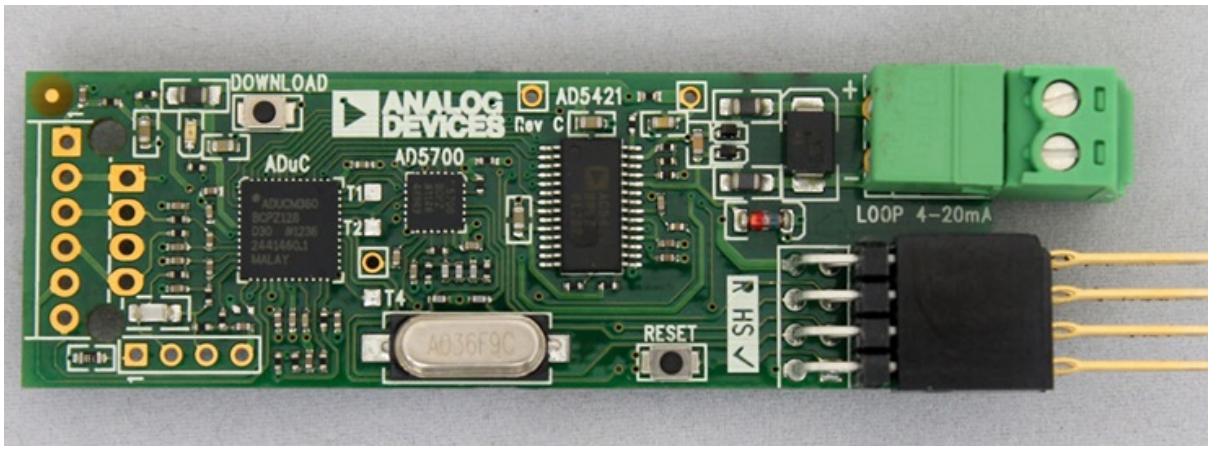
E4DSA Case 1 - FSK transmission

Janus Bo Andersen [1]

17. februar 2020



[1] ja67494@post.au.dk

Indhold

1. Indledning

Første case i E4DSA er transmission af digital information vha. metoden Audio Frequency Shift Keying (AFSK). Dette dækker over en modulationsteknik, hvor digital data repræsenteres ved forskellige frekvenser, og en tone moduleres til disse frekvenser. Der benyttes en "audio"-tone, så signaler kan overføres per radio, telefon, luft eller andet lydbærende medium. FSK-metoden er også bag DTMF-signaler, kendt fra tonerne i trykknap-tlf. FSK er også teknikken bag HART-protokollen i fx industriautomatisering. Forsiden viser et board med AD5700 IC'en, der implementerer et FSK HART-modem. Hurtigere kommunikation implementeres ofte vha. PSK, QAM, og lign., som fx i (A)DSL.

1.1 Opsummering af teori

Vi antager en kommunikationskanal, her en luftbåret audiokanal, der lader alle frekvenser i transmissionsbåndet passere uændret. Det antages, at amplituder kan lagres kontinuert (uden kvantiseringsfejl). I første omgang antages kanalen støjfri. Der tilføjes senere støj.

På denne kanal skal overføres $N_{sym} = 256$ forskellige symboler, med koder fra 0 til 255 (diskrete værdier). Koderne fortolkes op mod den udvidede ASCII-tabel. Fortolkning er egentlig arbitrær ift. algoritmerne, og koderne kunne også fortolkes som tal eller et andet tegnsæt.

Transmissionen er asynkron (ingen fælles clock), så en standard skal fastlægge **baudrate**, **transmissionsbånd** og antal mulige symboler, nævnt ovf.

Baudraten definerer transmitterede symboler per sekund. Symboltiden er den inverse af baudrate. To ASCII-tegn per sekund er en baudrate på 2 Bd og en symboltid på $T_{sym} = \frac{1}{2}$ s.

Transmissionsbåndet er f_1 til f_2 Hz. Der benyttes en audiokanal, så det giver mening at lægge båndet i det hørbare spektrum. Således også mere sandsynligt, at udstyr gengiver frekvenser korrekt. Symbolkoderne, $S \in \{0, 1, \ldots, N_{sym} - 1\}$, spredes ud over transmissionsbåndet med ens afstand. Kodning og afkodning til/fra symbolfrekvens:

$$f_{sym}(S) = f_1 + S\frac{f_2 - f_1}{N_{sym} - 1} \tag{1.1}$$

$$S(f_{sym}) = \frac{f_{sym} - f_1}{f_2 - f_1} \cdot (N_{sym} - 1) \tag{1.2}$$

Bemærk, at den oprindelige `FSKgen`-funktion har en lille indekseringsfejl på 1, og ikke tillader ASCII-symbolet `NUL`. Indekseringsfejlen er rettet i implementering af `FSKgen2`.

Eksempel: ASCII-tegn A har symbolkode (decimaltal) $S = 65$. I et bånd fra $f_1 = 1000$ Hz til $f_2 = 2000$ Hz kodes det med symbolfrekvens $f_{sym} = 1255$ Hz. B kodes med $f_{sym} = 1259$ Hz. Eksemplet viser, at de to symboler i dette bånd ligger inden for 4 Hz. Så en detektionsalgoritme ville her skulle have en opløsning højere end 4 Hz, eller båndet kan tilpasses til den tilgængelige frekvensopløsning.

2. Opgave 1: Signalgenerering/-kodning

```
clc; clear all; close all;
```

2.1 Transmissionssignal

Som nævnt i teoriafsnittet, kræver metoden en aftale om hvordan transmissionen skal foregå. Dette gemmes i en struct. Derefter benyttes `FSKgen2`-funktionen til at enkodere payload.

```
comm_std.baudrate = 10;                 % symboler/sekund
comm_std.T_sym = 1/comm_std.baudrate;   % sekunder (/symbol)
comm_std.N_sym = 256;                   % ASCII-tabel
comm_std.f1 = 1e3;                      % Nedre grænse i frekv.bånd
comm_std.f2 = 2e3;                      % Øvre grænse i frekv.bånd
comm_std.fs = 5e3;                      % samplingsfrekvens 5 kHz (>> 2*f2)

% Payload er beskeden, der skal transmitteres
payload = 'DSA';

% Generer signal, der kan transmitteres over audiokanal.
signal = FSKgen2(payload, comm_std);

%soundsc(signal, comm_std.fs);
```

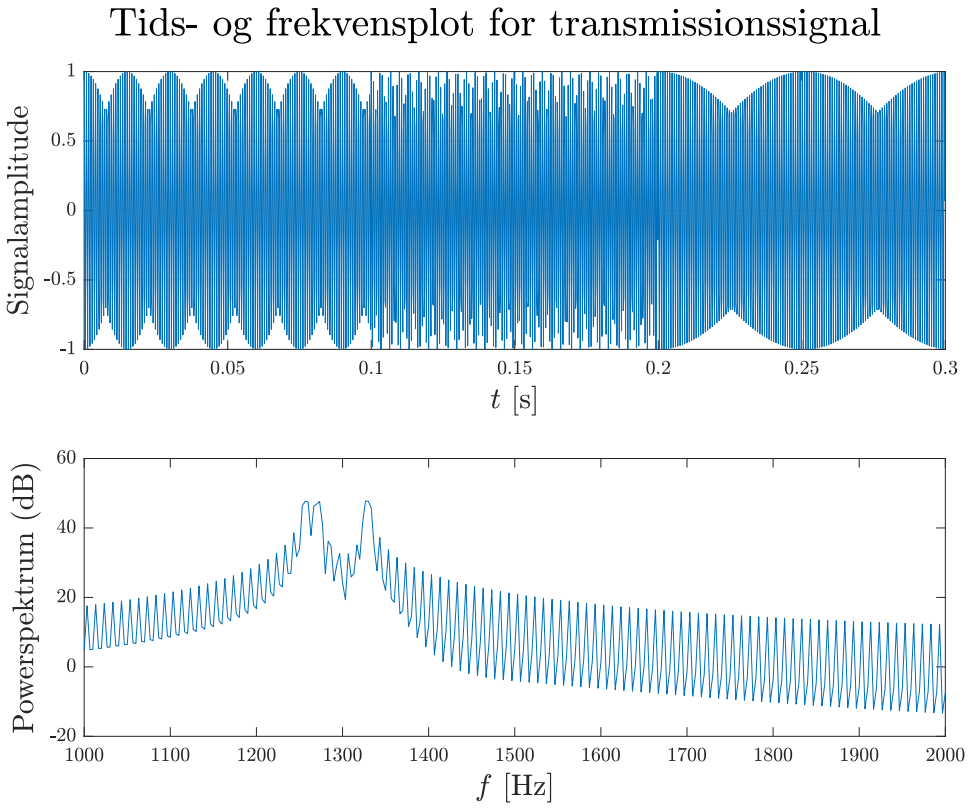
2.2 Signalindhold

Signalindholdet vises i både tids- og frekvensdomæne. For frekvensdomænet vises et powerspektrum. Kun frekvenssamples i transmissionsbåndet vises.

Signalet er per konstruktion ikke-stationært. Det ses i plottet af lydbølgen: Der ses 3 sektioner med hver sit tydeligt forskellige lydsignal, hver svarende til et symbol i payload. Hver sektion har per konstruktion en tidslængde $T_{sym} = \frac{1}{baudrate}$, og indeholder $n = T_{sym} \cdot f_s$ datapunkter.

Frekvenskomponenterne ses via powerspektrum, som viser, at indholdet er 3 forskellige frekvenser, repræsenterende fra lavest til højest A (65 $\rightarrow$ 1255 Hz), D (68 $\rightarrow$1267 Hz) og S (83 $\rightarrow$ 1326 Hz).

```
show_timefreq(signal, comm_std);
```



Bemærk også, at enkodering med ASCII vil betyde, at de fleste beskeder med normalt tekstindhold vil "klumpe" sig sammen i en mindre del af spektret. Dette fordi symbolkoder 0-31 samt 127-255 er infrekvente i normal tekst (fraregnet æÆ øØ åÅ). En bedre mapping fra symbol til symbolkode ville benytte hele transmissionsbåndet for beskeder med hyppige tegn. Det ville mindske fejlrate og forbedre støjimmunitet sfa. større frekvensafstand mellem symboler.

2.3 Spektrogram

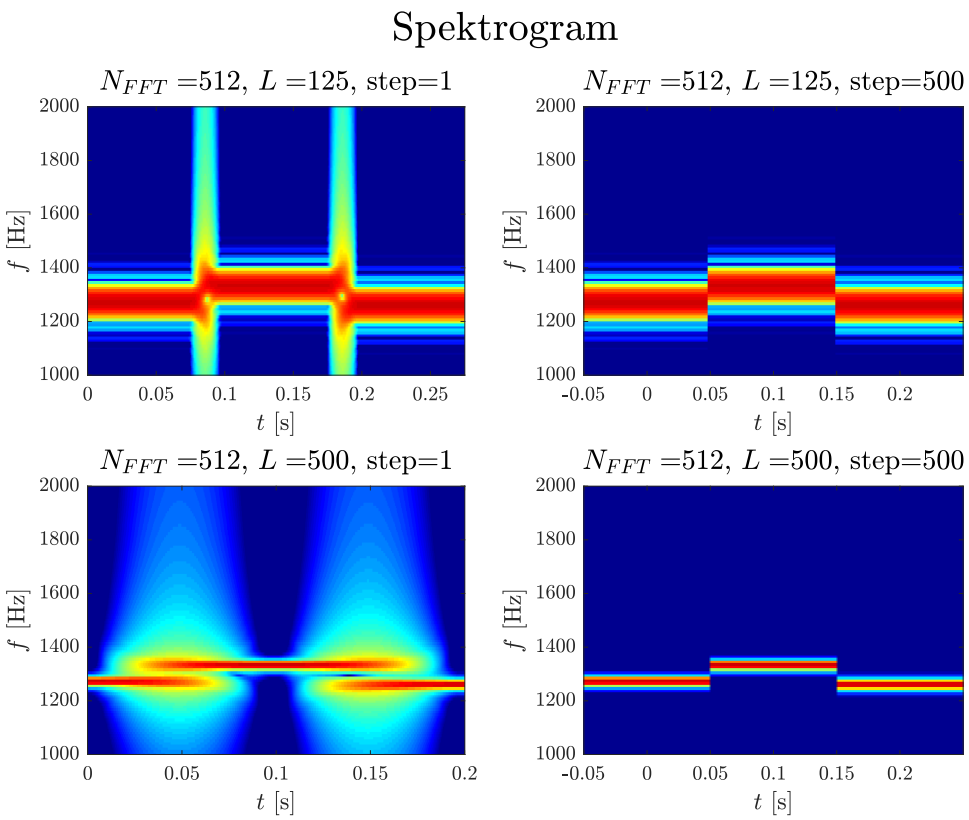
Rækkefølgen af symboler kan ikke udledes fra et powerspektrum, da DFT-analysen er lavet på hele tidsserien. Frekvensindhold over tid analyseres med et spektrogram. Dvs. en række kortere DFT'er (= STFT'er) udføres på et rullende vindue.

Der er et trade-off i længden af vinduet: Et langt vindue giver finere frekvensopløsning ($\Delta f = \frac{f_s}{L}$). Men giver også en grovere tidsopløsning, da DFT'en regnes på et længere tidsinterval ($T_{STFT} = L \cdot T_s = \frac{L}{f_s}$). De to mål er altså inverse, med trade-off i at have finmasket adskilning i enten tid eller frekvens.

De 2 spektrogrammer nedenfor t.v. illustrerer den trade-off: I øverste række t.v. benyttes et kort vindue, og der er klar separation i tid for hvornår en frekvens er indeholdt i signalet eller ej. Altså fin tidsopløsning. Til gengæld er frekvensspektret bredt, altså grov frekvensopløsning. Det ville være svært at pin-pointe en eksakt frekvens, eller at adskille flere tætliggende frekvenser. I nederste række t.v., med et langt vindue, er situation omvendt.

```
1 nsamp = comm_std.fs * comm_std.T_sym;          % samples per symbol
2 nup = 2^nextpow2(nsamp);                      % oprundet samples per symbol
```

```
Ls = [nsamp/4, nsamp/4, nsamp nsamp];        % forskellige vindueslængder
steps = [1, nsamp, 1, nsamp];                % forskellige stepstørrelser

figure
iterated_spectrogram0(signal, ...            % transmissionssignal
              Ls, ...                        % vindueslængde, L
              nup, ...                       % FFT-størrelse, N_fft
              steps, ...                     % stepstørrelse
              comm_std.fs, ...               % samplingsfrekvens
              [comm_std.f1, comm_std.f2]);   % transmissionsbånd
```



De 2 spektrogrammer t.h. illustrerer, at forhåndsinformation om signalet er værdifuldt. Stepstørrelsen er tilpasset antallet af samples for hvert symbol. Nu er der tydelig separation i tid, og med et langt vindue fås også en fin frekvensopløsning. Denne observation benyttes til detektionsalgoritmen.

Overordnet viser spektrogrammerne, at frekvensindhold skifter efter baudrate, med $T_{sym} = 0.1$ s. Dette illustrerer også princippet i FSK: Frekvensen moduleres over tid til et antal diskrete frekvenser, og hver unik frekvens repræsenterer et symbol. Nu kunne signalet principielt afkodes manuelt ved aflæsning af frekvens og efterfølgende konvertering til symbolkode og opslag i ASCII-tabellen.

3. Opgave 2: Dekodning

3.1 Metodevalg

Udover aflæsning fra spektrogrammet findes adskillige løsningsmetoder til dekodning:

- Filtrering, fx en filterbank med båndpasfiltre.
- Goertzels algoritme, udregner specifikke dele af DFT.
- STFT og efterfølgende thresholding direkte på spektrogram-data.
- DFT, men kun med realdelen (cosinus-delen) af eksponentialfunktionen.
- Dele signalet op i ikke-overlappende sektioner, køre hver sektion igennem en DFT, og bruge en beslutningsalgoritme til at afgøre væsentligste frekvens i hver sektion af signalet.
- Samme, men kun med udvalgte frekvenskomponenter, der ligger "tæt" på symbolfrekvenserne.

Filtre og Goertzel er for upraktisk pga. det høje antal symboler. Og da vi har en constraint om ikke at benytte MATLABs FFT, vælger jeg en tilgang baseret på de to sidstnævnte, og justerer DFT'en til at være målrettet symbolfrekvenserne, og derfor med færre bins og beregninger.

DFT'en omskrives i vektornotation for at udvikle en simpel detektionsalgoritme, der kan implementeres som én matrixmultiplikation. Det burde være hurtigere - i det mindste sjovere (valuta for ETALA-pengene).

3.2 Teori

DFT'en er et skift af basis i $\mathbb{C}^N$, og hver frekvenssample er et indre produkt mellem en samplevektor fra $\mathbb{R}^N$ og en Fourier-basisvektor fra $\mathbb{C}^N$. Dvs. en projektion af samplevektoren ind på Fourier-basisen. Fourier-basisen udspænder $\mathbb{C}^N$, og består af N ortogonale vektorer, hver med N elementer. Fourierbasisen er $\vec{w}^{(k)} \equiv \begin{bmatrix} 1 & w^k & \cdots & w^{nk} & \cdots & w^{(N-1)k} \end{bmatrix}$, for $k = 0, \ldots, N-1$, og $w = e^{-j\frac{2\pi}{N}}$. Det er centralt i DFT'en, at denne basis er ortogonal, så $\langle \vec{w}^{(k)}, \vec{w}^{(h)} \rangle = 0$ for $k \neq h$. Signalvektoren er $\vec{x} = \begin{bmatrix} x_0 & \cdots & x_n & \cdots & x_{N-1} \end{bmatrix}$. Her er valgt konventionen, at signalvektorer er rækkevektorer. DFT omskrives derfor med vektornotation til:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn} = \langle \vec{w}^{(k)}, \vec{x} \rangle = \vec{x}\vec{w}^{(k)\top} \tag{3.1}$$

Hvor $^\top$ angiver transponering [1]. En rækkevektor af alle frekvenssamples er givet ved matrixproduktet

$$\vec{X} = \vec{x}\mathbf{W}$$

Med $\mathbf{W} = \begin{bmatrix} \vec{w}^{(0)\top} & \cdots & \vec{w}^{(k)\top} & \cdots & \vec{w}^{(N-1)\top} \end{bmatrix}$ værende basisskiftematricen.

[1] Jeg har snydt lidt ved at benytte normal transponering i stedet for kompleks-konjugeret transponering, men ender ved samme resultat fordi eksponenten i w allerede er blevet givet negativt fortegn.

3.3 Detektionsalgoritme

3.3.1 Afkodningsmatrix og Powerspektrum

DFT'en dækker hele frekvensspektret (udspænder $\mathbb{C}^N$), men til AFSK er kun behov for at detektere $N_{sym} = 256$ *specifikke* frekvenser. Dvs. vi kan nøjes med at projicere samplevektoren ind i et underrum af $\mathbb{C}^N$. Dette "symbolunderrum", $\mathbb{S}$, udspændes af 256 specifikke basisvektorer. Hver basisvektor er bygget fra en symbolfrekvens, og symbolfrekvenserne er som tidligere beskrevet:

$$f_{sym}(S) = f_1 + S\frac{f_2 - f_1}{N_{sym} - 1} \tag{3.2}$$

Med $S = 0, \ldots, N_{sym} - 1$. DFT'en benytter en normaliseret frekvens $2\pi\frac{k}{N}$, der ligger mellem 0 og 2π (spejling fra π til 2π). Til symbolfrekvenserne bruges tilsvarende den normaliserede digitalfrekvens:

$$2\pi\frac{f_{sym}(S)}{f_s} \tag{3.3}$$

Denne *skal* ligge mellem 0 og π. Det er sfa. samplingteoremet, så den øvre frekvens $f_2 < \frac{1}{2}f_s$. Omskrives DFT'en med disse ændringer, fås:

$$X_{\mathbb{S}}(S) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi\frac{f_{sym}(S)}{f_s}n} = \langle\vec{w}_{\mathbb{S}}^{(S)}, \vec{x}\rangle = \vec{x}\vec{w}_{\mathbb{S}}^{(S)\top} \tag{3.4}$$

Vi kunne nøjes med at bruge realdelen af den komplekse eksponentialfunktion i det indre produkt, dvs. blot cosinus-delen, fordi signalet netop er genereret sådan i `FSKgen2`. Dog beholdes hele den komplekse eksp.fkt. De 256 basisvektorer for "symbolunderrummet", $\mathbb{S}$, defineres med $S = 0, \ldots, 255$ ved:

$$\vec{w}_{\mathbb{S}}^{(S)} \equiv \begin{bmatrix} 1 & e^{-j2\pi\frac{f_{sym}(S)}{f_s}\cdot 1} & \cdots & e^{-j2\pi\frac{f_{sym}(S)}{f_s}\cdot n} & \cdots & e^{-j2\pi\frac{f_{sym}(S)}{f_s}\cdot(N-1)} \end{bmatrix} \tag{3.5}$$

Disse basisvektorer er lineært uafhængige og udspænder det ønskede underrum. Dvs. alle *relevante* frekvenser kan detekteres. Basisvektorerne er givetvis ikke ortogonale, så selvom signalfrekvenserne ligger præcis oveni analysefrekvenserne, fås ikke et "rent" spektrum. Der vil altså være en slags "lækage". Det er acceptabelt, fordi vi kun bruger spektrum til at selektere på højeste power.

Værdien N i ovenstående sættes efter længden på STFT. I spektrogram-eksemplet ovenfor ville $N = 500$ give mening (step=500).

Detektionen foregår, analogt til DFT'en, ved en matrixmultiplikation med afkodningsmatricen:

$$\vec{X}_{\mathbb{S}} = \vec{x}\mathbf{W}_{\mathbb{S}} \tag{3.6}$$

Hvor

$$\mathbf{W}_{\mathbb{S}} = \begin{bmatrix} \vec{w}_{\mathbb{S}}^{(0)\top} & \cdots & \vec{w}_{\mathbb{S}}^{(S)\top} & \cdots & \vec{w}_{\mathbb{S}}^{(N_{sym}-1)\top} \end{bmatrix} \tag{3.7}$$

Et powerspektrum kan regnes ved[2]:

$$|X_{\mathbb{S}}(S)|^2 = \vec{X}_{\mathbb{S}} \odot \vec{X}_{\mathbb{S}}^* = \vec{X}_{\mathbb{S}}\,\mathrm{diag}(\vec{X}_{\mathbb{S}}^*) \tag{3.8}$$

[2] Operatoren * angiver her kompleks konjugering. Operatoren $\odot$ er elementvis multiplikation. Operatoren diag$(\cdot)$ tager elementerne fra en vektor og placerer dem på diagonalen af en 0-matrix.

3.3.2 Algoritmer

En afkodning beregnes for hver sektion af signalet. En sektion dækker et symbol. Frekvensen for den sample fra $|X_{\mathbb{S}}(S)|^2$ i hver sektion, som har den største power, erklæres for det detekterede symbol. Indekset, S, giver symbolkoden, der med ASCII-tabellen oversættes til det oprindelige symbol vha. `char`-funktionen. Algoritmen er implementeret i funktionen `FSKdemodulate`.

Basisskiftematricen $\mathbf{W}_{\mathbb{S}}$ benyttes i ovenstående funktion, men beregnes kun én gang for et givent sæt af indstillinger (baudrate, transmissionsbånd, N, f_s). Dertil er funktionen `change_of_basis_matrix` implementeret.

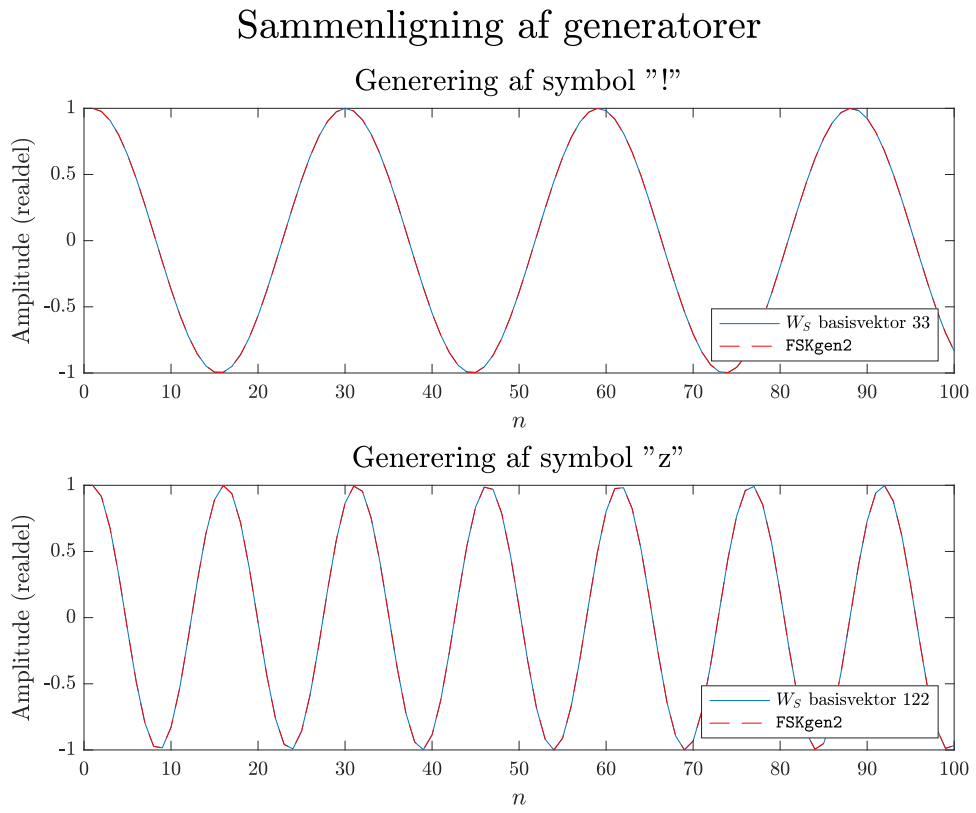
Til *data wrangling* er implementeret yderligere to algoritmer:

- `trim_ends`: Vha. indhyldningskurve fjernes ikke-signalbærende dele af en lydoptagelse, dvs. stille sektioner før/efter signaltransmission.
- `triggered_record`: Trigger-drevet start for optagelse, når lydniveau overstiger et baseline-niveau + trigger-level. Optagelsen stopper automatisk, når lydniveauet igen falder under denne tærskel.

3.3.3 Kodning med afkodningsmatricen

Afkodningsmatricen kan bruges til at enkodere beskeder, da hver søjle (basisvektor) indeholder N samples af den komplekse eksponentialfunktion med en given symbolfrekvens. Denne observation benyttes i opgave 4, hvor $\mathbf{W}_{\mathbb{S}}$ erstatter `FSKgen2` til at enkodere en arbitrær bitværdi. Eksemplet illustrerer overensstemmelsen:

```
comm_std.fs = 44.1e3;        % Hz, audio standard
comm_std.baudrate = 20;      % 20 symboler/sekund
comm_std.T_sym = 1/comm_std.baudrate;
comm_std.f1 = 1e3;           % Hz, nedre grænse i transmissionsbånd
comm_std.f2 = 5e3;           % Hz, øvre grænse i transmissionsbånd
comm_std.N_sym = 256;

W = change_of_basis_matrix(comm_std);   % Dan afkodnings/kodningsmatrix

c = '!z';                               % Tekst til enkodering
x1 = real(W(:,double(c)+1)');                                % gen. med W_S
x2 = [FSKgen2(c(1), comm_std); FSKgen2(c(2), comm_std)];   % gen. med FSKg

% plot sammenligning af signalgeneratorer
plot_gen_comparison(x1, x2, 'W_S', 'FSKgen2', c);
```



Figuren viser, at generatorerne outputter ens signaler - i de to ender af alfabetet :)

3.4 Test af detektionsalgoritme

3.4.1 Test med enkoderet signal og sammenligning med FFT

Første test er afkodning af et signal, der kommer direkte fra `FSKgen2`. Dvs. transmission over en audiokanal uden støj. Transmissionsindstillingerne defineret ovenfor benyttes igen. Bemærk at frekvensbåndet nu er 1-5 kHz.

```
msg_sent = 'Aarhus Universitet';
sig = FSKgen2(msg_sent, comm_std);

% Signalet afkodes:
sym_len = comm_std.T_sym*comm_std.fs;      % udregn symbollængde -> 2205
sym_sent = length(sig) / sym_len;          % udregn antal symboler -> 18
msg_rcvd = FSKdemodulate(sig, sym_len, sym_sent, W);    % afkod!
```

Den modtagne besked er som forventet:

```
disp(msg_rcvd);
```

```
Aarhus Universitet
```

Dette illustrerer brugen af algoritmerne til kodning og afkodning. Kodning og afkodning er i denne opsætning invertible transformationer.

Detektionsprocessen kan sammenlignes med en alm. DFT/FFT. Symbolet 'A' (dec. 65) findes ved 2019 Hz (bånd fra 1-5 kHz). Afstanden mellem symbolfrekvenserne kan udregnes til 16 Hz. Frekvensafstand i ovenst. detektionsmetode er per definition også 16 Hz. Frekvensopløsningen i en FFT på denne data er derimod på 20 Hz (44.1 kHz / 2205 samples/sym).

```
% == Projektion ==
X = sig(1:sym_len)*W;       % Udvælg første symbol og afkod med W
XP = X.*conj(X);            % Regn powerspektrum, alt. XP = X*diag(conj(X));
S = 0:comm_std.N_sym-1;     % Vektor for symbolkoder 0-255

% == FFT-metode ==
df = (comm_std.fs/sym_len); % frekv.opløsning 44100 Hz / 2205 = 20 Hz
f = (0:sym_len-1)*df;       % frekv.akse
Xfft = fft(sig(1:sym_len)); % FFT på første symbol
XPfft = Xfft.*conj(Xfft);   % Powerspektrum
```

De detekterede symboler kan uddrages fra powerspektra:

```
% == Projektionsmetode ==
[v, id] = max(XP);       % Højeste power
c = char(id-1);          % Konverter symbolkode til ASCII (Matlab offset)
disp(['Symbolkode: ', num2str(id-1) , ' -> ', 'Symbol: ', c]);
```

```
Symbolkode: 65 -> Symbol: A
```

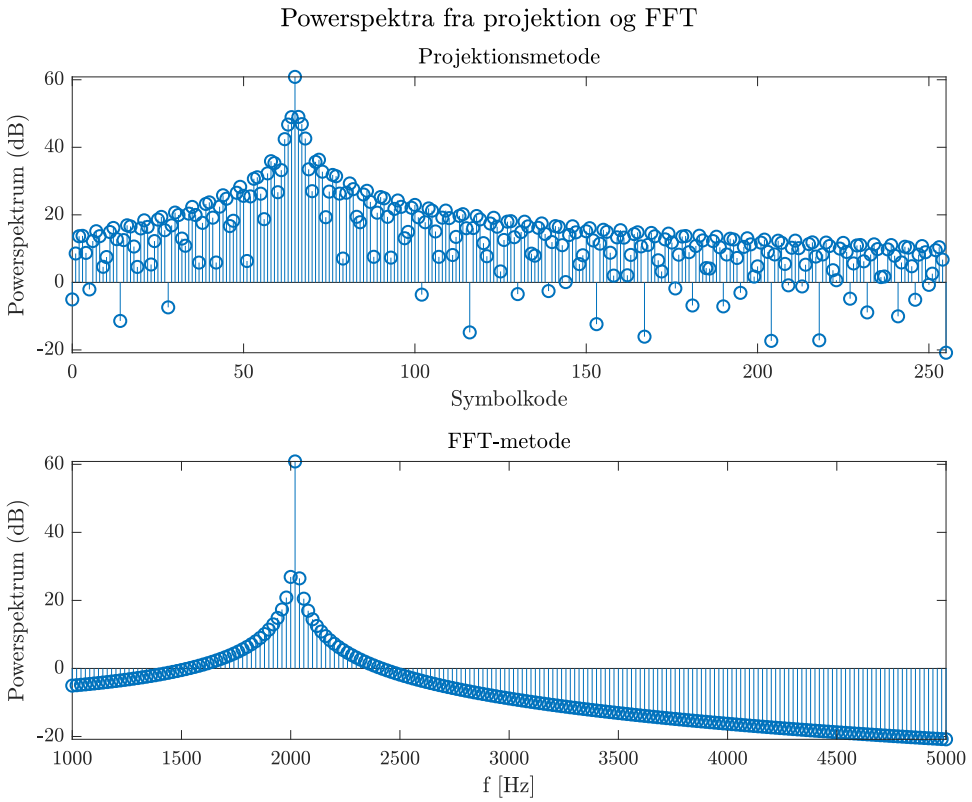
```
% == FFT-metode ==
[~, bid] = max(XPfft);                          % Højeste power bin id
nbors = [2003 2019 2035 2051];                  % Nabofrekv. til 'A'
nhood = '@ABC';                                 % Tilhørende symboler
[~, fid] = min(abs(nbors-(bid-1)*df) );         % Nærmeste værdi
disp(['Detekteret frekv.: ', num2str((bid-1)*df), ' Hz', newline, ...
      '=> Symbolfrekv.: ', num2str(nbors(fid)), ' Hz -> ', ...
      nhood(fid)]);
```

```
Detekteret frekv.: 2020 Hz
=> Symbolfrekv.: 2019 Hz -> A
```

Som forventet er symbolet 'A' både ved projektion og FFT. Sidstnævnte detektion krævede oprunding. For at detektere symbolet korrekt skal principielt bruges flere samples i en FFT end i den implementerede metode for at opnå den tilstrækkelige frekvensopløsning.

Spektra fra ovenstående proces vises i figuren nedenfor. Her ses, at en enkelt frekvens står frem med høj power. Desuden ses i projektionsspektret, at der er lækage sfa. ikke-ortogonale basisvektorer. Det er også tilfældet i FFT'en, da symbolfrekvensen ligger mellem bins (lækage og scalloping loss). Dog i mindre grad. Nedenstående figur viser også, at frekvensen med mest power (2020 Hz) ligger tæt på det forventede for 'A' (2019 Hz).

```
figure; setlatexstuff('latex');
sgtitle('Powerspektra fra projektion og FFT');

subplot(211); stem(S, 10*log10(XP));            % Ej skaleret!
xlabel('Symbolkode'); ylabel('Powerspektrum (dB)'); xlim([0 255]);
title('Projektionsmetode');

subplot(212); stem(f, 10*log10(XPfft));         % Ej skaleret!
xlim([comm_std.f1 comm_std.f2]);                % Kun transm.bånd
xlabel('f [Hz]'); ylabel('Powerspektrum (dB)');
title('FFT-metode');
```



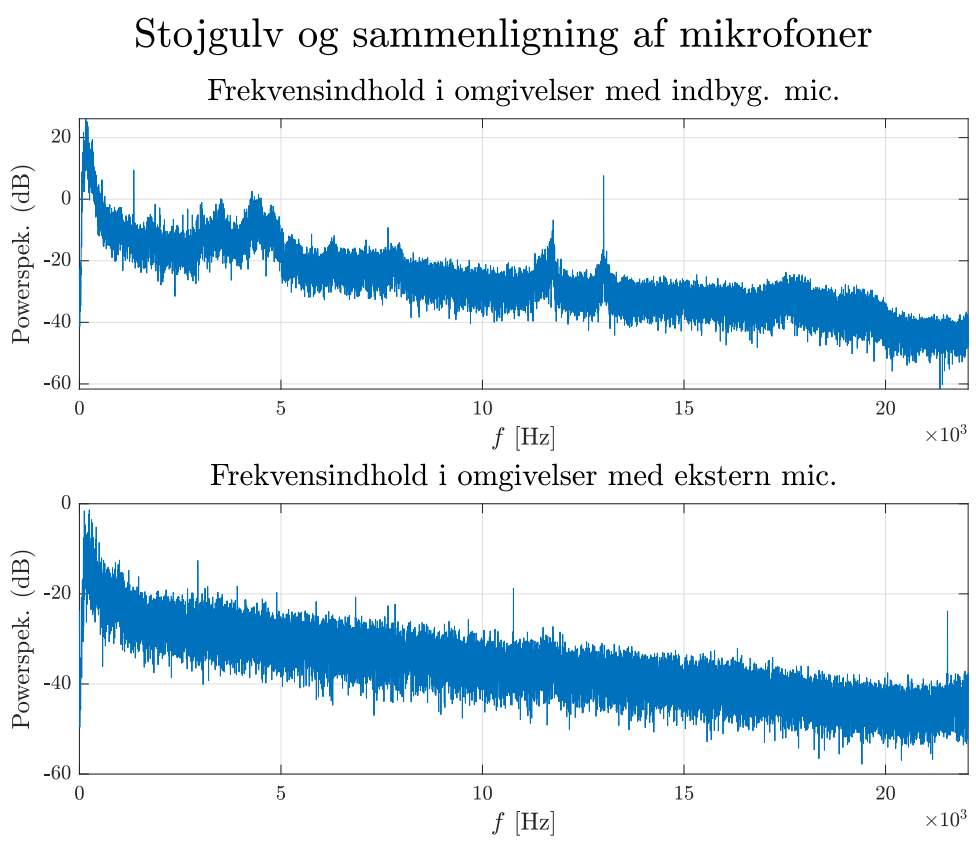
Hvorvidt denne tilgang er en god idé eller ej må stå sin prøve i en transmission over en kanal med støj.

3.4.2 Undersøgelse af støjforhold forud for transmissionstest

Jeg har mulighed for at benytte en ekstern eller en indbygget mikrofon, og ved arbejdsstationen, hvor forsøget med transmission udføres, er en del støjkilder: Computere, aircon, trafikstøj udefra, mv. Derfor måles på baseline for støj over en periode på 5 s med både den indbyggede mikrofon og en eksternt tilsluttet mikrofon.

```
load('ambient1.mat'); % Optaget med indbygget mikrofon, load til PDF-gen.
% ambient1 = measure_baseline_noise(5);
% save('ambient1.mat', 'ambient1');

```

```
load('ambientextmic.mat'); % Optaget med ekstern mikrofon, load til PDF-gen
% ambientextmic = measure_baseline_noise(5);
% save('ambientextmic.mat', 'ambientextmic');

% Plot sammenligning med smoothed powerspektrum
plot_mic_comparison(ambient1, ambientextmic, ...
                    {'indbyg. mic.','ekstern mic.'});
```



Der er væsentligt bedre forhold ved brug af den eksterne mikrofon. Med den undgås støj fra strømforsyning og ventilator/køling i computeren. Den benyttes til forsøget, og det er kun lave frekvenser under 0.5 kHz der evt. kunne give systematiske problemer. Denne lavfrekvente støj skyldes nok aircon, trafik, og lignende. Den kunne evt. filtreres bort. Evt. sammen med et anti-aliaseringsfilter foran A/D'en (dvs. lydkortet). Transmissionsbåndet lægges, så dette lavfrekvente område undgås.

3.4.3 Transmissionstest gennem luft

Afvikling af dette forsøg kan ses på video (https://youtu.be/UgcKJgXLqvw). Kommunikationsstandarden sættes op så:

- Baudrate: 20 baud. Har forsøgt mig frem, og op til denne rate overføres korrekt hver gang.
- Transmissionsbånd: 1 kHz til 5 kHz. Lidt arbitrært valgt, ud fra antagelsen at lydudstyret performer bedst i det frekvensområde, som dækker menneskelig tale, og at undgå støj.
- Samplingsfrekvens modulation/demodulation: 44.1 kHz. CD-kvalitet, dækker det hørbare spektrum.

Herefter skrives en lydfil, som afspilles fra et andet device (iPad).

```
comm_std.fs = 44.1e3;       % Hz, audio std.
comm_std.baudrate = 20;     % 20 symboler/sekund
comm_std.T_sym = 1/comm_std.baudrate;
comm_std.f1 = 1e3;          % Hz, nedre grænse i transmissionsbånd
comm_std.f2 = 5e3;          % Hz, øvre grænse i transmissionsbånd

Nstft = comm_std.T_sym*comm_std.fs;     % 2205 samples i hvert symbol
                                        % og derfor også i hver STFT.

payload = 'Digital Signalbehandling er sjovt.. Kek kek.';
signal2 = FSKgen2(payload, comm_std);
% audiowrite('signal.wav', signal2, comm_std.fs);
```

3.4.4 Transmissionstest - demodulering og detektion

Signalet transmitteres fra afsender via lydbølger i luften, og optages og efterbehandles, som vist nedenfor. Visse kommandoer er udkommenteret for at kunne generere en PDF. I stedet er gemte værdier fra eksperimentet loadet:

```
% Start optageren, og vent på signal
% signal_sample = triggered_record(0.001, comm_std.fs, 16);
% save('eks20baud.mat','signal_sample');
load('eks20baud.mat');

% Trim enderne væk
% Se signalet før trimning af ender
% t = (0:length(signal_sample)-1)/comm_std.fs;
% figure; plot(t, signal_sample);
sstrim = trim_ends(signal_sample, max(round(Nstft*0.001),201), 0.02);
% t = (0:length(sstrim)-1)/comm_std.fs;
% figure; plot(t, sstrim);

W = change_of_basis_matrix(comm_std);           % Afkodningsmatrix
symbols_sent = floor(length(sstrim)/Nstft);     % Antal sendte symboler

msg = FSKdemodulate(sstrim, Nstft, symbols_sent, W); % Afkod/demodulér
```

Resultatet for transmissionen:

```
disp(['Payload: ', msg, newline ...
      num2str(symbols_sent), ' symboler, ', ...
      num2str(symbols_sent*comm_std.T_sym), ' sek.'] );
```

```
Payload: Digital Signalbehandling er sjovt.. Kek kek.
44 symboler, 2.2 sek.
```

Beskeden er transmitteret korrekt med en baudrate på 20.

3.5 Læringer fra forsøget

- Eksperimentér med udstyret:
  - Den eksterne mikrofon fungerer væsentligt bedre end den interne, bl.a. fordi den fysisk er distanceret fra forskellige støjkilder.
  - Benyttede først en BOSE Soundlink II ekstern højttaler (Bluetooth), som tilsyneladende ikke gengiver frekvenser korrekt. Symboler blev shiftet, typisk op til +/-5 i ASCII.
  - iPad-højttaleren (og en iPhone-højttaler) var et hit. Klar og præcis frekvensgengivelse op til 20 baud.
- Transmissionsbånd kan med fordel vælges, så det er bredt og udnytter lydudstyrets "bedste" frekvensområder. Transmission blev også foretaget i området 5-15 kHz, men fungerede umiddelbart bedst i båndet 1-5 kHz.
- Støj i omgivelserne og resonans har betydning: Bedre transmission kan opnåes, når aircon slukkes, der afskærmes fra trafikstøj og når resonans reduceres (se fx i videoen, at iPad'en ligger på et håndklæde for at undgå resonans fra skrivebordet).
- Det er med givent udstyr / algoritme muligt at transmittere med op til 20 baud i en afstand 1-45 cm.
- Ved større afstande falder performance hurtigt (høj fejlrate).
- Som baudraten stiger, betyder hurtige frekvensskift, at der opstår højfrekvente "switching" transienter i højttaleren, der opleves som kortvarige "klik" eller "skrat". Hvis baudraten skal højere op, så bør disse overlejringer nok filtreres bort inden detektion.

4. Opgave 3: Signal-støj-forhold

Parsevals sætning benyttes til at regne SNR på signalet i frekvensdom. Til beregning af effekt bruges:

$$\sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2. \tag{4.1}$$

Beregning af SNR kompliceres af, at "signal" skal adskilles fra "støj". I frekvensdomænet kan et threshold (Lyons s. 877 ff.) bruges, som gjort i øvelser uge 6-7; men det har også sine begrænsninger. Hvis det er kendt, hvilke(t) frekvensbin(s), signalet ligger i, kan disse isoleres direkte. Under alle omstændigheder:

$$\text{SNR} = \frac{\text{Signal power}}{\text{Noise power}} = \frac{\text{Sum af } |X(k)|^2 \text{ for signal}}{\text{Sum af } |X(k)|^2 \text{ for støj}} \tag{4.2}$$

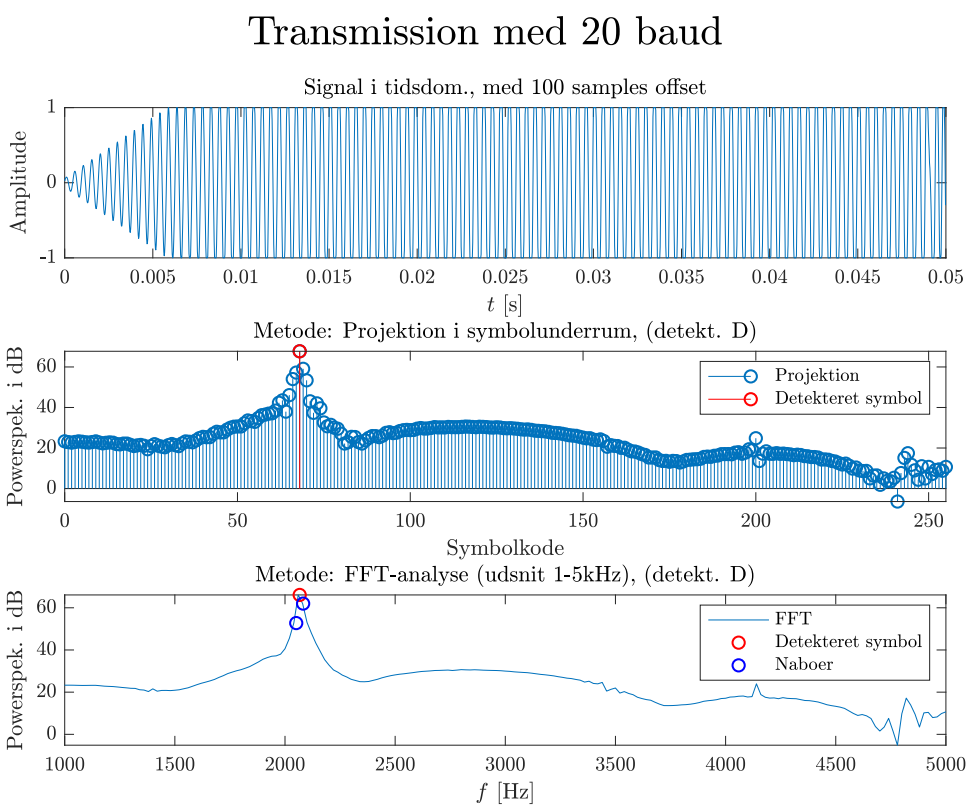
$$\text{SNR}_{dB} = 10 \log_{10}(\text{SNR}) \tag{4.3}$$

I denne case "defineres", at SNR kun regnes for transmissionbåndet: Der er kun informationsindhold i båndet ml. f_1 og f_2, mens støjen ligger fordelt ud over alle frekvenser. Støj, der ligger uden for båndet, kan altså principielt ignoreres (i denne case).

4.1 SNR-beregning på eksperimentdata

I et tidligere afsnit blev data overført gennem luften med 20 baud. Første symbol i beskeden er 'D', med ASCII-kode 68, i det benyttede bånd indkodet som 2067 Hz. Tidsdomæne, FFT-powerspektrum og afkodning vha. projektion i symbolunderrum for det første symbol (0'te symbol) ser ud som følger:

```
% 0'te symbol i sstrim afkodes med W, og der benyttes manuelt offset på 100
% samples. True for at plotte output.
[projm, fftm] = detect_compare_snr(sstrim, 0, W, comm_std, true, 100);
```



Figuren viser, at begge metoder giver næsten ens udseende spektra. Begge metoder identificerer samme symbolkode (2067 Hz → kode 68 → ASCII 'D'). For projektionsmetoden er forhold mellem effekt for detekteret signal og det nærmeste nabosymbol på 8.7 dB. I spektrum fra FFT er der kun 4.1 dB. Dvs. umiddelbart højere støjimmunitet med projektionsmetoden.

Effektberegning foretages ved udvælgelse af enkelt bin med signalindhold. Dette gælder for begge metoder, da nabobins repræsenterer andre symboler, og ikke (på trods af lækage og scalloping) kan betragtes som "signal". Signal-støj-ratio er estimeret:

```
disp(['Estimation af SNR (dB):', newline, ...
      'Projektionsmetode: ', num2str(projm.SNR_dB), ' dB', newline, ...
```

```
3       'FFT-metode: ', num2str(fftm.SNR_dB), ' dB']);
```

```
Estimation af SNR (dB):
Projektionsmetode: 4.5946 dB
FFT-metode: 2.6492 dB
```

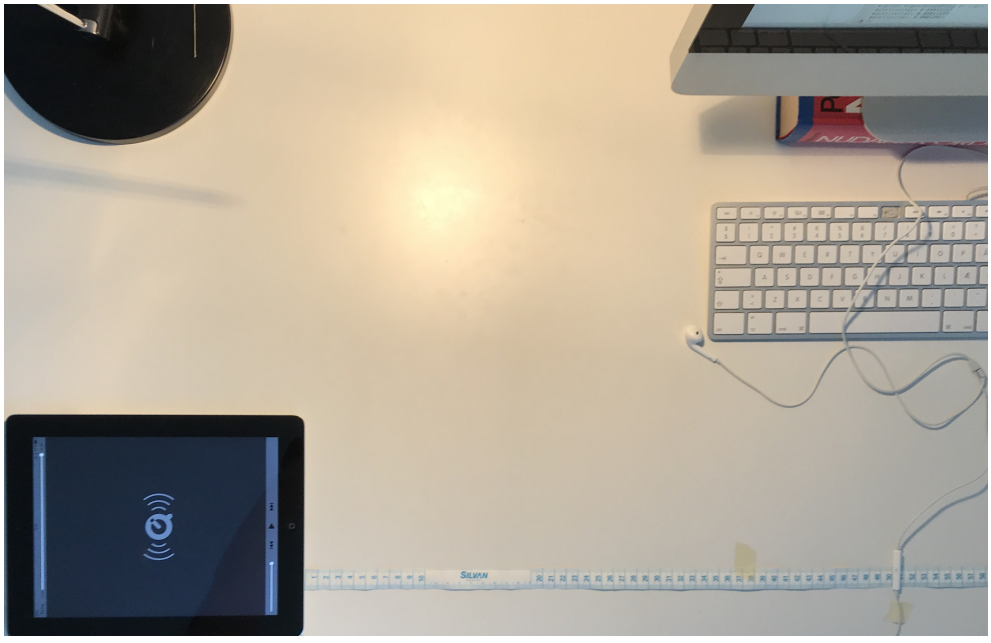
Der er selvfølgelig kun én korrekt SNR. I dette tilfælde ligger det rigtige resultat formodentlig tættest på projektionsmetoden. For FFT-metoden er signaleffekten nemlig bredt ud over flere bins (lækage / scalloping loss), men der måles kun på ét bin, som begrundet tidligere. Det forklarer formodentlig også forskellen i 8.7 dB vs. 4.1 dB til nabosignaler.

Det er oplagt at overveje, om en vinduesfunktion ville dæmpe nabofrekvenserne, og give højere støjimmunitet. Dette er forsøgt (Hann, Blackman, Hamming). I dette eksempel er frekvensopløsningen for FFT'en presset til grænsen ift. afstanden mellem symbolfrekvenserne, og hvert bin repræsenterer et symbol. Der er ingen "ubrugte" bins. Dette gælder også for projektionsmetoden. Så da "main lobe" i alle tilfælde bliver bredere end med rektangulært vindue, er trade-off at dæmpning af nabosymboler blivere værre, mens dæmpning af fjernere symboler bliver bedre.

Vinduer forbedrer i dette tilfælde ikke diskrimination af signal og støj. MEN, algoritmen *kunne* gentænkes, med hhv. "ubrugte" bins indlagt mellem signalbins el. flere samples for bedre Δf. Det er et trade-off i beregningstid og baudrate vs. præcision / støjimmunitet.

4.2 SNR vs. afstand til mikrofon

Formålet med denne test er at afgøre maksimal afstand mellem afsender og modtager, hvormed der kan transmitteres fejlfrit, og at forstå udviklingen i SNR. Der benyttes samme transmissionsindstillinger og signal som før. Opstillingen til testen er vist i figuren herunder. Afstanden varieres ved flytning af mikrofonen langs målebåndet.



Figur 4.1: Opstilling til bestemmelse af maksimal afstand

Følgende kode er gentaget for afstandene 1, 5 og 10 cm derefter i intervaller af 10 cm op til 50 cm. Ved 50 cm opstod en fejl i transmissionen, og en sidste OK transmission blev gennemført over 45 cm. Koden viser også, hvordan transmissionsresultat kunne monitoreres undervejs.

```
% Start optageren, og vent på signal
% sig50cm = triggered_record(0.001, comm_std.fs, 16);
% Gem optagelsen i en struct:
% sigd.d45cm = sig45cm;
% Gem struct til senere brug
% save('sigd.mat','sigd');
load('sigd.mat'); % load data til gen. af PDF

sigprocess = sigd.d45cm;

% Trim enderne væk
sigtrim = trim_ends(sigprocess, max(round(Nstft*0.001),201), 0.02);

% Afkod
symbols_sent = floor(length(sigtrim)/Nstft);     % Antal sendte symboler
msg = FSKdemodulate(sigtrim, Nstft, symbols_sent, W); % Afkod/demodulér
```

```
disp(msg);
```

```
Digital Signalbehandling er sjovt.. Kek kek..
```

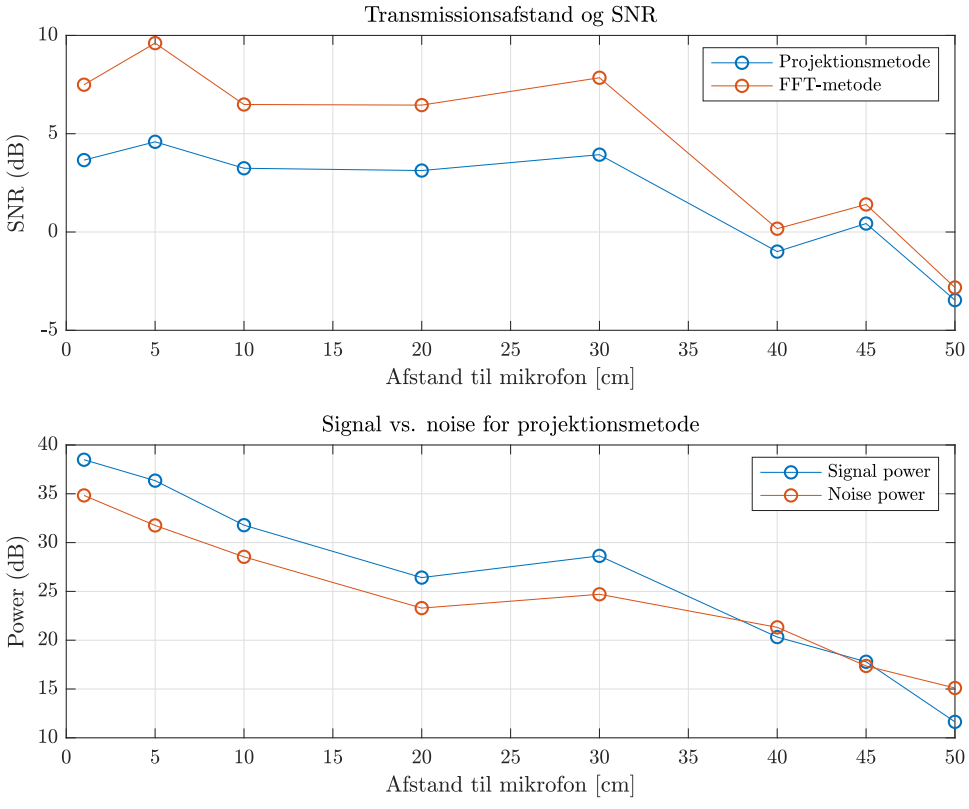
Ved 50 cm opstår første fejl i transmissionen. Beskeden *'Digital Signalbehandling er sjovt.. Kek kek.'* bliver til *'Digital Signalbehandling ee tjovt.. Kek kek.'* Dvs. i symbol nr. 27 m.fl. opstår fejl, hvor 'r' bliver afkodet som 'e'. Fejlen skyldes bl.a. at der opbygges et offset i STFT vs. symboler, altså algo'en rammer "skævt" på symbolsamples, fordi der ikke synkroniseres mellem afsender og modtager. Beskeden kan nemt fikses og aflæses korrekt; et manuelt offset på 100 samples er nok. Men det er jo netop pointen, at algoritmen "skal klare det selv". Nedenfor beregnes udvikling i SNR (dB) og analyse af hhv. signal- og støjeffekt (dB), med fokus på tegn nr. 27.

```
analyse_symbol = 27; % fejl opstår i symbol nr. 27 ('r' -> 'e')

distance = {'01', '05', '10', '20', '30', '40', '45', '50'};   % afst. cm
d_vec = str2num(cell2mat(distance'))';              % målte afstande i cm
SNR_dB_proj = []; SNR_dB_fft = [];      % vektorer til at gemme midl. res.
sig_pwr = []; noise_pwr = [];

for d = 1:length(distance)
    sigprocess = sigd.(['d', distance{d},'cm']);   % Udvælg datasæt, beskær
    sigtrim = trim_ends(sigprocess, max(round(Nstft*0.001),201), 0.02);

    % Demoduler og analyser signal
    [projm, fftm] = detect_compare_snr(sigtrim, analyse_symbol,...
```

```
                                                    W, comm_std, false);

    SNR_dB_proj = [SNR_dB_proj projm.SNR_dB];  % Gem SNR (dB)
    SNR_dB_fft = [SNR_dB_fft fftm.SNR_dB];

    sig_pwr = [sig_pwr projm.sig_pwr];       % Gem power for signal og støj
    noise_pwr = [noise_pwr projm.noise_pwr]; % kun for projektionsmetode
end
```

Udvikling i SNR (dB) og opdeling i signal vs. noise vises i figuren nedenfor, baseret på beregninger foretaget i foregående kode.

```
% Sammensæt til plots
SNR_dB = [SNR_dB_proj;SNR_dB_fft];
sigvnoise = [10*log10(sig_pwr); 10*log10(noise_pwr)];

figure;
subplot(211); plot(d_vec, SNR_dB, 'o-');
legend({'Projektionsmetode', 'FFT-metode'}); grid on;
xlabel('Afstand til mikrofon [cm]'); ylabel('SNR (dB)');
title('Transmissionsafstand og SNR');
subplot(212); plot(d_vec, sigvnoise, 'o-');
legend({'Signal power', 'Noise power'}); grid on;
xlabel('Afstand til mikrofon [cm]'); ylabel('Power (dB)');
title('Signal vs. noise for projektionsmetode');
```



Pga. støjindhold er SNR også en stokastisk variabel. Figuren viser, at SNR mindskes som afstanden mellem afsender og modtager øges. Både projektionsmetoden og FFT-metoden viser samme tendens. Når SNR bliver for lav, kan signalet ikke længere detekteres korrekt.

Andet panel i figuren viser, at signal- og støjeffekt begge falder som afstanden øges, og at signalkomponenten mindskes hurtigere end støjen. Ved 50 cm, da fejlen opstår, er effekt fra støj væsentligt større end den fra signalet (omkring 4 dB over).

Mulige forklaringer på at støjniveauet ikke er konstant, men derimod højt korreleret med signalniveauet:

- Lydtryk (amplitude) i en lydbølge "falder af" med $\frac{1}{r}$, intensitet (effekt) med $\frac{1}{r^2}$.
- Højttaleren kan have frembragt uønskede støjfrekvenser, med lavere effekt end signalfrekvenserne, men som begge aftager med samme faktor.
- Støj opstår, idet lydbølgen reflekteres fra bordpladen, udstyret selv og lignende. Jo tættere på kilden, jo mere effekt har denne støj.
- Lydintensiteten kan være så høj, at den momentant presser mikrofonen ind i et ikke-lineæert område, med forvrængning til følge.
- Der er et baseline-niveau for støj i rummet ("støjgulv"), som adderes uanset afstand til kilden, og bliver dominerende jo længere væk fra kilden, mikrofonen placeres.

Der er uden tvivl mange øvrige gode forklaringer :)

5. Opgave 4: Baudrate og bitrate

5.1 Baudrate

I ovenstående tests er det allerede forsøgt at maksimere baudraten ift. hvad er muligt at transmittere med det givne udstyr og algoritme. Som algoritmen er sat op nu, er grænsen **omkring 20 baud**, som vist på https://youtu.be/UgcKJgXLqvw. For en given samplingsfrekvens betyder følgende relation, at antallet af samples for hver sektion af en STFT vil falde, som baudraten stiger.

$$N_{\text{sektion}} = \frac{f_s}{\text{baudrate}}$$

Med FFT-metoden betyder dette, at frekvensopløsningen vil forværres, idet

$$\Delta f = \frac{f_s}{N_{\text{sektion}}} = \text{baudrate}$$

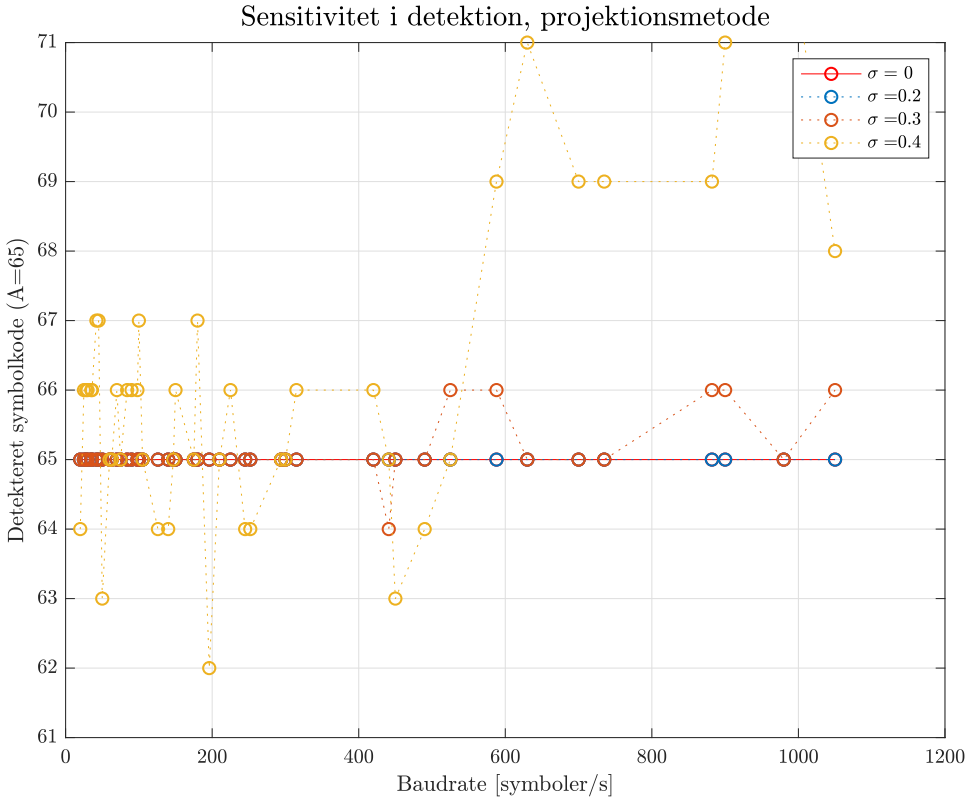
Med en baudrate på 90 symboler/sek., vil frekvensopløsningen i en FFT blive 90 Hz. Med vores ASCII-enkodering vil det kræve et transmissionsbånd fra ca. 1-25 kHz at opnå en frekvensadskillelse på 90 Hz. Det er upraktisk!

Med projektionsmetoden er det krævede antal samples derimod det antal, der skal til, for at beregne en "god" korrelation (indre produkt). Hvad det betyder i praksis, afhænger af støjens varians. Nedenstående "mikro Monte-Carlo simulation" illustrerer denne påstand:

```
% En række praktiske baudrater (heltalsdivisorer af 44.1 kHz)
baud = [20, 25, 28, 30, 35, 36, 42, 45, 50, 60, 63, 70, 75, 84, 90, ...
        98, 100, 105, 126, 140, 147, 150, 175, 180, 196, 210, 225, ...
        245, 252, 294, 300, 315, 420, 441, 450, 490, 525, 588, 630, ...
        700, 735, 882, 900, 980, 1050];

stdev = [0 0.2 0.3 0.4];    % Standardafvigelse for støj
c = 'A';                    % Testbesked
res_noise = [];

% Kør alle baudrater igennem, og tjek hvordan algo detekterer 'A'
for b = 1:length(baud)
    comm_std.baudrate = baud(b);            % Sæt ny baudrate
    comm_std.T_sym = 1/comm_std.baudrate;   % Sæt symboltid
    W = change_of_basis_matrix(comm_std);   % Dan ny afkodningsmatrix
    testsig = FSKgen2(c, comm_std);         % Enkoder testbesked
    sym_len = comm_std.T_sym*comm_std.fs;   % Udregn symbollængde

    % Tillæg støj i forskellige niveauer, og afkod
    for s = 1:length(stdev)
        testsig_noise = testsig + stdev(s)*randn(1,length(testsig));
        res_noise = [res_noise FSKdemodulate(testsig_noise, sym_len,1,W)];
    end
end

res_noise = reshape(res_noise, length(baud), [])'; % Rækkevektor til matrix
```

Plot baudrate vs detekteret symbol (husk, 'A' er 65):

```
figure
plot(baud, double(res_noise(1,:)), 'ro-'); hold on;
plot(baud, double(res_noise(2:length(stdev),:)), 'o:'); hold off;
set(gca,'ytick',0:comm_std.N_sym-1);
ylim([double(c)-4 double(c)+6]); grid on;
ylabel('Detekteret symbolkode (A=65)'); xlabel('Baudrate [symboler/s]');
leg = strcat('$\sigma=$', cellstr(num2str(stdev')));
legend(leg, 'Interpreter', 'Latex');
title('Sensitivitet i detektion, projektionsmetode', 'FontSize', 14);
```

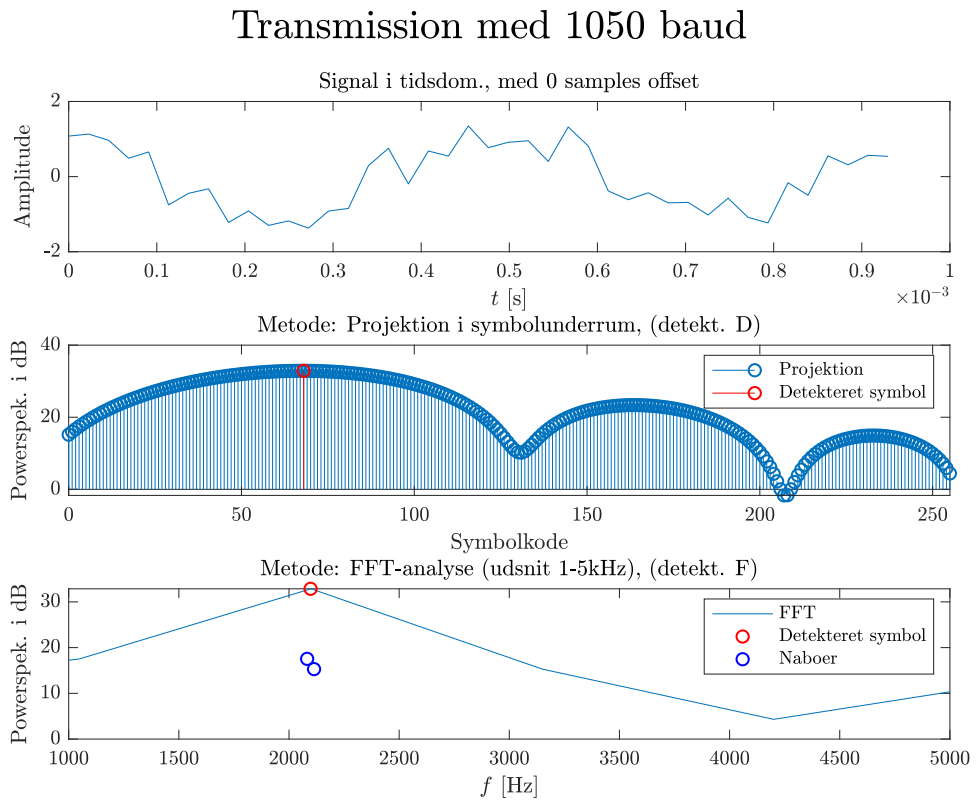


Figuren viser, at med en standardafvigelse på $\sigma \leq 0.2$, så performer algoritmen pænt hele vejen op til den højeste testede baudrate på 1050. Med $\sigma \leq 0.3$ er algoritmen stabil op til ca. 400 baud. Med $\sigma \leq 0.4$ er algoritmen ustabil for alle testede baudrater.

Standardafvigelsen kan sættes i relation til amplituden på sinussignalet, derved at 95% af samples trukket fra normalfordelingen vil falde inden for ca. $\pm 2\sigma$. For $\sigma = 0.4$ er sinusen altså overlejret med et støjsignal, der med 95% konfidens lægger op til ± 0.8 til amplituden. Potentielt en fundamental ændring af signalet, der har $A_{pk} = 1$.

Den sidst udførte spektralanalyse vises nedenfor. Det er tydeligt, at kombinationen af høj baudrate og støj ($\sigma = 0.4$) er er vanskelig for algoritmerne. Hhv. en meget bred main lobe og alt for lav frekvensopløsning, koblet med væsentlig overlejret støj, betyder at begge algo'er fejldetekterer det sendte 'A'.

```
1 projm = detect_compare_snr(testsig_noise, 0, W, comm_std, true);
```



SNR for denne

```
disp(['SNR (dB): ', num2str(projm.SNR_dB)]);
```

```
SNR (dB): -18.2151
```

5.2 Bitrate og parallelkommunikation

Dette sidste afsnit er teoretiske overvejelser og en demo. I opgaveoplægget foreslås en transmissionsteknik, hvor hver frekvens repræsenterer en bit (1 hvis frekvensen er til stede i sig., ellers 0).

Man implementerer altså en slags "parallelkommunikation" el. en "bus" ved fx at sende 256 bits samtidig over et transmissionsbånd. Fortolkningen kunne være et 256-bit binærtal mellem 0 og $2^{256} - 1$, hvilket er et ubrugeligt stort tal, så mere sandsynlig brug ville være at enkodere data som 8 x 32-bit, eller 16 x 16-bit.

For at denne teknik kan lykkes, skal antallet af samples være så tilpas højt, at frekvensopløsningen kan adskille bitfrekvenserne *og* der skal være hurtigt nok roll-off og attenuering/dæmpning af sidelobes til at nabobits ikke fejlregistreres.

Der er altså - igen - et trade-off mellem hurtig baudrate ($\rightarrow$ lavt antal samples per sektion) og antal mulige frekvenser i båndet, der pålideligt kan afkodes samtidig. Når SNR falder, skal der også (generelt set) benyttes flere samples til at afkode signalet sikkert.

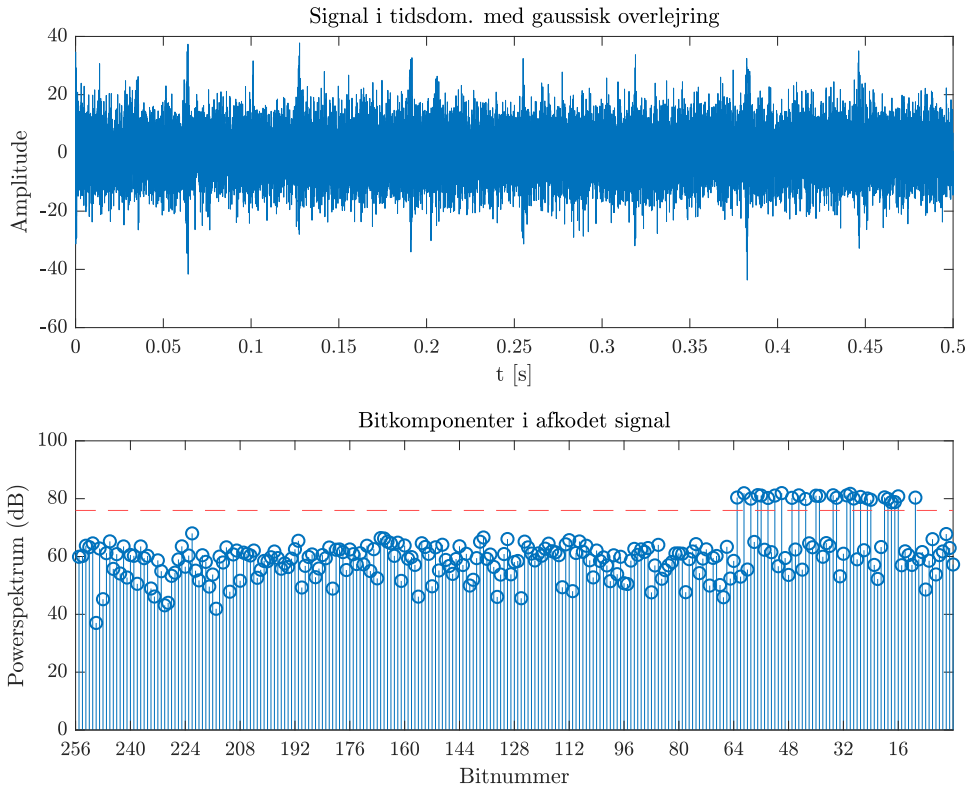
Bitraten er givet ved

$$\text{bitrate} = (\text{bits i bånd}) \cdot (\text{baudrate})$$

Hvor det ses, at en højere bitrate kan opnås ved at øge antal bits i båndet og overføre med en lavere baudrate, eller vice versa. Det optimale punkt er (nok) en funktion af frekvensrespons og støj i transimissionskanalen. Nedenfor laves en lille demo på metoden. Der sendes 64 bits samtidig over en kanal med en bredde på 256 bits. Baudrate er 2 baud, altså potentielt 512 bits/sekund:

```
bit_ch = 256;                                   % 256 bits i kanalen
send_num = 12345678901234567890;                % decimaltal til transm.

binary_code = dec2bin(send_num);                % dec. tal bliver 64-bit binær
bitmask = logical(zeros([1 bit_ch]));           % 256-bit bitmaske med 0'er

% Tænd relevante bits i mask (MSB'er i lave frekv.område, LSB'er i høje):
bitmask(end-length(binary_code)+1:end) = str2num(binary_code')';

% Opsæt kommunikation
comm_std.fs = 44.1e3;       % Hz, audio standard
comm_std.baudrate = 2;      % 2 symboler/sekund
comm_std.T_sym = 1/comm_std.baudrate;
comm_std.f1 = 1e3;          % Hz, nedre grænse i transmissionsbånd
comm_std.f2 = 5e3;          % Hz, øvre grænse i transmissionsbånd
comm_std.N_sym = bit_ch;

Nstft = comm_std.fs / comm_std.baudrate;

% Enkodering med afkodningsmatricen Husk: hver søjle i denne matrix er en
% basisvektor fra C^N. Vi skal altså udvælge de relevante søjler og addere.
W = change_of_basis_matrix(comm_std);   % Dan afkodnings/kodningsmatrix

% De aktive søjler (per bitmask) beholder værdi, resten får 0.
% Transponeringer er for at få rækkevektor med det samlet signal
x = bitmask*W';

% Transmissionssignal
% -> vi vælger kun at sende realdelen over kanalen
% -> adderer en væsentlig mængde støj
xtr = real(x) + 2*std(real(x))*randn(1,length(x));
t_vec = (0:Nstft-1)./comm_std.fs;

% Afkod transmitteret signal
X = xtr*W;

% Beslut threshold -> vælger at threshold skal være 6 dB ned fra maks
tr_band = -6;                       % 6 dB ned fra max
tr = max(abs(X))*10^(tr_band/20);   % thresholdværdi

% Se afsendt af afkodet signal
```

```
figure; subplot(211); plot(t_vec, xtr);
title('Signal i tidsdom. med gaussisk overlejring');
xlabel('t [s]'); ylabel('Amplitude');
subplot(212); stem(mag2db(abs(X)));
yline(mag2db(tr), 'r--');
title('Bitkomponenter i afkodet signal');
xlabel('Bitnummer'); ylabel('Powerspektrum (dB)');
xticks(0:16:bit_ch-1); xticklabels(flip(0:16:bit_ch));
xlim([0 bit_ch]);
```



Resultatet for den modtagne besked:

```
recv_bitmask = X > tr;                          % Kun med hvis < 6 dB ned
bitsum_vec = flip(2.^(0:bit_ch-1));             % MSB først
recv_num = recv_bitmask * bitsum_vec';          % Konvertér til decimaltal
disp(['Forskel modtaget og sendt tal: ', num2str(recv_num - send_num)]);
```

```
Forskel modtaget og sendt tal: 0
```

Hvilket illustrerer, at teknikken virker, selv med væsentlig støj. Hvis baudraten øges, falder støjimmuniteten. Dette trade-off er diskuteret ovenfor. Der kunne kompenseres herfor ved at sænke antallet af bits i kanalen. Fx kunne kanalen reduceres til 2 frekvenser, repræsenterende værdien for et enkelt bit (hhv. høj og lav tilstand). De 256 overførte bits kunne fx enkodere:

- 32 8-bit ASCII-symboler (eller et variende antal UTF-8 tegn).
- 16 16-bit samples til et digitalt lydsignal.

- En strøm af 8-bit pixels til en katte-GIF.

Hvilket illustrerer, at dette er en fleksibel kommunikationsprotokol, hvor (næsten) kun fantasien sætter grænser.

6. Forbedringsmuligheder

- Støjfiltrering inden demodulering sammen med anti-aliasering.
- Kan der findes en teknik til at bruge et differentielt signal (fx over kobber), så common-mode-støj kan fjernes?
- Automatisk detektion af baudrate og symbolgrænser. Det kunne også muliggøre oversampling på modtagesiden.
- Kommunikation på flere bånd ad gangen ("sub-bands").
- Dele frekvensbåndet, så en portion er dedikeret til upstream komm. og en anden til downstream.
- Forsøg med at overføre signalet gennem kobber, lysleder, el. lign.
- Implementere PSK, QAM, QPSK, el. lign.

Øvrigt: MATLAB-funktionerne `double()` og `char()`, som benyttes til at konvertere til/fra ASCII-symbolkoder, er kun portable for ASCII-tabellens tegn 0 til 127 (decimal). Tegn 128 til 255 er fra den udviddede ASCII-tabel, som afhænger af tegnsættet på en given computer. Charset i MATLAB kan ses med `slCharacterEncoding()`. På min computer har jeg sat charset i MATLAB til UTF-8, så hvis koden eksekveres på en anden computer, er resultaterne muligvis anderledes.

7. Konklusion

I denne case er vist, hvorledes frekvensdomænet kan benyttes til at kode, analysere og afkode datatransmissioner. Der er implementeret to teknikker baseret på AFSK, hhv. hvor symboler sendes serielt ét ad gangen (opg. 1-3) og hvor adskillige bits sendes parallelt (opg. 4). Signal-støj-forhold er diskuteret, og det er illustreret til hvilken grad afkodnings-algoritmerne er robuste over for forskellige støjniveauer målt ved både SNR (dB) og standardafvigelse for gaussisk støj.

Det var en interessant case, og den har åbnet blikket for perspektiver i datakommunikation.

8. Hjælpefunktioner

Der er til projektet implementeret en række hjælpefunktioner.

8.1 FSKgen2

Fra kurset E4DSA. Modificeret let, Janus, feb. 2020.

```
function x = FSKgen2(payload, cstd)
% Inputs:
%   payload:    besked-streng, der skal enkoderes, fx 'abcde'
%   cstd:       transmissionsstandard, indeholdende:
%       f1:         nedre frekvens i transmissionsbånd
%       f2:         øvre  frekvens i transmissionsbånd
%       T_sym:      varighed af hvert symbol i sekunder (1/baudrate)
%       N_sym:      antal af mulige symboler
%       fs:         samplingsfrekvens

    % 256 jævnt fordelte frekvenser mellem f1 og f2 der svarer
    % til den udviddede ASCII tabel: http://www.asciitable.com/
    freqarray = linspace(cstd.f1, cstd.f2, cstd.N_sym);

    % Konverterer input ASCII-karakterer til heltal,
    % ex. 'abc!'->[97 98 99 33]
    ids = double(payload);

    A    = 1;                                % signalamplitude
    nsym = 0 : cstd.T_sym * cstd.fs - 1;     % sampletidstæller per symbol
    N    = length(ids);                      % antal symboler i transmission

    x = []; % definerer tomt array
    for k = 1:N                              % for alle symbolerne i payload
        f_sym = freqarray(ids(k) + 1);       % omsæt kode til signalfrekvens
        xs = A*cos(2*pi*f_sym/cstd.fs*nsym);% beregn outputsignal
        x = [x xs];                          % sæt sektion in i samlet sign.
    end

end % end of function
```

8.2 setlatexstuff

```
function [] = setlatexstuff(intpr)
% Sæt indstillinger til LaTeX layout på figurer: 'Latex' eller 'none'
```

```
% Janus Bo Andersen, 2019
    set(groot, 'defaultAxesTickLabelInterpreter',intpr);
    set(groot, 'defaultLegendInterpreter',intpr);
    set(groot, 'defaultTextInterpreter',intpr);
end
```

8.3 show_timefreq

```
function [] = show_timefreq(x, cstd)
% Viser tidsserie og frekvensindhold for transmissionssignal
% Janus Bo Andersen, Feb 2020
    setlatexstuff('Latex'); figure      % Figurindstillinger
    N = length(x);
    n = 0:N-1;
    t = n / cstd.fs;                    % tidsvektor

    X = fft(x);                         % frekvensindhold

    % Frekvbins, vi vil se - beregn kun i transmissionsbånd
    kmin = round(cstd.f1/cstd.fs * N);
    kmax = round(cstd.f2/cstd.fs * N);
    k = kmin:kmax;                      % interessante bins
    f = cstd.fs * k / N;                % frekvensakse

    subplot(2,1,1)
    plot(t,x);
    ylabel('Signalamplitude', 'Interpreter','Latex', 'FontSize', 15);
    xlabel('$t$ [s]', 'Interpreter','Latex', 'FontSize', 15);
    grid on;

    subplot(2,1,2)
    plot(f, 10*log10(X(k).*conj(X(k))) );          % power spektrum dB
    xlabel('$f$ [Hz]', 'Interpreter','Latex', 'FontSize', 15);
    ylabel('Powerspektrum (dB)', 'Interpreter','Latex', 'FontSize', 15);
    xlim([cstd.f1, cstd.f2]);

    sgtitle('Tids- og frekvensplot for transmissionssignal', ...
            'Interpreter', 'Latex', 'FontSize', 20);
end
```

8.4 iterated_spectrogram0

```
function [] = iterated_spectrogram0(x,Ls,Nfft,steps,fs,ylims)

```

```
    for i=1:4
        subplot(2,2,i)
        spectrogram0(x,Ls(i),Nfft,steps(i),fs,ylims);
    end
    sgtitle('Spektrogram', ...
            'Interpreter', 'Latex', 'FontSize', 20);
end
```

8.5 spectrogram0

Implementeret af Kristian Lomholdt, E4DSA. Let modificeret, Janus, feb. 2020. Baseret på Manolakis m.fl., s. 416.

```
function S=spectrogram0(x,L,Nfft,step,fs,ylims)
% Spektrogram. Beregner og viser spektrogram
% Baseret på: Manolakis & Ingle, Applied Digital Signal Processing,
%             Cambridge University Press 2011, Figure 7.34 p. 416
% Parametre:  x:    inputsignal
%             L:    vinduesbredde ("segmentlængde")
%             Nfft: DFT størrelse. Der zeropaddes hvis Nfft>L
%             step: stepstørrelse
%             fs:   samplingsfrekvens
% Forlkaring:
% x  |-------------------------------------------------------------------|
%    |----------|                                                      N-1
%               L-1
%          |----------|
%        step
%
% KPL 2019-01-30

% transpose if row vector
    if isrow(x); x = x'; end

    N = length(x);
    K = fix((N-L+step)/step);
    w = hanning(L);
    time = (1:L)';
    Ts = 1/fs;
    N2 = Nfft/2+1;
    S = zeros(K,N2);
    for k=1:K
        xw = x(time).*w;
        X  = fft(xw,Nfft);
        X1 = X(1:N2)';
        S(k,1:N2) = X1.*conj(X1); % samme som |X1|^2 - effektspektrum
        time = time+step;
```

```
    end
    S = fliplr(S)';
    S = S/max(max(S)); % normalisering



    tk = (0:K-1)'*step*Ts;
    F = (0:Nfft/2)'*fs/Nfft;

    colormap(jet);      % farveskema, prøv også jet, summer, gray, ...
    imagesc(tk,flipud(F),20*log10(S),[-100 10]);

    axis xy
    ylabel('$f$ [Hz]', 'Interpreter','Latex', 'FontSize', 12);
    ylim(ylims);

    xlabel('$t$ [s]', 'Interpreter','Latex', 'FontSize', 12);

    title(['$N_{FFT}=$' num2str(Nfft) ', $L=$' num2str(L)...
           ', step=' num2str(step)], ...
            'Interpreter', 'Latex', 'FontSize', 14)
end
```

8.6 change_of_basis_matrix

```
function W = change_of_basis_matrix(cstd)
% Janus, feb. 2020
% Returnerer en basisskiftematrix (quasi-DFT) for de givne indstillinger
% Basisskiftematricren er en afkodningsmatrix, som beskrevet i teoriafsn.

    % Lav en anonym funktion til at give symbolfrekvenser
    fsym_vec = linspace(cstd.f1, cstd.f2, cstd.N_sym);
    fsym = @(S) fsym_vec(S+1);                          % funktionshandle

    % Lav en anonym funktion til at give basisvektorer
    % Bemærk, at vektoren med n=0..(N-1) genereres ved at N=fs*T_sym
    w = @(S) exp(-1j*2*pi*fsym(S)/cstd.fs*(0:cstd.T_sym*cstd.fs-1));

    W = [];
    for s = 0:cstd.N_sym - 1  % 0 til 255 for 256 forskellige ASCII-værdier
        W = [W w(s)'];        % Tilføj basisvektor som en søjle
    end

end
```

8.7 plot_gen_comparison

```
function [] = plot_gen_comparison(x1, x2, name1, name2, symbols)
% Janus, feb. 2020
% Plotter sammenligning af de to generatormetoder
% x1 og x2 skal indeholde så mange rækker, som der er symboler i symbols

    setlatexstuff('Latex'); figure
    totplots = length(symbols);

    for plotnum = 1:totplots
        subplot(totplots, 1, plotnum)

        plot(x1(plotnum,:)); hold on
        plot(x2(plotnum,:), 'r--'); hold off
        xlim([0 100]);

        ylabel('Amplitude (realdel)', ...
               'Interpreter','Latex', 'FontSize', 12);
        xlabel('$n$', 'Interpreter','Latex', 'FontSize', 12);
        title(['Generering af symbol "', symbols(plotnum), '"'], ...
              'Interpreter', 'Latex', 'FontSize', 15);
        legend(['$', name1, '$', ' basisvektor ', ...
                num2str(double(symbols(plotnum)))], ...
                ['\texttt{', name2, '}'], ...
                'Location', 'SouthEast');
    end

    sgtitle('Sammenligning af generatorer', ...
              'Interpreter', 'Latex', 'FontSize', 20);
end
```

8.8 detect_compare_snr

```
function [projm, fftm] = detect_compare_snr(sstrim, symi, W, ...
                                            comm_std, doplot, ...
                                            offset)
% Janus, feb. 2020
% detekterer med 2 metoder og sammenligner frekvensspektr. og SNR
% Argumenter:
%   sstrim  : trimmet signal
%   symi    : i'te symbol i signalet
%   W       : afkodningsmatrix
%   cstd    : indstillinger for transmission
%   doplot  : sand/falsk -> plotning af spektre
```

```
% Returværdier: hhv. projm og fftm for projektionsmetode og FFT-metode
%   XSP     : Powerspektrum (skaleret) for projektionsmetode
%   XP      : Powerspektrum (skaleret) for FFT-metode
%   SNR, SNR_dB
%   Andre styringsvariable

    % Sikrer at sstrim er en rækkevektor
    if iscolumn(sstrim); sstrim = sstrim'; end

    % Beregn antallet af samples til hvert symbol
    Nstft = comm_std.T_sym*comm_std.fs;

    % Her indsættes evt. manuelt offset for at få marginalt pænere signal
    % Hvis intet offset-argument er givet initialiseres til nul
    if (~exist('offset', 'var'))
        offset = 0;
    end

    % Udvælg data for i'te symbol, evt. offset
    x = sstrim(1 + (Nstft*symi) + offset : Nstft*(symi+1) + offset);
    t_vec = (0:Nstft-1) / comm_std.fs;  %tidsvektor

    % Lav evt. vinduer
    win = ones([1 Nstft]);
    %win = hamming(Nstft)'; % hann, blackman

    x = x .* win;  % Windowed signal

    % == Detektion via symbolunderrum ==
    XS = x*W;                   % Afkod med W
    XSP = XS.*conj(XS);         % Regn powerspektrum
    S = 0:comm_std.N_sym-1;     % Vektor for symbolkoder
    [~, sym] = max(XSP);        % Detekteret symbol

    % == Detektion via FFT ==
    X = fft(x);
    XP = X.*conj(X);
    f_vec = (0:Nstft-1) * comm_std.fs / Nstft;  % frekvens-akse

    fsym_vec = linspace(comm_std.f1, comm_std.f2, comm_std.N_sym);
    [~, bid] = max(XP);                         % Bin med højeste power
    binfreq = (bid-1)*(comm_std.fs / Nstft);    % Tilh. frekv. for bin
    [~, fid] = min(abs(fsym_vec-binfreq));      % Nærmeste symbolfrekv


    % == Plot ==
    if (doplot)
```

```
59
60      figure; setlatexstuff('latex');
61      sgtitle(['Transmission med ', num2str(comm_std.baudrate) ,' baud'], ...
62            'Interpreter', 'Latex', 'FontSize', 20);
63
64      subplot(311);
65      plot(t_vec, x);
66      xlabel('$t$ [s]'); ylabel('Amplitude');
67      title(['Signal i tidsdom., med ', num2str(offset), ' samples offset']);
68
69      subplot(312);
70      stem(S, 10*log10(4*XSP)); hold on;
71      stem([sym-1], 10*log10(4*XSP(sym)), 'ro'); hold off;
72      legend('Projektion','Detekteret symbol');
73      title(['Metode: Projektion i symbolunderrum, (detekt. ', ...
74              char(sym-1), ')']);
75      xlabel('Symbolkode'); ylabel('Powerspek. i dB'); xlim([0 255]);
76
77      subplot(313);
78      plot(f_vec, 10*log10(4*XP)); xlim([comm_std.f1 comm_std.f2]); hold on;
79      plot(fsym_vec(fid), 10*log10(4*XP(bid)), 'ro');
80      plot(fsym_vec(fid-1), 10*log10(4*XP(bid-1)), 'bo'); % nabo tv
81      plot(fsym_vec(fid+1), 10*log10(4*XP(bid+1)), 'bo'); hold off; % nabo th
82      xlabel('$f$ [Hz]'); ylabel('Powerspek. i dB');
83      legend({'FFT','Detekteret symbol','Naboer'})
84      title(['Metode: FFT-analyse (udsnit 1-5kHz), (detekt. ', ...
85          char(fid-1), ')']);
86      ax = gca; ax.XAxis.Exponent = 0;
87
88      end %end do plot
89
90      % == SNR for projektionsmetode ==
91      projm.mask = (S == sym-1);
92
93      % skalering divideres ud i SNR
94      projm.sig_pwr = sum( XSP(projm.mask) )/comm_std.N_sym;
95      projm.noise_pwr = sum( XSP(~projm.mask) )/comm_std.N_sym;
96      projm.SNR = projm.sig_pwr / projm.noise_pwr;
97      projm.SNR_dB = 10*log10(projm.SNR);
98
99      %indsæt powerspektrum i returværdi
100     projm.XSP = XSP/comm_std.N_sym;          % skalering
101
102     % == SNR for FFT-metode ==
103     fftm.mask1 = (0:Nstft-1 == bid-1);
104
105     % udvælg samples i transmissionsbånd, ekskl. detekt. symbol
```

```
    fftm.mask2 = (f_vec >= comm_std.f1 & f_vec <= comm_std.f2 ...
                  & ~fftm.mask1);

    % skalering divideres ud i SNR
    fftm.sig_pwr = sum( XP(fftm.mask1) )/Nstft;
    fftm.noise_pwr = sum( XP(fftm.mask2) )/Nstft;
    fftm.SNR = fftm.sig_pwr / fftm.noise_pwr;
    fftm.SNR_dB = 10*log10(fftm.SNR);

    %indsæt powerspektrum i returværdi
    fftm.XP = XP/Nstft;          % skalering

end
```

8.9 FSKdemodulate

```
function m = FSKdemodulate(sstrim, Nstft, symbols_sent, W)
% Janus, feb. 2020
% Demodulerer AFSK efter metode beskrevet i teoriafsnit
% Returnerer en tekststreng (array af chars)
% sstrim: Trimmed signal sample, uden døde sektioner

    % Sikrer at sstrim er en rækkevektor
    if iscolumn(sstrim); sstrim = sstrim'; end

    msg = [];

    % demodulér med en STFT ad gangen, en for hvert symbol
    for part = 0:symbols_sent-1

        % udvælg sektion til STFT
        x = sstrim(part * Nstft + 1 : (part+1) * Nstft);

        % beregn "DFT" og power spektrum
        X = x*W;
        XP = X.*conj(X);

        % konvertér fra sample med højeste power til symbolkode -> ASCII
        [val, idx] = max(XP);
        msg = [msg char(idx-1)];
    end

    m = msg;    % returnér afkodet besked
end
```

8.10 measure_baseline_noise

```
function x = measure_baseline_noise(seconds)
% Janus, feb. 2020
% Måler støjniveau ved forbundet mikrofon, over en periode

    % Optag med normal audiokvalitet
    fs = 44.1e3; % Hz

    % Optag baselinestøj
    base_rec = audiorecorder(fs,16,1);
    recordblocking(base_rec, seconds);
    x = getaudiodata(base_rec);
end
```

8.11 plot_mic_comparison

```
function [] = plot_mic_comparison(sig1, sig2, navn)
% Janus, feb. 2020
% Vis sammenligning i stacked diagram

    setlatexstuff('Latex');
    figure;
    subplot(211);
    plot_baseline_noise(sig1, navn{1});
    subplot(212);
    plot_baseline_noise(sig2, navn{2});
    sgtitle(['Stojgulv og sammenligning af mikrofoner'], ...
          'Interpreter', 'Latex', 'FontSize', 20);

end
```

8.12 plot_baseline_noise

```
function [] = plot_baseline_noise(signal, navn)
% Janus, feb. 2020
% Plotter en FFT for at se støjens frekvensindhold.

    % Optaget med normal audiokvalitet
    fs = 44.1e3; % Hz

    % Vis powerspektrum
    N = length(signal);
    df = fs/N;
```

```
    fvec = (0:N-1)*df;
    Ps = smoothMag( mag2db(abs(fft(signal'))), 5 ); % smoothing 5 bins

    plot(fvec, Ps);
    xlim([0 fs/2]);
    ax = gca; ax.XAxis.Exponent = 3;                % visning i kHz
    grid on;

    ylabel('Powerspek. (dB)', ...
           'Interpreter','Latex', 'FontSize', 12);
    xlabel('$f$ [Hz]', 'Interpreter','Latex', 'FontSize', 12);
    title(['Frekvensindhold i omgivelser med ', navn], ...
          'Interpreter', 'Latex', 'FontSize', 15);
end
```

8.13 trim_ends

```
function x_trim = trim_ends(x, nhood, threshold)
% trimmer de stille sektioner i et lydsignal væk
% Janus, feb. 2020
%
% x         : signal med stille sektioner, der skal fjernes
% nhood     : omegn af hvert punkt, der indregnes (antal samples)
% threshold : grænse for overgang fra "stille" til "signal"
%
% Virkemåde: Indhyldningskurven er abs(max-min), hvor max og min er regnet
% på glidende gennemsnit omkring hver sample (størrelse nhood).
% Funktionaliteten kunne nemt implementeres med simple funktioner og fx et
% MA-filter, men Matlab-funktionen imdilate håndterer en masse corner cases
% for os. Så den bruger vi.
%
% Baseret på: https://www.mathworks.com/matlabcentral/answers/
%             168185-can-anyone-tell-me-how-to-remove-unvoiced
%             -or-silenced-region-from-audio-file

    envelope = imdilate(x, true(nhood,1));  % Indhyldningskurve (abs)
    mask = envelope < threshold;            % Områder med kurve under grns
    x(mask) = [];                           % Fjern stille sektioner
    x_trim = x;                             % Returnér signal ud sektioner
end
```

8.14 triggered_record

```
function x = triggered_record(trig_lvl, fs, nBits)
```

```
% Optager et lydsignal baseret på threshold trigger
% Janus, feb. 2020
% Virkemåde: Benytter en ikke-blokerende optager, og monitorerer
% signalstyrken. Når den overstiger baseline+trig_lvl, begyndes
% optagelsen. Optagelsen stopper, når signalet igen falder under
% baseline+trig_lvl.

    pre = 0.2;   % sek, inkluderet signal før trig_lvl brydes opadgående
    post = 0.2;  % sek, inkluderet signal efter trig_lvl brydes nedadgående

    presamp = pre * fs; % antal samples i pre-perioden

    % Opret lydoptager-objekt
    rec = audiorecorder(fs, nBits, 1);  % Benytter kun 1 kanal

    record(rec);                % Start ikke-blokerende optagelse
    pause(5*pre);               % Vent på noget data i optageren

    data = getaudiodata(rec);   % Datavektor
    baseline = mean(abs(data)); % etabler baselineniveau
    disp( ['Baselinestøj: ', num2str(baseline)] );

    % Venter i dette loop indtil niveauet overskrides
    while ( mean(abs(data(end-presamp:end))) < baseline + trig_lvl)
        pause(pre);
        data = getaudiodata(rec);   % Opdatér datavektor
    end

    % Ude af første loop: trig_lvl er brudt
    % Marker sample hvorfra data optages
    n_start = rec.TotalSamples;

    % Bliv i dette loop indtil signal falder under trig_lvl
    while (mean(abs(data(end-2*presamp:end))) > baseline + trig_lvl)
        %disp( mean(abs(data(end-2*presamp:end))) );
        pause(pre);
        data = getaudiodata(rec);   % Opdatér datavektor
    end

    % Ude af første loop: trig_lvl er brudt nedadgående
    pause(post);    % medtag afsluttende buffer
    stop(rec);      % stop optagelse

    data = getaudiodata(rec);         % Opdatér datavektor med endelig data
    x = data(n_start-presamp:end);    % Returnér data for endelig optagelse

end
```

8.15 smoothMag

KPL E3DSB

```
function Y = smoothMag(X,M)
% Smoothing of signal. Eg. frequency magnitude spectrum.
% X must be a row vector, and M must be odd.
% KPL 2016-09-19
    N=length(X);
    K=(M-1)/2;
    Xz=[zeros(1,K) X zeros(1,K)];
    Yz=zeros(1,2*K+N);
    for n=1+K:N+K
        Yz(n)=mean(Xz(n-K:n+K));
    end
    Y=Yz(K+1:N+K);
end
```