Lab 9b - Speech Processing (part 2)*

Charles A. Bouman

Questions or comments concerning this laboratory should be directed to Prof. Charles A. Bouman, School of Electrical and Computer Engineering, Purdue University, West Lafayette IN 47907; (765) 494-0340; bouman@ecn.purdue.edu

1 Introduction

This is the second part of a two week experiment. During the first week we discussed basic properties of speech signals, and performed some simple analyses in the time and frequency domain.

This week, we will introduce a system model for speech production. We will cover some background on **linear predictive coding**, and the final exercise will bring all the prior material together in a speech coding exercise.

1.1 A Speech Model

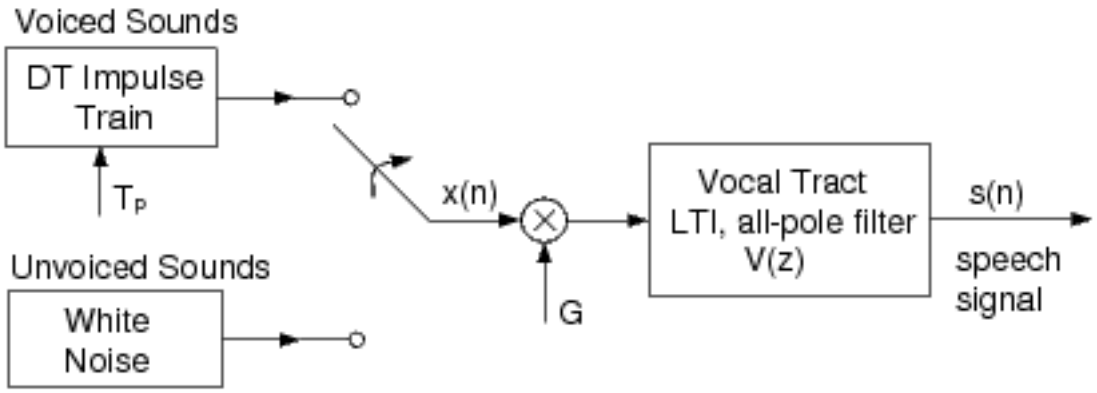

Figure 1: Discrete-Time Speech Production Model

*Version 1.3: Sep 17, 2009 2:49 pm -0500
†http://creativecommons.org/licenses/by/2.0/

From a signal processing standpoint, it is very useful to think of speech production in terms of a model, as in Figure 1. The model shown is the simplest of its kind, but it includes all the principal components. The excitations for voiced and unvoiced speech are represented by an impulse train and white noise generator, respectively. The pitch of voiced speech is controlled by the spacing between impulses, T_p, and the amplitude (volume) of the excitation is controlled by the gain factor G.

As the acoustical excitation travels from its source (vocal cords, or a constriction), the shape of the vocal tract alters the spectral content of the signal. The most prominent effect is the formation of resonances, which intensifies the signal energy at certain frequencies (called **formants**). As we learned in the Digital Filter Design lab, the amplification of certain frequencies may be achieved with a linear filter by an appropriate placement of poles in the transfer function. This is why the filter in our speech model utilizes an all-pole LTI filter. A more accurate model might include a few zeros in the transfer function, but if the order of the filter is chosen appropriately, the all-pole model is sufficient. The primary reason for using the all-pole model is the distinct computational advantage in calculating the filter coefficients, as will be discussed shortly.

Recall that the transfer function of an all-pole filter has the form

$$V(z) = \frac{1}{1 - \sum_{k=1}^{P} a_k z^{-k}} \tag{1}$$

where P is the order of the filter. This is an IIR filter that may be implemented with a recursive difference equation. With the input $G \cdot x(n)$, the speech signal $s(n)$ may be written as

$$s(n) = \sum_{k=1}^{P} a_k s(n-k) + G \cdot x(n) \tag{2}$$

Keep in mind that the filter coefficients will change continuously as the shape of the vocal tract changes, but speech segments of an appropriately small length may be approximated by a time-invariant model.

This speech model is used in a variety of speech processing applications, including methods of speech recognition, speech coding for transmission, and speech synthesis. Each of these applications of the model involves dividing the speech signal into short segments, over which the filter coefficients are almost constant. For example, in speech transmission the bit rate can be significantly reduced by dividing the signal up into segments, computing and sending the model parameters for each segment (filter coefficients, gain, etc.), and re-synthesizing the signal at the receiving end, using a model similar to Figure 1. Most telephone systems use some form of this approach. Another example is speech recognition. Most recognition methods involve comparisons between short segments of the speech signals, and the filter coefficients of this model are often used in computing the "difference" between segments.

1.2 Synthesis of Voiced Speech

Download the file coeff.mat[1] for the following section.

Download the file coeff.mat[2] and load it into the Matlab workspace using the `load` command. This will load three sets of filter coefficients: $A1$, $A2$, and $A3$ for the vocal tract model in (1) and (2). Each vector contains coefficients $\{a_1, a_2, \cdots, a_{15}\}$ for an all-pole filter of order 15.

We will now synthesize voiced speech segments for each of these sets of coefficients. First write a Matlab function `x=exciteV(N,Np)` which creates a length N excitation for voiced speech, with a pitch period of **Np** samples. The output vector x should contain a discrete-time impulse train with period **Np** (e.g. [1 0 0 $\cdots$ 0 1 0 0 $\cdots$]).

Assuming a sampling frequency of 8 kHz (0.125 ms/sample), create a 40 millisecond-long excitation with a pitch period of 8 ms, and filter it using (2) for each set of coefficients. For this, you may use the command

```
s = filter(1,[1 -A],x)
```

[1] See the file at <http://cnx.org/content/m18087/latest/coeff.mat>

[2] See the file at <http://cnx.org/content/m18087/latest/coeff.mat>

where A is the row vector of filter coefficients (see Matlab's help on **filter** for details). Plot each of the three filtered signals. Use `subplot()` and `orient tall` to place them in the same figure.

We will now compute the frequency response of each of these filters. The frequency response may be obtained by evaluating (1) at points along $z = e^{j\omega}$. Matlab will compute this with the command `[H,W]=freqz(1,[1 -A],512)` , where A is the vector of coefficients. Plot the magnitude of each response versus frequency in Hertz. Use `subplot()` and `orient tall` to plot them in the same figure.

The location of the peaks in the spectrum correspond to the formant frequencies. For each vowel signal, estimate the first three formants (in Hz) and list them in the figure.

Now generate the three signals again, but use an excitation which is 1-2 seconds long. Listen to the filtered signals using `soundsc`. Can you hear qualitative differences in the signals? Can you identify the vowel sounds?

INLAB REPORT

Hand in the following:

- A figure containing the three time-domain plots of the voiced signals.
- Plots of the frequency responses for the three filters. Make sure to label the frequency axis in units of Hertz.
- For each of the three filters, list the approximate center frequency of the first three formant peaks.
- Comment on the audio quality of the synthesized signals.

2 Linear Predictive Coding

The filter coefficients which were provided in the previous section were determined using a technique called **linear predictive coding** (LPC). LPC is a fundamental component of many speech processing applications, including compression, recognition, and synthesis.

In the following discussion of LPC, we will view the speech signal as a discrete-time random process.

2.1 Forward Linear Prediction

Suppose we have a discrete-time random process $\{..., S_{-1}, S_0, S_1, S_2, ...\}$ whose elements have some degree of correlation. The goal of **forward linear prediction** is to predict the sample S_n using a linear combination of the previous P samples.

$$\hat{S_n} = \sum_{k=1}^{P} a_k S_{n-k} \tag{3}$$

P is called the **order** of the predictor. We may represent the error of predicting S_n by a random sequence e_n.

$$\begin{aligned} e_n &= S_n - \hat{S_n} \\ e_n &= S_n - \textstyle\sum_{k=1}^{P} a_k S_{n-k} \end{aligned} \tag{4}$$

An optimal set of prediction coefficients a_k for (4) may be determined by minimizing the mean-square error $E\left[e_n^2\right]$. Note that since the error is generally a function of n, the prediction coefficients will also be functions of n. To simplify notation, let us first define the following column vectors.

$$\mathbf{a} = \left[a_1 \ a_2 \cdots a_P\right]^T \tag{5}$$

$$\mathbf{S}_{n,P} = \left[S_{n-1} \ S_{n-2} \cdots S_{n-P}\right]^T \tag{6}$$

Then,

$$\begin{aligned} E\left[e_n^2\right] &= E\left[\left(S_n - \sum_{k=1}^{P} a_k S_{n-k}\right)^2\right] \\ &= E\left[\left(S_n - \mathbf{a}^T \mathbf{S}_{n,P}\right)^2\right] \\ &= E\left[S_n^2 - 2S_n \mathbf{a}^T \mathbf{S}_{n,P} + \mathbf{a}^T \mathbf{S}_{n,P}\ \mathbf{a}^T \mathbf{S}_{n,P}\right] \\ &= E\left[S_n^2\right] - 2\mathbf{a}^T E\left[S_n \mathbf{S}_{n,P}\right] + \mathbf{a}^T E\left[\mathbf{S}_{n,P}\mathbf{S}_{n,P}^T\right]\mathbf{a} \end{aligned} \tag{7}$$

The second and third terms of (7) may be written in terms of the autocorrelation sequence $r_{SS}(k,l)$.

$$E\left[S_n \mathbf{S}_{n,P}\right] = \begin{bmatrix} E\left[S_n S_{n-1}\right] \\ E\left[S_n S_{n-2}\right] \\ \vdots \\ E\left[S_n S_{n-P}\right] \end{bmatrix} = \begin{bmatrix} r_{SS}(n, n-1) \\ r_{SS}(n, n-2) \\ \vdots \\ r_{SS}(n, n-P) \end{bmatrix} \equiv \mathbf{r}_S \tag{8}$$

$$\begin{aligned} E\left[\mathbf{S}_{n,P}\mathbf{S}_{n,P}^T\right] &= E\begin{bmatrix} S_{n-1}S_{n-1} & S_{n-1}S_{n-2} & \cdots & S_{n-1}S_{n-P} \\ S_{n-2}S_{n-1} & S_{n-2}S_{n-2} & \cdots & S_{n-2}S_{n-P} \\ \vdots & \vdots & \ddots & \vdots \\ S_{n-P}S_{n-1} & S_{n-P}S_{n-2} & \cdots & S_{n-P}S_{n-P} \end{bmatrix} \\ &= \begin{bmatrix} r_{SS}(n-1, n-1) & r_{SS}(n-1, n-2) & \cdots & r_{SS}(n-1, n-P) \\ r_{SS}(n-2, n-1) & r_{SS}(n-2, n-2) & \cdots & r_{SS}(n-2, n-P) \\ \vdots & \vdots & \ddots & \vdots \\ r_{SS}(n-P, n-1) & r_{SS}(n-P, n-2) & \cdots & r_{SS}(n-P, n-P) \end{bmatrix} \equiv \mathbf{R}_S \end{aligned} \tag{9}$$

Substituting into (7), the mean-square error may be written as

$$E\left[e_n^2\right] = E\left[S_n^2\right] - 2\mathbf{a}^T \mathbf{r}_S + \mathbf{a}^T \mathbf{R}_S \mathbf{a} \tag{10}$$

Note that while $\mathbf{a}$ and $\mathbf{r}_S$ are vectors, and $\mathbf{R}_S$ is a matrix, the expression in (10) is still a scalar quantity.

To find the optimal a_k coefficients, which we will call $\hat{\mathbf{a}}$, we differentiate (10) with respect to the vector $\mathbf{a}$ (compute the gradient), and set it equal to the zero vector.

$$\nabla_{\mathbf{a}} E\left[e_n^2\right] = -2\mathbf{r}_S + 2\mathbf{R}_S \hat{\mathbf{a}} \equiv \mathbf{0} \tag{11}$$

Solving,

$$\mathbf{R}_S \hat{\mathbf{a}} = \mathbf{r}_S \tag{12}$$

The vector equation in (12) is a system of P scalar linear equations, which may be solved by inverting the matrix $\mathbf{R}_S$.

Note from (8) and (9) that $\mathbf{r}_S$ and $\mathbf{R}_S$ are generally functions of n. However, if S_n is wide-sense stationary, the autocorrelation function is only dependent on the difference between the two indices, $r_{SS}(k,l) =$

$r_{SS}(|k-l|)$. Then $\mathbf{R}_S$ and $\mathbf{r}_S$ are no longer dependent on n, and may be written as follows.

$$\mathbf{r}_S = \begin{bmatrix} r_{SS}(1) \\ r_{SS}(2) \\ \vdots \\ r_{SS}(P) \end{bmatrix} \tag{13}$$

$$\mathbf{R}_S = \begin{bmatrix} r_{SS}(0) & r_{SS}(1) & \cdots & r_{SS}(P-1) \\ r_{SS}(1) & r_{SS}(0) & \cdots & r_{SS}(P-2) \\ r_{SS}(2) & r_{SS}(1) & \cdots & r_{SS}(P-3) \\ \vdots & \vdots & \ddots & \vdots \\ r_{SS}(P-1) & r_{SS}(P-2) & \cdots & r_{SS}(0) \end{bmatrix} \tag{14}$$

Therefore, if S_n is wide-sense stationary, the optimal a_k coefficients do not depend on n. In this case, it is also important to note that $\mathbf{R}_S$ is a Toeplitz (constant along diagonals) and symmetric matrix, which allows (12) to be solved efficiently using the Levinson-Durbin algorithm (see [2]). This property is essential for many real-time applications of linear prediction.

2.2 Linear Predictive Coding of Speech

An important question has yet to be addressed. The solution in (12) to the linear prediction problem depends entirely on the autocorrelation sequence. How do we estimate the autocorrelation of a speech signal? Recall that the applications to which we are applying LPC involve dividing the speech signal up into short segments and computing the filter coefficients for each segment. Therefore we need to consider the problem of estimating the autocorrelation for a short segment of the signal. In LPC, the following "biased" autocorrelation estimate is often used.

$$\hat{r}_{SS}(m) = \frac{1}{N} \sum_{n=0}^{N-m-1} s(n)\, s(n+m), \quad 0 \le m \le P \tag{15}$$

Here we are assuming we have a length N segment which starts at $n = 0$. Note that this is the single-parameter form of the autocorrelation sequence, so that the forms in (13) and (14) may be used for $\mathbf{r}_S$ and $\mathbf{R}_S$.

2.3 LPC Exercise

Download the file test.mat[3] for this exercise.

Write a function `coef=mylpc(x,P)` which will compute the order-P LPC coefficients for the column vector x, using the autocorrelation method ("lpc" is a built-in Matlab function, so use the name **mylpc**). Consider the input vector x as a speech segment, in other words do not divide it up into pieces. The output vector **coef** should be a column vector containing the P coefficients $\{\hat{a}_1, \hat{a}_2, \cdots, \hat{a}_P\}$. In your function you should do the following:

1. Compute the biased autocorrelation estimate of (15) for the lag values $0 \le m \le P$. You may use the `xcorr` function for this.
2. Form the $\mathbf{r}_S$ and $\mathbf{R}_S$ vectors as in (13) and (14). Hint: Use the `toeplitz` function to form $\mathbf{R}_S$.
3. Solve the matrix equation in (12) for $\hat{\mathbf{a}}$.

[3] See the file at <http://cnx.org/content/m18087/latest/test.mat>

To test your function, download the file test.mat[4], and **load** it into Matlab. This file contains two vectors: a signal x and its order-15 LPC coefficients a. Use your function to compute the order-15 LPC coefficients of x, and compare the result to the vector a.

INLAB REPORT: Hand in your `mylpc` function.

3 Speech Coding and Synthesis

Download the file phrase.au[5] for the following section.

One very effective application of LPC is the compression of speech signals. For example, an **LPC vocoder** (voice-coder) is a system used in many telephone systems to reduce the bit rate for the transmission of speech. This system has two overall components: an analysis section which computes signal parameters (gain, filter coefficients, etc.), and a synthesis section which reconstructs the speech signal after transmission.

Since we have introduced the speech model in "A Speech Model" (Section 1.1: A Speech Model), and the estimation of LPC coefficients in "Linear Predictive Coding" (Section 2: Linear Predictive Coding), we now have all the tools necessary to implement a simple vocoder. First, in the analysis section, the original speech signal will be split into short time frames. For each frame, we will compute the signal energy, the LPC coefficients, and determine whether the segment is voiced or unvoiced.

Download the file phrase.au[6]. This speech signal is sampled at a rate of 8000 Hz.

1. Divide the original speech signal into 30ms non-overlapping frames. Place the frames into L consecutive columns of a matrix S (use **reshape**). If the samples at the tail end of the signal do not fill an entire column, you may disregard these samples.
2. Compute the energy of each frame of the original word, and place these values in a length L vector called **energy**.
3. Determine whether each frame is voiced or unvoiced. Use your **zero_cross** function from the first week to compute the number of zero-crossings in each frame. For length N segments with less than $\frac{N}{2}$ zero-crossings, classify the segment as voiced, otherwise unvoiced. Save the results in a vector **VU** which takes the value of "1" for voiced and "0" for unvoiced.
4. Use your `mylpc` function to compute order-15 LPC coefficients for each frame. Place each set of coefficients into a column of a $15 \times L$ matrix A.

To see the reduction in data, add up the total number of bytes Matlab uses to store the encoded speech in the arrays **A**, **VU**, and **energy**. (use the `whos` function). Compute the **compression ratio** by dividing this by the number of bytes Matlab uses to store the original speech signal. Note that the compression ratio can be further improved by using a technique called **vector quantization** on the LPC coefficients, and also by using fewer bits to represent the gain and voiced/unvoiced indicator.

Now the computed parameters will be used to re-synthesize the phrase using the model in Figure 1. Similar to your `exciteV` function from "Synthesis of Voiced Speech" (Section 1.2: Synthesis of Voiced Speech), create a function `x=exciteUV(N)` which returns a length N excitation for unvoiced speech (generate a Normal(0,1) sequence). Then for each encoded frame do the following:

1. Check if current frame is voiced or unvoiced.
2. Generate the frame of speech by using the appropriate excitation into the filter specified by the LPC coefficients (you did this in "Synthesis of Voiced Speech" (Section 1.2: Synthesis of Voiced Speech)). For voiced speech, use a pitch period of 7.5 ms. Make sure your synthesized segment is the same length as the original frame.
3. Scale the amplitude of the segment so that the synthesized segment has the same energy as the original.

[4]See the file at <http://cnx.org/content/m18087/latest/test.mat>
[5]See the file at <http://cnx.org/content/m18087/latest/phrase.au>
[6]See the file at <http://cnx.org/content/m18087/latest/phrase.au>

4. Append the frame to the end of the output vector.

Listen to the original and synthesized phrase. Can you recognize the synthesized version as coming from the same speaker? What are some possible ways to improve the quality of the synthesized speech? `Subplot` the two speech signals in the same figure.

INLAB REPORT

Hand in the following:

- Your analysis and synthesis code.
- The compression ratio.
- Plots of the original and synthesized words.
- Comment on the quality of your synthesized signal. How might the quality be improved?

References

[1] J. G. Proakis J. H. Hansen J. R. Deller, Jr. *Discrete-Time Processing of Speech Signals*. Macmillan, New York, 1993.

[2] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, New Jersey, 3rd edition, 1996.