**Case 2 FSK Transmission**

**Gruppe 2**

|  |  |  |
| --- | --- | --- |
| **Studienummer** | **Navn** | **Studieretning** |
| 202310755 | Peter Thule Kirketerp Linstad | E |
| 202004347 | Otto Sejrskild Santesson | E |
| 202001087 | Mudar Issam | E |

Contents

[Indledning og formål 3](#_Toc211884253)

[Opgave 1 – Signal generation / kodning 3](#_Toc211884254)

[A. Generer et lydsignal-array med ”FSKgenerator” funktionen. 3](#_Toc211884255)

[B. Analyser signalet 3](#_Toc211884256)

[C. Analyser signalet vha. Short-Time Fourier Transform 5](#_Toc211884257)

[D. Eksperimenter med ”FSKgenerator” funktionen 6](#_Toc211884258)

[Lydsignal ved Mindre båndbredde 7](#_Toc211884259)

[lydsignal ved støre Tsymbol 7](#_Toc211884260)

[Lydsignal ved mindre sampling frequency 8](#_Toc211884261)

[Opgave 2 - ? 9](#_Toc211884262)

[Opgave 3 – Signal-støj-forhold 10](#_Toc211884263)

[Formål 10](#_Toc211884264)

[Teori 10](#_Toc211884265)

[Implementation 10](#_Toc211884266)

[Resultater og fortolkning 12](#_Toc211884267)

[Opgave 4 – Bit rate 14](#_Toc211884268)

[Formål 14](#_Toc211884269)

[Del A – Bitrate vs. Fejlrate 14](#_Toc211884270)

[Del B – Vindueslængde 15](#_Toc211884271)

[Del C – SNR’s betydning 17](#_Toc211884272)

[Del D – Flere frekvenser pr. symbol 19](#_Toc211884273)

[Del E – Overordnet trade-off 20](#_Toc211884274)

[Referencer 21](#_Toc211884275)

[Appendix 22](#_Toc211884276)

[FSKdecoder.m 22](#_Toc211884277)

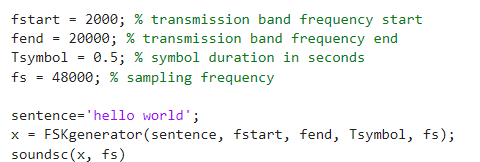
[Case2.m 24](#_Toc211884278)

# Indledning og formål

# Opgave 1 – Signal generation / kodning

## A. Generer et lydsignal-array med ”FSKgenerator” funktionen.

I denne opgave brugte vi funktionen "FSKgenerator.m" fra Brightspace til at generere et lydsignal bestående af syv toner, som repræsenterer sætningen "hello world".



## B. Analyser signalet

*Analyser signalet for at finde ud af, hvilke karakterer, som svarer til hvilke frekvenser. I skal se på signalet i både tids- og frekvens-domænet.*

Det genererede signal analyseres ved at plotte det i tids- og frekvensdomænet ved hjælp af koden nedenfor.

A screenshot of a computer program

AI-generated content may be incorrect. A white background with black text

AI-generated content may be incorrect.

A graph of a graph

AI-generated content may be incorrect.

Figur Plot af signal i tidsdomæne

A graph of a graph

AI-generated content may be incorrect.

Figur Plot af signal i frekvens

For at bestemme, hvilke bogstaver der er knyttet til specifikke frekvenser, lavede vi et sæt programmer, der genererede individuelle lydfiler for hvert bogstav. Efterfølgende blev lydfilerne analyseret i frekvensdomænet for at undersøge deres karakteristiske frekvensprofiler.

A graph of a graph

AI-generated content may be incorrect.

Figur Identifikation af symbol til hver frekvens

## C. Analyser signalet vha. Short-Time Fourier Transform

*Analyser signalet vha. Short-Time Fourier Transform (kan læses om i bogen) – dvs. med spektrogram-plot. Forklar trade-off imellem opløsningen i tid og frekvens.*

For at foretage en videre analyse af signalet blev der oprettet et spektrogram ved hjælp af koden vist nedenfor.

A number and text on a white background

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

Figur Spektrogram af signalet

Signalet blev analyseret i MATLAB ved hjælp af funktionen spectrogram(), som anvender Short-Time Fourier Transformation (STFT). Resultatet er vist i figur 4.

STFT er et effektivt værktøj til at visualisere, hvordan frekvenserne ændrer sig over tid. Dog eksisterer der en afvejning mellem tids- og frekvensopløsning.

Et kort vindue giver høj tidsopløsning, men reducerer frekvensopløsningen – mens et langt vindue giver det modsatte. En øget samplingsfrekvens kan forbedre begge dele, men medfører samtidig større datamængder og længere beregningstid.

I figur 4 ses et tydeligt frekvensmønster over tid, som illustrerer denne balance mellem tids- og frekvensopløsning.

## D. Eksperimenter med ”FSKgenerator” funktionen

*Eksperimenter med ”FSKgenerator” funktionen for at få en forståelse af input parametrene.*

Vi tog udgangspunkt i koden fra opgave A og eksperimenterede med forskellige inputparametre for at undersøge, hvordan de påvirker signalet.

### Lydsignal ved Mindre båndbredde

A graph of a graph

AI-generated content may be incorrect.

Figur Plot lydsignal ved mindre bånbbredde

### lydsignal ved støre Tsymbol

A graph with blue lines

AI-generated content may be incorrect.

Figur Plot lydsignal ved støre Tsymbol

### Lydsignal ved mindre sampling frequency

A graph of a graph

AI-generated content may be incorrect.

Figur Plot Lydsignal ved mindre sampling frequency

Ved eksperimentering med funktionen "FSKgenerator()" opnåede vi følgende forståelse af parametrene:

**mysymbolseq**: Den besked, der skal sendes.

**fstart:** Den laveste frekvens, der bruges til at sende et symbol.

**fend:** Den højeste frekvens, der bruges til at sende et symbol. Sammen med fstart definerer den det anvendelige frekvensområde samt afstanden mellem de benyttede frekvenser.

**Tsymbol**: Den tid, hver symbols frekvens udsendes i.

fs: Samplingsfrekvensen, som signalet genereres med.

# Opgave 2 - ?

# Opgave 3 – Signal-støj-forhold

Formål  
Formålet er at undersøge, hvordan signal-til-støj-forholdet (SNR) forringes, når afstanden mellem sender og modtager øges. Vi ønsker at kvantificere, hvorvidt faldet i SNR primært skyldes reduceret signalstyrke eller ændret støjniveau, samt at relatere observationerne til teorien om effektspektre og SNR-beregning i frekvensdomænet.

## Teori

SNR defineres som forholdet mellem signalets effekt og støjens effekt og udtrykkes ofte i dB [1]:

,

Effekten for et diskret signal beregnes typisk som middelværdien af kvadratet af signalets amplitude og er givet ved [1]:

I opgaven er vi interesseret i at måle SNR i frekvens-domænet og dermed gælder Parsevals teorem [1]:

Ud fra dette kan vi beregne Ps og Pn ud fra fft’en, hvor powerspektrum er givet som [1]:

Ved FSK-signal består hvert symbol af én tone med en frekvens, der repræsenterer et tegn. For et enkelt symbol fremstår signalet derfor som et klart peak i spektret, mens resten af spektret repræsenterer støjgulvet. Faldende SNR kan derfor ses som et lavere peak-niveau relativt til støjgulvet.

## Implementation

Vi har først optaget den samme besked ”hello world” i afstandende 0-4 m og for hver fil har vi:

Fjernet stille perioder før og efter signalet, samt fjernet DC-komponent – dette gøres på samme måde som i opg 2:

|  |
| --- |
| % --- indlæs og trim stille perioder (samme som i opg. 2)  path = matlab.desktop.editor.getActiveFilename;  thisFolder = fileparts(path);  audioFile = fullfile(thisFolder, files(k));  [y, fs] = audioread(audioFile);  threshold = 0.01;  mask = abs(y) > threshold;  yTrimmed = y(find(mask,1,'first'):find(mask,1,'last'));  yTrimmed = yTrimmed - mean(yTrimmed); % fjern DC-komponent |

Herefter har vi udvalgt ét symbol/tone segment, som også er beskrevet i opgavebeskrivelsen, og beregnet fft og powerspektrum:

|  |
| --- |
| % --- tag ét symbol til analyse ---  N = round(Tsymbol \* fs);  seg = yTrimmed(1:N);  % --- FFT og power-spektret ---  Y = fft(seg);  P = abs(Y).^2 / (N\*N); % lineær effekt  f = (0:N-1)\*(fs/N); % frekvensakse [Hz] |

Herefter bestemmer vi signal-peak som maksimum af P[m] og støjgulvet som median uden for peakområdet, samt omregner SNR til SNRdB:

|  |
| --- |
| % --- vælg kun FSK-båndet ---  band = (f >= fstart) & (f <= fend);  fB = f(band);  PB = P(band);  % --- beregn SNR (peak vs. median noise) ---  [Ppk, ipk] = max(PB);  excl = false(size(PB));  excl(max(ipk-1,1):min(ipk+1,numel(PB))) = true;  noise\_lin = median(PB(~excl));  SNRdB(k) = 10\*log10(Ppk / max(noise\_lin, eps)); |

## Resultater og fortolkning

|  |
| --- |
| Figur - fuld skala powerspektrum af de fem signaler |
| Figur - Zoomet powerspektrum af de fem signaler med fokus på peak-niveauer |

Effektspektret for de fem optagelser (0–4 m) ses på Figur 8 og Figur 9.  
Generelt ligger støjgulvet stabilt omkring –80 til –100 dB, hvilket indikerer et jævnt, hvidt støjspektrum i optagelserne. Dog ses ved 0 m en let forhøjelse af støjgulvet, sandsynligvis pga. måleopstilling og andre fejlkilder foregået ved optagelse og afspilning.

Som forventet er peakniveauerne højere for korte afstande, hvor signalet modtages med større amplitude.  
Et mindre afvigende tilfælde optræder ved 3 m, hvor både peak og beregnet SNR er højere end ved 2 m. Dette kan sandsynligvis også forklares ved uregelmæssigheder ved optagelse og afspilning af lydsignalet, som er lidt svært at redegøre præcis for.

|  |
| --- |
| Figur - Plot af SNR og afstand af afspillet lydklip |

På ovenstående Figur 10 ser vi plottet af SNR i dB op ad y-aksen og afstanden mellem højttaler og mikrofon på x-aksen. Som vi også kom ind på før, ser vi en overordnet tendens med at SNR falder i takt med at afstanden øges. Vi ser som benævnt et lidt lave SNR ved 0 m optagelsen, hvilket skyldes det lidt højere støjgulv ved denne optagelse. Ved at kigge på Figur 9 kan vi nemlig se at signalet ved 0 m har den højeste peak-værdi og den lavere SNR må derfor skyldes at støjens effekt er større er end ved de andre signaler.

Derimod ved afvigelsen mellem signalet på 2 og 3 meter skyldes det ikke at støjens effekt er anderledes ved de to signaler, men derimod at signalets effekt ved optagelsen på 3 meter er højere end ved 2 meter. Igen skyldes dette nok nogle målefejl under optagelse eller afspilning.

Til sidst er det værd at bemærke at der gennemgående for alle signaler er et relativt højt SNR, hvilket vil sige, at det er nemt at skelne mellem signal og støj, da SNR ligger mellem ca. 58-68 dB.

# Opgave 4 – Bit rate

## Formål

Formålet er at undersøge, hvordan symboltid (), vindueslængde (), og signal-til-støj-forhold (SNR) påvirker systemets evne til at overføre information fejlfrit. Derved kan den maksimale pålidelige bitrate og de vigtigste trade-offs i et FSK-baseret kommunikationssystem bestemmes.

## Del A – Bitrate vs. Fejlrate

Vi undersøger, hvor kort kan være, altså længden af sinus-tonen, før dekoderen begynder at lave fejl.

Antagelse: 1 symbol = 1 byte (ASCII) => 8 bits per symbol.

Bitrate = 8 /

Ved at variere fra 0.5 s til 0.0019 s er FER (frame error rate) og BER (bit error rate) beregnet ud fra egen dekoder:

|  |
| --- |
| for k = 1:numel(Ts\_list)  Tsymbol = Ts\_list(k);  % --- Generér FSK-signal for hele beskeden  x = FSKgenerator(msg, fstart, fend, Tsymbol, fs);  % --- Dekod med vores egen decoder (samme som i opgave 2, men kun med 256 frekvenser)  yhat = FSKdecoder(x, fstart, fend, Tsymbol, fs);  % --- Fejlmåling  % Frame Error Rate: 1 hvis bare ét tegn er forkert, ellers 0  FER(k) = any(yhat ~= msg);  % Bit Error Rate (grov): antal forkerte tegn \* 8 / total bits  n\_char\_err = count\_char\_errors(yhat, msg);  BER(k) = (n\_char\_err \* 8) / (length(msg) \* 8);  end |

|  |
| --- |
| Figur - Plot af Frame Error Rate vs bitrate(øverst) og Bit Error Rate vs bitrate (nederst) |

Resultaterne viser, at der kan sendes fejlfrit ned til en symboltid/ på ca. 0.002 s (≈ 4000 bit/s). Under denne grænse øges BER kraftigt, fordi de enkelte symbolers spektrum overlapper hinanden.

## Del B – Vindueslængde

Vi vil her undersøge hvad betydningen af vindueslængden har for amplitude-spektret af en sinus-tone

Ved at ændre () ses, at hovedlobebredden er omvendt proportional med . Kortere reducerer frekvensopløsningen:

|  |
| --- |
| for k = 1:numel(Ts\_list)  Ts = Ts\_list(k);  N = round(Ts\*fs); % vindueslængde (antal samples)  n = 0:N-1;  x = cos(2\*pi\*f0\*n/fs); % ren sinus (rektangulært vindue)  % FFT (zero-padding for pæn kurve; opløsningen bestemmes stadig af N)  Nfft = 2^nextpow2(max(4096, N));  X = fft(x, Nfft);  f = (0:Nfft-1)\*(fs/Nfft);  % Amplitude (normaliser peak til 0 dB for sammenligning)  A = abs(X);  A = A / max(A);  AdB = 20\*log10(A + 1e-12); |

|  |
| --- |
| Figur - Amplitude-spektrer for forskellige N-værdier |

Lille giver smallere hovedlobe i DFT’en og dermed bedre adskillelse mellem symbolfrekvenser. Når bliver for kort, breder hovedloben sig og overlapper nabotoner, hvilket øger fejlraten.

## Del C – SNR’s betydning

For en fast er systemet testet ved SNR fra 0 til 60 dB.

Vi har herudover også valgt en markant smallere bånd, da SNR var så høj, at der skulle noget ekstra til, for at vise effekten af SNR’s betydning ved større støj-effekt.

Vi laver et rent FSK-signal, altså uden nogen form for støj. Herefter udregner vi signalets gennemsnitlige effekt. Til sidst laver vi så et støjsignal ud fra den ønskede SNR og forsøger at decode dette:

|  |
| --- |
| fs = 30000;  fstart = 9000;  fend = 10000; % Bruger smallere bånd  Ts = 0.01; % og kortere symboltid for at vise SNR betydning  msg = 'hello world hello world';  % Liste af SNR-værdier i dB  SNR\_dB = 0:5:60; % fra 0 dB (meget støj) til 60 dB (næsten perfekt)  FER = zeros(size(SNR\_dB));  BER = zeros(size(SNR\_dB));  % Original FSK-signal  x = FSKgenerator(msg, fstart, fend, Ts, fs);  Px = mean(x.^2); % signalets gennemsnitlige effekt  for k = 1:length(SNR\_dB)  snr\_val = SNR\_dB(k);    % Beregn nødvendig støj-effekt for den ønskede SNR  Pn = Px / (10^(snr\_val/10));  noise = sqrt(Pn) \* randn(size(x)); % hvidt støjsignal    % Tilføj støj  y = x + noise;  % Decode  yhat = FSKdecoder(y, fstart, fend, Ts, fs);    % Frame- og bitfejl  FER(k) = any(yhat ~= msg); % 1 hvis mindst én fejl  BER(k) = sum(yhat ~= msg) / length(msg);    end |

|  |
| --- |
| Figur - Plot af SNR's betydning for vores decoder |

Vi ser her, at når SNR er over 10 dB har vores decoder ingen problemer med at decode signalet, da støjen er svag ift. signalet. Når SNR derimod er under 10 dB bliver støjen så kraftig, at decoderen ikke kan skelne mellem støj og signaler og begynder derfor at gætte forkert, hvilket kan ses på Figur 13 ved at der bliver introduceret Bit Errors.

## Del D – Flere frekvenser pr. symbol

Hvis flere samtidige frekvenser kodes i ét symbol, kan bitraten øges:

hvor er antallet af samtidige toner.  
Dog vokser overlap og interferens mellem nærliggende frekvenser hurtigt, især ved lav SNR. Dette sætter en praktisk grænse for, hvor mange frekvenser systemet kan håndtere uden øget fejlrate.

Så Ja, der er en grænse, som bestemmes af forholdet mellem symboltid og båndbredde.  
Frekvenserne skal ligge mindst fra hinanden, så antallet af mulige frekvenser maksimalt er . Hvor B er båndbredden  
I praksis er grænsen dog væsentligt lavere på grund af støj og spektral overlapning.

Del E – Overordnet trade-off  
De samlede resultater fra delopgaverne 4A–4D viser tydeligt, at systemets ydeevne afhænger af et grundlæggende kompromis mellem symboltid, båndbredde og SNR.  
Når symboltiden reduceres, øges bitraten tilsvarende, da flere symboler kan sendes pr. sekund.  
Dette opnås dog på bekostning af frekvensopløsningen, som bliver dårligere, når hvert symbol består af færre samples .  
Som det fremgår af resultaterne fra opgave 4A og 4B, bliver tonepeaks i spektret gradvist bredere ved kortere symboltider, hvilket øger risikoen for overlap og dermed forveksling af frekvenser i dekodningen.

I opgave 4C sås, at systemet generelt fungerer fejlfrit ved høje SNR-værdier (over 20–30 dB), men at støj hurtigt reducerer pålideligheden ved lavere SNR.  
Et højt støjniveau medfører, at de smalle peaks fra FSK-symbolet bliver svære at skelne fra støjgulvet, og selv små forvrængninger i amplitude kan resultere i fejlafkodning.  
Dette kan kompenseres ved at forlænge symboltiden, hvilket forbedrer frekvensopløsningen og dermed systemets robusthed, men samtidig sænker bitraten.

# Referencer

[1] Spektogram og SNR slide fra brightspace

<https://brightspace.au.dk/content/enforced/183504-LR50157/csfiles/home_dir/SpektrogramSNR/SpectrogramSNR_Slides.pdf?isCourseFile=true&ou=183504>

# Appendix

## FSKdecoder.m

|  |
| --- |
| % function x = FSKgenerator(signal, fstart, fstop, Tsymbol, fs)  %  % signal is the audio signal in the time domain  % fs is sampling frequency  % fstart = transmission band frequency start  % fend = transmission band frequency end  % Tsymbol = symbol duration in seconds  % The output is the decoded message from the audio signal  % function x = FSKdecoder(signal, fstart, fend, Tsymbol, fs)  %  % N = length(signal);  %  % samples\_per\_symbol = Tsymbol\*fs;  %  % remainder = mod(N, samples\_per\_symbol);  % if remainder ~= 0  % pad = samples\_per\_symbol - remainder;  % signal = [signal; zeros(pad,1)];  % end  %  % N = length(signal);  %  % n\_symbols = N/samples\_per\_symbol;  %  % x = '';  %  % farray = linspace(fstart,fend, 256);  %  % for k = 1:n\_symbols  %  % start\_idx = (k-1)\*samples\_per\_symbol+1;  % end\_idx = k\*samples\_per\_symbol;  % segment = signal(start\_idx:end\_idx);  % N = length(segment);  %  % Perform DFT on the segment, find the dominant frequency, map that  % frequency to the synmbol and append that symbol to the x string  %  % X\_m = [];  % m\_unique = ceil((N+1)/2);  %  % for i = 0:m\_unique-1  % sum = 0;  % for j = 0:N-1  % sum = sum + segment(j+1)\*exp((-(2\*pi\*1i)/N)\*i\*j);  % end  % X\_m(i+1) = sum;  % end  % frequency\_bins = (0:m\_unique-1)\*(fs/N);  %  % [~, max\_idx] = max(abs(X\_m));  %  % dom\_freq = frequency\_bins(max\_idx);  %  % [~, symbol\_idx] = min(abs(farray - dom\_freq));  %  %  % x = [x, char(symbol\_idx)];  % end  % Opdateret funktion så den kun regner på de 256 frekvenser  function x = FSKdecoder(signal, fstart, fend, Tsymbol, fs)  N = length(signal);  % 1) HELTAL samples pr. symbol  samples\_per\_symbol = round(Tsymbol\*fs);  % Pad til helt antal symboler  remainder = mod(N, samples\_per\_symbol);  if remainder ~= 0  pad = samples\_per\_symbol - remainder;  signal = [signal; zeros(pad,1)];  end  N = length(signal);  n\_symbols = N / samples\_per\_symbol;  x = '';  % 2) 256 kandidater (rækkevektor)  farray = linspace(fstart, fend, 256);  farray = farray(:).'; % tving til 1×256  % 3) Forbered eksponential-bank én gang (Ns×256)  Ns = samples\_per\_symbol;  n = (0:Ns-1).' / fs; % Ns×1  E = exp(-1j\*2\*pi \* (n \* farray)); % (Ns×1)\*(1×256) => Ns×256  for k = 1:n\_symbols  start\_idx = (k-1)\*samples\_per\_symbol + 1;  end\_idx = k\*samples\_per\_symbol;  % 4) Sørg for rækkevektor 1×Ns  segment = signal(start\_idx:end\_idx);  segment = segment(:).'; % 1×Ns  % Projektion kun på 256 frekvenser: (1×Ns)\*(Ns×256) = 1×256  X = segment \* E;  % Vælg stærkeste kandidat og map direkte til tegn  [~, symbol\_idx] = max(abs(X).^2);  x = [x, char(symbol\_idx)];  end  %{  % Compute the frequency bins for the DFT  frequency\_bins = (0:m\_unique-1)\*(fs/N);  % Find the dominant frequency in the DFT result  [~, max\_idx] = max(abs(X\_m));  dominant\_freq = frequency\_bins(max\_idx);  % Map the dominant frequency to the corresponding symbol  symbol = mapFrequencyToSymbol(dominant\_freq, fstart, fend);  x = [x, symbol]; % Append the decoded symbol to the output string    %}  %{  X = [];  for j = 0:length(frequency\_bins)-1  sum = 0;  for i = 0:N-1  sum = sum + x(i+1)\*exp(((-2\*pi\*1i)/N)\*i\*j);  end  X(j+1) = sum;  end  %}  %{  The encoder algorithm  function x = FSKgenerator(mysymbolseq, fstart, fend, Tsymbol, fs)  farray = linspace(fstart, fend, 256); % 256 frequencies spread out in band  A = 1; % amplitude  n = 0:(round(Tsymbol\*fs)-1);  myids = double(mysymbolseq); % convert 'abcd' to [97 98 99 100].. ie. 256 possible values  x = []; %empty array  for i=1:length(myids),  myfreq = farray(myids(i)); % choose freq for current char  sig = A\*cos(2\*pi\*n\*myfreq/fs); % create signal  x = [x sig]; % add to full signal  end  %} |

## Case2.m

|  |
| --- |
| %% Opgave 1 – Signal generation / kodning  clear;  close all;  clc;  % A. Generer et lydsignal-array med "FSKgenerator" funktionen.  fstart = 2000; % transmission band frequency start  fend = 20000; % transmission band frequency end  Tsymbol = 0.5; % symbol duration in seconds  fs = 48000; % sampling frequency  sentence='hello world';  x = FSKgenerator(sentence, fstart, fend, Tsymbol, fs);  soundsc(x, fs)  %%  % B. Analyser signalet for at finde ud af, hvilke karakterer, som svarer  % til hvilke frekvenser. I skal se på signalet i både tids- og frekvens-domænet.%  figure(1)  N=length(x); % antal sample  plot((0:N-1)/fs,x) %% Plot x signal i tide-domænet  xlim([1 1.009]);  xlabel("Tid [s]")  ylabel("Amplitude")  title("plot lydsignal tids-domænet")  figure(2)  plot(0:fs/N:fs-fs/N,abs(fft(x))) %% Plot x signal i frekvens-domænet  xlabel("Frekvens [Hz]")  ylabel("|X(f)|")  title("plot lydsignal frekvens-domænet")  figure(3)  plot(0:fs/N:fs-fs/N,abs(fft(x))) %% Plot x signal i frekvens-domænet  xlabel("Frekvens [Hz]")  ylabel("|X(f)|")  title("plot lydsignal frekvens-domænet")  xlim([500 10000])  h = FSKgenerator('h', fstart, fend, Tsymbol, fs);  e = FSKgenerator('e', fstart, fend, Tsymbol, fs);  l = FSKgenerator('l', fstart, fend, Tsymbol, fs);  o = FSKgenerator('o', fstart, fend, Tsymbol, fs);  space = FSKgenerator(' ', fstart, fend, Tsymbol, fs);  w = FSKgenerator('w', fstart, fend, Tsymbol, fs);  r = FSKgenerator('r', fstart, fend, Tsymbol, fs);  d = FSKgenerator('d', fstart, fend, Tsymbol, fs);  figure(4)  N =length(h);  x\_akse = (0:fs/N:fs-fs/N);  plot(x\_akse,abs(fft(h)))  hold on  N =length(e);  x\_akse = (0:fs/N:fs-fs/N);  plot(x\_akse,abs(fft(e)))  N =length(l);  x\_akse = (0:fs/N:fs-fs/N);  plot(x\_akse,abs(fft(l)))  N =length(o);  x\_akse = (0:fs/N:fs-fs/N);  plot(x\_akse,abs(fft(o)))  N =length(space);  x\_akse = (0:fs/N:fs-fs/N);  plot(x\_akse ,abs(fft(space)))  N =length(w);  x\_akse = (0:fs/N:fs-fs/N);  plot(x\_akse ,abs(fft(w)))  N =length(r);  x\_akse = (0:fs/N:fs-fs/N);  plot(x\_akse,abs(fft(r)))  N =length(d);  x\_akse = (0:fs/N:fs-fs/N);  plot(x\_akse ,abs(fft(d)))  xlabel("Frekvens [Hz]")  ylabel("|X(f)|")  title("plot lydsignal frekvens-domænet")  legend(["h","e","l", "o"," ", "w", "r", "d"])  xlim([1000 15000])  hold off  %%  %C. Analyser signalet vha. Short-Time Fourier Transform (kan læses om i bogen) – dvs. med  % spektrogram-plot. Forklar trade-off imellem opløsningen i tid og frekvens. %  figure(5)  spectrogram(x, hanning(512), 500, 1024, fs, 'yaxis')  ylim([0.5 10]);  figure(6)  spectrogram(x, hanning(2048), 500, 1024, fs, 'yaxis')  ylim([0.5 10]);  %%  %D. Eksperimenter med "FSKgenerator” funktionen for at få en forståelse af input  % parametrene.  %% lydsignal ved Mindre båndbredde  figure(7)  fstart = 1200; % transmission band frequency start  fend = 6000; % transmission band frequency end  Tsymbol = 0.5; % symbol duration in seconds  fs = 48000; % sampling frequency  sentence='hello world';  x = FSKgenerator(sentence, fstart, fend, Tsymbol, fs);  N=length(x);  plot(0:fs/N:fs-fs/N,abs(fft(x))) %% Plot x signal i frekvens-domænet  xlim([0 8000]);  xlabel("Frekvens [Hz]")  ylabel("|X(f)|")  title("frekvens-domænet til lydsignal ved Mindre båndbredde")  sound(x,fs)  %% lydsignal ved støre Tsymbol  figure(8)  fstart = 2000; % transmission band frequency start  fend = 20000; % transmission band frequency end  Tsymbol = 0.8; % symbol duration in seconds  fs = 48000; % sampling frequency  sentence='hello world';  x = FSKgenerator(sentence, fstart, fend, Tsymbol, fs);  N=length(x);  plot(0:fs/N:fs-fs/N,abs(fft(x))) %% Plot x signal i frekvens-domænet  xlim([0 12000]);  xlabel("Frekvens [Hz]")  ylabel("|X(f)|")  title("frekvens-domænet til lydsignal ved støre Tsymbol")  sound(x,fs)  %% lydsignal ved mindre sampling frequency  figure(9)  fstart = 500; % transmission band frequency start  fend = 1500; % transmission band frequency end  Tsymbol = 0.5; % symbol duration in seconds  fs = 4000; % sampling frequency  sentence='hello world';  x = FSKgenerator(sentence, fstart, fend, Tsymbol, fs);  N=length(x);  plot(0:fs/N:fs-fs/N,abs(fft(x))) %% Plot x signal i frekvens-domænet  xlim([0 2000]);  xlabel("Frekvens [Hz]")  ylabel("|X(f)|")  title("frekvens-domænet til lydsignal ved mindre sampling frequency")  sound(x,fs)  %% Opgave 2 – Dekodning  % A. Send besked til en anden gruppe  % OBS: Vi sender en besked til os selv; vi afspiller nedenstående indkodede  % besked og optager på en telefon  fs = 30000;  fstart = 100;  fend = 10000;  lydsignal = FSKgenerator('hello world', 100, 10000, 0.2, fs);  soundsc(lydsignal,fs);  %%  % Load audio file from the current work space folder  path = matlab.desktop.editor.getActiveFilename;  thisFolder = fileparts(path);  audioFile = fullfile(thisFolder, 'hello\_world\_0m.m4a');  [y, fs] = audioread(audioFile);  %%  % Play said audio file  soundsc(y,fs);  %%  % Tilklip filen, så de stille perioder før og efter lydsignalet i  % optagelsen er fjernet  threshold = 0.01;  mask = abs(y) > threshold;  firstSample = find(mask, 1, 'first');  lastSample = find(mask, 1, 'last');  yTrimmed = y(firstSample:lastSample,:);  soundsc(yTrimmed,fs);  %% Visually inspecting the signal via spectrogram  figure(43)  spectrogram(yTrimmed, blackman(1000), 0,1000, fs, 'yaxis')  %% Decoding the signal using self-made algorithm  % Adding path to the current work space, so Matlab can access the function  addpath(thisFolder);  % received\_message = FSKdecoder(signal, fs)  % mysymbolseq, fstart, fend, Tsymbol, fs  received\_message = FSKdecoder(yTrimmed,100, 10000, 0.2, fs)  % Current version is quite slow, because it's O(N^2) complexity – could be  % cut down to 256, because those are the frequencies that we are interested  % in    %%  N = length(y);  t = (0:N-1)/fs;  figure(42)  plot(t,y)  %%  % Decode the received signal using a simple thresholding method  threshold = 0.5; % Define a threshold for detection  decodedChars = ''; % Initialize the decoded characters string  for i = 1:length(x)  if x(i) > threshold  decodedChars = [decodedChars, '1']; % Detected signal  else  decodedChars = [decodedChars, '0']; % No signal  end  end  %% Opgave 3 – SNR-analyse af FSK-signal ved forskellige afstande  clear; close all; clc;  files = ["hello\_world\_0m.m4a", "hello\_world\_1m.m4a", "hello\_world\_2m.m4a", "hello\_world\_3m.m4a", "hello\_world\_4m.m4a"];  dists = [0 1 2 3 4]; % afstande i meter  fstart = 100;  fend = 10000;  Tsymbol = 0.2; % symboltid (samme som ved optagelse)  SNRdB = nan(size(files));  figure; clf; hold on; grid on;  for k = 1:numel(files)  % --- indlæs og trim stille perioder (samme som i opg. 2)  path = matlab.desktop.editor.getActiveFilename;  thisFolder = fileparts(path);  audioFile = fullfile(thisFolder, files(k));  [y, fs] = audioread(audioFile);  threshold = 0.01;  mask = abs(y) > threshold;  yTrimmed = y(find(mask,1,'first'):find(mask,1,'last'));  yTrimmed = yTrimmed - mean(yTrimmed); % fjern DC-komponent  % --- tag ét symbol til analyse ---  N = round(Tsymbol \* fs);  seg = yTrimmed(1:N);  % --- FFT og power-spektret ---  Y = fft(seg);  P = abs(Y).^2 / (N\*N); % lineær effekt  f = (0:N-1)\*(fs/N); % frekvensakse [Hz]  % --- vælg kun FSK-båndet ---  band = (f >= fstart) & (f <= fend);  fB = f(band);  PB = P(band);  % --- beregn SNR (peak vs. median noise) ---  [Ppk, ipk] = max(PB);  excl = false(size(PB));  excl(max(ipk-1,1):min(ipk+1,numel(PB))) = true;  noise\_lin = median(PB(~excl));  SNRdB(k) = 10\*log10(Ppk / max(noise\_lin, eps));  % --- konverter til dB ---  PdB = 10\*log10(PB + eps);  % --- plot i dB ---  plot(fB, PdB, 'LineWidth', 1.2, 'DisplayName', ...  sprintf('%s (SNR≈%.1f dB)', files(k), SNRdB(k)));  end  xlabel('Frekvens [Hz]');  ylabel('Effekt [dB]');  title('Power Spectrum (dB) for FSK-signal ved forskellige afstande');  legend('Location','best');  xlim([fstart fend]);  %% Opgave 3C – SNR som funktion af afstand  % Til sidst plotter vi de beregnede SNR-værdier for hver afstand.  % Vi forventer, at SNR falder, når afstanden øges, da signalstyrken  % aftager hurtigere end støjniveauet ændrer sig.  figure; clf;  plot(dists, SNRdB, 'o-','LineWidth',1.5);  grid on;  xlabel('Afstand [m]');  ylabel('SNR [dB]');  title('SNR (peak/median) vs. afstand');  %% test for at tjekke decoder virker  clear; close all; clc;  fs = 30000;  fstart = 100;  fend = 10000;  Tsymbol = 0.2;  testmsg = 'hello';  x = FSKgenerator(testmsg, fstart, fend, Tsymbol, fs);  yhat = FSKdecoder(x, fstart, fend, Tsymbol, fs);  disp("Expected: " + testmsg);  disp("Decoded: " + yhat);  %% Opgave 4A – Bit rate vs. fejlfri dekodning  clear; close all; clc;  % A) Opsæt test: sweep Tsymbol og mål fejlrate + bitrate  % Vi undersøger, hvor kort Tsymbol kan være, før dekoderen begynder at lave fejl.  % Antagelse: 1 symbol = 1 byte (ASCII) => 8 bits per symbol.  % Bitrate = 8 / Tsymbol [bits/s]  %  % Parametre holdes som i opg. 2 for konsistens.  fs = 30000; % samplingsfrekvens  fstart = 100; % start på FSK-bånd  fend = 10000; % slut på FSK-bånd  msg = 'hello world hello world'; % lidt længere teststreng  Ts\_list = [0.50 0.30 0.20 0.15 0.10 0.08 0.06 0.05 0.04 0.03 0.025 0.020 0.010 0.005 0.002 0.0019];  bitrate = 8 ./ Ts\_list;  FER = nan(size(Ts\_list)); % Frame Error Rate (fejl i hele tekst-strengen)  BER = nan(size(Ts\_list)); % Bit Error Rate (groft estimeret som char-fejl\*8 / (len\*8))  % Hjælpere til BER-estimat (pr. tegn): vi tæller tegn der er forkerte  count\_char\_errors = @(a,b) sum(a ~= b);  for k = 1:numel(Ts\_list)  Tsymbol = Ts\_list(k);  % --- Generér FSK-signal for hele beskeden  x = FSKgenerator(msg, fstart, fend, Tsymbol, fs);  % --- Dekod med vores egen decoder (samme som i opgave 2, men kun med 256 frekvenser)  yhat = FSKdecoder(x, fstart, fend, Tsymbol, fs);  % --- Fejlmåling  % Frame Error Rate: 1 hvis bare ét tegn er forkert, ellers 0  FER(k) = any(yhat ~= msg);  % Bit Error Rate (grov): antal forkerte tegn \* 8 / total bits  n\_char\_err = count\_char\_errors(yhat, msg);  BER(k) = (n\_char\_err \* 8) / (length(msg) \* 8);  end  %% B) Plot: bitrate vs. fejlrate (FER & BER)  figure;  subplot(2,1,1);  stem(bitrate, FER, 'filled'); grid on;  xlabel('Bitrate [bits/s]'); ylabel('FER');  title('Frame Error Rate vs. Bitrate (1 symbol = 1 byte)');  ylim([-0.05 1.05]);  subplot(2,1,2);  plot(bitrate, BER, 'o-','LineWidth',1.2); grid on;  xlabel('Bitrate [bits/s]'); ylabel('BER');  title('Bit Error Rate vs. Bitrate');  % Find maksimal fejlfri bitrate:  ok\_idx = find(FER==0);  if ~isempty(ok\_idx)  max\_ok\_bitrate = max(bitrate(ok\_idx));  fprintf('Maks. fejlfri bitrate ≈ %.1f bits/s (Tsymbol ≈ %.3f s)\n', ...  max\_ok\_bitrate, 8/max\_ok\_bitrate);  else  fprintf('Ingen fejlfri bitrate i dette sweep – forøg Tsymbol eller SNR.\n');  end  %% C) Kort diskussion  % Vi ser, at når Tsymbol bliver kort (høj symbolrate/bitrate), begynder  % dekoderen at lave fejl. Det hænger sammen med DFT/vindue-teori:  % Færre samples pr. symbol => bredere hovedlobe (= ringere frekvensopløsning)  % og mere lækage, hvilket gør peaks mindre adskilte ved samme frekvensafstand.  % Dermed stiger sandsynligheden for, at nabofrekvenser forveksles – især ved  % begrænset SNR. (Se slides om vinduer/leakage/oploesning).  %% Opgave 4B – Betydning af vindueslængde N for amplitude-spektret  clear; close all; clc;  fs = 30000; % samplingsfrekvens  f0 = 4000; % en tone inde i båndet (kun ét symbol)  Ts\_list = [0.50 0.20 0.10 0.05 0.02]; % symboltider => forskellige N  colors = lines(numel(Ts\_list));  figure; hold on;  for k = 1:numel(Ts\_list)  Ts = Ts\_list(k);  N = round(Ts\*fs); % vindueslængde (antal samples)  n = 0:N-1;  x = cos(2\*pi\*f0\*n/fs); % ren sinus (rektangulært vindue)  % FFT (zero-padding for pæn kurve; opløsningen bestemmes stadig af N)  Nfft = 2^nextpow2(max(4096, N));  X = fft(x, Nfft);  f = (0:Nfft-1)\*(fs/Nfft);  % Amplitude (normaliser peak til 0 dB for sammenligning)  A = abs(X);  A = A / max(A);  AdB = 20\*log10(A + 1e-12);  plot(f(1:Nfft/2), AdB(1:Nfft/2), 'LineWidth', 1.2, 'Color', colors(k,:));  % Teoretisk hovedlobebredde (rektangulært vindue): ca. 2\*fs/N (nul-til-nul)  dF\_mainlobe = 2\*fs/N;  text(f0+200 + 150\*k, -6-2\*k, sprintf('N=%d \\Delta f\_{HL}≈ %.1f Hz', N, dF\_mainlobe), ...  'Color', colors(k,:), 'FontSize', 9);  end  grid on; xlim([0 9000]); ylim([-80 3]);  xlabel('Frekvens [Hz]'); ylabel('Amplitude [dB]');  title('Amplitude-spektre for forskellige N (rektangulært vindue)');  legend(arrayfun(@(Ts)sprintf('Ts=%.02f s (N=%d)', Ts, round(Ts\*fs)), Ts\_list, 'uni',0), 'Location','southwest');  %% Vindue-tradeoff: sammenlign rektangel vs. Hann ved fast N  figure; hold on;  Ts = 0.10; N = round(Ts\*fs);  n = 0:N-1;  x\_rect = cos(2\*pi\*f0\*n/fs);  x\_hann = x\_rect .\* hann(N).';  Nfft = 2^nextpow2(max(4096, N));  f = (0:Nfft-1)\*(fs/Nfft);  Xr = fft(x\_rect, Nfft); Xh = fft(x\_hann, Nfft);  Ar = abs(Xr)/max(abs(Xr)); Ah = abs(Xh)/max(abs(Xh));  plot(f(1:Nfft/2), 20\*log10(Ar(1:Nfft/2)+1e-12), 'LineWidth',1.2);  plot(f(1:Nfft/2), 20\*log10(Ah(1:Nfft/2)+1e-12), 'LineWidth',1.2);  grid on; xlim([0 9000]); ylim([-100 3]);  xlabel('Frekvens [Hz]'); ylabel('Amplitude [dB]');  title(sprintf('Vindue-effekt ved N=%d (Ts=%.02f s), f\_0=%g Hz', N, Ts, f0));  legend('Rektangel (smal HL, høje sidelobes)','Hann (bredere HL, lavere sidelobes)', 'Location','southwest');  %% Lille tabel til rapporten  fprintf('\nTeoretisk hovedlobebredde (rektangulært vindue, nul-til-nul): Δf ≈ 2\*fs/N\n');  for Ts = Ts\_list  N = round(Ts\*fs);  fprintf('Ts=%.3f s, N=%5d -> Δf≈ %.1f Hz\n', Ts, N, 2\*fs/N);  end  %% Opgave 4C – SNR'ens betydning for systemets ydeevne  clear; close all; clc;  fs = 30000;  fstart = 9000;  fend = 10000; % Bruger smallere bånd  Ts = 0.01; % og kortere symboltid for at vise SNR betydning  msg = 'hello world hello world';  % Liste af SNR-værdier i dB  SNR\_dB = 0:5:60; % fra 0 dB (meget støj) til 60 dB (næsten perfekt)  FER = zeros(size(SNR\_dB));  BER = zeros(size(SNR\_dB));  % Original FSK-signal  x = FSKgenerator(msg, fstart, fend, Ts, fs);  Px = mean(x.^2); % signalets gennemsnitlige effekt  for k = 1:length(SNR\_dB)  snr\_val = SNR\_dB(k);    % Beregn nødvendig støj-effekt for den ønskede SNR  Pn = Px / (10^(snr\_val/10));  noise = sqrt(Pn) \* randn(size(x)); % hvidt støjsignal    % Tilføj støj  y = x + noise;  % Decode  yhat = FSKdecoder(y, fstart, fend, Ts, fs);    % Frame- og bitfejl  FER(k) = any(yhat ~= msg); % 1 hvis mindst én fejl  BER(k) = sum(yhat ~= msg) / length(msg);    fprintf('SNR = %2d dB | FER = %.2f | BER = %.3f\n', snr\_val, FER(k), BER(k));  end  % Plot  figure;  subplot(2,1,1);  semilogy(SNR\_dB, BER, 'o-','LineWidth',1.2);  xlabel('SNR [dB]'); ylabel('BER');  title('Bit Error Rate som funktion af SNR');  grid on; ylim([1e-4 1]);  subplot(2,1,2);  stem(SNR\_dB, FER,'filled');  xlabel('SNR [dB]'); ylabel('FER');  title('Frame Error Rate som funktion af SNR');  grid on; ylim([-0.05 1.05]); |