The MFCC

1- How are MFCCs used in speech recognition ?

Generally speaking, a conventional automatic speech recognition (ASR) system can be organized in two blocks: the feature extraction and the modeling stage. In practice, the modeling stage is subdivided in acoustical and language modeling, both based on HMMs as described in Figure 1.

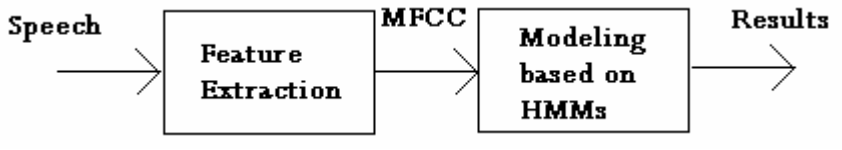


Figure 1- Simple representation of a conventional ASR.

The feature extraction is usually a non-invertible (lossy) transformation, as the MFCC described pictorially in Figure 2. Making an analogy with filter banks, such transformation does not lead to perfect reconstruction, i.e., given only the features it is not possible to reconstruct the original speech used to generate those features.

Computational complexity and robustness are two primary reasons to allow loosing information. Increasing the accuracy of the parametric representation by increasing the number of parameters leads to an increase of complexity and eventually does not lead to a better result due to robustness issues. The greater the number of parameters in a model, the greater should be the training sequence.

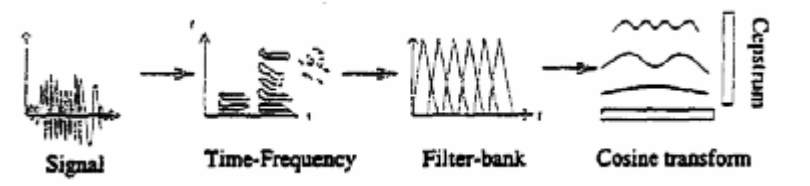


Figure 2- Pictorial representation of mel-frequency cepstrum (MFCC) calculation.

Speech is usually segmented in frames of 20 to 30 ms, and the window analysis is shifted by 10 ms. Each frame is converted to 12 MFCCs plus a normalized energy parameter. The first and second derivatives (Δ's and $\Delta\Delta$'s) of MFCCs and energy are estimated, resulting in 39 numbers representing each frame. Assuming a sample rate of 8 kHz, for each 10 ms the feature extraction module delivers 39 numbers to the modeling stage. This operation with overlap among frames is equivalent to taking 80 speech samples without overlap and representing them by 39 numbers. In fact, assuming each speech sample is represented by one byte and each feature is represented by four bytes (float number), one can see that the parametric representation increases the number of bytes to represent 80 bytes of speech (to 136 bytes). If a sample rate of 16 kHz is assumed, the 39 parameters would represent 160 samples. For higher sample rates, it is intuitive that 39 parameters do not allow to reconstruct the speech samples back. Anyway, one should notice that the goal here is not speech compression but using features suitable for speech recognition.

2- What are MFCCs ? How are they calculated ?

The block diagrams for calculating MFCCs is given below.

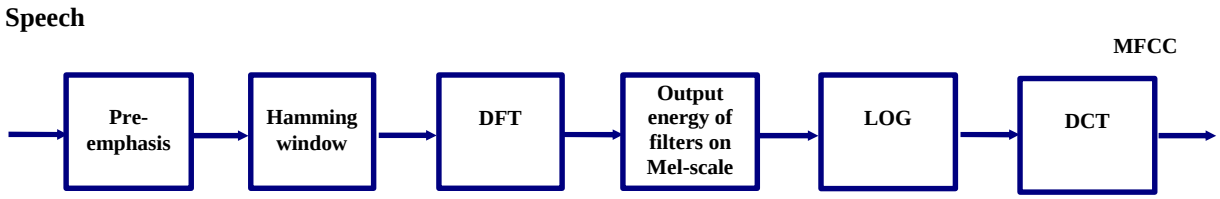


Figure 3- MFCC calculation.

The point is: **Why these blocks ? What is the intuition behind each one ?**

In my opinion there are two ways of looking to the MFCCs: (a) as a filter-bank processing adapted to speech specificities and (b) as a modification of the conventional cepstrum, a well known deconvolution technique based on homomorphic processing. These points of view are complementary and help getting insight about MFCCs. I will briefly describe each one.

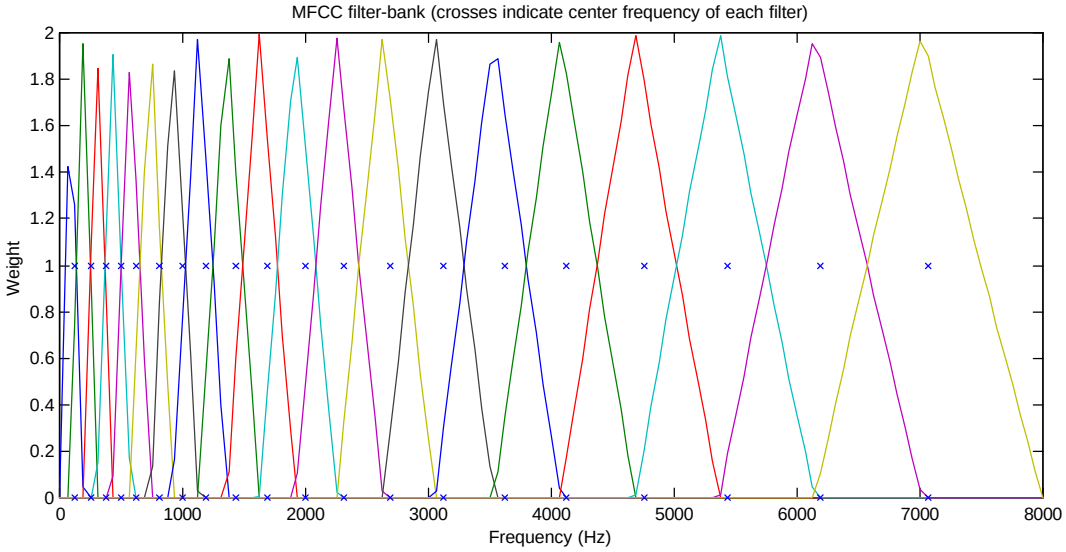
2.1- Mel-scale: from auditory modeling

Before proceeding, let us take in account some characteristics of the human auditory system. Two famous experiments generated the Bark and mel scales, given below. Describe the experiments.

	Bark		Mel	
Filter	Frequency (Hz)	BW (Hz)	Frequency (Hz)	BW (Hz)
1	50	100	100	100
2	150	100	200	100
3	250	100	300	100
4	350	100	400	100
5	450	110	500	100
6	570	120	600	100
7	700	140	700	100
8	840	150	800	100
9	1000	160	900	100
10	1170	190	1000	124
11	1370	210	1149	160
12	1600	240	1320	184
13	1850	280	1516	211
14	2150	320	1741	242
15	2500	380	2000	278
16	2900	450	2297	320
17	3400	550	2639	367
18	4000	700	3031	422
19	4800	900	3482	484
20	5800	1100	4000	556
21	7000	1300	4595	639
22	8500	1800	5278	734
23	10500	2500	6063	843

24	13500	3500	6964	969

So, we use the mel scale to organize the filter bank used in MFCC calculation. Using function *melbankm* (this function returns a sparse matrix, so I used command *full* to convert it to a regular matrix). Notice the weight is 2.




```
filterAmplitudes=full(melbankm(NfiltersOfMelBank ,frame_duration,Fs));
   peak = max(filterAmplitudes');
   for index = 1:length(peak)
      filterCenter(index) = find(filterAmplitudes(index,:)==peak(index));
   end
   xaxis_in_Hz = (0:128)*Fs/frame_duration;
   plot(xaxis_in_Hz,filterAmplitudes'); hold on

   HzScale = filterCenter * Fs / frame_duration;

   plot(HzScale,ones(1,length(filterCenter)),'x');
   plot(HzScale,zeros(1,length(filterCenter)),'x');

   xlabel('Frequency (Hz)');
   ylabel('Weight');
   title('MFCC filter-bank (crosses indicate center frequency of each filter)');
```

2.1 - Cepstral analysis, the historical father of the MFCCs

Homomorphic processing is well discussed by Oppenheim in his textbooks. Cepstrum is maybe the most popular homomorphic processing because it is useful for deconvolution. To understand it, one should remember that in speech processing, the basic human speech production model adopted is a **source-filter** model.

- ➔ **Source**: is related to the air expelled from the lungs. If the sound is **unvoiced**, like in "s" and "f", the glottis is open and the vocal cords are relaxed. If the sound is **voiced**, "a", "e", for example, the vocal cords vibrate and the frequency of this vibration is related to the pitch.
- ➔ **Filter**: is responsible for giving a shape to the spectrum of the signal in order to produce different sounds. It is related to the vocal tract organs.

Roughly speaking: a good parametric representation for a speech recognition system tries to eliminate the influence of the source (the system must give the same "answer" for a high pitch female voice and for a low pitch male voice), and characterize the filter. The ***problem*** is: source e(n) and filter impulse response h(n) are convoluted. Then we need deconvolution in speech recognition applications. Mathematically:

In the time domain, convolution: source * filter = speech,

$$e(n) * h(n) = x(n). \quad (1)$$

In the frequency domain, multiplication: source x filter = speech,

$$E(z) \quad H(z) = X(z). \quad (2)$$

How can we make the deconvolution ? ***Cepstral analysis*** is an alternative.

➔ Working in the frequency domain, use the logarithm to transform the multiplication in (2) into a summation (obs: log ab = log a + log b). It is not easy to separate (to filter) things that are multiplied as in (2), but it is easy to design filters to separate things that are parcels of a sum as below:

$$C(z) = \log X(z) = \log E(z) + \log H(z). \quad (3)$$

We hope that H(z) is mainly composed by low frequencies and E(z) has most of its energy in higher frequencies, in a way that a simple low-pass filter can separate H(z) from E(z) if we were dealing with E(z) + H(z). In fact, let us suppose for the sake of simplicity that we have, instead of (3), the following equation:

$$C_o(z) = E(z) + H(z). \quad (4)$$

We could use a linear filter to eliminate E(z) and then calculate the Z-inverse transform to get a time-sequence $c_o(z)$. Notice that in this case, co(z) would have dimension of time (seconds, for example).

Having said that, let us now face our problem: the log operation in (3). Log is a non-linear operation and it can "create" new frequencies. For example, expanding the log of a cosine in Taylor series shows that harmonics are created. So, even if E(z) and H(z) are well separated in the frequency domain, log E(z) and log H(z) could eventually have considerable overlap. Fortunately, that is not the case in practice for speech processing. The other point is that, because of the log operation, the Z-inverse of C(z) in (3) has NOT the dimension of time as in (4). We call ***cepstrum*** the Z-inverse of C(z) and its dimension is ***quefrency*** (a time domain parameter).

➔ There are 2 basic types of cepstrum: *complex cepstrum* and real *cepstrum*. Besides, there are two ways of calculating the real cepstrum (used in speech processing because phase is not important): *LPC cepstrum* and *FFT cepstrum*.

LPC cepstrum: the cepstral coefficients are obtained from the LPC coefficients
FFT cepstrum: from a FFT

Which one is better ? The most widely parametric representation for speech recognition is the FFT cepstrum derived based on a mel scale [Davis, 80].

2.2 - Filter-bank interpretation: the simplest way to look at MFCCs

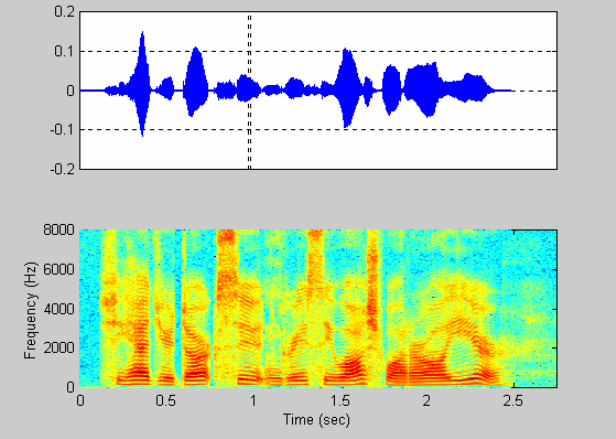
We go to frequency domain and disregard phase, working only with the power spectrum. Then, we use log because our ears work in decibels. To reduce dimensionality, we use a filter-bank with around 20 filters. The filters follow mel-scale. We take the DCT-II because it is good for compressing information, uncorrelating the vector, doing a work asymptotically close to KLT. Then we disregard the high-frequency DCT coefficients. Much easier, is it not ?

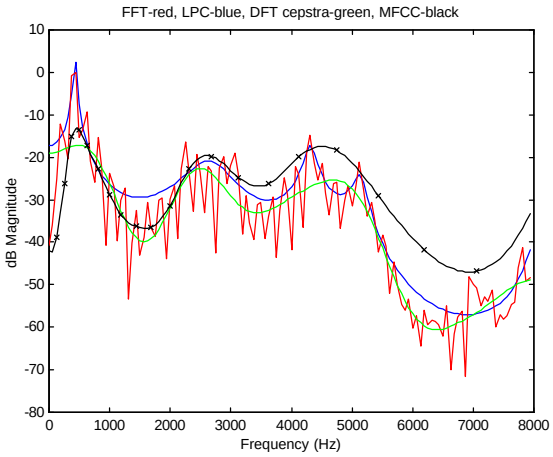
Reconstructing the spectrum based on MFCC

Given MFCCs, how to reconstruct the speech spectrum back ? See the appendix.

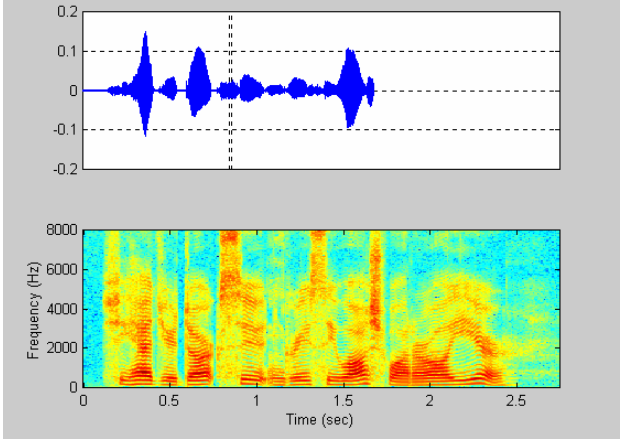
Some examples below: one segment of voice speech and another unvoiced.

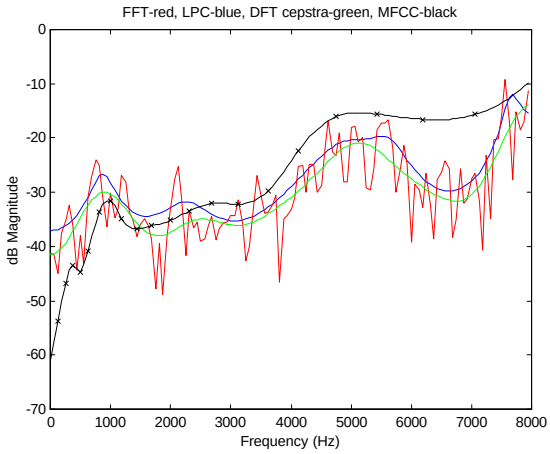
Voiced speech



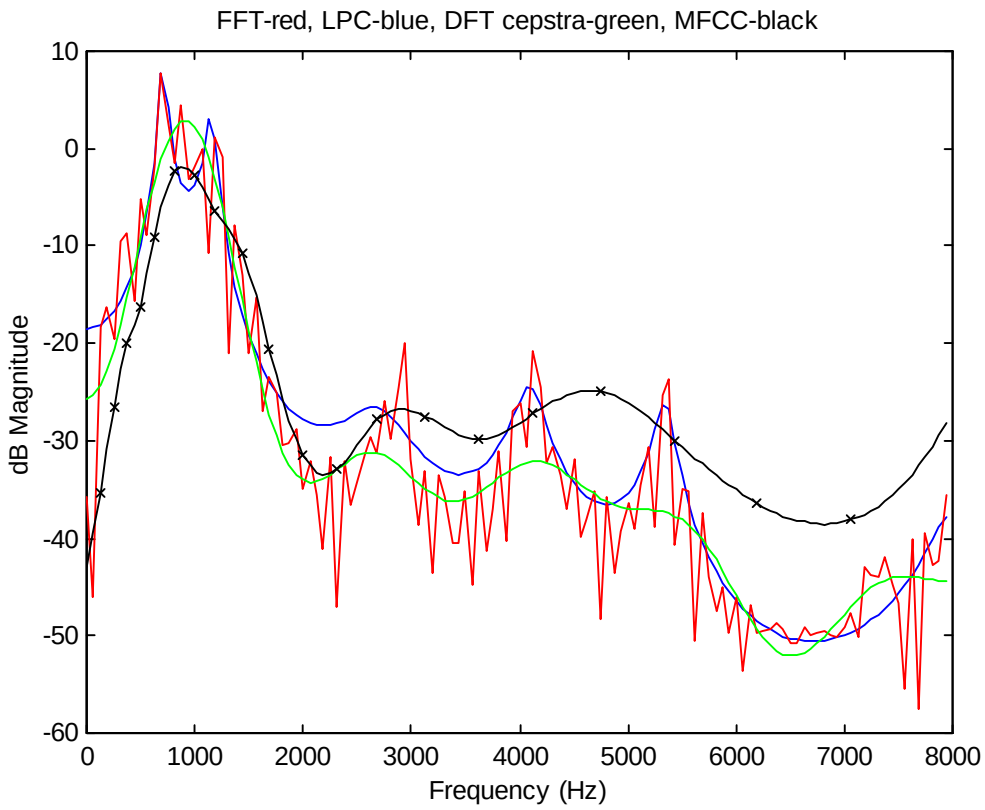


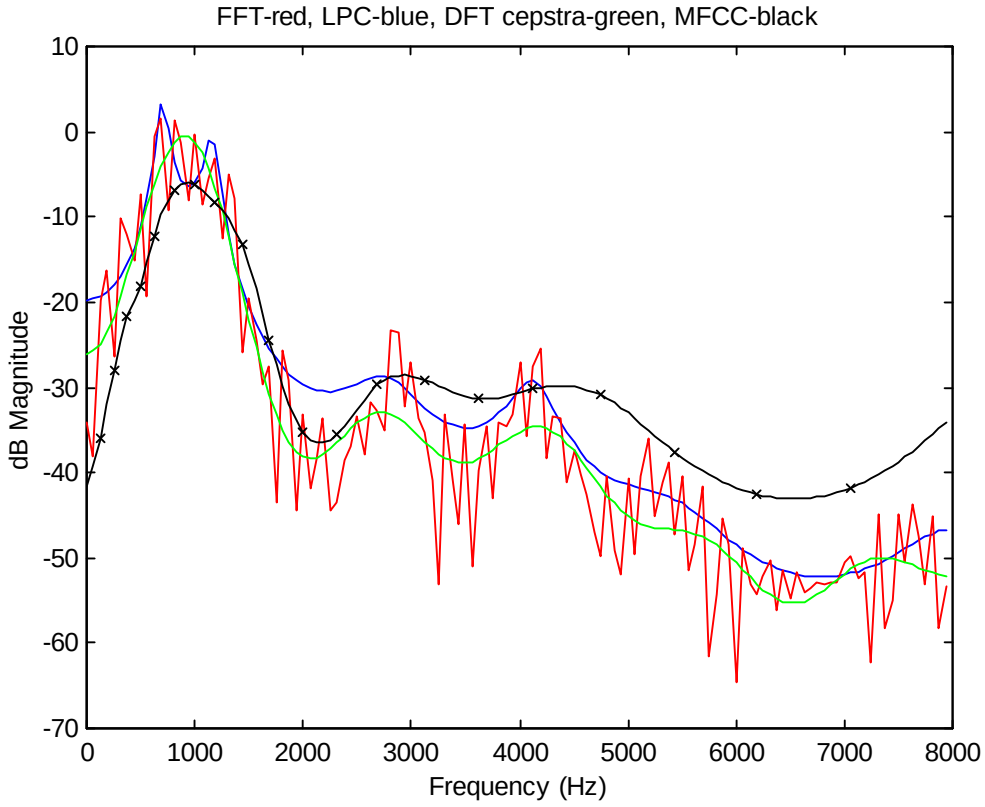
Unvoiced speech





The examples below show cases where the MFCC did not capture the formants structure, i.e., they did not perform a good job.


FFT-red, LPC-blue, DFT cepstra-green, MFCC-black
dB Magnitude
Frequency (Hz)
10
0
-10
-20
-30
-40
-50
-60
0
1000
2000
3000
4000
5000
6000
7000
8000


FFT-red, LPC-blue, DFT cepstra-green, MFCC-black
dB Magnitude
Frequency (Hz)
10
0
-10
-20
-30
-40
-50
-60
-70
0
1000
2000
3000
4000
5000
6000
7000
8000

```
                                INVERTING DFT CEPSTRUM
Nfft = frame_duration;
engthOfPositivePart=floor(Nfft/2+1);
Npad = lengthOfPositivePart - length(cep);
cep = [cep zeros(1,Npad)];
cep = getNegativeFreq(cep);
cep = fft(cep);
cep = exp(real(cep));
%convert to db
cep = 20*log10(cep);
                            INVERTING DFT MEL - CEPSTRUM
melcep = [melcep zeros(1,NfiltersOfMelBank-length(melcep))]
melcep = irdct(melcep);
melcep = exp(melcep);
melcep = 20*log10(melcep) - 20;
filterAmplitudes=full(melbankm(NfiltersOfMelBank ,frame_duration,Fs));
peak = max(filterAmplitudes');
for index = 1:length(peak)
  filterCenter(index) = find(filterAmplitudes(index,:)==peak(index));
end
HzScale = filterCenter * Fs / frame_duration;
```

Each 10 milliseconds of speech is represented by a vector of parameters, calculated as mentioned below.

Details of calculating the features based on MFCCs

Delta and Acceleration Coefficients:

The first order regression coefficients (delta coefficients) are computed by the following regression equation:

$$d_i = \frac{\sum_{n=1}^{N} n(c_{n+i} - c_{n-i})}{2\sum_{n=1}^{N} n^2}$$

where d_i is the delta coefficient at frame i computed in terms of the corresponding basic coeffecients c_{n+i} to c_{n-i} . The same equation is used to compute the acceleration coefficients by replacing the basic coefficients with the delta coefficients.

Cepstral Mean Normalization:

This option is aimed at reducing the effect of multiplicative noise on the feature vectors. Mathematically it is:

$$c_i = c_i - \frac{1}{N}\sum_{k=1}^{N} c_{ik}$$

where c_i is the ith feature element in the feature vector and c_{ik} is the ith feature element at frame k. N is the number of total input frames of data.

Energy and Energy Normalization:

Energy is calculated as the log signal energy using the following equation:

$$E = \log \sum_{n=1}^{N} s_n^2$$

where s_n is the nth input speech data sample and N is the total number of input samples per frame. The corresponding normalization is to subtract the noise floor from the input data. Note that the silence floor is usually set to 50 dB and the minimum energy value is -1.0e+10.

`Liftering:`

Liftering is applied according to the following equation.

$$c_n' = \left(1 + \frac{N}{2}\sin\frac{\pi n}{N}\right)c_n$$

`Pre-emphasis:`

The first order difference equation:

$$s_n' = s_n - \alpha s_{n-1}$$

is applied a window of input samples. Here a is the pre-emphasis filter coefficient in the range $[0, 1)$.

Studying the DCT in MFCC calculation

The DCT used in MFCC is DCT type II as shown below.

```
%studying the DCT in MFCC calculation proposed by Davis and Merlmestein [1980]
totalFilters = 24;
cepstralCoefficients = 6;

%Davis proposes
dct2Matrix = cos((0:totalFilters-1)'/totalFilters * ((0:cepstralCoefficients-1)+0.5)*pi);
subplot(211);
plot(dct2Matrix);
title('Basis functions for DCT-II used in MFCC by Davis');

%I would expect in case the transform was derived from FFT:
dct1Matrix = cos((0:totalFilters-1)'/totalFilters * ((0:cepstralCoefficients-1))*pi);
subplot(212);
plot(dct1Matrix);
title('Basis functions for DCT-I');

%but considering that the transform comes from the definition of DCT-II
x = randn(totalFilters,1);
cepstralCoefficients = totalFilters;
dct2Matrix = cos((0:totalFilters-1)'/totalFilters * ((0:cepstralCoefficients-1)+0.5)*pi);
dct2Matrix = sqrt(2/totalFilters) * dct2Matrix; %scaling factor
dct2Matrix(1,:) = dct2Matrix(1,:) / sqrt(2); %first row has different scaling factor
y = dct2Matrix * x; %using definition by Davis
y2 = dct(x); %Matlab DCT-II implementation
e = y - y2; %one can see they are equivalent
```

Plots of MFCCs

We used the TIMIT database, that consists of labeled segments of speech. As an example, a phrase is shown in Figure 3.2. The TIMIT database gives the first and last samples of each word, and the first and last sample of each phoneme too. In Figure 3.2, the vertical line in blue indicates the beginning of a word and the red line indicates its end.

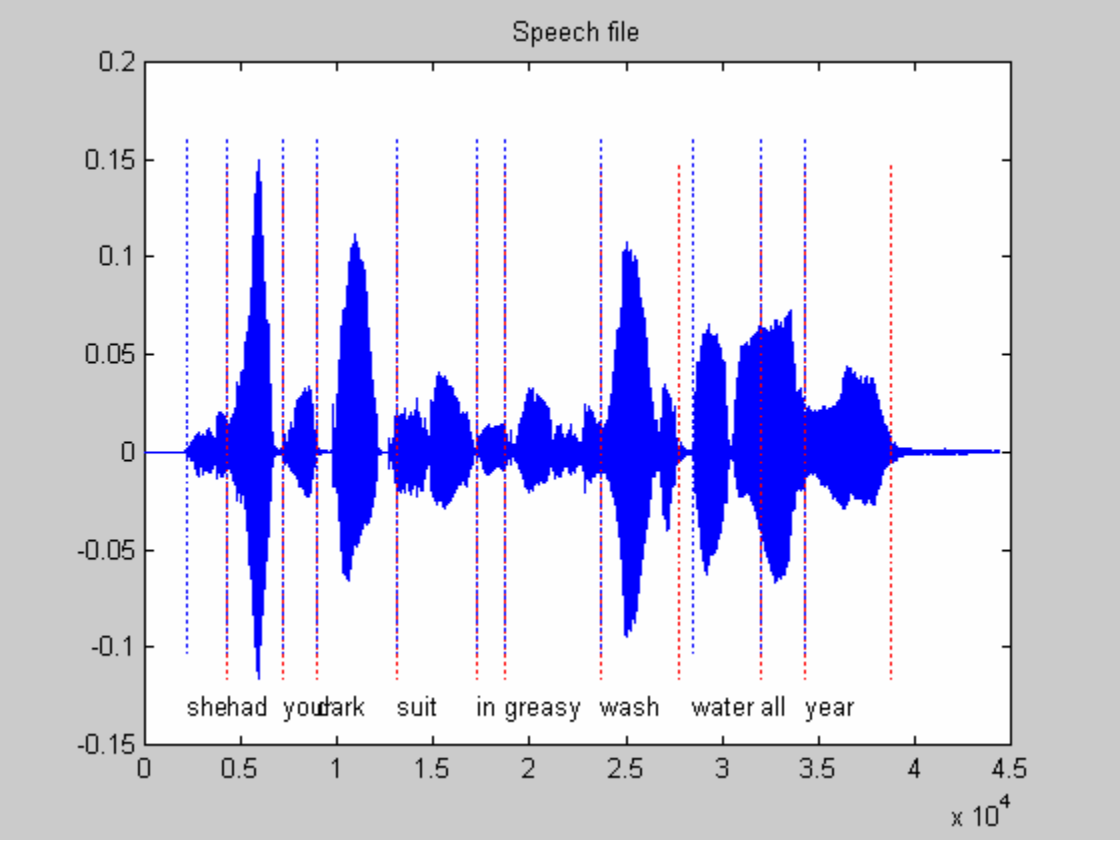


Figure 3.2- Example of a phrase in the TIMIT database: "she had your dark suit in greasy wash water all year".

For the sake of comparison, the first 3 MFCCs are ploted in Figure 3.3. It can be seen that the MFCCs vary significantly even during a given word.

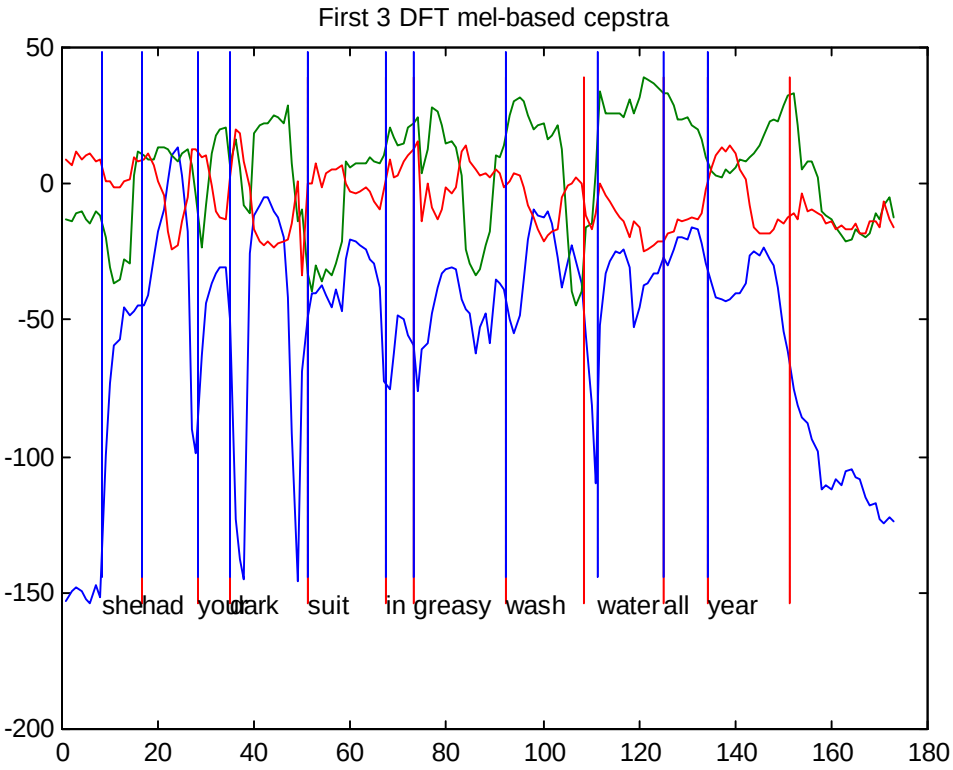


Figure 3.3- 3 first MFCCs for the phrase in the TIMIT database described in Figure 2.30.

Using 40 frames (each frame represented by 8 MFCCs), we generated figures 3.4 - 3.23 showing the trajectory in time obtained for the first 3 MFCCs. Figures 3.24 and 3.25 show all the segments concatenated.

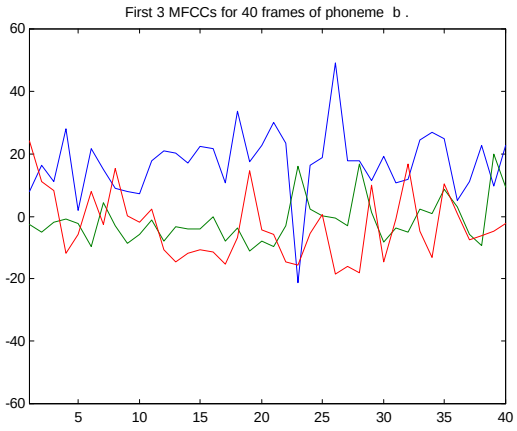


Figure 3.4- MFCCs for phoneme b.

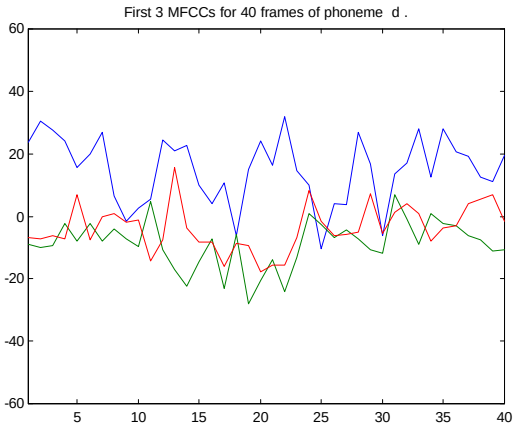


Figure 3.5- MFCCs for phoneme d.

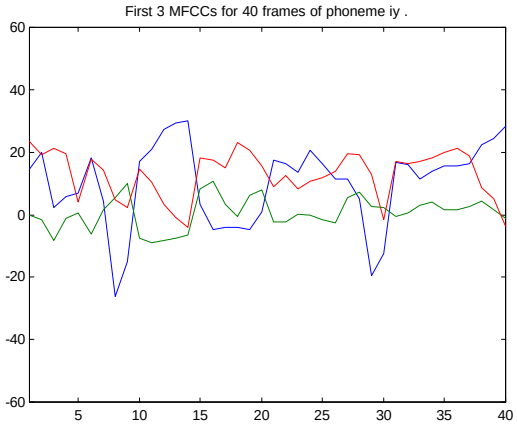


Figure 3.6- MFCCs for phoneme iy.

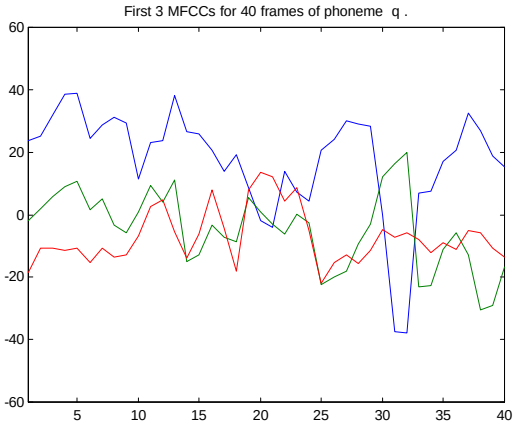


Figure 3.12- MFCCs for phoneme q.

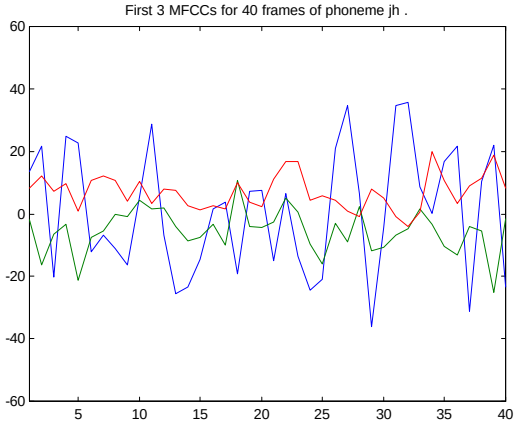


Figure 3.10- MFCCs for phoneme jh.

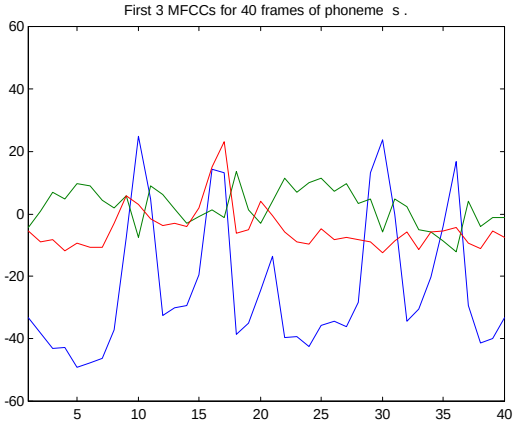


Figure 3.16- MFCCs for phoneme s.

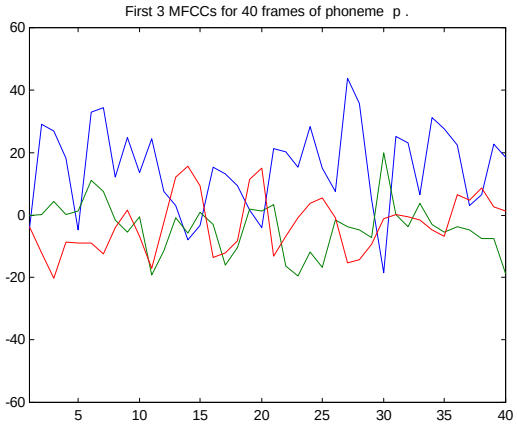


Figure 3.11- MFCCs for phoneme p.

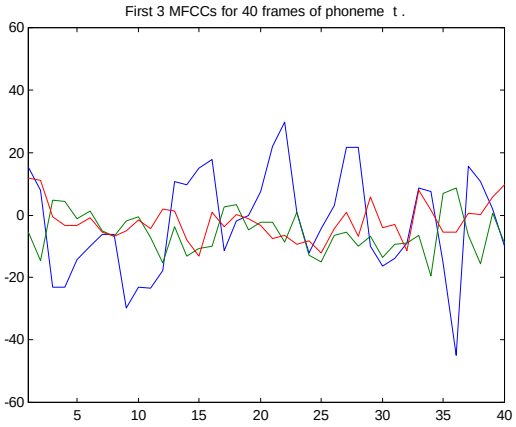


Figure 3.17- MFCCs for phoneme t.

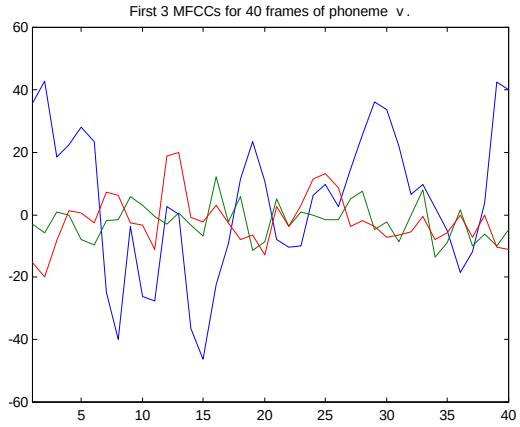


Figure 3.18- MFCCs for phoneme v.

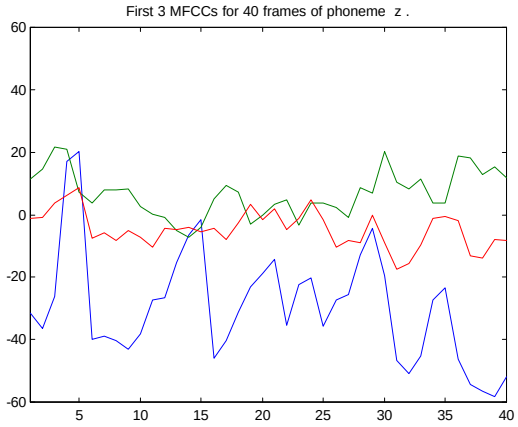


Figure 3.22- MFCCs for phoneme z.

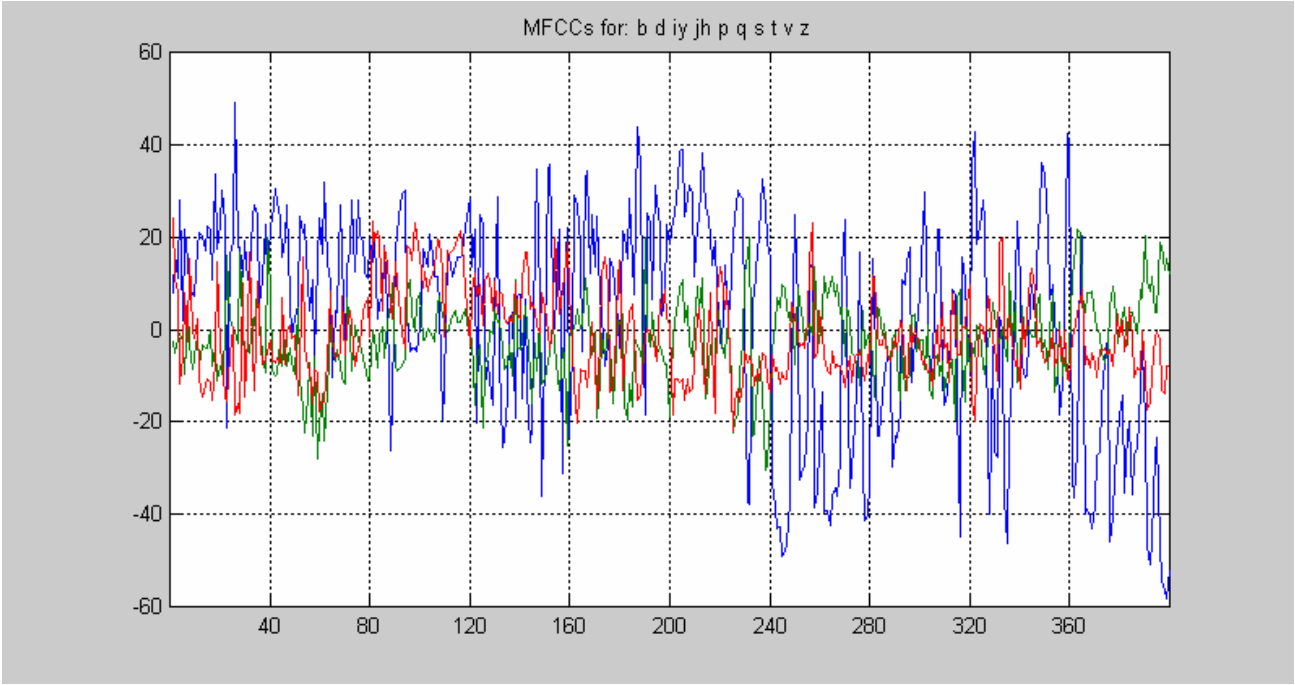


Figure 3.24- MFCC trajectories in time for all E-set (10 phonemes) concatenated.

References

[Davis, 80] Davis, S. and Merlmestein, P., "***Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences***", IEEE Trans. on ASSP, Aug, 1980. pp. 357-366.

Appendix - Inverting cepstrum

```
%Invert cepstrum to study correspondent spectrum
%1) DFT based cepstra
%2) DFT based mel-cepstra
%3) LPC

clear all

Fs=16000;

y=readsph('sa1.wav');
[siz temp] = size(y);

%numberofwords=11;

frame_duration = 256; %frame duration
number_of_frames = floor(siz / frame_duration)
number_of_frames = input('Give the number of frames you want process: ');

numberoffeatures = 12; %number of MFCC
figure(1); subplot(211); plot(y); grid;
subplot(212); Nfft=512; window = blackman(Nfft); Fs=16000;
specgram(y,Nfft,Fs,window,round(3/4*length(window)));
ylabel('Frequency (Hz)'); xlabel('Time (sec)');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:number_of_frames
   %take one frame of speech signal
   begin_sample = 1+(i-1)*frame_duration;
   end_sample = i*frame_duration;
   s = y(begin_sample:end_sample );
   s = s(:);

   %calculate LPC cepstra
   sw = s.*hamming(frame_duration); %windowing
   [ar, g] = lpc(sw,numberoffeatures); g = g^2;

   %freqz(g,ar); pause;

   %cc=lpcar2cc(ar,numberoffeatures);
   %rc=cc / 2; % the "real" cepstrum is half the complex cepstrum
   %testing:
   %n=5; ar=lpc(s,n), cc=lpcar2cc(ar,n+2), lpccc2ar(cc)
   %rc=[log(g) rc];
   %lpccepstra(i,:)=rc;

   %LPC based mel-cepstra
   %lpcmelcepstra(i,:) = lpclc2wlc(rc, 0.6);

   %DFT real cepstra
   %dft_rc = real( ifft( log ( abs( fft( sw ) ) ) ) );
   dft_rc_temp = log ( abs( fft( sw ) ) ); %plot(dft_rc_temp); pause
   dft_rc = real( ifft( dft_rc_temp) ); %plot(dft_rc); title('cepstrum'); pause
   %plot(imag(ifft(dft_rc_temp))); title('imag dft_rc_temp'); pause

   %besta = fft(dft_rc); plot(imag(besta)); title('imag besta'); pause
   %besta = exp(real(besta));
   %plot(real(besta)); title('besta'); pause;

   dft_rc = dft_rc(1:floor(frame_duration/2 + 1)); %take out negative quefrencies
   dft_rc = dft_rc(1:numberoffeatures);
   dftcepstra(i,:)=dft_rc';

   %now, i'll try to reconstruct the spectrum. Assume even numbers
   cep = dftcepstra(i,:);
   Nfft = frame_duration; %bigger than 2 times number of parameters
   lengthOfPositivePart = floor(Nfft/2 + 1);
   %lengthOfNegativePart = Nfft - lengthOfPositivePart
```

```
   Npad = lengthOfPositivePart - length(cep);
   cep = [cep zeros(1,Npad)];
   cep = getNegativeFreq(cep);
   %plot(cep); pause
   %magfft(cep,128,0); pause;
   %return
   cep = fft(cep);
   %plot(imag(cep));   pause
   %plot(real(cep)); pause
   cep = exp(real(cep));
   %convert to db
   cep = 20*log10(cep);
   %plot(cep); pause;

   %DFT based mel-cepstra
   NfiltersOfMelBank = 20;

fmcc=akmelcepst(s,Fs,'Mta0',numberoffeatures,NfiltersOfMelBank,frame_duration,frame_duration,0,0.5);
   dftmelcepstra(i,:)=fmcc;

   %now, i'll try to reconstruct the spectrum. Assume even numbers
   melcep = fmcc;
   melcep = [melcep zeros(1,NfiltersOfMelBank-length(melcep))]
   melcep = irdct(melcep);
   melcep = exp(melcep);
   melcep = 20*log10(melcep) - 20;

   filterAmplitudes=full(melbankm(NfiltersOfMelBank ,frame_duration,Fs));
   peak = max(filterAmplitudes');
   for index = 1:length(peak)
      filterCenter(index) = find(filterAmplitudes(index,:)==peak(index));
   end
   xaxis_in_Hz = (0:128)*Fs/frame_duration;
   plot(xaxis_in_Hz,filterAmplitudes'); hold on

   HzScale = filterCenter * Fs / frame_duration;

   plot(HzScale,ones(1,length(filterCenter)),'x');
   plot(HzScale,zeros(1,length(filterCenter)),'x');

   xlabel('Frequency (Hz)');
   ylabel('Weight');
   title('MFCC filter-bank (crosses indicate center frequency of each filter)');


   %subplot(411); plot(lpccepstra(i,:));
   %subplot(412); plot(lpcmelcepstra(i,:));
   %subplot(413); plot(dftcepstra(i,:));
   %subplot(414); plot(dftmelcepstra(i,:));
   %pause

   %[ar,g]=lpc(sw,10);
   figure(2); clf
   Xv=(abs(fft(sw,Nfft)));
   [H,F]=freqz(sqrt(g),ar,Nfft/2,Fs);
   plot(F,20*log10(abs(H)),'b',F,20*log10(Xv(1:Nfft/2)),'r'); hold on
   plot(F,cep(1:Nfft/2),'g');
   plot(HzScale,melcep,'xk');
   interpolatedMelCep = spline(HzScale,melcep,F);
   plot(F,interpolatedMelCep,'k');
   xlabel('Frequency (Hz)')
   ylabel('dB Magnitude')
   title('FFT-red, LPC-blue, DFT cepstra-green, MFCC-black');

   figure(1); subplot(211);
   set(gca,'xtick',[begin_sample end_sample]);
   pause;
   %plot(s); pause;
end
```