Talkompression

- att komprimera och dekomprimera en wave-fil som innehåller tal

Projekt i kursen Signaler och System
ht 2002

Olle Widell – olle@widell.se - 781210-1611
Erik Salomonsson – erik@salomonsson.net - 780906-8518
Kristina Johnsson – krjo8402@student.uu.se - 781007-4026

Sammanfattning

En wav-fil med tal innehåller mycket information, mer än vad som egentligen är nödvändigt. Då den ska överföras mellan två punkter, t.ex. via en telefon, kan detta medföra problem. När man studerar en wav-fil närmare inser man att punkter kan tas bort utan att information förloras. Denna tanke låg till grund för vår undersökning om hur en fil med tal skulle kunna komprimeras utan att väsentlig information förlorades. En första idé var att komprimera ursprungsfilen för att kunna överföra en så liten wav-fil som möjligt, vi kom dock snabbt fram till att betydligt bättre resultat kunde uppnås genom att komprimera ljudet hos sändaren och sedan dekomprimera det hos mottagaren.

För att enkelt kunna testa och jämföra olika metoder skrevs generella funktioner i Matlab. I denna rapport presenteras den grundläggande teorin som ligger bakom de olika metoder som använts i försöken samt resultaten av dessa.

Innehållsförteckning

1 Inledning

En ljudfil av typen wave innehåller mycket information, mer än vad som egentligen behövs för att lagra mänskligt tal. Vår uppgift var att titta på hur man skulle kunna komprimera just en wave-fil för att lättare kunna göra en överföring av den t.ex. via en telefonledning, för att senare dekomprimera filen och få en godtagbar ljudkvalitet. Kravet på ljudkvaliteten är inte hundraprocentig utan vikten läggs på att det utan problem ska gå att höra vad som sägs. Mottagaren använder sig av en dekomprimeringsfunktion för att från den komprimerade filen få en lyssningsbar wave-fil. Det är vid konstruktion av denna dekomprimeringsfunktion som vi lagt tyngdpunkten av vårt arbete. I denna rapport presenterar vi teoretisk bakgrund för de metoder vi använt tillsammans med de lösningar vi funnit bäst. Programkod från Matlab återfinns i bilagor längst bak.

2 Utförande

Vi började med att analysera vår ljudfil med avseende på frekvensinnehållet. Vi testade därefter olika metoder för att komprimera filstorleken, dock med varierande ljudkvalitet som följd. Efterhand fann vi att man under komprimeringsfasen kan ta bort åtminstone fyra av fem datapunkter med fullt godtagbar ljudkvalitet. Man kan ta bort ännu fler datapunkter men får naturligtvis räkna med sjunkande ljudkvalitet. Den ursprungliga inriktningen på projektet var att enbart komprimera en ljudfil utan att i ett senare skede behöva dekomprimera den. Vi ville alltså att den komprimerade varianten fortfarande skulle gå att lyssna på och ha en acceptabel ljudkvalitet. Rätt snabbt kom vi dock fram till att man på detta sätt åstadkom dålig kompression och mycket dåligt ljud.

Istället koncentrerade vi oss på att komprimera ljudfilen för att senare kunna dekomprimera den. En dekomprimeringsfunktion skulle då skapas för att återställa bortplockade värden så att man åter fick en lyssningsbar ljudfil. För att återskapa datapunkter provades några olika metoder. Vi valde att studerade metoden för Linear Prediction Coding (LPC) mer ingående och implementerade denna metod i Matlab.
Som verktyg för att bygga komprimerings- och dekomprimeringsprogrammen användes uteslutande Matlab. Vi spenderade stor del av tiden på att skriva så generella funktioner som möjligt där vi genom inparametrar kunde styra t.ex. hur många punkter som skulle tas bort, läggas till eller interpoleras över. Detta gjorde att vi enkelt kunde testa funktionerna för olika värden och på så sätt jämföra olika metoder.

3 Analys av ljudfiler

När vi tittade på amplitudspektrat för de ljudfiler som vi spelat in visade dessa att största delen av energin återfanns i frekvensbandet från 0 till ca 700 Hz. För att i ett senare skede kunna sampla filerna med lägre samplingshastighet valde vi att lågpassfiltrera dem. Vi kom fram till att ett filter med passband upp till 800 Hz och spärrband från 950 Hz bevarade en tillräcklig ljudkvalitet. Det hade varit möjligt att filtrera bort ännu fler frekvenser men med reducerad ljudkvalitet som följd. Det hade även gått att få en något bättre återgivning av det ursprungliga talet genom att flytta upp filtrets gränsfrekvens. Vi gjorde dock den subjektiva bedömningen att ovanstående värden var optimala för vår fortsatta behandling av ljudfilen.

Då våra filer innehåller tal (inga högre frekvenser än ca 700 Hz) hade det inte varit helt nödvändigt att utföra en lågpassfiltrering. Vi ville dock säkerställa att det inte förekom några störande högfrekventa ljud som eventuellt skulle kunna ge vikningseffekter vid en lägre samplingsfrekvens.

Under arbetet med att ta fram lämpligt lågpassfilter kom vi fram till att filtertypen inte påverkar resultatet av filteringen nämnvärt. Det som skiljer mest mellan olika filtertyper är ordningen (givet specificerade värden för maximalt rippel i passband och hur snabb övergången ska vara från pass- till spärrband samt hur stor dämpningen ska vara i spärrbandet). Filtret som vi senare i arbetet använde oss av var ett Butterworthfilter av ordning 28. I ljudfilerna som bifogas ges några exempel på de tester vi gjorde.

4 Metoder för komprimering

4.1 Borttagning av datapunkter

I Matlab kan man enkelt läsa in en wavefil och spara datainnehållet i en vektor. Genom att ta bort datapunkter med jämna intervall får man som resultat en vektor med betydligt mindre data. När vektorn sedan sparas till en fil blir denna betydligt mindre än den ursprungliga wave-filen. (Om man tar bort fyra av fem datapunkter tar filen upp 20% av den ursprungliga storleken). De komprimerade filerna sparas i mat-format.

4.2 Kvantisering med färre bitar

Denna metod bygger på att man sparar undan varje datapunkt med färre antal bitar än tidigare och får på så vis en mindre fil. Waveformatet använder 16 bitar till att spara varje punkt medan matlab som standard använder 64 bitar per datapunkt. För att inte få en större fil än den ursprungliga wave-filen måste man på grund av detta reducera bitantalet för varje datapunkt. Vi gjorde här försök att spara ner punkterna i den komprimerade filen (med hjälp av matlabkommandot fwrite) med endast 8 bitar, men fann att det inte gick att återskapa en bra ljudkvalitet utifrån 8-bitarsfilen och därför utvecklade vi inte detta mer utan sparade filerna med 16 bitar per datapunkt. Detta medgav en bra ljudkvalitet efter dekomprimering.

5 Metoder för dekomprimering

Dekomprimering innebär att återskapa borttagna datapunkter så att de blir så lika ursprungsvärdena som möjligt, detta kan göras på många olika sätt. Vi valde att titta på några olika former av enklare interpolationsmetoder samt *linear prediction coding* (LPC-metoden). En sak som bör nämnas är att ljudkvaliteten efter dekomprimering kan förbättras avsevärt av att man lågpassfiltrerar den dekomprimerade filen innan man spelar upp den så att man får bort eventuella högfrekventa störningar.

5.1 Utfyllnad av punkter

Det enklaste sättet att fylla i tomma datapunkter består av att man tar punkten före den/de punkter man vill återskapa och kopierar den så många gånger man önskar till de efterliggande punkterna efter. Vi skapade för detta ändamål en funktion där man kan ange hur många punkter man vill återskapa efter varandra. Denna variant gav sämst resultat av de metoder vi testat särskilt då flera värden på rad ska återskapas.

5.2 Medelvärde

Denna metod bygger på att man återskapar en punkt genom att ta medelvärdet av punkten före och punkten efter. Då flera punkter ska återskapas använder man medelvärdet av de omgivande kvarvarande punkterna och lägger in samma värde på alla mellanliggande punkter. Att ta medelvärdet av två punkter gav bra ljudåtergivning. Principen bygger dock på att man alltid känner till framtida värden, vilket kanske inte är fallet om man tänker sig en realtidsapplikation där ljudet ska återskapas direkt.

5.3 Förbättrat medelvärde

I stället för att lägga in samma värde på alla mellanliggande punkter kan man beräkna varje punkt så att man gradvis går från värdet i punkten före de mellanliggande upp till värdet i punkten omedelbart efter. På detta vis slipper man de skarpa hopp som den ursprungliga medelvärdesmetoden ger vilket ger bättre kvalitet. Denna metod implementeras i filen interpol.m (se bilaga).

5.4 Linear Prediction Coding - LPC

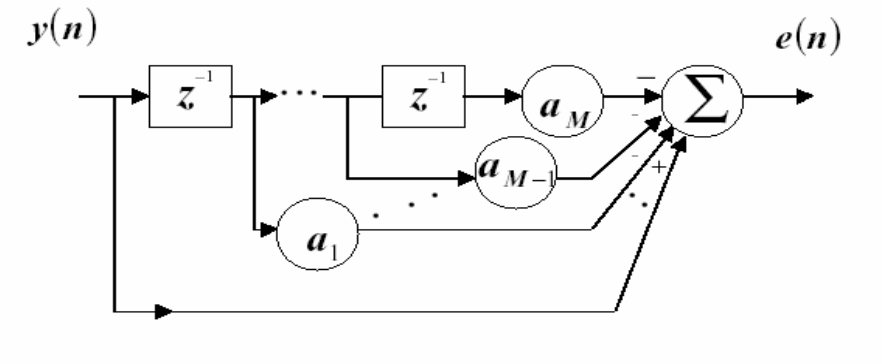
Linear Prediction Coding, LPC, är en mycket vanlig metod inom talkodning. Metoden bygger på att data i en talsignal beror av tidigare data på ett visst sätt. Med hjälp av detta kan man använda en linjärt viktad kombination av M föregående sampel för att räkna ut vilket värde en viss datapunkt ska ha. Det nya värdet (x(m-hatt)) beräknas enligt:

$$\textit{nyttvärde} = \hat{y}(n) = \sum_{k=1}^{N} a_k \, y(n-k)$$

Detta löses genom att minimera felet som uppstår mellan den tidigare punkten och den funna punkten. Minimeringen kan lösas med t.ex. minsta kvadrat-metoden .

Genom att skriva om problemet ytterligare kan det lösas som ett ekvationssystem. Det svåra i beräkningen är att hitta koefficienterna a1, a2,..., aM som används för viktning i summan av de tidigare punkterna

Följande diagram illustrerar LPC:



källa: Spanias A., *http://hitchcock.dlt.asu.edu/media3/a_spanias/dsp-lect1-10/pdf%20files/e506-matlab.PDF*

5.4.1 Vår användning av LPC

Då tyngdpunkten av vårt arbete låg på att komprimera/dekomprimera filerna på olika vis använde vi oss av Matlabs inbyggda lpc-funktion för att finna koefficienterna. En funktion skrevs för att kunna utföra tester genom att använda olika många värden att interpolera över (M) samt olika värden för hur många punkter som används för att skapa koefficienterna. De bästa värdena fås här för höga värden (upp mot 10000) punkter för att skapa koefficienter. I antalet punkter att interpolera över fann vi inga stora skillnader, men ett minimivärde på 10 tiopunkter bör användas. Ett exempel på hur en sådan funktion kan se ut som återfinns som bilaga (lpcinter.m).

6 Kommentarer

Med facit i hand kan man konstatera att en stor del av tiden vi i gruppen lagt ner på detta projekt har gått åt till problem som mera rör programmering i Matlab än signalbehandling. Därmed inte sagt att vi inte har arbetat med ämnen som rör signalbehandling. Däremot hade kunnat använda mer tid till relevanta problem än till rent programmeringstekniska saker om vi hade varit mer hemma på Matlab som program.

Det har varit intressant att sätta sig in i ljudfiler och hur man kan manipulera dem i olika riktningar. Något vi jobbat hårt med är att få LPC-metoden i Matlab att fungera som vi vill. Även efter att själva projektarbetet är avslutat tycker vi att vi borde ha kunnat få ett ännu bättre resultat med hjälp av LPC. Visserligen blir ljudkvaliteten riktigt bra när vi använder oss av LPC men om man jämför med resultaten av de betydligt enklare metoder vi implementerat själva så tycker vi att vi borde fått ett ännu bättre resultat med LPC.

Som helhet tycker vi att vi lärt oss mycket inom området och känner att vi hade velat jobba mera på projektet om våra scheman hade medgett det.

6.1 Möjlig utökning

Man skulle kunna bygga vidare på detta projekt exempelvis genom att försöka implementera andra metoder för komprimering av talat ljud, eller kanske försöka komprimera andra sorters ljud.

7 Referenser:

Cox, Richard V. "Speech Coding", 2000 CR Press LLC *http://www.engnetbase.com*

Johansson Mathias,föreläsningsanteckningar till kursen Signaler och System ht2002
http://www.signal.uu.se/Courses/CourseDirs/SignSyst/main.html

Saeed V. Vaseghi, Advanced digital Signal Processing and Noise Reduction, 2 ed 2001

Spanias Andreas, http://hitchcock.dlt.asu.edu/media3/a_spanias/dsp-lect1-10/pdf%20files/e506-matlab.PDF (2002-12-04), Mars 2002

Svärdström A., Signaler och system, Studentlitteratur, 1999

http://washer.ee.ic.ac.uk/msc_csp/PDF/pan.pdf (2002-12-04)

http://www.cen.uiuc.edu/~ece320/handouts/speech.pdf (2002-12-04)

MATLAB hjälptexter

Kommentar till bilagor

Bifogade Matlabfiler

cut.m
argument: toFile, fromFile, antalKoeff, nrPoints, A, B
Läser in wav-filen (fromFile), behåller vart "nrPoints" värde och skriver ut till toFile som är av typen .mat. Vektorerna A och B beskriver lågpassfiltret som filtrerar den ursprungliga wav-filen. Parametern antalKoeff är antalet koefficienter som kommer att användas vid LPC-dekomprimeringen.

interpol.m
argument: toFile, fromFile, nrPoints, A, B
Läser in från en .mat fil (fromFile). Skapar nrPoints-1 nya värden mellan de redan befintliga med algoritm som beskrivs under rubriken "förbättrat medelvärde". Den nya vektorn skrivs ut till wav-filen toFile. A och B beskriver samma filter som ovan. Detta används här för att lågpassfiltrera den nya wav-filen.

lpcinter.m
argument: toFile, fromFile, antalKoeff, antalNya, A, B)
Läser in från en .mat fil (fromFile). Skapar "antalNya" punkter mellan varje gammal punkt med LPC-algoritmen. LPC utförs med hjälp av antalKoeff koefficienter. Vektorerna A och B används även här för att filtrera den dekomprimerade wav-filen.

cut.m

```
function[] = cut(fromFile,toFile,antalKoeff,nrPoints,A,B)

%The function cut(fromFile,toFile,antalKoeff,nrPoints,A,B) reads the wavefile
%'fromFile' %into a vector. This vector is filtered through a lowpass filter
%which is specified by A and B. Then every nrPoints element is put into
%another vector, which is saved into the file 'toFile'. The coefficient vector
%for the LPC compression is calculated and saved to the file koef.mat.
%
%(The variables A and B specifies the frequency response for the
%lowpass filter)


clear Z;
X = WAVREAD(fromFile);              %Läser in wav-filen till vektorn X
Z = zeros(length(X)/nrPoints,1);    %Skapar vektorn som ska innehålla var
                                    %nrPoints:e av datapunkterna
Y = filter(B,A,X);                  %Lågpassfiltrerar den inlästa filen

j =1;
for i=1:nrPoints:length(X)          %Tar ut var nrPoint:e punkt ur X och
                                    %lägger i Z
    Z(j,1)= Y(i,1);
    j = j+1;
end

nrKoeff = 10000;
E=zeros(nrKoeff,1);
KOEF = zeros(antalKoeff+1);

for m=1:1:nrKoeff
      E(m) = Y(m+5000);
end

KOEF = lpc(E,antalKoeff);           %Skapar koefficientvektorn med antalKoeff
                                    %punkter

for i=1:1:length(Z)                 %Multiplicerar vektorns värden med 10000
                                    %för att kunna
    Z(i) = Z(i)*10000;              %använda heltalsrepresentation (16 bitar)
                                    %utan att förlora
end                                 %alltför mycket information

fid = fopen(toFile,'w');
fwrite(fid,Z,'int16');              %Skriver ut Z till 'toFile'.mat med
                                    %16 bitar/datapunkt

fid = fopen('koef.mat','w');
fwrite(fid,KOEF,'double');          %Skriver ut KOEF till filen koef.mat
```

interpol.m

```
function[] = interpol(fromFile, toFile, nrPoints, A, B)

%The function interpol(fromFile,toFile,nrPoints,A,B) reads the mat-file
%'fromFile' into a vector. The points in this vector are used to interpolate
%the remaining points in the initial vector. The vector is then filtered
%through a lowpass filter which is specified by A and B. The vector is written
%to the file 'toFile' av a wav-file.
%
%(The variables A and B specifies the frequency response for the lowpass
% filter)
%(The elements in the vector are the nrPoints:th elements in the initial
% vector)


fid = fopen(fromFile);
W = fread(fid,'int16');             %Läser in .mat-filen som ska interpoleras
                                    %(innehåller en vektor)
U = zeros(length(W)*nrPoints,1);

for i=1:1:length(W)                 %Återställer vektorns värden
    W(i) = W(i)/10000;
end


j=1;
for i=1:1:length(W)-1

    U(j)=W(i);

    if W(i+1) >= W(i)
        for k=1:1:nrPoints-1
            U(j+k)=W(i) + (W(i+1)-W(i)) * (k/nrPoints);
        end
    end
    if W(i+1) < W(i)
        for k=1:1:nrPoints-1
            U(j+k)=W(i+1) + (W(i)-W(i+1)) * ((nrPoints-k)/nrPoints);
        end
    end
    j = j + nrPoints;

end
Y = filter(B,A,U);
WAVWRITE(Y,44100,toFile);
```

lpcinter.m

```
function[] = lpcinter(fromFile,toFile,antalKoeff,antalNya,A,B)

%The function lpcinter(fromFile,toFile,antalKoeff,antalNya,A,B) reads the
%mat-file 'fromFile' into a vector. The points in this vector are used to
%LPC-interpolate the remaining points in the initial vector. The coefficient
%vector is loaded in from the file koef.mat. The new vector is then filtered
%through a lowpass filter which is specified by A and B. The vector is written
%to the file 'toFile' as a wav-file.
%(The variables A and B specifies the frequency response for the
%lowpass filter)
%(antalKoeff specifies the number of elements in the koefficient vector)
%(antalNya specifies the number of elements that are to be LPC-interpolated
%for every element in the vector)

fid = fopen(fromFile);
W = fread(fid,'int16');             %Läser in .mat-filen som ska interpoleras
                                    %(innehåller en vektor)
NY = zeros(length(W)+length(W)*antalNya,1);
for i=1:1:length(W)                 %Återställer vektorns värden
    W(i) = W(i)/10000;
end

fid = fopen('koef.mat');            %Läser in filen som innehåller
KOEF = fread(fid,'double');         %koefficientvektorn

% Första antalet punkter har vi inte tillräckligt många att interpolera över.
% Det räcker därför med enkel fördubbling av punkterna.
j=1;
for k=1:antalNya+1:antalKoeff+(antalKoeff*antalNya)-antalNya-1
    NY(k)=W(j);
    for u=1:1:antalNya
        NY(k+u) = W(j);
    end
    j=j+1;
end

start=antalKoeff+(antalKoeff*antalNya)-antalNya;
antalTillagda=0;
for i=start:1:length(W)-antalKoeff*antalNya-1
    NY(i+antalTillagda) = W(j);   %lägger till "gamla" datapunkter direkt i
                                  %den nya vektorn
    nypunkt=0;
    for p=1:1:antalKoeff          %räknar ut ny punkt utifrån KOEF-vektorn
        nypunkt = nypunkt-KOEF(p+1)*W(j-p+1);
    end
    for q=1:1:antalNya
        NY(i+antalTillagda+1) = real(nypunkt);
        antalTillagda=antalTillagda + 1;
    end
    j=j+1;
end

Y = filter(B,A,NY);
langdW = length(W)
langdNY = length(NY)
samplingsf= 44100/(312992/length(NY));
WAVWRITE(Y,samplingsf,toFile);
```

.mat-filer (endast på CD-skiva)

Dessa är alla komprimerade med funktionen cut.m och innehåller vektorn med kvarvarande värden för ljudfilen. Det är dessa filer vi tänker oss att man skickar över till en mottagare. Man skickar då också med filen *koef.mat* som innehåller koefficientvektorn, vilken används vid LPC-dekomprimering. Den intresserade kan studera vektorerna i Matlab genom att använda kommandona *fread* och *fopen*.

komp2.mat - innehåller hälften så många punkter som ursprungsfilen
komp3.mat - innehåller var tredje punkt av ursprungsfilen
komp5.mat - innehåller var femte punkt av ursprungsfilen.
komp8.mat - innehåller var åttonde punkt
komp10.mat - innehåller var tionde punkt
komp20.mat - innehåller var 20:e punkt
komp40.mat - innehåller var 40:e punkt
komp50.mat - innehåller var 50:e punkt
komp100.mat - innehåller var 100:e punkt

.wav-filer (endast på CD-skiva)

original - originalljudfilen som vi arbetat med

För att utröna vilket filter som var bäst att använda testade vi lite olika varianter. När vi testade fick vi ibland fel samplingsfrekvens, vilket gav upphov till lite lustiga ljud (t.ex. kenny nedan).

filter1 - den ursprungliga ljudfilen filtrerad med ett Butterworth IIR-filter. Passband 0-100 Hz, gränsfrekvens 150 Hz, ordning 6.
filter2 - den ursprungliga ljudfilen filtrerad med ett Chebyshev I IIR-filter. Passband 0-200 Hz, gränsfrekvens 300 Hz,ordning 4.
kenny - lågpassfilterad med samma som filter2, samplad med för hög samplingsfrekvens.
filter3 - Butterworth IIR-filter. Passband 0-800 Hz, gränsfrekvens 950 Hz, ordning 28. Detta filter är det vi använt i det fortsatta projektet.

Dekomprimerade filer
Filer som dekomprimerats med interpol.m efter att ha komprimerats av cut.m. Siffran i filnamnet anger vilken komprimeriad .mat-fil som använts som input. Dek5 motsvarar alltså att 4 nya punkter satts in.

dek2
dek3
dek5
dek8
dek10
dek20
dek40
dek50
dek100

LPC-dekomprimerade filer
Filer som dekomprimerats med lpcinter.m efter att ha komprimerats av cut.m. Siffran i filnamnet anger vilken komprimerad .mat-fil som använts som input (pss som ovan). Dessa filer är gjorda med interpolation över 16 tidigare punkter.

lpc2
lpc3
lpc5
lpc8
lpc10
lpc20
lpc40
lpc50
lpc100

Spårförteckning till medföljande CD-skiva

1. original.wav
2. filter1.wav
3. filter2.wav
4. kenny.wav
5. filter3.wav
6. dek2.wav
7. lpc2.wav
8. dek3.wav
9. lpc3.wav
10. dek5.wav
11. lpc5.wav
12. dek8.wav
13. lpc8.wav
14. dek10.wav
15. lpc10.wav
16. dek20.wav
17. lpc20.wav
18. dek40.wav
19. lpc40.wav
20. dek50.wav
21. lpc50.wav
22. dek100.wav
23. lpc100.wav