I3DSB - Mini projekt 1

Digitale bølger

SW | Christian Bach Johansen au577526

1 Introduktion

I dette mini projekt vil jeg arbejde med sampling og analyse af digital lyd-signaler. Øvelsen vil generelt omhandle behandlingen af tre udleverede lyd-signaler, hvorpå 9 forskellige øvelser vil blive foretaget. Projektet vil desuden indeholde arbejde med funktioner i Mattab, som f.eks. plotning af data og udregninger på arrays.

2 Delopgave 1 & 2

2.1 Introduction

I de første to øvelser vil jeg bestemme antallet af sampless i hver af de tre lyd-signaler, og derefter plotte hvert signal, med fokus på korrekte akse-beskrivelser.

2.2 Fremgangsmåde

Listing 1: Matlab kode for øvelse 1 & 2

```
% Lægger filerne ind i arrays
[y(1).samples, ~] = audioread('Signal_s1.wav');
[y(2).samples, ~] = audioread('Signal_s2.wav');
[y(4).samples, f_s] = audioread('Signal_s3.wav');

%Splitter s2 i to kanaler
y(3).samples = y(2).samples(:,2);
y(2).samples = y(2).samples(:,1);

y(1).name = "Signal\_s1.wav";
y(2).name = "Signal\_s2.wav - channel 1";
y(3).name = "Signal\_s2.wav - channel 2";
y(4).name = "Signal\_s3.wav";

%Udregner tidsakser for signalerne
for i = 1:length(y)
    y(i).samples = y(i).samples';
    % number of sampless 1, 2, 3
    y(i).nS = length(y(i).samples);
    %Calculating x axis for signals
    y(i).time = [0:y(i).nS-1]*(1/f_s);
end

f1 = figure;
%Plotter signalerne
for i = 1:length(y)-1
    subplot(2,2,i)
    plot(y(i).time, y(i).samples)
    title(y(i).name)
    xlabel("Time(s)")
    ylabel("Amplitude(~)")
end
```

2.3 Resultater

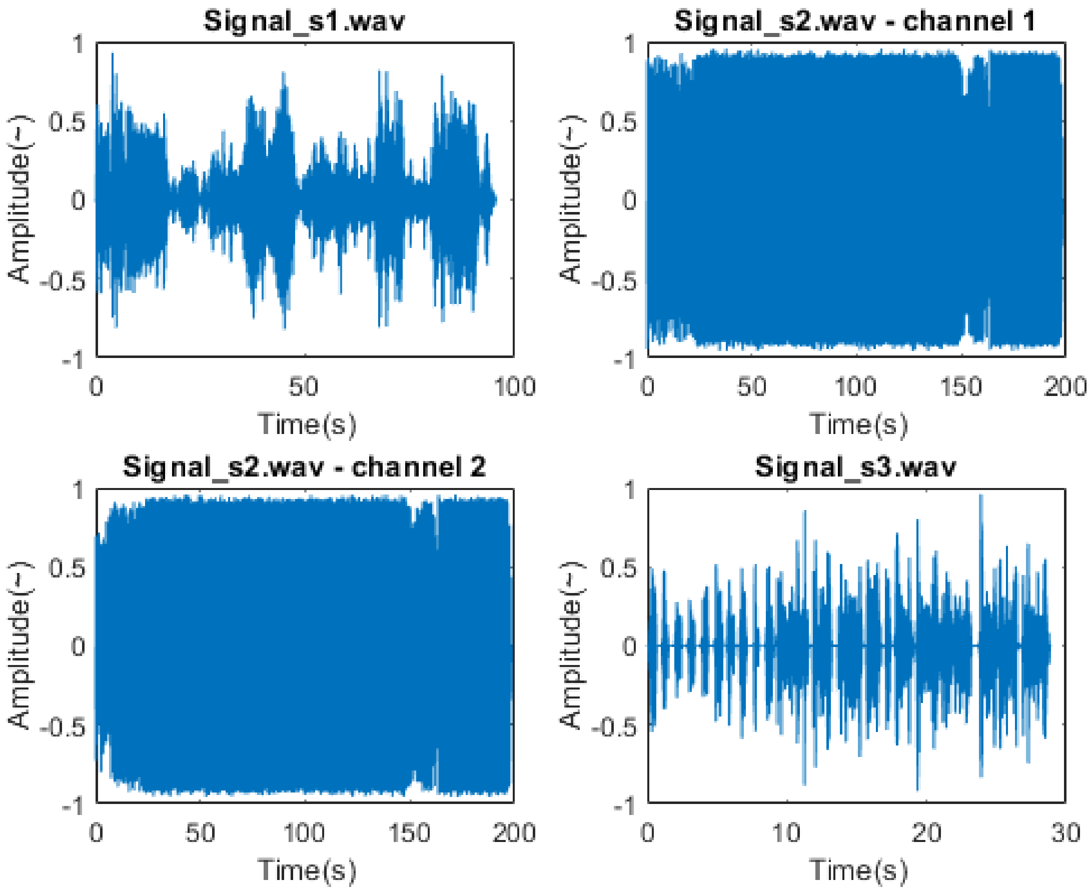

Figure 1: Plot over alle fire mono-signaler

2.4 Diskussion

Som det ses af Listing 1, benyttes audioread() til at aflæse antallet af sampless,y, og samplingsfrekvensen af alle tre lydsignale, f_s. y er her en struct, hvor hvert signal får lagret deres antal sampless, i arrays ved navnet samples. f_s er blot en enkelt variable, da jeg ved at alle signalerne har den samme samplingsfrekvens. Jeg

ved at 'Signal_s2.wav' er et stereo signal, og vil derfor have signaler opbevaret i to seperate kanaler. På linje 7 og 8 ses hvordan jeg opdeler dem i hver sit array. et for-loop, muliggjort af mit brug af structs, bruger jeg til at udregne tidsakserne vi skal bruge til at plotte signalerne. På linje 17 vender jegi først arraysne, så Matlab vil lade mig regne med dem. Derefter sættes Ns til længden af hvert array af samples. Jeg opnår til sidst mit array af tider ved at lave et array der ganger antallet af samples med svingningstiden, altså $T = 1/f$. Jeg

benytter til sidst endnu et for-loop til at printe hvert signal som et sub-plot, og skriver korrekte aksenavne, som kan ses på figur 1.

2.5 Konklusion

Jeg konkludere at jeg kan benytte Matlab's funktioner til at bearbejde, og analysere lyd-signaler. Desuden kan jeg benytte Matlab til at visualisere dataen, til overskueliggørelse og sammenligning.

3 Delopgave 3 & 4

3.1 Introduction

I denne del af øvelsen vil jeg foretage en række udregninger for at bestemme max, min, mean, rms og effekt værdierne for hvert mono-signal. Jeg vil derefter bestemme crest-værdierne for hvert signal.

3.2 Fremgangsmåde

Listing 2: Matlab kode for øvelse 3 & 4

```
for i = 1:length(y)
    %Find max, min, mean, rms & effect
    y(i).max = max(y(i).sample);
    y(i).min = min(y(i).sample);
    y(i).mean = mean(y(i).sample);
    y(i).rms = rms(y(i).sample);
    y(i).effect = sum(y(i).sample.^2);
    %Calculating Crest value
    y(i).crest = 20*log10(y(i).max/y(i).rms);
end
```

3.3 Resultater

```
s1

ans =

    0.9306

s2_left

ans =

    0.9558

s2_right

ans =

    0.9558

s3

ans =

    0.9600
```

Figure 2: Max-værdier for de fire mono-signaler

```
s1

ans =

   -0.8205

s2_left

ans =

   -0.9558

s2_right

ans =

   -0.9558

s3

ans =

   -0.9192
```

Figure 3: Min-værdier for de fire mono-signaler

```
s1

ans =

  -4.2834e-05

s2_left

ans =

  -1.9413e-05

s2_right

ans =

  -1.9376e-05

s3

ans =

  -5.7438e-04
```

Figure 4: Mean-værdier for de fire mono-signaler

```
s1

ans =

    0.1199

s2_left

ans =

    0.3383

s2_right

ans =

    0.3548

s3

ans =

    0.0919
```

Figure 5: RMS-værdier for de fire mono-signaler

```
ans =

   17.8006


ans =

    9.0213


ans =

    8.6064


ans =

   20.3770
```

Figure 6: Crest-værdier for de fire mono-signaler

3.4 Diskussion

Som det ses af Listing 2 har Matlab mange funktioner til at udregne værdier for dig. Funktionerne $max(), min(), mean()$ og $rms()$ køres blot på mit sample-array, for at bestemme de tilsvarende værdier.Effekten udregnes som summen af alle samples i anden, altså $rms = (sum(y.sample))^2$. Lignende er crest-værdien for et signal givet ved $crest\_val = 20 * log10(y.max/y.rms)$. Disse funktioner indkorporeres i for-loopet, og udregner derved rms -og crest-værdierne for hvert signal. Jeg ser at alle signalernes max -og min-værdier er nogenlunde ens,

og desuden alle har mean-værdier der ligger tæt på 0. Dette skulle de også gerne, da de jo er bølger, der svinger omkring 0 på y-aksen. Jeg ser dog en ret stor variation i deres crest-værdier.

Dette skyldes, at crest-værdien jo er forholdet mellem signalets peak-værdier og rms-værdier. Og som vi jo kan se fra Figur 1, foretager signalerne meget forskellige svingninger. Kanal 1 og 2 fra signal 2 foretager dog meget ens svingninger, hvilket også er hvorfor deres crest-værdier er meget tæt på hinanden.

3.5 Konklusion

Jeg konkludere at jeg kan opnå en stor mængde oplysinnger om et signal ved at analysere de forskellige værdier tilknyttet hvert signal, da de kan fortælle mig hvordan det foretager sine svingninger.Dog er det vigtig at have alle oplysningerne, da man ellers nemt kan tror at to signaler minder om hnanden, når de ikke gør det.

4 Delopgave 5 & 6

4.1 Introduction

I denne del af opgaven vil jeg vil jeg nedsamples signalet s1 med en faktor 4, hvorefter jeg vil sammenligne den med originalen ved bl.a plots og lyd.

4.2 Fremgangsmåde

Listing 3: Matlab kode for øvelse 5 & 6

```
%Definer nedsampling af signal
y(5).samples = downsamples(y(1).samples, 4);
y(5).name = "Resampling of " + y(1).name;

y(5).samples = y(5).samples';
% Antal af samples
y(5).nS = length(y(5).samples);
%Udregner x aksen for signalet
y(5).time = [0:y(5).nS-1]*(1/Fs);

figure
subplot(1,2,1)
plot(y(1).time, y(1).samples)
title(y(1).name)
xlabel("Time(s)")
ylabel("Amplitude(~)")
```

```
subplot(1,2,2)
plot(y(5).time, y(5).samples)
title(y(5).name)
xlabel("Time(s)")
ylabel("Amplitude(~)")

soundsc(y(5).samples, Fs/4);
```

4.3 Resultater

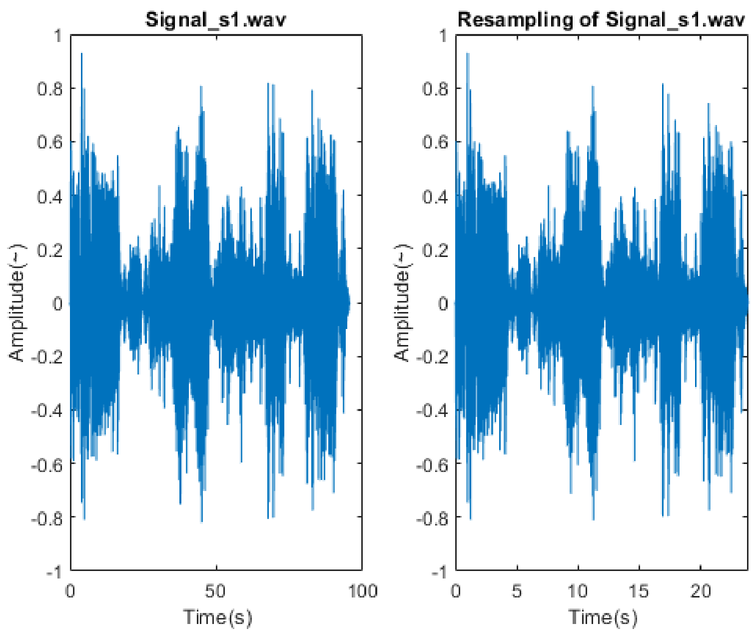

Figure 7: Max-værdier for de fire mono-signaler

4.4 Diskussion

Jeg benytter Matlab funktionen *downsamples*() til at fjerne hvert fjerde element i mit samples-array. Herefter korrigere der for nedsampling på frekvensen og samplestiden. Signalet før og efter plottes så som subplots, så de bedre kan sammenlignes, som ses på figur 7. Der kan ikke ses den store forskel, da samplingsraten stadig er relativt høj.

Derfor benytter jeg også *soundsc*() funktionen i Matlab til at lytte til begge signaler, hvor jeg kan hører en klar forskel, med lavere kvalitet lyd efter nedsamplingen. Det bemærkes her at frekvensen skal justeres for at lyden spiller i den rigtige hastighed, da der jo er fire gange mindre samples i signalet.

4.5 Konklusion

Jeg konkludere at en nedsampling af et signal vil forværre kvaliteten af signalet, da det ikke kan gengive det 'originale' signal lige så præcist. Desuden finder jeg også at en nedsamling også vil førre til et krav om justering af frekvensen.

5 Delopgave 7 & 8

5.1 Introduction

Jeg vil nu sammenligne de to stereo kanaler bestemt fra s2 i øvelse 1. Derefter vil jeg kvantisere kanal 2 af signal s2 til 4 bits, og undersøge hvad det ændre.

5.2 Fremgangsmåde

Listing 4: Matlab kode for øvelse 7 & 8

```
figure
subplot(1,2,1)
plot(y(2).time, y(2).sample)
title(y(2).name)
xlabel("Time(s)")
ylabel("Amplitude(~)")
subplot(1,2,2)
plot(y(3).time, y(3).sample)
title(y(3).name)
xlabel("Time(s)")
ylabel("Amplitude(~)")

%Benytter funktion til at kvantisere til 4 bits
y(6).sample = quantizeN(y(2).sample, 4);
soundsc(y(6).sample)
figure
plot(y(2).time, y(6).sample)
```

5.3 Resultater

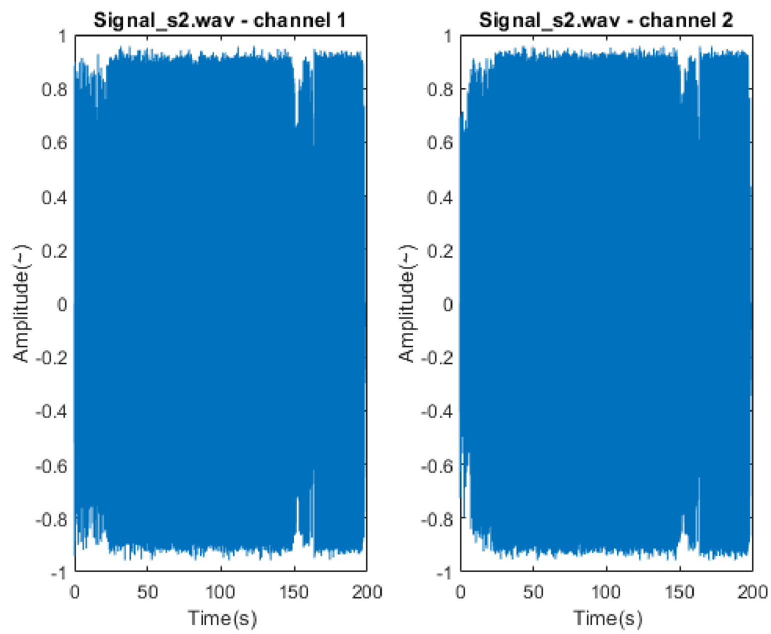

Figure 8: Sammenligning af de to kanaler

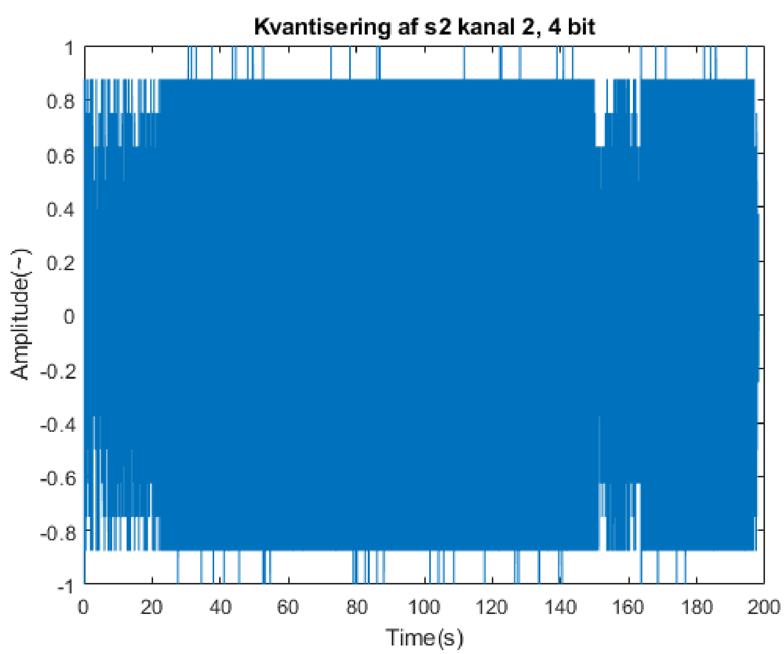

Figure 9: Sammenligning af de to kanaler

5.4 Diskussion

Det ses fra Figur 8 at der forekommer forskelle mellem de to kanaler. Primært ses det er kanal 1 har større amplituder omkring 0 sekunder, mens kanal to har stigende amplituder, der bliver mere konsistente efter ca. 40-50 sekunder. De to kanaler er tilnærmelsesvist ens fra 50 sekunder til omkring 150 sekunder, hvor kanal ikke foretager et større udsving end kanal 2. Amplituderne begynder så at stige igen, og når igen en enshed, dog foretager kanal en højere amplitude omkring de 200 sekunder.

Funktionen $quantizeN()$ benyttes til at kvantisere kanal 2 af signalet. Resultatet kan ses på Figur 9.Her ser jeg at signalet har fået en meget grovere opløsning, og ved brug af *soundsc*() funktionen hører jeg også at signalet er blevet meget mere støjfyldt. Jeg forestiller mig at dette skyldes de lavere amplituder i signalet, som ved kvantiseringen er blevet forstærket, og nu kan høres som støj.

5.5 Konklusion

Jeg konkludere at man kan se en forskel mellem de to kanaller af signal 2, specielt omkring 0 sekunder, og 150 sekunder. Videre konkludere jeg også at en kvantisering af et signal kan fører til mere baggrundsstøj, samt mindre præcise amplitudeaflæsninger.

6 Delopgave 9

6.1 Introduction

I denne øvelse vil jeg lave et eksponentielt 'fade-out' på signalet s3. Fade-outet vil blive foretaget over den sidste tredjedel af signalet, og vil ende med en 5% signalstyrke.

6.2 Fremgangsmåde

Listing 5: Matlab kode for øvelse 9

```
t = y(4).time(1:y(4).nS/3+1);
a = (0.05/1)^(1/(t(end)-t(1)));
eksp = a.^t; %Fra 1 til 0.05
figure
plot(t,eksp)

y(7).samples = y(4).samples;

y(7).samples(length(y(7).samples)*2/3:end) = eksp.*y(7).samples(length(y(7).samples)*2/3:en
figure
plot(y(4).time, y(7).samples)
soundsc(y(7).samples)
```

6.3 Resultater

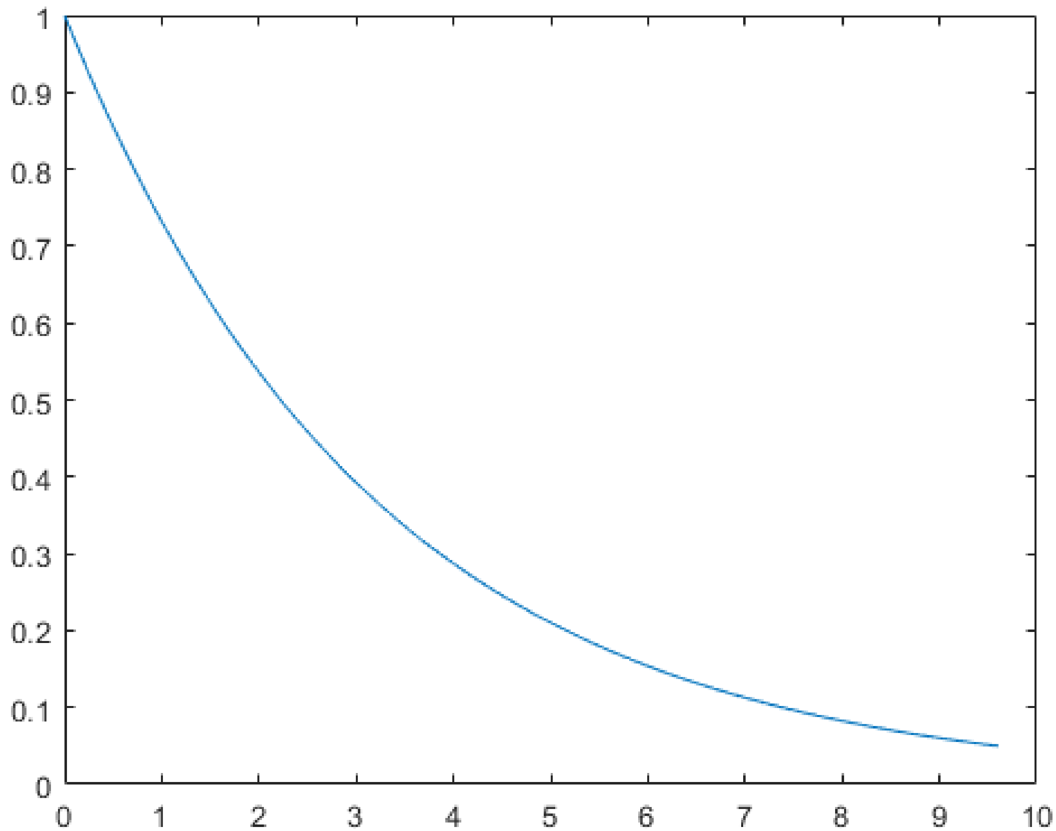

Figure 10: Eksponential funktion

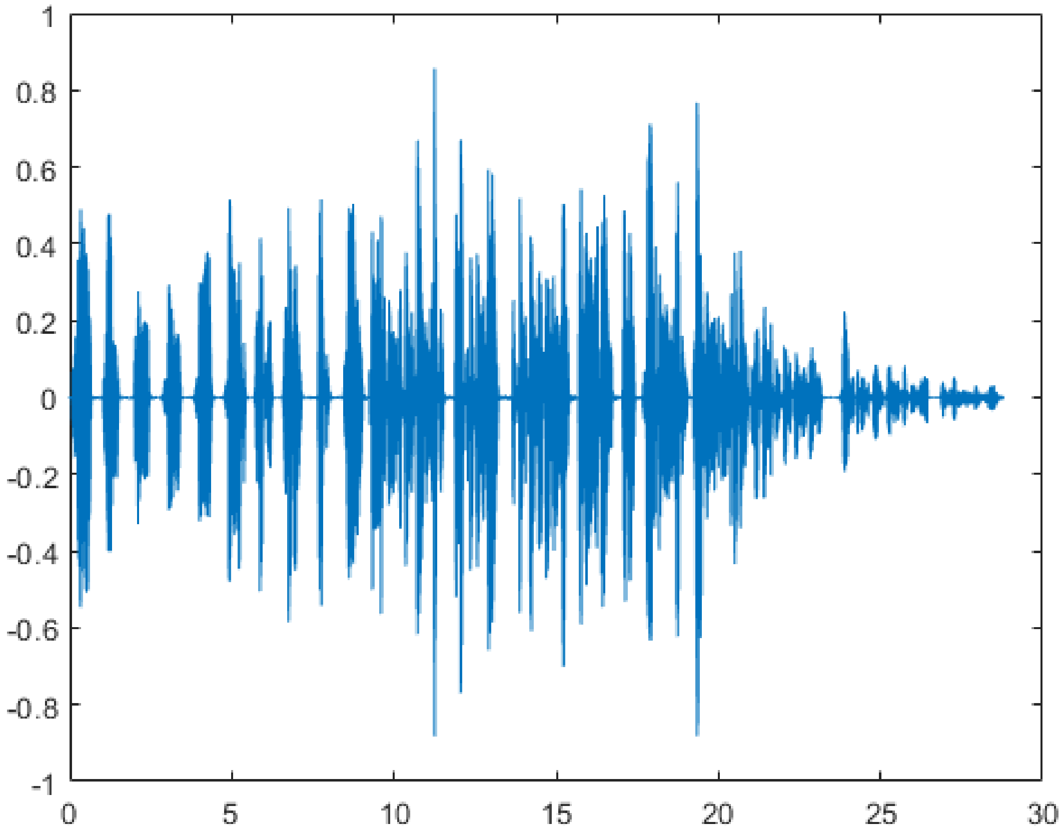

Figure 11: signal s3 med fade out

6.4 Diskussion

Jeg bestemmer først eksponential funktionen der skal benyttes til fade-out. Funktionen skal kun løbe fra 2/3 af hele samplesarray til 3/3, og arrayet t oprettes derfor på linje 1. Konstanten a bestemme vha. funktionen $\sqrt[x_1-x_2]{\frac{y_1}{y_2}}$, som ses på linje 2. Eksponentialfunktionen konstrueres så, og på figur 10 ses hvordan den falder fra 1 til 0.05, altså 5%, over det rigtige interval.

Jeg opstiller så mit signal, ved at gange den sidste tredjedel af s3 signalet med eksponentialfunktionen. Der er her værd at notere at jeg ikke behøvede at Matlab hvilke værdier af eksponentialfunktionen der skulle ganges på hvilke værdier af signalet. Så længe den vidste hvor jeg ville starte og slutte tog programmet sig af resten. På Figur 11 kan man se hvordan signalet aftager på den sidste tredjedel af grafen. Dette er specielt fremhævet ved sammenligning med Figur 1.

6.5 Konklusion

Jeg konkludere at ved brug af en eksponential funktion kan jeg skabe et fade out på et signal over et ønsket interval.

7 opgave 1.15 fra DSB lektion 3

7.1 Introduction

I den sidste del af øvelsen vil jeg besvare opgave 1.15 fra DSB uge 38. Opgaven går ud på at konstruere et ekko-signal på 150ms. Jeg vil efterfølgende variere ekkoet til 40ms og 300ms respektivt.

7.2 Fremgangsmåde

Listing 6: Matlab kode for øvelse 1.15, uge 39 DSB

```
f_s = 5000; %Samplingsfrekvens
T_s = 1/Fs; % Periode

length = 1; % Længde i sekunder
Ns = Fs*length; % Antallet af samples
t = [0:Ns-1]*Ts; % Tiden over samplingen

%Tone beskrivelse
f = 2350; %440 Hz
A = 3; % Amplitude er 3
sig = A * sin(f0*2*pi*t); %funktion for signal

ekko.delay = 150; %ekko med 150 ms forsinkelse
ekko.Ns = ekko.delay * (f_s/1000); %Antallet af samples i ekkoet

tone_ekko = [zeros(1,ekko.Ns), sig]; %Forsinker ved at sætte 0-array foran
tone_ext = [sig, zeros(1,ekko.Ns)]; % Forlænger array ved at sætte 0-array bagved

tone_sum = tone_ekko + tone_ext
ekko_t = [0:Ns+ekko.Ns-1]*T_s;

figure
plot(t, sig);
title("Original tone");
xlabel("Tid - sekunder");
ylabel("Amplitude - ~");

figure
plot(ekko_t, tone_ekko);
title("Ekko");
xlabel("Tid - sekunder");
ylabel("Amplitude - ~");

figure
```

```
plot(ekko_t, tone_sum);
title("Ekko + Original");
xlabel("Tid - sekunder");
ylabel("Amplitude - ~");

soundsc(tone_sum)
```

7.3 Resultater

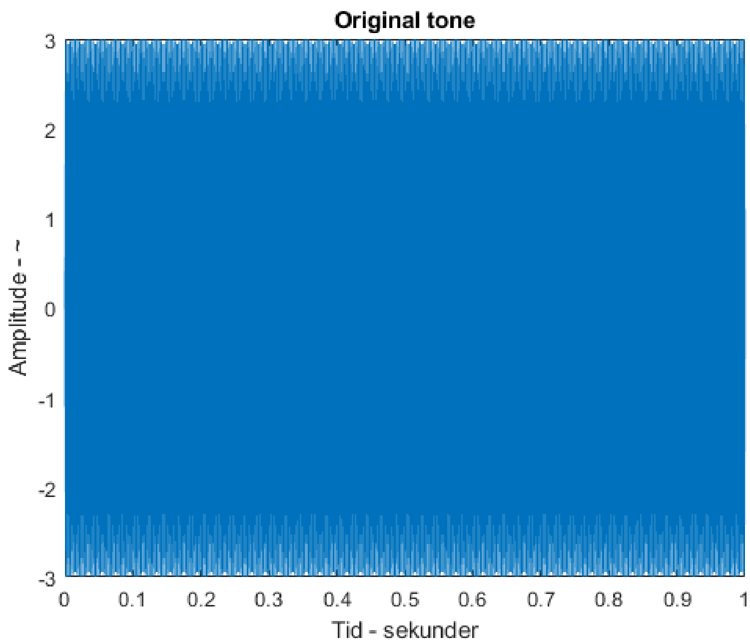

Figure 12: Signal *sig* uden ekko

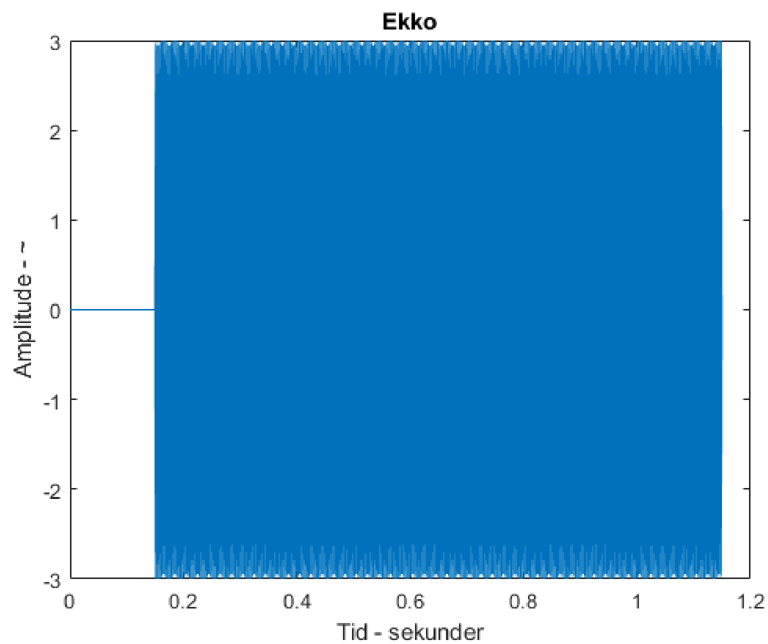

Figure 13: ekko signal på 150ms af *sig*

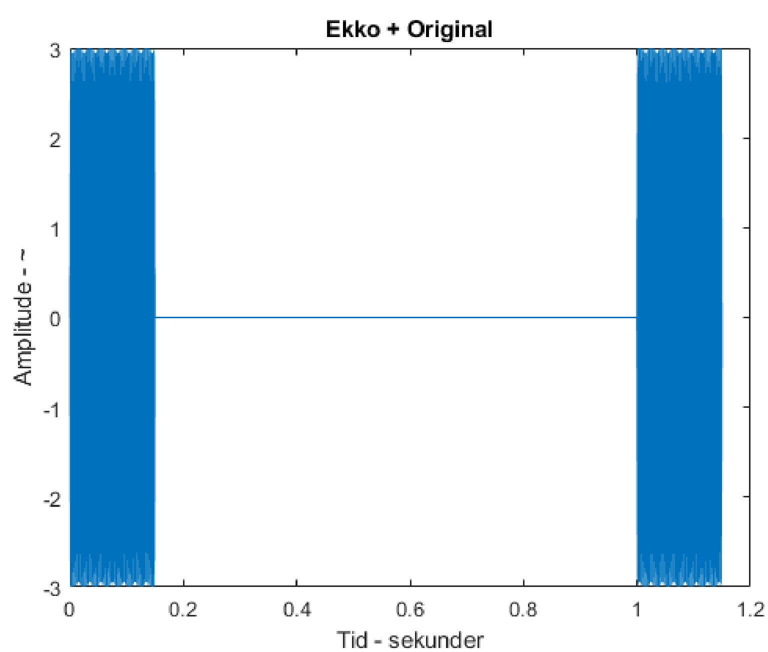

Figure 14: Interferens mellem de to signaler, 150ms

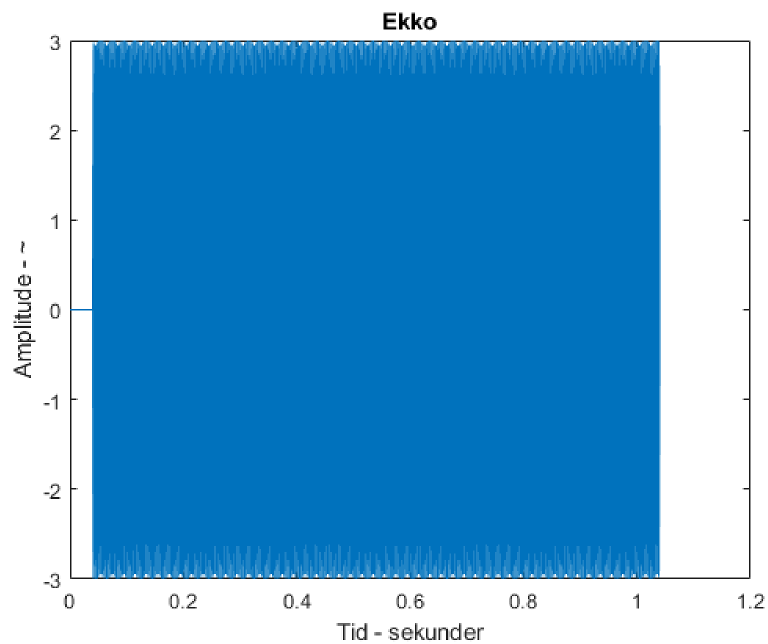

Figure 15: Ekko signal på 40ms af *sig*

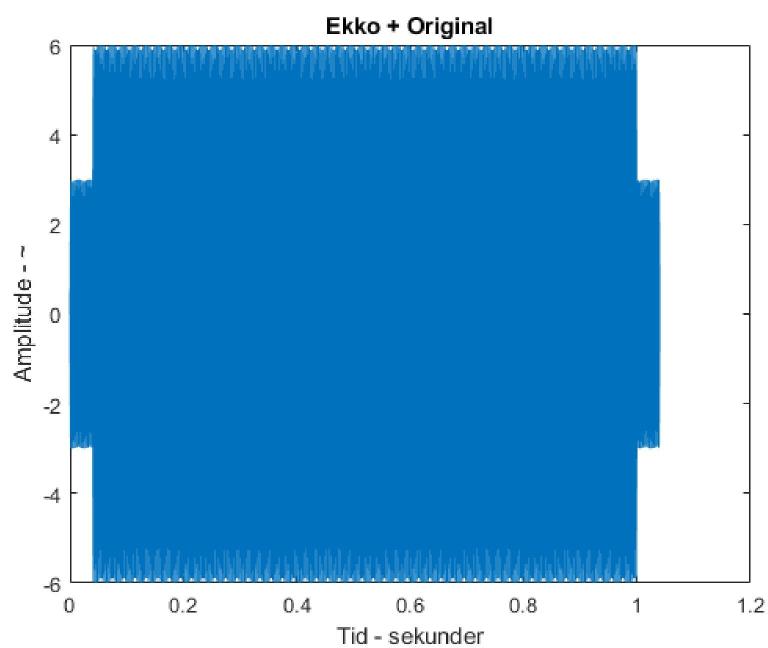

Figure 16: Interferens mellem de to signaler, 40ms

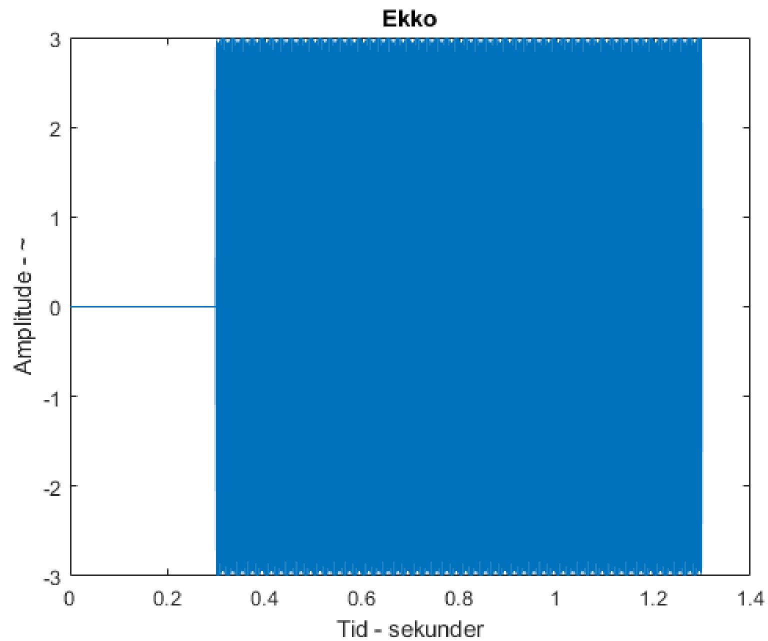

Figure 17: Ekko signal på 300ms af *sig*

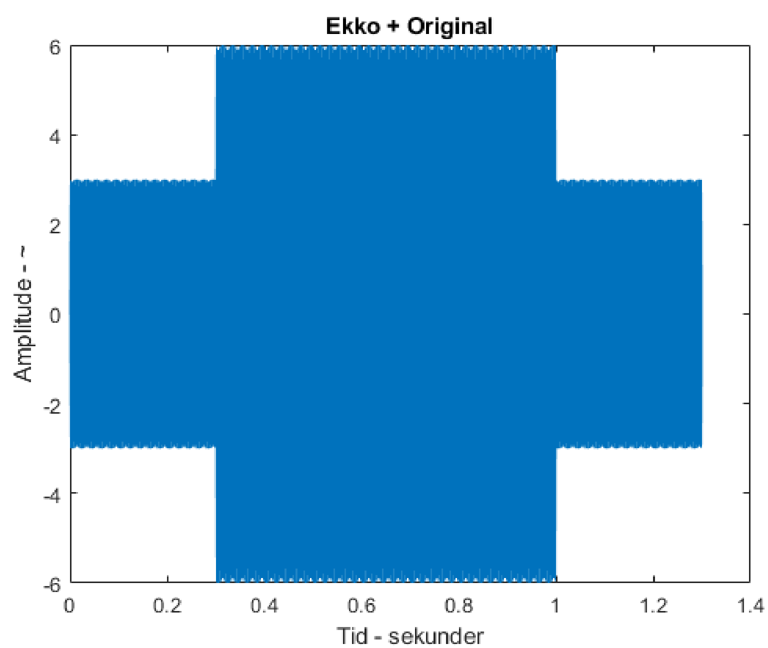

Figure 18: Interferens mellem de to kanaler, 300ms

7.4 Diskussion

For at kunne plotte signalet laver jeg først en ny tidsakse ud fra samplingfrekvensen f_s og perioden T_s. Jeg opstiller derefter min funktion for signalet, *sig*, og definerer derefter mit ekko-delay. Jeg benytter *zeros* funktionen til at indsætte nuller foran sig-signalet, så det får det ønskde delay. Ligeledes gør jeg det samme, dog nu med nullerne bagefter. Dette er så de har samme mængde elementer i deres arrays, og kan plottes sammen. Jeg plotter så den originale signal-funktion sammen med ekko-funktionen.

På figur 12 og 13 kan man se de to funktioner plottet, og på figur 14 ses hvordan de inteferere. Det ses at der fra ca. 150ms til 1000ms forekommer stærk destruktiv interferens, og de to signaler ødelægger hinanden, idet de er faseforskudt med pi radianer. på figur 15 og 17 ses istedet resultatet af interferens med et ekko signal på 40ms og 300ms henholdsvist. Begge steder ser jeg konstruktiv interferens, altså sker der en addition af de to signalers amplituder. Altså må signalet *sig* være faseforskudt med 0 grader med både 40ms og 300ms ekkoet.

7.5 Konklusion

Jeg konkludere at man kan konstruere et ekko-signal vha. Matlab's zeros() funktion. Jeg har desuden set hvordan Matlab kan benyttes til at analysere interferens mellem signaler, og bl.a kan bruges til at besteme hvornår der vil ske konstruktiv og destruktiv interferens.