FYS2130 – Oblig 1

Anders Hafreager

28.01.2009

OPPGAVE 1

I denne oppgaven skal jeg prøve å bestemme kvalitetsfaktoren (Q-verdien) for svingehårene i basillarmembranen som ligger i øret. Jeg skal gjøre dette for to forskjellige frekvenser innenfor mitt intervall [300,1000] Hz. Jeg har skrevet et MATLAB-program (se vedlegg 1) hvor jeg kan velge to frekvenser f_1 og f_2 og programmet vil da 16 ganger spille av en lydsekvens som ved tilfeldighet enten består av samme frekvens eller et bytte halvveis i lydsekvensen. Jeg har lagt inn fase på den andre lyden, så vi ikke får noe hakk mellom et eventuelt frekvensbytte. Etter hver avspilling skriver jeg inn om jeg tror det var samme frekvens eller ikke, og får resultater etter alle avspillingene. Dette vil hjelpe meg i å kunne finne hvilke frekvenser jeg kan høre mer nøyaktig:

Frekvens 1: 440 Hz:

#	Frekvens 1 / Frekvens 2 [Hz]	Resultat [RIKTIGE/GALE]
1	440 / 447	16/0
2	440 / 445	16/0
3	440 / 444	16/0
4	440 / 443	16/0
5	440 / 442	13/3
6	440 / 441.5	10/6

Tabell 1: Resultat etter forsøket i oppgave 1 med 440 Hz som grunnfrekvens

Her ser jeg at jeg klarer tydelig å høre forskjell helt til 443 Hz ($\Delta f = 6\,Hz$), tabellen viser at jeg ikke er konsistent på lavere frekvenser enn det. Jeg kan derfor konkludere med en Q-verdi regnet ut med dette valget for Δf.
Vi får da:

$$Q_{440} = \frac{f}{\Delta f} = \frac{440}{6} = 73.$$

Frekvens 2: 880 Hz:

#	Frekvens 1 / Frekvens 2 [Hz]	Resultat [RIKTIGE/GALE]
1	880 / 890	16/0
2	880 / 888	16/0
3	880 / 887	16/0
4	880 / 886	16/0
5	880 / 885	16/0
6	880 / 884	15/1
7	880 / 883	14/2

Tabell 2: Resultat etter forsøket i oppgave 1 med 880 Hz som grunnfrekvens

Etter disse resultatene ser jeg at jeg med sikkerhet kan avgjøre toneforskjeller med en $\Delta f = 10\,Hz$. Dette gir:

$$Q_{880} = \frac{880}{10} = 88.$$

Konklusjon:
Ved høyere frekvenser har jeg en høyere Q-verdi. Dette virker rimelig siden vi har best hørsel i mellomtoneområdet, rundt 1000 Hz. Det kan være flere feilkilder her, det kan f.

eks. være kvantiserte målinger av frekvensforskjeller (hører på hele Hz). Flere kommentarer om Q-verdier kommer i Oppgave 2.

OPPGAVE 2

I denne oppgaven skal jeg finne hvilken minste tid t en lyd må spilles av i for å kunne gjenkjenne tonen. Har laget et MATLAB-program for dette (se vedlegg 2). Programmet er et lite spill hvor den 16 ganger spiller av to lyder med 0,3 sekunders mellomrom, den ene er rask (det er denne tiden jeg skal finne), den andre er lang så jeg tydelig hører hvilken tone det var. Halvparten av gangene spiller den samme lyd, den andre halvparten er første lyd en lyd som er noe ulik (Δf står oppført i tabellen). Jeg blir bedt om å si om det er samme lyd etter hver gang, og får resultatet etter de 15 gangene. I tabell 3 har jeg tatt med noen forsøk med forskjellige frekvenser og tidslengder på den korte lyden.

Utførelse:

#	Grunnfrekvens [Hz]	Δf [Hz][1]	Δt [ms][2]	[RIKTIGE/GALE]	$\Delta t \Delta f$
1	400	12	80	16/0	0,96
2	400	8	80	16/0	0,64
3	400	4	80	16/0	0.32
4	400	12	60	15/1	0,72
5	400	8	60	16/0	0,48
6	400	4	60	12/4	0,24
7	400	12	40	16/0	0,48
8	400	8	40	16/0	0,32
9	400	4	40	13/3	0,16
10	400	12	20	13/3	0,24
11	400	8	20	13/3	0,16
12	400	4	20	11/5	0,08

Tabell 3: Resultater etter flere forsøk på å finne laveste tid t en lyd kan spilles for å kunne avgjøre frekvensen.

Konklusjon:

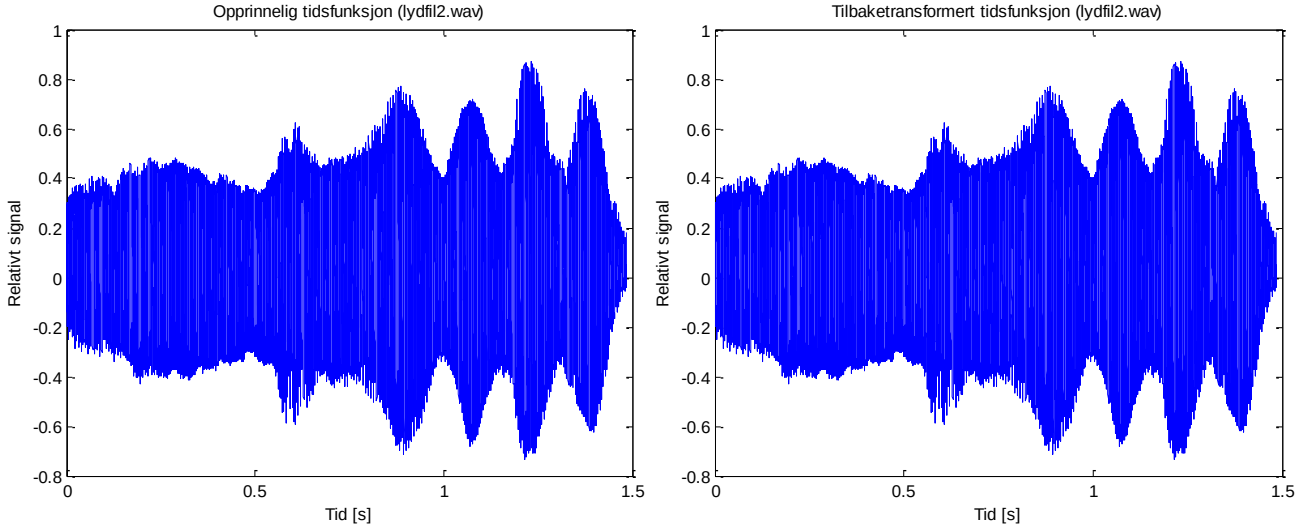
Legger merke til at tallet $\Delta f \Delta t$ er en viktig faktor. Jo lavere frekvensforskjell Δf jeg har mellom to toner, jo lengre tid Δt trenger jeg for å kunne avgjøre om det er samme tone eller ikke. Dette er tett knyttet opp mot Q-verdiene vi fant i oppgave 1. En høy Q-verdi betyr at vi er i stand til å kunne skille nærliggende frekvenser, men trenger desto lengre tid for å kunne gjøre det. Q-verdien er direkte knyttet opp til tiden det tar før en svingning (på f. eks. hårene i øret) har stabilisert seg og kan måles nøyaktig. Vi ser derfor at jo høyere tall $\Delta f \Delta t$ vi har, jo lettere er det å avgjøre hvilken frekvens det er.

OPPGAVE 3

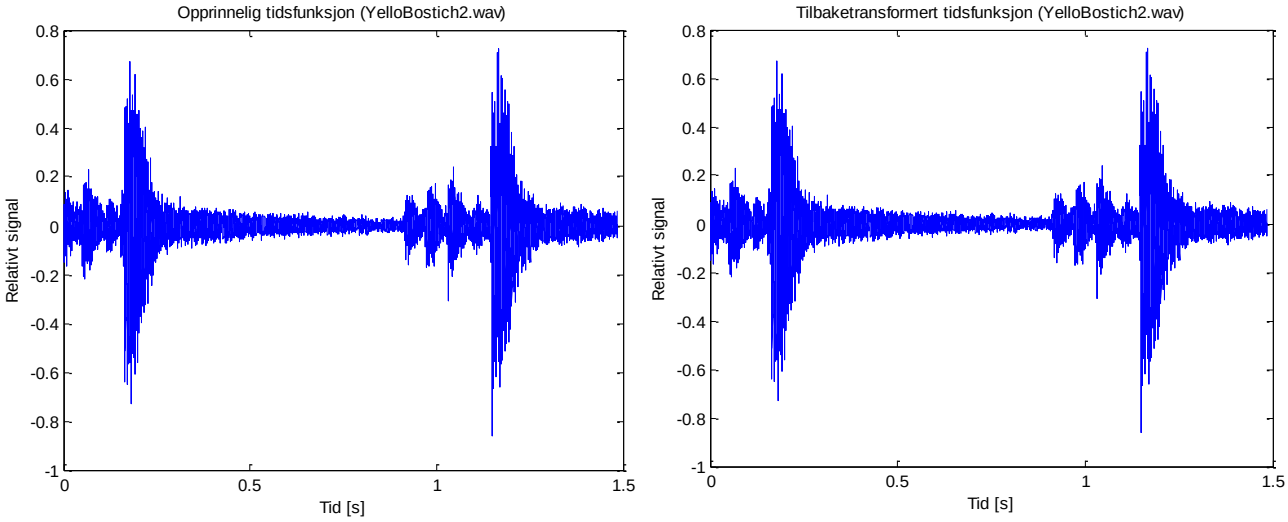
Del 1:

Her skal jeg fouriertransformere en bit av de to lydfilene jeg har og tilbaketransformere før jeg spiller av. Den første fila, 'lydfil2.wav', har en lengde på 101620 samplinger. For å få rask transformasjon velger jeg ut 2^n samplinger av lydfila fra sampling 8000. Jeg bruker da samplingene i intervallet [8000, 2^{16}+8000] = [8000, 73536]. Grunnen til at jeg korter ned fila er fordi $2^{16} < length('lydfil2.wav') < 2^{17}$. Hadde jeg brukt 2^{17} ville jeg fått en god mengde samplinger med verdi 0, og fourierspekteret ville bestått av en utrolig kompleks og 'feil' mengde med frekvenser for å skape disse 0-verdiene. Koden brukt for å gjøre denne oppgaven ligger som Vedlegg 3.

Jeg gjør som beskrevet over for begge lydfilene og kan verken høre eller se (se figur 1 og 2) noen forskjell i de to lydsnuttene (før og etter transformasjon) for noen av lydfilene. Vi kan også ta og sjekke differansen mellom kvadratet av de to N-dimensjonale tuplene for lydarrayet før og etter transformasjonen, denne fant jeg til å være $\Delta s \approx 1{,}8 \cdot 10^{-14}$ for *'lydfil2.wav'*, en svært liten feil. Vi kan altså konkludere med at transformasjonen ikke gjør noe særlig med tidsrommet! (Det kan naturligvis forekomme avrundingsfeil og andre unøyaktigheter).



Figur 2: Tidsfunksjon før og etter uendret fouriertransformasjon på 'lydfil2.wav'



Figur 1: Tidsfunksjon før og etter uendret fouriertransformasjon på 'YelloBostich2.wav'

Del 2:

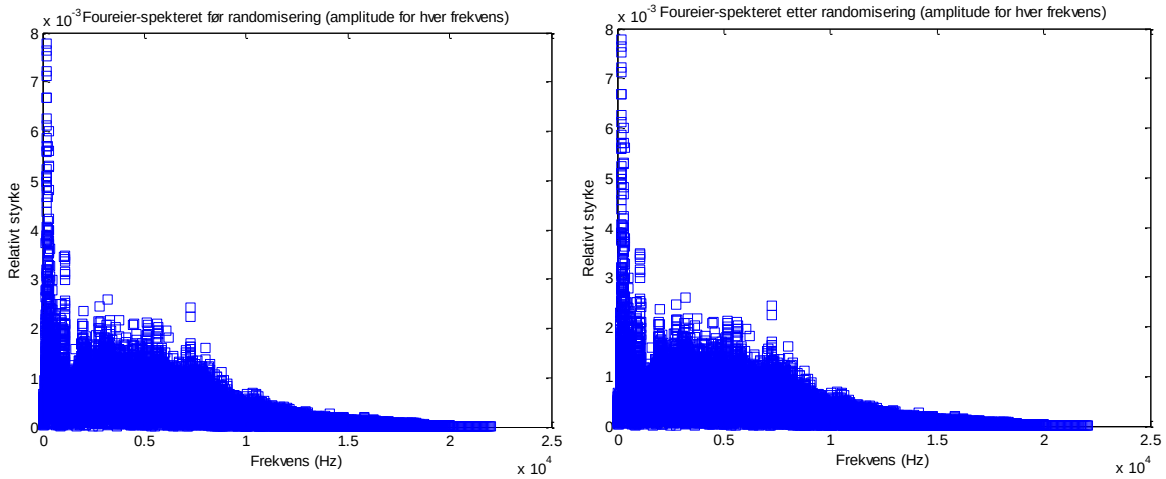
Her skal jeg randomisere de transformerte fasene før vi transformerer tilbake og spiller av. Jeg passe på at amplituden $A_n = \sqrt{Real(z_w)^2 + Imag(z_w)^2}$ må være konstant for alle frekvensene i fourierspekteret. Det vi skal modifisere, er fasen for hver frekvens $\phi = \arctan\left(\frac{Imaag\ (z_w)}{Real\ (z_w)}\right)$, altså forholdet mellom real- og imaginær-delen til det komplekse tallet. Jeg velger å ta en vilkårlig fase i intervallet $[0,2\pi]$ og beregne real- og imaginær-delen (a og b) ved:

$$a_n = A_n\ cos(\phi)$$
$$b_n = A_n\ sin(\phi)$$

der ϕ er den nye fasen, og det nye komplekse tallet er gitt ved

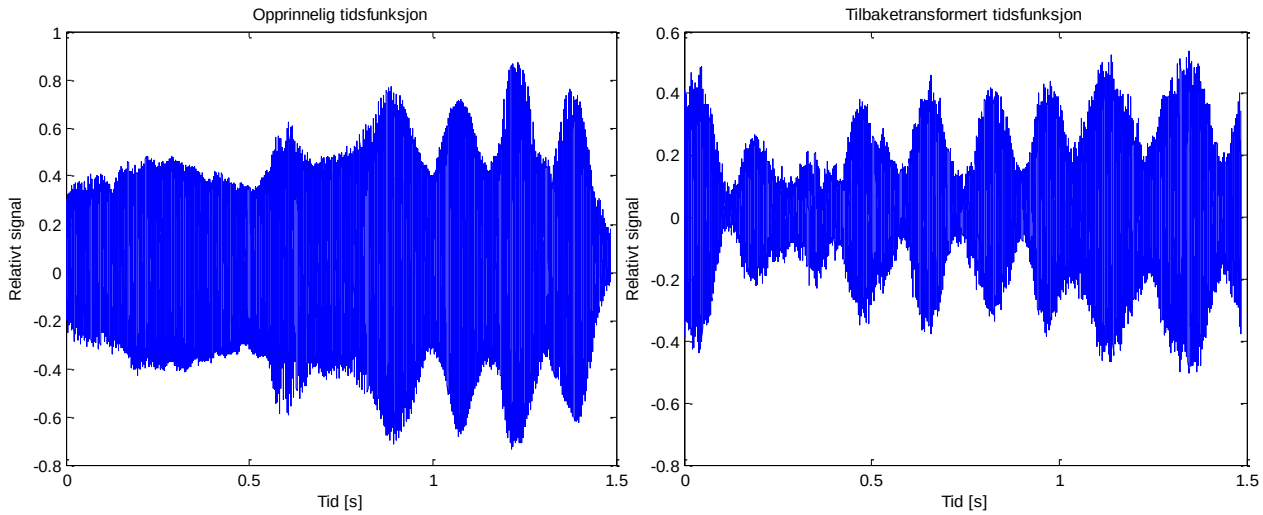
$$z_n = a_n + ib_n$$

Dette gjøres for alle frekvensene i fourierspekteret. Vi kan se at fourierspekteret er identisk før og etter randomisering (se figur 3), noe som er riktig siden vi har plottet kun amplituden for hver frekvens (A_n som ikke endrer seg under faserandomisering).

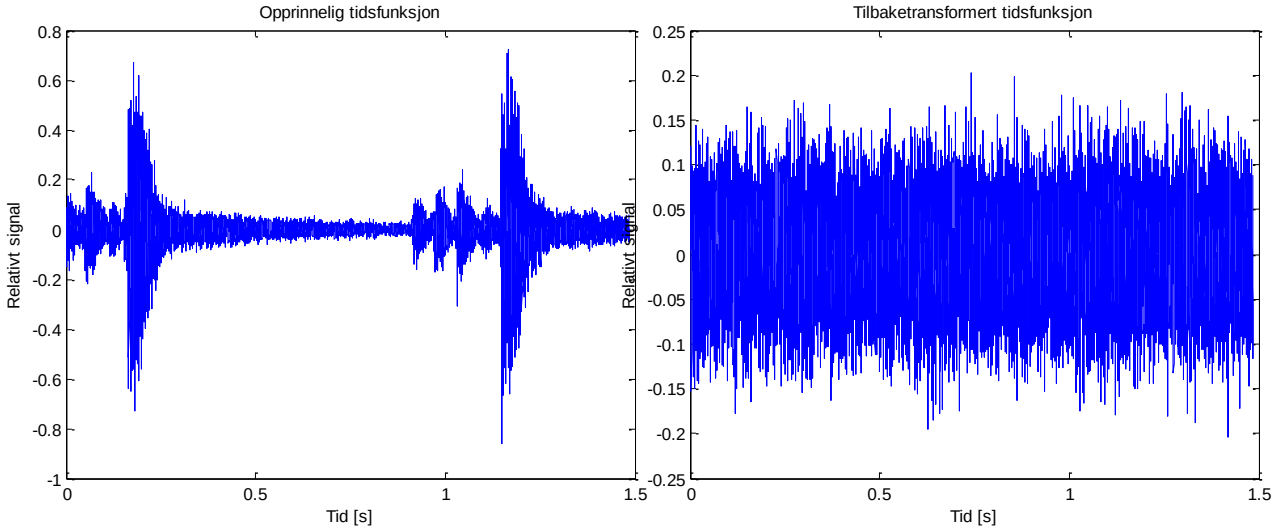


Figur 3: Lydplot før og etter fouriertransformasjon av 'lydfil2.wav'

Når jeg tilbaketransformerer til tidsrommet og plotter det, ser jeg et helt annet plot enn den opprinnelige tidsfunksjonen (se figur 4 og 5). Dette er jo naturlig siden fasene ligger til grunn for at komplekse sammensetninger av cosinus- og sinus-funksjoner skal gjengi nøyaktig de samplingene vi har.



Figur 4: Tidsfunksjon før og etter faserandomisering av 'lydfil2.wav'

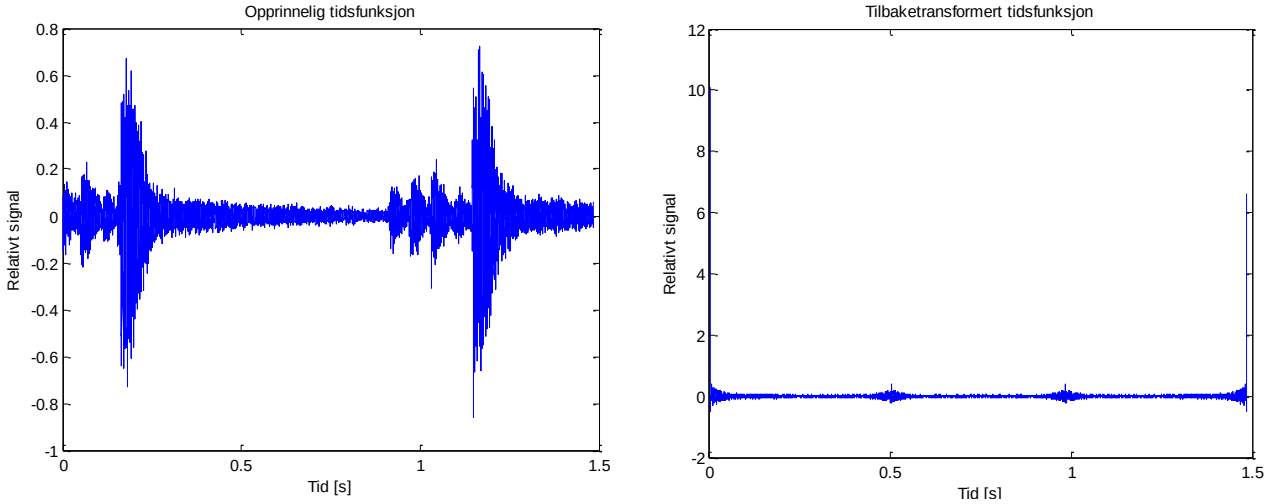


Figur 4: Tidsfunksjon før og etter faserandomisering av 'YelloBostich2.wav'

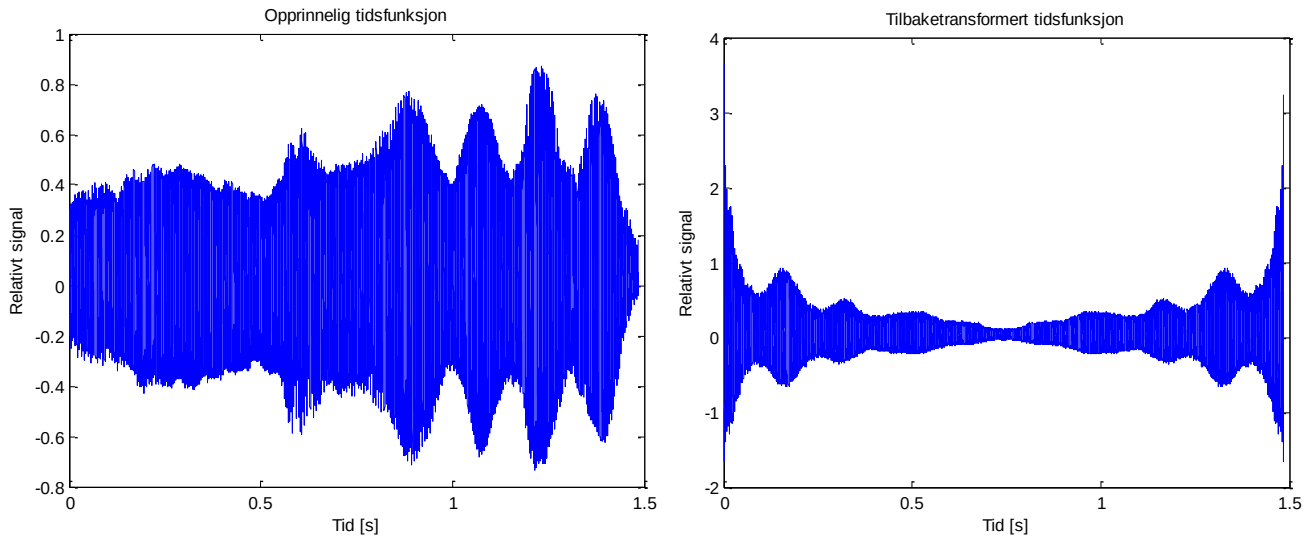
Lyden høres svært annerledes ut for begge lydfilene, trommeslagene blir til ren støy og det vil være umulig å gjenkjenne akkurat den lyden. Den andre lydfilen er mer gjenkjennelig.

Del 3:

Her er oppgaven å nullstille fasene. Dette betyr i praksis å gjøre tallene i fourierspekteret om til bare reelle tall, siden $\sin(\phi = 0) = 0$. Programmet for denne oppgaven ligger som vedlegg 4. Plott for tidsfunksjoner har jeg i figur 5 og 6.



Figur 5: Tidsfunksjon før og etter nullstilling av faser for 'YelloBostich2.wav'



Figur 3: Tidsplot etter nullstilling av faser for 'lydfil2.wav'

Vi kan tydelig høre forskjell på lydene etter nullstilling av fasene. Vi hører en slags gjentakelse (det ser vi jo på tidsplottet, det er en speiling om midten). Men det er her mulig å gjenkjenne lyden i motsetning til randomiseringen hvor trommelyden ble til ren støy. Lyden blir allikevel svært forandret!

Konklusjon:

Vi ser tydelig at modifisering av faser er hørbart. Det var en stor forskjell på resultatet etter de samme forandringene på de to lydene. Trommeslagene er jo ikke en rent sammenhengende lyd, det er mer to 'pulser' av lyder (hvert slag), og frekvensspekteret blir derfor ganske mye mer komplisert og består av langt flere frekvenser enn den andre lyden. I trommelyden har vi flere samplinger med svært lite lyd, og det krever derfor mange forskjellige frekvenser til for å kunne gjenskape både lyd og stillhet ved rene sinus- og cosinus-funksjoner. Det er derfor randomisering av faser for trommeslagene blir såpass annerledes, faseinformasjonen er kjempeviktig for at de forskjellige funksjonene (med forskjellige frekvenser) skal utlikne hverandre og skape tidspunkt med lite lyd og samtidig lage trommelydene riktig.

KODEVEDLEGG:

Oppgave 1:

```
function oppg1(pl,extraPhase)
close all; %Lukk alle figurer

Fs = 22050; %Samplingsfrekvens
tid = 1; %Lengde i sekunder
N = Fs*tid; %Antall samplinger totalt

t = linspace(0,tid,tid*Fs);
D2 = zeros(1,Fs*tid/2);

%Bestem frekvens
f1 = 884;
f2 = 880;

%Generer vinkelfrekvens
w1 = 2*pi*f1;
w2 = 2*pi*f2;

%Her lager jeg en sammenhengene lyd med to frekvenser (enten samme
to
%ganger, eller to forskjellige)
soundSame = zeros(tid*Fs,1); %Tom plass til lyd
soundDifferent = zeros(tid*Fs,1);

midtNedre = floor(N / 2);
midtOvre = midtNedre+1; %Neste sample

soundSame(1:midtNedre) = sin(w1*t(1:midtNedre));
soundSame(midtOvre:N) = sin(w1*t(midtOvre:N));

soundDifferent(1:midtNedre) = sin(w2*t(1:midtNedre));
phase = asin(soundDifferent(midtNedre)) - (w1*t(midtNedre));
soundDifferent(midtOvre:N) = sin(w1*t(midtOvre:N)+phase+extraPhase);

%Gjør klar variabler for de 16 avspillingene
i = 1;
riktig = 0;
feil = 0;
y = 'y';
n = 'n';
a = 0;
b = 0;
z = zeros(1,16);

if(pl == 1)
    %Plot rundt midtpunktet for å se om fasen må forskyves med
ytligere pi
    %radianer
    plot(t(Fs*0.5-100:Fs*0.5+100),soundSame(Fs*0.5-100:Fs*0.5+100))
    %plot(t,soundSame)
    title('Samme lyd');
    figure
    plot(t(Fs*0.5-100:Fs*0.5+100),soundDifferent(Fs*0.5-
100:Fs*0.5+100))
    %plot(t,soundDifferent)
    title('Annen lyd');
```

```
end
input('Trykk når du er klar...');

while(i<=16)
    r = rand(1);
    if(r>0.5)
       wavplay(soundSame,Fs);
       a = a + 1;
       z(i) = 1;
    else
       wavplay(soundDifferent,Fs);
       b = b + 1;
    end

    ans = 2; %Random value != y || n
    while(ans ~= y && ans ~= n)
        ans = input('Was that the same sound? (y/n)');
    end

    if((ans == y && r>0.5) || (ans == n && r<=0.5))
       riktig = riktig + 1;
    else
        feil = feil + 1;
    end
    i = i + 1
end
('Riktig:')
riktig
('Galt:')
feil
end
```

Oppgave 2:

```
function oppg2()
Fs = 22050; %Samplingsfrekvens
t2 = 0.5; %Totalt tid i s
t1 = 20 * (1/1000); %ms
pauseTime = 0.2;
t_1 = linspace(0,t1,t1*Fs);
t_2 = linspace(0,t2,t2*Fs);
D1 = zeros(1,t1*Fs);
D2 = zeros(1,t2*Fs);
D3 = zeros(1,t1*Fs);

%Bestem frekvens
f1 = 440;
f2 = 444;

%Generer vinkelfrekvens
w1 = 2*pi*f1;
w2 = 2*pi*f2;
D1 = sin(w1*t_1); %Lag lyd med den ene frekvensen
D2 = sin(w1*t_2); %Lag lyd med den ene frekvensen
D3 = sin(w2*t_1);

%Her lager jeg en sammenhengene lyd med to frekvenser (enten samme
to
%ganger, eller to forskjellige)
```

```
soundSame = 0;%Nullstill zeros(length(D1) + length(D2) +
2*Fs*pauseTime,1); %Tom plass til lyd
soundDifferent = 0; %zeros(length(D3) + length(D2) +
2*Fs*pauseTime,1);

%Regn ut plasseringer av lydene i lydsekvensen
shortSoundStart = Fs*pauseTime+1;
shortSoundEnd = shortSoundStart + length(D1) - 1;
longSoundStart = shortSoundEnd + Fs*pauseTime + 1;
longSoundEnd = longSoundStart + length(D2) - 1;

%Lag lydene
soundSame(shortSoundStart:shortSoundEnd) = D1;
phase = asin(soundSame(shortSoundEnd)); % - w1*t_1(shortSoundEnd);
%Beregn fase for overgangen
D2 = sin(w1*t_2+phase); %Lag ny lyd
soundSame(longSoundStart:longSoundEnd) = D2;

soundDifferent(shortSoundStart:shortSoundEnd) = D3;
phase = asin(soundDifferent(shortSoundEnd)); %Beregn fase for
overgangen
D2 = sin(w1*t_2+phase); %Lag ny lyd
soundDifferent(longSoundStart:longSoundEnd) = D2;

i = 1;
riktig = 0;
feil = 0;
y = 'y';
n = 'n';
a = 0;
b = 0;
z = zeros(1,16);
input('Trykk når du er klar...');

while(i<=16)
    r = rand(1);
    if(r>0.5)
        wavplay(soundSame)
        a = a + 1;
        z(i) = 1;
    else
        wavplay(soundDifferent)
        b = b + 1;
    end

    ans = 2;

    while(ans ~= y && ans ~= n)
        ans = input('Was that the same sound? (y/n)');
    end

    if((ans == y && r>0.5) || (ans == n && r<=0.5))
       riktig = riktig + 1;
    else
        feil = feil + 1;
    end
    i = i + 1
end
('Riktig:')
riktig
```

```
('Galt:')
feil
end
```

Oppgave 3a:

```
function oppg3()
    m = 'Sound (wav) file';
    c = 'YelloBostich2.wav'; %nstart=8000

    N = 1024*64; %2^n for at det skal gå fort
    nstart = 8000; %Hvor i lydfila starter vi?

    [y,Fs,type] = wavread(c);
    nslutt = length(y); %Hvor slutter vi?
    length(y)
    y = y(nstart:nslutt); %kutt lydsnutten

    %Fyll ut lydarrayet vårt, begrens oss til 2^n
    if(length(y) > N)
        sound = y(1:N);
    else
        sound = zeros(1,N);
        sound(1:length(y)) = y;
    end

    %Total tid lydsnuttet tar i sek
    T = N/Fs;
    t = linspace(0,T,N); %lag tidsarray for tidsromplot
    hold off
    plot(t,sound) %plot lyden FØR transformasjon
    title('Opprinnelig tidsfunksjon (YelloBostich2.wav)');
    xlabel('Tid [s]');
    ylabel('Relativt signal');

    wavplay(sound,Fs) %Spill av lyden FØR transformasjon
    G = fft(sound,N)/N; %Utfør FFT
    m = abs(G); %Beregn absoluttverdien til de transformerte
(komplekse) tallene
    M = N;
    f_max = N/T;
    f = linspace(0,f_max*(N-1)/N,N);
    r = ifft(G,N)*N;

    figure
    plot(t,r)
    title('Tilbaketransformert tidsfunksjon (YelloBostich2.wav)');
    xlabel('Tid [s]');
    ylabel('Relativt signal');
    norm(sound.^2-r.^2) %Beregn normen til differansvektoren mellom
lydene

    figure
    plot(f(1:M/2),2*m(1:M/2),'s');
    wavplay(r,Fs)
    title('Foureier-spekteret (amplitude for hver frekvens)');
    xlabel('Frekvens (Hz)');
    ylabel('Relativt styrke');
end
```

Oppgave 3b)

```
function oppg3b
    close all;
    format long

    m = 'Sound (wav) file';
    %c = 'lydfil2.wav'; %nstart=8000
    c = 'YelloBostich2.wav';
    N = 1024*64; %2^n for at det skal gå fort
    nstart = 8000; %Hvor i lydfila starter vi?

    [y,Fs,type] = wavread(c); %Les inn lydfil
    nslutt = length(y); %Hvor slutter vi?
    y = y(nstart:nslutt); %kutt lydsnutten

    %Fyll ut lydarrayet vårt, begrens oss til 2^n
    if(length(y) > N)
        sound = y(1:N);
    else
        sound = zeros(1,N);
        sound(1:length(y)) = y;
    end

    T = N/Fs; %Total tid lydsnuttet tar i sek
    t = linspace(0,T,N); %lag tidsarray for tidsromplot

    G = fft(sound,N)/N; %Utfør FFT

    G_rand = randomizePhase(G); %Randomiser faser

    m = abs(G); %Beregn absoluttverdien til de orginale
transformerte (komplekse) tallene
    m_rand = abs(G_rand); %Beregn absoluttverdien til de
randomiserte transformerte (komplekse) tallene
    M = N;
    f_max = N/T;
    f = linspace(0,f_max*(N-1)/N,N);

    sound_ifft = real(ifft(G_rand,N))*N; %Transformer tilbake

    plot(t,sound) %plot lyden FØR transformasjon
    title('Opprinnelig tidsfunksjon');
    xlabel('Tid [s]');
    ylabel('Relativt signal');

    figure
    plot(f,real(G))
    hold on
    plot(f,imag(G),'r')
    figure
    %end

    plot(t,sound_ifft) %plot lyden ETTER transformasjonen

    title('Tilbaketransformert tidsfunksjon');
    xlabel('Tid [s]');
    ylabel('Relativt signal');

    figure
```

```
    plot(f(1:M/2),2*m(1:M/2),'s');
    %plot(f,2*m,'s');
    title('Foureier-spekteret før randomisering (amplitude for hver
frekvens)');
    xlabel('Frekvens (Hz)');
    ylabel('Relativt styrke');

    figure
    plot(f(1:M/2),2*m_rand(1:M/2),'s');
    %plot(f,2*m_rand,'s');
    title('Foureier-spekteret etter randomisering (amplitude for
hver frekvens)');
    xlabel('Frekvens (Hz)');
    ylabel('Relativt styrke');
end

function G_rand = randomizePhase(G)
    G_rand = zeros(1,length(G));
    for i=1:length(G)
       phi = rand(1)*2*pi;
       A = sqrt(real(G(i))^2 + imag(G(i))^2);

       a = A*cos(phi);%*(real(G(i))/abs(real(G(i))));
       b = A*sin(phi);%*(imag(G(i))/abs(imag(G(i))));
       G_rand(i) = a + b*sqrt(-1);
    end
end
```

Oppgave 3c)

```
function oppg3c
    close all;
    format long

    m = 'Sound (wav) file';
    c = 'lydfil2.wav'; %nstart=8000
    %c = 'YelloBostich2.wav';
    N = 1024*64; %2^n for at det skal gå fort
    nstart = 8000; %Hvor i lydfila starter vi?

    [y,Fs,type] = wavread(c); %Les inn lydfil
    nslutt = length(y); %Hvor slutter vi?
    y = y(nstart:nslutt); %kutt lydsnutten

    %Fyll ut lydarrayet vårt, begrens oss til 2^n
    if(length(y) > N)
        sound = y(1:N);
    else
        sound = zeros(1,N);
        sound(1:length(y)) = y;
    end

    T = N/Fs; %Total tid lydsnuttet tar i sek
    t = linspace(0,T,N); %lag tidsarray for tidsromplot

    G = fft(sound,N)/N; %Utfør FFT

    G_rand = randomizePhase(G); %Randomiser faser
```

```
    m = abs(G); %Beregn absoluttverdien til de orginale
transformerte (komplekse) tallene
    m_rand = abs(G_rand); %Beregn absoluttverdien til de
randomiserte transformerte (komplekse) tallene
    M = N;
    f_max = N/T;
    f = linspace(0,f_max*(N-1)/N,N);

    sound_ifft = real(ifft(G_rand,N))*N; %Transformer tilbake

    plot(t,sound) %plot lyden FØR transformasjon
    title('Opprinnelig tidsfunksjon');
    xlabel('Tid [s]');
    ylabel('Relativt signal');

    figure
    plot(t,sound_ifft) %plot lyden ETTER transformasjonen
    title('Tilbaketransformert tidsfunksjon');
    xlabel('Tid [s]');
    ylabel('Relativt signal');

    figure
    plot(f(1:M/2),2*m(1:M/2),'s');
    title('Foureier-spekteret før randomisering (amplitude for hver
frekvens)');
    xlabel('Frekvens (Hz)');
    ylabel('Relativt styrke');

    figure
    plot(f(1:M/2),2*m_rand(1:M/2),'s');
    title('Foureier-spekteret etter randomisering (amplitude for
hver frekvens)');
    xlabel('Frekvens (Hz)');
    ylabel('Relativt styrke');

    wavplay(sound,Fs) %Spill av lyden FØR transformasjon
    pause(0.5)
    wavplay(real(sound_ifft),Fs)
end

function G_rand = randomizePhase(G)
    G_rand = zeros(1,length(G));
    for i=1:length(G)
       phi = 0;
       A = sqrt(real(G(i))^2 + imag(G(i))^2);

       a = A*cos(phi);
       b = A*sin(phi);
       G_rand(i) = a + b*sqrt(-1);
    end
end
```