A 12-week project in

Speech Coding and Recognition

by Vedrana Andersen
vedrana@itu.dk
(130274-xxxx)

and Fu-Tien Hsiao
hsiao@itu.dk
(031278-xxxx)

Project supervisor:
John Aasted Sørensen

IT University of Copenhagen
September – November 2005.

Contents

1 Introduction

The human sense of hearing and the human's ability to talk are very important means of communication, which are gaining importance for IT-systems. Therefore, after completing the 'Signal Processing' course in our first semester of studies at IT University of Copenhagen, we decided to join the 'Speech Coding and Recognition' project cluster and to carry out the 12-week project in 'Speech Coding and Recognition'. The project aims at introducing basic principles and fundamental models for production, perception, coding and recognition of the speech signals. Those models are necessary for the understanding, construction and performance evaluation of IT-systems, which use speech as one of the input/output media.

We started the work on this project by attending the weekly lessons given by our supervisor, John Aasted Sørensen. First set of lessons covered the models for speech production, human vocal tract and linear prediction used for parameter estimation. After that we moved to speech coding using analysis-by-synthesis method. Lastly we turned to speech recognition using the hidden Markov model.

In parallel with attending the lessons, we worked on the hands-on exercises using the speech processing algorithms represented in MATLAB. Finally, toward the end of the project period we compiled the completed exercises into this project report.

There are in total four exercises in the report. The first two exercises are well-documented, stating both the detailed explanation of the considered topic, implementations of needed MATLAB functions, generated plots and our comments. In the last two exercises we put focus on our own observations and conclusions. The reader should keep in mind that this report forms a comprehensive text only together with the exercises. The numbers in the form (x.x.x) are references to the exercise number in the exercise sheets.

First exercise, 'An Introduction to Speech Signals', presents the simplified model for the speech production process, where the speech is described as a series of the steady-state sounds. The difference between modeling voiced and unvoiced sounds is explained. In the time domain, we use short-time power and zeros crossing measure to determine whether the signal is voiced or unvoiced. In the frequency domain we analyze DFT based magnitude spectrum of the speech signals. We determine formants and we estimate pitch period from the harmonic product spectrum. Based on those analyses, we also try to estimate the pitch in a sliding window. We observe the speech spectrograms and notice the temporal changes in magnitude spectra.

In the second exercise, 'Linear Prediction Analysis', we first introduce the short-term autocorrelation. We then assume all-pole model of the speech production. This model leads us to the autoregressive speech production, where the speech signal can be predicted using a linear combination of its past values. We use autocorrelation method to find the LP coefficients, first by 'brute force' and later by Levison-Durbin algorithm. This recursive algorithm utilizes the fact that the autocorrelation matrix is Toeplitz, and that it's elements are is in a special relationship

to the elements of the autocorrelation vector. Finally, we perform frame based LP analysis of the speech signals. We also used different techniques for pitch and gain estimation and use the results to synthesize the signal.

Third exercise, 'Speech Coding and Synthesis', considers the problem of representing speech signals digitally in the way appropriate for transmission over communication channels. In the coding part (sending end), we apply LP analysis and perceptual weighting. We focus on Code-Excited Linear Prediction coder (CELP) analyze-by-synthesis method, in which the excitation sequence is selected from a Gaussian codebook. We use MATLAB function that performs frame-based estimation of LP parameters, gain and excitation parameters. In the decoding part (receiving end) we synthesize the signal and discuss the quality of it for different settings. We also try using quantization of the transmitted parameters, and we discuss influence of the quantization on the synthesized signal.

In the last exercise, 'Speech Recognition', we gradually build a word recognizer able to recognize ten words. The word recognizer is based on the hidden Markov model (HMM). First we apply the LP analysis to extract feature vectors containing cepstral coefficients from the training words. We then perform vector quantization to produce a codebook, so that each speech signal can be represented as series of symbols. Then we use forward-backward reestimation algorithm to train each HMM on a given word, by maximizing the probability that the word was produced by the HMM. Finally, we can use built models to recognize new, unknown words by finding the HMM with the highest probability of producing the word. We analyzed each of those steps, and performed the tests on the word recognizer changing the block size and spacing, the LPC and cepstrum order, and the codebook size.

Through this project we gained the theoretical insight of the covered topics and of algorithms used in speech processing. We also obtained the practical experience on implementing and using simple speech processing tools which can serve as building blocks for more advanced applications.

We would like to thank our supervisor, John Aasted Sørensen, for his guidance during the lessons, and valuable comments and suggestions for carrying out the exercises and writing the project report.

2 An Introduction to Speech Signals

2.1 Speech Signals in the Time Domain

In technical discussions, the entire combination of all speech production cavities is referred to as the vocal tract and comprises the main acoustic filter. The filter is excited by organs below it (vocal cords, lungs, etc.) and loaded at its main output by a radiation impedance due to the lips.

(1.4.1) figure 1 represents an amplitude waveform of the speech signal ‘She had your dark suit in greasy wash water all year’. From the figure we can see that the speech is a series of steady-state segments with intermediate transitions. Locally stationary speech segments (or frames) are denoted short-time descriptions.

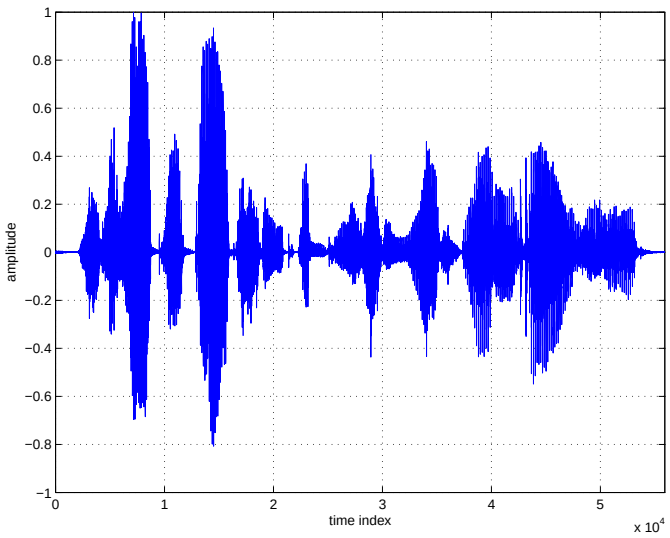


Figure 1: Amplitude waveform of the speech signal ‘She had your dark suit in greasy wash water all year’ uttered by an adult male (3.5 seconds at 16 kHz).

Depending on the manner of excitation, speech segments can be divided into voiced and unvoiced speech sounds. A voiced speech sound is generated from a quasi-periodic vocal-cord sound with a fundamental frequency or pitch usually found to be below a few hundred Hertz. An unvoiced speech sound is generated from a random sound produced by turbulent airflow.

(1.4.2) On the figure 2 we have plots of two frames, one representing the voiced sound /a/ in ‘dark’, and the other representing unvoiced sound /S/ in ‘wash’. Let’s look closer at the some characteristics of the two types of excitation.

(1.4.3) As expected, we observe clear periodicity in the voiced sound, with the fundamental period of around 130 samples. There is no obvious periodicity in the unvoiced sound—it looks random.

The amplitudes of the voiced sounds are approximately 10 times higher than

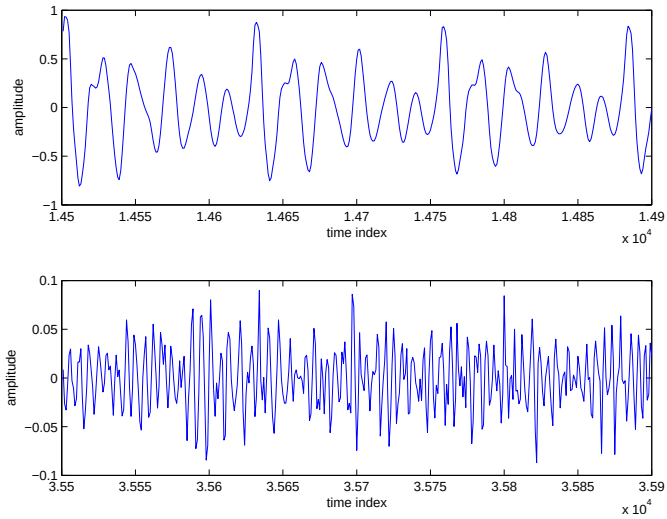


Figure 2: Plots of the two speech frames of 400 samples (25 ms) from the speech signal plotted at figure 1. Top: /a/ in 'dark' (sample numbers 14501–14900), bottom: /S/ in 'wash' (sample numbers 35501–35900).

those of the unvoiced sound. In other words, the average power of voiced sounds is much higher than the average power of the unvoiced sounds.

The number of zero crossings is much larger for the unvoiced sound than for the voiced sound. In this particular case of voiced sound we have 12 zero crossings in one period. In the same time frame, the number of zero crossings for the unvoiced sound is well above 40.

(1.4.4) For time-variant signal $x(n)$, the short-time power $P_x(m)$ can be measured for the N-length frame ending at time m, i.e.,

$$P_x(m) = \frac{1}{N} \sum_{n=m-N+1}^{m} |x(n)|^2$$

Each time the window is shifted one sample, the power P_x could be recalculated, however, it is easier to update the previous value of P_x as

$$P_x(m) = P_x(m-1) + \frac{1}{N}(|x(m)|^2 - |x(m-N)|^2)$$

The MATLAB function `stpower.m` calculates the short-time power of a signal using a sliding window.

```
function Px = stpower(x,N)

M  = length(x);
Px = zeros(M,1);
Px(1:N) = x(1:N)'*x(1:N)/N;
```

```
for (m=(N+1):M)
  Px(m) = Px(m-1) + (x(m)^2 - x(m-N)^2)/N;
end
```

(1.4.5) A short-time zero crossing measure for the N-length interval ending at time m is

$$Z_x(m) = \frac{1}{N} \sum_{n=m-N+1}^{m} \frac{|signx(n) - signx(n-1)|}{2}$$

The MATLAB function `stzerocross.m` implements $Z_x(m)$ using a sliding window.

```
function Zx = stzerocross(x,N)

M  = length(x);
Zx = zeros(M,1);
Zx(1:N+1) = sum(abs(sign(x(2:N+1)) - sign(x(1:N))))/(2*N);

for (m=(N+2):M)
  Zx(m) = Zx(m-1) + (abs(sign(x(m)) - sign(x(m-1))) ...
          - abs(sign(x(m-N)) - sign(x(m-N-1))))/(2*N);
end
```

(1.4.6) We used short-time power and short-time zero crossing measure to analyze the utterance ‘four’.

```
load digits;
N = 300;
x = digits.four1;
Px = stpower_r(x,N);
Zx = stzerocross_r(x,N);
plot([Px*1e-5 Zx x/2000])
```

On the figure 3 we have the plot of the speech signal ‘four’, together with the short-time power and zero crossing measure. From the plot of the speech signal we can easily see that the utterance ‘four’ consists of the unvoiced part /f/ and the voiced part /o/ . We can see that the short-time power of the unvoiced part is close to zero, while for the voiced part it reaches 10 000 times larger values. The short-time zero crossing measure shows opposite behavior, but the differences between voiced and unvoiced part are not so drastic in this case. The unvoiced part of the utterance has the short-time zero crossing measure that is approximately five times larger than the short-time zero crossing measure for the voiced part.

Short-time power and zero crossing measure can be used for initial voiced/unvoiced segmentation. For a given speech signal, short-time power and zero crossing measure can be calculated using the sliding window of a certain length. The middle sample in the window can be labeled voiced if the short-time power is above a certain threshold and the short-time zero crossing measure below certain threshold. Power thresholds can be expressed relative to maximal values of short-time power.

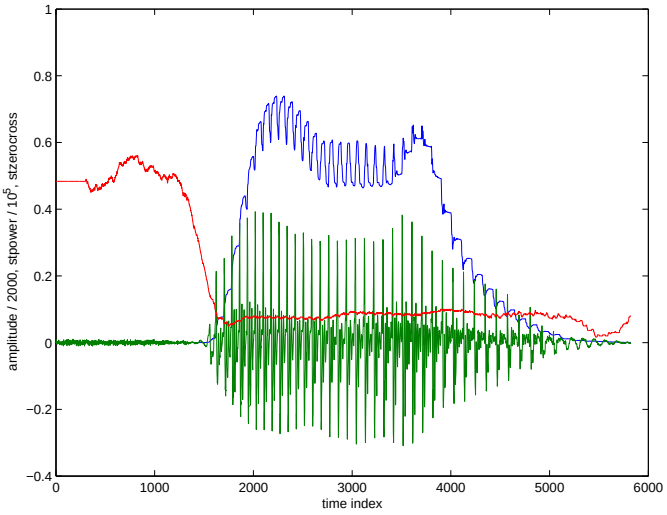


Figure 3: Scaled amplitude waveform of the utterance ‘four’ (green), together with the scaled short-time power (blue) and zero crossing measure (red) obtained using the sliding window of 300 samples (30 ms at 10 kHz).

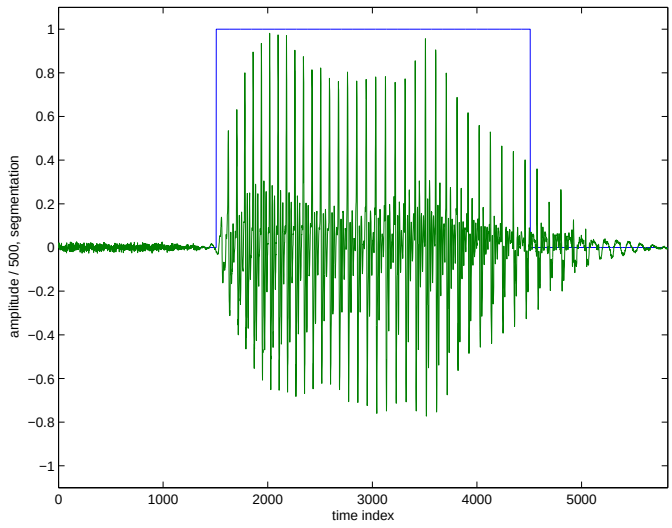


Figure 4: Amplitude waveform of the speech signal ‘four’ (green), and the initial voiced/u-vioced segmentation (blue). Segmentation is based on short-time power and zero crossing measure as on the figure 3. Thresholds used were 0.1 for short-time power, and 0.3 for short-time zero crossing measure.

MATLAB function `voiunvoi.m` implements this segmentation.

```
function voi = voiunvoi(x,N,Pth,Zth)

Px = stpower(x,N);
Zx = stzerocross(x,N);

voi = (Px>Pth*max(Px)) & (Zx<Zth);
voi = [voi(fix(N/2)+1:length(voi));voi(length(voi))*ones(fix(N/2),1)];
```

The results of using function `voiunvoi` on the speech signal 'four' are shown on figure 4.

```
Sx=voiunvoi(x,300,0.1,0.3);
plot([Sx, x/800])
```

The classical discrete-time model for the speech production process assumes that the sound-generating excitation is linearly separable from the vocal tract filter. The vocal tract changes shape relatively slowly with time, and thus it can be modeled as a slowly time-varying filter which imposes its frequency-response properties on the spectrum of the excitation.

A voiced speech sound can be modeled by a sequence of impulses, which are spaced by a fundamental period equal to the pitch period. This signal then excites a linear filter whose impulse response equals the vocal-cord sound pulse.

An unvoiced speech sound is generated from an excitation which consists simply of a white noise source.

(1.4.7) Looking at the two speech frames we analyzed in the task (1.4.2), figure 2, it is not difficult to recognize elements from this model. Voiced sound can be modeled by the filtered impulse train, and the unvoiced sound can be modeled by filtered white noise.

(1.4.8) On figure 5 we have plots of four vowels in frames of 300 samples (30 ms at 10 kHz) and we'll try to estimate the pitch period for each utterance. By looking at the plot of the speech signal we can conclude that for all four plots there is almost the same distance between two neighboring pitch peaks, so the pitch period is almost the same for each vowel. Let's estimate it.

The distance between two neighboring pitch peaks is approximately 90 samples, so we have $T_p = 90$. We know the sampling frequency $F_s = 10$ kHz, so we have

$$T_p = \frac{t_p}{F_s} = \frac{90}{10\text{ kHz}} = 9\text{ ms}$$

$$F_p = f_p \cdot F_s = \frac{F_s}{t_p} = \frac{10\text{ kHz}}{90} \approx 111\text{ Hz}$$

The pitch period is 9 ms, and the pitch frequency 111 Hz. We can also conclude that the speaker is a man, since the pitch frequency of 111 Hz falls into the range typical for men. This is confirmed by listening to the speech signal.

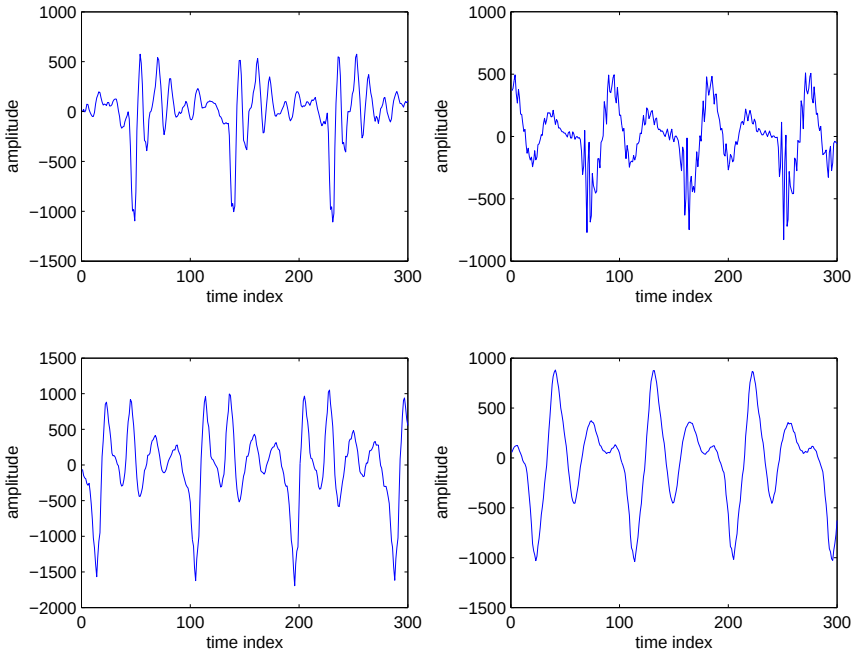


Figure 5: Plots of the four vowels in frames of 300 samples (30 ms at 10 kHz). Top left: /a/, top right: /i/, bottom left: /o/, bottom right: /u/.

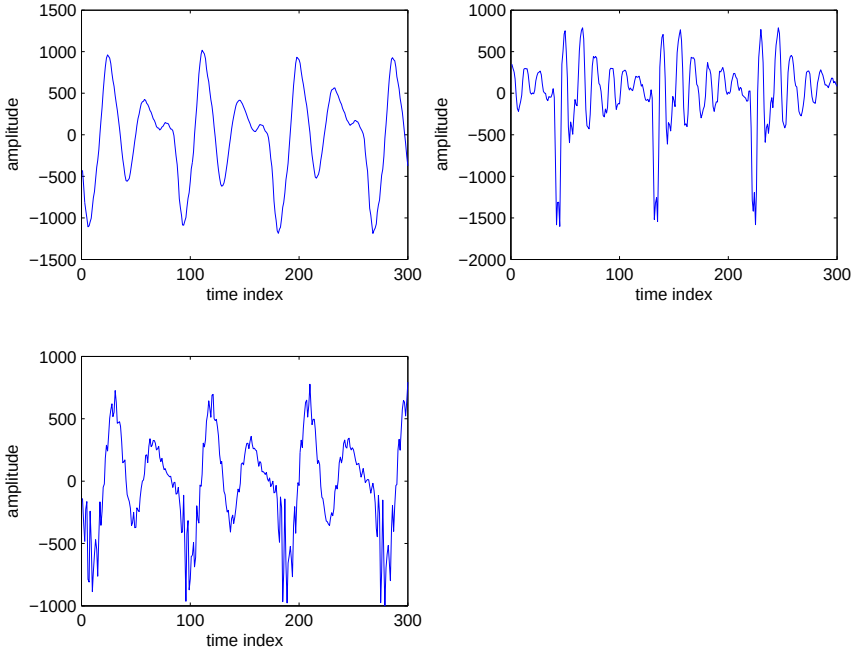


Figure 6: Plots of the three vowels in frames of 300 samples (30 ms at 10 kHz). Those plots should be compared with plots on figure 5.

(1.4.9) On the figure 6 we see the plots of another three vowels. We can try to determine which vowels those plots likely represent by comparing the plots on the figure 6 with the plots on the figure 5. We can see that the top left plot is similar to the plot of vowel /u/, top right plot is similar to the plot of vowel /a/, and the bottom left plot is similar to the plot of vowel /i/, so we guessed that the plots represent vowels /u/, /a/ and /i/. By listening to the speech signals we could verify that our guess was correct.

2.2 Speech Signals in the Frequency Domain

The Fourier transform $X(\omega)$ is a continuous function of frequency, so in the digital domain the sampled spectrum is used to represent an aperiodic signal $x(n)$ of length L, which leads to the discrete Fourier transform (DFT) pair

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j2\pi kn/N} \qquad \leftrightarrow \qquad X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}$$

If $L \leq N$ it is possible to recover $x(n)$ without time-domain aliasing.

Limiting the duration of a sequence $x(n)$ to L samples can be done by multiplying $x(n)$ by a window function $w(n)$ of length L. According to the windowing theorem (PM [3], page 302) we have

$$\hat{x}(n) = x(n)w(n) \qquad \leftrightarrow \qquad \hat{X}(\omega) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\theta)W(\omega-\theta)d\theta$$

The effect of windowing is that energy originating at a single frequency leaks out in the entire frequency range due to the sidelobes of $W(\omega)$, and that the spectral resolution is reduced due to the main lobe width of $W(\omega)$.

(1.5.1) When choosing the window length L to analyze speech signals we have certain limitations. Speech signals are short-time stationary, so the window should be short enough to capture just the steady-state sound. On the other hand, if the window is too short, the unwanted effects of windowing are going to dominate the frequency representation of the speech signal. So, choosing the window length, we need to consider the tradeoff between time resolution and frequency resolution.

(1.5.2) At figure 7 we have plotted one frame of the speech signal ‘The prices have gone up enormously in spite of the technological advances’ corresponding to the voiced /Y/ sound in ‘prices’, together with the magnitude spectrum of the same frame. Since typical speech communication is limited to a bandwidth of 7–8 kHz, used speech signal has been low-pass filtered (3.5 kHz) before sampling. It is evident from the plot of magnitude spectrum that the magnitudes are decreasing for frequencies towards the end of the frequency range.

(1.5.3) In the simplified model for the speech production process, the vocal tract filter models the entire combination of all speech production cavities. A voiced speech sound is then modeled by a sequence of impulses (spaced by a fundamental period equal to pitch period) which excites a vocal tract filter. The

frequency response of the vocal tract filter $H(z)$ determines the short-time spectral envelope of the speech signal. We can verify this by comparing the magnitude spectrum $|H(\omega)|$ for the voiced sound with the DFT based magnitude spectrum of the voiced sound—one could say that $|H(\omega)|$ is obtained by connecting the peak frequencies of DFT spectrum.

(1.5.4) $H(z)$ can be described by an 8–12 order all-pole filter, i.e., 4–6 resonant frequencies (formants) usually denoted F_1, F_2, ... We can try to determine the formants from the DFT based magnitude spectrum. To do that we need to locate 4–6 highest peeks of the DFT magnitude spectrum, find their coefficients k, and then determine which frequencies they represent by calculating

$$f = \frac{k}{N}, \qquad F = f \cdot F_s$$

We need to use the number of points at which the DFT is evaluated $N = 1024$, and the sampling frequency $F_s = 8$ kHz. By looking at figure 7 we found

$$\begin{array}{lllll}
k_1 = 78 & \Rightarrow & f_1 = \frac{78}{1024} = 0.7617 & \Rightarrow & F_1 = \frac{78}{1024} 8 \text{ kHz} = 609 \text{ Hz} \\
k_2 = 158 & \Rightarrow & f_2 = \frac{158}{1024} = 0.1543 & \Rightarrow & F_2 = \frac{158}{1024} 8 \text{ kHz} = 1234 \text{ Hz} \\
k_3 = 278 & \Rightarrow & f_3 = \frac{278}{1024} = 0.2715 & \Rightarrow & F_3 = \frac{278}{1024} 8 \text{ kHz} = 2172 \text{ Hz} \\
k_4 = 395 & \Rightarrow & f_4 = \frac{395}{1024} = 0.3857 & \Rightarrow & F_4 = \frac{395}{1024} 8 \text{ kHz} = 3086 \text{ Hz}
\end{array}$$

(1.5.5) The excitation of the vocal tract filter is not a single unit pulse but a periodic repetition of pulses (pulse train), so the frequency representation of a voiced speech sound is a Fourier series. Ideally we should be able to represent a voiced speech sound by magnitudes of fundamental frequency and its harmonics. The magnitude spectrum of a voiced sound would in that case be a number of equally spaced spikes, where the spike closest to 0 represents fundamental frequency. Due to windowing, we don't have such a ideal spectrum, but we can still see that a magnitude spectrum of a voiced sound is represented by a number of equally spaced peaks (lobes), where the peak closest to 0 represents pitch.

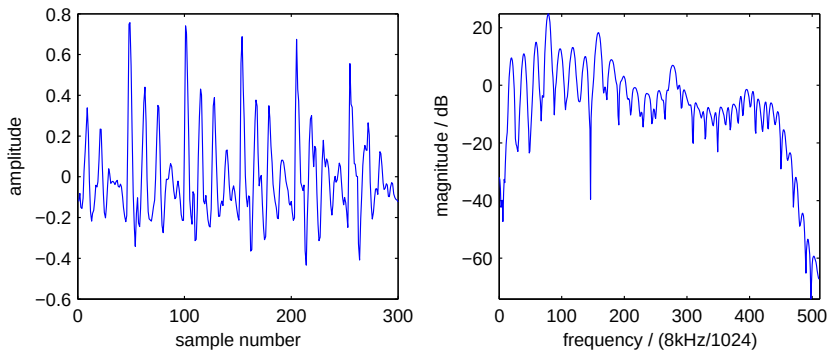


Figure 7: Voiced speech frame (37.5 ms at 8 kHz) corresponding to the /Y/ sound in 'prices'. Left: Amplitude waveform, right: magnitude spectrum evaluated at 1024 points.

Let's try to estimate the pitch frequency from the spectrum. We need to locate the peak closest to 0, find it's coefficient and determine which frequency it represents, using the same method as in the previous task. By looking at the figure 7 we found

$$k_0 = 20$$

$$F_0 = f_0 \cdot F_s = \frac{k_0}{N} F_s = \frac{20}{1024} 8 \text{ kHz} = 156 \text{ Hz}$$

(1.5.6) Frequency representation of a voiced speech sound is a Fourier series. However, the DFT based magnitude spectrum of the voiced sound is be influenced by the spectral characteristics of the window function $w(n)$. The energy originating at a single frequency leaks out in the entire frequency range due to the sidelobes of $W(\omega)$, and the spectral resolution is reduced due to the width of the main lobe of $W(\omega)$. So the magnitude spectrum of a voiced sound does not have a number of equally spaced spikes, but a number of equally spaced lobes, where the width of the lobes is determined by the width of the main lobe of $W(\omega)$.

We can try to estimate the main lobe width by looking at the figure 7. We see that the lobes of the DFT based magnitude spectrum of the voiced sound span across the range of 20 coefficients k. We can express it in terms of normalized frequency using the same method as in the previous two tasks

$$f = \frac{k}{N} = \frac{20}{1024} = 0.0195$$

So we can estimate the width of the main lobe of the $W(\omega)$ to be 0.02 in terms of normalized frequency.

(1.5.7) If the DFT is computed with sufficient spectral resolution, then the harmonics of the pitch frequency will be apparent in the spectrum. Thus, the harmonic product spectrum defined as

$$HPS_x(\omega) = \prod_{r=1}^{R} X(r\omega)$$

can be used to estimate the pitch for some small R, typically five. The MATLAB function `hpspectrum.m` implements $HPS_x(\omega)$.

```
function HPSx = hpspectrum(x,N,R)

K = ceil(N/(2*R));
k = 1:K;
X = fft(x.*hann(length(x)),N);
HPSx = X(k);

for (r=2:R)
    HPSx = HPSx.*X(r*k-r+1);
end
```

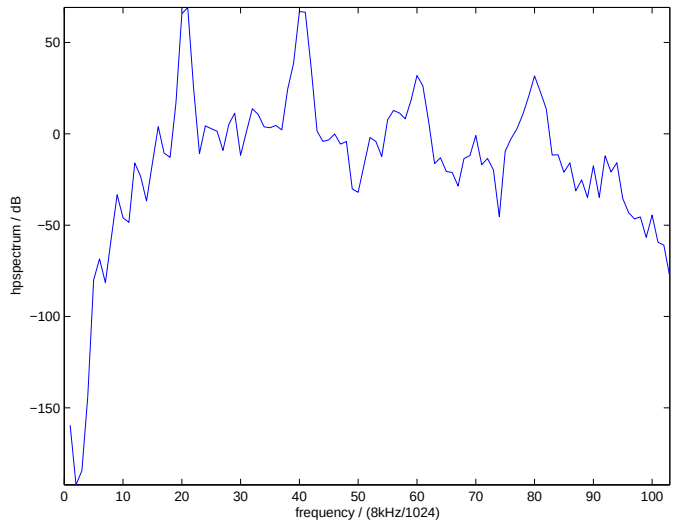


Figure 8: Harmonic product spectrum of the voiced speech frame from the figure 7. We used $R = 5$ and $N = 1024$ to obtain the harmonic product spectrum.

We applied MATLAB function `hpspectrum.m` to the voiced speech frame at figure 7. The harmonic product spectrum of the speech frame is shown at the figure 8. We can use the harmonic product spectrum to estimate the pitch by locating the maximal value of harmonic product spectrum and determining which frequency it represents. By looking at the figure 8 we found

$$k_0 = 21$$

$$F_0 = f_0 \cdot F_s = \frac{k_0}{N} F_s = \frac{21}{1024} 8 \text{ kHz} = 164 \text{ Hz}$$

This result varies a bit from the result obtained in the exercise (1.5.5).

(1.5.8) On the figure 9 we have magnitude spectra of the vowels /i/ as in ‘tree’ and /u/ as in ‘boot’. For each of those spectra we can do the same analysis as for the spectrum on the figure 7, i.e. we can try to determine formants, we can try to estimate pitch period, and we can look at the effects of windowing in the DFT based spectra.

(1.5.9) For unvoiced frames (stochastic signal), the DFT should be viewed as a step toward computing a short-time power density spectrum

$$\Gamma_x(\omega) = E\{X(\omega)X(\omega)^*\}$$

The phase spectrum is not meaningful in the stochastic case.

On the figure 10 we have plotted one frame of the speech signal ‘The prices have gone up enormously in spite of the technological advances’ corresponding to the unvoiced /s/ sound in ‘spite’, together with the magnitude spectrum of the same frame. One should note very small magnitudes of this spectrum.

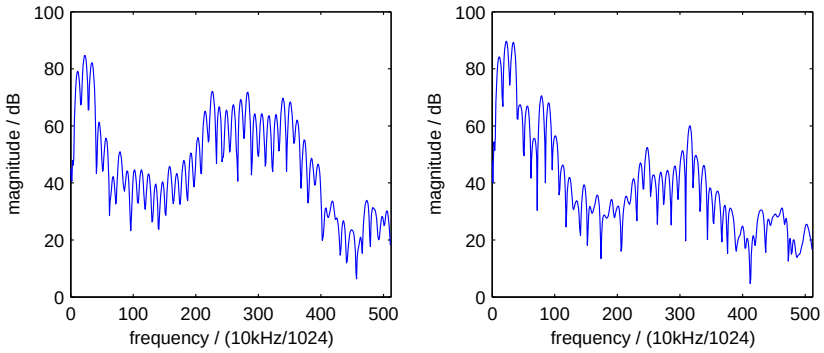


Figure 9: Magnitude spectra of two vowels. Both speech frames had the length of 300 samples, and the magnitude spectra was evaluated at 1024 points. Left: /i/ as in ‘tree’, right: /u/ as in ‘boot’.

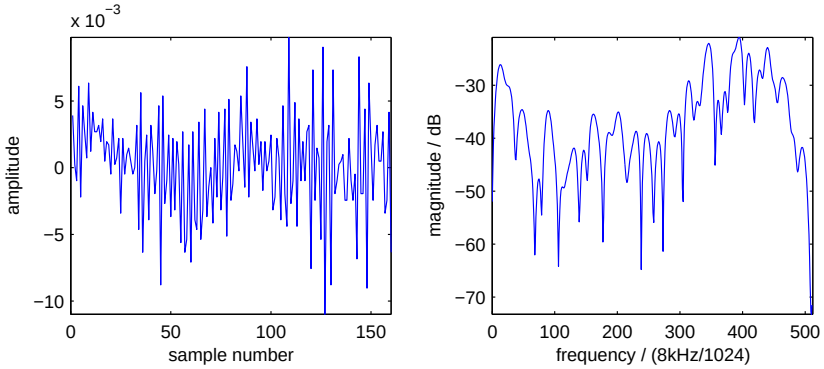


Figure 10: Unvoiced speech frame (20 ms at 8 kHz) corresponding to the /s/ sound in ‘spite’. Left: Amplitude waveform, right: magnitude spectrum evaluated at 1024 points.

2.3 Speech Spectrogram

(1.6.1) On the figure 11 we have an amplitude waveform of the speech signal ‘She had your dark suit in greasy wash water all year’, together with the spectrogram of the same speech signal.

We can see that the formants structure changes relatively slowly in some time intervals, but those intervals are separated with shorter or longer transitions where the formant structure is not evident. The intervals where formant structure is evident and changing slowly correspond to the voiced parts of the sentence, while the transitions correspond to unvoiced parts.

(1.6.2) On the figure 12 we have an amplitude waveform of the sound /i/ glissandos, together with the spectrogram of the same sound. Glissandos (or pitch sweeps) is a musical term that refers to sliding from one pitch to another, produced for example by sliding the finger along a keyboard.

Comparing the spectrogram on the figure 12 with the magnitude spectrum of a voiced sound /i/ on the figure 9, it is possible to recognize the formant spectrum for the /i/ sound—very high magnitudes for very low frequencies (up to 0.5 kHz), followed by a short range of frequencies with small magnitudes (1–1.5 kHz), followed again by a longer range of frequencies with high magnitudes (2–3.5 kHz).

Looking at the spectrogram on the figure 12 it is easy to notice the shift in the pitch frequency. Pitch frequency is represented by the lowest slanted line in the spectrogram (also the line with highest magnitudes) and the shift in the pitch corresponds to the slope of the line. The shift in pitch frequency is reflected in all the harmonics, resulting in spectrogram composed of slanted lines.

(1.6.3) MATLAB function `stpitch.m` estimates the short-time pitch in a sliding window of length N, but only for voiced parts of the speech signal. The function is based on the voiced/unvoiced segmentation `voiunvoi.m` (1.4.6) and the harmonic product spectrum `hpspectrum.m` (1.5.7). If a frame is voiced, the pitch is estimated using maximal value of harmonic product spectrum for that frame.

```
function Fp = stpitch(x,N,Pth,Zth,NFFT,R,Fs)

M = length(x);
Fp = zeros(M,1);
voi = voiunvoi(x,N,Pth,Zth);

for (m=N:fix(N/2):M)
    n = m-N+1:m;
    if all(voi(n))
        HPS = abs(hpspectrum(x(n),NFFT,R));
        k = find(HPS==max(HPS));
        Fp(n) = k;
    end
end

wsave = warning; warning('off');
Fp = (Fp*Fs/NFFT).*(Fp./Fp);
warning(wsave);
```

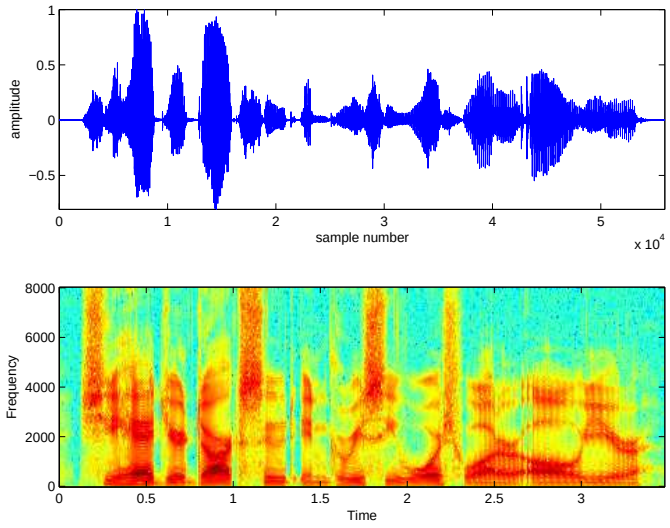


Figure 11: Speech signal ‘She had your dark suit in greasy wash water all year’. Top: amplitude waveform, bottom: spectrogram. Magnitude spectrum was calculated in frames of 256 samples, and DFT was evaluated at 256 points.

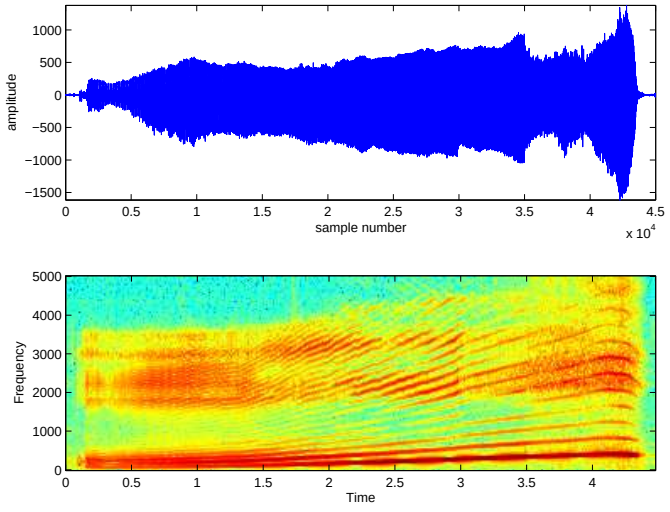


Figure 12: Voiced sound /i/ glissandos. Top: amplitude waveform, bottom: spectrogram. Magnitude spectrum was calculated in frames of 256 samples, and DFT was evaluated at 256 points.

(1.6.4) We applied function `stpitch.m` to two speech signals: the sentence 'She had your dark suit in greasy wash water all year' uttered by an adult male and the same sentence repeated by an adult female.

```
load timit1;
load timit2;
N = 300;
Pth = 0.01;
Zth = 0.3;
NFFT = 1024;
R = 5;
Fs = 16000;
pt1=stpitch_r(timit1,N,Pth,Zth,NFFT,R,Fs);
pt2=stpitch_r(timit2,N,Pth,Zth,NFFT,R,Fs);
plot(pt1)
plot(pt2)
```

Figure 13 contains amplitude waveforms of the speech signals, and the pitch estimations obtained using the function `stpitch.m`.

The pitch range for men is usually between 50–250 Hz, while for women the range falls in the interval 120–150 Hz. The results of our pitch estimation fall roughly into these ranges. The pitch estimated from sentence uttered by a woman reached higher values and is generally higher. There is a certain pitch variation in both sentences, and the pitch variation is larger in the sentence uttered by a woman.

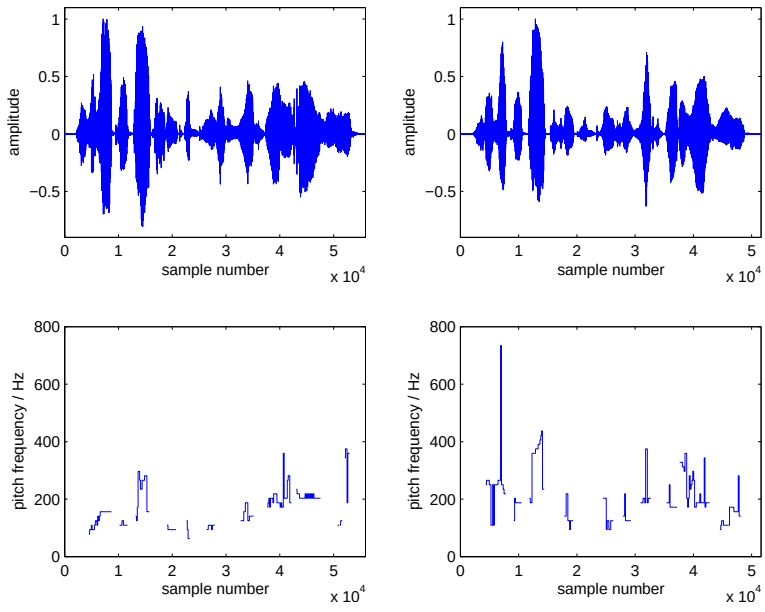


Figure 13: Speech signal 'She had your dark suit in greasy wash water all year'. Left: uttered by an adult male, right: uttered by an adult female, top: amplitude waveform, bottom: pitch estimation using the function `stpitch`.

2.4 Homework

(1.7) In the above exercises, a number of important features has been extracted from frames of speech as these frames move through time. These features are either related to the excitation signals or the vocal-tract. The features that could be used in speech recognition are the features related to the vocal-tract, for example the formants of the magnitude spectrum. The discrete Fourier transform could be used to extract/characterize those features.

3 Linear Prediction Analysis

3.1 Short-term Autocorrelation

The (long-term) autocorrelation of a power signal $x(n)$ is defined as

$$r_x(\eta) = \lim_{M\to\infty} \frac{1}{2M+1} \sum_{n=-M}^{M} x(n)x(n-\eta)$$

where the index η is the time shift or lag parameter. In practice, we are dealing with finite-duration sequences $x(m-N+1), x(m-N+2), \ldots, x(m)$, and the short-term autocorrelation estimate for the N points ending at m may be defined as

$$r_x(\eta; m) = \frac{1}{N-\eta} \sum_{n=m-N+1+\eta}^{m} x(n)x(n-\eta), \qquad 0 \le \eta \le N-1$$

where the autocorrelation sequence for negative lags can be obtained from the relation $r_x(-\eta) = r_x(\eta)$. Autocorrelation function estimates as given by the above mentioned formula can be obtained by using the MATLAB function

`[r,eta] = xcorr(x,eta_max,'unbiased')`

where `x` is the signal vector, `r` is the autocorrelation vector, and `eta` is a vector of lag indices in the range from `-eta_max` to `eta_max`.

(2.1.1) The autocorrelation function for a sinusoid $x(n) = A\sin(2\pi f n + \phi)$ is given by

$$r_x(\eta) = \frac{A^2}{2}\cos(2\pi f \eta), \qquad P_x = r_x(0) = \frac{A^2}{2}$$

We used MATLAB to verify this result for a sinusoidal signal with normalized frequency $f = 0.01$, by using the following MATLAB commands

```
x = 3*sin(2*pi*0.01*(0:499)'+10);
[r,eta] = xcorr(x,100,'unbiased');
plot(0:499,x)
plot(eta,r)
```

On the figure 14 we have a plot of a sinusoidal signal $x(n) = 3\sin(2\pi \cdot 0.01 + 10)$, together with the autocorrelation of the signal. We can see that the autocorrelation is periodic with the period $N = 100$, and the same applies to the sinusoidal signal because $N = \frac{1}{f} = \frac{1}{0.01} = 100$. We can also see that $r_x(0) = 4.5$ which is the average power of the sinusoidal signal $P_x = \frac{A^2}{2} = \frac{3^2}{2} = 4.5$.

(2.1.2) The autocorrelation function for white noise (with variance 1) is given by

$$r_w(\eta) = \delta(\eta), \qquad P_w = r_w(0) = 1$$

We used MATLAB to verify this result for a noise generated by the MATLAB func-

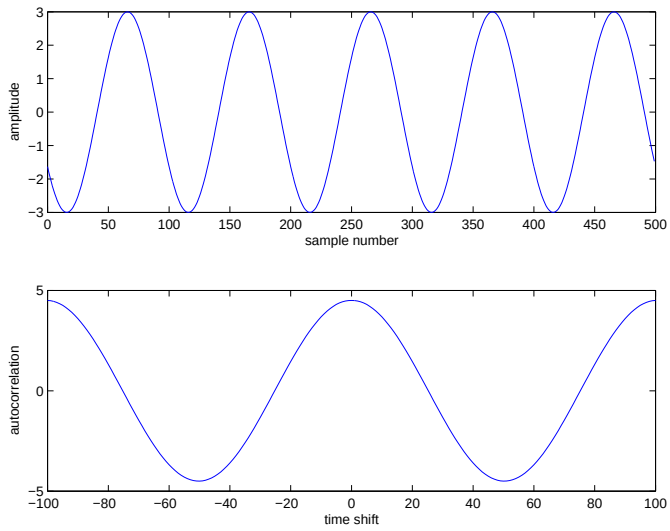


Figure 14: The sinusoidal signal $x(n) = 3\sin(2\pi \cdot 0.01 + 10),\ \ 0 \leq n \leq 499$. Top: amplitude waveform, bottom: short-term autocorrelation over the range $[-100, 100]$.

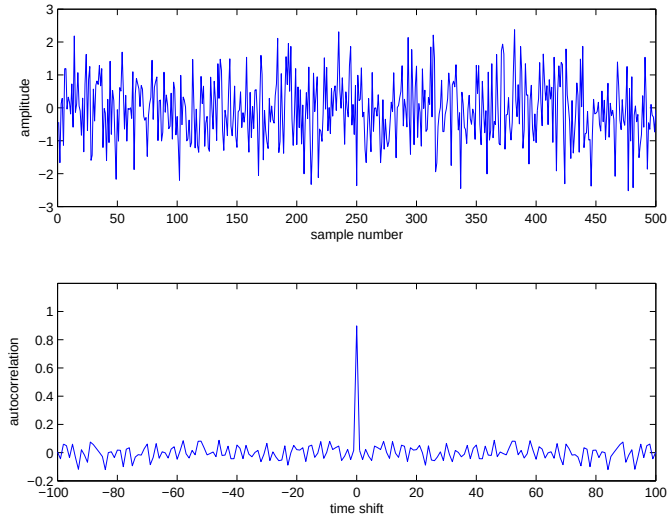


Figure 15: The white noise with mean zero and variance one. Top: amplitude waveform, bottom: short-term autocorrelation over the range $[-100, 100]$.

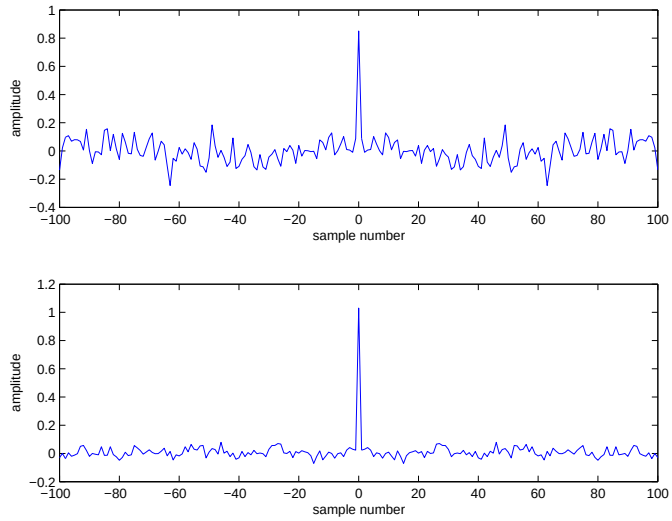


Figure 16: Autocorrelation of two white noise signals with different lengths. Top: signal length 200 samples, bottom: signal length 1000 samples.

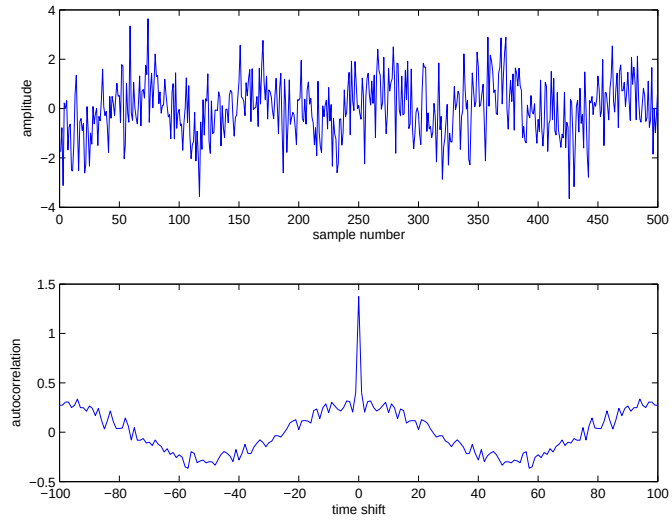


Figure 17: The sinusoidal signal $x(n) = 0.8 \cdot \sin(2\pi \cdot 0.01 + 10)$ corrupted with the white noise with mean zero and variance one. Top: amplitude waveform, bottom: short-term autocorrelation over the range $[-100, 100]$.

tion `randn` by using the following MATLAB commands.

```
w = randn(500,1);
[r,eta] = xcorr(w,100,'unbiased');
plot(w)
plot(eta,r)
```

On the figure 15 we have a plot of a random noise, together with the autocorrelation of the noise. We can see the large peak at $r_w(0) \approx 1$, while the rest of the autocorrelation is very small.

The accuracy of the estimated autocorrelation function depends on the signal length—estimation is more accurate for the longer signals (larger average interval). To verify this, on the figure 16 we plotted the autocorrelation of two signals with different lengths by using the following commands

```
w1=randn(200,1);
[r1,eta]=xcorr(w1,100,'unbiased');
w2=randn(1000,1);
[r2,eta]=xcorr(w2,100,'unbiased');
plot(eta,r1)
plot(eta,r2)
```

(2.1.3) Now we added the white noise (with variance one) to the sinusoidal signal $x(n) = 0.8 \cdot \sin(2\pi \cdot 0.01 + 10)$ with normalized frequency $f = 0.01$. The observed signal is $y(n) = x(n) + w(n)$.

```
x = 0.8*sin(2*pi*0.01*(0:499)'+10);
w = randn(500,1);
y=x+w;
[r,eta] = xcorr(y,100,'unbiased');
plot(y)
plot(eta,r)
```

On the figure 17 we have a plot of the observed signal, together with the autocorrelation of the noise. We can see the large peak at $r_y(0)$ which is the contribution of the random noise, but it is still possible to determine the period of the sinusoidal signal from the autocorrelation. If we ignore the peak at $r_y(0)$ and 'smooth' the rest, we can still see that the autocorrelation is periodic with the period $N = 100$, which is also the period of the sinusoidal signal.

(2.1.4) We can use the short-term autocorrelation on the speech signal. On the figure 18 we have a amplitude waveform of the utterance 'three'. We have chosen two 256 point windows from this speech signal: the first window corresponds to the voiced phoneme /i/, and the second window contains an unvoiced region corresponding to sound /T/.

```
load digits;
x = digits.three1;

m = 2756;
```

```
N = 256;
n = m-N+1:m;
[r,eta] = xcorr(x(n),250,'unbiased');
plot(1:256, x(n))
plot(eta,r)

m = 500;
N = 256;
n = m-N+1:m;
[r,eta] = xcorr(x(n),250,'unbiased');
plot(1:256, x(n))
plot(eta,r)
```

Figure 19 shows the amplitude waveform of the voiced window, together with the autocorrelation of the same window. It is obvious that the short-term autocorrelation captures the periodicity of the voiced sound and we can easily determine the period to be $N = 79$. There is obviously not a lot of noise in this signal, since the peak at $r_x(0)$ is just a little bit higher than the other peaks.

The discrete period $N = 79$ corresponds to continuous time period of $79 \cdot \frac{1}{10\text{ kHz}} = 7.9$ ms, since the sampling frequency $F_S = 10$ kHz. The pitch frequency $F_P = \frac{10\text{ kHz}}{79} = 126$ Hz.

Figure 20 shows the amplitude waveform of the unvoiced window, together with the autocorrelation of the same window. The short-term autocorrelation of the unvoiced sound looks like the autocorrelation of the white noise—there is a large peak at $r_x(0)$ while the rest is relatively small and without obvious periodicity.

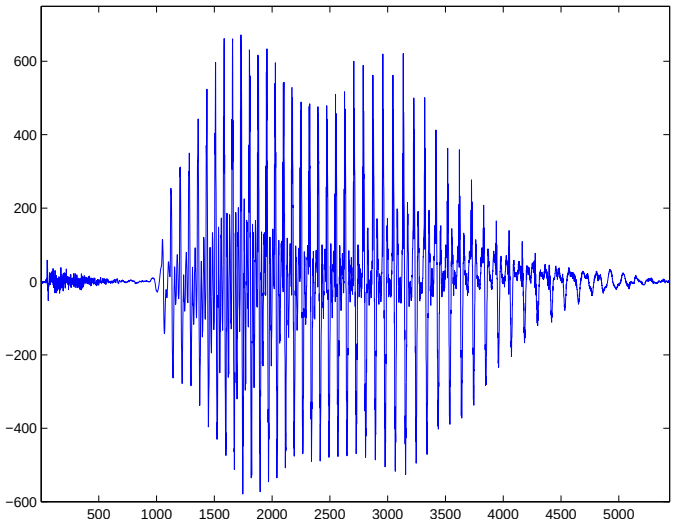


Figure 18: Amplitude waveform of the speech signal ‘three’ sampled at 10 kHz.

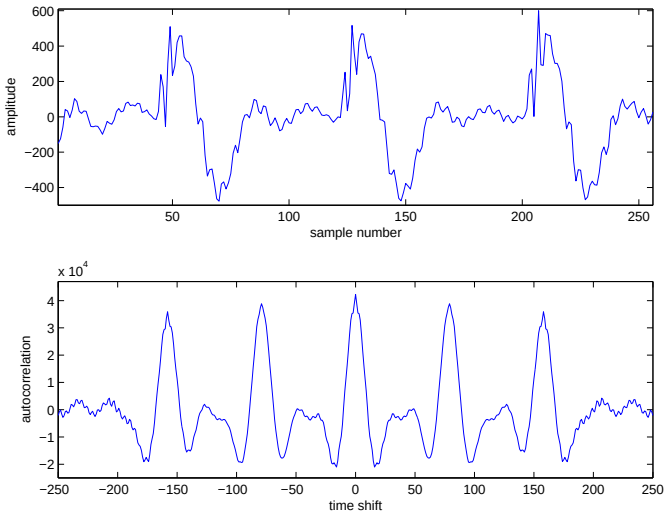


Figure 19: One 256-points frame of the signal from the figure 18 corresponding to the voiced sound /i/ (sample numbers 2501–2756). Top: amplitude waveform, bottom: short-term autocorrelation over the range $[-250, 250]$.

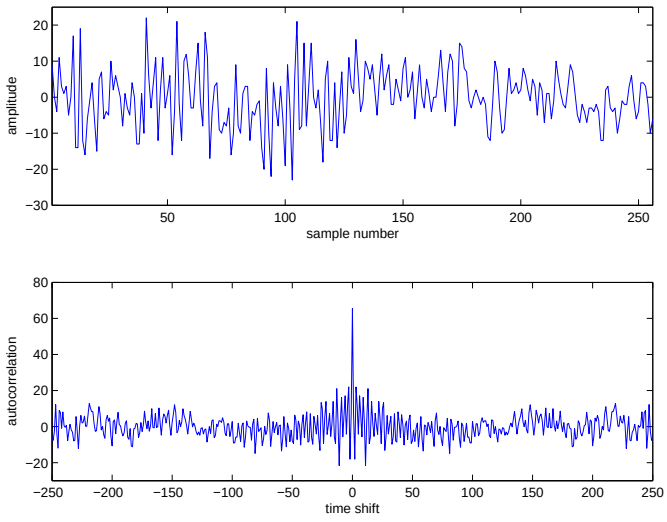


Figure 20: One 256-points frame of the signal from the figure 19 corresponding to the unvoiced sound /T/ (sample numbers 245–500). Top: amplitude waveform, bottom: short-term autocorrelation over the range $[-250, 250]$.

3.2 Autocorrelation Method for LP Analysis

In the discrete-time model for speech production, speech can be modeled as the filter $\Theta(z)$, which combines all the processes of producing a sound, driven by an impulse-train or white noise sequence

$$S(z) = E(z) \cdot \Theta(z)$$

In an all-pole model of the system function we have

$$\hat{\Theta}(z) = \frac{1}{\hat{A}(z)} = \frac{1}{1 - \sum_{i=1}^{M} \hat{a}(i) z^{-i}}$$

and our objective is to estimate the coefficients $\hat{a}(i)$ that constitute (together with pitch and gain) a parametric representation for the waveform. The coefficients $\hat{a}(i)$ are referred to as the linear prediction coefficients, and their estimation is termed linear prediction analysis.

The name linear prediction comes from the fact that from

$$[1 - \sum_{i=1}^{M} \hat{a}(i) z^{-i}] \cdot S(z) = E(z)$$

in the time domain we have

$$s(n) = \sum_{i=1}^{I} \hat{a}(i) s(n-i) + e(n)$$

so the speech sample $s(n)$ is approximated as a linear combination of past speech samples.

We estimate the coefficients $\hat{a}(i)$ by minimizing the expectation of squared prediction error $\hat{e}(n)$

$$\hat{e}(n) = s(n) - \hat{s}(n) = s(n) - \sum_{i=1}^{I} \hat{a}(i) s(n-i)$$

and the solution of minimization is the following system of linear equations

$$\sum_{i=1}^{M} \hat{a}(i) r_s(\eta - i) = r_s(\eta)$$

In the matrix form the system of linear equations becomes

$$\mathbf{R}_s(m) \hat{a}(m) = \mathbf{r}_s(m)$$

or written out

$$\begin{pmatrix} r_s(0;m) & r_s(1;m) & \cdots & r_s(M-1;m) \\ r_s(1;m) & r_s(0;m) & \cdots & r_s(M-2;m) \\ \vdots & \vdots & \ddots & \vdots \\ r_s(M-1;m) & r_s(M-2;m) & \cdots & r_s(0;m) \end{pmatrix} \begin{pmatrix} \hat{a}(1;m) \\ \hat{a}(2;m) \\ \vdots \\ \hat{a}(M;m) \end{pmatrix} = \begin{pmatrix} r_s(1;m) \\ r_s(2;m) \\ \vdots \\ r_s(M;m) \end{pmatrix}$$

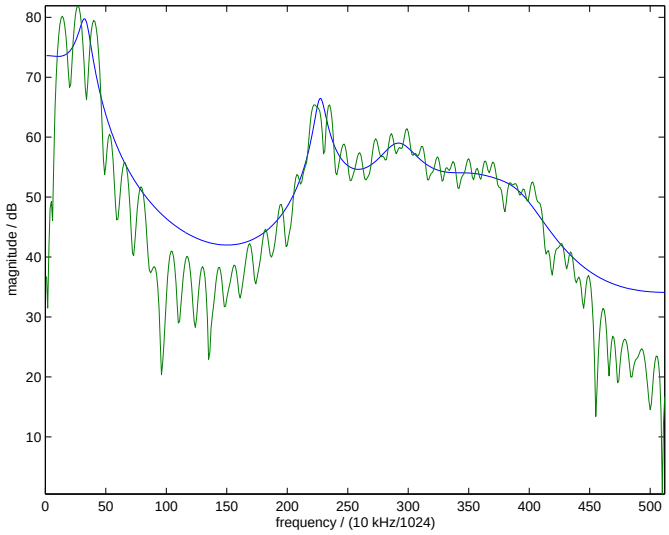


Figure 21: The magnitude spectrum of the 256 samples frame corresponding to the voiced phoneme /i/ in the utterance ‘three’ (green), together with the (scaled) response of the order-14 linear prediction filter (blue).

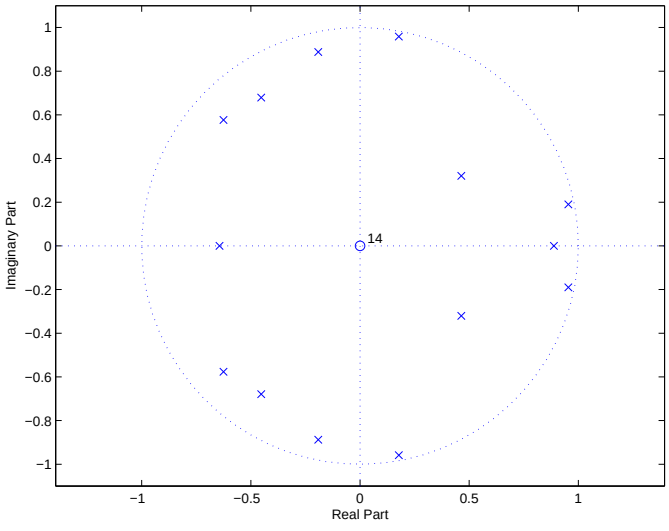


Figure 22: Zero-pole plot of the 14-order linear prediction filter from the figure 21.

Matrix $\mathbf{R}_s$ is called autocorrelation matrix and the vector $\mathbf{r}_s$ is called autocorrelation vector.

(2.2.1) To start with, we can find the solution to the equation

$$\mathbf{R}_s(m)\hat{a}(m) = \mathbf{r}_s(m)$$

using the 'brute force', i.e. finding the inverse of the autocorrelation $M \times M$ matrix. The cost of this method is generally $O(M^3)$.

We implemented this approach in MATLAB by finding the order-14 predictor for the voiced phoneme /i/ in the utterance 'three' using the following commands

```
x = digits.three1;
m = 2756;
N = 256;
n = m-N+1:m;
M = 14;
[r,eta] = xcorr(x(n),M,'biased');
Rx = toeplitz(r(M+1:2*M));
rx = r(M+2:2*M+1);
a = Rx\rx;

NFFT = 1024;
k = 1:NFFT/2;
X = fft(x(n).*hann(N),NFFT);
Theta = 1./fft([1; -a],NFFT);
plot(k,20*log10(abs([353*Theta(k) X(k)])))
```

On the figure 21 we plotted the response of the filter obtained using the calculated linear prediction coefficients together with the magnitude spectrum of the 265 samples frame that we used. It is clear that the filter determines spectral envelope of the speech signal.

On the figure 22 we included the zero-pole plot of the obtained linear prediction filter to see how in the all-pole model, poles are used to determine the shape of filter response.

(2.2.2) The energy $\xi(m) = \frac{1}{N}\sum_{m=-\infty}^{\infty}\hat{e}^2(n;m)$ of the prediciton residual sequence $\hat{e}(n;m)$ is given by $\xi(m) = r_s(0;m) - r_s^T(m)\hat{a}(m)$. For the solution in previous question, this energy (relative to $r_s(0;m)$) is `1 - rx'*a/r(M+1) = 0.0373`.

3.3 Levison-Durbin Recursion

Levison-Durbin recursion is an efficient way to solve the normal equation $\mathbf{R}_s(m)\hat{a}(m) = \mathbf{r}_s(m)$, exploiting the fact that $\mathbf{R}_s(m)$ is Toeplitz, symmetric, and positive definite, and the right-hand side $\mathbf{r}_s(m)$ has a special relation to the elements of $\mathbf{R}_s(m)$.

(2.3.1) MATLAB function `durbin.m` implements Levison-Durbin recursion. The input arguments are the vector `r` containing autocorrelation coefficients and the prediction order `M`. The output arguments are estimated LP parameters in the vector `a`, prediction error energies in the vector `xi`, and estimated reflection coefficients in the vector `kappa`.

```
function [a,xi,kappa] = durbin(r,M)

kappa = zeros(M,1);
a     = zeros(M,1);
xi    = [r(1); zeros(M,1)];

for (j=1:M)
  kappa(j) = (r(j+1) - a(1:j-1)'*r(j:-1:2))/xi(j);
  a(j)     = kappa(j);
  a(1:j-1) = a(1:j-1) - kappa(j)*a(j-1:-1:1);
  xi(j+1)  = xi(j)*(1 - kappa(j)^2);
end
```

(2.3.2) Now we use function `durbin.m` to solve the same problem as in the task (2.2.1) To compare the output vector `a` from this function with the one obtained using the 'brute force' we printed the parameters in 15 decimal places, and could see the difference in the 1–2 last decimal places. In fact, the largest difference of the two corresponding parameters is $2.6 \cdot 10^{-13}$.

(2.3.3) The energy $\xi(m)$ of the prediction residual sequence should be non-increasing for the increasing model order i.e. the prediction should be improving with the increasing model order. We verified this by plotting the `xi` vector on the figure 23.

The sequence $\xi^{(j)}(m)$ can be used to determine when the sufficient model order has been reached, by for example by stopping the recursion when the slope $\frac{\xi^{(j)}(m)-\xi^{(j-1)}(m)}{j-(j-1)} = \xi^{(j)}(m) - \xi^{(j-1)}(m)$ is small enough.

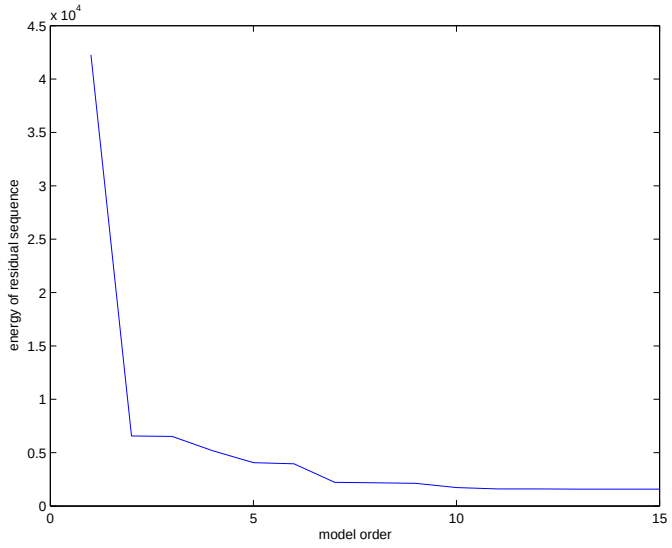


Figure 23: The plot of the energy $\xi(m)$ of the prediction residual sequence for the increasing model order.

(2.3.4-6) Those three tasks are concerned with the conversion between the linear prediction parameters $\hat{a}(i;m)$ and the M reflected coefficients $\kappa(i;m)$. We wrote MATLAB functions that implement the conversion in both directions, and we tested the conversion.

The maximal difference between corresponding parameters when converting from linear prediction parameters $\hat{a}(i;m)$ to reflected coefficients $\kappa(i;m)$ was $3.03 \cdot 10^{-15}$, and the maximal difference when converting form $\kappa(i;m)$ to $\hat{a}(i;m)$ was 0. In conclusion, both the 'brute force' solution to the normal equations, and the Levison-Durbin recursion give the same result, the difference is the efficiency. Levison-Durbin is much more efficient, and it solves normal equation in $O(M^2)$ steps.

3.4 Inverse Filtering Computation

The prediction error sequence $\hat{e}(n;m)$ can be obtained by filtering $s(n;m)$ through the estimated inverse system

$$\hat{e}(n;m) = s(n;m) - \sum_{i=1}^{M} \hat{a}(i;m)s(n-i;m)$$

or in the Z-transform domain

$$\hat{E}(z) = \hat{A}(z)S(z), \qquad \hat{A}(z) = 1 - \sum_{i=1}^{M} \hat{a}(i)z^{-i}$$

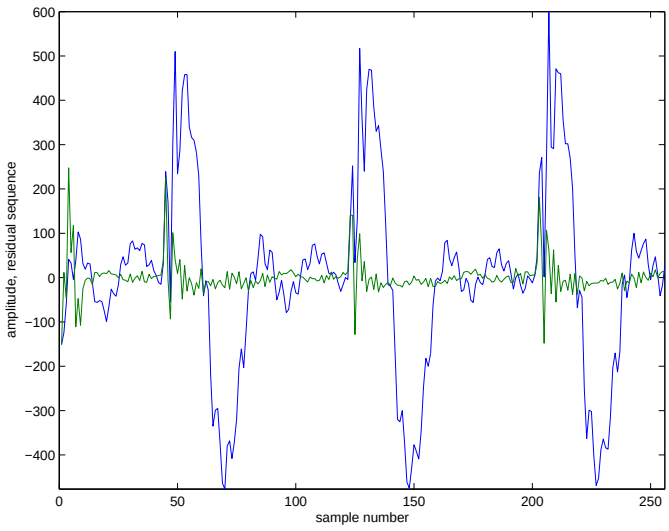


Figure 24: Frame of 256 samples corresponding to the voiced phoneme /i/ in the utterance 'three' (blue), together with the residual sequence $\hat{e}(n;m)$ of the order-14 linear prediction filter (green).

(2.4.1) We found the residual sequence $\hat{e}(n;m)$ for the order-14 predictor from the task (2.2.1) by inverse filtering and we plotted it on the figure 24 together with the amplitude waveform of the original frame. Impulse train excitation sequence is easily recognizable from the residual sequence.

(2.4.2) MATLAB function `lpcauto.m` preforms frame based linear prediction analysis on a speech signa. Input parameters are the signal `x`, the prediction order `M`, the window function `win`, and the overlap between adjacent frames `Olap`. Function returns linear prediction coefficients in the matrix `ar`, prediction error energies in the matrix `xi`, residual signal in the vector `e`, and the index of the last sample in each frame in the vector `m`.

```
function [ar,xi,e,m] = lpcauto(x,M,win,Olap)

Nx = length(x);
N = length(win);
if (N == 1)
  N   = win;
  win = ones(N,1);
end
F = fix((Nx-Olap)/(N-Olap));

ar = zeros(M+1,F);
xi = zeros(M+1,F);
e  = zeros(Nx,1);
m  = zeros(F,1);

n  = 1:N;
n1 = 1:Olap;
n2 = N-Olap+1:N;
n3 = Olap+1:N;

win1 = win(n1)./(win(n1)+win(n2)+eps);
win2 = win(n2)./(win(n1)+win(n2)+eps);

for (f=1:F)
  [r,eta] = xcorr(x(n).*win,M,'biased');
  [a,xi(:,f),kappa] = durbin(r(M+1:2*M+1),M);
  ar(:,f) = [1; -a];
  ehat = filter(ar(:,f),1,x(n));
  e(n) = [e(n(n1)).*win2 + ehat(n1).*win1; ehat(n3)];
  m(f) = n(N);
  n = n + (N-Olap);
end
```

(2.4.3) We applied function `lpcauto.m` to the speech signal 'She had your dark suit in greasy wash water all year', using half overlapping Hann windows of length 256 samples, and prediction order 14. On the figure 25 we plotted the amplitude waveform of the speech signal, together with residual sequence.

By listening to residual sequence we concluded that sf one din't know the original signal beforehand, it would be hard to understand residual sequence. Knowing

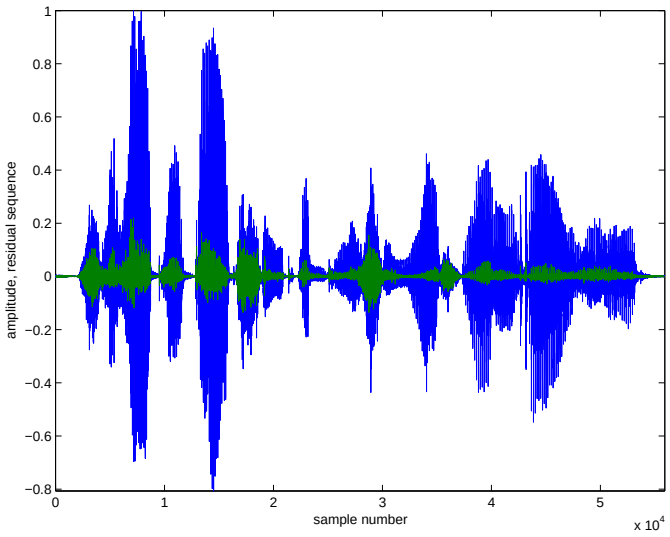


Figure 25: Amplitude waveform of the speech signal ‘She had your dark suit in greasy wash water all year’ (blue), together with the residual sequence of the order-14 linear prediction filter (green).

the original speech signal, we could still understand it, which implies that some of the vocal tract spectrum is passing through the inverse filter. We could hear that the residual sequence is quite clear for the sounds /S/ as in ‘she’, ‘wash’ and /s/ as in ‘suit’, ‘greasy’. This can also be observed on the plot where we see that for those sounds, residual sequence has relatively high amplitudes relative to original signal.

3.5 Formant Estimation

Formant frequencies and bandwidths are principal analytical features of the speech spectrum, and a simple technique for formant estimation could be based on peak finding in an LP-derived magnitude spectrum. It is easier to find peaks of the LP-derived magnitude spectrum, than to estimate formants directly from the spectrum of the sound.

(2.5.1) MATLAB function `lpcplot.m` makes a 2D plot of the magnitude spectrum, if an input is a vector of LP coefficients, or it makes a 3D mesh plot of magnitude spectra, if an input is a matrix of LP coefficients.

```
function lpcplot(A,Nfft,Fs,m)

[M,N] = size(A);
if (N==1)
  [Theta,F] = freqz(1,A,Nfft,Fs);
  plot(F,20*log10(abs(Theta)));
  xlabel('Frequency, {\it F}  [Hz]');
  ylabel('Magnitude, |\theta(\omega)|  [dB]');
```

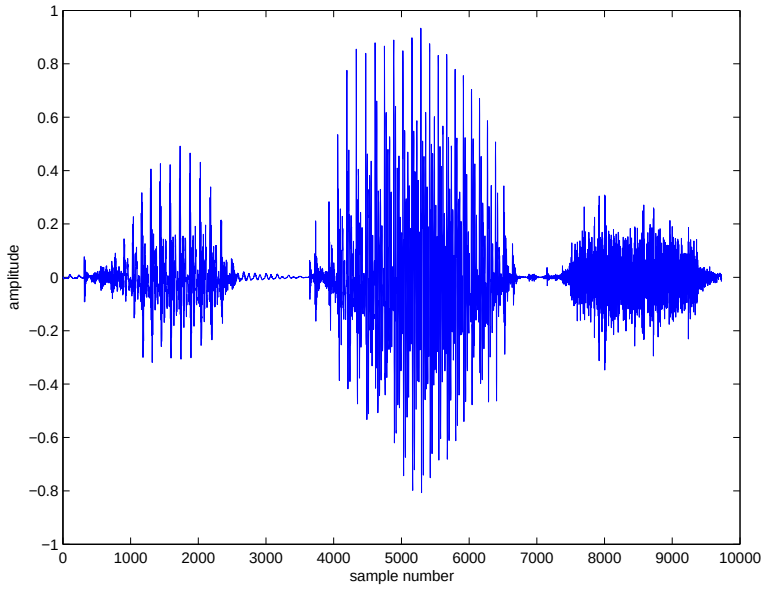


Figure 26: A plot of larger frame ($38 \cdot 256$ samples) taken from the utterance plotted at figure 26 (sample numbers 9216–18944), corresponding to the segment 'your dark s…'.

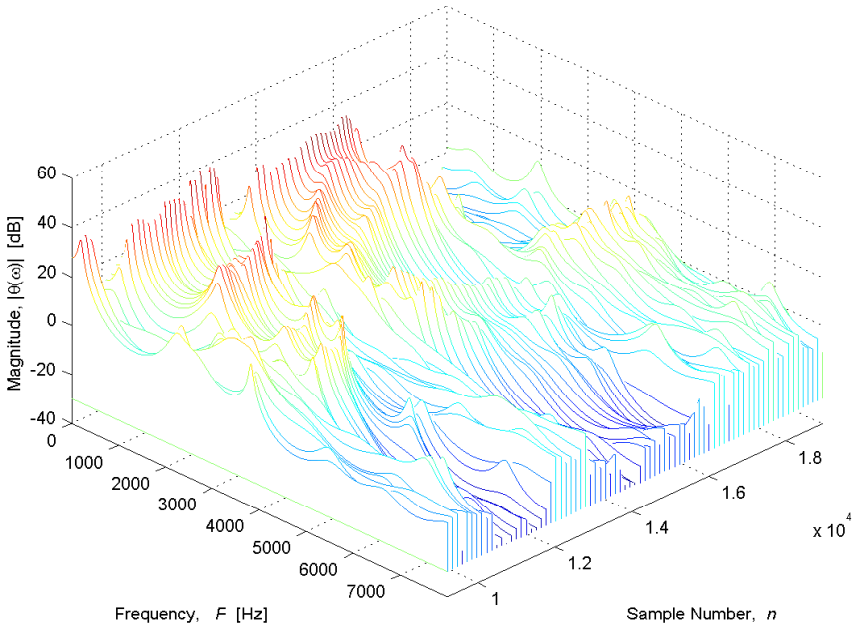


Figure 27: A 3D mesh plot containing the LP-derived magnitude spectra of 76 half-overlapping frames of 256 samples corresponding to the utterance plotted at figure 26.

```
else
  Theta = zeros(Nfft,N);
  for (n=1:N)
    [Theta(:,n),F] = freqz(1,A(:,n),Nfft,Fs);
  end
  MeshHndl = meshz(m,F,20*log10(abs(Theta)));
  axis ij; view(-45,45); set(MeshHndl,'MeshStyle','Column');
  axis tight; axis 'auto y'; axis 'auto z';
  xlabel('Sample Number, {\it n}');
  ylabel('Frequency, {\it F}  [Hz]');
  zlabel('Magnitude, |\theta(\omega)|  [dB]');
end
```

(2.5.2) Now we can look at the LPC spectra for a number of frames corresponding to the segment 'your dark s...' (frames 71-147) taken from the the utterance we analyzed in (2.4.3).

On the figure 26 we have a plot of amplitude waveform of the segment we will consider. On the figure 27 we have a 3D mesh plot containing the LP-derived magnitude spectra of 76 half-overlapping frames of 256 samples, which correspond to our segment. On the figure 28 we have isolated three specific frames, and plotted the LP-derived magnitude spectrum for each. Those frames correspond to voiced sound /u/ in 'your', voiced sound /a/ in 'dark', and unvoiced sound /s/ in 'suit'.

Both on figure 27 and the first two plots in figure 28 we can observe that the bandwidth of the formants typically increases with its central frequency—formants of lower frequencies will have more pointed and narrower peaks, while the formants on higher frequencies typically have flatter and wider peaks.

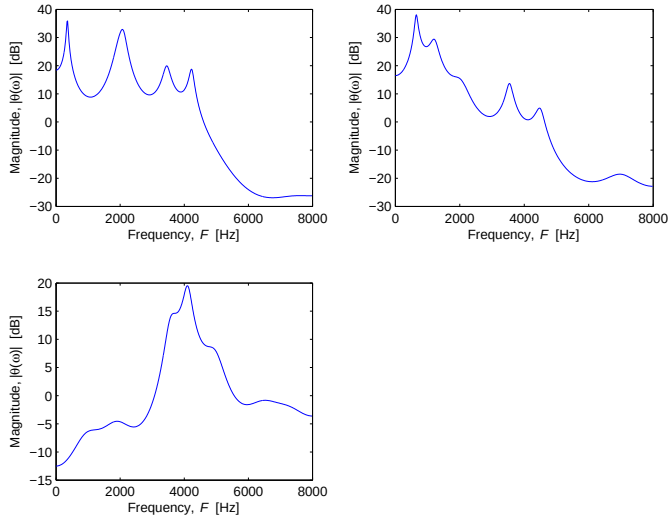


Figure 28: Plots of LP-derived magnitude spectra of three 256 sample frames from the utterance 'your dark s...'. Top left: corresponding to sound /u/ in 'your', top right: corresponding to sound /a/ in 'dark', bottom: corresponding to sound /s/ in 'suit'.

3.6 Pitch and Gain Estimation

The autocorrelation function of the prediction error sequence $\hat{e}(n;m)$ can be used to estimate the pitch period for voiced frames as

$$\max_{\eta} r_{\hat{e}}(\eta, m), \qquad \eta \neq 0$$

If this peak value is below some threshold based on the total residual energy, for example $0.25 r_{\hat{e}}(0;m)$ the frame is unvoiced.

(2.6.1) We used the autocorrelation of the residual sequence $\hat{e}(n;m)$) for the order-14 predictor of the voiced phoneme /i/ in the utterance 'three' to estimate the pitch.

On the figure 29 we plotted the autocorrelation of the residual sequence. Not considering the high peak at $r_{\hat{e}}(0)$, the autocorrelation reaches maximum for the time shift $\eta = 79$. We got exactly the same result by looking at the short-time autocorrelation of the original signal in the exercise (2.1.4), figure 20. We calculated there that the discrete period $\eta = 79$ corresponds to the pitch frequency $F_P = \frac{10 \text{ kHz}}{79} = 126$ Hz. That is a valid value for the pitch frequency, so the above described method of pitch estimation is useful.

However, in this example we have the problem that the peak value is rather small relative to the total residual energy (300 : 1600) so if the threshold value is 0.25 this frame will be labeled unvoiced.

(2.6.2) MATLAB function `lpcpitch.m` implements pitch estimation by finding the maxima of residual sequence. The threshold `th` is used for voiced/unvoiced decision, and the peak search is preformed in the range `minlag` to `maxlag`.

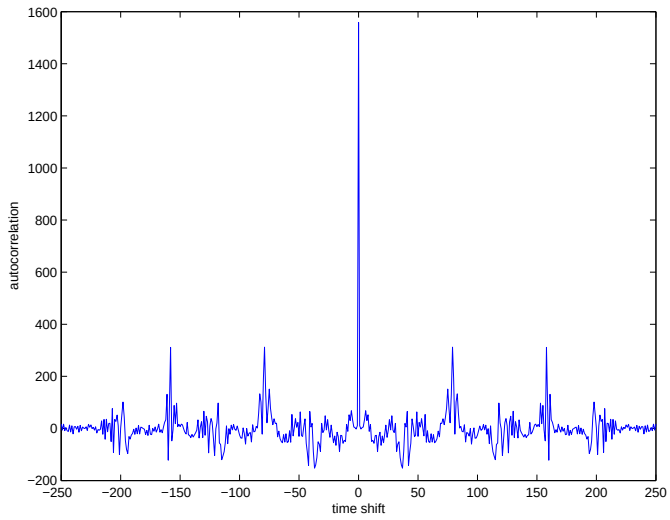


Figure 29: Short-term autocorrelation over the range $[-250, 250]$ of the residual sequence $\hat{e}(n;m)$) for the order-14 predictor of the voiced phoneme /i/.

```
function P = lpcpitch(e,m,N,th,minlag,maxlag)

F = length(m);
P = zeros(F,1);
for (f=1:F)
   n = m(f)-N+1:m(f);
  [re,eta] = xcorr(e(n),maxlag,'biased');
  [remax,idx] = max(re(maxlag+minlag+1:2*maxlag+1));
  if (remax > th*re(maxlag+1))
    P(f) = eta(maxlag+minlag+idx);
  end
end
```

(2.6.3) We applied function `lpcpitch.m` to the residual signal obtained from LP analysis of the speech signal 'She had your dark suit in greasy wash water all year', and we plotted the results on the figure 30.

This pitch estimation method is rather simple to use, comparing to the pitch estimation from the task (1.6.3). Limiting the peak search to the range `minlag` to `maxlag`, limits the search in a certain frequency range, automatically disregarding pitch values that are outside the expected range. But, as we have seen in the task (2.6.2), this method can also give wrong result.

The LP modeling problem focuses on the minimum-phase component in the speech production system. However, the gain $\Theta_0(m)$ accros frames are also important, and can be estimated from the prediction error energy $\xi(m)$ and pitch period N_p as

$$\hat{\Theta}_0(m) \approx \sqrt{\xi(m)}, \qquad \text{unvoiced case}$$

$$\hat{\Theta}_0(m) \approx \sqrt{N_p \xi(m)}, \qquad \text{voiced case}$$

(2.6.4) MATLAB function `lpcgain.m` implements gain estimation. Function takes as inputs the prediction error energy in vector `xi` and the pitch period in vector `P`, and it returns the gain in the vector `G`.

```
function G = lpcgain(xi,P)

F = length(xi);
G = zeros(F,1);
for (f=1:F)
  if (P(f))
    G(f) = sqrt(P(f)*xi(f));
  else
    G(f) = sqrt(xi(f));
  end
end
```

(2.6.5) We applied function `lpcgain` to the signal 'She had your dark suit in greasy wash water all year', and we plotted the results on the figure 31.

(2.6.6) MATLAB function `lpcsyn` performs speech synthesis from LP parameters. Function takes as inputs the LP coefficients in the matrix `A`, pitch period in the

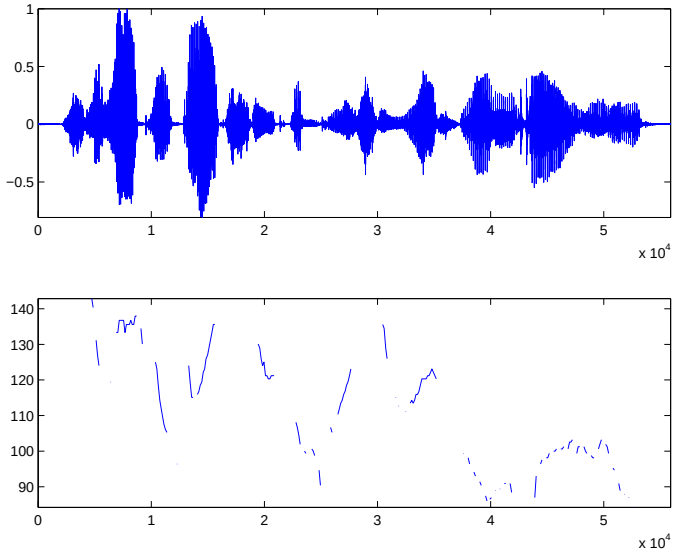


Figure 30: Speech signal ‘She had your dark suit in greasy wash water all year’. Top: amplitude waveform, bottom: result of applying pitch estimation function `lpcpitch.m` to the signal. Threshold value used for voiced/unvoiced segmentation was 0.18, and peak search was done in lag range 100 to 200, which corresponds to frequencies 80 Hz to 160 Hz.

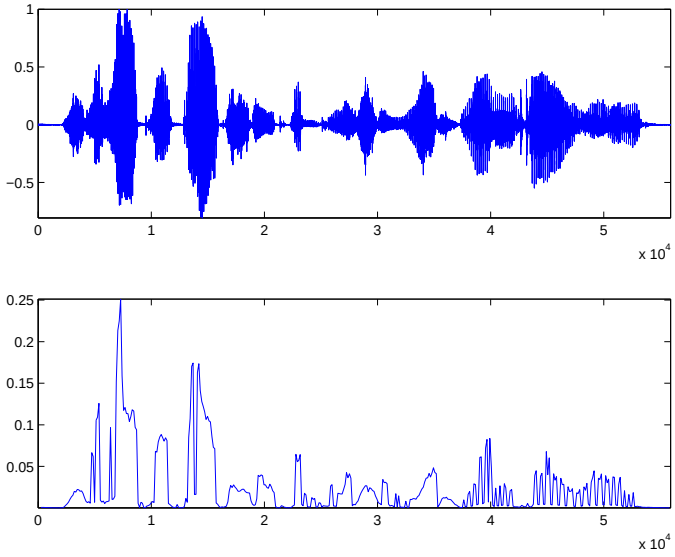


Figure 31: Speech signal ‘She had your dark suit in greasy wash water all year’. Top: amplitude waveform, bottom: result of applying gain estimation function `lpcgain.m` to the signal.

vector P, gain in the vector G, and the vector m with the indices of the last sample in each frame. Function returns synthesized speech signal ehat.

```
function xhat = lpcsyn(A,P,G,m)

F = length(m);
N = m(2) - m(1);
xhat = [];
for (f=1:F)
  if (P(f))
    e = zeros(N,1); e(1:P(f):N) = 1;
  else
    e = randn(N,1);
  end
  xhat = [xhat; filter(G(f),A(:,f),e)];
end
```

(2.6.7) By listening to the synthesized speech we concluded that it is easily understandable, but also very different from the original signal, somehow monotone. Synthesized speech lacks quick variations that make the real speech lively.

3.7 Homework

We consider a toy example where the vocal tract system has order $M = 2$, i.e., the speech signal $s(n)$ is modeled by an AR(2) process with coefficients a_1 and a_2.

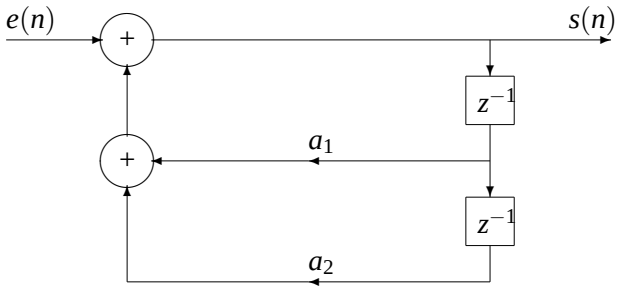
(2.7.1)To sketch the IIR filter for the considered vocal tract model, we first looked at the filter

$$\Theta(z) = \frac{S(z)}{E(z)} = \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

and than we looked at what do we have in the time domain

$$s(n) = a_1 s(n-1) + a_2 s(n-2) + e(n)$$

so the block diagram looks like following illustration



(2.7.2) When the autocorrelation sequence is given by $r_s(0) = 1$, $r_s(1) = 1/2$, and $r_s(2) = 1/8$, the normal equations for this particular order are

$$\begin{pmatrix} r_s(0) & r_s(1) \\ r_s(1) & r_s(0) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} r_s(1) \\ r_s(2) \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 1/2 \\ 1/8 \end{pmatrix}$$

or written out

$$\begin{aligned} a_1 + \frac{1}{2}a_2 &= \frac{1}{2} \\ \frac{1}{2}a_1 + a_2 &= \frac{1}{8} \end{aligned}$$

Solution for the two unknowns is

$$\begin{aligned} a_1 &= \frac{7}{12} \\ a_2 &= -\frac{1}{6} \end{aligned}$$

(2.7.3) In general, as we increase the model order M, the prediction error sequence will be more and more similar to the excitation sequence.

4 Speech Coding and Synthesis

4.1 Perceptual Weighting Filter

Linear prediction analysis estimates the all-pole (vocal-tract) filter for each frame of the signal. The synthesized speech is then generated by exciting vocal tract filter. The difference between the synthesized speech and original speech constitutes an error signal that will be minimized by optimizing the excitation signal. Prior to minimization, error signal is spectrally weighted to emphasize perceptually important frequencies.

Function `lpcana` performs LP analysis on the speech frame using Levison-Durbin recursion, and function `lpcweight` returns the coefficients of the perceptual weighting filter.

(3.2.2) We performed LP analysis on the voiced phoneme /i/ in the utterance 'three' using order-14 prediction, and we found the corresponding perceptual error weighting filter for $c = 0.8$.

```
load digits;
x = digits.three1;
m = 2756;
N = 200;
n = m-N+1:m;
M = 14;
[ar,xi,kappa,ehat] = lpcana(x(n),M);
c = 0.8;
ac = lpcweight(ar,c);

Nfft = 1024;
k = 1:Nfft/2;
Theta = 1./fft(ar,Nfft);
W = fft(ar,Nfft)./fft(ac,Nfft);
plot(k,20*log10(abs([Theta(k) W(k)])))
zplane(ar',ac')
```

On the figure 32 we plotted LP-derived magnitude spectrum of a frame, together with the frequency response of perceptual error weighting filter for the same frame. We can see that the perceptual error weighting filter de-emphasizes formant frequencies.

On the figure 33 we have zero-pole plot of a perceptual error weighting filter. By comparing it to the zero-pole plot of the linear prediction filter on figure 22, we can see how the perceptual weighting filter is obtained by placing zeros at the positions where LP filter has poles, and adding corresponding poles at the distance determined by parameter c.

On the figure 34 we have plotted the frequency response of perceptual weighting filter for three different values of parameter c. If the value c is close to 1, frequency response of the filter is rather flat—it is not doing a lot of filtering. When the value of c decreases, filtering becomes more pronounced. The optimal value of c (around 0.8) depends on the human perception.

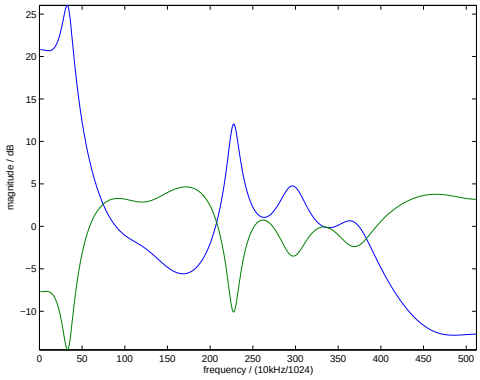


Figure 32: LP-derived magnitude spectrum of a frame corresponding to the voiced phoneme /i/, together with the frequency response of perceptual error weighting filter for the same frame.

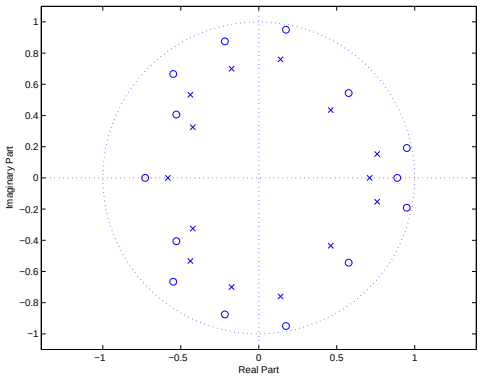


Figure 33: Zero-pole plot of a perceptual error weighting filter from the figure 32. Should be compared to the zero-pole plot on the figure 22.

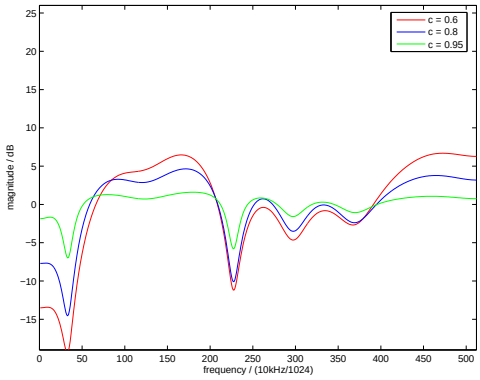


Figure 34: Frequency response of perceptual weighting filter for three different values of parameter *c*.

4.2 Excitation Sequence

The signal used to excite the LP synthesis filter is determined dynamically few times in each frame. An excitation sequence is first selected from the codebook and a long-delay correlation filter is used to generate the pitch periodicity. Pitch and gain are found by performing an exhaustive search on the certain range, and the codebook vector is found by performing an exhaustive search of the codebook to minimize the energy of perceptually weighted error.

(3.3.1) We generated a Gaussian codebook containing 1024 sequences of length 40 by using following commands.

```
randn('state',0);
cb = randn(40,1024);
```

4.3 CELP Synthesizer

CELP coder estimates model parameters from frames of speech, encodes and transmits the parameters to the receiver on a frame-to frame basis. The speech signal is than reconstructed at the receiver. Parameters being transmitted between CELP coder and decoder are: estimated reflection coefficients, gain, index of the codebook vector, scale factor (excitation parameter used for long-delay correlation filter) and pitch.

CELP synthesizer consists of the cascade of the pitch synthesis filter and the LP synthesis filter. The excitation signal is taken from the codebook of stored sequences.

MATLAB function `celpsyn.m` implements the CELP synthesizer, while the MATLAB function `celpana` implements the CELP coder.

(3.4.2) We applied the functions `celpana.m` and `celpsyn.m` to the single speech frame, i.e. we first coded and than synthesized the speech frame.

```
load digits;
x = digits.three1;
m = 2756;
N = 200;
n = m-N+1:m;
L = 40;
M = 12;
c = 0.8;
Pidx = [16 160];
bbuf = 0;
ebuf = zeros(N,1);
Zf = [];
Zw =  [];

[kappa,k,theta0,P,b,ebuf,Zf,Zw] = ...
    celpana(x(n),L,M,c,cb,Pidx,bbuf,ebuf,Zf,Zw);
[xhat,ebuf,Zi] = celpsyn(cb,kappa,k,theta0,P,b,ebuf,[]);
plot([xhat x(n)])
```

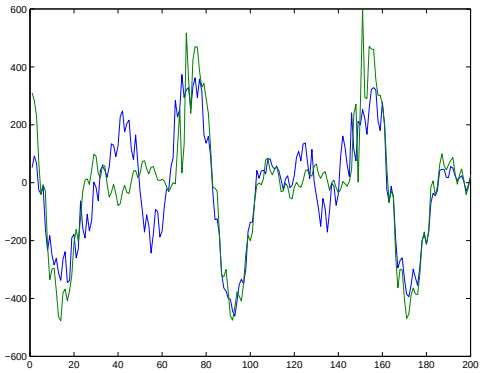


Figure 35: A 256 samples frame corresponding to the voiced phoneme /i/ in the utterance ‘three’ (blue) together with the frame obtained by CELP coding/decoding the signal (green).

On the figure 35 we plotted the original speech frame, and the synthesized frame. The similarities between the synthesized and original speech frame are obvious, but we can also notice that the synthesized signal has a stronger ‘attack’ than the original signal.

(3.4.4) MATLAB function `celp.m` preforms frame base CELP coding and synthesis on speech signals. We applied this function to the speech signal ‘The prices have gone up enormously in spite of the technological advances’.

```
randn('state',0);
cb = randn(40,1024);

load ma1_1;
x = ma1_1;
N = 160;
L = 40;
M = 10;
c = 0.8;
Pidx = [16 160];
[xhat,e,k,theta0,P,b] = celp(x,N,L,M,c,cb,Pidx);

plot([x xhat])
```

On the figure 36 we can see the original speech signal and the synthesized speech signal. By listening to the synthesized speech, we concluded that the quality of the synthesized speech is good—it sounds quite natural. We have plotted the excitation signal, and we can see that its shape resembles the shape of the speech signal.

We have also extracted one frame corresponding to the /Y/ sound in ‘prices’ and plotted the original signal together with the synthesized. Now it is easier to notice how good the synthesized signal approximates the original. We can also better see the excitation signal for a voiced segment. We concluded that the excitation signal isn’t just a pulse train—it looks more like as if the part of the speech

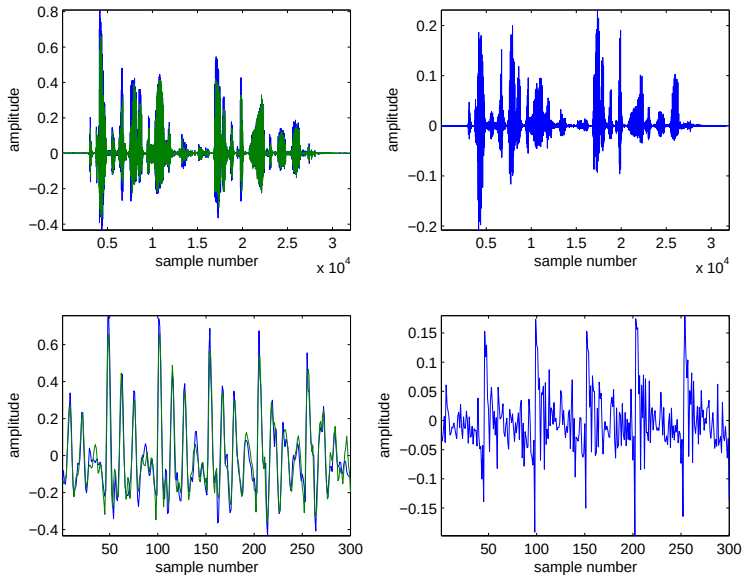


Figure 36: Top left: Speech signal ‘The prices have gone up enormously in spite of the technological advances’, original signal (blue) together with CELP coded/decoded signal (green). Top right: Corresponding excitation signal used in CELP synthesis of the signal. Bottom left: One frame corresponding to the /Y/ sound in ‘prices’ (sample numbers 4161–4460), original signal (blue) together with the CELP coded/decoded signal (green). Bottom right: Corresponding excitation signal used in CELP synthesis.

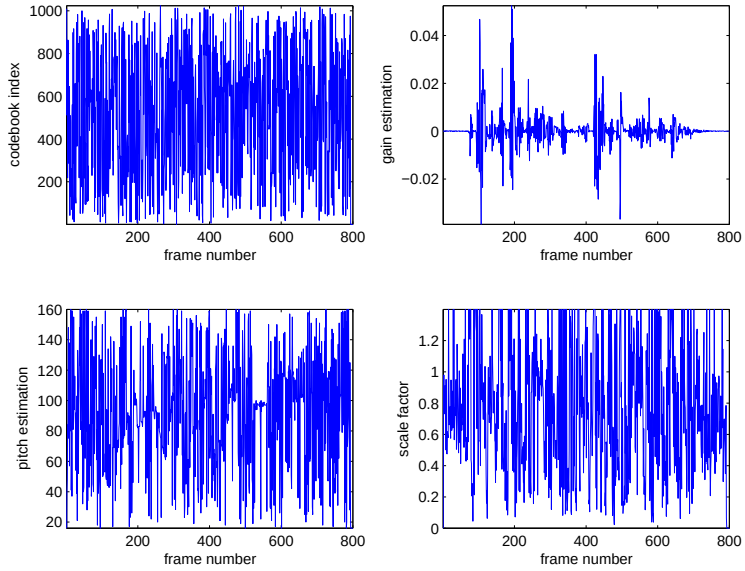


Figure 37: Excitation parameters used in CELP synthesis of the signal from the figure 36. Top left: codebook index. Top right: gain estimation. Bottom left: pitch estimation. Bottom right: scale factor.

signal is present in the excitation signal. We verified this conclusion by listening to excitation signal.

On the figure 37 we have plotted excitation parameters: codebook index, gain estimation, pitch estimation and scale factor that controls amounts of the contribution from the codebook sequence and from an interval of past excitation. As expected, codebook indexes are chosen random-like. Gain reflects the shape of the signal, since gain is connected to the amplitude/energy. Pitch estimation is not resulting in a nice pitch curve, and it seems hard to determine pitch from the sequence of pitch estimations. Scale factor is also not providing any obvious information.

(3.4.5) We repeated the experiment using LPC model order $M = 14$. By listening to this synthesized speech signal, we couldn't hear any improvement. However, listening to the difference of the two signals (i.e. one signal subtracted from the other) verified our expectation that higher model order has to improve signal approximation.

We also repeated experiment using model order $M = 2$, and we were surprised to hear quite muffed, but still understandable speech signal.

(3.4.6) We repeated the experiment modifying the perceptual weighting coefficient c. We used a few different values between 0 and 1, but could not hear a big improvement/degradation of the synthesized speech. Even using $c = 1$ (no perceptual weighting at all) didn't decrease the quality of the signal significantly, or at least not so much that we could hear it. On the other hand, using the same method as in the previous task, and listening to differences between signals, confirms that the difference exists.

(3.4.7) We also tried modifying codebook size and the length of the random signals in the codebook. We noticed that the size of the codebook greatly influences the speed of the computation. For example, reducing the number of vectors from 1024 to 256 reduced the computational time from 13 to 6 seconds. This is easy to understand—initial excitation sequence is found by preforming exhaustive search of the codebook, so it must take longer time to search longer codebook. The length of the vectors is not influencing computational time so strongly—having shorter frames means having more frames.

As for the quality, again we didn't hear any significant improvement/degradation of the speech signal when modifying the number or the length of the vectors.

4.4 Quantization

Because of limited bandwidth between coder and decoder, the parameters obtained by CELP coder are first quantized according to the bit allocation table and than transmitted.

(3.5.2) Function `celp16k.m` implements 16000 bps CELP coder and we applied it to the speech signal 'The prices have gone up enormously in spite of the technological advances'. On the figure 38 we plotted the original signal together with coded/encoded signal. By listening to synthesized speech signal, we could yet again not hear any significant degradation.

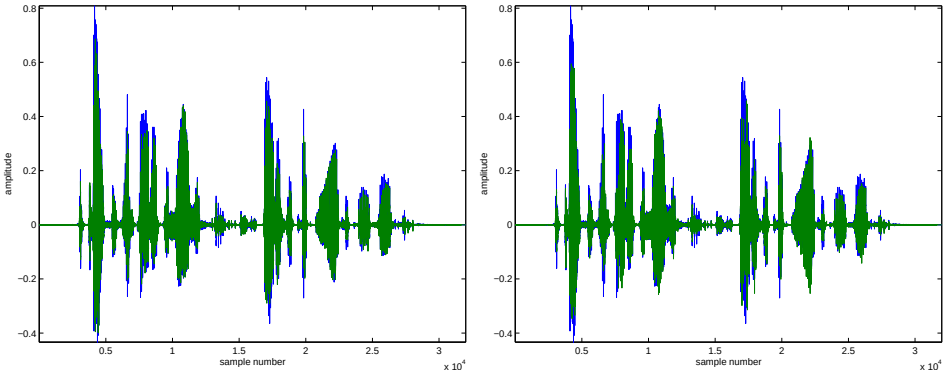


Figure 38: Speech signal ‘The prices have gone up enormously in spite of the technological advances’, original signal (blue) together with the synthesized signal (green). Left: using 16000 bps CELP coder/decoder. Right: using 9600 bps CELP coder/decoder.

(3.5.4) Function `celp9600k.m` implements 9600 bps CELP coder. We repeated the experiment by applying function `celp9600k.m` to the speech signal. We plotted the original and coded/decoded signal on the figure 38. By listening to the synthesized signal we did observe degradation—signal was muffed, but still very understandable.

4.5 Homework

(3.6) We should discuss the improvements of the LPC based vodocoders that can be obtained by using linear interpolation of adjacent frame parameters.

Using linear interpolation of adjacent frame parameters is surely not influencing the transmission bandwidth of the LP parameters, since interpolation is done only on the decoding side. So there is a good chance that interpolation could be efficient (and bandwidth ‘cheap’) way to improve the quality of the synthesized signal. The question is still whether using interpolation actually improves the synthesized signal quality.

We know that LP coefficients in adjacent frames can change rapidly. Linear interpolation of LP coefficients would result in smoother variation of LP coefficients. It is possible that the rapid change of LP coefficients introduces some unwanted effects and interpolation could help in that case. On the other hand, it is also possible that smoothing of LP parameters would result with unwanted flatness. Experiment is a way to solve this dilemma.

5 Speech Recognition

5.1 Feature Extraction

Feature vector sequence for each frame are usually obtained from the LP parameters. In our exercise, the feature vector is composed of 12 cepstral coefficients and 12 difference cepstral coefficients.

Function `hmmfeatures.m` performs feature extraction procedure on a speech signal.

(4.1.2) We verified function `hmmfeatures.m` on the speech signal ‘one’ using the following commands.

```
load digits;
s = digits.one1;
N = 320;
deltaN = 80;
M = 12;
Q = 12;
y = hmmfeatures(s,N,deltaN,M,Q);
plot(y)
```

On the figure 39 we can see the obtained feature vectors. Each feature vector is composed of 12 cepstral coefficients and 12 differenced cepstral coefficients, and we can see that cepstral coefficients are rather dependant on each other—a lot of vectors has the same value for the first 3 features. Differenced cepstral coefficients are not so dependant on each other.

We assumed that cepstral coefficients change slowly from frame to frame, and to verify that on the figure 40 we plotted the change of feature values from frame

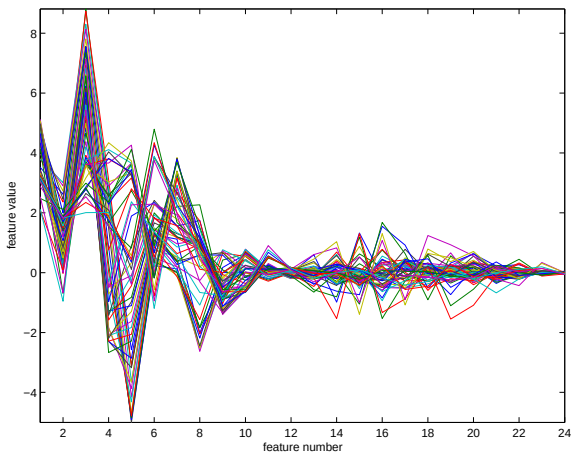


Figure 39: Feature vectors extracted form speech signal ‘one’. There are in total 54 feature vectors corresponding to 54 overlapping frames. Feature numbers 1–12: cepstrum, feature numbers 13–24 differenced cepstrum.

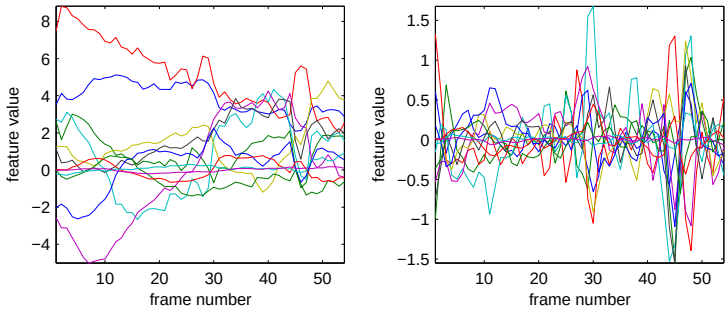


Figure 40: Change of feature values from frame to frame. Left: 12 cepstral coefficients, right: 12 differenced cepstral coefficients.

to frame. We can see that our assumption was true, and we can also see that the differenced cepstral coefficients change rapidly in time.

5.2 Vector Quantization

We want to use HMM with a discrete observation symbol instead of continuous vectors shown on the figure 39. Therefor we use function `kmeans.m` that implements vector quantization algorithm. For each word, the feature extraction constitutes an observation sequence of 24 dimensional vectors, which are than quantized into one of the permissible set resulting in the scalar, discrete observation sequence.

Function `kmeans` implements k-means vector quantization algorithm.

(4.2.2) We verified the `kmeans` function on 2 dimensional random vectors because it is easier to visualize the process of quantization in 2 dimensions.

```
Y = randn(1000,2);
K = 10;
maxiter = 500;
[Yc,c,errlog] = kmeans(Y,K,maxiter);
Yc
figure, hold on,
plot(Y(c==1,1),Y(c==1,2),'.',Y(c==2,1),Y(c==2,2),'.',...
    Y(c==3,1),Y(c==3,2),'.',Y(c==4,1),Y(c==4,2),'.',...
    Y(c==5,1),Y(c==5,2),'.',Y(c==6,1),Y(c==6,2),'.',...
    Y(c==7,1),Y(c==7,2),'.',Y(c==8,1),Y(c==8,2),'.',...
    Y(c==9,1),Y(c==9,2),'.',Y(c==10,1),Y(c==10,2),'.','MarkerSize',10)
for i=1:10, text(Yc(i,1),Yc(i,2),num2str(i),'FontWeight','bold'), end,
axis([-2.5,2.5,-2.5,2.5])
```

We plotted the result of quantization on the figure 41.

Function `hmmcodebook.m` makes a codebook `cb` containing feature vector prototypes based on training sequences defined by the cell-array containing words that will be used for training. For each word, a frame based analysis is performed using `hmmfeatures.m` to give observation vectors. The feature vectors for all words are

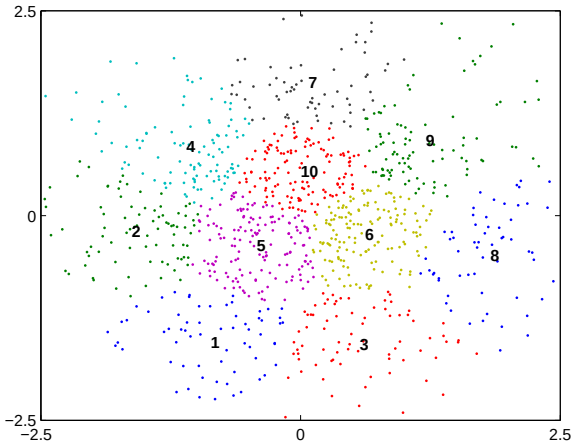


Figure 41: Result of performing vector quantization of 1000 random points. Centroids are marked with the cluster number, and vector points assigned to a certain cluster are represented with dots of the same color.

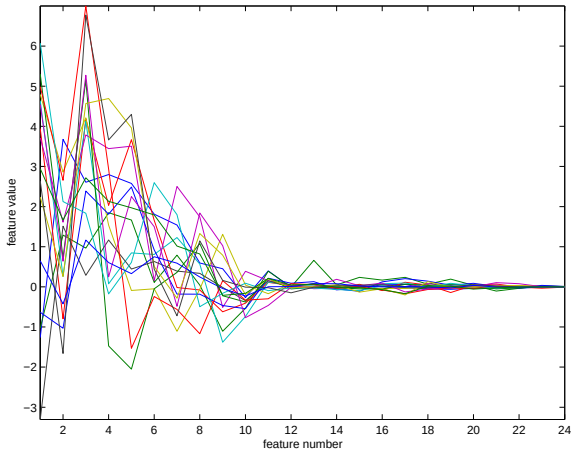


Figure 42: Feature vector prototypes (codebook vectors) obtained by vector quantization of the feature vectors extracted from the training set consisting of 15 occurrences of each spoken word ‘zero’ to ‘nine’. Feature vectors were quantized into 16 clusters. Feature numbers 1–12: cepstrum, feature numbers 13–24 differenced cepstrum.

than concatenated and vector quantized into `K` feature vector prototypes, where `K < Kmax`.

(4.2.5) We generated a codebook from a training set of 15 occurrences of each of spoken words ‘zero’ to ‘nine’ using following commands.

```
load ti46;
data = ti46.case(27:36);
N = 320;
deltaN = 80;
M = 12;
Q = 12;
Kmax = 16;
[cb,K,T,dist] = hmmcodebook(data,N,deltaN,M,Q,Kmax);
plot(cb)
```

On the figure 42 we can see the resulting 16 vector prototypes. We can see that the vector prototypes are not as dependent as the feature vectors from a single word. To cover the whole training set, the feature vectors need to show more variation and need to be on a certain distance one from another.

(4.2.6) One measure of the quality of a codebook is the average distance of an observation vector in the training data from its corresponding symbol. We checked how does the distortion depend on the codebook size `K` using following commands.

```
load ti46;
data = ti46.case(27:36);
N = 320;
deltaN = 80;
M = 12;
Q = 12;
Kmax = [2 4 8 16 32];
dist = zeros(1,length(Kmax));
for i=1:length(Kmax)
    [cb,K,T,dist(i)] = hmmcodebook(data,N,deltaN,M,Q,Kmax(i));
end
plot(Kmax, dist)
```

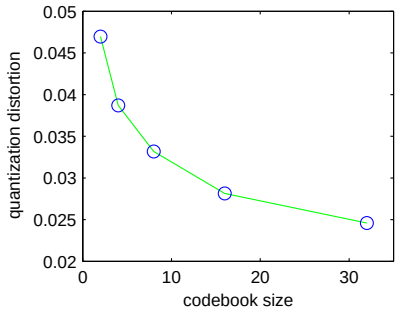


Figure 43: Quantization distortion versus codebook size for the feature vectors extracted from the training set consisting of 15 occurrences of each spoken word ‘zero’ to ‘nine’.

On the figure 43 we can see that, as expected, distortion decreases with increasing codebook size. Since a larger codebook implies more computation (increased size of $K \times S$ observation probability matrix B), there is an incentive to keep the codebook as small as possible without jeopardizing recognition ability. Based on these two conditions, the reasonable codebook size is guided by the experiment evidence. When implementing the quantization, it can be decided that the desired codebook size is reached when the curve of distortion versus codebook size is flat enough.

5.3 Training the HMM

Training the HMM for its dedicated word means using the training words to estimate state transition matrix A, observation probability matrix B and state probability vector $\pi(1)$. The aim is to maximize probability of a HMM to produce it's dedicated word. This is done iteratively using the forward-backward reestimation algorithm until the desired tolerance is reached. We also need to train for the same word multiple times.

Function `hmmfb.m` implements F-B algorithm and function `hmmtrain.m` implements iterative training of multiple HMM's based on the feature extraction, vector quantization, and F-B algorithm.

(4.3.5) We generated a codebook from a training set of 10 occurrences of each of the spoken words 'zero' to 'nine'. The codebook was then used to train a set of ten HMM's each having 5 hidden states.

```
load ti46;
data = ti46.case(27:36);
for(i=1:10)
    train{i} = data{i}(1:10);
end
N = 320;
deltaN = 80;
M = 12;
Q = 12;
Kmax = 16;
[cb,K,T,dist] = hmmcodebook(train,N,deltaN,M,Q,Kmax);
S = 5;
maxiter = 5000;
tol = 1e-3;
[A_m,B_m,pi_m,loglike_m] = hmmtrain(train,N,deltaN,M,Q,cb,S,maxiter,tol);
plot(loglike_m{1})
plot(loglike_m{6})
```

We can observe the log-likelihood as function of the iteration number in the F-B reestimation algorithm on the figures 44 where we plotted log-likelihoods computed when building HMM's for recognizing words 'one' and 'six'. We can see that it took 16 iterations to build satisfactory model for the word 'one' and 17 iterations to build the model for word 'six'. We can conclude that the typical training

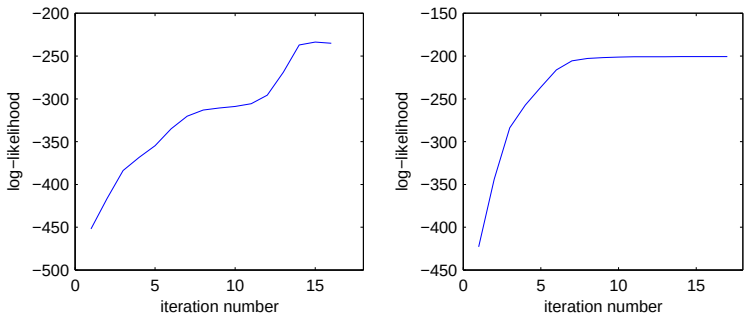


Figure 44: Log-likelihood versus iteration number computed when building HMM's for recognizing the word 'one' (left), and the word 'six' (right).

length is around 20 iterations. Log-likelihoods reached the final, maximal value of approximately -250, which corresponds to probability of 10^{-250}.

We can also observe the structure of the matrices `A` and `B`.

```
mesh(A_m{1})
mesh(B_m{1})
```

We can see the mesh plot of matrices `A` and `B` on the figures 45 and 46. Matrix `A` has largest probabilities on the main diagonal. This means that if the process is in a certain state at the time t, the probability is highest that it will stay in that state in the time $t+1$. This implies that the states don't change rapidly in time.

Looking at the visualization of matrix `B` on figure 46 we can see that for each state, there is just a couple of observations that have a high probability. Usually it is just one high probability per state. For example, for state 4 there is very high probability of observing symbol 9, and for state 2 there a high probabilities of observing symbols 6 and 15. We can conclude that the hidden Markov model is not totaly hidden.

The structure of matrices `A` and `B` is in accordance with our conclusion for the task (4.1.2)—if the chance is big to stay in the present state, and if the chance of observing a single symbol for each state is also big, then feature vectors will change slowly in time.

5.4 Recognition using the HMM

For a given (but unknown) observation sequence and a given HMM trained on certain word, we can calculate the log-likelihood that HMM produced a sequence. To recognize a given signal, we first have to utilize `hmmfeatures.m` to extract the feature vectors. Than we use function`hmmlogp.m` to calculate the log-likelihood for a given sequence and a given HMM. Finally, we have to find the HMM that gives the highest probability of producing the sequence.

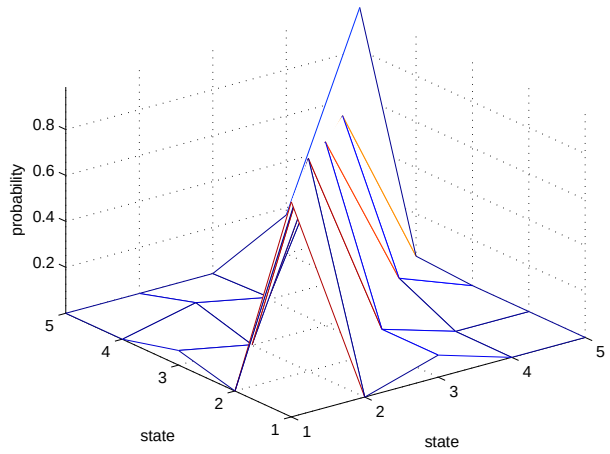


Figure 45: Visualization of the 5×5 state transition matrix `A` of a HMM trained on 10 sequences of the word ‘one’.

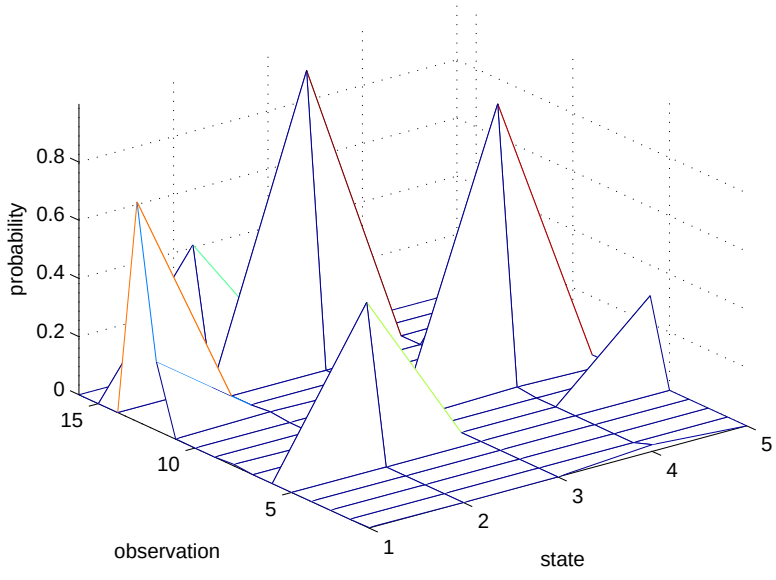


Figure 46: Visualization of the 16×5 observation probability matrix `B` trained on 10 sequences of the word ‘one’.

Function `hmmrecog.m` implements HMM based recognition, and returns the classification of the HMM most probably producing the sequence.

(4.4.3) We try to recognize the word 'one' not taken from the training set by the following command:

```
[logp,guess]=hmmrecog(data{1}(12),A_m,B_m,pi_m,cb,N,deltaN,M,Q)
```

Function `hmmrecog.m` correctly returned '1' as a guess. The returned log-likelihoods are (from up to down, for HMM's dedicated to 'one', 'two',...):

```
logp =

  1.0e+003 *

   -0.0231
   -0.5045
   -1.0025
   -0.7222
   -1.0036
   -1.0167
   -1.0058
   -1.0025
   -1.0030
   -0.5042
```

That corresponds to probabilities (multiplied by 10^{1000}):

```
Ps =

    0.9482
    0.3130
    0.0994
    0.1896
    0.0992
    0.0962
    0.0987
    0.0994
    0.0993
    0.3132
```

So, the calculated probabilities are very small, which is the reason behind using log-likelihoods. However, it is evident that the HHM dedicated to word 'one' had much larger probability of producing the sequence, than any other HMM.

(4.4.4) We should try to repeat the experiment using different words and we decided to test the recognizer on all the words that are not in the training set.

```
for i=1:10
    for j=1:5
        [logp,guess] = hmmrecog(data{i}(10+j),A_m,B_m,pi_m,cb,N,deltaN,M,Q);
        guesses(i,j) = guess;
    end
end
guesses(find(guesses==10)) = 0;
```

The returned matrix of guesses is

one	two	three	four	five	six	seven	eight	nine	zero
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0

So we can see that the recognizer had success rate of 100% on this set.

(4.4.5) We repeated the experiments using various block size and spacing—original size was `N = 320` and `deltaN = 80`. We expected more wrong guesses as `N` and `deltaN` increase, but the recognizer proved to be rather robust, and we needed to increase `N` up to 2000 and `deltaN` up to 500 to get a few wrong results. We include results for `N = 3200` and `deltaN = 800` where we can see the deterioration in the quality of recognizing—word ‘nine’ was wrongly recognized as ‘one’ in 4 out of 5 cases.

one	two	three	four	five	six	seven	eight	nine	zero
1	2	3	4	5	6	7	8	1	0
1	2	3	4	5	6	7	8	1	0
1	2	3	4	5	6	7	8	9	2
1	2	8	4	5	6	7	8	1	0
9	2	3	4	5	6	7	8	1	0

We assume that the success rate will depend more on the block size and spacing for recognizers that have to capture subtle difference between words, for example for medium size vocabularies recognizers. Longer frames do increaseimprove the computation speed, so again, one has to evaluate the computation speed and correctness by experiment to solve this dilemma.

(4.4.6) We repeated the experiments using LPC/cepstrum coding of smaller and larger order—originally was `M = 12` and `Q = 12`. Success rate was generally smaller for smaller order and the recognition was becoming more unstable, changing the output from experiment to experiment. Still, we found the recognizer rather robust to the changes of LPC/cepstrum order. We include the results for `M = 4` and `Q = 4`.

one	two	three	four	five	six	seven	eight	nine	zero
1	5	5	4	5	6	7	6	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	5	5	4	5	6	7	6	9	0

(4.4.7) We repeated the experiments again, this time using codebooks of different sizes—originally was `K = 16`. Quantizing the feature vectors into smaller number of clusters worsened the success rate. We include the results for `K = 6`.

one	two	three	four	five	six	seven	eight	nine	zero
1	2	8	2	5	6	0	8	9	0
1	2	3	4	5	6	0	8	1	0
1	2	3	4	5	6	7	8	9	0
1	2	8	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0

In conclusion, the choice of parameters has to be determined by experiment, considering the requirements for the specific speech recognizer.

5.5 Homework

(4.5) We should discuss improvements on the word classifier, based on what we have learned during the course.

When changing the parameters of the word classifier we noticed that classifier often makes two wrong guesses for a single word while guessing everything else correctly. We assumed that the model for the word in question wasn't optimized correctly during training.

As we learned in the lesson, reestimation algorithm finds the model that gives the local maximum of the probability, and finding global maximum is not ensured. So, the model found by reestimation algorithm depends on initial (random) point of search.

The speech recognizer we implemented in the exercise is not taking this problem into consideration. To improve the chances of finding the optimal model, reestimation algorithm could be performed a number of times from different starting points used by F-B algorithm, and the best of the found models could be chosen.

References

[1] Sadaoki Furui. Speech recognition technology in the ubiquitous/wearable computing environment. *Proceedings of the 2000 IEEE International Conference on Acousitcs, Speech, and Signal Processing*.

[2] John H. L. Hansen John R. Deller and John G. Proakis. *Discrete-Time Processing of Speech Signals*. IEEE Press, 2000.

[3] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing*. Prentice-Hall, third edition, 1996.

[4] Thomas F. Quatieri. *Discrete-Time Speech Signal Processing*. Prentice-Hall, third edition, 1996.

[5] Andreas Spanias Ted Painter. Perceptual coding of digital audio. *Proceedings of IEEE*, 88(4), April 2000.