

I3DSB - Mini projekt A

Sampling og analyse af digital musik- og talesignaler

SW - Gustav Nørgaard Knudsen - au612485



Introduktion

Formålet med dette miniprojekt er at give en forståelse for hvordan MATLAB håndterer digitale signaler såsom lyd og hvordan disse manipuleres. Lyd opbevares i matricer i MATLAB, som så kan plottes til deres tidsenhed, for at skabe plots af lydsignalet. Desuden er det også muligt at høre lyde, f.eks. med funktioner som soundsc(). Før jeg kan begynde på opgaverne, starter jeg med at indlæse mine .wav filer med audioread(). Jeg opretter en struct, der kan indeholde alt information om mine forskellige lydfile, samt får opdelt s2 op i left og right dele. Dette kan ses på listing 1. Desuden nulstiller jeg alle tidligere defineret variabler med med clear og clc, for ikke at få overlap fra tidligere.

Listing 1: Setup af signalerne i MATLAB som structs

```
1 clear ;
2 clc ;
3
4 % Define samples
5 s1 = 1;
6 s2left = 2;
7 s2right = 3;
8 s3 = 4;
9
10 % Loading files into arrays of samples
11 [y(s1).sample, ~] = audioread('Signal_s1.wav');
12 [y(s2left).sample, ~] = audioread('Signal_s2.wav');
13 [y(s3).sample, Fs] = audioread('Signal_s3.wav');
14
15 % Splits signal 2 into its own channels
16 y(s2right).sample = y(s2left).sample(:,2);
17 y(s2left).sample = y(s2left).sample(:,1);
18
19 % Name samples
20 y(s1).name = "Signal\_s1.wav";
21 y(s2left).name = "Signal\_s2.wav - channel 1";
22 y(s2right).name = "Signal\_s2.wav - channel 2";
23 y(s3).name = "Signal\_s3.wav";
```

For at gøre nogle af de kommende øvelser nemmere at skrive og forstå, er har jeg oprettet et for-loop, der kan løbe igennem alle structs, og oprette de variabler og matricer der kan blive nødvendige.

Listing 2: Oprettelse af varaibler til alle structs

```
1 % Predefine variables for s1, s2left, s2right, s3 and s1reS
2 for i = 1:length(y)
3     y(i).sample = y(i).sample';
4     % number of Samples 1, 2, 3
5     y(i).nS = length(y(i).sample);
6     %Calculating x axis for signals
7     y(i).time = [0:y(i).nS-1]*(1/Fs);
8     %Find max, min, mean, rms & effect
9     y(i).max = max(y(i).sample);
10    y(i).min = min(y(i).sample);
11    y(i).mean = mean(y(i).sample);
12    y(i).rms = rms(y(i).sample);
```

```

13     y(i).effect = sum(y(i).sample.^2);
14     %Calculating Crest value
15     y(i).crest = 20*log10(y(i).max/y(i).rms);
16 end

```

Opgave 1

For at finde antallet af samples for alle 3 signaler, kan jeg ganske simpelt kalde funktionen **length()** på alle samples. **length()** returnerer antallet af elementer i matricen, hvilket dermed vil sige antallet af samples. Koden for at gøre dette kan ses i listing 3

Listing 3: Print af antal samples

```

1 length(y(s1).sample) % = 4213759
2 length(y(s2left).sample) % = 8753617
3 length(y(s2right).sample) % = 8753617
4 length(y(s3).sample) % = 1270957

```

Her fra kommer jeg frem til at s1 indeholder 4.213.759 samples, s2 (left og righth) indeholder 8.753.617 samples og s3 indeholder 1.270.957 samples. Dette lade til at være rigtigt, da jeg kan se s2 f.eks. varer meget længere end s3.

Opgave 2

For at plotte alle 3 signaler med rigtige labels og titler, bruges funktionerne beskrevet i 4. For at gøre MATLAB koden mere overskuelig, er der brugt et for-loop på min struct, så jeg kan loope igennem hvert signal, og plottet dem med de allerede forudbestemte varaibler. Alle plots tegnes på figur f1, for overskuelighed.

Listing 4: Plot af s1, s2 og s3

```

1 %% Exercise 2
2 f1 = figure;
3
4 for i = 1:length(y)
5     subplot(2,2,i)
6     plot(y(i).time, y(i).sample)
7     title(y(i).name)
8     xlabel("Time(s)")
9     ylabel("Amplitude(~)")
10 end

```

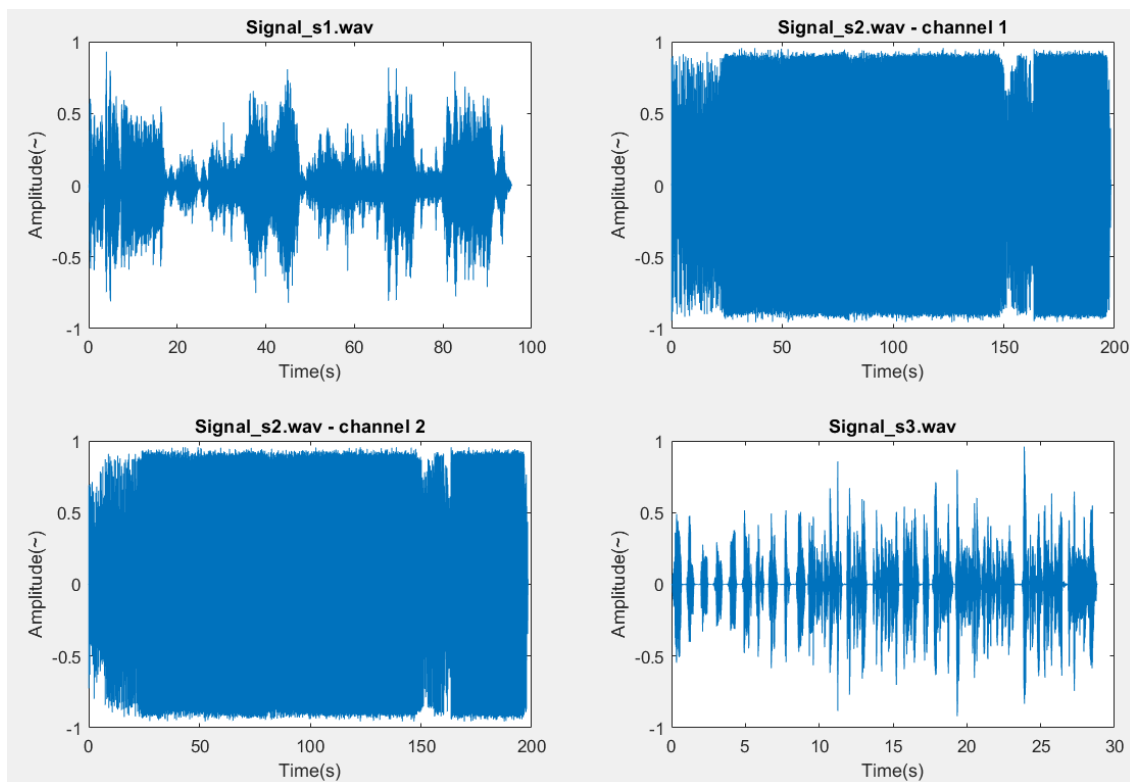


Figure 1: Plot af alle 3 signaler

Opgave 3

For at finde max, min, gennemsnit (mean), rms og effekt for alle 3 signaler, kan jeg kalde de forudbestemte værdier, jeg beregnede i listing 2. Et udsnit af hvordan de blev beregnet kan ses på listing 5.

Listing 5: Udsnit af beregning af max, min, gennemsnit (mean), rms og effekt

```

1 for i = 1:length(y)
2   y(i).max = max(y(i).sample);
3   y(i).min = min(y(i).sample);
4   y(i).mean = mean(y(i).sample);
5   y(i).rms = rms(y(i).sample);
6   y(i).effect = sum(y(i).sample.^2);
7 end

```

Table 1: Værdier for opgave 3

signal	max	min	mean	rms	effekt
s1	0.930	-0.820	≈ 0	0.119	60557.9
s2left	0.955	-0.955	≈ 0	0.338	1001787.6
s2right	0.955	-0.955	≈ 0	0.354	1102211.3
s3	0.960	-0.919	≈ 0	0.091	10738.9

Opgave 4

Ligesom i opgave 3, er crest allerede beregnet i listing 2. Til at beregne crest-faktoren, bruges formelen som beskrevet i ligning 1. Da der tages $20 \cdot \log_{10}$, kan det antages at crest-faktoren vil være i decibel. Det på listing 6 kan det også ses, hvordan crest-faktoren er beregnet i MATLAB.

$$20 \cdot \log_{10} \frac{x_{peak}}{x_{rms}} \quad (1)$$

Listing 6: Bergning af crest-faktor for alle 3 signaler

```

1 for i = 1:length(y)
2     y(i).crest = 20*log10(y(i).max/y(i).rms);
3 end

```

Table 2: Værdier af crest-faktor

signal	crest
s1	17.80 dB
s2left	9.02 dB
s2right	8.60 dB
s3	20.37 dB

I tabel 2 ses beregningen for de 3 signalers crest-faktor. Her ses der, at både s1 og s2 har en cirka dobbelt så stor crest-faktor, som s3 (både right og left). Svaret på hvorfor, kan findes i udregningen for crest-faktoren, da foruden omregningen til dB, tages x_{peak}/x_{rms} . Alle signalernes peak ligger omkring 0.95. Den store forskel er deres rms-værdi, hvor s2 har om rms-værdi på cirka 0.33, mens s1 og s3 begge ligger på en rms-værdi lige omkring 0.1. Herved har vi at x_{peak} divideres med et større tal for s2, end for s1 og s3. At s2 har en lavere crest-faktor giver meget god mening, når crest-faktoren beskriver hvor ekstrem forskellen er mellem peaks og rms af signalet. Da s2 er elektronisk musik, med masser af "gang i den", må vi dermed også kunne forvente en høj rms. Omvendt er der ikke lige så meget gang i den, i et signal med Mozart musik, eller tale.

Opgave 5 og 6

Jeg vil nu forsøge at nedsample mit signal s1 med en faktor på 4. Dette gøres med funktionen `decimate`, der ganske simpel fjerner hvert 4 sample, fra signalet s1. Her er jeg selvfølgelig opmærksom på, der derfor også er en ny afstand mellem hvert sample, så jeg bliver nødt til at beregne tiden igen, med $1/4$ sample rate. Dette kan ses her under på listing 7

Listing 7: Nedsampel af s1

```

1 s1reS = 5;
2 y(s1reS).sample = decimate(y(s1).sample, 4);
3 y(s1reS).time = [0:y(s1reS).nS-1]*(1/(Fs/4));
4
5 %% exercise 5
6 figure
7 subplot(1,2,1)
8 plot(y(s1).time(1:200), y(s1).sample(1:200))
9 title(y(s1).name)
10 xlabel("Time(s)")
11 ylabel("Amplitude(~)")
12 subplot(1,2,2)
13 plot(y(s1reS).time(1:200/4), y(s1reS).sample(1:200/4))
14 title(y(s1reS).name)
15 xlabel("Time(s)")
16 ylabel("Amplitude(~)")
17
18 % Listen to exercise 5
19 %soundsc(y(s1).sample, Fs);
20 %soundsc(y(s1reS).sample, Fs/4);

```

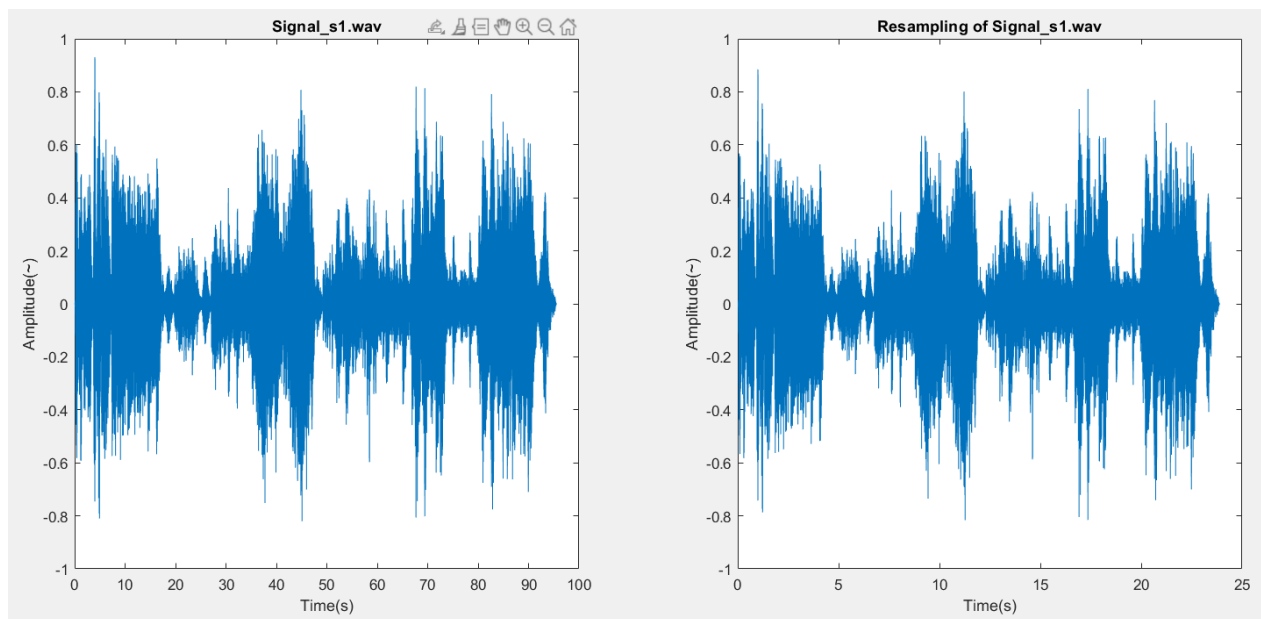


Figure 2: Resampling af signal s1

På plottet i figur 2 ses det ikke særlig tydeligt hvad, forskellen egentlig er belvet på de 2 signaler. Der er derfor taget et billede, hvor antallet af samples er taget meget længere ned, som det kan ses på figur 3. Her ses det mere tydeligt hvad decimate() har gjort. Da hvert fjerde sample er væk, er der tydeligt mistet en masse detaljer i signalet. F.eks. kan det ses at spændet af amplituder nu er -3.5 - 1.5, hvor der før var -6 - 2.

Lyttes der til signalet, hører man forskellen tydeligt. Det nye signal lyder meget komprimeret, som om det er en dårlig .mp3 fil, der afspilles på en Nokia 3210.

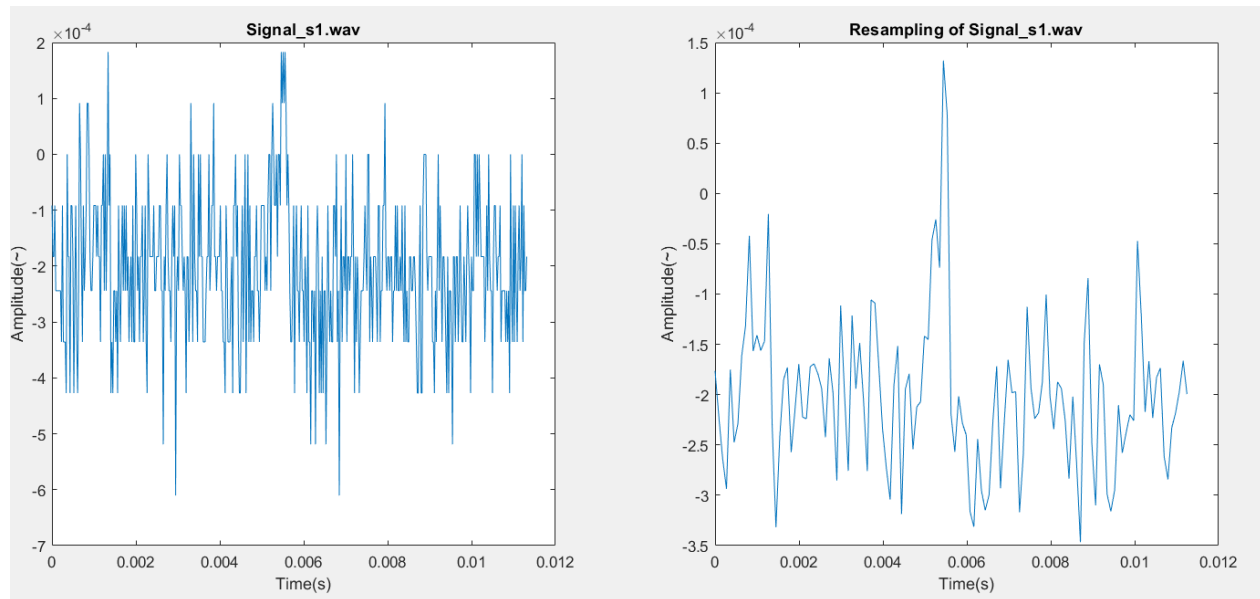


Figure 3: Zoom af resampling

Opgave 7

Til at sammenligne de channels på s2, er der først plottet begge channels ved siden af hinanden på figur 4. Da dette ikke har givet nogen stor information, er der derfor lavet et plot af en korter tidsperiode, et tilfældigt sted i signalet. Her ses en lille forskel, hvor amplituden er lidt større generelt på channel 2, men ikke noget tydeligt. Lyttes der til de 2 signaler, kan det være svært at spotte forskellen, da de egentlig er meget ens. Eneste forskel jeg har været i stand til at finde er, at der på channel 1 (left) kan findes guitar riffet, mens der på channel 2 (right) kan findes baselinen.

Listing 8: Kode for plot af s2left og s2 right

```
1 figure
2 subplot(1,2,1)
3 plot(y(s2left).time, y(s2left).sample)
4 title(y(s2left).name)
5 xlabel("Time(s)")
6 ylabel("Amplitude(~)")
7 subplot(1,2,2)
8 plot(y(s2right).time, y(s2right).sample)
9 title(y(s2right).name)
10 xlabel("Time(s)")
11 ylabel("Amplitude(~)")
```

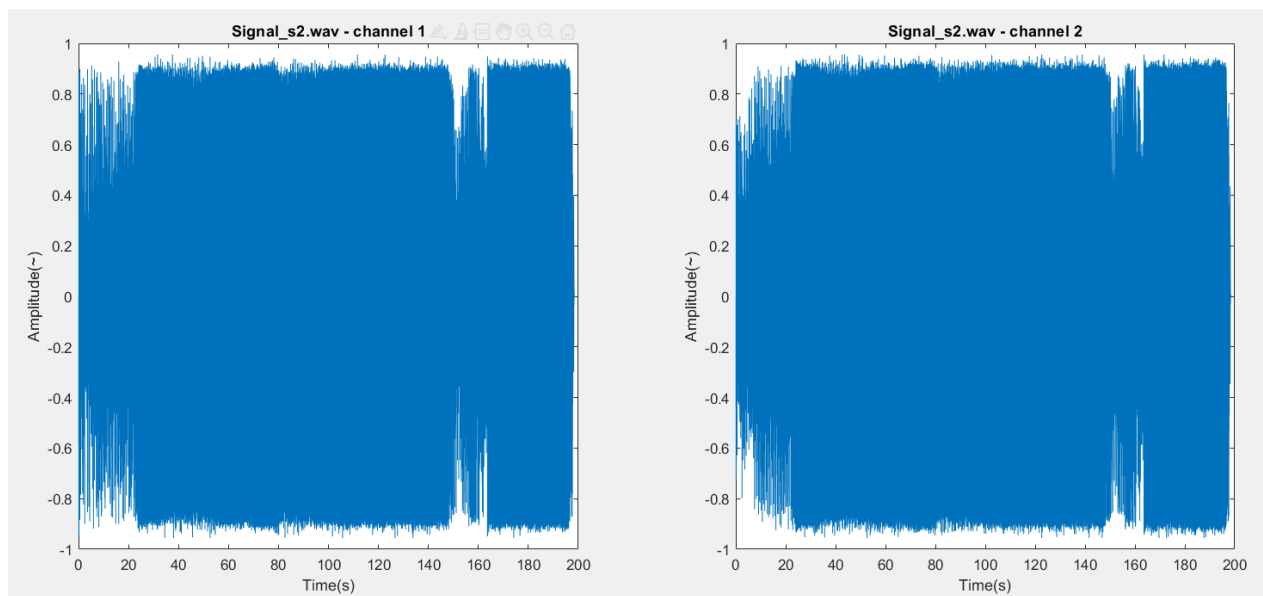


Figure 4: Plot af begge channels for s2

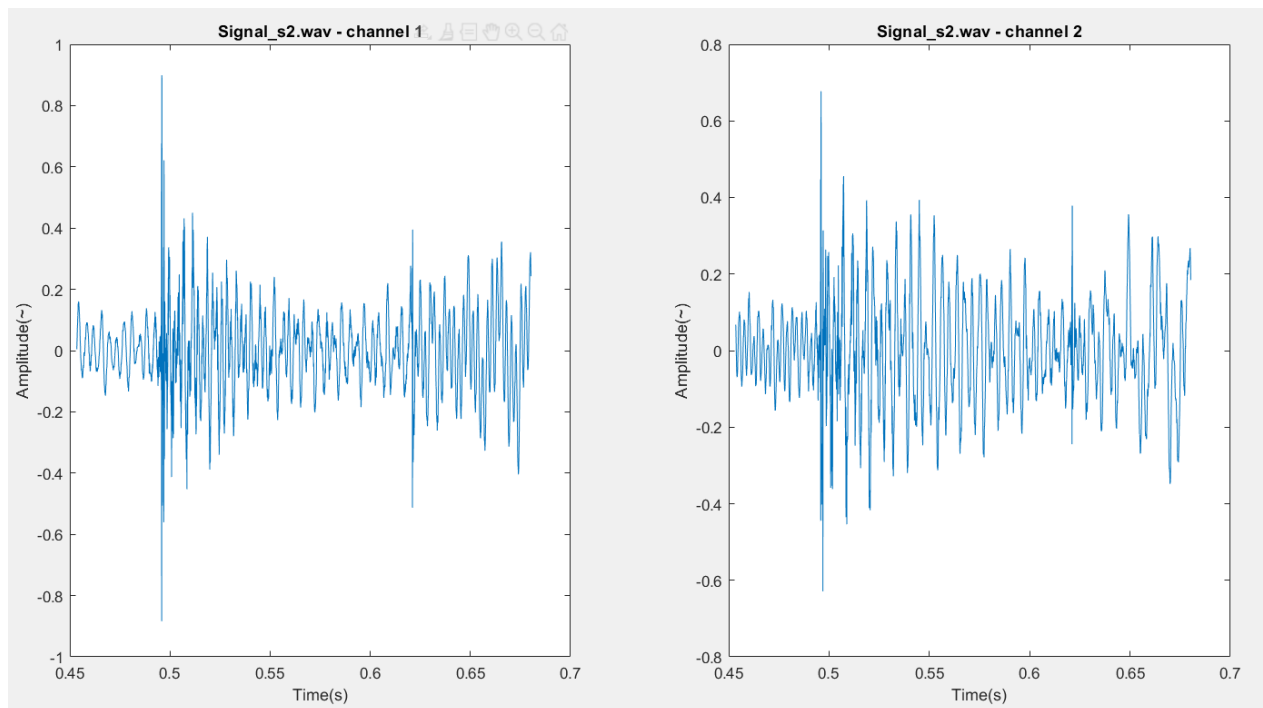


Figure 5: Plot af begge channels for s2, Zoom

Opgave 8

Til at kvantifiseret channel 1 (left), er der brugt den givne funktion `quantaziseN()` fra Blackboard. Når der kvantifiseres, fjernes detaljerne mellem amplituderne, så hvert sample af amplituden, kan repræsenteres med et færrer antal bits. Her er valgt at kvantifisere `s2` med 4 bits. Koden til dette kan ses i listing 9

Listing 9: Kvantifisering af `s2`

```
1 %% Exercise 8
2 %Using BB function to quantize to 4 bits
3 s2quan = 6;
4 y(s2quan).sample = quantizeN(y(s2left).sample, 4);
5 soundsc(y(s2right).sample, Fs) % uncomment for sound
6 figure
7 subplot(1,2,1)
8 plot(y(s2left).time(200000:205000), y(s2quan).sample(200000:205000))
9 xlabel("Time(s)")
10 ylabel("Amplitude(~)")
11 title("s2 - Left channel");
12 subplot(1,2,2)
13 plot(y(s2left).time(200000:205000), y(s2left).sample(200000:205000))
14 xlabel("Time(s)")
15 ylabel("Amplitude(~)")
16 title("s2 kvantifiseret 4bit - Left Channel");
```

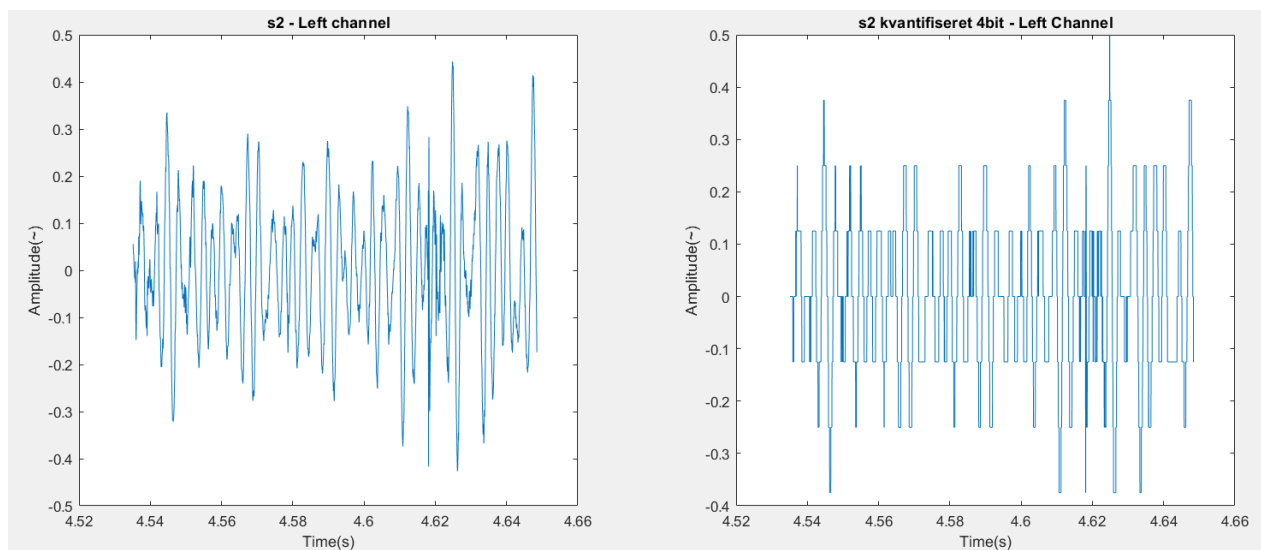


Figure 6: `s2` kvantifiseret

På figur 6, som er et lille udsnit af både `s2` og `s2 kvantifiseret`, ses tydeligt hvad `quantaziseN` har gjort ved signalet. Hvor amplituden i `s2 left` normalt repræsenteres med et så stort antal bits, det kanp er muligt at se steps imellem, kan der på det nye signal tydeligt ses de individuelle steps. Dette skyldes der samples med 4 bit, hvilket kun giver 16 mulige værdier for amplituden. Lytter man til signalt, kan der også høres hvordan

kvaliteten af musikken er faldet tydeligt. Mest tydeligt er det for instrumenterne, der har mistet en stor del af deltagerne, og nærmest cutter ud jævnlgt. Modsat er sangerens stemme stadig forholdsvis tydeligt. Musikken lyder meget som om det er ventemusik, der afspilles når man venter på at komme igennem til f.eks. telefon kundeservice. Dette kunne være et hint til hvordan telefonlyd komprimeres.

Opgave 9

For at få en eksponentielt aftagning på s3, kræver det at s3 ganges med en "envelope" der kan fungere som der ønskes. Her at der på den sidste 1/3 af s3, skal aftages eksponentielt, ned til en amplitude på 5% af den originale amplitude. Dette kan opnås med funktionen for en eksponentielt aftagende funktion, da vi kender vores skæring med y-aksen, og 2 punkter på grafen, skæringen med y-aksen i (0,1) samt slutpunktet, der skal være ($\frac{1}{3}time, 0.05$). Funktionen kan ses her under i ligning 2.

$$y(x) = b \cdot a^x \quad (2)$$

$$b = \text{y-værdi for skæring med y-aksen} \quad (3)$$

$$a = \left(\frac{y_2}{y_1} \right)^{\frac{1}{x_2 - x_1}}, \text{ hvor } (x_1, y_1) \text{ og } (x_2, y_2) \text{ er punkter på grafen} \quad (4)$$

Listing 10: Fade out af s3label

```
1 %% Exercise 9
2 % Tage sidste tredjedel af s3
3 t = y(s3).time(1:y(s3).nS/3+1);
4 % Eksponentielt aftagende: b*a^x
5 % b = Skæring med y i (0,1)
6 % a = (y2/y1)^1/(x2-x1)
7 a = (0.05/1)^(1/(t(end)-t(1)));
8 % Matrice der er eksponentielt aftagenende
9 % Starter i 1 og går til 0.05 ved 1/3 af time
10 e = 1*a.^t;
11
12 % s3fade
13 s3fade = 7;
14 y(s3fade).sample = y(s3).sample;
15 y(s3fade).time = y(s3).time;
16 % Påfør e på sidste 1/3 af samples
17 y(s3fade).sample(length(y(s3fade).sample)*2/3:end)
18     = e.*y(s3fade).sample(length(y(s3fade).sample)*2/3:end);
19
20 figure
21 plot(y(s3fade).time, y(s3fade).sample, Fs)
22 %soundsc(y(s3fade).sample)
```

Som det kan ses på figur 7, er den ønskede effekt opnået. På den sidste 1/3 af signalet, bliver stemmen dæmpet eksponentielt. Ved at lytte til signalt kan det også høre, den ønskede effekt er opnået.

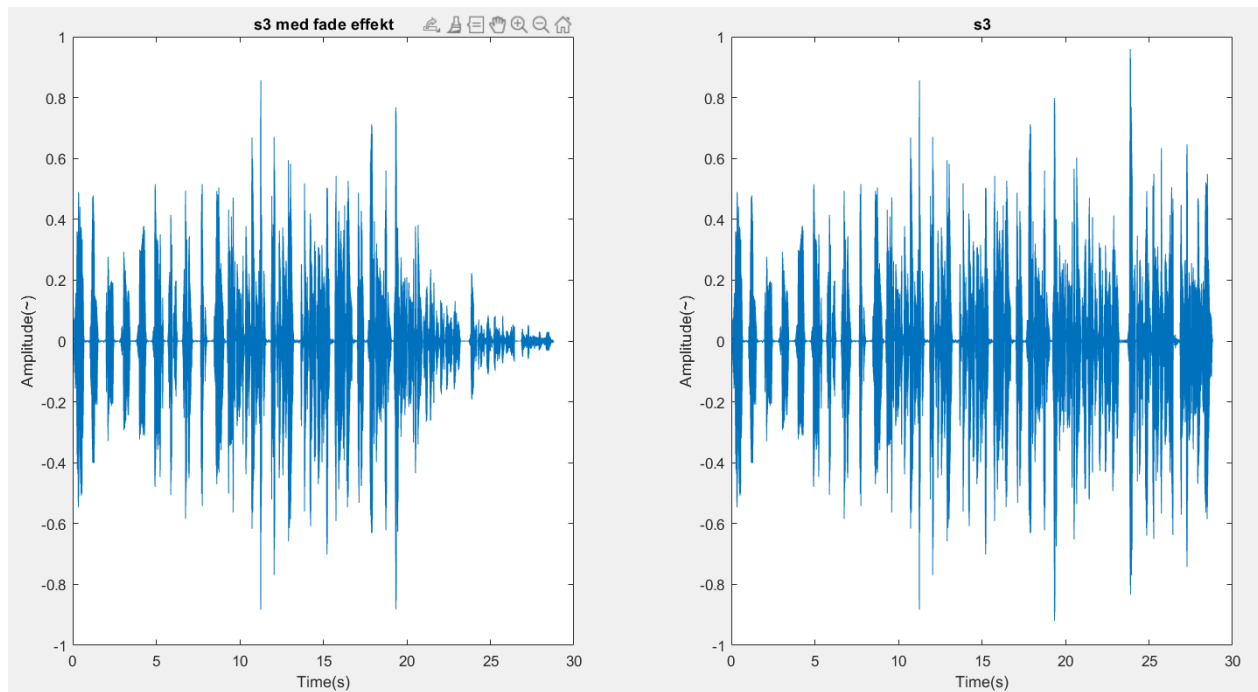


Figure 7: s3 med fade effekt

Opgave 1.15

Til denne opgave ønskes der at simulere et ”ekko” på et tone signal, hvor ekkoet har forskellige forskydning, til tonen. Først køres der med et ekko, der er 150ms forsinket, der efter et ekko med en forsinkelse på 40ms og 300ms. Tonen der er valgt her er 440Hz, kammertonen.

Listing 11: Kode til opgave 1.15

```

1 Fs = 5000; %Samplingsfrekvens
2 Ts = 1/Fs; % Tid per sampling
3
4 length = 1; % Længde i sekunder
5 Ns = Fs*length; % Antallet af samples
6 t = [0:Ns-1]*Ts; % Tiden over samplingen
7
8 %Tone beskrivelse
9 f0 = 2350; %440 Hz tone
10 A = 3; % Amplitude er 3
11 tone = A * sin(f0*2*pi*t); %Tonen som funktion
12
13 e.delay = 150; %ekko med 150 ms forsinkelse
14 e.Ns = e.delay * (Fs/1000); %Antallet af samples i ekkoet
15
16 tone_ekko = [zeros(1,e.Ns), tone]; %Forsinker ved at smide 0'er ind foran
17 tone_ext = [tone, zeros(1,e.Ns)]; % Forlænger ved at smide 0'er ind bagved

```

```

18
19 tone_sum = tone_ekko + tone_ext;
20 ekko_t = [0:Ns+e.Ns-1]*Ts;

```

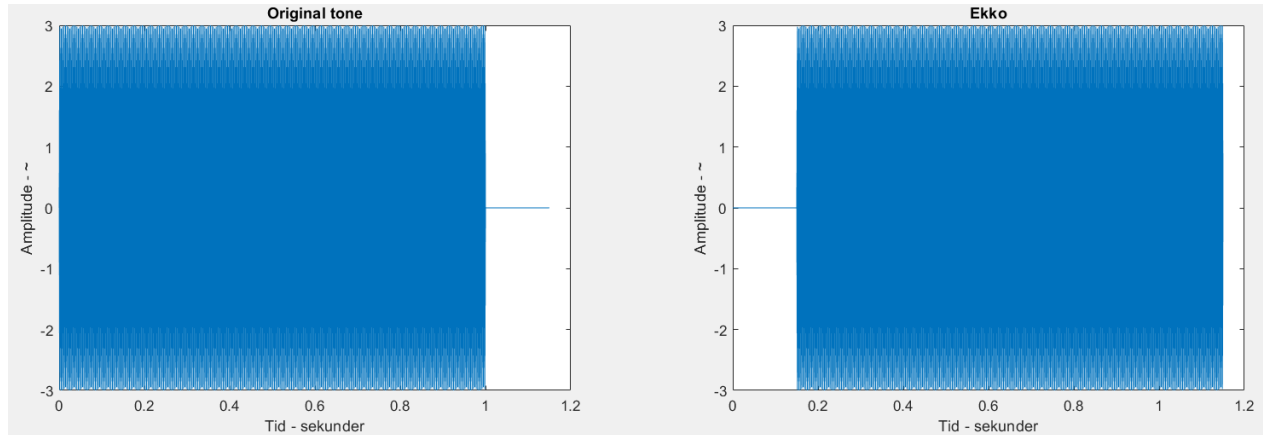


Figure 8: Plot af 440Hz tone med ekko

På figur 8 kan ses hvordan den originale tone ser ud, ved siden af sit ekko. Ekko effekten er lavet, ved at sætte et antal nuller foran det originale signal, for på den måde at "udskyde" hvornår signalet starter. Derfor er der også sat ekstra nuller bag på det originale signal, da de 2 matricer ellers ikke vil være lige store.

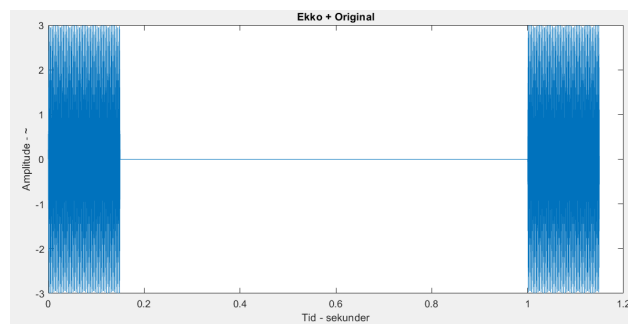


Figure 9: Plot af sum af 440Hz tone og Ekko på 150ms

På figur 9 ses hvad summen af 440Hz tonen, sammen med sit ekko ser ud. Vi kan se, der hvor de 2 toner overlapper, har de interfereret, og gået ud med hindanden. Dette skyldes forskydningen af ekko-signalet, der er blevet drejet 180 deg i forhold til det originale signal. Effekten høres tydeligt ved at afspille tonen, da der cuttes hurtigt ud, med et dødt stykke i midten, og en kort tone til sidst.

Forsøget er også blevet udført med 40ms delay og 300ms delay. I disse tilfælde har tonen ikke været forskudt, og original tonen og ekkoet har derfor i stedet forstærket hindanden. Denne effekt kan ses på figur 10. Her kan også ses at amplituden for interference signalet er fordoblet, til en amplitude på 6.

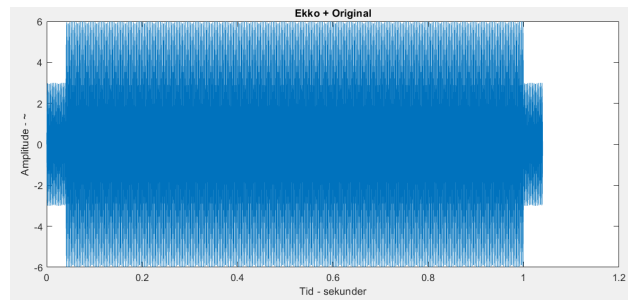


Figure 10: Plot af sum af 440Hz tone og Ekko på 40ms