalt signal som funksjon x



Figur 8: Hanning-vinduet som funksjon av x



Figur 9: Hanning-vinduet funksjonen ganget med originalt signal

2.5.2 Parabolsk toppinterpolasjon (Quadratic interpolation)

FFT gir diskrete frekvenspunkter, men den faktiske frekvensen til en tone kan og vil ofte ligge mellom to slike punkter. Parabolsk toppinterpolasjon muliggjør estimasjon av toppfrekvensen mer presist ved å tilpasse en parabel rundt det største spektralt punktet og dets to naboer. Uten interpolasjon får vi en oppløsning gitt ved:

$$\Delta f = \frac{f_s}{N}$$

der f_s er samplingsfrekvensen og N er antall FFT-frekvenspunkter. Dette innebærer at den faktiske toppfrekvensen til et signal ligger mellom to slike FFT-frekvenspunkter. Vi

benytter parabolisk toppinterpolasjon for å estimere toppfrekvensen mer presist. Prinsippet bak metoden er at toppen av FFT-magnitude-spekteret kan beskrives som en parabol rundt maksimumet. Vi kan beskrive en slik parabolisk andregradsfunksjon med tre punkter (toppunkt og to naboer). Dette uttrykkes matematisk slik under.

$$\delta = \frac{1}{2} \cdot \frac{M_{k-1} - M_{k+1}}{M_{k-1} - 2M_k + M_{k+1}}$$

Der M_k er amplituden ved toppunktet (M_{k-1}, M_{k+1}) naboene. Den estimerte frekvensen kan da uttrykkes som:

$$\hat{f} = (k + \delta) \cdot \frac{f_s}{N} \quad (5)$$

2.6 Root Mean Square (RMS)

RMS er nyttig når man skal måle eller sammenligne styrken til et signal over tid. Det brukes også i støyanalyse for å finne et representativt “gjennomsnittsnivå” av lydintensiteten. RMS gir et mål på den **effektive verdien av amplituden** til et signal, og gir dermed et bedre bilde av den **opplevde energien i lyden** enn et vanlig gjennomsnitt. I vårt system brukes RMS for å stabilisere frekvensanalysen ved å unngå et svak bakgrunnsstøy blir tolket som betydningsfullt signal.

Matematisk defineres RMS for et diskret signal $x[n]$ med N prøver som:

$$x_{RMS} = \sqrt{\frac{1}{N} \sum_{n=1}^N x[n]^2} \quad (6)$$

Matematisk ser vi at beregningen innebærer å kvadrere hver verdi av $x[n]$ for å gjøre alle amplituder positive, deretter tar gjennomsnittet og til slutt roten av resultatet. RMS kan brukes til å måle lydnivået over tid, sammenligne styrken til ulike signaler og redusere støy.

3 Metode

I dette prosjektet har vi benyttet oss av Python som programmeringsspråk for implementering av Fourier-baserte analyser. Vi vil derfor gå gjennom oppbyggingen av koden, og hvordan de ulike delene fungerer sammen for å utføre STFT-analyse i sanntid. Alt av koden og tilhørende informasjon er vedlagt i vedlegg [1](#).

3.1 Oppbygging av koden

For å forstå hvordan vi utfører STFT i koden tar vi for oss følgende deler:

- Initialverdier
- Sanntids innhenting av lyd
- Fourier-analyse av lydsignalet
- Støygating og glatting
- Beregning av tone og note

3.2 Initialverdier

Vi har følgende initialverdier:

Tabell 1: Initialverdier for STFT-analyse

Parameter	Verdi
f_s	48000
N	2048
H	256

Med disse verdiene har vi følgende beregnede parametere for, frekvensoppløsning, tidsoppløsning og antall FFT-vinduer per sekund:

$$\Delta f = \frac{f_s}{N} = \frac{48000}{2048} \approx 23.44 \text{ Hz}$$

$$\Delta t = \frac{H}{f_s} = \frac{256}{48000} \approx 5.33 \text{ ms}$$

$$R = \frac{f_s}{H} = \frac{48000}{256} = 187.5 \text{ rammer/s}$$

$$t_{\text{ramme}} = \frac{N}{f_s} = \frac{2048}{48000} \approx 42.67 \text{ ms}$$

3.3 Sanntids innhenting av lyd

Lyd hentes kontinuerlig inn fra mikrofonen ved hjelp av biblioteket PyAudio, som muliggjør direkte tilgang til lydstrømmer i sanntid. Signalet samles inn som 16-biters heltallsprøver (int16) og deles opp i mindre blokker, eller såkalte chunks, lik H (256) prøver. Disse blokkene legges i en kø (queue), som fungerer som et bindeledd mellom to tråder: en som produserer lyddata, og en som forbruker og analyserer dem.

Denne produsent og forbruker-strukturen gjør at innsamlingen av lyd og selve analysen kan foregå parallelt uten å forstyrre hverandre. Hvis køen blir full, fjernes de eldste dataene automatisk for å unngå forsinkelser og sikre jevn oppdatering.

3.4 Fourier-analyse av lydsignalet

For å analysere signalet brukes Fast Fourier Transform (FFT), som regnes ut med funksjoner fra NumPy-biblioteket. FFT omgjør lydsignalet fra tidsdomenet til frekvensdomenet, slik at vi kan se hvilke frekvenser som er mest dominerende i hvert lydvindu.

```
class AudioAnalyzerConsumer(threading.Thread):

    def __init__(self, queue, my_window=None):
        ...
        self.window = np.hanning(N).astype(np.float32) # w[n]
        ...

    def run(self):
        while not self._stop_consumer:
            ...
            while self.total >= N:
                ...
                frame_windowed = frame * self.window # x[n] * w[n]
                ...
                freq_domain = np.fft.rfft(frame_windowed, n=N)
                self.mags = np.abs(freq_domain)
                ...
```

I koden ovenfor ser vi hvordan vindusfunksjonen (her en Hann-vindu) påføres lyddrammen før FFT beregnes. Dette reduserer lekkasjeeffekter i frekvensspekteret. Resultatet av FFT-en lagres som magnituden av frekvenskomponentene i variabelen `self.mags` for videre analyse.

3.5 Støygating og glatting

For å redusere påvirkningen av bakgrunnsstøy brukes en enkel RMS-basert støygate. Ved hver FFT-beregning estimeres energien i rammen med hjelp av den matematiske definisjonen for RMS vist i Ligning (6). En glattet støyterskel beregnes kontinuerlig ved hjelp av en eksponentiell glatting.

I koden under (side 14) ser vi hvordan RMS-verdien av den vindusveide lyddrammen beregnes og normaliseres med vinduets RMS-verdi for å få en relativ effekt. Denne verdien sammenlignes deretter med en dynamisk støyterskel som oppdateres kontinuerlig basert på tidligere rammer. Hvis RMS-verdien er under terskelen, hopper analysen over denne rammen og fortsetter til neste. Denne metoden kalles *RMS-gating* og hindrer at bakgrunnsstøy og lavintensive signaler analyseres som gyldige toner.

```

class AudioAnalyzerConsumer(threading.Thread):

    def __init__(self, queue, my_window=None):
        ...
        self.win_rms = np.sqrt(np.mean(self.window**2))
        self.noise = 0.004 * INT16_MAX
        self.noise_multiplier = 3
        self.alpha = 0.99
        ...

    def run(self):
        while not self._stop_consumer:
            ...
            while self.total >= N:
                ...
                rms = np.sqrt(np.mean(frame_windowed**2))
                rms /= self.win_rms

                if rms < self.noise_multiplier * self.noise:
                    self.noise *= self.alpha
                    self.noise += (1 - self.alpha) * rms

            RMS_THRESHOLD = self.noise_multiplier * self.noise

            if rms < RMS_THRESHOLD:
                continue # skip lav effekts rammer
            ...

```

3.6 Beregning av tone og note

Når støygrensen passeres og signalet anses som gyldig, fortsetter analysen for å beregne hvilken tone og note som er tilstede i signalet. Dette gjøres i hovedsak i to trinn frekvensestimering og notefinning. Etter at FFT-en er gjennomført, lokaliseres maksimumet i det beregnede spekteret innenfor et gjenkjennbart frekvensområde (for eksempel 20 Hz til 8 kHz). Indeksen for maksimalverdien tilsvarer den mest framstående (dominerende) frekvensen i den aktuelle rammen, men her er det viktig å ta hensyn frekvensoppløsningen, og hann vinduets påvirkning på sidelobene.

Som vist i koden under (side 15), benyttes `pick_peaks_nms` for å identifisere de mest fremtredende frekvensene i spekteret, med hensyn til en eksklusjonsmargin for å unngå nære duplikater forårsaket av sidelobene i vinduet. Dette gjøres ved å hente ut kandidater for toppfrekvenser, sortere dem etter styrke, og deretter velge de sterkeste som er tilstrekkelig adskilt fra hverandre. Videre benyttes parabolisk toppinterpolasjon for å estimere den eksakte frekvensen til toppen mer presist, som forklart i Teoridelen (side 10). Den estimerte frekvensen konverteres deretter til en musikalsk note ved hjelp av formelen for notefinning basert på liketemperert stemming.

```

...
...

```

```
1
2 def pick_peaks_nms(mags, num_peaks=5, exclusion_zone=5):
3     peaks = []
4     for i in range(1, len(mags) - 1):
5         if mags[i] > mags[i - 1] and mags[i] > mags[i + 1]:
6             peaks.append(i)
7     peaks = sorted(peaks, key=lambda x: mags[x], reverse=True)
8     return [p for p in peaks if all(abs(p - q) > exclusion_zone for q in
9         ↪ peaks[:num_peaks])]
```

4 Signalanalyse med Python

I dette kapitlet vil vi se på resultatene fra python programmet vi har laget, *Pitch Perfect* Presenter hva kapitlet innebærer. Info om programmet: Resultater: REFERER TIL TEORI OG METODE KAPITTEL

4.1 Pitch Perfect

Absolutt gehør, eller på engelsk perfect pitch, er evnen hos noen mennesker til å identifisere musikktoner i sanntid ved å kunne navngi hvilken note som spilles eller synges. *Pitch Perfect* er et python program som brukes til å analysere lydsignaler og visualisere dem, og simulerer absolutt gehør ved bruk a fourieranalyser. Hovedideen for programmet er at det skal kunne analysere lydsignaler og visualisere dem i sanntid ved bruk av STFT.

PyAudio bruker systemets standard inndataenhet som kilde for lydsignalet. Når programmet kjøres startes en tråd som håndterer lydinnhenting, og en annen tråd som håndterer analysen og visualiseringen. Ved hjelp av samtidige tråder som parallelt håndterer lydinnhenting, analyse og visualiseringen, kan programmet kjøre i sanntid, i stedet for at programmet må bli gitt en lydfil først. Etter start av signal prosessering listes de høyeste 10 frekvensene i lydsignalet, med tilsvarende noter. Etter stans av signal prosessering kan frekvensspekteret, spektogrammet og tidsdomene signalet for det siste lydvinduet presenteres.

4.2 Resultater

...

5 Diskusjon

6 Feilkilder

Flere forhold kan ha påvirket nøyaktigheten og stabiliteten i resultatene våre. En viktig feilkilde er bakgrunnsstøy og variasjoner i opptaksmiljøet. Selv om programmet har en adaptiv støygate, vil endringer i bakgrunnsstøy, for eksempel tastetrykk, pust eller romklang, kunne føre til at terskelen enten blir for høy eller for lav. Dette kan igjen føre til at svake toner blir ignorert, eller at støy feiltolkes som en faktisk tone.

En annen feilkilde ligger i opptaksutstyret. Mikrofonens kvalitet og plassering påvirker både lydstyrke og frekvensrespons, og enkle datamikrofoner kan forvrengte signalet eller ha begrenset båndbredde. Dette påvirker resultatet av Fourier-analysen, spesielt i de høyere frekvensene.

I tillegg er brukerens egen påvirkning en viktig faktor. Avstanden til mikrofonen, retningen lyden kommer fra, og hvor stabil tonen som spilles faktisk er, påvirker hvordan signalet fanges opp. I tillegg kan små variasjoner i stemningen til instrumentet eller ustabilitet i tonehøyden føre til at analysen viser et avvik selv om metoden i seg selv fungerer korrekt.

7 Konklusjon

Referanser

- [1] L. N. Bakken, *Telegrafligningen*, PDF-dokument, Utledelse og løsning av telegrafligningen, vedlagt i prosjektet, 2024. adresse: [Telegrafligningen.pdf](#)
- [2] G. contributors, *Telegrapher's Equation in Electrical Engineering*, Online; accessed 9 October 2025, 2025. adresse: <https://www.geeksforgeeks.org/electrical-engineering/telegraphers-equation/>
- [3] W. contributors, *Telegrapher's equations — Solutions of the telegrapher's equations as circuit components*, Online; accessed 9 October 2025, 2025. adresse: https://en.wikipedia.org/wiki/Telegrapher%27s_equations#Solutions_of_the_telegrapher%27s_equations_as_circuit_components
- [4] A. Croft, P. Davison, M. Hargreaves og J. Flint, *Engineering Mathematics*. Pearson Education Limited, 2017.
- [5] T. Instruments, *Implementing a Variable-Length Cat5e Cable Equalizer*, PDF-dokument, 2020. adresse: https://www.ti.com/lit/an/sboa125/sboa125.pdf?ts=1759787274591&ref_url=https%253A%252F%252Fwww.google.com%252F#:~:text=Table%20%20shows%20the%20typical%20electrical%20characteristics%20of%20Cat5e%20cable.&text=Applying%20Equation%20%20to%20the,is%20equal%20to%20100.5%CE%A9.
- [6] W. H. H. Jr. og J. A. Buck, *Engineering Electromagnetics*, 8th Edition. New York: McGraw-Hill Education, 2018, Kapittel om transmisjonslinjer og propagasjonskonstanten. sjekket 6. okt. 2025. adresse: <https://dn790009.ca.archive.org/0/items/electromagnetic-field-theory/William%20H.%20Hayt%2C%20Jr.%20and%20John%20A.%20Buck%20-%20Engineering%20Electromagnetics-McGraw-Hill%20Education%20%282018%29.pdf>
- [7] S. Palermo. «Propagation on Transmission Lines.» Forelesningsnotat i ECEN689, sjekket 6. okt. 2025. adresse: https://people.engr.tamu.edu/spalermo/ecen689/lecture3_ee689_tlines.pdf
- [8] K. Technologies, *Root Cause of Eye Closure*, Webside, 2021. adresse: <https://docs.keysight.com/eesofapps/post-5-root-cause-of-eye-closure-678068340.html>

Vedlegg

[1] GitHub repository for prosjektet:

Url: ... Commit: ... (commit fra ...)

Inneholder all kode og data brukt i prosjektet, mer informasjon i README.md filen.