CASE 4

SONAR

|  |  |
| --- | --- |
| Navn: Jonathan H. Schougaard | Stdnr: 201407553 |
| Navn: Laimonas Ignas Bendikas | Stdnr: 201408723 |
| Navn: Simon Strandby Kristiansen | Stdnr: 201405845 |

Indholdsfortegnelse

[Opgave 1 – Signal generation 2](#_Toc450502272)

[Hvorfor ikke et sinus signal… ? 4](#_Toc450502273)

[Signal længde, præcision, måle afstand, afstand pr. sample m.m. 5](#_Toc450502274)

[Opgave 2 – Implementering af lyd-afspilning/-optagelse 6](#_Toc450502275)

[Løsning til problemstillingen 6](#_Toc450502276)

[Design af interpolation og decimation. 6](#_Toc450502277)

[Klargørelse af data til blackfin 7](#_Toc450502278)

[Skalere signal og h koefficienter. 7](#_Toc450502279)

[Gem data på Blackfin 8](#_Toc450502280)

[Blackfin implementering. 8](#_Toc450502281)

# Opgave 1 – Signal generation

I dette afsnit vil en gennemgang blive givet af de overvejelser der er blevet gjort angående signal generationen samt hvordan forskellige problem så som signalets længde, måle afstanden m.m. tænkes løst.

Der er valgt at generere et chirp signal fremfor f.eks. et sinus signal, forklaringen hvorfor fremkommer længere nede i et senere delafsnit.

Nedenfor ses kode afsnittet for generationen af koden samt forklarelse på hvad der sker.

%% Signal generation

%%% NOTE – Det oprindelige signal indeholder 3 chirps i SIGNAL 1, men for at gøre det mere overskueligt, er der valgt kun at medtage 2 chirps i SIGNAL 1 og analyse dette.

%Vælg hvilken type af signal der vil benyttes, sig=1 => 2 frekvenssweep lige efter hinanden.

%På test resultaterne ses signal 1 som "maale\_data..."

%sig = 2 => blot 1 frekvenssweep. På test resultaterne ses det som "maale\_data\_signal\_2..."

sig**=**1**;**

%Valg af frekvenser, f0 = start frekvens for frste sweep i signal 1, hvor

%f1 er start frekvens for det andet sweep i signal 1. Begge frekvens sweep

%slutter hhv. f0-500 og f1-500, dvs. ved 2500Hz og 700hz.

f0 **=** 3000**;**

f1 **=** 1200**;**

%Chip1 = 3000-2500Hz, Chirp 2 = 1200-700Hz.

%Hvis sig=2 => Chirp = 1200-600Hz.

**if(**sig**==**1**)**

T\_step**=**0.02**;**

t**=[**0**:**fs**\***T\_step**]/**fs**;**

Signal**=[**

chirp**(**t**(**1**:**round**(end/**4**)),**f0**,**0.01**/**4**,**f0**-**500**).\***blackman**(**length**(**t**(**1**:**round**(end/**4**)))).^**0.2**'**

chirp**(**t**(**1**:**round**(end/**4**)),**f1**,**0.01**/**4**,**f1**-**500**).\***blackman**(**length**(**t**(**1**:**round**(end/**4**)))).^**0.2**'];**

**elseif(**sig**==**2**)**

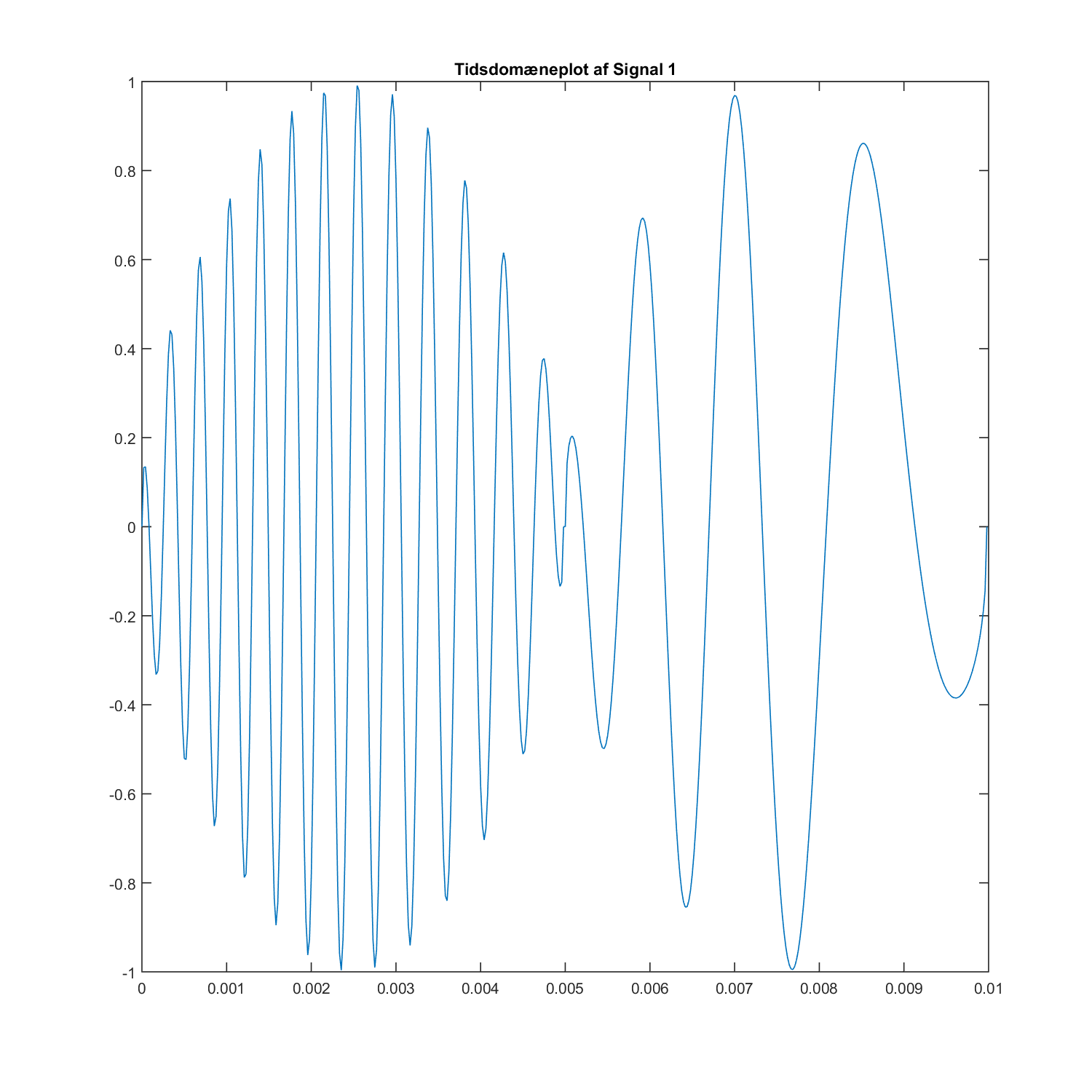
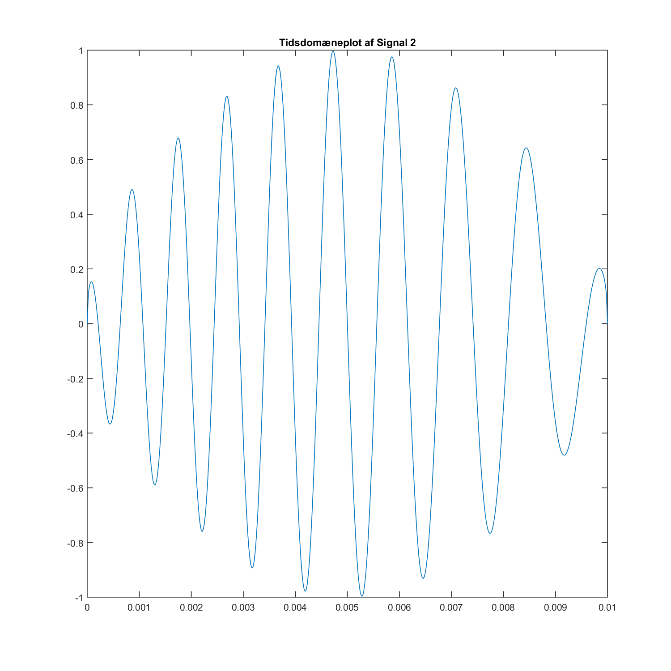
T\_step**=**0.01**;**

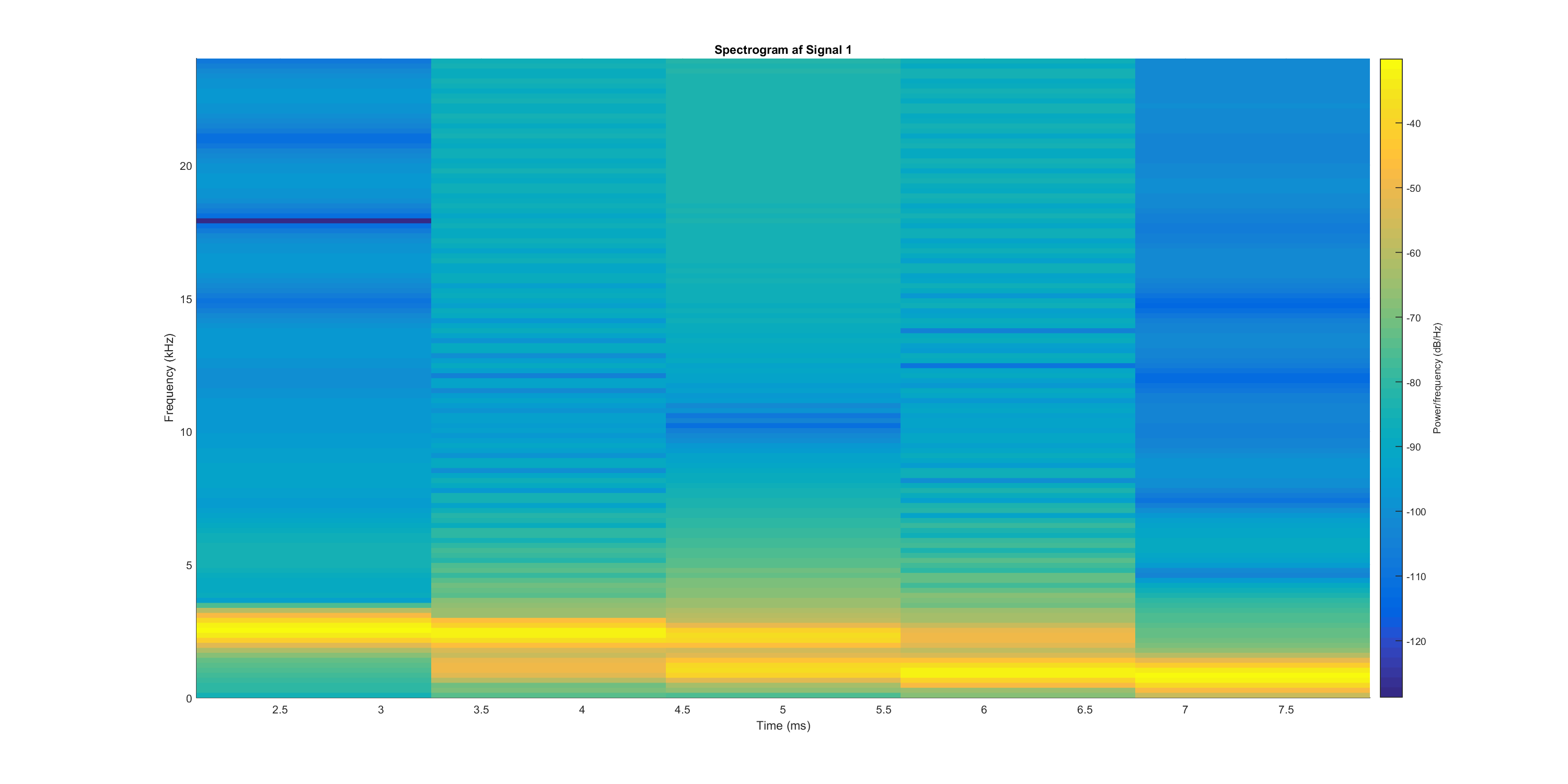
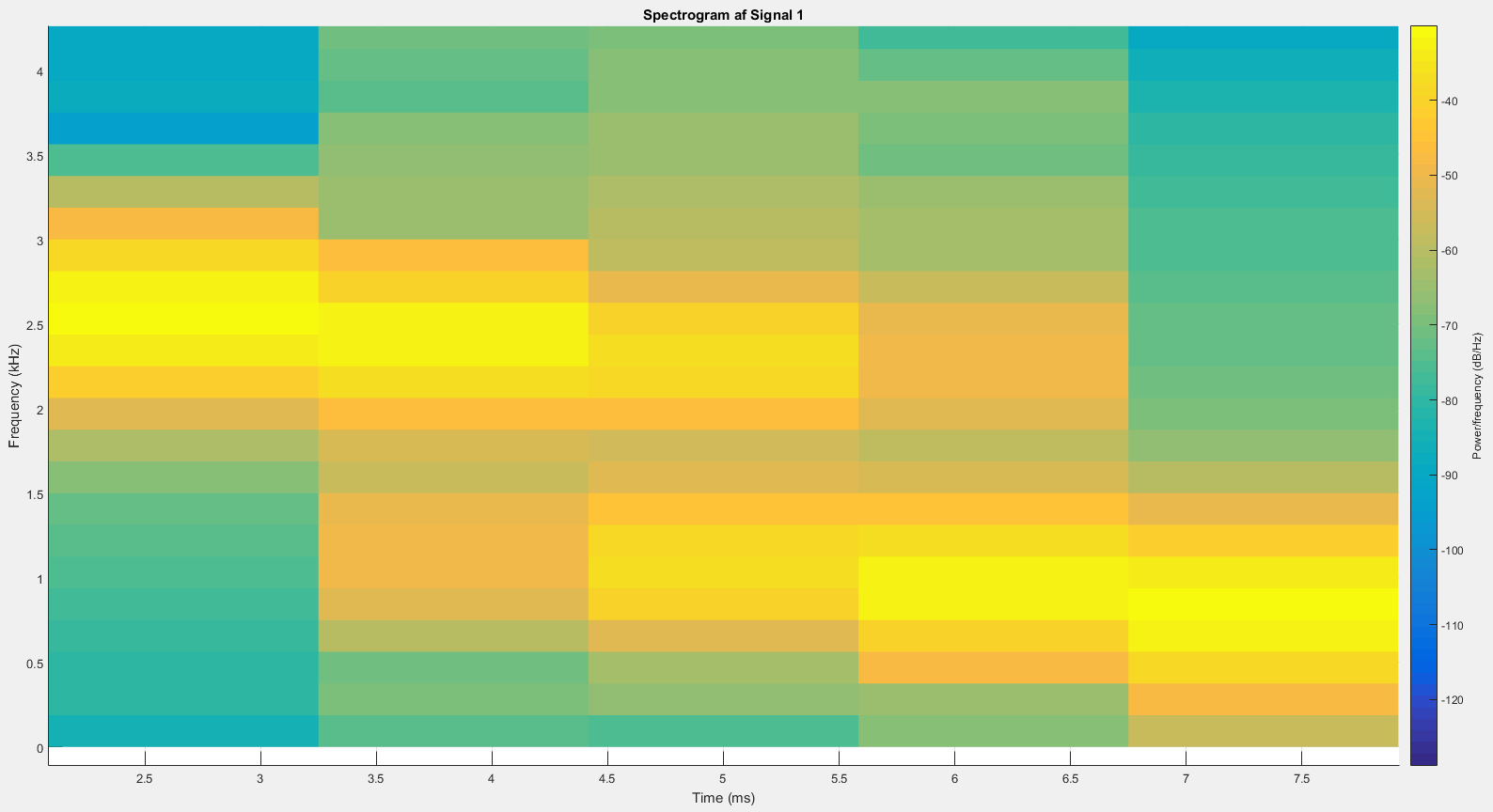
t**=[**0**:**fs**\***T\_step**]/**fs**;**

Signal **=** chirp**(**t**,**1200**,**T\_step**,**600**).\***blackman**(**length**(**t**)).^**0.2**';**

**end**

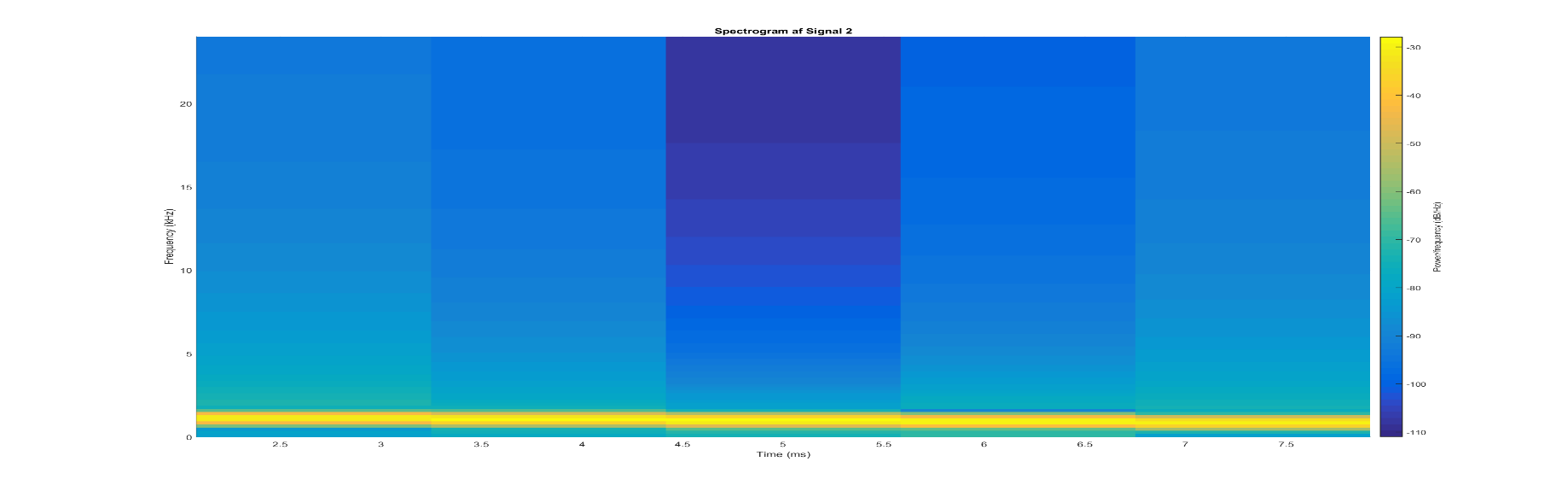
Forskellen på signalerne er som omtalt ovenfor, at det ene signal indeholder 2 chirps af forskellige start og slut frekvenser, mens det andet signal blot indeholder et frekvens sweep. Hertil plottes signalerne.

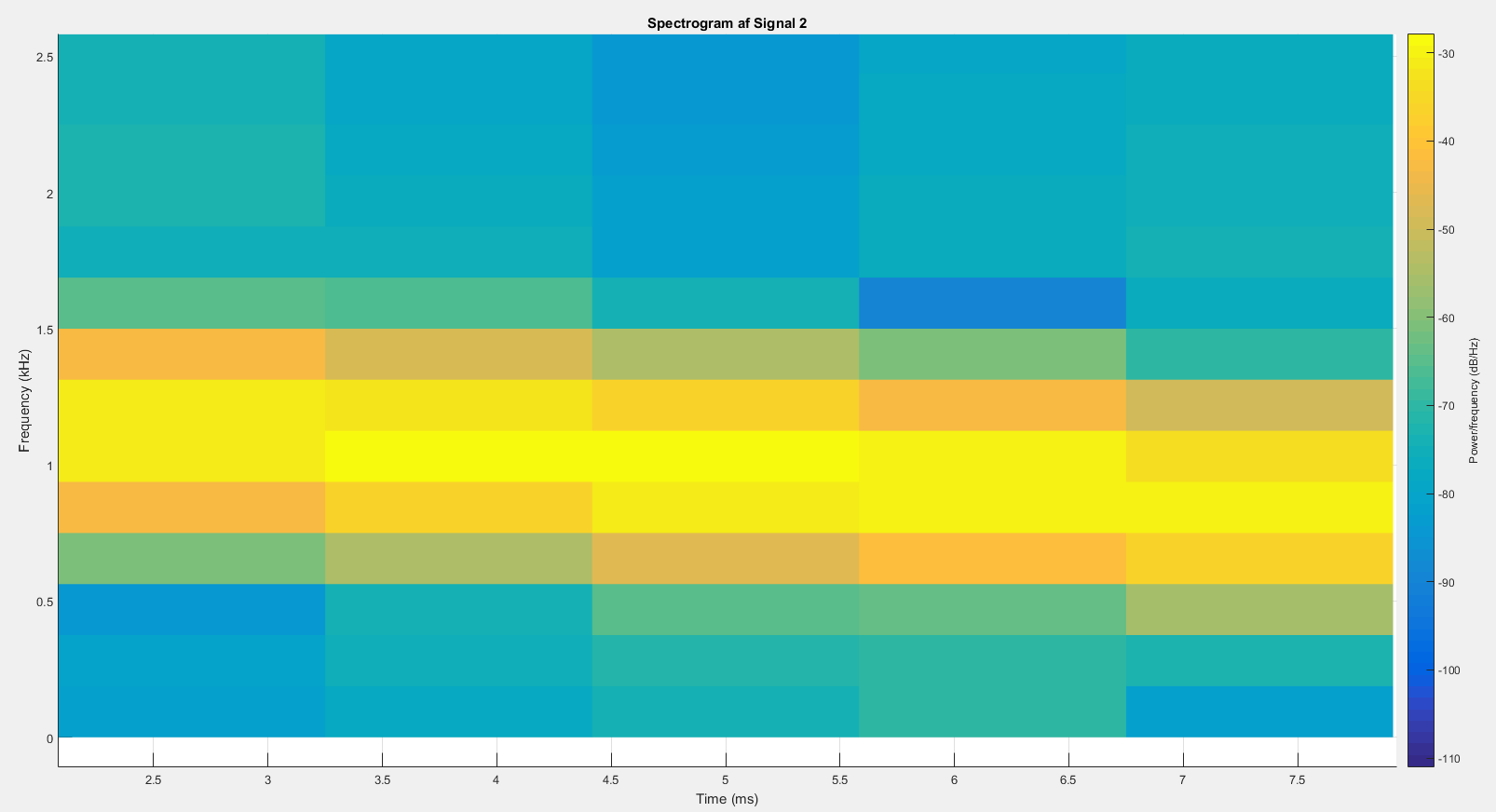


Til venstre ses tidsdomæneplottet af Signal 1, som indeholder 2 chirps, dette kommer også til udtryk, da man kan se, at signalet har to forskellige generelle udseender, hvor det tydeligt ses at der fremkommer et chirp med en høj frekvens i starten som så småt begynder at oscillere mindre, men også det chirp som startede ved en mindre frekvens. Her ses at oscillationen er langsommere og bliver langsommere da perioden vokser så småt. På billedet til højre ses tidsdomæneplottet af signal 2, og her ses det også, at man først starter ved en højere frekvens end slut frekvensen, da oscillationen aftager over den lille periode tid.

Her ses så spektrogramplottet for signal 1. Og det ses tydeligt, at der forekommer 2 chirps/frekvenssweeps hvor det ene starter omkring 3kHz og slutter ved ca 2,5khz mens det andet chirp starter ved ca. 1,5kHz og slutter ved ca 0,5 kHz hvilket fra øjenmål passer meget godt med de genererede signaler.

Nedenfor ses spektrogramplottene for signal 2, som kun indeholdte 1 chirp. Og her ser man også, at frekvensen ændrer sig over tid. Og øjenaflæste frekvenskarakteristikker stemmer også overens med det af det genererede signal. Se det er disse to signaler der vil blive testet på blackfin og omtalt i et senere afsnit.





## Hvorfor ikke et sinus signal… ?

Til dette formål kunne vi f.eks. ikke bruge et sinus signal som vores signal. Fordi i et sinus signal forekommer der uendeligt mange harmoniske og på den måde er der andre faktorer som kan spille ind ved et sinus signal i modsætning til f.eks. et chirp signal hvor man benytter forskellige frekvenser, og dermed er chancen for forstyrrelse fra andre harmoniske er nærmest ikke eksisterende.

Derudover hvis man krydskorrelerede på et sinus med det modtagne signal, så ville man blot få tykke streger i plottet af krydskorrelationen, fordi den måske ikke er færdig med at sende signalet ud, men man ville blive ved med at modtage en sinus, som jo ikke ændrer sig. Derfor ville man tro at man hele tiden får det samme ind som man sendte ud. Og dertil ville det være nærmest umuligt at aflæse afstanden ved brug af sinusser. Eftersom man ikke ville kunne se forskel på peaks’ne i efter korrelationen.

## Signal længde, præcision, måle afstand, afstand pr. sample m.m.

I princippet kan vi måle uendelig langt væk hvis vi havde et uendelig højt signal, så mængden af signalet ikke ville blive overdøvet af støj, harmoniske etc. Men der forekommer en anden begrænsning og det er systemet som skal optage signalet. Dets hukommelse er en begrænsning og den afhænger af sample frekvensen og længden af optagelsen. Dvs. hvis man f.eks. optog i 2 sekunder ved 40kHz ville man skulle gemme 80000 samples på et begrænset mængde plads på systemet. Og når blackfin processoren kun har 10-20kB på dens L1 cache til rådighed resulterer dette i op til højst 5000-10000 samples. Dette skyldes, at blackfin processoren arbejder med 16bit, og da 1 byte kun er 8 bit… Vil det sige, at 10-20kB divideres med 2, da man benytter 2 byte per sample. Så en måde at kunne optage mere på 2 sekunder ville være f.eks. at downsample til en sample frekvens på 10kHz, på den måde ville man plot få 20000 samples hvis man optog i 2 sekunder. Fint! Tænker man… Men eftersom vi arbejder på et presset system ville denne løsning ikke være tilstrækkelig. Vores signal skal jo også ligge på blackfin processoren og dette optager også noget plads, og vores signal vil fylde ca. 5kB. Så vi har kun **5-15kB** plads til rådighed, som svarer til **2500-7500** samples af et signal, dog skal man have i minde, at der også skal være plads til filtre og lign.

Til at løse dette problem har vi valgt først at benytte os af et signal på **48kHz** som er standard lydformat. Dertil har vi downsamplet 4 gange, så vores sample frekvens er på ca. **12kHz**. Det vil så gøre det muligt for os at optage op til ca. 7500/12000 => **0,625 sekunder**. Og i dette tilfælde er det godt, at vi nemlig ikke benytter os at et sinus signal. For hvis vi f.eks. havde et sinus signal som vi ville sende ud, og dertil optage i 0,5 sekunder så ville det betyde, at vi ville få det fænomen som er blevet omtalt tidligere, at plottet af krydskorrelationen ville indeholde mange peaks. Især når vores signal er **0,01 sekunder** i længden ville det resultere i at lyden ville kunne bevæge sig en længde af 0,01s\*340m/s = 3,4m. Og hvis vi måler i en afstand på 1 meter, ville sinussen have gentaget sig i det forløb og dertil skabe de mange peaks i krydsproduktet. Her skal det huskes, at det bliver reflekteret, så minimums afstanden er det halve. Se nedenfor, hvor beregningerne er foretaget.

%% Minimum, maximum afstand samt tollerance for signalet

%Her beregnes længden af signalet, den minimale afstand som signalet skal have for så der undgås overlap, den maksimale afstand ift. antal samples, og dertil tolerancen på afstanden.

T\_signal=length(Signal)/fs %Signal 1: 0.0100s %Signal 2: 0.0100s

afstand\_min=v\_sound\*T\_signal/2 %Signal 1: 1.7000m %Signal 2: 1.7035m

afstand\_max=v\_sound\*2500/((fs/M\_down)\*2) %Signal 1: 35.4167m %Signal 2: 35.4167m

For at snakke om tolerancen og dermed også hvor mange meter en sample svarer til så kan det udregnes fra lydens hastighed, og sample frekvensen:

afstand\_tol=v\_sound/((fs/M\_down)\*2) %Signal 1: 0.0142m %Signal 2: 0.0142m

Dvs. begge vores signaler, har en længde af 1ms, man skal holde en minimum afstand på 1.7m for så at forvente et fornuftigt krydskorrelationsprodukt. Dertil en maks afstand af ca 35meter. Vores tolerance med disse mål er ca. 1,42cm per sample, hvilket er forholdsvist nøjagtigt nok til vores formål. Så vi kan måle helt ned til 1,42cm nøjagtighed på de forskellige afstande vi vil foretage mål over.  
I næste afsnit omtales hvor decimations og interpolations filtrene er opbbygede samt hvordan disse er implementeret på blackfin processoren.

# Opgave 2 – Implementering af lyd-afspilning/-optagelse

Opgaven lyder simple, hoved princippet er det bare et array af data der skal afspilles ud på højtaleren sammentid med at inputtet fra mikrofonen bliver lagt ned i et andet array. Problemet kommer dog fordi der køres med sample frekvens på 48kS/s og med en L1 cache på omkring 10-20 kB, dette resultere i at der er ikke særlig meget hukommelse at arbejde rundt på.

Der er også et krav om at signalet skal ligge i det hørebare område cirka 50-1600Hz.

Problemer og krav

1. Signal skal være i det hørebare område 50-1600Hz.
2. Sampler med 48kS/s, og har kun 10-20 kB L1 cache. Hver sample fylder 2 byte.

## Løsning til problemstillingen

Efter som det er i det hørebare område der skal laves et signal i og opfanges et signal. Til dette vil der blive brugt up-/down-sampling, hvor der menes interpolation og decimation. På denne måde fylder signalet ikke meget, og det samme gælder det når signalet bliver optaget igen, dette stiller dog også nogle højere krav til udregning af afstand, da der skal tages højde for group delay, der skal selvfølgelig heller ikke ske signal forvrængning derfor bliver der brugt FIR filter, da den har den egenskab at have lineær fase, som betyder at alle frekvenser bliver forsinket med den samme tid (= group delay). Der skal også laves et program der automatisk kan lave vores signal fra Matlab over til Blackfin.

## D:\Cludes\Google Drev\uni\semester 4\DSA\Cases E4DSA\Case4_Sonar\Blackfin_design.pngDesign af interpolation og decimation.

Der bruges decimation i stedet for bare at down-sample, efter som der vil ske aliasering problemer da alle frekvenser der er over Nyquist frekvens vil danne aliasering. Dette kan løses med et digital filter før downsampling. Ved upsampling bruges der bare et midlingsfilter for at udjævne trappesignalet der bliver dannet, eftersom det skaber masser af højfrekvente signaler.

Til at designe decimation filter, bruges matlabs hjælpe funktion fir1.

h**=**fir1**(**50**,(**1**/**M\_down**));**

Der bliver lavet et 50 ordens fir filter, som knækker i den halve downsample frekvens.

## Klargørelse af data til blackfin

Der bliver brugt en lidt lang omvej når det kommer til at få dataet ned på Blackfin i så fald når det kommer til signalet. Dette skyldes til dels fordi matlab koden er lavet udfra at det kendte signal bliver lavet oprindeligt med den samme sample frekvens som Blackfin bruger. Hvorefter signalet bliver downsamplet, som er det signal der bliver gemt på Blackfin og resamplet til den downsamplet frekvens, dette er det kendte signal. Dette gøres for at simulere det signalet kommer igennem når det kommer igennem interpolation og decimation på blackfin, og resultere i at der ikke længere skal tages højde for group delay når afstanden skal udregnes. Der er også en anden god grund til at det bliver gjort på denne måde, et fordi der er ikke fastlagt hvor meget systemet skal up-/down-sample, så det gøre det nemmere at teste forskellige løsninger uden at der skal laves et nyt signal udfra en anden sample frekvens.

%% Downsample signalet før det kommer over på blackfin.

% først kommer signalet igennem filteret.

Signal\_deci=conv(h,Signal);

% hvorefter det bliver downsamplet. Det er også dette data blackfin får.

Signal\_down=downsample(Signal\_deci,M\_down);

% udregning af den nye frekvens.

fs\_down=fs/M\_down

% vi resampler vores signal tilbage igen, så vi har en ide på hvordan

% signalet kommer til at se ud efter at være kommet igennem begge filter.

Signal\_resample=conv(conv(upsample(Signal\_down,M\_down),ones(1,M\_down)\*M\_down),ones(1,M\_down));

% Til sidst downsample signalet så det kommer til at passe sammen med det

% der bliver samplet på blackfin. Dette er også vores kendte signal, og er

% det signal der vil blive ledt efter når der skal laves en afstand måling.

Signal\_kendt=downsample(conv(Signal\_resample,h),M\_down);

save(signal\_navn,'Signal\_kendt', 'fs\_down', 'h');

### Skalere signal og h koefficienter.

Det data der skal ned på Blackfin skal først skaleres før det kan laves om til short (int16). Signalet bliver skaleret ved at finde den største amplitude og normalisere signalet og bagefter multiplicere det med 2^15-1. ”h” koefficienterne er lidt anderledes, igen bliver den største værdi fundet, hvorefter der bliver udregnet hvor meget h koefficienterne kan shiftes op før der sker overflow.

%% gøre data klar til blackfin.

% finder den største værdi i signalet, så signalet kan blive skaleret bedst

max\_signal=max(abs(Signal\_kendt));

% skalere signalet og laver det om til short (int16).

blackfin\_data=int16((Signal\_kendt/max\_signal)\*(2^(16-1)-1));

plot([0:length(blackfin\_data)-1]/fs\_down,blackfin\_data)

soundsc(double(blackfin\_data),fs\_down)

% shifter h koefficienterne, så meget som muligt for at give så præcist

% resultat som muligt.

bit=16;

max\_h=max(abs(h))

shift\_max=floor((1/max\_h)/2)+(bit-1)

blackfin\_filter=int16(h\*2^(shift\_max))

### Gem data på Blackfin

Der bliver brugt et script til at gemme dataet i en header fil, som gemmer information om hvor meget der skal upsamples og downsamples. Og lidt information om dataet, til sidst så indeholder filen signalet og h koefficienterne. For at få dataet ind programmet skal der bare bruges ” #include "blackfin\_signal.h" ”.

%% gemmer data i header fil

fileID=fopen('AudioNotchFilter/src/blackfin\_signal.h','w');

fprintf(fileID, '#define M %d\n', length(blackfin\_filter));

fprintf(fileID, '#define M\_SHIFT %d\n', shift\_max);

fprintf(fileID, '#define UP\_M %d\n', M\_down);

fprintf(fileID, '#define UP\_M\_SHIFT %d\n', round(log2(M\_down)));

fprintf(fileID,'#define SIGNAL\_SIZE %d\n',length(blackfin\_data));

fprintf(fileID,'short h[M] = {\n%d', blackfin\_filter(1));

for i=2:length(h)

fprintf(fileID,', %d', blackfin\_filter(i));

end

fprintf(fileID,'\n};\n');

fprintf(fileID,'short SIGNAL[SIGNAL\_SIZE] = {\n');

fprintf(fileID,'%d', blackfin\_data(1));

for i=2:length(blackfin\_data)

fprintf(fileID,', %d', blackfin\_data(i));

end

fprintf(fileID,'\n};\n');

fclose(fileID);

## Blackfin implementering.

Nu kan filterne implementeres på blackfin.

Decimation filteret har en circle buffer til indeholde sine tidligere x værdier, på denne måde spare Blackfin processeren masser af tid på at flytte rundt på værdier. Filteret bliver realiseret med ”for-løkke” hvor x værdierne bliver kørt igennem h koefficienter, til sidst før y værdien returneres bliver den shiftet ned eftersom h koefficienter var shiftet op.

short x\_**[**M**];**

unsigned char p\_ **=** 0**;**

short decimFilter**(**short x**)**

**{**

long y **=** 0**;**

x\_**[**p\_**]** **=** x**;**

**for(**unsigned char i **=** 0**;**i **<** M**;** i**++)**

**{**

y**+=(**int**)**h**[**i**]\*(**int**)**x\_**[(**i**+**p\_**)%**M**];**

**}**

p\_**=(**p\_**+**1**)%**M**;**

**return** **(**short**)(**y**>>**M\_SHIFT**);**

**}**

Interpolation filteret er stort set bygget op på samme måde som Decimation filteret, der er ikke nogle h koefficienter, eftersom der bliver brugt et midlingsfilter, og alle h koefficienter er ens, det gøre det muligt at summere alle værdierne og bagefter shifte ned med log2(UP\_M). UP\_M er hvor meget signalet bliver upsamplet.

short x2\_**[**UP\_M**];**

unsigned char p2\_ **=** 0**;**

short interpolFilter**(**short x**)**

**{**

int y **=** 0**;**

x2\_**[**p2\_**]** **=** x**;**

**for(**unsigned char i **=** 0**;**i **<** UP\_M**;** i**++)**

**{**

y**+=(**int**)**x2\_**[**i**];**

**}**

p2\_**=(**p2\_**+**1**)%**UP\_M**;**

**return** **(**short**)(**y**>>**UP\_M\_SHIFT**);**

**}**

For at gøre det nemmere at forklare hvordan koden fungere tages der udgangspunkt i at der bliver down-/up-samplet med 4. signalet der modtages kommer først igennem decimation filter, hvor hver 4 gang bliver værdien gemt i data bufferen. Dette sker sammentid med at højtaleren afspiller vores signal, hvor det downsamplet signal bliver upsamplet og ført igennem interpolation filteret og ud på højtaleren.

**case** IIR\_FILTER\_ACTIVE **:** // Button PF8 presse

xn **=** **(**short**)** **(**iChannel0LeftIn **>>** 16**);** // Keeping 16 bits

yn **=** decimFilter**(**xn**);**

**if(**data\_point **<** DATA\_SIZE**)**

**{**

**if(**point**%**UP\_M**==**0**)**

**{**

data**[**data\_point**]** **=** yn**;**

data\_point**++;**

**}**

yn **=** interpolFilter**(**SIGNAL**[(**data\_point **<** SIGNAL\_SIZE **?** data\_point **:** **(**SIGNAL\_SIZE**-**1**))]);**

iChannel0LeftOut **=** yn **<<** 16**;** // Convert to 24 bits

iChannel0RightOut **=** yn **<<** 16**;**

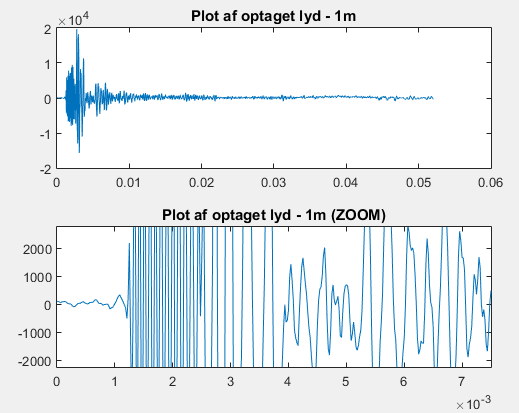
point**++;**

**}**

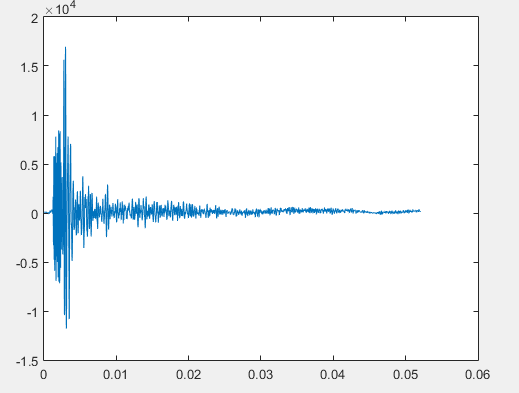
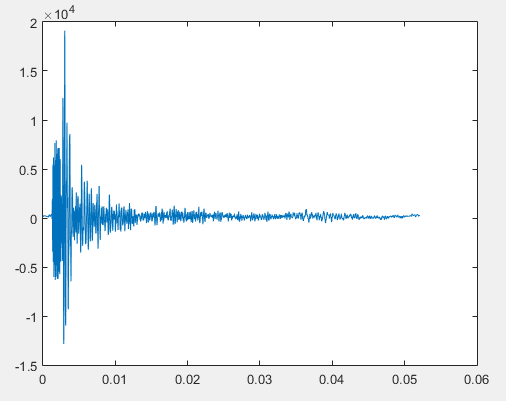
**break;**

# Opgave 3 & 4 – Eksperimenter; Signal analyse i Matlab

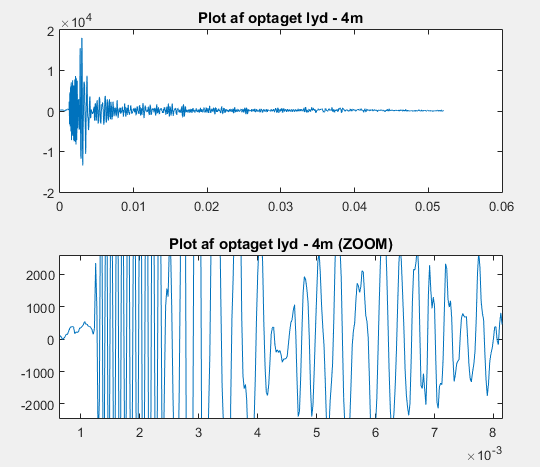
Der er først blevet lavet et par eksperimenter med opstillingen, hvor signalet er sendt ind mod en væk og optages igen med en mikrofon. Denne test er lavet på forskellige afstande med følgende resultater:



Ovenstående er et plot af den optagne lyd på 1m. afstand, fra **signal 1**. Det ses tydligt at dette er det 2 chirp signal der sendes, som også modtages. Vha. zoom kan det ses, hvordan intensiteten af frekvensen mindskes ved 2,5\*10-3 ca. Sammenlignes dette med plottet fra opgave 1, af dette signal, vil det kunne noteres, at signalet dykker når det skifter chirp, som ønsket.

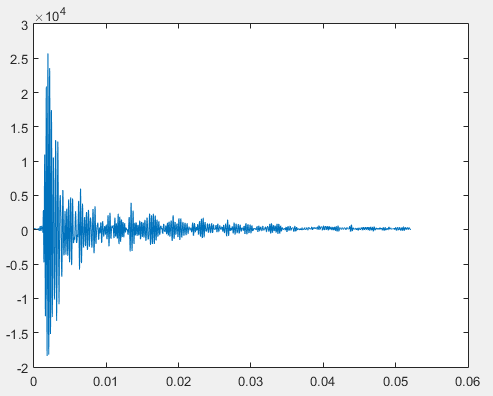
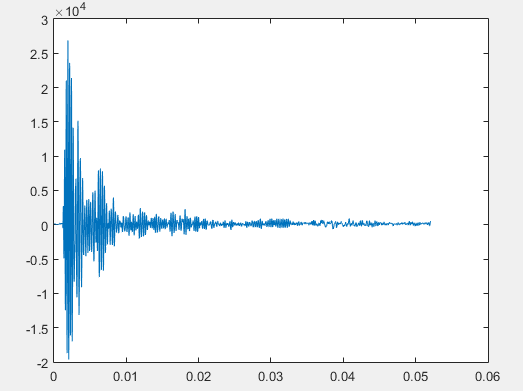


Plots af eksperimenter på 2m. og 3m., henholdsvis venstre og højre. (Signal 1 – 2 chirp)



Ovenstående er igen signal 1, men nu på 4m. Her ses en lille ændring, i forhold til 1m, på amplituden af peaken.

**Signal 2** er et enkelt chirp signal/frekvenssweep, hvilket resulterede i:



Signalerne har deres peak i starten, hvorefter de falder forholdsvist jævnt til ro. Der er en svag ændring af peaken når signalet, når afstanden når længere ud. (Vesntre 1m. afstand – Højre 3m. afstand)

## Krydskorrelation af signalerne

Input signalet og det målte anvendes nu til korrelationsprocessering. Vha. krydskorrelationen kan delayet findes på signalet, og afstanden beregnes til målet.

n=length(data1);

x1 = zeros(1,n);

x1(1:1, 1:length(in1)) = in1;

length(x1)

x2=data1;

xc=zeros(2\*n-1,1);

for i=1:2\*n-1

if(i>n)

j1=1;

k1=2\*n-i;

j2=i-n+1;

k2=n;

else

j1=n-i+1;

k1=n;

j2=1;

k2=i;

end

xc(i)=sum(conj(x1(j1:k1)).\*x2(j2:k2));

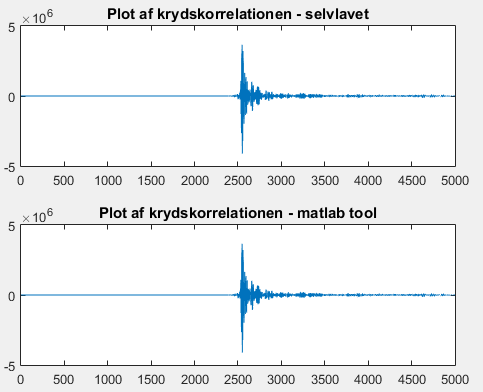
end

xc=flipud(xc);

Ideen ved krydskorrelation er, at finde lighederne mellem to signaler, eller bølger rettere. Denne lighed findes som funktion af en tidsforskydning på det ene signal, som også er til stede her.

Via. For-løkken, implementeret i kode stykket til højre, kunne hvert signals spektrum af værdier gennemgåes og foldes med modpartens. Herved blev krydskorrelationesproduktet opnået ved signal 1, som er bestående af 2 chirp signaler.

Det foldede signal for signal 1 og dets modpart ses her.



I plottet ses det tydeligt, at vores signal peaker ved 2550 samples, ca. Vha. matlab kan værdien findes til 2450.

[~,I] = max(abs(x))

Ud fra ovenstående kan afstanden beregnes, tolerancen på en sample er kendt til 1,42 cm.

**Præcisionen** af målingen kommer udelukkende an på antallet af samples der anvendes. Anvendes få samples til foldningen vil signalet blive begrænset og ”ufuldendt”.