Komprimering av ljudsignaler

Ett projekt i kursen Signaler & System
given vid Uppsala Universitet under höstterminen 2002

Utfört av David Nilsson-Jatko
Pnr 770710-1957, email dani4965@student.uu.se

1. Innehållsförteckning

2. Sammanfattning

Uppgiften, att utveckla metoder för komprimering och dekomprimering av tal- och ljudsignaler har genomförts med mycket gott resultat. Implementeringen av dessa resulterade i verktyg för komprimering och uppspelning av det samtidigt nyskapade ljudformatet DP3. Funktionen bygger på att bl.a. dela upp en ljudvåg i många småblock och skala bort icke intressant information i både frekvens och amplitud hos vart och ett. En komprimeringsfaktor mellan 5 och 10 har uppmätts för slumpmässigt valda signaler av intressant karaktär, vilket är ett mycket gott resultat.

3. Inledning & bakgrund

Detta projekt har utförts som en del i kursen Signaler & System, given vid Uppsala Universitet under höstterminen 2002. Syftet med projektet har varit att bekanta sig med något område inom signalbehandling som har anknytning till kursen.

Jag, David Nilsson-Jatko, valde att arbeta med talsignaler.

Om mänskligt tal samplas med 44,1 kHz i 16 bitar, så fyller ju detta sällan upp hela signalens spektrum, utan innehåller väldigt mycket redundans som kan skalas bort för att erhålla en mindre datamängd, men med bibehållen meningsfull information. Detta kan t.ex. ske genom att minimera antalet nödvändiga frekvenser, nödvändig signalstyrka, etc. vid olika tidpunkter i signalen. Sådana komprimeringar finns i nästan oändligt många utföranden i olika redan existerande ljudformat.

Att sätta sig in i ljudkomprimeringsområdet, och utveckla och implementera något eget inom området verkade som ett lämpligt projekt att ta sig an i kursen.

4. Problemformulering

Uppgiften var, att givet en signal i wav-form bestående av samplat mänskligt tal, komprimera denna m.h.a. egenutvecklade rutiner till en fil av mindre storlek utan större avkall på mängden viktig (hörbar) information. Dessutom skulle en avkodare skapas, vilken skulle dekomprimera den skapade filen och åtkådliggjorde denna på lämpligt vis, för jämförelse med originalet. Lämpliga algoritmer utvecklas och implementeras. Slutprodukten skulle alltså bli en samling algoritmer, som alla verkar på den givna indatan och utnyttjar olika egenskaper hos samplat tal och andra former av ljud såsom musik, etc. för att minska redundandsen och därmed även datamängden så mycket som möjligt. Algoritmerna väljs efter en bedömning av vilka och hur många som kan anses lämpligt för uppgiften.

5. Teori

5.1. Funktionsval

En kombination av olika tekniker valdes för att lösa problemet. Dessa valdes utifrån att ha studerat principerna för andra ljudformat samt kunskaperna hos projektmedlemmen.

5.2. Funktionsbeskrivning

5.2.1. Komprimering

Komprimeringen fungerar i stora drag på följande vis:

- Öppna den fil som ska innehålla den komprimerade signalen för skrivning.
- Dela upp den givna insignalen (I) i block av längden (L).
- För varje block, gör följande:
    1. Ta fram frekvensspektrum för blocket
    2. Använd Welchs metod för att ta bort störande varians och utjämna spektrumet
    3. Betrakta det erhållna spektrat i logaritmisk form
    4. Finn den högsta frekvensen (F), vars signalstyrka är >= en given tröskelkonstant (T).
    5. Lågpassfiltrera bort frekvenser över F Hz.
    6. Sampla ner datan i blocket till Fx2 Hz.
    7. Övergå till tidsdomänen, och finn den största amplituden (A).
    8. Normalisera hela blockets amplitud med avseende på A.
    9. Minska noggranheten för varje sampels upplösning från 16 bitar till 8 bitar.
    10. Skriv den nersamplade datan, dess frekvens, dess amplitudnormaliseringskonstant (A) och dess längd till filen.

5.2.2. Avkomprimering

Dekomprimering av av den ovan beskrivna algoritmen producerade utdatan fungerar på följande vis:

- Öppna den fil som innehåller den komprimerade signalen
- Så länge som filen inte är slut, läs in information om för frekvens, längd, amplitudnormaliseringskonstant och data blockvis.
- För varje block, gör följande:
    1. Uppsampla datablocket från den inlästa frekvensen till 44,1 kHz (standard-CD-kvalitet) genom interpolering.
    2. Ändra varje sampels amplitudupplösning från 8 bitar till 16 bitar.
    3. Kompensera för amplitudnormaliseringen genom att multiplicera varje sampels amplitudvärde med inversen till A.
    4. Foga ihop det erhållna datablocket med tidigare uppackade block
- Spela upp hela slutliga erhållna datan för åskådliggöring.

5.3. Bakgrund

Metoden förutsätter att användaren vid komprimering specificerat tröskelkonstanten T (vilken tjänar som ett mått på vilken kvalitet som är önskvärd), blocklängden (L) samt källa och destination för in- och utdata.

Enligt Nyqvistteoremet kan varje signal som innehåller högsta frekvensen F_{max} entydigt beskrivas med en samplingsperiod på F_{max} x 2. Högre samplingsfrekvenser tillför endast redundans, varpå frekvensen F_{max} x 2 räcker för att kunna återskapa den nersamplade signalen entydigt och korrekt.

Om man skulle ha betraktat frekvensspektrat för hela signalen istället för blockvis, skulle den högsta frekvensen F som uppnår tröskelkonstanten T för denna datamängd ha varit densamma som det största värdet som uppnår tröskelkonstanten för alla delblock. Detta är inte effektivt, då vi strävar efter ett så lågt värde på F som möjligt för att minska datamängden. Därför används uppdelningen av den totala datamängden i delblock, vilket ökar redundansborttagningen dramatiskt.

Den ändring i amplitudupplösning från 16 till 8 bitar som utföres behöver inte alls medföra någon kvalitetsförsämring. Då en talsignals amplitud oftast inte sträcker sig över hela sitt möjliga område, motverkar ofta amplitudnormaliseringen upplösningsändringen, varpå kvalitén ej minskas även om datamängden minskas. Även denna åtgärd drar stor fördel av blockvis behandling av den totala signalen.

Ovanstående algoritmer begränsar alltså inte användningen till att endast operera på ljuddata i talområdet, utan kan med hjälp av sitt variabla frekvensval även komprimera musik och andra ljudsignaler.

6. Utförande

Till uppgiften valdes den mest emminente IT-ingenjören, David Nilsson-Jatko. Eftersom det finns enormt många olika tekniker man kan använda för att lösa problemet, skulle man egentligen kunna arbeta på projektet i all evighet, men här nöjdes med att implementera ovan beskrivna metoder. Efterforskningar om hur man skulle kunna lösa problemet och med hjälp av vilka algoritmer gjordes. Bland annat studerades olika befintliga ljudformats uppbyggnad, bl.a. MP3 och dess uppdelning av datan i s.k. "frames" eller "block".

Efter en tids efterforskning ritades den schematiska funktionen hos projektets slutprodukt upp, och ovan nämnd funktionalitet (se kapitlet "Teori") fastställdes. Allting beslöts utföras med hjälp av Matlab (version 6.1); såväl utvecklingen som implementeringen av samtliga nödvändiga algoritmer. Detta val togs p.g.a. Matlabs utvecklingsvänliga miljö och breda inbyggda stöd för signal- och matrismanipulering.

Det framtagna nya ljudformatet döptes till DP3, såväl en förkortning av DavidP3 efter den geniale IT-ingenjör som uppfann det hela, som en referens till andra ljudformat som MP3. Implementeringen gjordes i form av ett DP3-enkodningsverktyg och en separat DP3-uppspelare.

För filtrering av ljudsignalen i nersamplingsledet används ett Butterworthfilter av ordning 8, då detta gav mycket tillfredsställande resultat vid utprovningar och tester. Vid analys av frekvensspektrum används ett enkelt hammingfönster som fönsterfunktion (datamängden är ju inte oändlig).

7. Resultat

Till sist hade två produkter för komprimering och dekomprimering av ljudsignaler framställts, med namnet DP3make och DP3play. Funktionen visade sig vara över förväntan. Utan märkbar (hörbar) kvalitetsskillnad kunde en talsignal i vissa fall förminskas med en faktor 10, och i allmänna fall med ungefär en faktor 5. Detta är att jämföra med t.ex. MP3, som i allmänhet ger en förminskning med en faktor 10, men som å andra sidan är enormt mycket mer komplex. Givetvis är prestandan helt beroende av insignalens komplexitet och natur, men i detta projekt har dock bara "vanliga" signaler som är intressanta som tal- eller musiksignaler ansetts viktiga, helt enligt specifikationerna.

Vid små blockstorlekar kan dock störningar orsakade av sammanfogningarna av de olika blocken märkas, men detta är inget strukturellt problem med den bakomliggande teorin, utan kan åtgärdas vid vidare utveckling av projektet. Som sagt kan man hålla på i evigheter med att finputsa och implementera bättre och mer avancerade algoritmer, men gränsen dras här, då det hittills erhållna resultatet mer än väl torde motsvara förväntningarna.

8. Slutsats

Vi har alltså lyckats implementera en ljudkomprimering som minskar datamängden med ungefär en faktor 5. Givetvis beror kvalitén och komprimeringsfaktorn på insignalens natur, men för denna tillämpning har endast "normalt" tal och musik ansetts intressanta. För dessa erhålles också en mycket tillfredsställande komprimering.

9. Appendix A: Användarmanual

9.1. dp3make

Syntax för användning av DP3-enkodaren är i Matlab följande:

```
dp3make(infil,utfil,kvalitet,blockstorlek,detaljer)
```

där infil är den wav-fil (44 kHz, 16 bit, mono) vi vill komprimera,
utfil är destinationsfilnamnet (*.dp3),
kvalitet är tröskelgräns för bortklippning av skräp (rekommenderat: 26),
blockstorlek blockstorlek som används vid komprimering (rekommenderat: 10000) och
detaljer: 0 för normal operation, 1 för att visa komprimeringsinfo.

Funktionen kräver att signalbehandlingspaketet är installerat i Matlab. Funktionen är testad med Matlav 6.1 och funkar fint med denna version.

9.2. dp3play

Syntax för användning av DP3-uppspelaren i Matlav är följande:

```
dp3play(filnamn,superkvalitet,diagram,detaljer)
```

där filnamn är den dp3-fil som ska spelas upp,
superkvalitet är 0 för snabbhet eller 1 för kvalitet,
diagram är 0 för normal användning, 1 för att visa diagram i slutet el. 2 för högdetaljerat och
detaljer är 0 för vanlig operation eller 1 för att visa dekomprimeringsinfo

Funktionen kräver att signalbehandlingspaketet är installerat i Matlab. Funktionen är testad med Matlav 6.1 och funkar fint med denna version.

För uppspelning av 44,1 kHz .wav-fil för jämförelse med DP3 är syntaxen:

```
wavplay(wavread(filnamn),44100)
```

10. Appendix B: Källkod

10.1. dp3make.m

```
function [] = dp3make(filnamn,utfil,troskel,chunkstorlek,detaljer)
% Syntax: dp3make(infil,utfil,kvalitet,blockstorlek,detaljer)
% infil        = den wav-fil (44 kHz, 16 bit, mono) vi vill komprimera
% utfil       = destinationsfilnamn (.dp3)
% kvalitet     = tröskelgräns för bortklippning av skräp (rekomm: 26)
% blockstorlek = blockstorlek som används vid komprimering (rekomm: 10000)
% detaljer     = 0 för normal operation, 1 för att visa komprimeringsinfo



% demonstration=1 el. 0 berornde på om man vill ha uppspelning
demonstration=0;
%filnamn='c:\m\fsol2.wav';
[total_data,ursp_frek]=wavread(filnamn);
datalangd=length(total_data);
%chunkstorlek=10000;
chunknummer=0;
%troskel=kvalitet;
disp('========================================================================')
disp('DP3 Encoder')
disp('========================================================================')
disp(sprintf('Total datalängd är %d med frekvens %d Hz',length(total_data),ursp_frek))
disp(sprintf('Chunkstorleken är %d samples',chunkstorlek))
disp(sprintf('Komprimeringströskeln (-kvaliteten) är %d',troskel))
disp('Komprimerar...')

fut=fopen(utfil,'w');

utfilstorlek=0;
while (chunknummer+1)*chunkstorlek<=datalangd

    % hämta ett nytt block av indatan att arbeta på
    if chunknummer*chunkstorlek<=datalangd-chunkstorlek
        dennachunkstorlek=chunkstorlek;
    else
        dennachunkstorlek=datalangd-chunknummer*chunkstorlek;
    end
    ursp_data=zeros(dennachunkstorlek,1);
    for t=1:dennachunkstorlek
        ursp_data(t)=total_data(chunknummer*chunkstorlek+t);
    end


    % luska ut vilka frekvenser vi kan klippa bort
    klippfrek=hamtafrekvens(ursp_data,ursp_frek,troskel,0);
    if klippfrek==0
        % om framen inte uppfyller kriterierna för att hitta frekvenströskeln i operationen
ovan
        utdata=ursp_data;
        nyfrek=ursp_frek;
    else
        % lågpassfiltrera bort höga, ointressanta frekvenser
        lagpassfiltrerat=lagpass(ursp_data,ursp_frek,klippfrek,0);
        nyfrek=klippfrek*2;
        % omsampla datan enligt den nya frekvensen
        utdata=omsampla(lagpassfiltrerat,ursp_frek,nyfrek);
        if detaljer==1
            disp(sprintf('Klippfrek. %5d Hz -> Datalängd %5d vid %5d Hz (%3d proc.
vunnet)',klippfrek,length(utdata),nyfrek,ceil(100*(1-length(utdata)/length(ursp_data)))))
        end
    end
    utfilstorlek=utfilstorlek+length(utdata);

    % vilket element har störst amplitud?
    storsta=0;
    for t=1:length(utdata)
        if abs(utdata(t))>storsta
            % normalisera amplituden m.a.p. det sampel i blocket med högst amplitud
            storsta=abs(utdata(t));
        end
    end
```

```
    % skriv all data om blocket till utfilen
    fwrite(fut,nyfrek,'integer*2');
    fwrite(fut,storsta,'real*4');
    fwrite(fut,length(utdata),'integer*2');
    fwrite(fut,int16(utdata*(128/storsta)),'integer*1');

    chunknummer=chunknummer+1;
end
fclose(fut);
disp(sprintf('Ursprunglig datastorlek:   %d bytes',datalangd*2))
disp(sprintf('Komprimerad datastorlek:   %d bytes (%d procent av
originalet)',utfilstorlek,round(100*utfilstorlek/(datalangd*2))))
disp(sprintf('Komprimeringseffektivitet: %d procent',100-
round(100*utfilstorlek/(datalangd*2))))

% klart
```

10.2. dp3play.m

```
function [] = dp3play(filnamn,kval,detaljer,visainfo)
% Syntax: dp3play(filnamn,superkvalitet,diagram,detaljer)
% filnamn       = den dp3-fil som ska spelas upp
% superkvalitet = 0 för snabbhet
%                 1 för kvalitet
% diagram       = 0 för normal användning
%                 1 för diagram i slutet
%                 2 för jättemånga detaljer (jobbigt!)
% detaljer      = 0 för vanlig operation
%                 1 för att visa dekomprimeringsinfo



% demonstration=1 el. 0 berornde på om man vill ha uppspelning
demonstration=0;

disp('========================================================================')
disp('DP3 Player')
disp('========================================================================')
totaltinlast=[];
blocknummer=0;
fin=fopen(filnamn,'r');
disp('Dekomprimerar...')

% läs in datablock så länge filen inte är slut
while feof(fin)==0
    [frek,count]=fread(fin,1,'integer*2');
    if count>0
        blocknummer=blocknummer+1;
        [storsta,count]=fread(fin,1,'real*4');
        [langd,count]=fread(fin,1,'integer*2');
        [datablock,count]=fread(fin,langd,'integer*1');
        if visainfo==1
            disp(sprintf('BLOCK %2d: %5d Hz, %5d samples, normaliseringsvärde
%1.4f',blocknummer,frek,langd,storsta))
        end

        % kompensera för komprimeringens amplitudnormalisering och amplitudupplösning
        datablock2=double(datablock)/(128/storsta);
        % uppsampla datan till 44,1 kHz
        if kval==1
            uppsamplat=resample(datablock2,44100,frek)';
        else
            uppsamplat=uppsampla(datablock2,frek,44100);
        end
        % lägg till bearbetat block till de andra, redan bearbetade
        totaltinlast=[totaltinlast uppsamplat];
        if detaljer==2
            wavplay(datablock2,frek,'sync');
            disp('Inläst ljuddata plottas - tryck på någon tangent för att fortsätta')
            plot(datablock2)
            pause;
        end
    end
end
fclose(fin);

% spela upp
disp('Spelar upp...')
wavplay(totaltinlast,44100,'sync');

% visa ev. detaljer
if detaljer>0
    disp('Tids- och frekvensspektrum visas.')
    plot(totaltinlast)
    plottafrekvens(totaltinlast,44100)
end

% slut
```

10.3. hamtafrekvens.m

```
function [utfrekvens] = hamtafrekvens(chan1,dt,tolerans, plotta)

echo off
if plotta==1

    % skapa nytt fönster att smacka ut grafik i
    chan1figtitle = ['Tidsdomän'];
    hf = figure('Name',chan1figtitle,'NmberTitle','on','Color','white',...
        'Pointer','crosshair','Units','normal','Position',[0.1,0.07,0.8,0.35],...
        'InvertHardCopy','off','PaperType','a4letter','PaperUnits','centimeters',...
        'PaperPosition',[3.0,10.0,14.0,12.0]);

    % normalisera datan m.a.p. amplitud
    ymax = 1.2*max(chan1);
    ymin = 1.2*min(chan1);
    if ymax<0.0
        ymax = ymax/(1.2*1.2);
    end
    if ymin>0.0
        ymin = ymin/(1.2*1.2);
    end

    npnts = length(chan1);
    xmin = 0.0;
    xmax = npnts*dt;
    t = 0:dt:dt*(npnts-1);

    % skapa axlar till fönstret
    ht = axes('Parent',hf,'Position',[0.1 0.2 0.85 0.75]);

    % plotta tidsdata
    plot(t,chan1,'b');

    % skapa axlelparametrar
    set(ht,'XLim',[xmin xmax],'XColor','black');
    set(ht,'YLim',[ymin ymax],'YColor','black');

    htx = text(0,0,'Time (sec)');
    set(htx,'Color','black');

    hty = text(0,0,'Amplitude');
    set(hty,'Color','black');
    set(hty,'Rotation',+90.0);

    set(ht,'XLabel',htx,'YLabel',hty);
end




% Visa frekvensspektrum
%      chan1 : tidsvektor
%      lwin  : längden av tidsfönsterfunktion
%      dt    : samplingsfrekvensen

lwin = 2048;

% vi ska loopa genom datan ett antal ggr
n = fix(length(chan1)/lwin);

% skala om tidsvektorn till en matris med n kolumner och längd lwin
x = reshape(chan1(1:lwin*n),lwin,n);
xwin = zeros(size(x));

% kolumner för fönstret -- Hammingfönster används.
win = .5*(1 - cos(2*pi*(1:lwin)'/(lwin+1)));;
for k=1:n
    xwin(1:lwin,k) = win.*x(:,k);
end

% fouriertransforma
f = fft(xwin);
```

```
% vi vill inte ha negativa frekvenser!
f(1+lwin/2:lwin,:) = [];


ps = abs(f).^2;

% Summera kolumner & ta medlevärde
[m,n] = size(ps);
if (m==1) | (n==1)
    psa = ps*(dt/lwin);
    psa(2:lwin/2) = psa(2:lwin/2)*2;
else
    psa = sum(ps')*(dt/lwin);
    psa(2:lwin/2) = psa(2:lwin/2)*2;
end

% Visa frekvensspektrat med linjär x-axel och logaritmisk y-axel
df = 1/(dt*lwin);
np = lwin/2;
f = 0:df:(np-1)*df;

if plotta==1

    % skalade axlar
    mn = min(10*log10(psa(1:np)));
    if mn<0.0,
        mn=mn*1.2;
    else
        mn=mn/1.2;
    end
    mx = max(10*log10(psa(1:np)));
    if mx>0.0,
        mx=mx*1.2;
    else
        mx=mx/1.2;
    end

    chan1figtitle = ['Frekvensspektrum ',wavefile];
    hf = figure('Name',chan1figtitle,'NumberTitle','on','Color','white',...
        'Pointer','crosshair','Units','normal','Position',[0.3,0.2,0.65,0.65],...
        'InvertHardCopy','off');

    hclx = text(0.6,0.9,'');
    hcly = text(0.93,0.9,'');
    set(hclx,'Color','red');
    set(hcly,'Color','red');

    hcx = text(0.57,0.97,'');
    hcy = text(0.9,0.97,'');
    set(hcx,'Color','blue');
    set(hcy,'Color','blue');

    % position axes within figure window
    %%ht = axes('Parent',hf,'Position',[0.2 0.1 0.75 0.85]);
    %plot(f,10*log10(psa(1:np)),'-b');
end

% Welch's metod för att minimera varians
nyvar=zeros(np,1);
for tt=11:np-11
    nyvar(tt)=(psa(tt-10)+psa(tt-6)+psa(tt-3)+psa(tt)+psa(tt+3)+psa(tt+6)+psa(tt+10))/7;
end
% lite larv för att inte få log av 0
for tt=1:np
    nyvar(tt)=nyvar(tt)+1;
end
desibell=10*log10(nyvar);
utfrekvens=0;
% vi loopar över vår array med steg om 10
for tt=1:10:length(desibell)
    if desibell(tt)>tolerans
        utfrekvens=tt/np*22050;
    end
end
utfrekvens=floor(utfrekvens);
```

```
if plotta==1
    plot(f,desibell,'-b');
end
```

10.4. lagpass.m

```
function [utfrekvens] = hamtafrekvens(chan1,dt,tolerans, plotta)

echo off
if plotta==1

    % skapa nytt fönster att smacka ut grafik i
    chan1figtitle = ['Tidsdomän'];
    hf = figure('Name',chan1figtitle,'NmberTitle','on','Color','white',...
        'Pointer','crosshair','Units','normal','Position',[0.1,0.07,0.8,0.35],...
        'InvertHardCopy','off','PaperType','a4letter','PaperUnits','centimeters',...
        'PaperPosition',[3.0,10.0,14.0,12.0]);

    % normalisera datan m.a.p. amplitud
    ymax = 1.2*max(chan1);
    ymin = 1.2*min(chan1);
    if ymax<0.0
        ymax = ymax/(1.2*1.2);
    end
    if ymin>0.0
        ymin = ymin/(1.2*1.2);
    end

    npnts = length(chan1);
    xmin = 0.0;
    xmax = npnts*dt;
    t = 0:dt:dt*(npnts-1);

    % skapa axlar till fönstret
    ht = axes('Parent',hf,'Position',[0.1 0.2 0.85 0.75]);

    % plotta tidsdata
    plot(t,chan1,'b');

    % skapa axlelparametrar
    set(ht,'XLim',[xmin xmax],'XColor','black');
    set(ht,'YLim',[ymin ymax],'YColor','black');

    htx = text(0,0,'Time (sec)');
    set(htx,'Color','black');

    hty = text(0,0,'Amplitude');
    set(hty,'Color','black');
    set(hty,'Rotation',+90.0);

    set(ht,'XLabel',htx,'YLabel',hty);
end




% Visa frekvensspektrum
%      chan1 : tidsvektor
%      lwin  : längden av tidsfönsterfunktion
%      dt    : samplingsfrekvensen

lwin = 2048;

% vi ska loopa genom datan ett antal ggr
n = fix(length(chan1)/lwin);

% skala om tidsvektorn till en matris med n kolumner och längd lwin
x = reshape(chan1(1:lwin*n),lwin,n);
xwin = zeros(size(x));

% kolumner för fönstret -- Hammingfönster används.
win = .5*(1 - cos(2*pi*(1:lwin)'/(lwin+1)));;
for k=1:n
    xwin(1:lwin,k) = win.*x(:,k);
end

% fouriertransforma
f = fft(xwin);
```

```
% vi vill inte ha negativa frekvenser!
f(1+lwin/2:lwin,:) = [];


ps = abs(f).^2;

% Summera kolumner & ta medlevärde
[m,n] = size(ps);
if (m==1) | (n==1)
    psa = ps*(dt/lwin);
    psa(2:lwin/2) = psa(2:lwin/2)*2;
else
    psa = sum(ps')*(dt/lwin);
    psa(2:lwin/2) = psa(2:lwin/2)*2;
end

% Visa frekvensspektrat med linjär x-axel och logaritmisk y-axel
df = 1/(dt*lwin);
np = lwin/2;
f = 0:df:(np-1)*df;

if plotta==1

    % skalade axlar
    mn = min(10*log10(psa(1:np)));
    if mn<0.0,
        mn=mn*1.2;
    else
        mn=mn/1.2;
    end
    mx = max(10*log10(psa(1:np)));
    if mx>0.0,
        mx=mx*1.2;
    else
        mx=mx/1.2;
    end

    chan1figtitle = ['Frekvensspektrum ',wavefile];
    hf = figure('Name',chan1figtitle,'NumberTitle','on','Color','white',...
        'Pointer','crosshair','Units','normal','Position',[0.3,0.2,0.65,0.65],...
        'InvertHardCopy','off');

    hclx = text(0.6,0.9,'');
    hcly = text(0.93,0.9,'');
    set(hclx,'Color','red');
    set(hcly,'Color','red');

    hcx = text(0.57,0.97,'');
    hcy = text(0.9,0.97,'');
    set(hcx,'Color','blue');
    set(hcy,'Color','blue');

    % position axes within figure window
    %%ht = axes('Parent',hf,'Position',[0.2 0.1 0.75 0.85]);
    %plot(f,10*log10(psa(1:np)),'-b');
end

% Welch's metod för att minimera varians
nyvar=zeros(np,1);
for tt=11:np-11
    nyvar(tt)=(psa(tt-10)+psa(tt-6)+psa(tt-3)+psa(tt)+psa(tt+3)+psa(tt+6)+psa(tt+10))/7;
end
% lite larv för att inte få log av 0
for tt=1:np
    nyvar(tt)=nyvar(tt)+1;
end
desibell=10*log10(nyvar);
utfrekvens=0;
% vi loopar över vår array med steg om 10
for tt=1:10:length(desibell)
    if desibell(tt)>tolerans
        utfrekvens=tt/np*22050;
    end
end
utfrekvens=floor(utfrekvens);
```

```
if plotta==1
    plot(f,desibell,'-b');
end
```

10.5. omsampla.m

```
function [filtrerat] = omsampla(data,gammalfrek,nyfrek)

steg=gammalfrek/nyfrek;

%disp(length(data))

filtrerat=zeros(floor(length(data)/steg),1);

for t=1:length(filtrerat)
    uppdelning=(t*steg-floor(t*steg));
    filtrerat(t)=(1-uppdelning)*data(floor(t*steg))+uppdelning*data(ceil(t*steg));
end
```

10.6. plottafrekvens.m

```
function [] = plottafrekvens(chan1,dt)
% visar frekvensspektra



% snarlik funktion finns i hamtafrekvens.m
% se den filen för kommentering!


lwin = 2048;

n = fix(length(chan1)/lwin);

x = reshape(chan1(1:lwin*n),lwin,n);
xwin = zeros(size(x));

win = .5*(1 - cos(2*pi*(1:lwin)'/(lwin+1)));;
for k=1:n
    xwin(1:lwin,k) = win.*x(:,k);
end

f = fft(xwin);
f(1+lwin/2:lwin,:) = [];
ps = abs(f).^2;

[m,n] = size(ps);
if (m==1) | (n==1)
    psa = ps*(dt/lwin);
    psa(2:lwin/2) = psa(2:lwin/2)*2;
else
    psa = sum(ps')*(dt/lwin);
    psa(2:lwin/2) = psa(2:lwin/2)*2;
end

df = 1/(dt*lwin);
np = lwin/2;
f = 0:df:(np-1)*df;

mn = min(10*log10(psa(1:np)));
if mn<0.0,
    mn=mn*1.2;
else
    mn=mn/1.2;
end
mx = max(10*log10(psa(1:np)));
if mx>0.0,
    mx=mx*1.2;
else
    mx=mx/1.2;
end

chan1figtitle = ['Frekvensspektrum '];
hf = figure('Name',chan1figtitle,'NumberTitle','on','Color','white',...
    'Pointer','crosshair','Units','normal','Position',[0.3,0.2,0.65,0.65],...
    'InvertHardCopy','off');

hclx = text(0.6,0.9,'');
hcly = text(0.93,0.9,'');
set(hclx,'Color','red');
set(hcly,'Color','red');

hcx = text(0.57,0.97,'');
hcy = text(0.9,0.97,'');
set(hcx,'Color','blue');
set(hcy,'Color','blue');

nyvar=zeros(np);
for tt=11:np-11
    nyvar(tt)=(psa(tt-10)+psa(tt-6)+psa(tt-3)+psa(tt)+psa(tt+3)+psa(tt+6)+psa(tt+10))/7;
end
desibell=10*log10(nyvar);
plot(f,desibell,'-b');
```

10.7. uppsampla.m

```
function [uppsamplat] = uppsampla(indata,gammalfrek,nyfrek)

%uppsamplat=resample(indata,nyfrek,gammalfrek);

%indata=[1 2 1 4 5]
%gammalfrek=500
%nyfrek=1000

steg=nyfrek/gammalfrek;
uppsamplat=zeros(floor(length(indata)*steg),1)';

borjan=1;
for t=1:length(indata)-1

    borjan=borjan;
    slutet=floor((t+1)*steg);

    antalsteg=slutet-borjan+1;
    for t2=0:antalsteg-1
        %((antalsteg-t2)/antalsteg)
        %(1-((antalsteg-t2)/antalsteg))
        uppsamplat(t2+borjan)=((antalsteg-t2)/antalsteg)*indata(t)+(1-((antalsteg-
t2)/antalsteg))*indata(t+1);
    end

    borjan=slutet;
end
```